一：基础集群环境搭建：

k8s基础集群环境主要是运行kubernetes管理端服务以及node节点上的服务部署及使用。

Kubernetes设计架构：

```
https://www.kubernetes.org.cn/kubernetes%E8%AE%BE%E8%AE%A1%E6%9E%B6%E6%9E%84
```

CNCF 云原生容器生态系统概要：

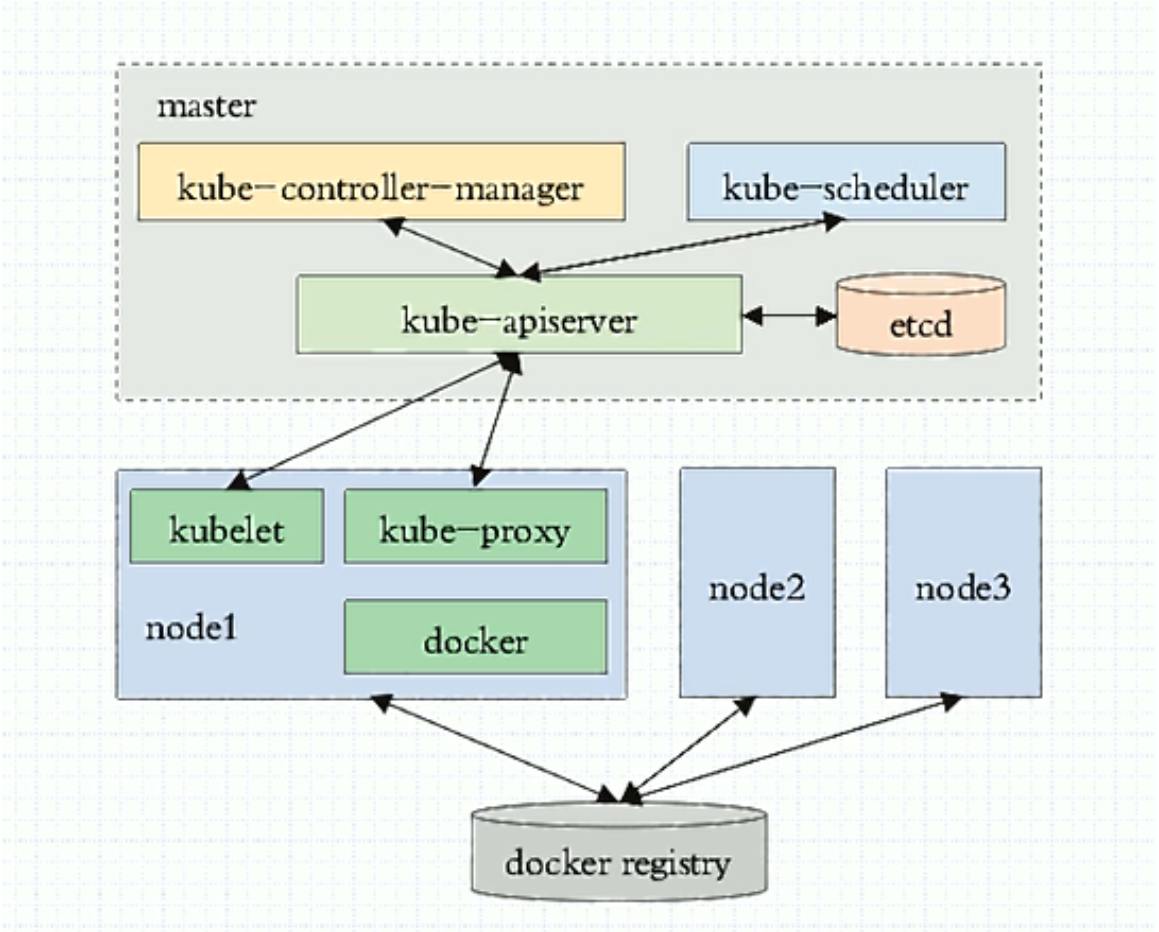
```
http://dockone.io/article/3006
```

1.1：k8s高可用集群环境规划信息：

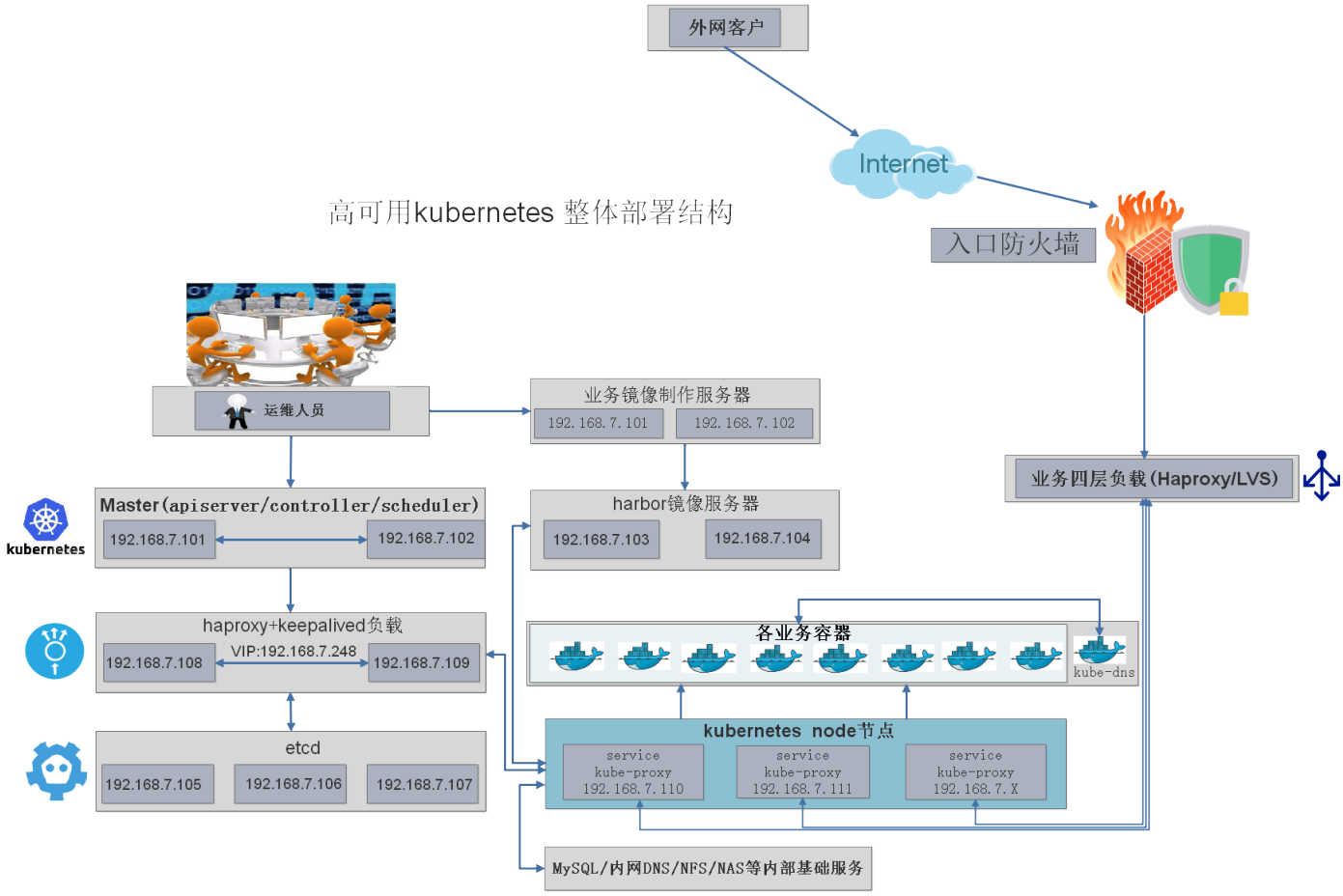
安装实际需求，进行规划与部署相应的单master或者多master的高可用k8s运行环境。

1.1.1：单master：

见 **kubeadm 安装k8s**



1.1.2：多master：



1.1.3：服务器统计：

类型	服务器IP地址	备注
Ansible(2台)	192.168.7.101/102	K8S集群部署服务器，可以和在一起
K8S Master(2台)	192.168.7.101/102	K8s控制端，通过一个VIP做主备高可用
Harbor(2台)	192.168.7.103/104	高可用镜像服务器
Etcd(最少3台)	192.168.7.105/106/107	保存k8s集群数据的服务器
Hproxy(2台)	192.168.7.108/109	高可用etcd代理服务器
Node节点(2-N台)	192.168.7.111/112/xxx	真正运行容器的服务器，高可用环境至少两台

1.2：主机名设置：

类型	服务器IP	主机名	VIP
K8S Master1	192.168.7.101	k8s-master1.magedu.net	192.168.7.248
K8S Master2	192.168.7.102	k8s-master2.magedu.net	192.168.7.248
Harbor1	192.168.7.103	k8s-harbor1.magedu.net	
Harbor2	192.168.7.104	k8s-harbor2.magedu.net	
etcd节点1	192.168.7.105	k8s-etcd1.magedu.net	
etcd节点2	192.168.7.106	k8s-etcd2.magedu.net	
etcd节点3	192.168.7.107	k8s-etcd3.magedu.net	
Haproxy1	192.168.7.108	k8s-ha1.magedu.net	
Haproxy2	192.168.7.109	k8s-ha2.magedu.net	
Node节点1	192.168.7.110	k8s-node1.magedu.net	
Node节点2	192.168.7.111	k8s-node2.magedu.net	

1.3：软件清单：

见当前目录下 **kubernetes软件清单**

API端口：

```
端口：192.168.7.248：6443  #需要配置在负载均衡上实现反向代理，dashboard的端口为8443
操作系统：ubuntu server 1804
k8s版本：  1.13.5
calico：3.4.4
```

1.4：基础环境准备：

http://releases.ubuntu.com/

系统主机名配置、IP配置、系统参数优化，以及依赖的负载均衡和Harbor部署

1.4.1：系统配置：

主机名等系统配置略

1.4.2：高可用负载均衡：

k8s高可用反向代理

参见博客http://blogs.studylinux.net/?p=4579

1.4.2.1：keepalived：

```
root@k8s-ha1:~# cat /etc/keepalived/keepalived.conf
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 1
    priority 100
    advert_int 3
    unicast_src_ip 192.168.7.108
    unicast_peer {
        192.168.7.109
    }

    authentication {
        auth_type PASS
        auth_pass 123abc
    }
    virtual_ipaddress {
        192.168.7.248 dev eth0 label eth0:1
    }
}
```

1.4.2.2：haproxy：

```
listen k8s_api_nodes_6443
    bind 192.168.7.248:6443
    mode tcp
    #balance leastconn
    server 192.168.7.101  192.168.7.101:6443  check inter 2000 fall 3 rise 5
    #server 192.168.100.202  192.168.100.202:6443  check inter 2000 fall 3 rise 5
```

1.4.3：Harbor之https：

内部镜像将统一保存在内部Harbor服务器，不再通过互联网在线下载。

1.4.3.1：安装harbor：

```
root@k8s-harbor1:/apps# pwd
/apps

#解压
root@k8s-harbor1:/apps# tar xvf harbor-offline-installer-v2.3.2.tgz
root@k8s-harbor1:/apps/harbor# mkdir  certs
root@k8s-harbor1:/apps/harbor# cd certs/
```

```
#生成私有key
root@k8s-harbor1:/apps/harbor/certs# openssl genrsa -out harbor-ca.key
Generating RSA private key, 2048 bit long modulus (2 primes)
......+++++
.............................................................+++++
e is 65537 (0x010001)

#签发证书
root@k8s-harbor1:/apps/harbor/certs# openssl req -x509 -new -nodes -key harbor-ca.key
-subj "/CN=harbor.magedu.net" -days 7120 -out harbor-ca.crt

#修改harbor配置文件
root@k8s-harbor1:/apps/harbor/certs# cd ..
root@k8s-harbor1:/apps/harbor# vim harbor.yml
root@k8s-harbor1:/apps/harbor# grep -v "#" harbor.yml | grep -v "^$"
hostname: harbor.magedu.net
http:
  port: 80
https:
  port: 443
  certificate: /apps/harbor/certs/harbor-ca.crt
  private_key: /apps/harbor/certs/harbor-ca.key
harbor_admin_password: 123456
database:
  password: root123
  max_idle_conns: 100
  max_open_conns: 900
data_volume: /data

root@k8s-harbor1:/apps/harbor# ./install.sh  --with-trivy
```

1.4.3.2：客户端 同步在crt证书：

```
#同步证书
root@k8s-harbor2:~# mkdir   /etc/docker/certs.d/harbor.magedu.net -p
root@k8s-harbor1:/usr/local/src# scp /apps/harbor/certs/harbor-ca.crt
172.31.7.105:/etc/docker/certs.d/harbor.magedu.net

#添加host文件解析
root@k8s-harbor2:~# vim /etc/hosts
172.31.7.104 harbor.magedu.net

#重启docker
root@k8s-harbor2:~# systemctl  restart docker
```

1.4.3.3：测试登录harbor：

```
root@k8s-harbor2:~# docker login harbor.magedu.net
Username: admin
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

1.4.3.4：测试push镜像到harbor：

```
root@k8s-harbor2:~# docker pull alpine

root@k8s-harbor2:~# docker tag alpine harbor.magedu.net/library/alpine
root@k8s-harbor2:~# docker push harbor.magedu.net/library/alpine
The push refers to repository [harbor.magedu.net/library/alpine]
e2eb06d8af82: Pushed
latest: digest: sha256:69704ef328d05a9f806b6b8502915e6a0a4faa4d72018dc42343f511490daf8a
size: 528
```

1.5：手动二进制部署：

见 k8s 1.11 ubuntu1804部署文档

1.6：ansible部署：

1.6.1：基础环境准备：

```
# apt-get install python2.7
# ln -s /usr/bin/python2.7 /usr/bin/python
# apt-get install git ansible -y
# ssh-keygen  #生成密钥对
# apt-get install sshpass #ssh同步公钥到各k8s服务器

#分发公钥脚本：
root@k8s-master1:~# cat scp.sh
#!/bin/bash
#目标主机列表
IP="
192.168.7.101
```

```
192.168.7.102
192.168.7.103
192.168.7.104
192.168.7.105
192.168.7.106
192.168.7.107
192.168.7.108
192.168.7.109
192.168.7.110
192.168.7.111
"
for node in ${IP};do
  sshpass -p 123456 ssh-copy-id  ${node}  -o StrictHostKeyChecking=no
  if [ $? -eq 0 ];then
    echo "${node} 秘钥copy完成"
  else
    echo "${node} 秘钥copy失败"
  fi
done


#同步docker证书脚本：
#!/bin/bash
#目标主机列表
IP="
192.168.7.101
192.168.7.102
192.168.7.103
192.168.7.104
192.168.7.105
192.168.7.106
192.168.7.107
192.168.7.108
192.168.7.109
192.168.7.110
192.168.7.111
"
for node in ${IP};do
  sshpass -p 123456 ssh-copy-id  ${node}  -o StrictHostKeyChecking=no
  if [ $? -eq 0 ];then
    echo "${node} 秘钥copy完成"
    echo "${node} 秘钥copy完成,准备环境初始化....."
      ssh  ${node}  "mkdir /etc/docker/certs.d/harbor.magedu.net -p"
      echo "Harbor 证书目录创建成功!"
      scp /etc/docker/certs.d/harbor.magedu.net/harbor-ca.crt
${node}:/etc/docker/certs.d/harbor.magedu.net/harbor-ca.crt
      echo "Harbor 证书拷贝成功!"
      scp   /etc/hosts ${node}:/etc/hosts
      echo "host 文件拷贝完成"
```

```
      scp -r  /root/.docker  ${node}:/root/
      echo "Harbor 认证文件拷贝完成!"
      scp -r  /etc/resolv.conf  ${node}:/etc/
  else
    echo "${node} 秘钥copy失败"
  fi
done


#执行脚本同步：
k8s-master1:~# bash scp.sh

root@s2:~# vim ~/.vimrc #取消vim 自动缩进功能
set paste
```

1.6.2：clone项目：

```
# git clone  -b  0.6.1 https://github.com/easzlab/kubeasz.git
root@k8s-master1:~# mv /etc/ansible/* /opt/
root@k8s-master1:~# mv kubeasz/* /etc/ansible/
root@k8s-master1:~# cd  /etc/ansible/
root@k8s-master1:/etc/ansible# cp example/hosts.m-masters.example ./hosts #复制hosts模板
文件
```

1.6.3：准备hosts文件：

```
root@k8s-master1:/etc/ansible# pwd
/etc/ansible
root@k8s-master1:/etc/ansible# cp example/hosts.m-masters.example ./hosts
root@k8s-master1:/etc/ansible# cat hosts
# 集群部署节点：一般为运行ansible 脚本的节点
# 变量 NTP_ENABLED (=yes/no) 设置集群是否安装 chrony 时间同步
[deploy]
192.168.7.101 NTP_ENABLED=no

# etcd集群请提供如下NODE_NAME，注意etcd集群必须是1,3,5,7...奇数个节点
[etcd]
192.168.7.105 NODE_NAME=etcd1
192.168.7.106 NODE_NAME=etcd2
192.168.7.107 NODE_NAME=etcd3

[new-etcd] # 预留组，后续添加etcd节点使用
#192.168.7.x NODE_NAME=etcdx

[kube-master]
192.168.7.101

[new-master] # 预留组，后续添加master节点使用
```

```
#192.168.7.5

[kube-node]
192.168.7.110

[new-node] # 预留组，后续添加node节点使用
#192.168.7.xx

# 参数 NEW_INSTALL：yes表示新建，no表示使用已有harbor服务器
# 如果不使用域名，可以设置 HARBOR_DOMAIN=""
[harbor]
#192.168.7.8 HARBOR_DOMAIN="harbor.yourdomain.com" NEW_INSTALL=no

# 负载均衡(目前已支持多于2节点，一般2节点就够了) 安装 haproxy+keepalived
[lb]
192.168.7.1 LB_ROLE=backup
192.168.7.2 LB_ROLE=master

#【可选】外部负载均衡，用于自有环境负载转发 NodePort 暴露的服务等
[ex-lb]
#192.168.7.6 LB_ROLE=backup EX_VIP=192.168.7.250
#192.168.7.7 LB_ROLE=master EX_VIP=192.168.7.250

[all:vars]
# ---------集群主要参数---------------
#集群部署模式：allinone, single-master, multi-master
DEPLOY_MODE=multi-master

#集群主版本号，目前支持：v1.8, v1.9, v1.10，v1.11, v1.12, v1.13
K8S_VER="v1.13"

# 集群 MASTER IP即 LB节点VIP地址，为区别与默认apiserver端口，设置VIP监听的服务端口8443
# 公有云上请使用云负载均衡内网地址和监听端口
MASTER_IP="192.168.7.248"
KUBE_APISERVER="https://{{ MASTER_IP }}:6443"

# 集群网络插件，目前支持calico, flannel, kube-router, cilium
CLUSTER_NETWORK="calico"

# 服务网段 (Service CIDR)，注意不要与内网已有网段冲突
SERVICE_CIDR="10.20.0.0/16"

# POD 网段 (Cluster CIDR)，注意不要与内网已有网段冲突
CLUSTER_CIDR="172.31.0.0/16"

# 服务端口范围 (NodePort Range)
NODE_PORT_RANGE="20000-60000"

# kubernetes 服务 IP (预分配，一般是 SERVICE_CIDR 中第一个IP)
```

```
CLUSTER_KUBERNETES_SVC_IP="10.20.0.1"

# 集群 DNS 服务 IP (从 SERVICE_CIDR 中预分配)
CLUSTER_DNS_SVC_IP="10.20.254.254"

# 集群 DNS 域名
CLUSTER_DNS_DOMAIN="linux36.local."

# 集群basic auth 使用的用户名和密码
BASIC_AUTH_USER="admin"
BASIC_AUTH_PASS="123456"

# ----------附加参数--------------------
#默认二进制文件目录
bin_dir="/usr/bin"

#证书目录
ca_dir="/etc/kubernetes/ssl"

#部署目录，即 ansible 工作目录，建议不要修改
base_dir="/etc/ansible"
```

1.6.4：准备二进制文件：

```
k8s-master1:/etc/ansible/bin# pwd
/etc/ansible/bin
k8s-master1:/etc/ansible/bin# tar xvf k8s.1-13-5.tar.gz
k8s-master1:/etc/ansible/bin# mv bin/* .
```

1.6.4：开始按步骤部署：

通过ansible脚本初始化环境及部署k8s 高可用集群

1.6.4.1：环境初始化

```
root@k8s-master1:/etc/ansible# pwd
/etc/ansible
root@k8s-master1:/etc/ansible# ansible-playbook  01.prepare.yml
```

1.6.4.2：部署etcd集群：

可选更改启动脚本路径

```
root@k8s-master1:/etc/ansible# ansible-playbook  02.etcd.yml
```

各etcd服务器验证etcd服务：

```
root@k8s-etcd1:~# export NODE_IPS="192.168.7.105 192.168.7.106 192.168.7.107"
root@k8s-etcd1:~# for ip in ${NODE_IPS}; do   ETCDCTL_API=3 /usr/bin/etcdctl   --
endpoints=https://${ip}:2379    --cacert=/etc/kubernetes/ssl/ca.pem   --
cert=/etc/etcd/ssl/etcd.pem   --key=/etc/etcd/ssl/etcd-key.pem   endpoint health; done
https://192.168.7.105:2379 is healthy: successfully committed proposal: took =
2.198515ms
https://192.168.7.106:2379 is healthy: successfully committed proposal: took =
2.457971ms
https://192.168.7.107:2379 is healthy: successfully committed proposal: took =
1.859514ms
```

1.6.4.3：部署docker:

可选更改启动脚本路径，但是docker已经提前安装，因此不需要重新执行

```
root@k8s-master1:/etc/ansible# ansible-playbook  03.docker.yml
```

1.6.4.4：部署master：

可选更改启动脚本路径

```
root@k8s-master1:/etc/ansible# ansible-playbook  04.kube-master.yml
```

1.6.4.5：部署node：

node节点必须安装docker

```
root@k8s-master1:/etc/ansible# vim roles/kube-node/defaults/main.yml
# 基础容器镜像
SANDBOX_IMAGE: "harbor.magedu.net/baseimages/pause-amd64:3.1"

root@k8s-master1:/etc/ansible# ansible-playbook  05.kube-node.yml
```

1.6.4.5：部署网络服务calico：

可选更改calico服务启动脚本路径，csr证书信息

```
# docker  load -i calico-cni.tar
# docker tag calico/cni:v3.4.4 harbor.magedu.net/baseimages/cni:v3.4.4
# docker push harbor.magedu.net/baseimages/cni:v3.4.4

# docker load -i calico-node.tar
```

```
# docker tag calico/node:v3.4.4  harbor.magedu.net/baseimages/node:v3.4.4
# docker push harbor.magedu.net/baseimages/node:v3.4.4

# docker load -i calico-kube-controllers.tar
# docker tag calico/kube-controllers:v3.4.4   harbor.magedu.net/baseimages/kube-
controllers:v3.4.4
# docker push harbor.magedu.net/baseimages/kube-controllers:v3.4.4

执行部署网络：
root@k8s-master1:/etc/ansible# ansible-playbook  06.network.yml
```

验证calico:

```
root@k8s-master1:/etc/ansible# calicoctl  node status
Calico process is running.

IPv4 BGP status
+---------------+-------------------+-------+----------+-------------+
| PEER ADDRESS  |     PEER TYPE     | STATE |  SINCE   |    INFO     |
+---------------+-------------------+-------+----------+-------------+
| 192.168.7.110 | node-to-node mesh | up    | 14:22:44 | Established |
+---------------+-------------------+-------+----------+-------------+

IPv6 BGP status
No IPv6 peers found.
```

kubectl run net-test1 --image=alpine --replicas=4 sleep 360000 #创建pod测试夸主机网络通信是否正常

1.6.4.6：添加node节点：

```
[kube-node]
192.168.7.110

[new-node] # 预留组，后续添加node节点使用
192.168.7.111

root@k8s-master1:/etc/ansible# ansible-playbook  20.addnode.yml
```

1.6.4.7：添加master节点：

注释掉lb，否则无法下一步

```
[kube-master]
192.168.7.101

[new-master] # 预留组，后续添加master节点使用
192.168.7.102

root@k8s-master1:/etc/ansible# ansible-playbook  21.addmaster.yml
```

1.6.4.8：验证当前状态：

```
root@k8s-master1:/etc/ansible# calicoctl  node status
Calico process is running.

IPv4 BGP status
+---------------+-------------------+-------+----------+-------------+
| PEER ADDRESS  |     PEER TYPE     | STATE |  SINCE   |    INFO     |
+---------------+-------------------+-------+----------+-------------+
| 192.168.7.110 | node-to-node mesh | up    | 14:22:45 | Established |
| 192.168.7.111 | node-to-node mesh | up    | 14:33:24 | Established |
| 192.168.7.102 | node-to-node mesh | up    | 14:42:21 | Established |
+---------------+-------------------+-------+----------+-------------+

IPv6 BGP status
No IPv6 peers found.

root@k8s-master1:/etc/ansible# kubectl  get nodes
NAME            STATUS                     ROLES    AGE   VERSION
192.168.7.101   Ready,SchedulingDisabled   master   37m   v1.13.5
192.168.7.102   Ready,SchedulingDisabled   master   41s   v1.13.5
192.168.7.110   Ready                      node     33m   v1.13.5
192.168.7.111   Ready                      node     15m   v1.13.5
```

1.7：k8s应用环境：

1.7.1：dashboard(1.10.1)

部署kubernetes的web管理界面dashboard

1.7.1.1：具体步骤：

```
1.导入dashboard镜像并上传至本地harbor服务器
# tar xvf dashboard-yaml_image-1.10.1.tar.gz
./admin-user-sa-rbac.yaml
./kubernetes-dashboard-amd64-v1.10.1.tar.gz
./kubernetes-dashboard.yaml
./read-user-sa-rbac.yaml
```

```
./ui-admin-rbac.yaml
./ui-read-rbac.yaml
# docker load -i kubernetes-dashboard-amd64-v1.10.1.tar.gz
# docker tag gcr.io/google-containers/kubernetes-dashboard-amd64:v1.10.1
harbor.magedu.net/baseimages/kubernetes-dashboard-amd64:v1.10.1
# docker push harbor.magedu.net/baseimages/kubernetes-dashboard-amd64:v1.10.1

2.修改yaml文件中的dashboard镜像地址为本地harbor地址
image: harbor.magedu.net/baseimages/kubernetes-dashboard-amd64:v1.10.1

3.创建服务
# kubectl  apply -f .
serviceaccount/admin-user created
clusterrolebinding.rbac.authorization.k8s.io/admin-user created
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
serviceaccount/dashboard-read-user created
clusterrolebinding.rbac.authorization.k8s.io/dashboard-read-binding created
clusterrole.rbac.authorization.k8s.io/dashboard-read-clusterrole created
clusterrole.rbac.authorization.k8s.io/ui-admin created
rolebinding.rbac.authorization.k8s.io/ui-admin-binding created
clusterrole.rbac.authorization.k8s.io/ui-read created
rolebinding.rbac.authorization.k8s.io/ui-read-binding created

4.验证dashboard启动完成：
# kubectl  get pods -n  kube-system
NAME                                    READY   STATUS    RESTARTS   AGE
calico-kube-controllers-77d9f69cdd-rbml2   1/1     Running   0          25m
calico-node-rk6jk                       1/1     Running   1          20m
calico-node-vjn65                       1/1     Running   0          25m
calico-node-wmj48                       1/1     Running   0          25m
calico-node-wvsxb                       1/1     Running   0          5m14s
kubernetes-dashboard-cb96d4fd8-th6nw    1/1     Running   0          37s

# kubectl  get  svc  -n  kube-system
NAME                   TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)         AGE
kubernetes-dashboard   NodePort   10.20.32.158   <none>        443:27775/TCP   79s

# kubectl  cluster-info #查看集群信息
Kubernetes master is running at https://192.168.7.248:6443
kubernetes-dashboard is running at https://192.168.7.248:6443/api/v1/namespaces/kube-
system/services/https:kubernetes-dashboard:/proxy
```

1.7.1.2：token登录dashboard：

```
# kubectl -n kube-system get secret | grep admin-user
# kubectl  -n kube-system describe secret admin-user-token-2wm96
```

1.7.1.3：Kubeconfig登录

制作Kubeconfig文件

1.7.1.4：修改iptables为ipvs及调度算法：

https://github.com/kubernetes/kubernetes/tree/master/pkg/proxy/ipvs

IPVS模式在Kubernetes v1.8中引入alpha版本，在v1.9中处于beta版本，在v1.11中处于GA(也就是release版本，国外都是说GA版本)版本，IPTABLES模式已在v1.1中添加，并成为v1.2之后的默认操作模式， IPVS和IPTABLES都基于netfilter， IPVS模式和IPTABLES模式之间的差异如下：

```
IPVS为大型群集提供了更好的可伸缩性和性能。
与IPTABLES相比，IPVS支持更复杂的负载平衡算法（最少连接，轮询，加权轮询等）。
IPVS支持服务器运行状况检查和连接重试等。
```

修改kubernetes使用ipvs：

https://kubernetes.io/zh/blog/2018/07/09/ipvs-based-in-cluster-load-balancing-deep-dive/

root@s6:~# vim /etc/systemd/system/kube-proxy.service

```
  --proxy-mode=ipvs \
  --ipvs-scheduler=sh

算法：
    rr: round-robin
    lc: least connection
    dh: destination hashing
    sh: source hashing
    sed: shortest expected delay
    nq: never queue
```

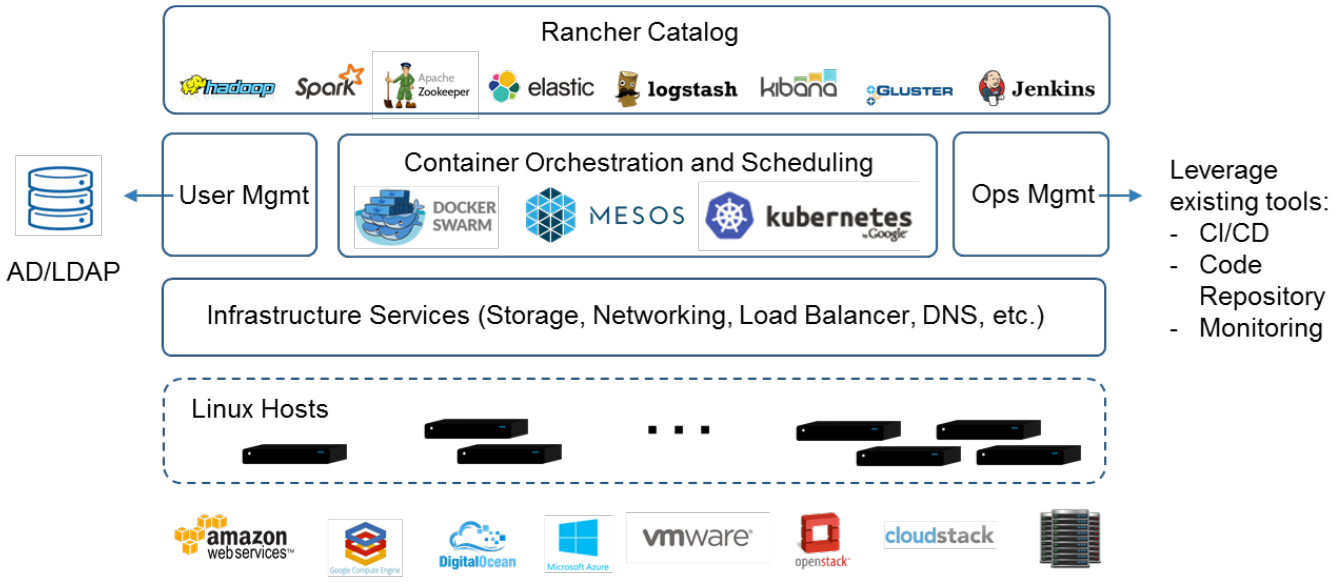
1.7.1.5：设置token登录会话保持时间

```
# vim dashboard/kubernetes-dashboard.yaml
        image: 192.168.200.110/baseimages/kubernetes-dashboard-amd64:v1.10.1
        ports:
        - containerPort: 8443
          protocol: TCP
        args:
          - --auto-generate-certificates
          - --token-ttl=43200
# kubectl  apply  -f .
```

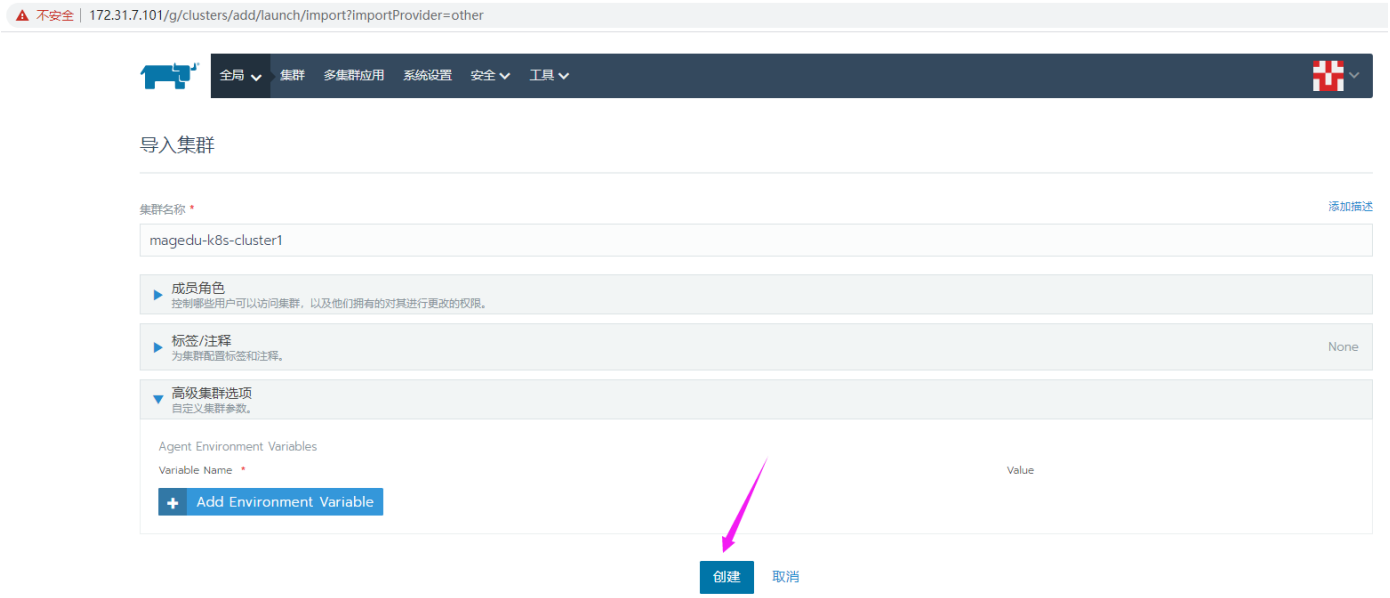
1.7.2：rancher

https://rancher.com/docs/rancher/v1.6/zh/

https://rancher.com/quick-start/

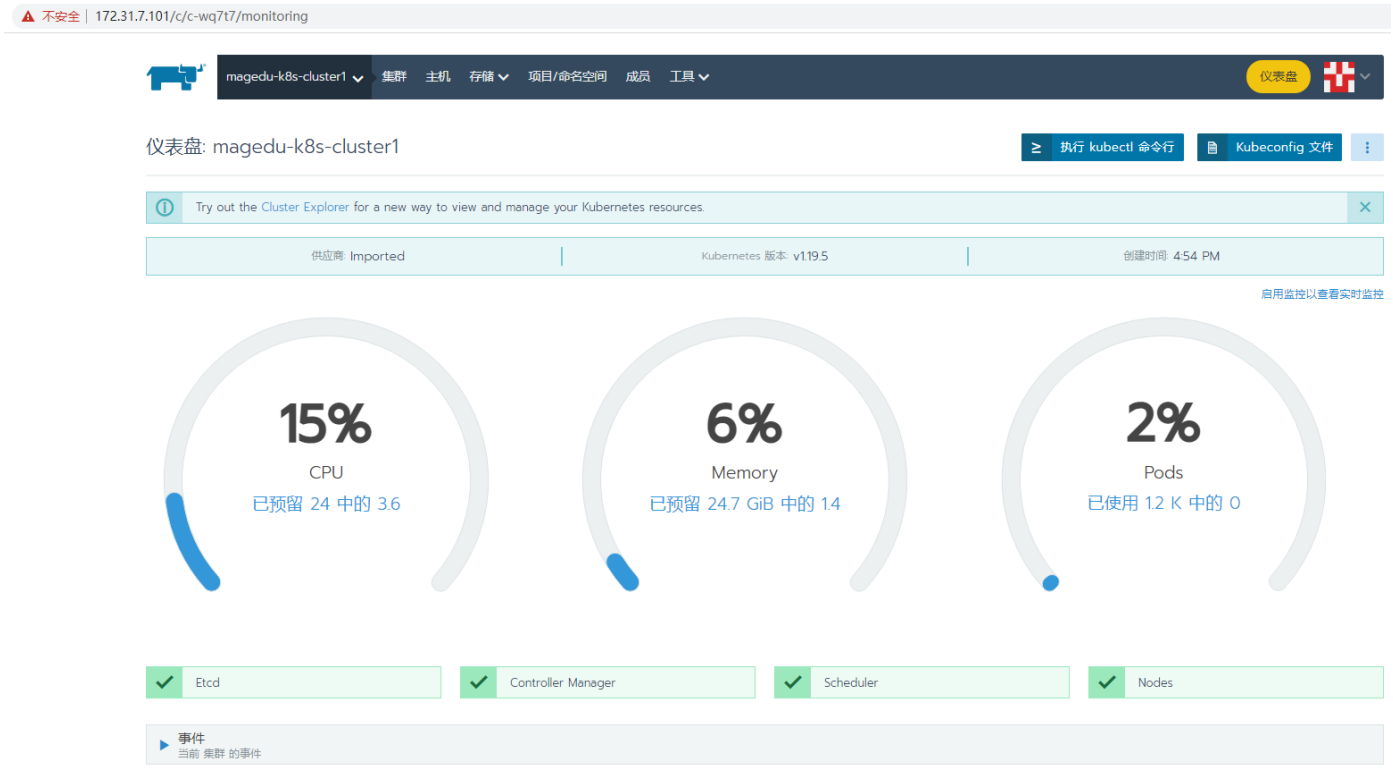


导入集群：

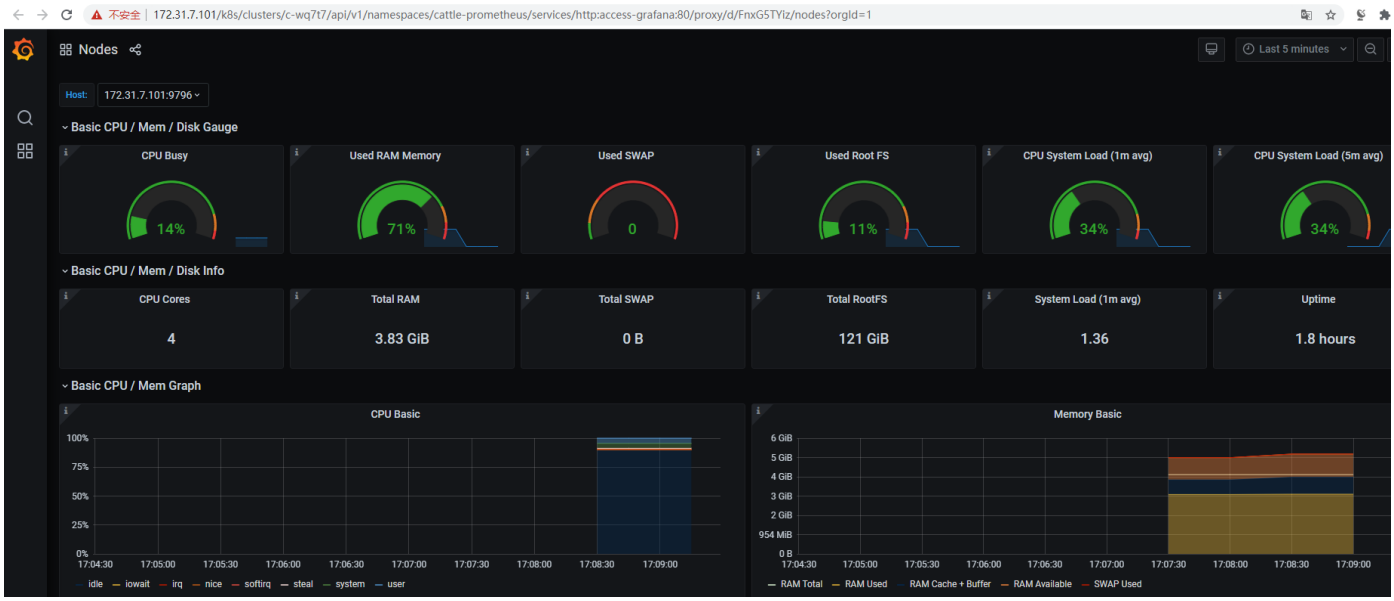


```
# curl --insecure -sfL
https://172.31.7.101/v3/import/c8khcm67zhvxrhbx4ddpmmc88htxmkwb5b4f4fpttn2jlkfnwbscsx_c
-wq7t7.yaml | kubectl apply -f -
```

web页面：



启用监控：

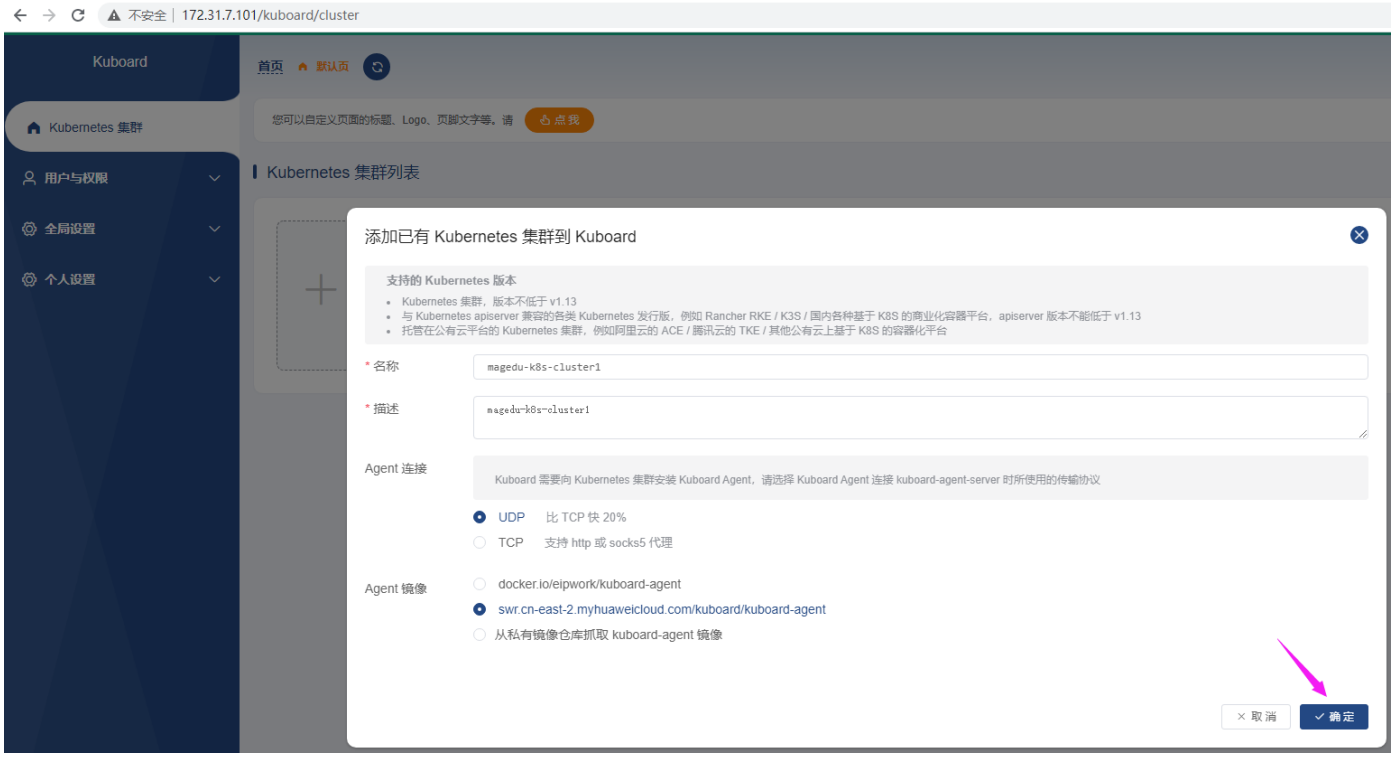


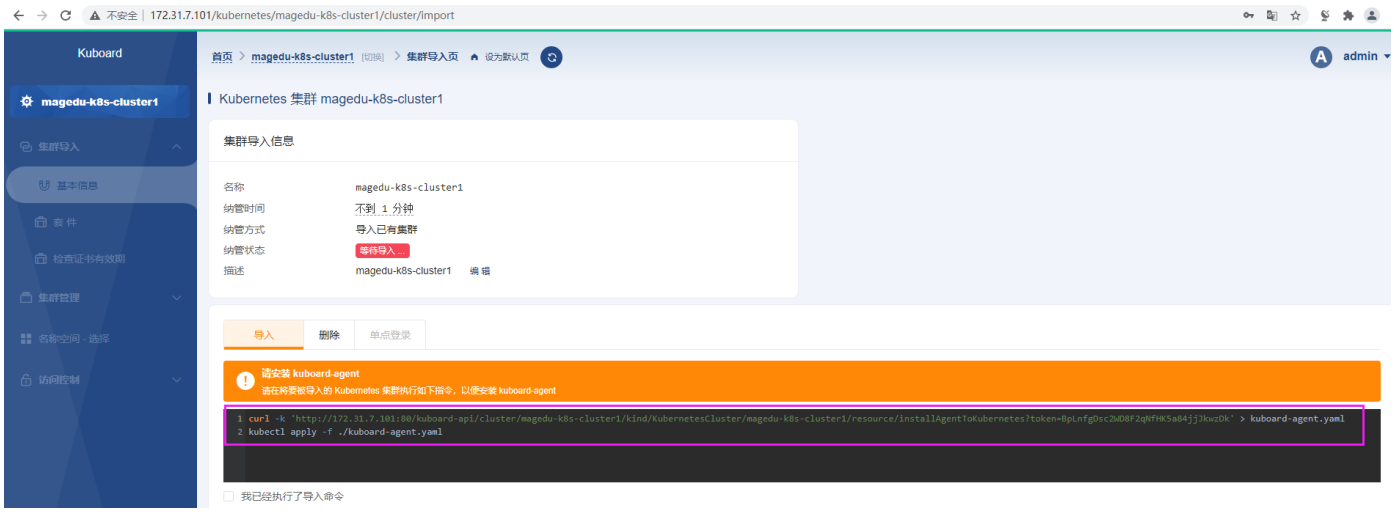
1.7.3：kuboard

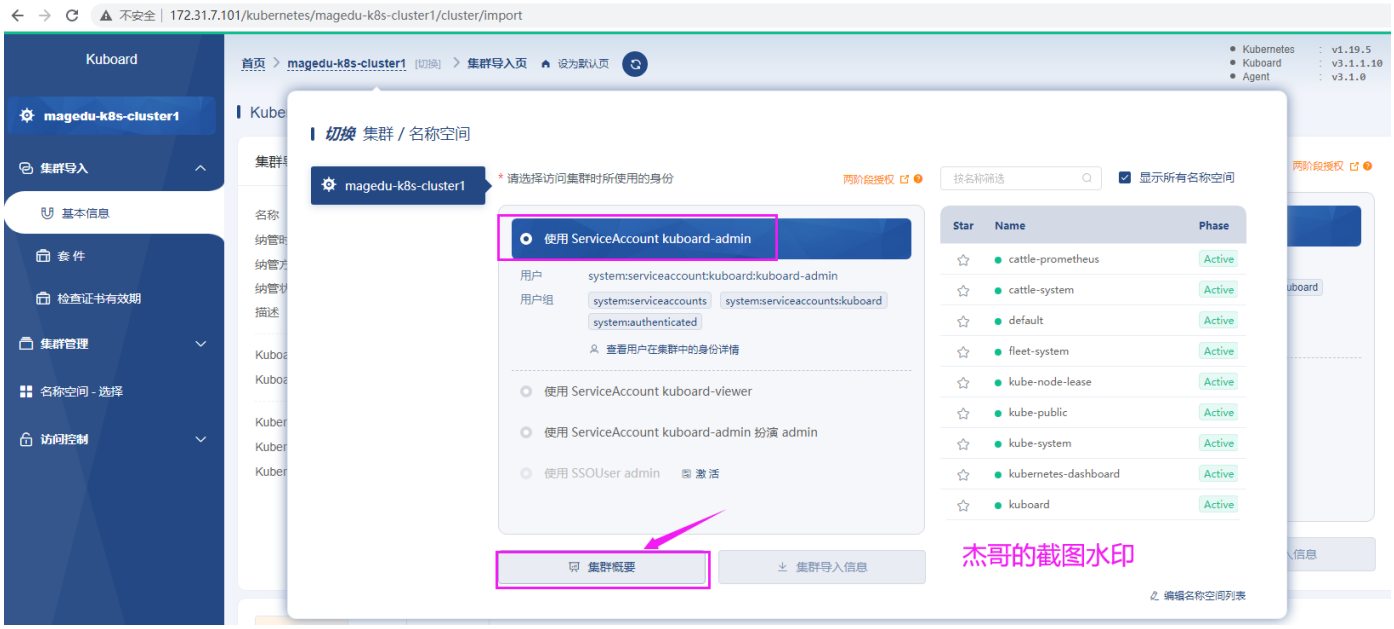
https://kuboard.cn/overview/#kuboard%E5%9C%A8%E7%BA%BF%E4%BD%93%E9%AA%8C

在浏览器输入 `http://your-host-ip:80` 即可访问 `Kuboard v3.x` 的界面，登录方式：

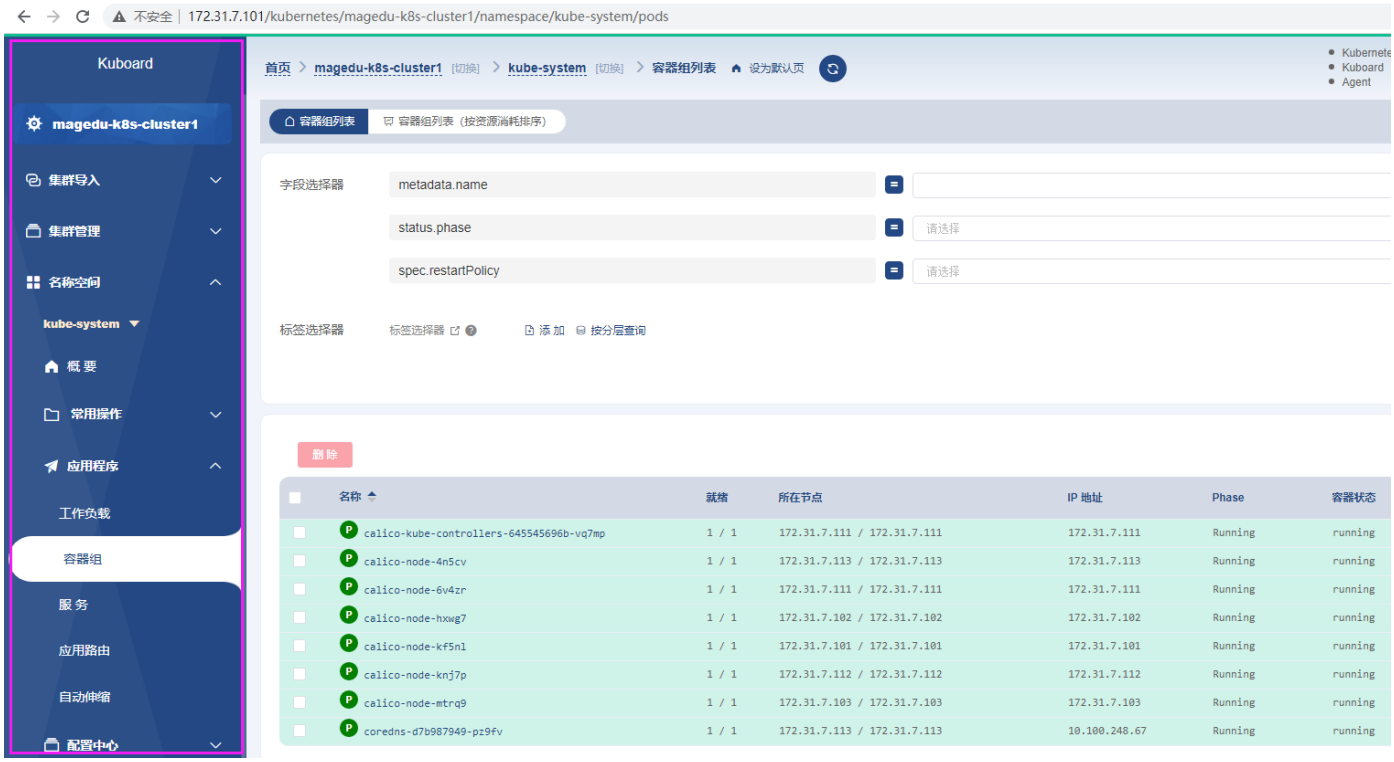
用户名： `admin`
密 码： `Kuboard123`



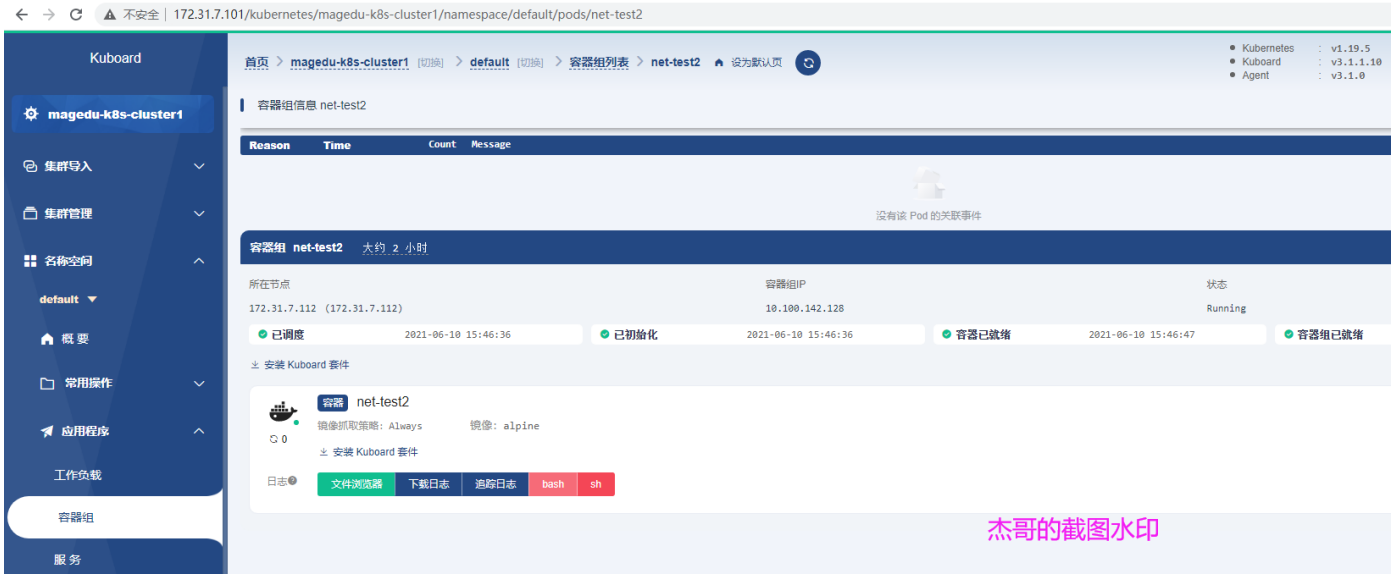




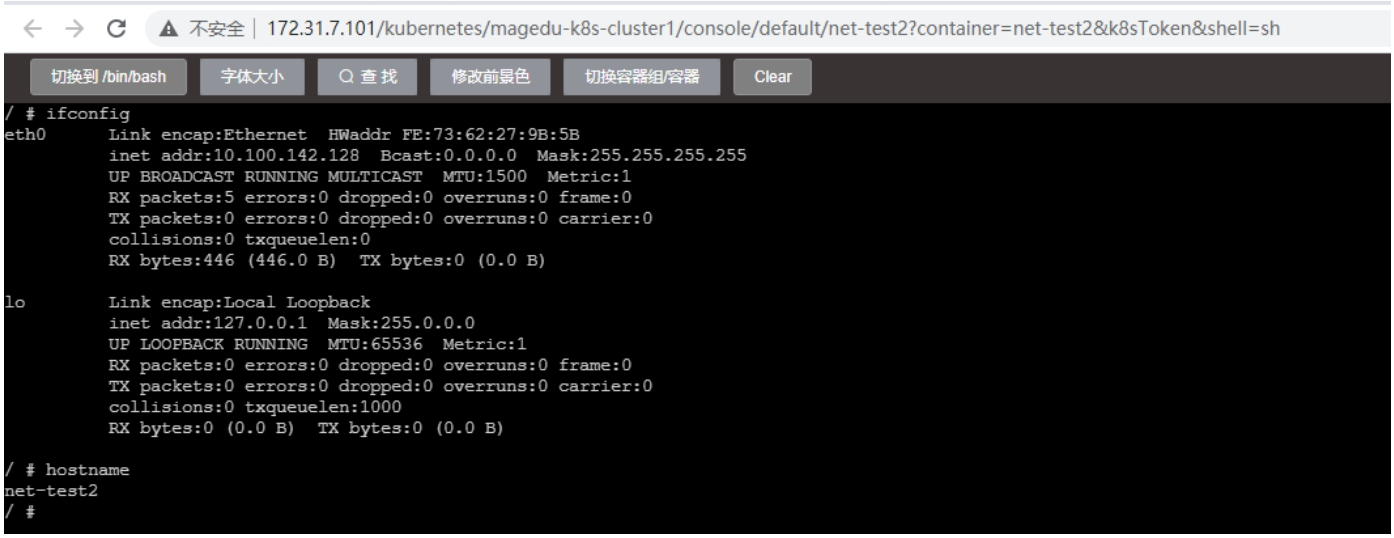
对k8s中的容器和其他资源进行管理：



#进入容器进行单独日志查等操作：



进入容器进行操作：



1.8：DNS服务：

目前常用的dns组件有kube-dns和coredns两个，即到目前k8s版本 1.17.X都可以使用，kube-dns和coredns用于解析k8s集群中service name所对应得到IP地址。

1.8.1：部署kube-dns：

k8s 1.18版本后将不再支持kube-dns。

https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.18.md#downloads-for-v1180

kubeadm: kube-dns is deprecated and will not be supported in a future version

```
1.skyDNS/kube-dns/coreDNS
kube-dns：提供service name域名的解析
```

```
dns-dnsmasq：提供DNS缓存，降低kubedns负载，提高性能
dns-sidecar：定期检查kubedns和dnsmasq的健康状态

2.导入镜像并上传至本地harbor
# docker load -i k8s-dns-kube-dns-amd64_1.14.13.tar.gz
# docker images
# docker tag gcr.io/google-containers/k8s-dns-kube-dns-amd64:1.14.13
harbor.magedu.net/baseimages/k8s-dns-kube-dns-amd64:1.14.13
# docker push harbor.magedu.net/baseimages/k8s-dns-kube-dns-amd64:1.14.13


# docker load -i  k8s-dns-sidecar-amd64_1.14.13.tar.gz
# docker images
# docker tag gcr.io/google-containers/k8s-dns-sidecar-amd64:1.14.13
harbor.magedu.net/baseimages/k8s-dns-sidecar-amd64:1.14.13
# docker push harbor.magedu.net/baseimages/k8s-dns-sidecar-amd64:1.14.13

# docker load -i k8s-dns-dnsmasq-nanny-amd64_1.14.13.tar.gz
# docker imgaes
# docker images
# docker  tag gcr.io/google-containers/k8s-dns-dnsmasq-nanny-amd64:1.14.13
harbor.magedu.net/baseimages/k8s-dns-dnsmasq-nanny-amd64:1.14.13
# docker push harbor.magedu.net/baseimages/k8s-dns-dnsmasq-nanny-amd64:1.14.13

3.修改yaml文件中的镜像地址为本地harbor地址
# vim kube-dns.yaml
      - name: kubedns
        image: harbor.magedu.net/baseimages/k8s-dns-kube-dns-amd64:1.14.13

      - name: dnsmasq
        image: harbor.magedu.net/baseimages/k8s-dns-dnsmasq-nanny-amd64:1.14.13

      - name: sidecar
        image: harbor.magedu.net/baseimages/k8s-dns-sidecar-amd64:1.14.13


4.创建服务
# kubectl  apply -f  kube-dns.yaml
```

1.8.2：部署coredns：

https://github.com/coredns/coredns

coredns 1.2/1.3/1.4/1.5版本：

```
# docker tag gcr.io/google-containers/coredns:1.2.6
harbor.magedu.net/baseimages/coredns:1.2.6
# docker push harbor.magedu.net/baseimages/coredns:1.2.6
```

1.6版本：

https://github.com/coredns/deployment/tree/master/kubernetes #1.6部署方式

```
# unzip deployment-master.zip
# ./deploy.sh  10.20.0.0/16  > magedu-coredns.yaml
# vim magedu-coredns.yaml #修改域名
```

1.8.3：域名解析测试：

```
# kubectl  delete -f /etc/ansible/manifests/dns/kube-dns/kube-dns.yaml #删除kube-dns
# kubectl  apply -f  coredns.yaml  #部署coredns

# kubectl exec busybox nslookup kubernetes
Server:    10.20.254.254
Address 1: 10.20.254.254 kube-dns.kube-system.svc.linux36.local

Name:      kubernetes
Address 1: 10.20.0.1 kubernetes.default.svc.linux36.local

# kubectl exec busybox nslookup kubernetes.default.svc.linux36.local
Server:    10.20.254.254
Address 1: 10.20.254.254 kube-dns.kube-system.svc.linux36.local

Name:      kubernetes.default.svc.linux36.local
Address 1: 10.20.0.1 kubernetes.default.svc.linux36.local

#  kubectl exec busybox nslookup kube-dns.kube-system.svc.magedu.local
Server:    10.10.0.2
Address 1: 10.10.0.2 kube-dns.kube-system.svc.magedu.local

Name:      kube-dns.kube-system.svc.magedu.local
Address 1: 10.10.0.2 kube-dns.kube-system.svc.magedu.local
```

1.8.4：监控组件heapster：

heapster：数据采集

influxdb：数据存储

grafana：web展示

```
1.导入相应的镜像
2.更改yaml中的镜像地址
3.创建服务
```

二：k8s运行机制及术语：

分解k8s集群中master、node、etcd、calico和flannel的运行机制

2.1：master运行机制：

master的运行机制

2.1.1：kube-apiserver：

https://kubernetes.io/zh/docs/reference/command-line-tools-reference/kube-apiserver/

k8s API Server提供了k8s各类资源对象（pod,RC,Service等）的增删改查及watch等HTTP Rest接口，是整个系统的数据总线和数据中心。

apiserver 目前在master监听两个端口，通过 --insecure-port int 监听一个非安全的127.0.0.1本地端口(默认为8080)：

```
该端口用于接收HTTP请求；
该端口默认值为8080，可以通过API Server的启动参数“--insecure-port”的值来修改默认值；
默认的IP地址为“localhost”，可以通过启动参数“--insecure-bind-address”的值来修改该IP地址；
非认证或未授权的HTTP请求通过该端口访问API Server(kube-controller-manager、kube-scheduler)。
```

通过参数--bind-address=192.168.7.101 监听一个对外访问且安全(https)的端口(默认为6443)：

```
该端口默认值为6443，可通过启动参数“--secure-port”的值来修改默认值；
默认IP地址为非本地（Non-Localhost）网络端口，通过启动参数“--bind-address”设置该值；
该端口用于接收客户端、dashboard等外部HTTPS请求；
用于基于Tocken文件或客户端证书及HTTP Base的认证；
用于基于策略的授权；
```

kubernetes API Server的功能与使用：

```
提供了集群管理的REST API接口(包括认证授权、数据校验以及集群状态变更)；
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）；
是资源配额控制的入口；
拥有完备的集群安全机制.

# curl 127.0.0.1:8080/apis #分组api
# curl 127.0.0.1:8080/api/v1 #带具体版本号的api
# curl  127.0.0.1:8080/  #返回核心api列表
# curl  127.0.0.1:8080/version #api 版本信息
# curl 127.0.0.1:8080/healthz/etcd #与etcd的心跳监测
# curl 127.0.0.1:8080/apis/autoscaling/v1 #api的详细信息
# curl  127.0.0.1:8080/metrics #指标数据
```

启动脚本：

```
cat /etc/systemd/system/kube-apiserver.service
```

```
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target

[Service]
ExecStart=/usr/bin/kube-apiserver \
  --admission-
control=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,ResourceQuota
,NodeRestriction,MutatingAdmissionWebhook,ValidatingAdmissionWebhook \
  --bind-address=192.168.7.101 \ #外部监听端口
  --insecure-bind-address=127.0.0.1 \  #本机监听端口
  --authorization-mode=Node,RBAC \
  --kubelet-https=true \
  --kubelet-client-certificate=/etc/kubernetes/ssl/admin.pem \
  --kubelet-client-key=/etc/kubernetes/ssl/admin-key.pem \
  --anonymous-auth=false \
  --basic-auth-file=/etc/kubernetes/ssl/basic-auth.csv \
  --service-cluster-ip-range=10.20.0.0/16 \
  --service-node-port-range=30000-60000 \
  --tls-cert-file=/etc/kubernetes/ssl/kubernetes.pem \
  --tls-private-key-file=/etc/kubernetes/ssl/kubernetes-key.pem \
  --client-ca-file=/etc/kubernetes/ssl/ca.pem \
  --service-account-key-file=/etc/kubernetes/ssl/ca-key.pem \
  --etcd-cafile=/etc/kubernetes/ssl/ca.pem \
  --etcd-certfile=/etc/kubernetes/ssl/kubernetes.pem \
  --etcd-keyfile=/etc/kubernetes/ssl/kubernetes-key.pem \
  --etcd-
servers=https://192.168.7.105:2379,https://192.168.7.106:2379,https://192.168.7.107:237
9 \
  --enable-swagger-ui=true \
  --endpoint-reconciler-type=lease \
  --allow-privileged=true \
  --audit-log-maxage=30 \
  --audit-log-maxbackup=3 \
  --audit-log-maxsize=100 \
  --audit-log-path=/var/lib/audit.log \
  --event-ttl=1h \
  --requestheader-client-ca-file=/etc/kubernetes/ssl/ca.pem \
  --requestheader-allowed-names= \
  --requestheader-extra-headers-prefix=X-Remote-Extra- \
  --requestheader-group-headers=X-Remote-Group \
  --requestheader-username-headers=X-Remote-User \
  --proxy-client-cert-file=/etc/kubernetes/ssl/aggregator-proxy.pem \
  --proxy-client-key-file=/etc/kubernetes/ssl/aggregator-proxy-key.pem \
  --enable-aggregator-routing=true \
  --runtime-config=batch/v2alpha1=true \
  --v=2
```

```
Restart=on-failure
RestartSec=5
Type=notify
LimitNOFILE=65536

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

2.1.2：kube-controller-manager：

Controller Manager作为集群内部的管理控制中心，非安全默认端口10252，负责集群内的Node、Pod副本、服务端点（Endpoint）、命名空间（Namespace）、服务账号（ServiceAccount）、资源定额（ResourceQuota）的管理，当某个Node意外宕机时，Controller Manager会及时发现并执行自动化修复流程，确保集群始终处于预期的工作状态。

启动脚本：

```
# vim /etc/systemd/system/kube-controller-manager.service

[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/GoogleCloudPlatform/kubernetes

[Service]
ExecStart=/usr/bin/kube-controller-manager \
  --address=127.0.0.1 \
  --master=http://127.0.0.1:8080 \ #调用kube-api-server的本地端口进行通信
  --allocate-node-cidrs=true \
  --service-cluster-ip-range=10.20.0.0/16 \
  --cluster-cidr=172.31.0.0/16 \
  --cluster-name=kubernetes \
  --cluster-signing-cert-file=/etc/kubernetes/ssl/ca.pem \
  --cluster-signing-key-file=/etc/kubernetes/ssl/ca-key.pem \
  --service-account-private-key-file=/etc/kubernetes/ssl/ca-key.pem \
  --root-ca-file=/etc/kubernetes/ssl/ca.pem \
  --horizontal-pod-autoscaler-use-rest-clients=true \
  --leader-elect=true \
  --v=2
Restart=on-failure
RestartSec=5

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

2.1.3：kube-scheduler：

Scheduler负责Pod调度，在整个系统中起"承上启下"作用，承上：负责接收Controller Manager创建的新的Pod，为其选择一个合适的Node；启下：Node上的kubelet接管Pod的生命周期。

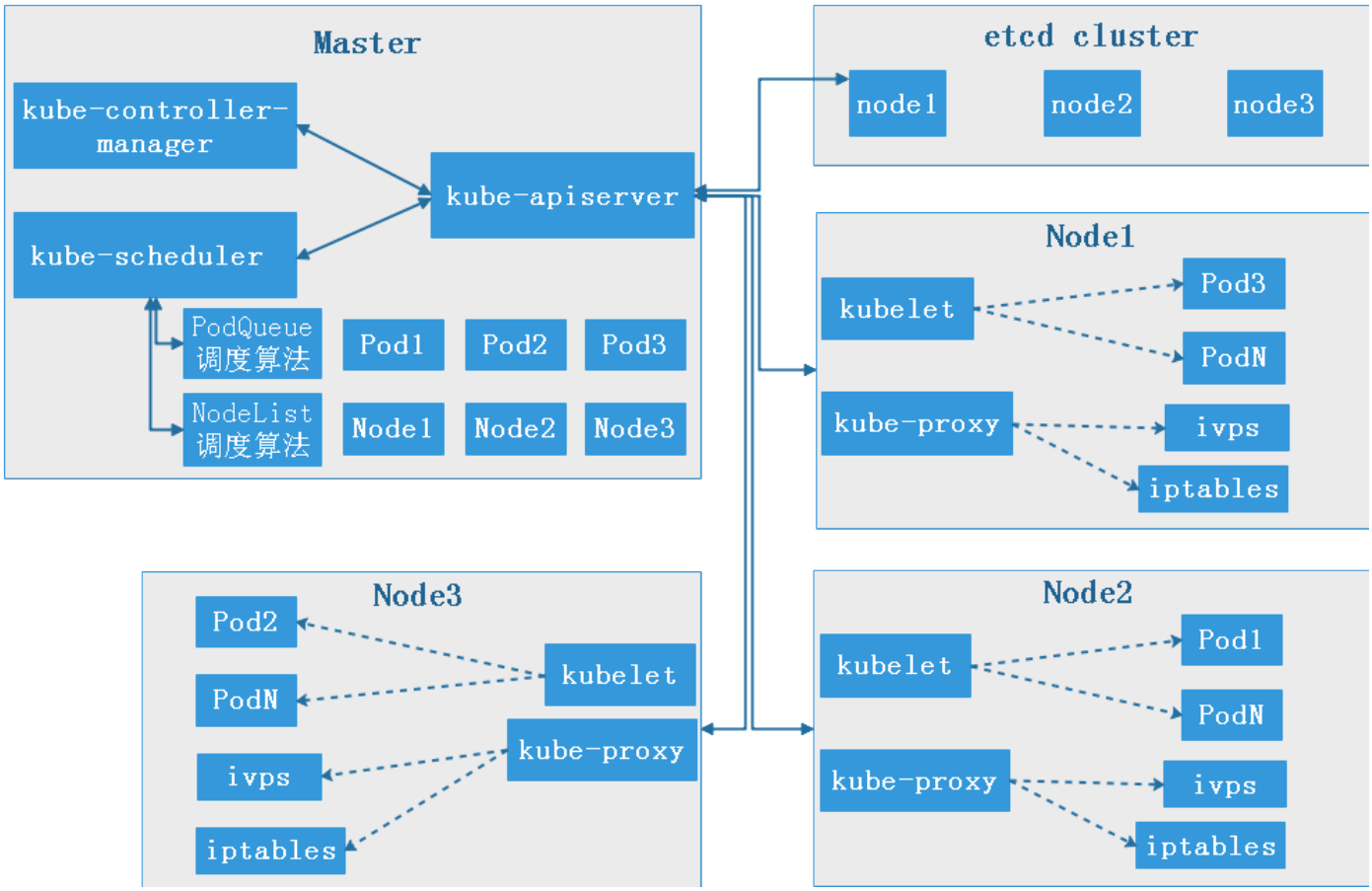
启动脚本：

```
# vim /etc/systemd/system/kube-scheduler.service

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

[Service]
ExecStart=/usr/bin/kube-scheduler \
  --address=127.0.0.1 \
  --master=http://127.0.0.1:8080 \ #调用kube-api-server的本地端口进行通信
  --leader-elect=true \
  --v=2
Restart=on-failure
RestartSec=5

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

通过调度算法为待调度Pod列表的每个Pod从可用Node列表中选择一个最适合的Node，并将信息写入etcd中
node节点上的kubelet通过API Server监听到kubernetes Scheduler产生的Pod绑定信息，然后获取对应的Pod清单，下载Image，并启动容器。

优选策略
1.LeastRequestedPriority
优先从备选节点列表中选择资源消耗最小的节点（CPU+内存）。
2.CalculateNodeLabelPriority
优先选择含有指定Label的节点。
3.BalancedResourceAllocation
优先从备选节点列表中选择各项资源使用率最均衡的节点。

图片：



2.2：node节点运行机制：

2.1.2：kubelet：

在kubernetes集群中，每个Node节点都会启动kubelet进程，用来处理Master节点下发到本节点的任务，管理Pod和其中的容器。kubelet会在API Server上注册节点信息，定期向Master汇报节点资源使用情况，并通过cAdvisor(顾问)监控容器和节点资源，可以把kubelet理解成Server/Agent架构中的agent，kubelet是Node上的pod管家。

https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/#options

启动脚本：

```
root@k8s-node2:~# cat  /etc/systemd/system/kubelet.service
[Unit]
Description=Kubernetes Kubelet
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=docker.service
Requires=docker.service

[Service]
WorkingDirectory=/var/lib/kubelet
ExecStart=/usr/bin/kubelet \
  --address=192.168.7.111 \
  --allow-privileged=true \
```

```
  --anonymous-auth=false \
  --authentication-token-webhook \
  --authorization-mode=Webhook \
  --client-ca-file=/etc/kubernetes/ssl/ca.pem \
  --cluster-dns=10.20.254.254 \
  --cluster-domain=linux36.local. \
  --cni-bin-dir=/usr/bin \
  --cni-conf-dir=/etc/cni/net.d \
  --fail-swap-on=false \
  --hairpin-mode hairpin-veth \
  --hostname-override=192.168.7.111 \
  --kubeconfig=/etc/kubernetes/kubelet.kubeconfig \  #配置文件
  --max-pods=110 \
  --network-plugin=cni \
  --pod-infra-container-image=harbor.magedu.net/baseimages/pause-amd64:3.1 \
  --register-node=true \
  --root-dir=/var/lib/kubelet \
  --tls-cert-file=/etc/kubernetes/ssl/kubelet.pem \
  --tls-private-key-file=/etc/kubernetes/ssl/kubelet-key.pem \
  --v=2
#kubelet cAdvisor 默认在所有接口监听 4194 端口的请求, 以下iptables限制内网访问
ExecStartPost=/sbin/iptables -A INPUT -s 10.0.0.0/8 -p tcp --dport 4194 -j ACCEPT
ExecStartPost=/sbin/iptables -A INPUT -s 172.16.0.0/12 -p tcp --dport 4194 -j ACCEPT
ExecStartPost=/sbin/iptables -A INPUT -s 192.168.0.0/16 -p tcp --dport 4194 -j ACCEPT
ExecStartPost=/sbin/iptables -A INPUT -p tcp --dport 4194 -j DROP
Restart=on-failure
RestartSec=5

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


PLEG：即Pod Lifecycle Event Generator,pleg会记录Pod生命周期中的各种事件,如容器的启动、终止等。

Container GC：Kubernetes 垃圾回收（Garbage Collection）机制由kubelet完成，kubelet定期清理不再使用的容器和镜像，每分钟进行一次容器的GC，每五分钟进行一次镜像的GC。

Pod Eviction 是k8s一个特色功能，它在某些场景下应用，如节点NotReady、Node节点资源不足，把pod驱逐至其它Node节点。

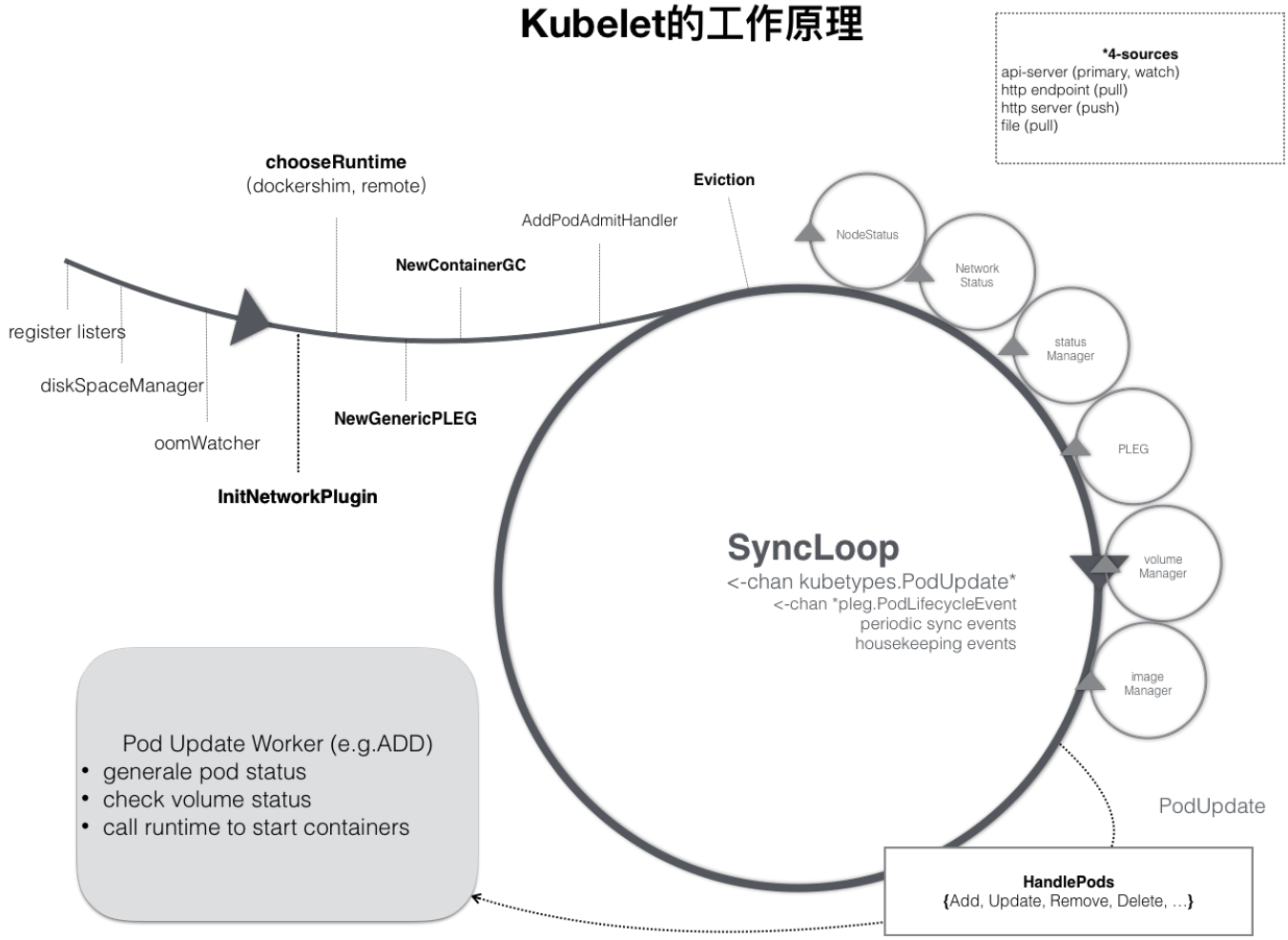
Kube-controller-manager：周期性检查所有节点状态，当节点处于NotReady状态超过一段时间后，驱逐该节点上所有 pod。

Kubelet：周期性检查本节点资源，当资源不足时，按照优先级驱逐部分 pod。

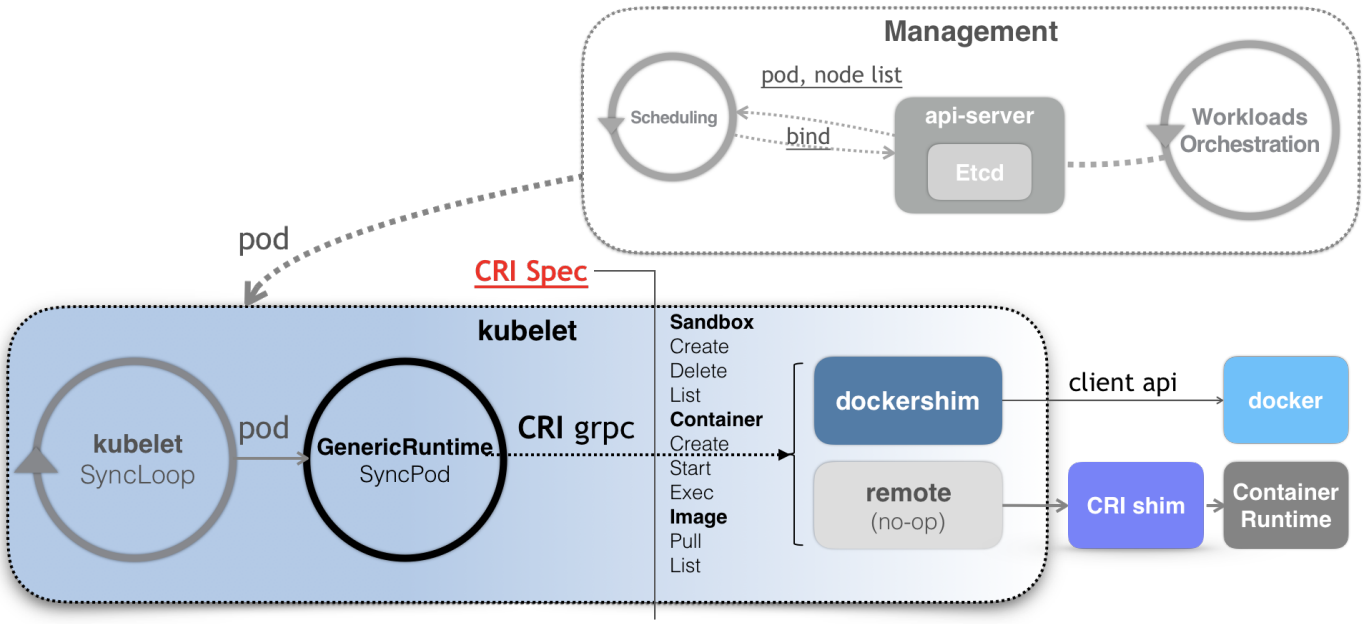
SyncLoop：控制pod 生命周期的制循环，驱动整个控制循环的事件有：pod更新事件、pod生命周期变化、kubelet本身设置的执行周期、定时清理事件等。

handlepods：kubelet针对pod的处理程序，创建pod、删除pod等。

Kubelet的工作原理



How CRI Works



2.1.2：kube-proxy：

kube-proxy 运行在每个节点上，监听 API Server 中服务对象的变化，再通过管理 IPtables或者IPVS规则 来实现网络的转发

Kube-Proxy 不同的版本可支持三种工作模式：

https://kubernetes.io/zh/docs/concepts/services-networking/service/ #service介绍

https://kubernetes.io/zh/blog/2018/07/09/ipvs-based-in-cluster-load-balancing-deep-dive/ #使用IPVS

```
UserSpace
  k8s v1.2 及以后就已经淘汰
IPtables
  目前默认方式，1.1开始支持，1.2开始为默认
IPVS
  1.9引入到1.11正式版本，需要安装ipvsadm、ipset 工具包和加载 ip_vs 内核模块
```

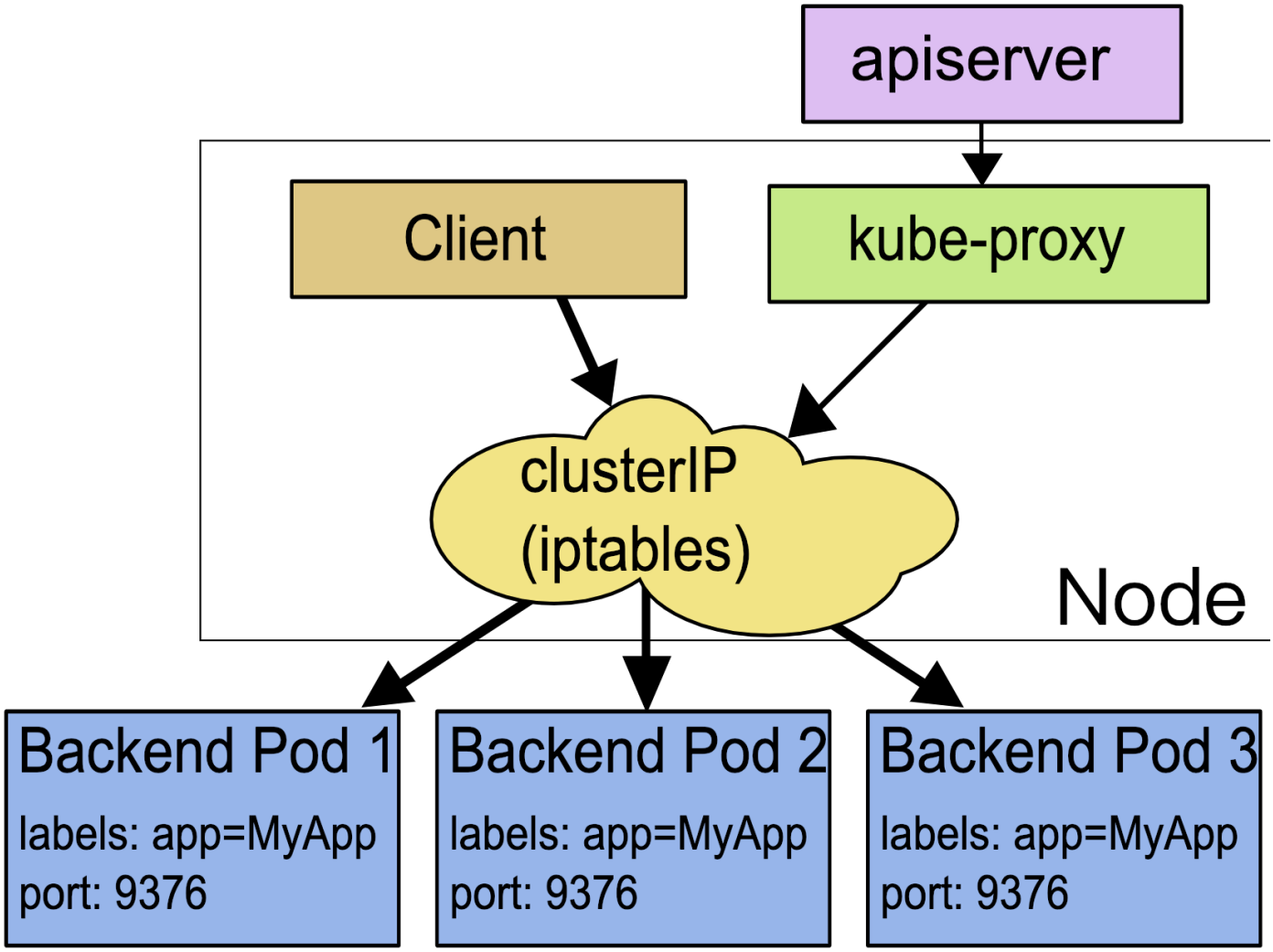
启动脚本：

```
root@k8s-node2:~# cat  /etc/systemd/system/kube-proxy.service
[Unit]
Description=Kubernetes Kube-Proxy Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target

[Service]
WorkingDirectory=/var/lib/kube-proxy
ExecStart=/usr/bin/kube-proxy \
  --bind-address=192.168.7.111 \
  --hostname-override=192.168.7.111 \
  --kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig \ #配置文件
  --logtostderr=true \
  --proxy-mode=iptables #配置转发模式
Restart=on-failure
RestartSec=5
LimitNOFILE=65536

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

2.1.2.1：iptables：

Kube-Proxy 监听 Kubernetes Master 增加和删除 Service 以及 Endpoint 的消息。对于每一个 Service，Kube Proxy 创建相应的 IPtables 规则，并将发送到 Service Cluster IP 的流量转发到 Service 后端提供服务的 Pod 的相应端口上。
注：
虽然可以通过 Service 的 Cluster IP 和服务端口访问到后端 Pod 提供的服务，但该 Cluster IP 是 Ping 不通的。其原因是 Cluster IP 只是 IPtables 中的规则，并不对应到一个任何网络设备。

IPVS 模式的 Cluster IP 是可以 Ping 通的。



2.1.2.2：IPVS：

kubernetes从1.9开始测试支持ipvs(Graduate kube-proxy IPVS mode to beta)，https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG-1.9.md#ipvs，从1.11版本正式支持ipvs(IPVS-based in-cluster load balancing is now GA)，https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG-1.11.md#ipvs。

IPVS 相对 IPtables 效率会更高一些，使用 IPVS 模式需要在运行 Kube-Proxy 的节点上安装 ipvsadm、ipset 工具包和加载 ip_vs 内核模块，当 Kube-Proxy 以 IPVS 代理模式启动时，Kube-Proxy 将验证节点上是否安装了 IPVS 模块，如果未安装，则 Kube-Proxy 将回退到 IPtables 代理模式。

> 使用IPVS模式，Kube-Proxy会监视Kubernetes Service对象和Endpoints，调用宿主机内核Netlink接口以相应地创建IPVS规则并定期与Kubernetes Service对象 Endpoints对象同步IPVS规则，以确保IPVS状态与期望一致，访问服务时，流量将被重定向到其中一个后端 Pod,IPVS使用哈希表作为底层数据结构并在内核空间中工作，这意味着IPVS可以更快地重定向流量，并且在同步代理规则时具有更好的性能，此外，IPVS 为负载均衡算法提供了更多选项，例如：rr (轮询调度)、lc (最小连接数)、dh (目标哈希)、sh (源哈希)、sed (最短期望延迟)、nq(不排队调度)等。

配置方式：

配置使用IPVS及指定调度算法：

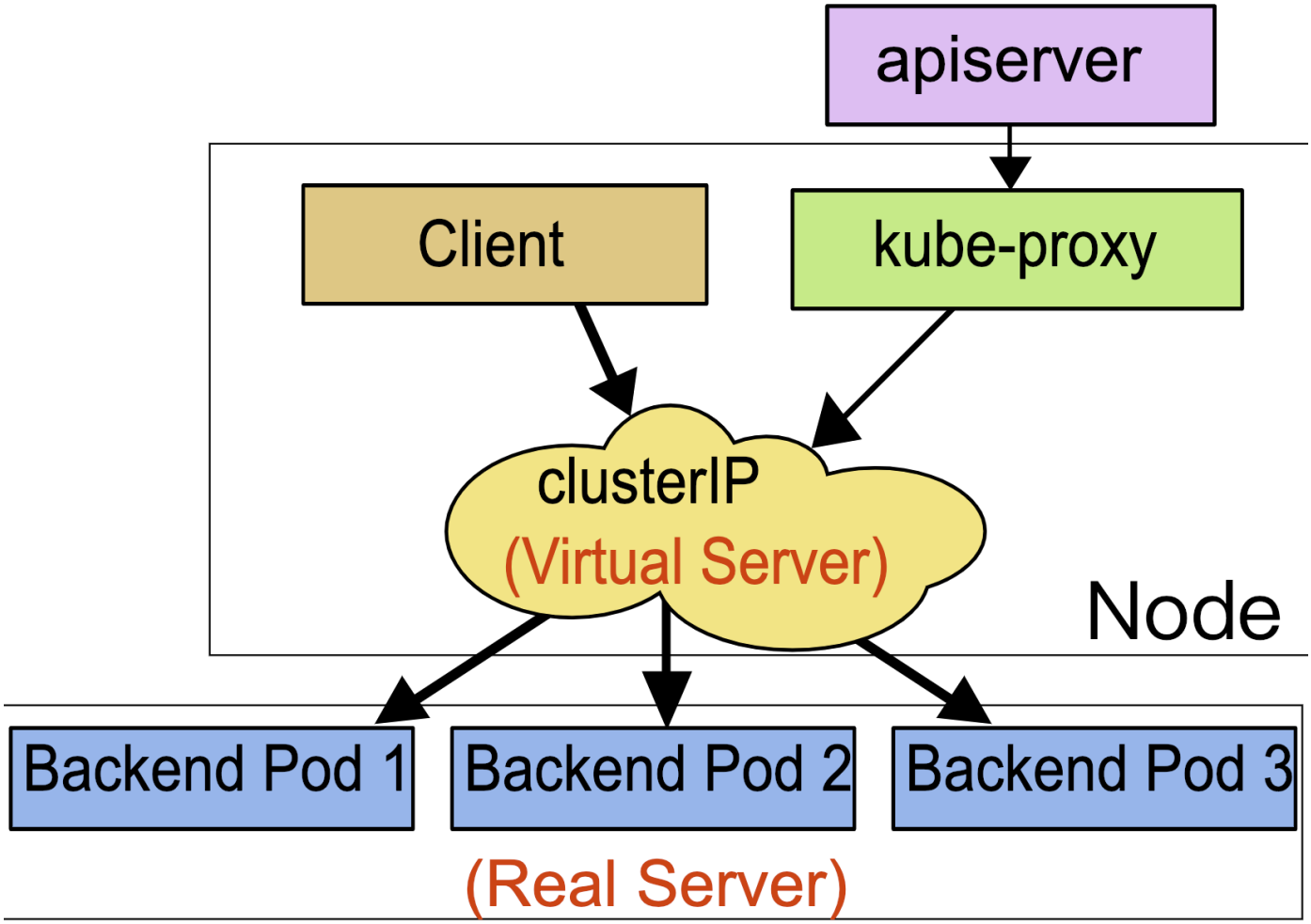
https://kubernetes.io/zh/docs/reference/config-api/kube-proxy-config.v1alpha1/#ClientConnectionConfiguration

```
# cat  /var/lib/kube-proxy/kube-proxy-config.yaml
kind: KubeProxyConfiguration
apiVersion: kubeproxy.config.k8s.io/v1alpha1
bindAddress: 172.31.7.111
clientConnection:
  kubeconfig: "/etc/kubernetes/kube-proxy.kubeconfig"
clusterCIDR: "10.100.0.0/16"
conntrack:
  maxPerCore: 32768
  min: 131072
  tcpCloseWaitTimeout: 1h0m0s
  tcpEstablishedTimeout: 24h0m0s
healthzBindAddress: 172.31.7.111:10256
hostnameOverride: "172.31.7.111"
metricsBindAddress: 172.31.7.111:10249
mode: "ipvs"
ipvs:
  scheduler: sh
```

开启service会话保持：

```
kind: Service
apiVersion: v1
metadata:
  labels:
    app: magedu-nginx-service-label
  name: magedu-nginx-service
  namespace: magedu
spec:
  type: NodePort
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
    nodePort: 30004
  selector:
    app: nginx
  sessionAffinity: ClientIP
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 1800
```

```
TCP  172.31.7.111:30004 sh persistent 1800
  -> 10.100.39.88:80                Masq    1      0          0
  -> 10.100.248.89:80               Masq    1      0          0
```

2.3：etcd运行机制：

etcd是CoreOS团队于2013年6月发起的开源项目，它的目标是构建一个高可用的分布式键值(key-value)数据库。etcd内部采用raft协议作为一致性算法，etcd基于Go语言实现。

官方网站：https://etcd.io/

github地址：https://github.com/etcd-io/etcd

https://etcd.io/docs/v3.4/op-guide/hardware/ #官方硬件推荐

```
Etcd具有下面这些属性：
完全复制：集群中的每个节点都可以使用完整的存档
高可用性：Etcd可用于避免硬件的单点故障或网络问题
一致性：每次读取都会返回跨多主机的最新写入
简单：包括一个定义良好、面向用户的API（gRPC）
安全：实现了带有可选的客户端证书身份验证的自动化TLS
快速：每秒10000次写入的基准速度
可靠：使用Raft算法实现了存储的合理分布Etcd的工作原理
```

2.3.1：启动脚本参数：

```
root@k8s-etcd1:~# cat /etc/systemd/system/etcd.service
[Unit]
Description=Etcd Server
After=network.target
After=network-online.target
Wants=network-online.target
Documentation=https://github.com/coreos

[Service]
Type=notify
WorkingDirectory=/var/lib/etcd/ #数据保存目录
ExecStart=/usr/bin/etcd \ #二进制文件路径
  --name=etcd1 \  #当前node 名称
  --cert-file=/etc/etcd/ssl/etcd.pem \
  --key-file=/etc/etcd/ssl/etcd-key.pem \
  --peer-cert-file=/etc/etcd/ssl/etcd.pem \
  --peer-key-file=/etc/etcd/ssl/etcd-key.pem \
  --trusted-ca-file=/etc/kubernetes/ssl/ca.pem \
  --peer-trusted-ca-file=/etc/kubernetes/ssl/ca.pem \
  --initial-advertise-peer-urls=https://192.168.7.105:2380 \ #通告自己的集群端口
  --listen-peer-urls=https://192.168.7.105:2380 \ #集群之间通讯端口
  --listen-client-urls=https://192.168.7.105:2379,http://127.0.0.1:2379 \ #客户端访问地址
  --advertise-client-urls=https://192.168.7.105:2379 \ #通告自己的客户端端口
  --initial-cluster-token=etcd-cluster-0 \  #创建集群使用的token，一个集群内的节点保持一致
  --initial-
cluster=etcd1=https://192.168.7.105:2380,etcd2=https://192.168.7.106:2380,etcd3=https:/
/192.168.7.107:2380 \  #集群所有的节点信息
  --initial-cluster-state=new \ #新建集群的时候的值为new,如果是已经存在的集群为existing。
  --data-dir=/var/lib/etcd #数据目录路径
Restart=on-failure
RestartSec=5
LimitNOFILE=65536

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

2.3.2：查看成员信息：

etcd有多个不同的API访问版本，v1版本已经废弃，etcd v2 和 v3 本质上是共享同一套 raft 协议代码的两个独立的应用，接口不一样，存储不一样，数据互相隔离。也就是说如果从 Etcd v2 升级到 Etcd v3，原来v2 的数据还是只能用 v2 的接口访问，v3 的接口创建的数据也只能访问通过 v3 的接口访问。

WARNING:
  Environment variable ETCDCTL_API is not set; defaults to etcdctl v2. #默认使用V2版本
  Set environment variable ETCDCTL_API=3 to use v3 API or ETCDCTL_API=2 to use v2 API. #设置API版本

```
root@k8s-etcd1:~# ETCDCTL_API=3 etcdctl --help
root@k8s-etcd1:~# ETCDCTL_API=3 etcdctl member --help
NAME:
   etcdctl member - member add, remove and list subcommands

USAGE:
   etcdctl member command [command options] [arguments...]

COMMANDS:
     list    enumerate existing cluster members
     add     add a new member to the etcd cluster
     remove  remove an existing member from the etcd cluster
     update  update an existing member in the etcd cluster

OPTIONS:
   --help, -h  show help
```

2.3.3：验证当前etcd所有成员状态：

验证etcd集群状态

2.3.3.1：心跳信息:

```
root@k8s-etcd1:~# export NODE_IPS="172.31.7.106 172.31.7.107 172.31.7.108"

root@k8s-etcd1:~# for ip in ${NODE_IPS}; do   ETCDCTL_API=3 /usr/bin/etcdctl   --endpoints=https://${ip}:2379    --cacert=/etc/kubernetes/ssl/ca.pem   --cert=/etc/kubernetes/ssl/etcd.pem   --key=/etc/kubernetes/ssl/etcd-key.pem   endpoint health; done
```

```
root@etcd1:~# export NODE_IPS="172.31.7.106 172.31.7.107 172.31.7.108"
root@etcd1:~# for ip in ${NODE_IPS}; do   ETCDCTL_API=3 /usr/bin/etcdctl   --endpoints=https://${ip}:2379  --cacert=/etc/kubernetes/ssl/ca.pem   --cert=/etc/kubernetes/ssl/etcd.pem   --key=/etc/kubernetes/ssl/etcd-key.pem   endpoint health; done
https://172.31.7.106:2379 is healthy: successfully committed proposal: took = 10.201334ms
https://172.31.7.107:2379 is healthy: successfully committed proposal: took = 12.624038ms
https://172.31.7.108:2379 is healthy: successfully committed proposal: took = 13.778818ms
root@etcd1:~#
```


2.3.3.2：显示集群成员信息：

```
root@k8s-etcd1:~# ETCDCTL_API=3 /usr/bin/etcdctl --write-out=table member list   --endpoints=https://172.31.7.106:2379 --cacert=/etc/kubernetes/ssl/ca.pem   --cert=/etc/kubernetes/ssl/etcd.pem   --key=/etc/kubernetes/ssl/etcd-key.pem
```

```
root@etcd1:~# ETCDCTL_API=3 /usr/bin/etcdctl --write-out=table member list   --endpoints=https://172.31.7.106:2379 --cac
ert=/etc/kubernetes/ssl/ca.pem   --cert=/etc/kubernetes/ssl/etcd.pem   --key=/etc/kubernetes/ssl/etcd-key.pem
+------------------+---------+------------------+--------------------------+--------------------------+------------+
|        ID        | STATUS  |       NAME       |        PEER ADDRS        |       CLIENT ADDRS       | IS LEARNER |
+------------------+---------+------------------+--------------------------+--------------------------+------------+
| 360a6fc679b60f19 | started | etcd-172.31.7.107 | https://172.31.7.107:2380 | https://172.31.7.107:2379 |      false |
| b786e8f07f81b519 | started | etcd-172.31.7.108 | https://172.31.7.108:2380 | https://172.31.7.108:2379 |      false |
| e7e623d2f4a7c9e4 | started | etcd-172.31.7.106 | https://172.31.7.106:2380 | https://172.31.7.106:2379 |      false |
+------------------+---------+------------------+--------------------------+--------------------------+------------+
root@etcd1:~#                                                    杰哥的截图水印
```

2.3.3.3：以表格方式显示节点详细状态：

```
root@k8s-etcd1:~# export NODE_IPS="172.31.7.106 172.31.7.107 172.31.7.108"

root@k8s-etcd1:~# for ip in ${NODE_IPS}; do   ETCDCTL_API=3 /usr/bin/etcdctl --write-
out=table endpoint status   --endpoints=https://${ip}:2379    --
cacert=/etc/kubernetes/ssl/ca.pem   --cert=/etc/kubernetes/ssl/etcd.pem   --
key=/etc/kubernetes/ssl/etcd-key.pem; done
```

```
root@etcd1:~# for ip in ${NODE_IPS}; do   ETCDCTL_API=3 /usr/bin/etcdctl --write-out=table endpoint status   --endpoints=https://${ip}:2379    --cace
rt=/etc/kubernetes/ssl/ca.pem   --cert=/etc/kubernetes/ssl/etcd.pem   --key=/etc/kubernetes/ssl/etcd-key.pem; done
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
|        ENDPOINT          |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
| https://172.31.7.106:2379 | e7e623d2f4a7c9e4 |  3.4.13 |   20 MB |      true |      false |         2 |    1908346 |            1908346 |        |
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
|        ENDPOINT          |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
| https://172.31.7.107:2379 | 360a6fc679b60f19 |  3.4.13 |   20 MB |     false |      false |         2 |    1908346 |            1908346 |        |
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
|        ENDPOINT          |        ID        | VERSION | DB SIZE | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
| https://172.31.7.108:2379 | b786e8f07f81b519 |  3.4.13 |   20 MB |     false |      false |         2 |    1908346 |            1908346 |        |
+--------------------------+------------------+---------+---------+-----------+------------+-----------+------------+--------------------+--------+
root@etcd1:~#                                                    杰哥的截图水印
```

2.3.4：查看etcd数据信息:

查看etcd集群中保存的数据

2.3.4.1：查看所有key：

```
# ETCDCTL_API=3 etcdctl get / --prefix --keys-only #以路径的方式所有key信息

pod信息：
root@etcd1:~# ETCDCTL_API=3 etcdctl get / --prefix --keys-only | grep  pod
/registry/pods/default/busybox
/registry/pods/default/net-test1
/registry/pods/default/net-test2
/registry/pods/default/net-test3
/registry/pods/default/net-test4

namespace信息：
root@etcd1:~# ETCDCTL_API=3 etcdctl get / --prefix --keys-only | grep namespaces
/registry/namespaces/default
/registry/namespaces/kube-node-lease
/registry/namespaces/kube-system
/registry/namespaces/kubernetes-dashboard
/registry/namespaces/magedu
```

```
控制器信息:
root@etcd1:~# ETCDCTL_API=3 etcdctl get / --prefix --keys-only | grep deployment
/registry/deployments/cattle-system/cattle-cluster-agent
/registry/deployments/fleet-system/fleet-agent
/registry/deployments/kube-system/calico-kube-controllers
/registry/deployments/kube-system/coredns

calico组件信息:
root@etcd1:~# ETCDCTL_API=3 etcdctl get / --prefix --keys-only | grep calico
/calico/ipam/v2/assignment/ipv4/block/10.100.142.128-26
/calico/ipam/v2/assignment/ipv4/block/10.100.248.64-26
/calico/ipam/v2/assignment/ipv4/block/10.100.39.64-26
```

2.3.4.2：查看指定的key：

```
root@etcd1:~# ETCDCTL_API=3 etcdctl get
/calico/ipam/v2/assignment/ipv4/block/10.100.39.64-26
```

```
root@etcd1:~# ETCDCTL_API=3 etcdctl get /calico/ipam/v2/assignment/ipv4/block/10.100.39.64-26
/calico/ipam/v2/assignment/ipv4/block/10.100.39.64-26
{"cidr":"10.100.39.64/26","affinity":"host:k8s-node1.magedu.local.priview","allocations":[0,1,null,null,null,null,null,2,null,null,null,null,nul
l,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,nu
ll,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null,null],"unallocated":[8,9,10,11,12,13
,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61
,62,63,2,3,5,4,6],"attributes":[{"handle_id":"k8s-pod-network.95ea13dfc1c9b0183c19138d89b5de8b83c22f0dc2e3058a506c57c17177dcaf","secondary":{"na
mespace":"kubernetes-dashboard","node":"k8s-node1.magedu.local.priview","pod":"kubernetes-dashboard-594944ccf9-bf8n9"}},{"handle_id":"k8s-pod-ne
twork.d12b4109d058e34b5cdeae437a757a0a95f14964f959bb5e6d794b48a997ee31","secondary":{"namespace":"kubernetes-dashboard","node":"k8s-node1.magedu
.local.priview","pod":"dashboard-metrics-scraper-696b4ffdb7-xvjkr"}},{"handle_id":"k8s-pod-network.4d8fea2b2a2adb51cb85a263ba8f38c89f1b4d4cd5628
cc040353763e6e91021","secondary":{"namespace":"kuboard","node":"k8s-node1.magedu.local.priview","pod":"kuboard-agent-2-5db545fcbc-64vpq"}}],"del
eted":false}
root@etcd1:~#
```


2.3.4.3：查看所有calico的数据：

```
root@etcd1:~# ETCDCTL_API=3 etcdctl  get --keys-only --prefix /calico | grep local  #查看etcd中calico的相关数据

/calico/resources/v3/projectcalico.org/nodes/k8s-master1.magedu.local.priview
/calico/resources/v3/projectcalico.org/nodes/k8s-master2.magedu.local.priview
/calico/resources/v3/projectcalico.org/nodes/k8s-master3.magedu.local.priview
/calico/resources/v3/projectcalico.org/nodes/k8s-node1.magedu.local.priview
/calico/resources/v3/projectcalico.org/nodes/k8s-node2.magedu.local.priview
/calico/resources/v3/projectcalico.org/nodes/k8s-node3.magedu.local.priview
```

2.3.5：etcd增删改查数据：

```
#添加数据
root@etcd1:~# ETCDCTL_API=3  /usr/bin/etcdctl put  /name "tom"
OK

#查询数据
```

```
root@etcd1:~# ETCDCTL_API=3  /usr/bin/etcdctl get /name
/name
tom

#改动数据，#直接覆盖就是更新数据
root@etcd1:~#  ETCDCTL_API=3  /usr/bin/etcdctl put  /name "jack"
OK

#验证改动
root@etcd1:~# ETCDCTL_API=3  /usr/bin/etcdctl get /name
/name
jack

#删除数据
root@etcd1:~# ETCDCTL_API=3  /usr/bin/etcdctl del /name
1
root@etcd1:~# ETCDCTL_API=3  /usr/bin/etcdctl get /name
```

2.3.6：etcd数据watch机制：

基于不断监看数据，发生变化就主动触发通知客户端，Etcd v3 的watch机制支持watch某个固定的key，也支持watch一个范围。

相比Etcd v2, Etcd v3的一些主要变化：

```
接口通过grpc提供rpc接口，放弃了v2的http接口，优势是长连接效率提升明显，缺点是使用不如以前方便，尤其对不方便维护长连接的场景。
废弃了原来的目录结构，变成了纯粹的kv，用户可以通过前缀匹配模式模拟目录。
内存中不再保存value，同样的内存可以支持存储更多的key。
watch机制更稳定，基本上可以通过watch机制实现数据的完全同步。
提供了批量操作以及事务机制，用户可以通过批量事务请求来实现Etcd v2的CAS机制（批量事务支持if条件判断）。
```

watch测试：

```
#在etcd node1上watch一个key，没有此key也可以执行watch，后期可以再创建：
root@etcd1:~# ETCDCTL_API=3  /usr/bin/etcdctl watch  /name

#在etcd node2修改数据，验证etcd node1是否能够发现数据变化
root@etcd2:~# ETCDCTL_API=3  /usr/bin/etcdctl put  /data "data v1"
OK
root@etcd2:~# ETCDCTL_API=3  /usr/bin/etcdctl put  /data "data v2"
OK
```

```
root@etcd1:~# ETCDCTL_API=3  /usr/bin/etcdctl watch  /data
PUT
/data
data v1
PUT
/data
data v2
```


2.3.7：etcd数据备份与恢复机制：

WAL是write ahead log的缩写，顾名思义，也就是在执行真正的写操作之前先写一个日志，预写日志。

wal: 存放预写式日志,最大的作用是记录了整个数据变化的全部历程。在etcd中，所有数据的修改在提交前，都要先写入到WAL中。

2.3.7.1：etcd v2版本数据备份与恢复：

```
V2版本帮助信息：
root@k8s-etcd2:~# /usr/bin/etcdctl backup --help
NAME:
   etcdctl backup - backup an etcd directory

USAGE:
   etcdctl backup [command options]

OPTIONS:
   --data-dir value        Path to the etcd data dir #源数据目录
   --wal-dir value         Path to the etcd wal dir
   --backup-dir value      Path to the backup dir #备份目录
   --backup-wal-dir value  Path to the backup wal dir

V2版本备份数据：
root@k8s-etcd2:~# ETCDCTL_API=2 etcdctl backup  --data-dir /var/lib/etcd/ --backup-dir
/opt/etcd_backup
2019-07-11 18:59:57.674432 I | wal: segmented wal file
/opt/etcd_backup/member/wal/0000000000000001-0000000000017183.wal is created

V2版本恢复数据：
#恢复帮助信息：
root@k8s-etcd2:~# etcd --help | grep force
  --force-new-cluster 'false'
    force to create a new one-member cluster.

etcd --data-dir=/var/lib/etcd/default.etcd --force-new-cluster &

root@k8s-etcd2:~# vim /etc/systemd/system/etcd.service
[Unit]
Description=Etcd Server
```

```
After=network.target
After=network-online.target
Wants=network-online.target
Documentation=https://github.com/coreos

[Service]
Type=notify
WorkingDirectory=/var/lib/etcd/
ExecStart=/usr/bin/etcd  \
  --name=etcd2 \
.................................
  --data-dir=/opt/etcd_backup  -force-new-cluster #强制设置为为新集群
Restart=on-failure
RestartSec=5
LimitNOFILE=65536

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

2.3.7.2：etcd 集群v3版本数据手动备份与恢复：

```
V3版本备份数据：
root@k8s-etcd2:~# ETCDCTL_API=3 etcdctl  snapshot save  snapshot.db
Snapshot saved at snapshot.db

V3版本恢复数据：
root@k8s-etcd2:~#  ETCDCTL_API=3  etcdctl snapshot restore  snapshot.db  --data-
dir=/opt/etcd-testdir #将数据恢复到一个新的不存在的目录中
2019-07-11 18:57:46.526757 I | mvcc: restore compact to 74541
2019-07-11 18:57:46.536114 I | etcdserver/membership: added member 8e9e05c52164694d
[http://localhost:2380] to cluster cdf818194e3a8c32

#自动备份数据
root@k8s-etcd2:~# mkdir /data/etcd-backup-dir/ -p
root@k8s-etcd2:~# cat  script.sh
#!/bin/bash
source /etc/profile
DATE=`date +%Y-%m-%d_%H-%M-%S`
ETCDCTL_API=3 /usr/bin/etcdctl  snapshot save  /data/etcd-backup-dir/etcd-snapshot-
${DATE}.db
```

2.3.7.3：etcd 集群v3版本数据自动备份与恢复：

```
root@k8s-master1:/etc/kubeasz# ./ezctl backup  k8s-01

root@k8s-master1:/etc/kubeasz# kubectl  delete pod net-test4
pod "net-test4" deleted

root@k8s-master1:/etc/kubeasz# ./ezctl restore k8s-01
```

2.3.7.4：ETCD数据恢复流程：

当etcd集群宕机数量超过集群总节点数一半以上的时候(如总数为三台宕机两台)，就会导致整合集群宕机，后期需要重新恢复数据，则恢复流程如下：

```
1、恢复服务器系统
2、重新部署ETCD集群
3、停止kube-apiserver/controller-manager/scheduler/kubelet/kube-proxy
4、停止ETCD集群
5、各ETCD节点恢复同一份备份数据
6、启动各节点并验证ETCD集群
7、启动kube-apiserver/controller-manager/scheduler/kubelet/kube-proxy
8、验证k8s master状态及pod数据
```

2.3.7.5：ETCD集群节点添加与删除:

节点维护主要是节点的添加或删除

```
add-etcd
del-etcd
```

2.4：网络通信机制

k8s中的网络主要涉及到pod的的各种访问需求，如同一pod的内部(单容器或者多容器)通信、pod A与pod B的通信、从外部网络访问pod以及从pod访问外部网络。

k8s的网络基于第三方插件实现，但是定义了一些插件兼容规范，该规范有CoreOS和Google联合定制，叫做CNI(Container Network Interface)。

https://kubernetes.io/zh/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/ #CNI简介

https://github.com/containernetworking/cni/blob/spec-v0.4.0/SPEC.md #CNI版本

目前常用的的CNI网络插件有calico和flannel：

2.4.1：calico：

官网：https://www.projectcalico.org/

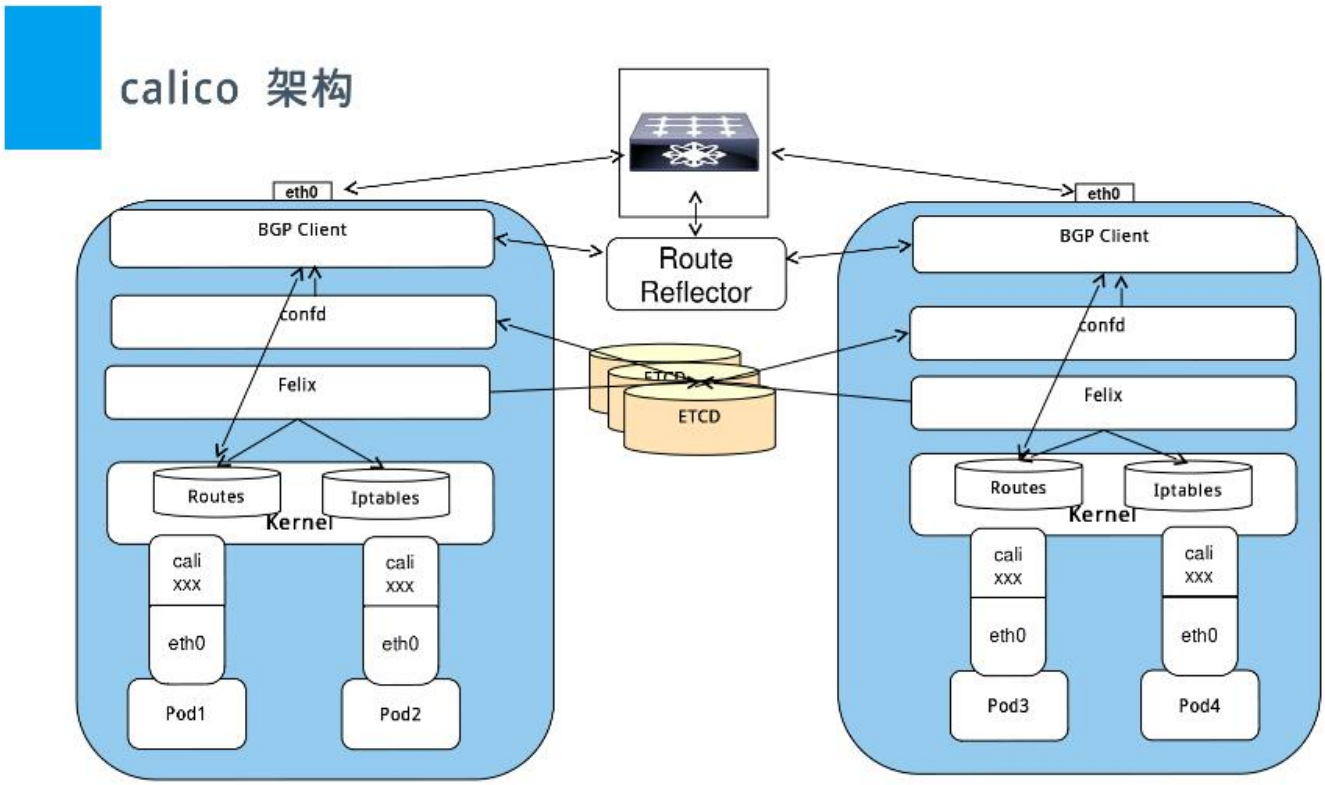
Calico是一个纯三层的网络解决方案，为容器提供多node间的访问通信，calico将每一个node节点都当做为一个路由器(router)，各节点通过BGP(Border Gateway Protocol) 边界网关协议学习并在node节点生成路由规则，从而将不同node节点上的pod连接起来进行通信。

BGP是一个去中心化的协议，它通过自动学习和维护路由表实现网络的可用性，但是并不是所有的网络都支持BGP，另外为了跨网络实现更大规模的网络管理，calico 还支持IP-in-IP的叠加模型，简称IPIP，IPIP可以实现跨不同网段建立路由通信，但是会存在安全性问题，其在内核内置，可以通过Calico的配置文件设置是否启用IPIP，在公司内部如果k8s的node节点没有跨越网段建议关闭IPIP。

```
IPIP是一种将各Node的路由之间做一个tunnel，再把两个网络连接起来的模式。启用IPIP模式时，Calico将在各Node上创建一个名为"tunl0"的虚拟网络接口。
BGP模式则直接使用物理机作为虚拟路由路（vRouter），不再创建额外的tunnel。
```

calico 核心组件：

```
Felix：calico的agent，运行在每一台node节点上，其主要是维护路由规则、汇报当前节点状态以确保pod的夸主机通信。
BGP Client：每台node都运行，其主要负责监听node节点上由felix生成的路由信息，然后通过BGP协议广播至其他剩余的node节点，从而相互学习路由实现pod通信。
Route Reflector：集中式的路由反射器，calico v3.3开始支持，当Calico BGP客户端将路由从其FIB(Forward Information dataBase，转发信息库)通告到Route Reflector时，Route Reflector会将这些路由通告给部署集群中的其他节点，Route Reflector专门用于管理BGP网络路由规则，不会产生pod数据通信。

注：calico默认工作模式是BGP的node-to-node mesh，如果要使用Route Reflector需要进行相关配置。
https://docs.projectcalico.org/v3.4/usage/routereflector
https://docs.projectcalico.org/v3.2/usage/routereflector/calico-routereflector
```



2.4.1.1：部署过程：

https://docs.projectcalico.org/v3.4/getting-started/kubernetes/

具体过程略

2.4.1.2：验证当前路由表：

calico 2.x 版本默认使用 etcd v2 API

calico 3.x 版本默认使用 etcd v3 API

```
root@k8s-node2:~# calicoctl  node status
Calico process is running.

IPv4 BGP status
+---------------+-------------------+-------+----------+-------------+
| PEER ADDRESS  |     PEER TYPE     | STATE |  SINCE   |    INFO     |
+---------------+-------------------+-------+----------+-------------+
| 192.168.7.101 | node-to-node mesh | up    | 13:02:28 | Established |
| 192.168.7.102 | node-to-node mesh | up    | 13:22:49 | Established |
| 192.168.7.110 | node-to-node mesh | up    | 13:02:38 | Established |
+---------------+-------------------+-------+----------+-------------+

IPv6 BGP status
No IPv6 peers found.
```

2.4.1.3：查看pod路由走向：

验证开启IPIP和关闭IPIP的k8s集群验证

2.4.1.3.1：开启IPIP的通信状态：

```
开启IPIP：
root@k8s-master1:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.7.254   0.0.0.0         UG    0      0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
172.31.13.128   192.168.7.111   255.255.255.192 UG    0      0        0 tunl0
172.31.58.64    192.168.7.110   255.255.255.192 UG    0      0        0 tunl0
172.31.89.192   0.0.0.0         255.255.255.192 U     0      0        0 *
172.31.185.128  192.168.7.102   255.255.255.192 UG    0      0        0 tunl0
192.168.0.0     0.0.0.0         255.255.248.0   U     0      0        0 eth0

root@k8s-master1:~# kubectl  exec -it net-test1-68898c85b7-z7rgs    sh
/ # traceroute 172.31.58.83
traceroute to 172.31.58.83 (172.31.58.83), 30 hops max, 46 byte packets
 1  192.168.7.111 (192.168.7.111)  0.011 ms  0.010 ms  0.004 ms
 2  172.31.58.64 (172.31.58.64)  0.300 ms  0.505 ms  0.324 ms
 3  172.31.58.83 (172.31.58.83)  0.498 ms  0.619 ms  0.422 ms
/ #
```

2.1.1.3.2：关闭IPIP的通信状态：

需要清空环境，重新部署k8s集群

```
当前路由表：
root@k8s-node1:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.7.254   0.0.0.0         UG    0      0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
172.31.13.128   192.168.7.111   255.255.255.192 UG    0      0        0 eth0
172.31.58.64    0.0.0.0         255.255.255.255 UH    0      0        0 cali2fd25f96bb2
172.31.58.64    0.0.0.0         255.255.255.192 U     0      0        0 *
172.31.58.65    0.0.0.0         255.255.255.255 UH    0      0        0 calia6efdd98c77
172.31.185.128  192.168.7.102   255.255.255.255 UGH   0      0        0 eth0
192.168.0.0     0.0.0.0         255.255.248.0   U     0      0        0 eth0

root@k8s-master1:/etc/ansible#  kubectl run net-test1  --image=alpine --replicas=4
sleep 360000 #创建pod进行网络测试

root@k8s-master1:/etc/ansible# kubectl  exec -it net-test1-68898c85b7-qrh7b sh
/ # traceroute 172.31.58.65
traceroute to 172.31.58.65 (172.31.58.65), 30 hops max, 46 byte packets
```

```
1  192.168.7.111 (192.168.7.111)  0.009 ms  0.008 ms  0.004 ms
2  192.168.7.110 (192.168.7.110)  0.282 ms  0.344 ms  0.145 ms
3  172.31.58.65 (172.31.58.65)  0.481 ms  0.363 ms  0.197 ms
```

2.4.2：flannel:

官网：https://coreos.com/flannel/docs/latest/

文档：https://coreos.com/flannel/docs/latest/kubernetes.html

由CoreOS开源的针对k8s的网络服务，其目的为解决k8s集群中各主机上的pod相互通信的问题，其借助于etcd维护网络IP地址分配，并为每一个node服务器分配一个不同的IP地址段。

Flannel 网络模型 (后端)，Flannel目前有三种方式实现 UDP/VXLAN/host-gw：

```
UDP：早期版本的Flannel使用UDP封装完成报文的跨越主机转发，其安全性及性能略有不足。

VXLAN：Linux 内核在在2012年底的v3.7.0之后加入了VXLAN协议支持，因此新版本的Flannel也有UDP转换为VXLAN，VXLAN本质上是一种tunnel（隧道）协议，用来基于3层网络实现虚拟的2层网络，目前flannel 的网络模型已经是基于VXLAN的叠加(覆盖)网络，目前推荐使用vxlan作为其网络模型。

Host-gw：也就是Host GateWay，通过在node节点上创建到达各目标容器地址的路由表而完成报文的转发，因此这种方式要求各node节点本身必须处于同一个局域网(二层网络)中，因此不适用于网络变动频繁或比较大型的网络环境，但是其性能较好。
```

Flannel 组件的解释：

```
Cni0:网桥设备，每创建一个pod都会创建一对 veth pair，其中一端是pod中的eth0，另一端是Cni0网桥中的端口（网卡），Pod中从网卡eth0发出的流量都会发送到Cni0网桥设备的端口（网卡）上，Cni0 设备获得的ip地址是该节点分配到的网段的第一个地址。
Flannel.1: overlay网络的设备，用来进行vxlan报文的处理（封包和解包），不同node之间的pod数据流量都从overlay设备以隧道的形式发送到对端。
```

Flannel的系统文件及目录：

```
root@k8s-node2:~#  find / -name flannel
/run/flannel
/usr/bin/flannel
/var/lib/cni/flannel
```

2.4.2.1：flannel pod状态：

需要再部署k8s的时候使用Flannel作为网络插件。

```
root@k8s-master1:~# kubectl  get  pods  -n kube-system
NAME                           READY   STATUS    RESTARTS   AGE
kube-dns-75d7c6cd6f-w46pq      3/3     Running   0          13m
kube-flannel-ds-amd64-5xpbl    1/1     Running   1          22m
kube-flannel-ds-amd64-dddt2    1/1     Running   1          22m
kube-flannel-ds-amd64-jrpjn    1/1     Running   1          22m
kube-flannel-ds-amd64-qzk9f    1/1     Running   1          22m
```

2.4.2.2：创建测试pod：

```
root@k8s-master1:/etc/ansible/manifests/dns/kube-dns#  kubectl run net-test1  --
image=alpine --replicas=4  sleep 360000
kubectl run --generator=deployment/apps.v1 is DEPRECATED and will be removed in a
future version. Use kubectl run --generator=run-pod/v1 or kubectl create instead.
deployment.apps/net-test1 created
```

2.4.2.3：验证pod状态：

```
root@k8s-master1:~# kubectl  get  pod -o wide
NAME                         READY   STATUS    RESTARTS   AGE   IP           NODE
     NOMINATED NODE   READINESS GATES
net-test1-68898c85b7-2lgl6   1/1     Running   0          16m   172.31.3.9
192.168.7.110   <none>           <none>
net-test1-68898c85b7-7xshv   1/1     Running   0          12m   172.31.2.5
192.168.7.111   <none>           <none>
net-test1-68898c85b7-cb4s4   1/1     Running   0          16m   172.31.3.6
192.168.7.110   <none>           <none>
net-test1-68898c85b7-dctmx   1/1     Running   0          16m   172.31.3.5
192.168.7.110   <none>           <none>
```

2.4.2.4：当前node主机IP地址范围：

```
root@k8s-node2:~# cat /run/flannel/subnet.env
FLANNEL_NETWORK=172.31.0.0/16
FLANNEL_SUBNET=172.31.2.1/24
FLANNEL_MTU=1450
FLANNEL_IPMASQ=true
```

2.4.2.5：当前node主机cni信息：

```
root@k8s-node2:~# cat
/var/lib/cni/flannel/a77f70994d452ea66c06f0cec194d937dacac1b69c07d31b478f844f216e84c7
{"hairpinMode":true,"ipMasq":false,"ipam":{"routes":
[{"dst":"172.31.0.0/16"}],"subnet":"172.31.2.0/24","type":"host-
local"},"isDefaultGateway":true,"isGateway":true,"mtu":1
450,"name":"cbr0","type":"bridge"}
```

2.4.2.6：当前node主机路由：

```
root@k8s-node2:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.7.254   0.0.0.0         UG    0      0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
172.31.0.0      172.31.0.0      255.255.255.0   UG    0      0        0 flannel.1
172.31.1.0      172.31.1.0      255.255.255.0   UG    0      0        0 flannel.1
172.31.2.0      0.0.0.0         255.255.255.0   U     0      0        0 cni0
172.31.3.0      172.31.3.0      255.255.255.0   UG    0      0        0 flannel.1
192.168.0.0     0.0.0.0         255.255.248.0   U     0      0        0 eth0
```

2.4.2.7：验证flannel的VXLAN配置:

验证当前backend类型：

```
kind: ConfigMap
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{"cni-conf.json":"{\n  \"name\": \"cbr0\",\n
\"cniVersion\": \"0.3.1\",\n  \"plugins\": [\n    {\n      \"type\": \"flannel\",\n
\"delegate\": {\n        \"hairpinMode\": true,\n        \"isDefaultGateway\": true\n
   }\n    },\n    {\n      \"type\": \"portmap\",\n      \"capabilities\": {\n
\"portMappings\": true\n      }\n    }\n  ]\n}\n","net-conf.json":"{\n  \"Network\":
\"10.200.0.0/16\",\n  \"Backend\": {\n    \"Type\": \"vxlan\"\n
}\n}\n"},"kind":"ConfigMap","metadata":{"annotations":{},"labels":
{"app":"flannel","tier":"node"},"name":"kube-flannel-cfg","namespace":"kube-system"}}
```

vxlan使用UDP端口8472发送封装好的报文。

```
root@k8s-node1:~# netstat  -tuanlp | grep 8472
udp        0      0 0.0.0.0:8472            0.0.0.0:*                           -
```

测试网络通信

```
root@k8s-master1:~# kubectl  exec -it net-test1-68898c85b7-2lgl6 sh
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 7A:D3:D5:8B:C1:E8
```

```
          inet addr:172.31.3.9  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:29 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2102 (2.0 KiB)  TX bytes:42 (42.0 B)

/ # ping 172.31.2.5
PING 172.31.2.5 (172.31.2.5): 56 data bytes
64 bytes from 172.31.2.5: seq=0 ttl=62 time=0.768 ms
64 bytes from 172.31.2.5: seq=1 ttl=62 time=0.803 ms
64 bytes from 172.31.2.5: seq=2 ttl=62 time=1.010 ms
^C
--- 172.31.2.5 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.768/0.860/1.010 ms
```

2.4.2.8：VXLAN Directrouting：

Directrouting 为在同一个二层网络中的node节点启用直接路由机制，类似于host-gw模式。

2.4.2.8.1：修改flannel支持Directrouting：

需要让配置文件在node节点重新生效，

```
root@k8s-master1:/etc/ansible# vim roles/flannel/templates/kube-flannel.yaml.j2
  net-conf.json: |
    {
      "Network": "{{ CLUSTER_CIDR }}",
      "Backend": {
        "Type": "{{ FLANNEL_BACKEND }}",
        "Directrouting": true
      }
```

2.4.2.8.2：验证修改后的路由表：

```
#修改为Directrouting之前的路由表
root@k8s-node2:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.7.254   0.0.0.0         UG    0      0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
172.31.0.0      172.31.0.0      255.255.255.0   UG    0      0        0 flannel.1
172.31.1.0      172.31.1.0      255.255.255.0   UG    0      0        0 flannel.1
172.31.2.0      0.0.0.0         255.255.255.0   U     0      0        0 cni0
172.31.3.0      172.31.3.0      255.255.255.0   UG    0      0        0 flannel.1
192.168.0.0     0.0.0.0         255.255.248.0   U     0      0        0 eth0
```

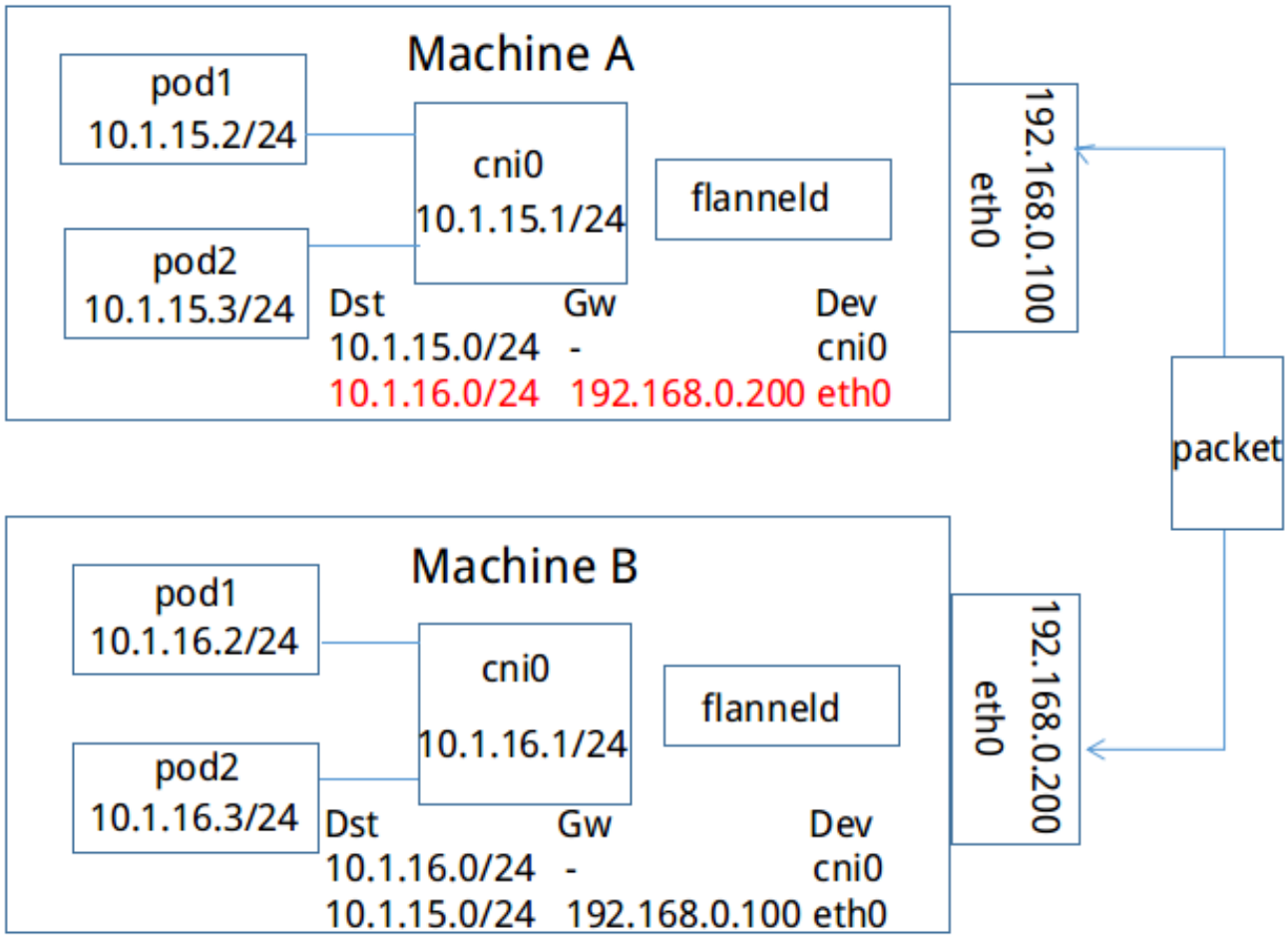
```
#修改为Directrouting之后的路由表
root@k8s-master1:/etc/ansible# ansible-playbook  06.network.yml
root@k8s-node1:~# reboot
root@k8s-node2:~# reboot
root@k8s-node2:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.7.254   0.0.0.0         UG    0      0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
172.31.0.0      192.168.7.102   255.255.255.0   UG    0      0        0 eth0
172.31.1.0      192.168.7.101   255.255.255.0   UG    0      0        0 eth0
172.31.2.0      0.0.0.0         255.255.255.0   U     0      0        0 cni0
172.31.3.0      192.168.7.110   255.255.255.0   UG    0      0        0 eth0
192.168.0.0     0.0.0.0         255.255.248.0   U     0      0        0 eth0
```

2.4.2.8.3：验证修改后的路由效果：

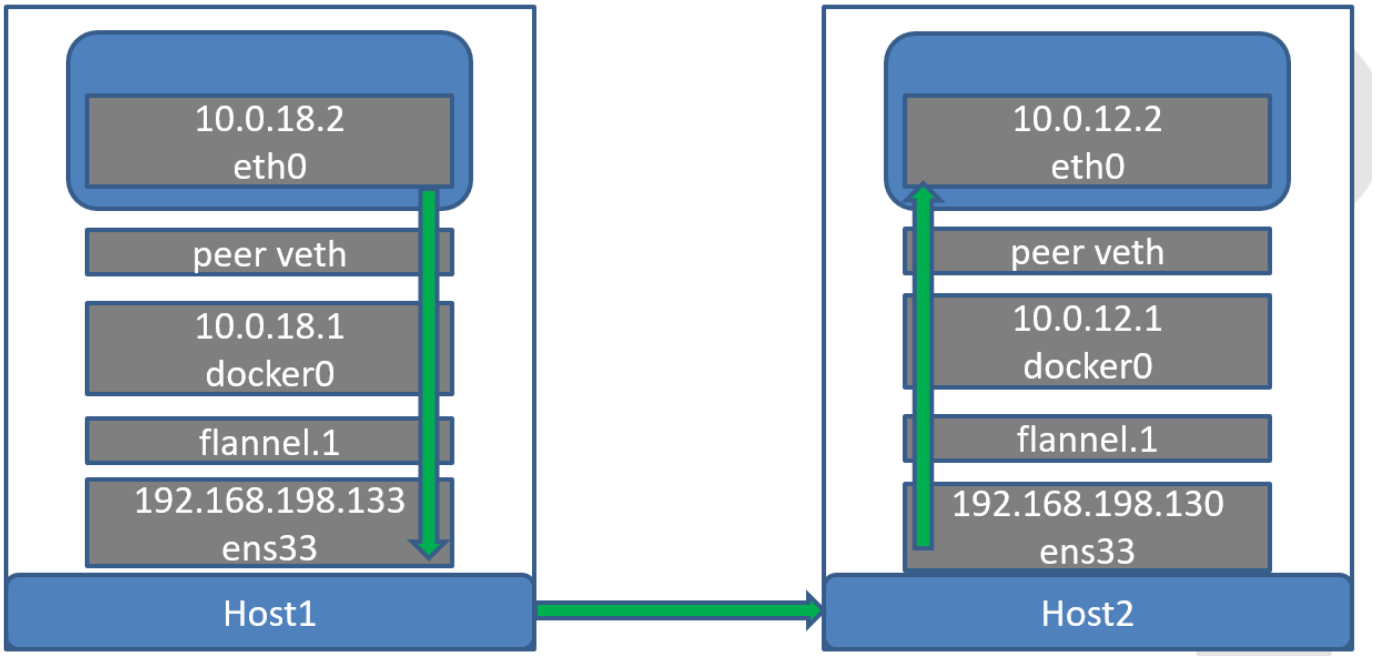
```
#改之前的路由效果
# ifconfig  eth0
eth0      Link encap:Ethernet  HWaddr BE:EE:B9:F7:62:47
          inet addr:172.31.3.4  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:48 errors:0 dropped:0 overruns:0 frame:0
          TX packets:35 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3700 (3.6 KiB)  TX bytes:2286 (2.2 KiB)

# traceroute  172.31.2.4
traceroute to 172.31.2.4 (172.31.2.4), 30 hops max, 46 byte packets
 1  172.31.3.1 (172.31.3.1)  0.049 ms  0.012 ms  0.007 ms
 2  172.31.2.0 (172.31.2.0)  0.520 ms  0.639 ms  4.061 ms
 3  172.31.2.4 (172.31.2.4)  3.544 ms  0.587 ms  0.464 ms


#改之后的路由效果
# ifconfig  eth0
eth0      Link encap:Ethernet  HWaddr 5E:83:2C:DD:66:D3
          inet addr:172.31.3.2  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:37 errors:0 dropped:0 overruns:0 frame:0
          TX packets:18 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2824 (2.7 KiB)  TX bytes:1170 (1.1 KiB)

# traceroute 172.31.2.2
traceroute to 172.31.2.2 (172.31.2.2), 30 hops max, 46 byte packets
 1  172.31.3.1 (172.31.3.1)  0.007 ms  0.018 ms  0.010 ms
 2  192.168.7.111 (192.168.7.111)  1.174 ms  0.244 ms  2.292 ms
 3  172.31.2.2 (172.31.2.2)  2.527 ms  0.549 ms  0.456 ms
```

2.4.2.9：flannel不同node上的pod的通信：

Flannel.1 是一个overlay网络的设备，用来进行 vxlan 报文的处理（封包和解包），不同node之间的pod数据流量都从overlay设备以隧道的形式发送到对端。




```
->: pod中产生数据，根据pod的路由信息，将数据发送到Cni0
->: Cni0 根据节点的路由表，将数据发送到隧道设备flannel.1
->: Flannel.1查看数据包的目的ip，从flanneld获得对端隧道设备的必要信息，封装数据包。
->: Flannel.1将数据包发送到对端设备,对端节点的网卡接收到数据包

->: 对端节点发现数据包为overlay数据包，解开外层封装，并发送到到本机flannel.1设备。
->: Flannel.1设备查看数据包，根据路由表匹配，将数据发送给Cni0设备。
->: Cni0匹配路由表，发送数据给网桥上对应的端口(pod)。
```



2.4.2.10：host-gw网络模型：

```
# 设置flannel 后端
FLANNEL_BACKEND: "host-gw"
#FLANNEL_BACKEND: "vxlan"
#DIRECT_ROUTING: true


/ # traceroute 10.10.4.6 #跨主机容器
traceroute to 10.10.4.6 (10.10.4.6), 30 hops max, 46 byte packets
 1  10.10.3.1 (10.10.3.1)  0.003 ms  0.004 ms  0.001 ms
 2  172.31.6.210 (172.31.6.210)  1.166 ms  0.495 ms  0.579 ms
 3  10.10.4.6 (10.10.4.6)  0.307 ms  0.335 ms  0.322 ms
/ #
/ # traceroute 172.31.6.207
traceroute to 172.31.6.207 (172.31.6.207), 30 hops max, 46 byte packets
 1  10.10.3.1 (10.10.3.1)  0.003 ms  0.004 ms  0.001 ms
 2  172.31.6.207 (172.31.6.207)  1.290 ms  0.776 ms  2.041 ms
```

2.4.2.11：UDP网络模型：

UDP是最早的实现方式，但是由于其性能原因，现已经被废弃。

2.4.2.11.1：修改backend为UDP：

重新执行网络并重启master与node服务器

```
# 设置flannel 后端
#FLANNEL_BACKEND: "host-gw"
#FLANNEL_BACKEND: "vxlan"
FLANNEL_BACKEND: "udp"
```

2.4.2.11.2：当前路由状态：

```
# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.7.254   0.0.0.0         UG    0      0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
172.31.0.0      0.0.0.0         255.255.0.0     U     0      0        0 flannel0
172.31.2.0      0.0.0.0         255.255.255.0   U     0      0        0 cni0
192.168.0.0     0.0.0.0         255.255.248.0   U     0      0        0 eth0
```

2.4.2.11.3：验证UDP模式下pod夸主机通信状态：

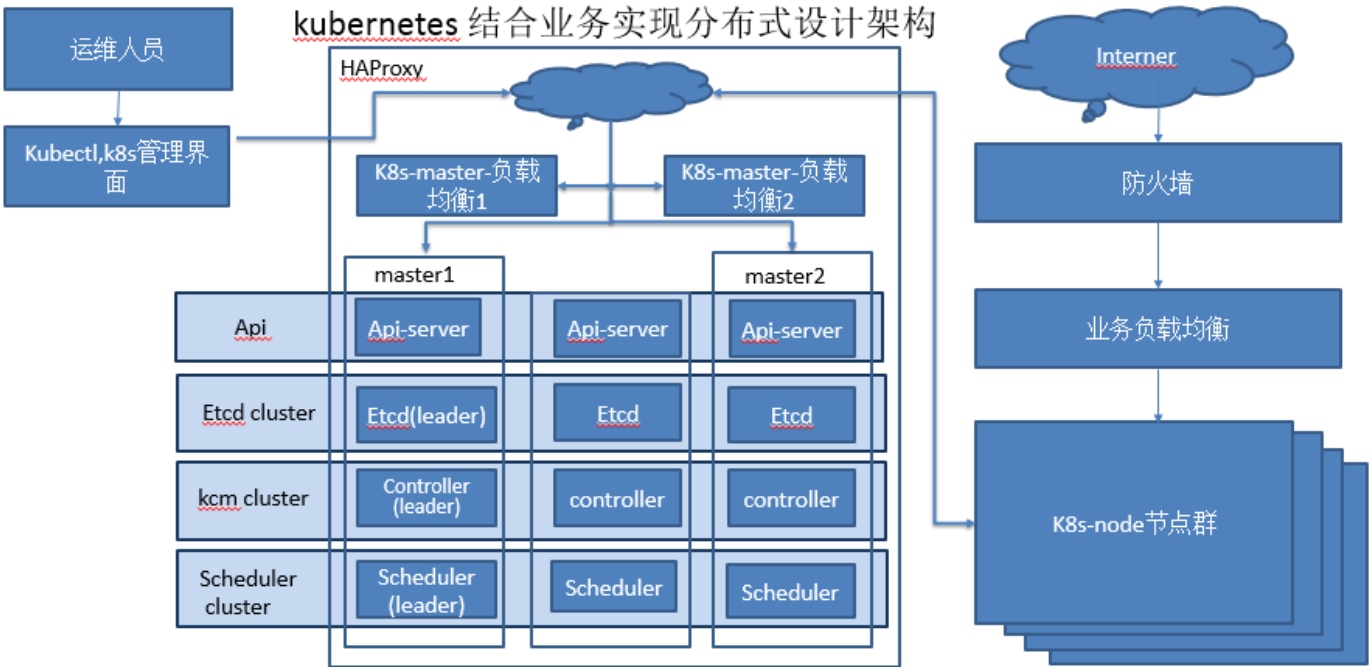
```
# kubectl  exec -it busybox sh
/ # traceroute 172.31.2.21
traceroute to 172.31.2.21 (172.31.2.21), 30 hops max, 46 byte packets
 1  172.31.3.1 (172.31.3.1)  0.110 ms  0.008 ms  0.018 ms
 2  *  *  *
 3  *  *  *
 4  172.31.2.0 (172.31.2.0)  2.801 ms  0.672 ms  0.915 ms
 5  172.31.2.21 (172.31.2.21)  1.006 ms  0.736 ms  1.190 ms
```

2.5：术语概念：

2.5.1：k8s中的kind类型：

三：运行web服务

当前k8s 运行规划图：

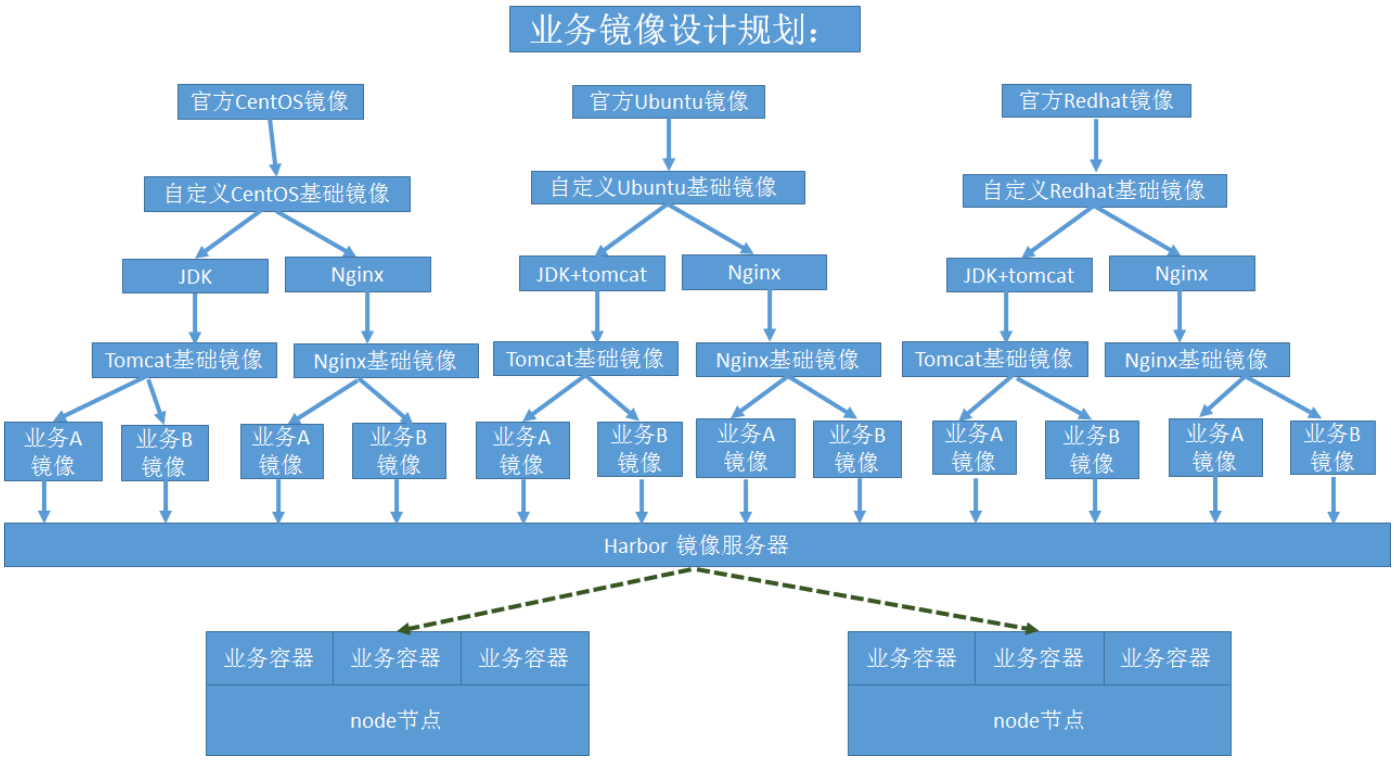


3.1：运行nginx：

将nginx运行在k8s中并可以从外部访问到nginx的web页面。

3.1.1：Nginx镜像制作规划：

基于基础的centos/ubuntu/alpine镜像，制作公司内部基础镜像-Nginx基础镜像--Nginx业务镜像：



3.1.1.1：Centos基础镜像制作：

3.1.1.1.1

```
# cd /opt/k8s-data/ #dockerfile与yaml目录
# cd dockerfile/system/centos/ #系统镜像目录
root@k8s-master1:/opt/k8s-data/dockerfile/system/centos# tree
.
├── build-command.sh
├── Dockerfile
└── filebeat-6.8.1-x86_64.rpm

0 directories, 3 files
root@k8s-master1:/opt/k8s-data/dockerfile/system/centos#
```

3.1.1.1.2：Dockerfile文件内容:

安装基础命令并配置语言环境为中文

```
root@k8s-master1:/opt/k8s-data/dockerfile/system/centos# cat Dockerfile
#制作基础镜像,centos:v7.5是基于官方centos 7.5镜像更改tag并上传到Harbor服务器的最原始官方镜像
FROM harbor.magedu.net/baseimages/centos:7.6.1810

MAINTAINER zhangshijie "zhangshijie@magedu.ent"

ADD  filebeat-6.8.1-x86_64.rpm /tmp/
RUN  yum install -y epel-release /tmp/filebeat-6.5.4-x86_64.rpm && rm -rf
/tmp/filebeat-6.5.4-x86_64.rpm
RUN  yum install -y vim wget tree  pcre pcre-devel gcc gcc-c++  zlib zlib-devel openssl
openssl-devel net-tools iotop unzip zip iproute ntpdate nfs-utils tcp
dump telnet traceroute
RUN rm -rf /etc/localtime && ln -snf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
```

3.1.1.1.3：build-command脚本：

基于脚本实现镜像自动build及上传到harbor功能

```
root@k8s-master1:/opt/k8s-data/dockerfile/system/centos# cat build-command.sh
#!/bin/bash
docker build -t harbor.magedu.net/baseimages/centos-base:v7.6 .
sleep 1
docker push harbor.magedu.net/baseimages/centos-base:v7.6
```

3.1.1.1.4：执行构建centos 基础镜像：

构建完成后自动上传至本地harbor服务器

```
root@k8s-master1:/opt/k8s-data/dockerfile/system/centos# bash build-command.sh
Sending build context to Docker daemon  11.91MB
Step 1/6 : FROM harbor.magedu.net/baseimages/centos:7.6.1810
 ---> 9f38484d220f
Step 2/6 : MAINTAINER zhangshijie "zhangshijie@magedu.ent"
 ---> Using cache
 ---> 64e3caf66048
Step 3/6 : ADD  filebeat-6.8.1-x86_64.rpm /tmp/
 ---> Using cache
 ---> 76e9f39f514f
Step 4/6 : RUN  yum install -y epel-release /tmp/filebeat-6.5.4-x86_64.rpm && rm -rf /tmp/filebeat-6.5.4-x86_64.rpm
 ---> Using cache
 ---> 1643f0fbaeae
Step 5/6 : RUN  yum install -y vim wget tree  pcre pcre-devel gcc gcc-c++  zlib zlib-devel openssl openssl-devel net-tools
s-utils tcpdump telnet traceroute
 ---> Using cache
 ---> 2dbd609100d9
Step 6/6 : RUN rm -rf /etc/localtime && ln -snf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
 ---> Using cache
 ---> 6573367a89a5
Successfully built 6573367a89a5
Successfully tagged harbor.magedu.net/baseimages/centos-base:v7.6
The push refers to repository [harbor.magedu.net/baseimages/centos-base]
d0809305f5d8: Pushed
5bc1fe4946dc: Pushing [==================================>              ]    251MB/350.1MB
d1dc86056301: Pushed
93e9804d983a: Pushed
d69483a6face: Pushed
```

3.1.1.2：Nginx 基础镜像制作：

制作一个通用的Ningx镜像

3.1.1.3.1：镜像文件列表：

```
# pwd
/opt/k8s-data/dockerfile/pub-images/nginx-base

# tree
.
├── build-command.sh
├── Dockerfile
└── nginx-1.14.2.tar.gz

0 directories, 3 files
```

3.1.1.3.2：Dockerfile文件内容：

```
# cat Dockerfile
#Nginx Base Image
FROM harbor.magedu.net/baseimages/centos-base:v7.6

MAINTAINER  zhangshijie@magedu.net

RUN yum install -y vim wget tree  lrzsz gcc gcc-c++ automake pcre pcre-devel zlib zlib-
devel openssl openssl-devel iproute net-tools iotop
ADD nginx-1.14.2.tar.gz /usr/local/src/
RUN cd /usr/local/src/nginx-1.14.2 && ./configure  && make && make install && ln -sv
/usr/local/nginx/sbin/nginx /usr/sbin/nginx && useradd nginx -u 2001
```

3.1.1.1.3：build-command脚本：

```
# cat build-command.sh
#!/bin/bash
docker build -t harbor.magedu.net/pub-images/nginx-base:v1.14.2  .
sleep 1
docker push  harbor.magedu.net/pub-images/nginx-base:v1.14.2
```

3.1.1.1.4：执行构建Nginx基础镜像：

```
Step 3/5 : RUN yum install -y vim wget tree  lrzsz gcc gcc-c++ automake pcre pcre-devel zlib zlib-devel openssl op
 ---> Using cache
 ---> 14cdfcd4b8d5
Step 4/5 : ADD nginx-1.14.2.tar.gz /usr/local/src/
 ---> Using cache
 ---> 1ef7af4521fe
Step 5/5 : RUN cd /usr/local/src/nginx-1.14.2 && ./configure  && make && make install && ln -sv  /usr/local/nginx/
 -u 2001
 ---> Using cache
 ---> 494ab6676ad1
Successfully built 494ab6676ad1
Successfully tagged harbor.magedu.net/pub-images/nginx-base:v1.14.2
The push refers to repository [harbor.magedu.net/pub-images/nginx-base]
3713c9956100: Pushing [==================================================>]  16.64MB
cf47e5488136: Pushing [============================>                      ]  3.715MB/6.07MB
9dd71f271b1c: Pushing [===>                                               ]  11.55MB/164.6MB
d0809305f5d8: Pushed
5bc1fe4946dc: Pushing [=>                                                 ]  12.07MB/350.1MB
d1dc86056301: Pushing [>                                                  ]  3.886MB/217.7MB
93e9804d983a: Waiting
d69483a6face: Waiting
```

3.1.1.3：Nginx业务镜像制作：

基于Nginx基础镜像，制作N个不同服务的Nginx业务镜像：

3.1.1.3.1：镜像文件列表：

```
# pwd
/opt/k8s-data/dockerfile/linux36/nginx
# tree
.
├── build-command.sh
├── Dockerfile
├── index.html
├── nginx.conf
└── webapp
    └── index.html

1 directory, 5 files
```

3.1.1.3.2：Dockerfile文件内容：

```
# cat  Dockerfile
#Nginx Base Image
FROM harbor.magedu.net/pub-images/nginx-base:v1.14.2

ADD nginx.conf /usr/local/nginx/conf/nginx.conf
ADD webapp/*  /usr/local/nginx/html/webapp/
ADD index.html  /usr/local/nginx/html/index.html
#RUN mkdir /usr/local/nginx/html/webapp/about /usr/local/nginx/html/webapp/images

EXPOSE 80 443

CMD ["nginx"]
```

3.1.1.3.3：build-command脚本：

```
# cat  build-command.sh
#!/bin/bash
docker build -t harbor.magedu.net/linux36/nginx-web1:v1 .
sleep 1
docker push harbor.magedu.net/linux36/nginx-web1:v1
```

3.1.1.3.4：测试页面文件内容：

```
# cat webapp/index.html
Nginx webapp test page
```

3.1.1.3.5：Nginx配置文件：

```
# vim nginx.conf

daemon off; #关闭后台运行
```

```
#upstream  tomcat_webserver {
#        server  linux35-tomcat-app1-spec.linux35.svc.linux35.local:80;
#        server  linux35-tomcat-app2-spec.linux35.svc.linux35.local:80;
#}

    server {
..................
#        location /myapp {
#             proxy_pass  http://tomcat_webserver;
#             proxy_set_header   Host    $host;
#            proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
#             proxy_set_header X-Real-IP $remote_addr;
#        }
..................
    }
```

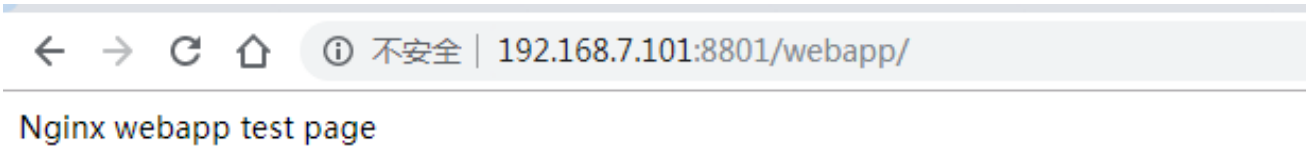
3.1.1.3.6：执行构建Nginx 业务镜像：

```
root@k8s-master1:/opt/k8s-data/dockerfile/linux36/nginx# bash build-command.sh
Sending build context to Docker daemon  9.728kB
Step 1/6 : FROM harbor.magedu.net/pub-images/nginx-base:v1.14.2
 ---> 494ab6676ad1
Step 2/6 : ADD nginx.conf /usr/local/nginx/conf/nginx.conf
 ---> 17c3e1e4a70d
Step 3/6 : ADD webapp/*  /usr/local/nginx/html/webapp/
 ---> d3ac25c4b3fb
Step 4/6 : ADD index.html  /usr/local/nginx/html/index.html
 ---> 9e33a48ebcdb
Step 5/6 : EXPOSE 80 443
 ---> Running in 6d1e111b2531
Removing intermediate container 6d1e111b2531
 ---> 126ad60f7731
Step 6/6 : CMD ["nginx"]
 ---> Running in d6b61a4a75ef
Removing intermediate container d6b61a4a75ef
 ---> 4e3d5ad9d0d6
Successfully built 4e3d5ad9d0d6
Successfully tagged harbor.magedu.net/linux36/nginx-web1:v1
The push refers to repository [harbor.magedu.net/linux36/nginx-web1]
c35373885901: Pushed
a8926b276091: Pushed
cef7d46f1d57: Pushed
3713c9956100: Mounted from pub-images/nginx-base
cf47e5488136: Mounted from pub-images/nginx-base
9dd71f271b1c: Mounted from pub-images/nginx-base
d0809305f5d8: Mounted from pub-images/nginx-base
5bc1fe4946dc: Mounted from pub-images/nginx-base
d1dc86056301: Mounted from pub-images/nginx-base
93e9804d983a: Mounted from pub-images/nginx-base
d69483a6face: Mounted from pub-images/nginx-base
v1: digest: sha256:6a7e9de9eb8c3c689be90c896aa321a0dceeb5d60c442ce893e1fceddc6cf893 size: 2629
root@k8s-master1:/opt/k8s-data/dockerfile/linux36/nginx#
```

3.1.1.3.7：测试nginx业务镜像可以启动为容器:

```
# docker run -it --rm -p 8801:80 harbor.magedu.net/linux36/nginx-web1:v1
```

3.1.1.3.8：访问测试Nginx业务web页面：



3.1.2：yaml文件及语法基础：

需要提前创建好yaml文件，并创建好好pod运行所需要的namespace、yaml文件等资源

3.1.2.1：创建业务namespace yaml文件：

```
# pwd
/opt/k8s-data/yaml/
# mkdir namespaces
# cd namespaces

# cat linux36.yaml
apiVersion: v1 #API版本
kind: Namespace #类型为namespac
metadata:  #定义元数据
  name: linux36 #namespace名称
```

3.1.2.2：创建并验证namespace：

```
# kubectl  apply -f linux36.yaml
namespace/linux36 created

# kubectl  get namespaces
NAME          STATUS   AGE
default       Active   21d
kube-public   Active   21d
kube-system   Active   21d
linux36       Active   45s
```

3.1.2.2：yaml与json：

yaml和json对比，在线yaml与json编辑器：http://www.bejson.com/validators/yaml_editor/

3.1.2.2.1：json格式：

```
{ "人员名单'"：
   { "张三"：{ "年龄"：18，"职业"："Linux运维工程师"，"爱好"：[ "看书"，"学习"，"加班" ] }，#
     "李四"：{ "年龄"：20，"职业"："Java开发工程师"，"爱好"：[ "开源技术"，"微服务"，"分布式存
储"] } } }

#json特点：
json 不能注释
json 可读性较差
json 语法很严格
比较适用于API 返回值，也可用于配置文件
```

3.1.2.2.2：yaml格式：

```
人员名单：
  张三：
    年龄：18 #
    职业：Linux运维工程师
    爱好：
      - 看书
      - 学习
      - 加班

  李四：
    年龄：20
    职业：Java开发工程师 # 这是职业
    爱好：
      - 开源技术
      - 微服务
      - 分布式存储

大小写敏感
使用缩进表示层级关系
缩进时不允许使用Tal键，只允许使用空格
缩进的空格数目不重要，只要相同层级的元素左侧对齐即可
使用"#" 表示注释，从这个字符一直到行尾，都会被解析器忽略
比json更适用于配置文件
```

3.1.2.2.3：yaml文件主要特性：

k8s中的yaml文件以及其他场景的yaml文件，大部分都是以下类型：

```
上下级关系
列表
键值对(也称为maps，即key:value 格式的键值对数据)
```

3.1.2.2.4：Nginx 业务yaml文件详解：

```
# pwd
/opt/k8s-data/yaml/linux36
# mkdir nginx tomcat-app1  tomcat-app2
# cd nginx/

# pwd
/opt/k8s-data/yaml/linux36/nginx
# cat nginx.yaml
kind: Deployment  #类型，是deployment控制器，kubectl explain  Deployment
apiVersion: extensions/v1beta1  #API版本，# kubectl explain  Deployment.apiVersion
metadata: #pod的元数据信息，kubectl explain  Deployment.metadata
  labels: #自定义pod的标签，# kubectl explain  Deployment.metadata.labels
    app: linux36-nginx-deployment-label #标签名称为app值为linux36-nginx-deployment-label，
后面会用到此标签
  name: linux36-nginx-deployment #pod的名称
  namespace: linux36 #pod的namespace，默认是defaule
spec: #定义deployment中容器的详细信息，kubectl explain  Deployment.spec
  replicas: 1 #创建出的pod的副本数，即多少个pod，默认值为1
  selector: #定义标签选择器
    matchLabels: #定义匹配的标签，必须要设置
      app: linux36-nginx-selector #匹配的目标标签，
  template: #定义模板，必须定义，模板是起到描述要创建的pod的作用
    metadata: #定义模板元数据
      labels: #定义模板label，Deployment.spec.template.metadata.labels
        app: linux36-nginx-selector #定义标签，等于Deployment.spec.selector.matchLabels
    spec: #定义pod信息
      containers:#定义pod中容器列表，可以多个至少一个，pod不能动态增减容器
      - name: linux36-nginx-container #容器名称
        image: harbor.magedu.net/linux36/nginx-web1:v1 #镜像地址
        #command: ["/apps/tomcat/bin/run_tomcat.sh"] #容器启动执行的命令或脚本
        #imagePullPolicy: IfNotPresent
        imagePullPolicy: Always #拉取镜像策略
        ports: #定义容器端口列表
        - containerPort: 80 #定义一个端口
          protocol: TCP #端口协议
          name: http #端口名称
        - containerPort: 443 #定义一个端口
          protocol: TCP #端口协议
          name: https #端口名称
        env: #配置环境变量
        - name: "password" #变量名称。必须要用引号引起来
          value: "123456" #当前变量的值
        - name: "age" #另一个变量名称
          value: "18" #另一个变量的值
        resources: #对资源的请求设置和限制设置
          limits: #资源限制设置，上限
            cpu: 500m  #cpu的限制，单位为core数，可以写0.5或者500m等CPU压缩值
```

```
            memory: 2Gi #内存限制，单位可以为Mib/Gib，将用于docker run --memory参数
          requests: #资源请求的设置
            cpu: 200m #cpu请求数，容器启动的初始可用数量,可以写0.5或者500m等CPU压缩值
            memory: 512Mi #内存请求大小，容器启动的初始可用数量，用于调度pod时候使用


---
kind: Service #类型为service
apiVersion: v1 #service API版本， service.apiVersion
metadata: #定义service元数据，service.metadata
  labels: #自定义标签，service.metadata.labels
    app: linux36-nginx #定义service标签的内容
  name: linux36-nginx-spec #定义service的名称，此名称会被DNS解析
  namespace: linux36 #该service隶属于的namespaces名称，即把service创建到哪个namespace里面
spec: #定义service的详细信息，service.spec
  type: NodePort #service的类型，定义服务的访问方式，默认为ClusterIP， service.spec.type
  ports: #定义访问端口， service.spec.ports
  - name: http #定义一个端口名称
    port: 80 #service 80端口
    protocol: TCP #协议类型
    targetPort: 80 #目标pod的端口
    nodePort: 30001 #node节点暴露的端口
  - name: https #SSL 端口
    port: 443 #service 443端口
    protocol: TCP #端口协议
    targetPort: 443 #目标pod端口
    nodePort: 30043 #node节点暴露的SSL端口
  selector: #service的标签选择器，定义要访问的目标pod
    app: linux36-nginx #将流量路到选择的pod上，须等于Deployment.spec.selector.matchLabels
```

3.1.3：k8s中创建Nginx pod：

创建Nginx pod 并测试通过node port访问

3.1.3.1：Nginx yaml文件：

```
root@k8s-master1:/opt/k8s-data/yaml/linux36/nginx# cat nginx.yaml
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  labels:
    app: linux36-nginx-deployment-label
  name: linux36-nginx-deployment
  namespace: linux36
spec:
  replicas: 1
  selector:
    matchLabels:
      app: linux36-nginx-selector
```

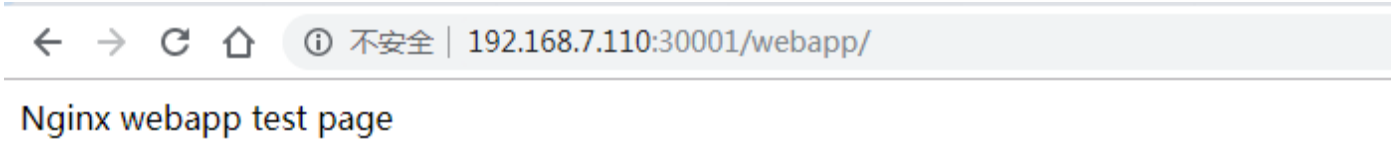
```
  template:
    metadata:
      labels:
        app: linux36-nginx-selector
    spec:
      containers:
      - name: linux36-nginx-container
        image: harbor.magedu.net/linux36/nginx-web1:v1
        #command: ["/apps/tomcat/bin/run_tomcat.sh"]
        #imagePullPolicy: IfNotPresent
        imagePullPolicy: Always
        ports:
        - containerPort: 80
          protocol: TCP
          name: http
        - containerPort: 443
          protocol: TCP
          name: https
        env:
        - name: "password"
          value: "123456"
        - name: "age"
          value: "18"
        resources:
          limits:
            cpu: 2
            memory: 2Gi
          requests:
            cpu: 500m
            memory: 1Gi


---
kind: Service
apiVersion: v1
metadata:
  labels:
    app: linux36-nginx-service-label
  name: linux36-nginx-service
  namespace: linux36
spec:
  type: NodePort
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
    nodePort: 30002
  - name: https
```

```
    port: 443
    protocol: TCP
    targetPort: 443
    nodePort: 30443
  selector:
    app: linux36-nginx-selector
```

3.1.3.2：创建Nginx pod：

```
# kubectl  apply -f nginx.yaml
deployment.extensions/linux36-nginx-deployment created
service/linux36-nginx-spec created
```

3.1.3.3：测试访问Nginx web界面：



Nginx webapp test page

3.2：运行tomcat：

基于基础的centos镜像，制作公司内部基础镜像--jdk镜像--tomcat基础镜像--tomcat业务镜像：

3.2.1：JDK基础镜像制作：

3.2.1.1：JDK基础镜像文件列表：

```
# pwd
/opt/k8s-data/dockerfile/pub-images #共用镜像目录
# cd jdk-1.8.212/
# tree
.
├── build-command.sh
├── Dockerfile
├── jdk-8u212-linux-x64.tar.gz
└── profile

0 directories, 4 files
```

3.2.1.2：Dockerfile文件内容：

```
# cat Dockerfile
#JDK Base Image
FROM harbor.magedu.net/baseimages/centos-base:v7.6
```

```
MAINTAINER zhangshijie "zhangshijie@magedu.net"


ADD jdk-8u212-linux-x64.tar.gz /usr/local/src/
RUN ln -sv /usr/local/src/jdk1.8.0_212 /usr/local/jdk  && groupadd tomcat -g 2018 &&
useradd  tomcat -u 2018 -g 2018
ADD profile /etc/profile


ENV JAVA_HOME /usr/local/jdk
ENV JRE_HOME $JAVA_HOME/jre
ENV CLASSPATH $JAVA_HOME/lib/:$JRE_HOME/lib/
ENV PATH $PATH:$JAVA_HOME/bin
```

3.2.1.3：build-command脚本：

```
# cat build-command.sh
#!/bin/bash
docker build -t harbor.magedu.net/pub-images/jdk-base:v8.212  .
sleep 1
docker push  harbor.magedu.net/pub-images/jdk-base:v8.212
```

3.2.1.4：执行构建JDK基础镜像：

```
root@k8s-master1:/opt/k8s-data/dockerfile/pub-images/jdk-1.8.162# bash build-command.sh
Sending build context to Docker daemon    195MB
Step 1/9 : FROM harbor.magedu.net/baseimages/centos-base:v7.6
 ---> 6573367a89a5
Step 2/9 : MAINTAINER zhangshijie "zhangshijie@magedu.net"
 ---> Running in 6bd84177e5ef
Removing intermediate container 6bd84177e5ef
 ---> 873ba8e6ae7c
Step 3/9 : ADD jdk-8u212-linux-x64.tar.gz /usr/local/src/
 ---> 8d7a335d63e9
Step 4/9 : RUN ln -sv /usr/local/src/jdk1.8.0_212 /usr/local/jdk  && groupadd tomcat -g 2018 && useradd  tomcat -u 2018 -g 2018
 ---> Running in 5dbe426a4c2a
'/usr/local/jdk' -> '/usr/local/src/jdk1.8.0_212'
Removing intermediate container 5dbe426a4c2a
 ---> 6ccad97c53fb
Step 5/9 : ADD profile /etc/profile
 ---> 3dde5b74977f
Step 6/9 : ENV JAVA_HOME /usr/local/jdk
 ---> Running in 969818cb4122
Removing intermediate container 969818cb4122
```

3.2.1.5：验证JDK镜像启动为容器后的java环境：

```
# docker  run -it --rm  harbor.magedu.net/pub-images/jdk-base:v8.212 bash
[root@73bfac24b94e /]# java -version
java version "1.8.0_212"
Java(TM) SE Runtime Environment (build 1.8.0_212-b10)
Java HotSpot(TM) 64-Bit Server VM (build 25.212-b10, mixed mode)
```

3.2.2：tomcat基础镜像制作：

3.2.2.1：基础镜像文件列表：

```
# pwd
/opt/k8s-data/dockerfile/pub-images/tomcat-base
# tree
.
├── apache-tomcat-8.5.43.tar.gz
├── build-command.sh
└── Dockerfile

0 directories, 3 files
```

3.2.2.2：Dockerfile文件内容：

```
# cat Dockerfile
#JDK Base Image
FROM harbor.magedu.net/pub-images/jdk-base:v8.212

MAINTAINER zhangshijie "zhangshijie@magedu.net"

RUN mkdir /apps /data/tomcat/webapps /data/tomcat/logs -pv
ADD apache-tomcat-8.5.43.tar.gz  /apps
RUN ln -sv /apps/apache-tomcat-8.5.43 /apps/tomcat && chown -R tomcat.tomcat /apps
/data -R
#ADD  filebeat-6.4.2-x86_64.rpm /tmp/
#RUN  yum install -y /tmp/filebeat-6.4.2-x86_64.rpm && rm -rf /tmp/filebeat-6.4.2-
x86_64.rpm
```

3.2.3.3：build-command脚本：

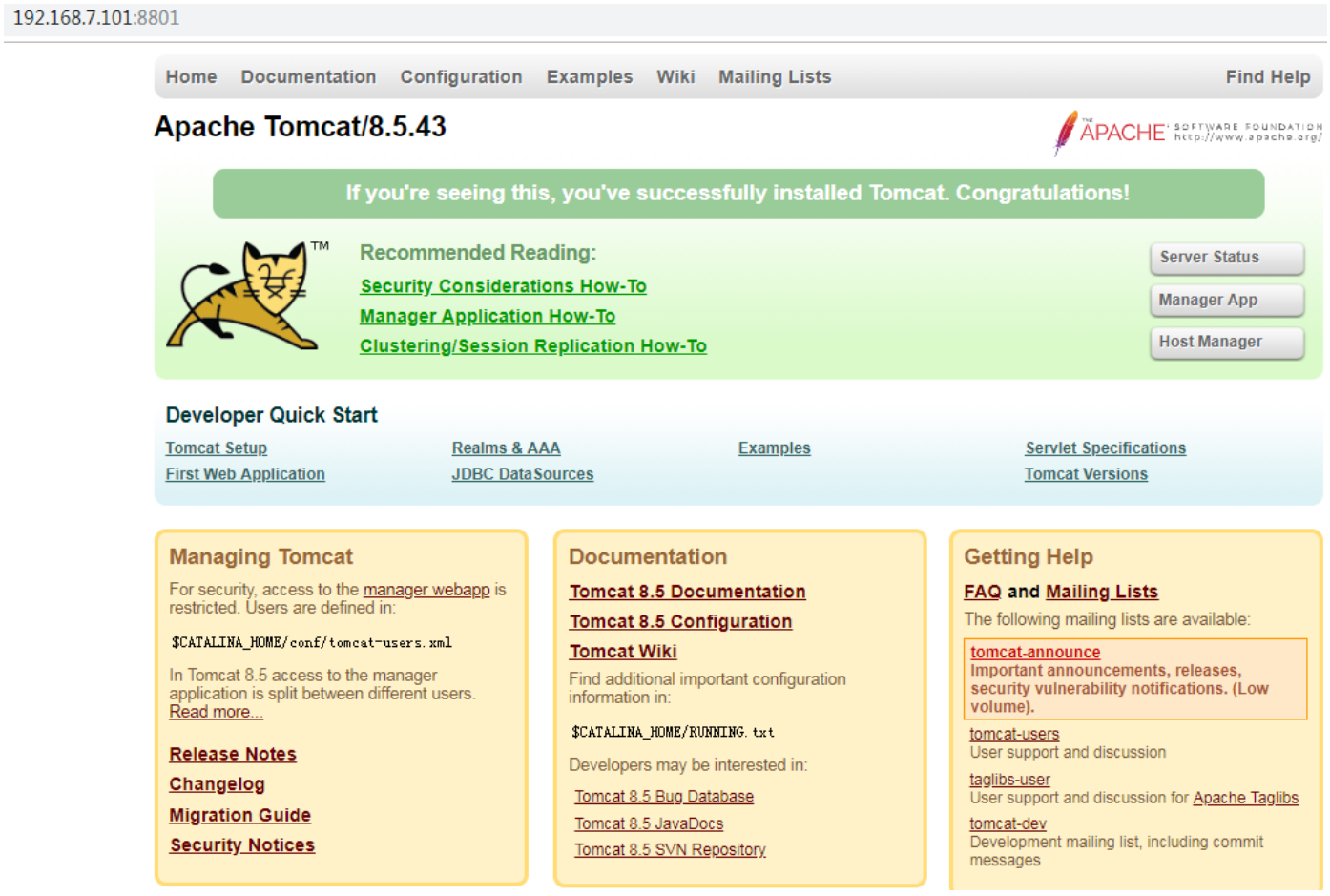
```
# cat build-command.sh
#!/bin/bash
docker build -t harbor.magedu.net/pub-images/tomcat-base:v8.5.43  .
sleep 3
docker push  harbor.magedu.net/pub-images/tomcat-base:v8.5.43
```

3.2.3.4：构建tomcat基础镜像：

```
root@k8s-master1:/opt/k8s-data/dockerfile/pub-images/tomcat-base# bash build-command.sh
Sending build context to Docker daemon  9.721MB
Step 1/5 : FROM harbor.magedu.net/pub-images/jdk-base:v8.212
 ---> da1b3eff0592
Step 2/5 : MAINTAINER zhangshijie "zhangshijie@magedu.net"
 ---> Using cache
 ---> ce51c8949ccb
Step 3/5 : RUN mkdir /apps /data/tomcat/webapps /data/tomcat/logs -pv
 ---> Using cache
 ---> db86698c4946
Step 4/5 : ADD apache-tomcat-8.5.43.tar.gz  /apps
 ---> Using cache
 ---> 2f5558704c8d
Step 5/5 : RUN ln -sv /apps/apache-tomcat-8.5.43 /apps/tomcat && chown -R tomcat.tomcat /apps /data -R
 ---> Using cache
 ---> 5464298f324e
Successfully built 5464298f324e
Successfully tagged harbor.magedu.net/pub-images/tomcat-base:v8.5.43
```

3.2.3.5：测试访问tomcat基础镜像启动为容器：

```
# docker run -it --rm -p 8801:8080 harbor.magedu.net/pub-images/tomcat-base:v8.5.43
[root@7a5c2e1b67b3 /]# /apps/tomcat/bin/catalina.sh  start
Using CATALINA_BASE:   /apps/tomcat
Using CATALINA_HOME:   /apps/tomcat
Using CATALINA_TMPDIR: /apps/tomcat/temp
Using JRE_HOME:        /usr/local/jdk/jre
Using CLASSPATH:       /apps/tomcat/bin/bootstrap.jar:/apps/tomcat/bin/tomcat-juli.jar
Tomcat started.
```



3.2.3：tomcat业务镜像app1制作：

后期按此步骤制作app2、appN镜像

3.2.3.1：cat业务镜像文件列表：

```
# pwd
/opt/k8s-data/dockerfile/linux36/tomcat-app1

# tree
.
├── app1.tar.gz
├── build-command.sh
├── catalina.sh
├── Dockerfile
├── filebeat.yml
├── myapp
│   └── index.html
├── run_tomcat.sh
└── server.xml

1 directory, 8 files
```

3.2.3.2：Dockerfile文件内容：

```
# cat Dockerfile
#tomcat web1
FROM harbor.magedu.net/pub-images/tomcat-base:v8.5.43

ADD catalina.sh /apps/tomcat/bin/catalina.sh
ADD server.xml /apps/tomcat/conf/server.xml
#ADD myapp/* /data/tomcat/webapps/myapp/
ADD app1.tar.gz /data/tomcat/webapps/myapp/
ADD run_tomcat.sh /apps/tomcat/bin/run_tomcat.sh
#ADD filebeat.yml /etc/filebeat/filebeat.yml
RUN chown  -R tomcat.tomcat /data/ /apps/

EXPOSE 8080 8443

CMD ["/apps/tomcat/bin/run_tomcat.sh"]
```

3.2.3.3：build-command脚本：

```
# cat  build-command.sh
#!/bin/bash
TAG=$1
docker build -t  harbor.magedu.net/linux36/tomcat-app1:${TAG} .
sleep 3
docker push  harbor.magedu.net/linux36/tomcat-app1:${TAG}
```

3.2.3.4：执行构建tomcat业务镜像：

```
# bash build-command.sh  2019-08-02_11_02_30
```

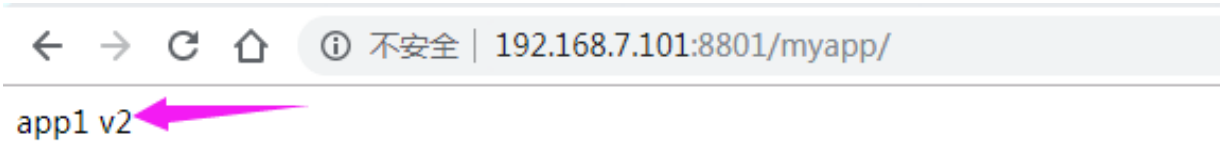
```
Step 5/8 : ADD run_tomcat.sh /apps/tomcat/bin/run_tomcat.sh
 ---> de8e527b3ce8
Step 6/8 : RUN chown  -R tomcat.tomcat /data/ /apps/
 ---> Running in fd1bf360ffcd
Removing intermediate container fd1bf360ffcd
 ---> 45caf98d5444
Step 7/8 : EXPOSE 8080 8443
 ---> Running in 35a3eaccaf63
Removing intermediate container 35a3eaccaf63
 ---> c5451dc1aad2
Step 8/8 : CMD ["/apps/tomcat/bin/run_tomcat.sh"]
 ---> Running in 841e6280ae08
Removing intermediate container 841e6280ae08
 ---> bb66c8a580be
Successfully built bb66c8a580be
Successfully tagged harbor.magedu.net/linux36/tomcat-app1:2019-08-02_11_02_30
The push refers to repository [harbor.magedu.net/linux36/tomcat-app1]
0fffa9c585fc: Pushing [======>                                            ]  1.682MB/13.87MB
c9d513cfc5ff: Pushing [==================================================>]  3.584kB
9807541f7ced: Pushing [==================================================>]  4.608kB
4451bc44cd65: Pushed
f09199c0891a: Pushed
```

3.2.3.5：测试tomcat业务镜像启动为容器：

```
# docker  run -it --rm -p 8801:8080 harbor.magedu.net/linux36/tomcat-app1:2019-08-
02_11_02_30
Using CATALINA_BASE:   /apps/tomcat
Using CATALINA_HOME:   /apps/tomcat
Using CATALINA_TMPDIR: /apps/tomcat/temp
Using JRE_HOME:        /usr/local/jdk
Using CLASSPATH:       /apps/tomcat/bin/bootstrap.jar:/apps/tomcat/bin/tomcat-juli.jar
Tomcat started.
127.0.0.1 localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

```
172.17.0.2   9e1909ef9dce
192.168.7.248 k8s-vip.example.com
```

3.2.3.6：访问tomcat业务镜像web页面：



3.2.4：在k8s环境运行tomcat：

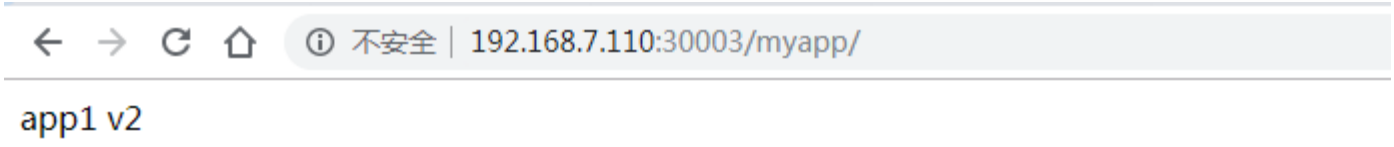
3.2.4.1：创建tomcat业务pod：

```
# pwd
/opt/k8s-data/yaml/linux36/tomcat-app1

# kubectl  apply  -f  tomcat-app1.yaml
deployment.extensions/linux36-tomcat-app1-deployment created
service/linux36-tomcat-app1-service created
```

3.2.4.2：验证pod启动成功：

```
# kubectl  get pods -n  linux36
NAME                                          READY   STATUS    RESTARTS   AGE
linux36-nginx-deployment-b59f56c67-4q6hw      1/1     Running   1          98m
linux36-tomcat-app1-deployment-794d7fcd6-stph2   1/1     Running   0          46s
```

3.2.4.3：测试访问tomcat业务pod的nodeport：

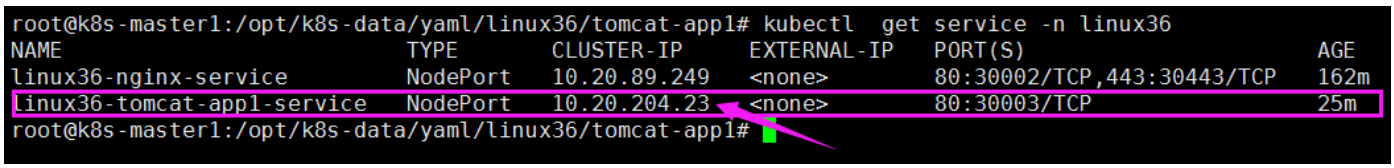


3.3：k8s中nginx+tomcat实现动静分离：

实现一个通用的nginx+tomcat动静分离web架构，即用户访问的静态页面和图片在由nginx直接响应，而动态请求则基于location转发至tomcat。

重点：Nginx基于tomcat的service name转发用户请求到tomcat业务app

3.3.1：查看tomcat app1的server name：



3.3.2：nginx业务镜像配置：

3.3.2.1：nginx配置文件：

```
upstream  tomcat_webserver {
        server  linux36-tomcat-app1-service.linux36.svc.linux36.local:80;
}

server {
        location /myapp {
             proxy_pass  http://tomcat_webserver;
             proxy_set_header   Host    $host;
             proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
             proxy_set_header X-Real-IP $remote_addr;
        }
}
```

3.3.2.3：重新构建nginx业务镜像：

```
root@k8s-master1:/opt/k8s-data/dockerfile/linux36/nginx# bash build-command.sh
Sending build context to Docker daemon  9.728kB
Step 1/6 : FROM harbor.magedu.net/pub-images/nginx-base:v1.14.2
 ---> 494ab6676ad1
Step 2/6 : ADD nginx.conf /usr/local/nginx/conf/nginx.conf
 ---> c271b06ccffa
Step 3/6 : ADD webapp/*  /usr/local/nginx/html/webapp/
 ---> e3846e6fb6c4
Step 4/6 : ADD index.html  /usr/local/nginx/html/index.html
 ---> 648853120b7b
Step 5/6 : EXPOSE 80 443
 ---> Running in 69c97bbcb7d3
Removing intermediate container 69c97bbcb7d3
 ---> 6d10ac0968b1
Step 6/6 : CMD ["nginx"]
 ---> Running in c23ca08bc20b
Removing intermediate container c23ca08bc20b
 ---> b3e844806543
Successfully built b3e844806543
Successfully tagged harbor.magedu.net/linux36/nginx-web1:v1
The push refers to repository [harbor.magedu.net/linux36/nginx-web1]
2b681d328553: Pushed
75c59c96eb3a: Pushed
3ddd53eae577: Pushed
3713c9956100: Layer already exists
```

3.3.2.3：镜像启动为容容器并验证配置文件：

```
root@k8s-master1:~# docker run -it --rm  harbor.magedu.net/linux36/nginx-web1:v1  bash
[root@dee461f9e008 /]#  grep -v "#" /usr/local/nginx/conf/nginx.conf | grep -v "^$"
user  nginx nginx;
worker_processes  auto;
daemon off;
events {
    worker_connections  1024;
}
http {
    include       mime.types;
    default_type  application/octet-stream;
    sendfile        on;
    keepalive_timeout  65;
upstream  tomcat_webserver {
        server  linux36-tomcat-app1-service.linux36.svc.linux36.local:80;
}
    server {
        listen       80;
        server_name  localhost;
        location / {
            root   html;
            index  index.html index.htm;
        }
        location /webapp {
            root   html;
            index  index.html index.htm;
        }
        location /myapp {
             proxy_pass  http://tomcat_webserver;
             proxy_set_header   Host    $host;
             proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
             proxy_set_header X-Real-IP $remote_addr;
        }
```

3.3.3：重新创建业务nginx pod：

两种实现方式

3.3.3.1：删除并重新创建nginx业务镜像：

```
# pwd
/opt/k8s-data/yaml/linux36/nginx

# kubectl  delete -f nginx.yaml
# vim nginx.yaml
image: harbor.magedu.net/linux36/nginx-web1:v1 #更新镜像地址
# kubectl  apply -f nginx.yaml
```

3.3.3.2：更新nginx业务镜像版本号：

重新构建新版本镜像，然后打一个新的tag号，然后通过指定镜像的方式对pod进行更新。

3.3.3.2.1：准备新版本nginx业务镜像：

```
# docker tag harbor.magedu.net/linux36/nginx-web1:v1 harbor.magedu.net/linux36/nginx-web1:v2
# docker push harbor.magedu.net/linux36/nginx-web1:v2
```

3.3.3.2.2：获取当前deployment：

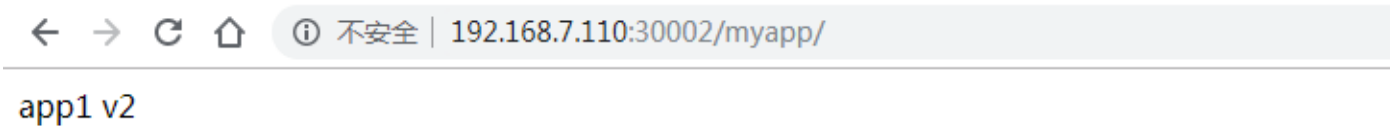
```
# kubectl  get deployment -n linux36
NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
linux36-nginx-deployment          1/1     1            1           7m30s
linux36-tomcat-app1-deployment    1/1     1            1           45m
```

3.3.3.2.3：执行更新nginx业务镜像版本：

```
# kubectl set image deployment/linux36-nginx-deployment  linux36-nginx-container=harbor.magedu.net/linux36/n
ginx-web1:v2 -n linux36deployment.extensions/linux36-nginx-deployment image updated
```

3.3.3.3：web访问测试：

验证能否通过nginx访问到tomcat的app项目



3.4：基于NFS实现动静分离：

图片的上传由后端服务器tomcat完成，图片的读取由前端的nginx响应，就需要nginx与tomcat的数据保持一致性，因此需要将数据保存到k8s环境外部的存储服务器，然后再挂载到各nginx与tomcat 的容器中进行相应的操作。

http://docs.kubernetes.org.cn/429.html #存储卷类型及使用

3.4.1：NFS 服务器环境准备：

```
# mkdir  /data/linux36 -p  #数据总目录
# mkdir /data/linux36/images  #图片目录
# mkdir /data/linux36/static  #静态文件目录

# vim  /etc/exports
/data/linux36 *(rw,no_root_squash)
# systemctl  restart nfs-server
```

3.4.2：NFS客户端挂载并测试写入文件：

```
# mount -t nfs 192.168.7.108:/data/linux36  /mnt
# cp /etc/passwd /mnt/ #必须能够写入数据
```

3.4.3：nginx 业务容器yaml：

```
# cat nginx.yaml
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  labels:
    app: linux36-nginx-deployment-label
  name: linux36-nginx-deployment
  namespace: linux36
spec:
  replicas: 1
  selector:
    matchLabels:
      app: linux36-nginx-selector
  template:
    metadata:
      labels:
        app: linux36-nginx-selector
    spec:
      containers:
      - name: linux36-nginx-container
        image: harbor.magedu.net/linux36/nginx-web1:v1
        #command: ["/apps/tomcat/bin/run_tomcat.sh"]
        #imagePullPolicy: IfNotPresent
        imagePullPolicy: Always
        ports:
        - containerPort: 80
          protocol: TCP
          name: http
        - containerPort: 443
          protocol: TCP
          name: https
        env:
        - name: "password"
          value: "123456"
        - name: "age"
          value: "18"
        resources:
          limits:
            cpu: 2
            memory: 2Gi
          requests:
```

```
            cpu: 500m
            memory: 1Gi
        volumeMounts:
        - name: linux36-images
          mountPath: /usr/local/nginx/html/webapp/images
          readOnly: false
        - name: linux36-static
          mountPath: /usr/local/nginx/html/webapp/static
          readOnly: false
      volumes: #kubectl  explain Deployment.spec.template.spec.volumes
      - name: linux36-images
        nfs:
          server: 192.168.7.108
          path: /data/linux36/images
      - name: linux36-static
        nfs:
          server: 192.168.7.108
          path: /data/linux36/static

---
kind: Service
apiVersion: v1
metadata:
  labels:
    app: linux36-nginx-service-label
  name: linux36-nginx-service
  namespace: linux36
spec:
  type: NodePort
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
    nodePort: 30002
  - name: https
    port: 443
    protocol: TCP
    targetPort: 443
    nodePort: 30443
  selector:
    app: linux36-nginx-selector
```

3.4.4：执行更新yaml文件：

```
# kubectl  apply  -f nginx.yaml
deployment.extensions/linux36-nginx-deployment configured
service/linux36-nginx-service unchanged
```

3.4.5：pod中验证NFS挂载：

```
root@k8s-master1:~#  kubectl  get  pods -n  linux36
NAME                                          READY   STATUS    RESTARTS   AGE
linux36-nginx-deployment-9968c6f5c-8twhk      1/1     Running   0          9m24s
linux36-tomcat-app1-deployment-58695bc978-nrklh   1/1     Running   0          2m39s
root@k8s-master1:~# kubectl  exec -it linux36-nginx-deployment-9968c6f5c-8twhk bash -n linux36
[root@linux36-nginx-deployment-9968c6f5c-8twhk /]# df -TH
Filesystem                          Type     Size  Used Avail Use% Mounted on
overlay                             overlay  105G  5.5G   94G   6% /
tmpfs                               tmpfs     68M     0   68M   0% /dev
tmpfs                               tmpfs    2.1G     0  2.1G   0% /sys/fs/cgroup
/dev/mapper/ubuntu--vg-root         ext4     105G  5.5G   94G   6% /etc/hosts
shm                                 tmpfs     68M     0   68M   0% /dev/shm
tmpfs                               tmpfs    2.1G   13k  2.1G   1% /run/secrets/kubernetes.io/serviceaccount
192.168.7.108:/data/linux36/images nfs4     105G  3.0G   96G   3% /usr/local/nginx/html/webapp/images
192.168.7.108:/data/linux36/static nfs4     105G  3.0G   96G   3% /usr/local/nginx/html/webapp/static
tmpfs                               tmpfs    2.1G     0  2.1G   0% /proc/acpi
tmpfs                               tmpfs    2.1G     0  2.1G   0% /proc/scsi
tmpfs                               tmpfs    2.1G     0  2.1G   0% /sys/firmware
[root@linux36-nginx-deployment-9968c6f5c-8twhk /]#
```

3.4.6：tomcat业务pod更新挂载：

```
root@k8s-master1:/opt/k8s-data/yaml/linux36/tomcat-app1# pwd
/opt/k8s-data/yaml/linux36/tomcat-app1

root@k8s-master1:/opt/k8s-data/yaml/linux36/tomcat-app1# pwd
/opt/k8s-data/yaml/linux36/tomcat-app1
root@k8s-master1:/opt/k8s-data/yaml/linux36/tomcat-app1# cat tomcat-app1.yaml
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  labels:
    app: linux36-tomcat-app1-deployment-label
  name: linux36-tomcat-app1-deployment
  namespace: linux36
spec:
  replicas: 1
  selector:
    matchLabels:
      app: linux36-tomcat-app1-selector
  template:
    metadata:
      labels:
        app: linux36-tomcat-app1-selector
    spec:
      containers:
```

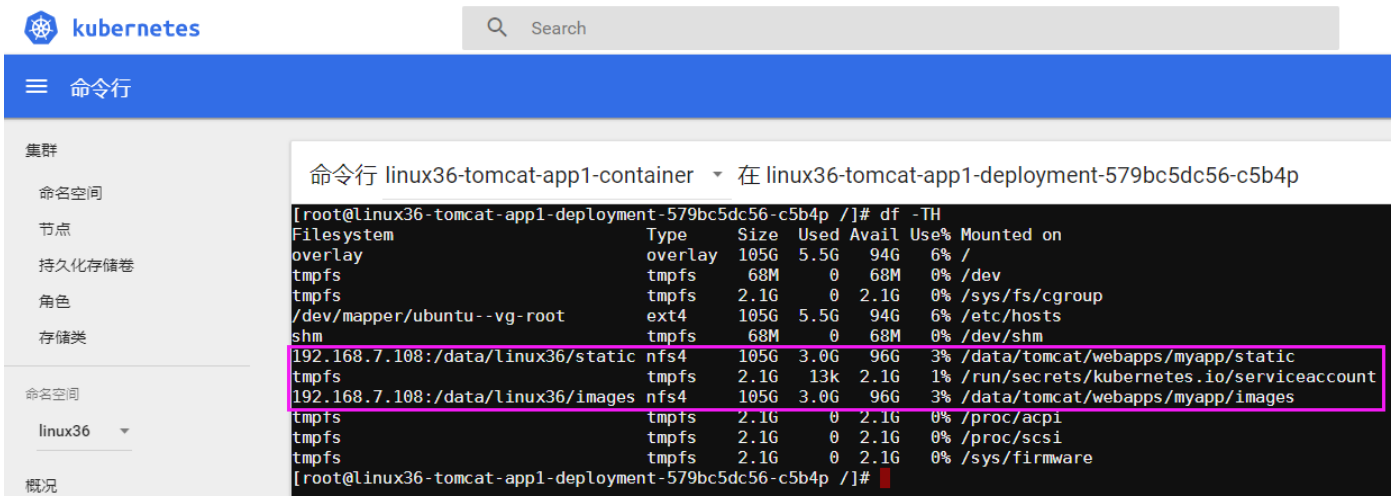
```
      - name: linux36-tomcat-app1-container
        image: harbor.magedu.net/linux36/tomcat-app1:2019-08-02_11_02_30
        #command: ["/apps/tomcat/bin/run_tomcat.sh"]
        #imagePullPolicy: IfNotPresent
        imagePullPolicy: Always
        ports:
        - containerPort: 8080
          protocol: TCP
          name: http
        #env:
        #- name: "password"
        #  value: "123456"
        #- name: "age"
        #  value: "18"
        #resources:
        #  limits:
        #    cpu: 4
        #    memory: 4Gi
        #  requests:
        #    cpu: 2
        #    memory: 4Gi

        volumeMounts:
        - name: linux36-images
          mountPath:  /data/tomcat/webapps/myapp/images
          readOnly: false
        - name: linux36-static
          mountPath:  /data/tomcat/webapps/myapp/static
          readOnly: false
      volumes:
      - name: linux36-images
        nfs:
          server: 192.168.7.108
          path: /data/linux36/images
      - name: linux36-static
        nfs:
          server: 192.168.7.108
          path: /data/linux36/static

---
kind: Service
apiVersion: v1
metadata:
  labels:
    app: linux36-tomcat-app1-service-label
  name: linux36-tomcat-app1-service
  namespace: linux36
spec:
  type: NodePort
```

```
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 8080
    nodePort: 30003
  selector:
    app: linux36-tomcat-app1-selector
```

3.4.7：执行更新tomcat app1业务容器yaml：

```
root@k8s-master1:/opt/k8s-data/yaml/linux36/tomcat-app1# kubectl  apply -f tomcat-app1.yaml
deployment.extensions/linux36-tomcat-app1-deployment configured
service/linux36-tomcat-app1-service unchanged
```

3.4.8：验证tomcat app1业务容器NFS挂载：



3.4.9：访问web测试：

手动祥NFS目录分别上传图片和html文件，测试访问

3.4.9.1：上传数据到NFS：

```
root@k8s-ha1:/data/linux36# tree
.
├── images
│?? └── 1.jpg
└── static
    └── index.html

2 directories, 2 files
```

3.4.9.2：访问nginx 业务pod：

http://192.168.7.110:30002/myapp/images/1.jpg

192.168.7.110:30002/myapp/images/1.jpg

3.4.9.3：访问tomcat业务pod：

```
http://192.168.7.110:30002/webapp/images/1.jpg

http://192.168.7.110:30003/myapp/images/1.jpg
```

192.168.7.110:30002/webapp/images/1.jpg

3.4.5：命令总结：

http://docs.kubernetes.org.cn/

```
# kubectl  get service    --all-namespaces -o wide
# kubectl  get  pods      --all-namespaces -o wide
# kubectl  get  nodes     --all-namespaces -o wide
# kubectl  get deployment --all-namespaces
# kubectl  get deployment -n linux35 -o wide #更改显示格式
# kubectl  describe pods linux35-tomcat-app1-deployment -n linux35 #查看某个资源详细信息
# kubectl  create -f tomcat-app1.yaml
# kubectl  apply  -f tomcat-app1.yaml
# kubectl  delete -f tomcat-app1.yaml
# kubectl  create -f tomcat-app1.yaml  --save-config --record
# kubectl  apply -f tomcat-app1.yaml  --record #推荐命令
# kubectl  exec -it linux35-tomcat-app1-deployment-6bccd8f9c7-g76s5 bash  -n linux35
# kubectl  logs  linux35-tomcat-app1-deployment-6bccd8f9c7-g76s5   -n linux35
# kubectl  delete pods  linux35-tomcat-app1-deployment-6bccd8f9c7-g76s5   -n linux35
```

四：k8s 运维示例：

和运维相关的日常运维事宜

4.1：手动调整pod数量：

kubectl scale 对运行在k8s 环境中的pod 数量进行扩容(增加)或缩容(减小)。

```
当前pod数量：
root@k8s-master1:~# kubectl  get deployment -n linux36
NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
linux36-nginx-deployment          1/1     1            1           3h22m
linux36-tomcat-app1-deployment    1/1     1            1           4h

#查看命令使用帮助
# kubectl  --help   | grep scale
  scale  Set a new size for a Deployment, ReplicaSet, Replication Controller, or Job
# kubectl scale --help

#执行扩容/缩容
# kubectl scale  deployment/linux36-tomcat-app1-deployment  --replicas=2  -n linux36
deployment.extensions/linux36-tomcat-app1-deployment scaled

#验证手动扩容结果
root@k8s-master1:~# kubectl  get deployment -n linux36
NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
linux36-nginx-deployment          1/1     1            1           3h23m
```

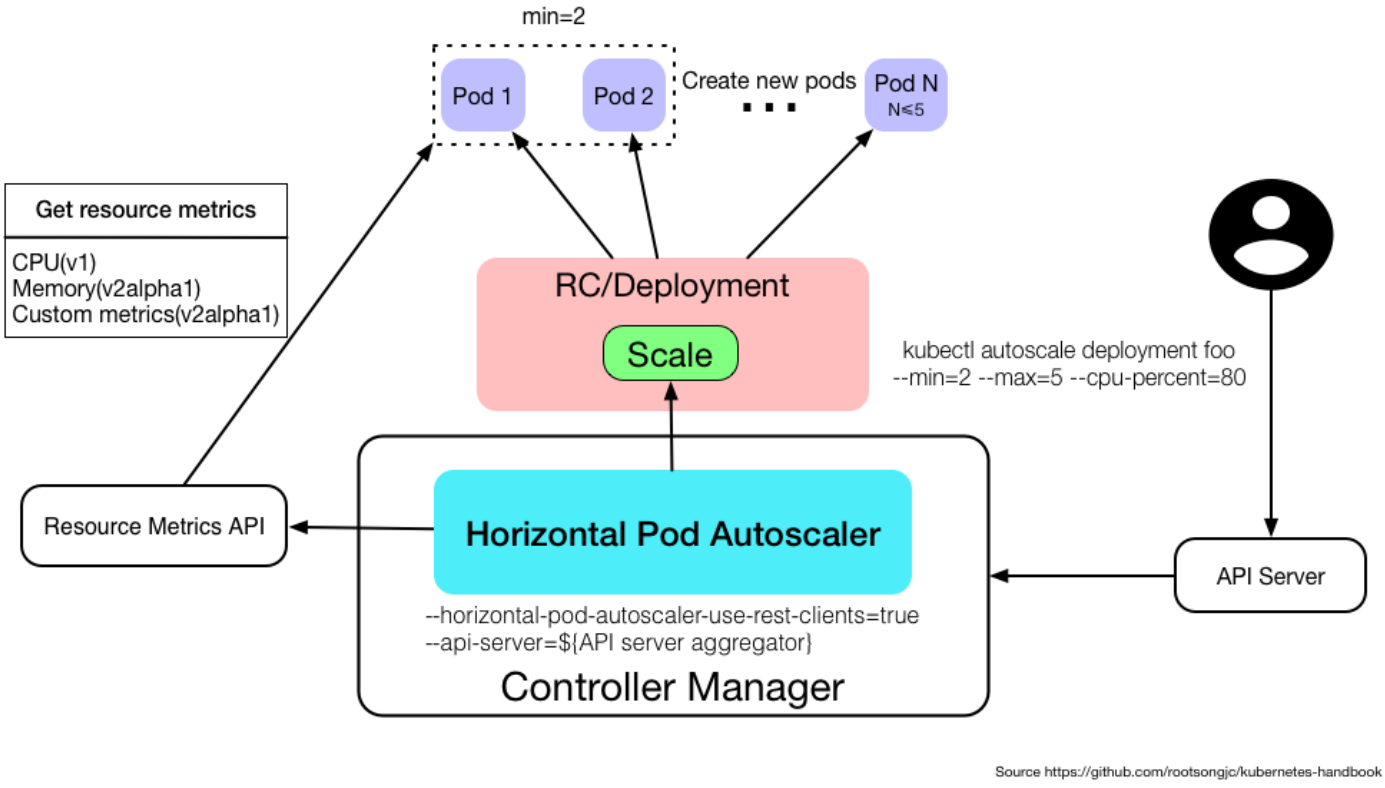
```
linux36-tomcat-app1-deployment   2/2     2            2           4h1m
```

4.2：HPA自动伸缩pod数量：

kubectl autoscale 自动控制在k8s集群中运行的pod数量(水平自动伸缩)，需要提前设置pod范围及触发条件。

k8s从1.1版本开始增加了名称为HPA(Horizontal Pod Autoscaler)的控制器，用于实现基于pod中资源(CPU/Memory)利用率进行对pod的自动扩缩容功能的实现，早期的版本只能基于Heapster组件实现对CPU利用率做为触发条件，但是在k8s 1.11版本开始使用Metrices Server完成数据采集，然后将采集到的数据通过API（Aggregated API，汇总API），例如metrics.k8s.io、custom.metrics.k8s.io、external.metrics.k8s.io，然后再把数据提供给HPA控制器进行查询，以实现基于某个资源利用率对pod进行扩缩容的目的。

```
控制管理器默认每隔15s（可以通过—horizontal-pod-autoscaler-sync-period修改）查询metrics的资源使用情况
支持以下三种metrics指标类型：
  ->预定义metrics（比如Pod的CPU）以利用率的方式计算
  ->自定义的Pod metrics，以原始值（raw value）的方式计算
  ->自定义的object metrics
支持两种metrics查询方式：
  ->Heapster
  ->自定义的REST API
支持多metrics
```



Source https://github.com/rootsongjc/kubernetes-handbook

4.2.1：准备metrics-server：

使用metrics-server作为HPA数据源。

https://github.com/kubernetes-sigs/metrics-server

4.2.1.1：clone代码:

```
https://github.com/kubernetes-sigs/metrics-server.git
# cd metrics-server/
```

4.2.1.2：准备image：

```
测试系统自带的指标数据：
# curl http://localhost:8080/apis/metrics.k8s.io/v1beta1/nodes
# curl http://localhost:8080/apis/metrics.k8s.io/v1beta1/pods

测试指标数据：
# kubectl  top
# kubectl  top nodes #报错如下
Error from server (NotFound): the server could not find the requested resource (get
services http:heapster:)

解决方案：
# docker pull  k8s.gcr.io/metrics-server-amd64:v0.3.5 #google镜像仓库
# docker pull registry.cn-hangzhou.aliyuncs.com/google_containers/metrics-server-
amd64:v0.3.5 #阿里云镜像仓库

或者
# docker tag k8s.gcr.io/metrics-server-amd64:v0.3.5
harbor.magedu.net/baseimages/metrics-server-amd64:v0.3.5
# docker push harbor.magedu.net/baseimages/metrics-server-amd64:v0.3.5
The push refers to repository [harbor.magedu.net/baseimages/metrics-server-amd64]
5f70bf18a086: Mounted from baseimages/heapster-influxdb-amd64
a41f1d5bc0e0: Pushed
3c0f8dc299d8: Pushed
v0.3.3: digest: sha256:e8efd0115ca97461715981072148cc1c8f33a06c9e0d3a34b18396cd1e7623a4
size: 944
```

4.1.2.3：修改yaml文件：

```
# pwd
/root/metrics-server

# cat deploy/1.8+/metrics-server-deployment.yaml
---
apiVersion: v1
kind: ServiceAccount
```

```
metadata:
  name: metrics-server
  namespace: kube-system
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: metrics-server
  namespace: kube-system
  labels:
    k8s-app: metrics-server
spec:
  selector:
    matchLabels:
      k8s-app: metrics-server
  template:
    metadata:
      name: metrics-server
      labels:
        k8s-app: metrics-server
    spec:
      serviceAccountName: metrics-server
      volumes:
      # mount in tmp so we can safely use from-scratch images and/or read-only
containers
      - name: tmp-dir
        emptyDir: {}
      containers:
      - name: metrics-server
        image: harbor.magedu.net/baseimages/metrics-server-amd64:v0.3.5 #本地harbor
        imagePullPolicy: Always
        volumeMounts:
        - name: tmp-dir
          mountPath: /tmp
```

4.1.2.4：创建metrics-server服务：

```
# kubectl  apply  -f deploy/1.8+/
clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator
created
rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
serviceaccount/metrics-server created
deployment.extensions/metrics-server created
service/metrics-server created
clusterrole.rbac.authorization.k8s.io/system:metrics-server created
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
```

4.1.2.5：验证metrics-server pod：

```
root@k8s-master1:~/metrics-server# kubectl  get pods  -n  kube-system
NAME                                    READY   STATUS    RESTARTS   AGE
calico-kube-controllers-578fbfffbc-9nmjz   1/1     Running   5          23d
calico-node-4z9zp                       2/2     Running   11         23d
calico-node-5vkd9                       2/2     Running   12         23d
calico-node-mxstr                       2/2     Running   9          23d
calico-node-rq985                       2/2     Running   4          23d
heapster-648cbf96fb-brnxr               1/1     Running   5          23d
kube-dns-75d7c6cd6f-5nk6h               3/3     Running   3          9h
kubernetes-dashboard-7dc549fb78-n4m5h   1/1     Running   2          9h
metrics-server-64878d678d-d4cjh         1/1     Running   0          13m
monitoring-grafana-7688bf4dbc-hspmx     1/1     Running   5          23d
monitoring-influxdb-7d7988665d-wqqz7    1/1     Running   5          23d
root@k8s-master1:~/metrics-server#
```

验证metrics-server 是否采集到node数据：

```
root@k8s-master1:/opt/metrics-server/deploy/1.8+# kubectl  top nodes
NAME           CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
172.18.0.101   167m         9%     806Mi           25%
172.18.0.108   349m         9%     1229Mi          29%
172.18.0.109   382m         10%    1243Mi          23%
```

验证metrics-server 是否采集到pod数据：

```
root@k8s-master1:/opt# kubectl  top nodes
NAME           CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
172.31.7.101   285m         7%     1802Mi          55%
172.31.7.102   249m         6%     1853Mi          57%
172.31.7.103   226m         5%     1780Mi          55%
172.31.7.111   138m         3%     1463Mi          28%
172.31.7.112   132m         3%     1478Mi          28%
172.31.7.113   117m         2%     1469Mi          28%

root@k8s-master1:/opt# kubectl  top pods  -n default
NAME        CPU(cores)   MEMORY(bytes)
busybox     0m           0Mi
net-test1   0m           1Mi
net-test2   0m           1Mi
net-test3   0m           0Mi
net-test4   0m           0Mi
```

4.1.2.6：修改controller-manager启动参数：

```
# kube-controller-manager  --help | grep horizontal-pod-autoscaler-sync-period
      --horizontal-pod-autoscaler-sync-period duration  The period for syncing the
number of pods in horizontal pod autoscaler. (default 15s)
      #定义pod数量水平伸缩的间隔周期，默认15秒
```

```
      --horizontal-pod-autoscaler-cpu-initialization-period  duration  The period after
pod start when CPU samples might be skipped. (default 5m0s)
      #用于设置 pod 的初始化时间， 在此时间内的 pod，CPU 资源指标将不会被采纳，默认为5分钟

     --horizontal-pod-autoscaler-initial-readiness-delay duration The period after pod
start during which readiness changes will be treated as initial readiness. (default
30s)
      #用于设置 pod 准备时间， 在此时间内的 pod 统统被认为未就绪及不采集数据,默认为30秒


# vim /etc/systemd/system/kube-controller-manager.service
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/GoogleCloudPlatform/kubernetes

[Service]
ExecStart=/usr/bin/kube-controller-manager \
  --address=127.0.0.1 \
  --master=http://127.0.0.1:8080 \
  --allocate-node-cidrs=true \
  --service-cluster-ip-range=10.20.0.0/16 \
  --cluster-cidr=172.31.0.0/16 \
  --cluster-name=kubernetes \
  --cluster-signing-cert-file=/etc/kubernetes/ssl/ca.pem \
  --cluster-signing-key-file=/etc/kubernetes/ssl/ca-key.pem \
  --service-account-private-key-file=/etc/kubernetes/ssl/ca-key.pem \
  --root-ca-file=/etc/kubernetes/ssl/ca.pem \
  --horizontal-pod-autoscaler-use-rest-clients=true \
  --leader-elect=true \
  --horizontal-pod-autoscaler-use-rest-clients=true \ #是否使用其他客户端数据
  --horizontal-pod-autoscaler-sync-period=10s \
  --v=2
Restart=on-failure
RestartSec=5

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

4.1.2.6：重启controller-manager：

验证以上参数是否生效

```
root@k8s-master1:~# ps -ef | grep kube-controller-manager
root      13687      1  3 22:49 ?        00:00:04 /usr/bin/kube-controller-manager --address=127.0.0.1 --
master=http://127.0.0.1:8080 --allocate-node-cidrs=true --service-cluster-ip-range=172.31.0.0/16 --cluste
r-cidr=10.10.0.0/16 --cluster-name=kubernetes --cluster-signing-cert-file=/etc/kubernetes/ssl/ca.pem --cl
uster-signing-key-file=/etc/kubernetes/ssl/ca-key.pem --node-cidr-mask-size=24 --service-account-private-
key-file=/etc/kubernetes/ssl/ca-key.pem --root-ca-file=/etc/kubernetes/ssl/ca.pem --horizontal-pod-autosc
aler-use-rest-clients=true --horizontal-pod-autoscaler-sync-period=10s --leader-elect=true --v=2
root      15039   4450  0 22:51 pts/0    00:00:00 grep --color=auto kube-controller-manager
root@k8s-master1:~#
```

4.2.1：通过命令配置扩缩容：

```
# kubectl  autoscale deployment/nginx-deployment --min=2 --max=10 --cpu-percent=80 -n
magedu
horizontalpodautoscaler.autoscaling/nginx-deployment autoscaled

# kubectl  autoscale deployment/linux36-nginx-deployment    --min=2 --max=10 --cpu-
percent=80 -n linux36
horizontalpodautoscaler.autoscaling/linux36-nginx-deployment autoscaled


验证信息：
# kubectl describe  deployment/linux36-nginx-deployment    -n linux36
desired 最终期望处于READY状态的副本数
updated 当前完成更新的副本数
total 总计副本数
available 当前可用的副本数
unavailable 不可用副本数
```

```
root@k8s-master1:~# kubectl describe  deployment/linux36-nginx-deployment    -n linux36
Name:                   linux36-nginx-deployment
Namespace:              linux36
CreationTimestamp:      Fri, 02 Aug 2019 11:53:57 +0800
Labels:                 app=linux36-nginx-deployment-label
Annotations:            deployment.kubernetes.io/revision: 3
                        kubectl.kubernetes.io/last-applied-configuration:
                          {"apiVersion":"extensions/v1beta1","kind":"Deployment","metadata":{"annotations"
ployment-label"},"nam...
Selector:               app=linux36-nginx-selector
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  1 max unavailable, 1 max surge
Pod Template:
  Labels:  app=linux36-nginx-selector
```

4.2.2：yaml文件中定义扩缩容配置：

定义在tomcat服务中的yaml中

```
# pwd
/opt/k8s-data/yaml/linux36/tomcat-app1

# cat tomcat-app1.yaml
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  labels:
    app: linux36-tomcat-app1-deployment-label
  name: linux36-tomcat-app1-deployment
  namespace: linux36
spec:
  replicas: 1
```

```
  selector:
    matchLabels:
      app: linux36-tomcat-app1-selector
  template:
    metadata:
      labels:
        app: linux36-tomcat-app1-selector
    spec:
      containers:
      - name: linux36-tomcat-app1-container
        image: harbor.magedu.net/linux36/tomcat-app1:2019-08-02_11_02_30
        #command: ["/apps/tomcat/bin/run_tomcat.sh"]
        #imagePullPolicy: IfNotPresent
        imagePullPolicy: Always
        ports:
        - containerPort: 8080
          protocol: TCP
          name: http
        #env:
        #- name: "password"
        #  value: "123456"
        #- name: "age"
        #  value: "18"
        #resources:
        #  limits:
        #    cpu: 4
        #    memory: 4Gi
        #  requests:
        #    cpu: 2
        #    memory: 4Gi

        volumeMounts:
        - name: linux36-images
          mountPath:  /data/tomcat/webapps/myapp/images
          readOnly: false
        - name: linux36-static
          mountPath:  /data/tomcat/webapps/myapp/static
          readOnly: false
      volumes:
      - name: linux36-images
        nfs:
          server: 192.168.7.108
          path: /data/linux36/images
      - name: linux36-static
        nfs:
          server: 192.168.7.108
          path: /data/linux36/static

---
```

```
kind: Service
apiVersion: v1
metadata:
  labels:
    app: linux36-tomcat-app1-service-label
  name: linux36-tomcat-app1-service
  namespace: linux36
spec:
  type: NodePort
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 8080
    nodePort: 30003
  selector:
    app: linux36-tomcat-app1-selector

---
apiVersion: autoscaling/v2beta1 #定义API版本
kind: HorizontalPodAutoscaler  #对象类型
metadata: #定义对象元数据
  namespace: linux36 #创建后隶属的namespace
  name: linux36-tomcat-app1-podautoscaler #对象名称
  labels: 这样的label标签
    app: linux36-tomcat-app1 #自定义的label名称
    version: v2beta1 #自定义的api版本
spec: #定义对象具体信息
  scaleTargetRef: #定义水平伸缩的目标对象，Deployment、ReplicationController/ReplicaSet
    apiVersion: apps/v1
    #API版本，HorizontalPodAutoscaler.spec.scaleTargetRef.apiVersion
    kind: Deployment #目标对象类型为deployment
    name: linux36-tomcat-app1-deployment  #deployment 的具体名称
  minReplicas: 2 #最小pod数
  maxReplicas: 5 #最大pod数
  metrics: #调用metrics数据定义
  - type: Resource #类型为资源
    resource:  #定义资源
      name: cpu #资源名称为cpu
      targetAverageUtilization: 80 #CPU使用率
  - type: Resource  #类型为资源
    resource:  #定义资源
      name: memory #资源名称为memory
      targetAverageValue: 1024Mi   #memory使用率
```

4.2.3：验证HPA：

```
  linux36-static:
    Type:      NFS (an NFS mount that lasts the lifetime of a pod)
    Server:    192.168.7.108
    Path:      /data/linux36/static
    ReadOnly:  false
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
OldReplicaSets:  <none>
NewReplicaSet:   linux36-tomcat-app1-deployment-579bc5dc56 (1/1 replicas created)
Events:
  Type    Reason             Age                  From                   Message
  ----    ------             ----                 ----                   -------
  Normal  ScalingReplicaSet  17m (x2 over 71m)    deployment-controller  Scaled up replica set linux36-tomcat-app1-deployment-579bc5dc56 to 2
  Normal  ScalingReplicaSet  16m                  deployment-controller  Scaled up replica set linux36-tomcat-app1-deployment-579bc5dc56 to 4
  Normal  ScalingReplicaSet  15m (x2 over 20m)    deployment-controller  Scaled up replica set linux36-tomcat-app1-deployment-579bc5dc56 to 5
  Normal  ScalingReplicaSet  7m55s (x3 over 24m)  deployment-controller  Scaled down replica set linux36-tomcat-app1-deployment-579bc5dc56 to 1
root@k8s-master1:/opt/k8s-data/yaml/linux36/tomcat-app1#
```

验证扩容、缩容效果：

```
  linux36-static:
    Type:      NFS (an NFS mount that lasts the lifetime of a pod)
    Server:    192.168.7.108
    Path:      /data/linux36/static
    ReadOnly:  false
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
OldReplicaSets:  <none>
NewReplicaSet:   linux36-tomcat-app1-deployment-579bc5dc56 (1/1 replicas created)
Events:
  Type    Reason             Age                  From                   Message
  ----    ------             ----                 ----                   -------
  Normal  ScalingReplicaSet  17m (x2 over 71m)    deployment-controller  Scaled up replica set linux36-tomcat-app1-deployment-579bc5dc56 to 2
  Normal  ScalingReplicaSet  16m                  deployment-controller  Scaled up replica set linux36-tomcat-app1-deployment-579bc5dc56 to 4
  Normal  ScalingReplicaSet  15m (x2 over 20m)    deployment-controller  Scaled up replica set linux36-tomcat-app1-deployment-579bc5dc56 to 5
  Normal  ScalingReplicaSet  7m55s (x3 over 24m)  deployment-controller  Scaled down replica set linux36-tomcat-app1-deployment-579bc5dc56 to 1
root@k8s-master1:/opt/k8s-data/yaml/linux36/tomcat-app1#
```

4.2.4：配置自动扩缩容：

先手动扩容至5个，验证在空闲时间是否会自动缩容

4.2.4.1：将pod扩容至5个：

容器组

名称	节点	状态	已重启	已创建	CPU（核）	内存（字节）
linux36-tomcat-app1-deployment-5fd...	192.168.7.111	Running	0	4 秒	-	-
linux36-tomcat-app1-deployment-5fd...	192.168.7.111	Running	0	4 秒	-	-
linux36-tomcat-app1-deployment-5fd...	192.168.7.110	Running	0	4 秒	-	-
linux36-tomcat-app1-deployment-5fd...	192.168.7.110	Running	0	9 分钟	0.002	94.430 Mi
linux36-tomcat-app1-deployment-5fd...	192.168.7.111	Running	0	9 分钟	0.002	94.055 Mi

4.2.4.2：验证HPA日志：

空闲一段时间，验证是否会对容器扩缩容

```
root@k8s-master1:~# kubectl  get hpa -n linux36
NAME                                    REFERENCE                                                TARGETS
                  M
INPODS    MAXPODS    REPLICAS    AGElinux36-tomcat-app1-podautoscaler
Deployment/linux36-tomcat-app1-deployment    <unknown>/200Mi, 0%/80%    2
         5          4           10m
root@k8s-master1:~# kubectl   describe hpa linux36-tomcat-app1-podautoscaler  -n
linux36
Events:
  Type     Reason              Age     From                       Message
  ----     ------              ----    ----                       -------
  Normal   SuccessfulRescale   5m52s   horizontal-pod-autoscaler  New size: 5; reason:
Current number of replicas above Spec.MaxReplicas
  Normal   SuccessfulRescale   8s      horizontal-pod-autoscaler  New size: 1; reason: All
metrics below target
```

kubernetes v1.15

```
root@k8s-master1:~# kubectl   describe hpa linux36-tomcat-app1-podautoscaler  -n linux36
Name:                                                  linux36-tomcat-app1-podautoscaler
Namespace:                                             linux36
Labels:                                                app=linux36-tomcat-app1
                                                       version=v2beta1
Annotations:                                           kubectl.kubernetes.io/last-applied-configuration:
                                                         {"apiVersion":"autoscaling/v2beta1","kind":"HorizontalPodAutoscaler","metadata":{"annotations":{},"labels":{"app":"
linux36-tomcat-app1","v...
CreationTimestamp:                                     Fri, 02 Aug 2019 20:05:13 +0800
Reference:                                             Deployment/linux36-tomcat-app1-deployment
Metrics:                                               ( current / target )
  resource memory on pods:                             <unknown> / 200Mi
  resource cpu on pods  (as a percentage of request):  0% (2m) / 80%
Min replicas:                                          2
Max replicas:                                          5
Deployment pods:                                       2 current / 2 desired
Conditions:
  Type            Status  Reason              Message
  ----            ------  ------              -------
  AbleToScale     True    ScaleDownStabilized  recent recommendations were higher than current one, applying the highest recent recommendation
  ScalingActive   True    ValidMetricFound     the HPA was able to successfully calculate a replica count from memory resource
  ScalingLimited  False   DesiredWithinRange   the desired count is within the acceptable range
Events:
  Type     Reason                        Age                From                       Message
  ----     ------                        ----               ----                       -------
  Normal   SuccessfulRescale             16m                horizontal-pod-autoscaler  New size: 2; reason: Current number of replicas below Spec.MinReplicas
  Normal   SuccessfulRescale             16m                horizontal-pod-autoscaler  New size: 3; reason: memory resource above target
  Warning  FailedGetResourceMetric       15m (x4 over 16m)  horizontal-pod-autoscaler  did not receive metrics for any ready pods
  Warning  FailedComputeMetricsReplicas  15m (x4 over 16m)  horizontal-pod-autoscaler  failed to get cpu utilization: did not receive metrics for any ready pods
  Normal   SuccessfulRescale             7m34s              horizontal-pod-autoscaler  New size: 4; reason: All metrics below target
  Normal   SuccessfulRescale             2m34s              horizontal-pod-autoscaler  New size: 3; reason: All metrics below target
  Normal   SuccessfulRescale             93s (x2 over 11m)  horizontal-pod-autoscaler  New size: 2; reason: All metrics below target
root@k8s-master1:~#
```

kubernetes v1.19

```
Annotations:                                           <none>
CreationTimestamp:                                     Tue, 15 Jun 2021 17:24:50 +0800
Reference:                                             Deployment/nginx-deployment
Metrics:                                               ( current / target )
  resource cpu on pods  (as a percentage of request):  0% (0) / 60%
Min replicas:                                          2
Max replicas:                                          10
Deployment pods:                                       2 current / 2 desired
Conditions:
  Type            Status  Reason            Message
  ----            ------  ------            -------
  AbleToScale     True    ReadyForNewScale  recommended size matches current size
  ScalingActive   True    ValidMetricFound  the HPA was able to successfully calculate a replica count from cpu
  ScalingLimited  True    TooFewReplicas    the desired replica count is less than the minimum replica count
Events:
  Type    Reason             Age    From                       Message
  ----    ------             ----   ----                       -------
  Normal  SuccessfulRescale  3m15s  horizontal-pod-autoscaler  New size: 2; reason: All metrics below target
root@k8s-master1:/opt#
```

4.3：动态修改资源内容kubectl edit：

用于临时修改某些配置后需要立即生效的场景

```
# kubectl  get deployment -n linux36
NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
linux36-nginx-deployment          4/4     4            4           5h48m
linux36-tomcat-app1-deployment    5/5     5            5           19m

# kubectl  edit deployment linux36-nginx-deployment -n linux36 #修改副本数/镜像地址

# kubectl  get  pods  -n linux36 #验证副本数是否与edit编辑之后的一致
NAME                                           READY   STATUS    RESTARTS   AGE
linux36-nginx-deployment-9968c6f5c-8twhk       1/1     Running   0          5h14m
linux36-nginx-deployment-9968c6f5c-r97xm       1/1     Running   0          38m
linux36-nginx-deployment-9968c6f5c-txlmm       1/1     Running   0          109m
```

4.4：定义node资源标签：

lable是一个键值对，创建pod的时候会查询那些node有这个标签，只会将pod创建在符合指定label值的node节点上。

4.4.1：查看当前node label：

```
# kubectl describe  node 192.168.7.110
Name:               192.168.7.110
Roles:              node
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/hostname=192.168.7.110
                    kubernetes.io/role=node
.......略!
```

4.4.2：自定义node label并验证：

```
# kubectl  label node  192.168.7.110 project=linux36
node/192.168.7.110 labeled

# kubectl label nodes 192.168.7.110  test_label=test
node/192.168.7.110 labeled
```

```
root@k8s-master1:/opt/k8s-data/yaml/linux36/nginx# kubectl describe  nodes 192.168.7.110
Name:               192.168.7.110
Roles:              node
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/hostname=192.168.7.110
                    kubernetes.io/role=node
                    project=linux36
                    test_label=test
Annotations:        node.alpha.kubernetes.io/ttl: 0
                    volumes.kubernetes.io/controller-managed-attach-detach: true
```

4.4.3：yaml引用node label：

```
root@k8s-master1:/opt/k8s-data/yaml/linux36/nginx# cat nginx.yaml
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  labels:
    app: linux36-nginx-deployment-label
  name: linux36-nginx-deployment
  namespace: linux36
spec:
  replicas: 1
  selector:
    matchLabels:
      app: linux36-nginx-selector
  template:
    metadata:
      labels:
        app: linux36-nginx-selector
    spec:
      containers:
      - name: linux36-nginx-container
        image: harbor.magedu.net/linux36/nginx-web1:v1
        #command: ["/apps/tomcat/bin/run_tomcat.sh"]
        #imagePullPolicy: IfNotPresent
        imagePullPolicy: Always
        ports:
        - containerPort: 80
          protocol: TCP
          name: http
        - containerPort: 443
          protocol: TCP
          name: https
        env:
        - name: "password"
          value: "123456"
        - name: "age"
          value: "18"
        resources:
```

```
          limits:
            cpu: 2
            memory: 2Gi
          requests:
            cpu: 500m
            memory: 1Gi
        volumeMounts:
        - name: linux36-images
          mountPath: /usr/local/nginx/html/webapp/images
          readOnly: false
        - name: linux36-static
          mountPath: /usr/local/nginx/html/webapp/static
          readOnly: false
      volumes:
      - name: linux36-images
        nfs:
          server: 192.168.7.108
          path: /data/linux36/images
      - name: linux36-static
        nfs:
          server: 192.168.7.108
          path: /data/linux36/static
      nodeSelector: #位置在当前containers参数结束后的部分
        project: linux36 #指定的label标签
```

4.4.4：应用yaml文件：

```
# pwd
/opt/k8s-data/yaml/linux36/nginx

# kubectl  apply  -f  nginx.yaml
# kubectl  get pods  -o wide -n linux36 #验证pod运行到了指定的node节点上
NAME                                         READY   STATUS             RESTARTS   AGE
  IP              NODE            NOMINATED NODE   READIN
ESS GATESlinux36-nginx-deployment-768c7fcb8-5dcq2   0/1     CrashLoopBackOff   5
   3m32s   172.31.58.121   192.168.7.110   <none>           <none
```

4.4.5：删除自定义node label：

```
删除自定义标签：
# kubectl  label nodes 192.168.7.110 test_label-
node/192.168.7.110 labeled
```

```
root@k8s-master1:/opt/k8s-data/yaml/linux36/nginx# kubectl  label nodes 192.168.7.110 test_label-
node/192.168.7.110 labeled
root@k8s-master1:/opt/k8s-data/yaml/linux36/nginx# kubectl describe  nodes 192.168.7.110
Name:               192.168.7.110
Roles:              node
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/hostname=192.168.7.110
                    kubernetes.io/role=node
                    project=linux36
Annotations:        node.alpha.kubernetes.io/ttl: 0
```

4.5：业务镜像版本升级及回滚：

在指定的deployment中通过kubectl set image指定新版本的 镜像:tag 来实现更新代码的目的。

构建三个不同版本的nginx镜像，第一次使用v1版本，后组逐渐升级到v2与v3，测试镜像版本升级与回滚操作

deployment控制器支持两种更新策略：默认为滚动更新

1.滚动更新(rolling update)：

滚动更新是默认的更新策略，滚动更新是基于新版本镜像创建新版本pod，然后删除一部分旧版本pod，然后再创建新版本pod，再删除一部分旧版本pod，直到就版本pod删除完成，滚动更新优势是在升级过程当中不会导致服务不可用，缺点是升级过程中会导致两个版本在短时间内会并存。

具体升级过程是在执行更新操作后k8s会再创建一个新版本的ReplicaSet控制器，在删除旧版本的ReplicaSet控制器下的pod的同时会在新版本的ReplicaSet控制器下创建新的pod，直到旧版本的pod全部被删除完后再把就版本的ReplicaSet控制器也回收掉。

在执行滚动更新的同时，为了保证服务的可用性，当前控制器内不可用的pod(pod需要拉取镜像执行创建和执行探针探测期间是不可用的)不能超出一定范围，因为需要至少保留一定数量的pod以保证服务可以被客户端正常访问，可以通过以下参数指定： #kubectl explain deployment.spec.strategy

deployment.spec.strategy.rollingUpdate.maxSurge #指定在升级期间pod总数可以超出定义好的期望的pod数的个数或者百分比，默认为25%，如果设置为10%，假如当前是100个pod，那么升级时最多将创建110个pod即额外有10%的pod临时会超出当前(replicas)指定的副本数限制。

deployment.spec.strategy.rollingUpdate.maxUnavailable #指定在升级期间最大不可用的pod数，可以是整数或者当前pod的百分比，默认是25%，假如当前是100个pod，那么升级时最多可以有25个(25%)pod不可用即还要75个(75%)pod是可用的。

#注意：以上两个值不能同时为0，如果maxUnavailable最大不可用pod为0，maxSurge超出pod数也为0，那么将会导致pod无法进行滚动更新。

2.重建更新(recreate)：

先删除现有的pod，然后基于新版本的镜像重建，优势是同时只有一个版本在线，不会产生多版本在线问题，缺点是pod删除后到pod重建成功中间的时间会导致服务无法访问，因此较少使用。

4.5.1：升级到镜像到指定版本：

```
# kubectl apply -f nginx.yaml --record=true #v1版本，--record=true为记录执行的kubectl

#镜像更新命令格式为：
# kubectl set image  deployment/deployment-name containers-name=image -n namespace

#V2
#  kubectl set image  deployment/linux36-nginx-deployment   linux36-nginx-
container=harbor.magedu.net/linux36/nginx-web1:v2  -n linux36
deployment.extensions/linux36-nginx-deployment image updated

#V3
#  kubectl set image  deployment/linux36-nginx-deployment   linux36-nginx-
container=harbor.magedu.net/linux36/nginx-web1:v3  -n linux36
deployment.extensions/linux36-nginx-deployment image updated
```

4.5.2：查看历史版本信息：

```
查看历史版本信息：
# kubectl  rollout history deployment/linux36-nginx-deployment -n linux36
deployment.extensions/linux36-nginx-deployment
REVISION  CHANGE-CAUSE
1         kubectl apply --filename=nginx.yaml --record=true
2         kubectl apply --filename=nginx.yaml --record=true
3         kubectl apply --filename=nginx.yaml --record=true
```

4.5.3：回滚到上一个版本：

```
# kubectl rollout undo deployment/linux36-nginx-deployment   -n linux36
deployment.extensions/linux36-nginx-deployment rolled back
```

4.5.4：回滚到指定版本：

```
查看当前版本号：
# kubectl  rollout history deployment/linux36-nginx-deployment -n linux36
deployment.extensions/linux36-nginx-deployment
REVISION  CHANGE-CAUSE
1         kubectl apply --filename=nginx.yaml --record=true
3         kubectl apply --filename=nginx.yaml --record=true
4         kubectl apply --filename=nginx.yaml --record=true

# kubectl rollout undo deployment/linux36-nginx-deployment --to-revision=1  -n linux36
deployment.extensions/linux36-nginx-deployment rolled back

回滚后的版本号：
```

```
# kubectl  rollout history deployment/linux36-nginx-deployment -n linux36
deployment.extensions/linux36-nginx-deployment
REVISION  CHANGE-CAUSE
3         kubectl apply --filename=nginx.yaml --record=true
4         kubectl apply --filename=nginx.yaml --record=true
5         kubectl apply --filename=nginx.yaml --record=true
```

4.6：配置主机为封锁状态且不参与调度：

```
# kubectl   --help | grep cordon #警戒线
  cordon          Mark node as unschedulable #标记为警戒，即不参加pod调度
  uncordon        Mark node as schedulable #去掉警戒，即参加pod调度

#设置192.168.7.110不参加调度
root@k8s-master1:/opt/k8s-data/yaml/linux36/nginx#  kubectl  cordon 192.168.7.110
node/192.168.7.110 cordoned
root@k8s-master1:/opt/k8s-data/yaml/linux36/nginx# kubectl  get node
NAME             STATUS                     ROLES    AGE   VERSION
192.168.7.101    Ready,SchedulingDisabled   master   23d   v1.13.5
192.168.7.102    Ready,SchedulingDisabled   master   23d   v1.13.5
192.168.7.110    Ready,SchedulingDisabled   node     23d   v1.13.5
192.168.7.111    Ready                      node     23d   v1.13.5

#设置192.168.7.110参加调度
root@k8s-master1:/opt/k8s-data/yaml/linux36/nginx#  kubectl  uncordon 192.168.7.110
node/192.168.7.110 uncordoned
root@k8s-master1:/opt/k8s-data/yaml/linux36/nginx# kubectl  get node
NAME             STATUS                     ROLES    AGE   VERSION
192.168.7.101    Ready,SchedulingDisabled   master   23d   v1.13.5
192.168.7.102    Ready,SchedulingDisabled   master   23d   v1.13.5
192.168.7.110    Ready                      node     23d   v1.13.5
192.168.7.111    Ready                      node     23d   v1.13.5
```

4.7：从etcd删除pod：

适用于自动化场景

4.7.1：查看和namespace相关的数据：

```
# ETCDCTL_API=3 etcdctl get  /registry/  --prefix --keys-only | grep linux36
```

```
/registry/events/linux36/linux36-nginx-deployment-7856b9544d.15b6f4be24dcd488
/registry/events/linux36/linux36-nginx-deployment-7856b9544d.15b6f4d0e4ef4a0b
/registry/events/linux36/linux36-nginx-deployment.15b6f532e61fde16
/registry/events/linux36/linux36-nginx-service.15b6f46c285307f3
/registry/namespaces/linux36
/registry/pods/linux36/linux36-nginx-deployment-59d4b875db-mj88v
/registry/replicasets/linux36/linux36-nginx-deployment-59d4b875db
/registry/secrets/linux36/default-token-l94cr
/registry/serviceaccounts/linux36/default
/registry/services/endpoints/linux36/linux36-nginx-service
/registry/services/specs/linux36/linux36-nginx-service
root@k8s-etcd2:~#
```

4.7.2：从etcd查看具体某个对象的数据：

```
# ETCDCTL_API=3 etcdctl get /registry/pods/linux36/linux36-nginx-deployment-59d4b875db-mj88v
```

```
root@k8s-etcd2:~# ETCDCTL_API=3 etcdctl get /registry/pods/linux36/linux36-nginx-deployment-59d4b875db-mj88v
/registry/pods/linux36/linux36-nginx-deployment-59d4b875db-mj88v
k8s

v1Pod,

)linux36-nginx-deployment-59d4b875db-mj88v$linux36-nginx-deployment-59d4b875db-linux36"*$aa4a57bb-b4c0-11e9-8f0f-000c2926f
9e82¿ 蓼
applinux36-nginx-selectorZ
pod-template-hash
59d4b875dbjd

ReplicaSet#linux36-nginx-deployment-59d4b875db"$aa4830b7-b4c0-11e9-8f0f-000c2926f9e8*apps/v108z
1
default-token-l94cr2
default-token-l94cr¤
```

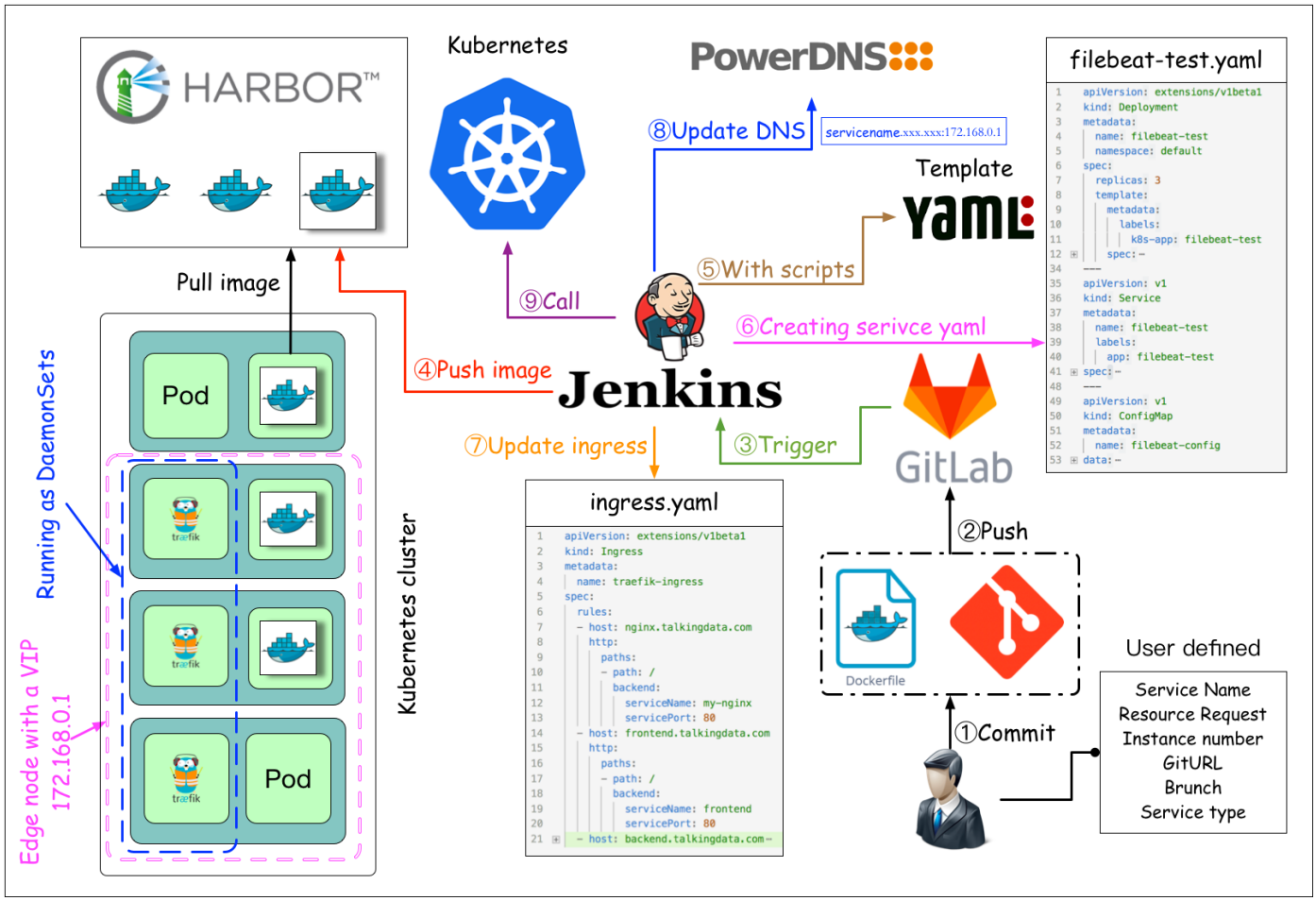
4.7.3：删除etcd指定资源：

```
root@k8s-etcd2:~# ETCDCTL_API=3 etcdctl del /registry/pods/linux36/linux36-nginx-deployment-59d4b875db-mj88v
1 #返回值为1表示命令执行成功
root@k8s-etcd2:~# ETCDCTL_API=3 etcdctl del /registry/pods/linux36/linux36-nginx-deployment-59d4b875db-mj88v
0 #返回值为0表示命令执行失败
root@k8s-etcd2:~#
```

```
root@k8s-etcd2:~# ETCDCTL_API=3 etcdctl del /registry/pods/linux36/linux36-nginx-deployment-59d4b875db-mj88v
1
root@k8s-etcd2:~# ETCDCTL_API=3 etcdctl del /registry/pods/linux36/linux36-nginx-deployment-59d4b875db-mj88v
0
root@k8s-etcd2:~#
```

五：持续集成与部署

持续集成与部署



5.1：jenkins环境准备：

java配置：

```
root@s4:/usr/local/src# tar xvf jdk-8u192-linux-x64.tar.gz
root@s4:/usr/local/src# ln  -sv /usr/local/src/jdk1.8.0_192 /usr/local/jdk
root@s4:~# vim /etc/profile
export JAVA_HOME=/usr/local/jdk
export JRE_HOME=$JAVA_HOME/jre
export PATH=$JAVA_HOME/bin:$JRE_HOME/bin:$PATH
export CLASSPATH=$JAVA_HOME/lib:$JRE_HOME/lib:.

root@s4:~# source  /etc/profile
root@s4:~# java -version
java version "1.8.0_192"
Java(TM) SE Runtime Environment (build 1.8.0_192-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.192-b12, mixed mode)

root@s4:~# apt install daemon
root@s4:~# dpkg -i jenkins_2.138.4_all.deb
```

5.2：gitlab环境准备：

见 Devops 之——基于Jenkins的CI与CD

5.3：脚本内容：

```
root@k8s-ha1:~# cat  /root/scripts/linux36-ningx-deploy.sh
#!/bin/bash
#Author: ZhangShiJie
#Date: 2019-08-03
#Version: v1

#记录脚本开始执行时间
starttime=`date +'%Y-%m-%d %H:%M:%S'`

#变量
SHELL_DIR="/root/scripts"
SHELL_NAME="$0"
K8S_CONTROLLER1="192.168.7.101"
K8S_CONTROLLER2="192.168.7.102"
DATE=`date +%Y-%m-%d_%H_%M_%S`
METHOD=$1
Branch=$2

if test -z $Branch;then
  Branch=develop
fi


function Code_Clone(){
  Git_URL="git@172.20.100.1:linux36/app1.git"
  DIR_NAME=`echo ${Git_URL} |awk -F "/" '{print $2}' | awk -F "." '{print $1}'`
  DATA_DIR="/data/gitdata/linux36"
  Git_Dir="${DATA_DIR}/${DIR_NAME}"
  cd ${DATA_DIR} &&  echo "即将清空上一版本代码并获取当前分支最新代码" && sleep 1 && rm -rf
${DIR_NAME}
  echo "即将开始从分支${Branch} 获取代码" && sleep 1
  git clone -b ${Branch} ${Git_URL}
  echo "分支${Branch} 克隆完成，即将进行代码编译!" && sleep 1
  #cd ${Git_Dir} && mvn clean package
  #echo "代码编译完成，即将开始将IP地址等信息替换为测试环境"
  ######################################################
  sleep 1
  cd ${Git_Dir}
  tar czf ${DIR_NAME}.tar.gz  ./*
}

#将打包好的压缩文件拷贝到k8s 控制端服务器
function Copy_File(){
  echo "压缩文件打包完成，即将拷贝到k8s 控制端服务器${K8S_CONTROLLER1}" && sleep 1
```

```
  scp ${Git_Dir}/${DIR_NAME}.tar.gz root@${K8S_CONTROLLER1}:/opt/k8s-
data/dockerfile/linux36/nginx/
  echo "压缩文件拷贝完成,服务器${K8S_CONTROLLER1}即将开始制作Docker 镜像!" && sleep 1
}

#到控制端执行脚本制作并上传镜像
function Make_Image(){
  echo "开始制作Docker镜像并上传到Harbor服务器" && sleep 1
  ssh root@${K8S_CONTROLLER1} "cd /opt/k8s-data/dockerfile/linux36/nginx  && bash
build-command.sh ${DATE}"
  echo "Docker镜像制作完成并已经上传到harbor服务器" && sleep 1
}

#到控制端更新k8s yaml文件中的镜像版本号,从而保持yaml文件中的镜像版本号和k8s中版本号一致
function Update_k8s_yaml(){
  echo "即将更新k8s yaml文件中镜像版本" && sleep 1
  ssh root@${K8S_CONTROLLER1} "cd /opt/k8s-data/yaml/linux36/nginx  && sed -i 's/image:
harbor.magedu.*/image: harbor.magedu.net\/linux36\/nginx-web1:${DATE}/g' nginx.yaml"
  echo "k8s yaml文件镜像版本更新完成,即将开始更新容器中镜像版本" && sleep 1
}

#到控制端更新k8s中容器的版本号,有两种更新办法，一是指定镜像版本更新，二是apply执行修改过的yaml文件
function Update_k8s_container(){
  #第一种方法
   ssh root@${K8S_CONTROLLER1} "kubectl set image deployment/linux36-nginx-deployment
linux36-nginx-container=harbor.magedu.net/linux36/nginx-web1:${DATE} -n linux36"
  #第二种方法,推荐使用第一种
  #ssh root@${K8S_CONTROLLER1} "cd /opt/k8s-data/yaml/web-test/tomcat-app1 && kubectl
apply -f web-test.yam --record"
  echo "k8s 镜像更新完成" && sleep 1
  echo "当前业务镜像版本: harbor.magedu.net/linux36/nginx-web1:${DATE}"
  #计算脚本累计执行时间，如果不需要的话可以去掉下面四行
  endtime=`date +'%Y-%m-%d %H:%M:%S'`
  start_seconds=$(date --date="$starttime" +%s);
  end_seconds=$(date --date="$endtime" +%s);
  echo "本次业务镜像更新总计耗时："$((end_seconds-start_seconds))"s"
}

#基于k8s 内置版本管理回滚到上一个版本
function rollback_last_version(){
  echo "即将回滚之上一个版本"
  ssh root@${K8S_CONTROLLER1}  "kubectl rollout undo deployment/linux36-nginx-
deployment  -n linux36"
  sleep 1
  echo "已执行回滚至上一个版本"
}

#使用帮助
usage(){
```

```
  echo "部署使用方法为 ${SHELL_DIR}/${SHELL_NAME} deploy "
  echo "回滚到上一版本使用方法为 ${SHELL_DIR}/${SHELL_NAME} rollback_last_version"
}

#主函数
main(){
  case ${METHOD}  in
  deploy)
    Code_Clone;
    Copy_File;
    Make_Image;
    Update_k8s_yaml;
    Update_k8s_container;
  ;;
  rollback_last_version)
    rollback_last_version;
  ;;
  *)
    usage;
  esac;
}

main $1 $2 $3
```

六：容器监控与报警：

容器监控的实现方对比虚拟机或者物理机来说比大的区别，比如容器在k8s环境中可以任意横向扩容与缩容，那么就需要监控服务能够自动对新创建的容器进行监控，当容器删除后又能够及时的从监控服务中删除，而传统的zabbix的监控方式需要在每一个容器中安装启动agent，并且在容器自动发现注册及模板关联方面并没有比较好的实现方式。

6.1：Prometheus：

k8s的早期版本基于组件heapster实现对pod和node节点的监控功能，但是从k8s 1.8版本开始使用metrics API的方式监控，并在1.11版本 正式将heapster替换，后期的k8s监控主要是通过metrics Server提供核心监控指标，比如Node节点的CPU和内存使用率，其他的监控交由另外一个组件Prometheus 完成。

6.1.1：prometheus简介：

https://prometheus.io/docs/ #官方文档

https://github.com/prometheus #github地址

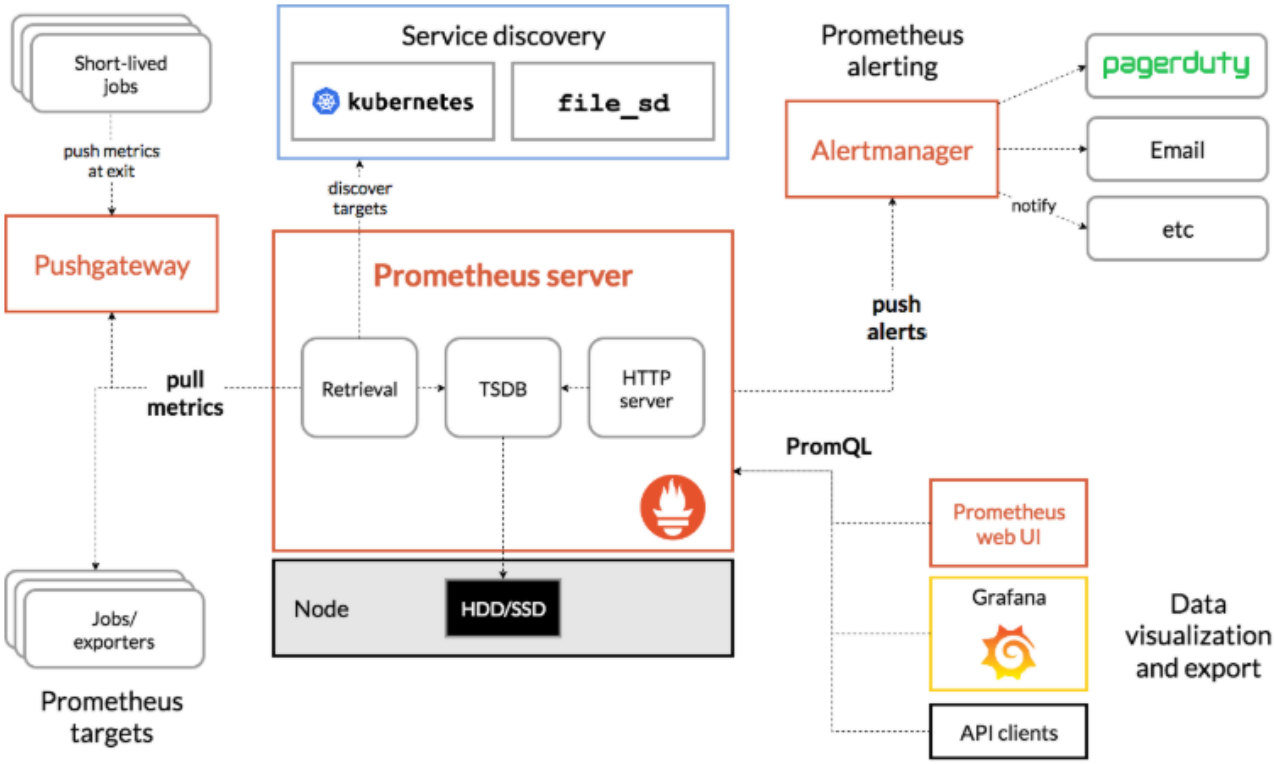
Prometheus是基于go语言开发的一套开源的监控、报警和时间序列数据库的组合，是由SoundCloud公司开发的开源监控系统,Prometheus于2016年加入CNCF（Cloud Native Computing Foundation,云原生计算基金会）,是CNCF继kubernetes 之后毕业的第二个项目，prometheus在容器和微服务领域中得到了广泛的应用，其特点主要如下：

```
使用key-value的多维度格式保存数据
数据不使用MySQL这样的传统数据库，而是使用时序数据库，目前是使用的TSDB
支持第三方dashboard实现更高的图形界面，如grafana(Grafana 2.5.0版本及以上)
功能组件化
不需要依赖存储，数据可以本地保存也可以远程保存
服务自动化发现
强大的数据查询语句功(PromQL,Prometheus Query Language)
```

https://www.aliyun.com/product/hitsdb #阿里云TSDB简介

6.1.2：prometheus系统架构图：

```
prometheus server：主服务，接受外部http请求，收集、存储与查询数据等
prometheus targets：静态收集的目标服务数据
service discovery：动态发现服务
prometheus alerting：报警通知
push gateway：数据收集代理服务器(类似于zabbix proxy)
data visualization and export： 数据可视化与数据导出(访问客户端)
```



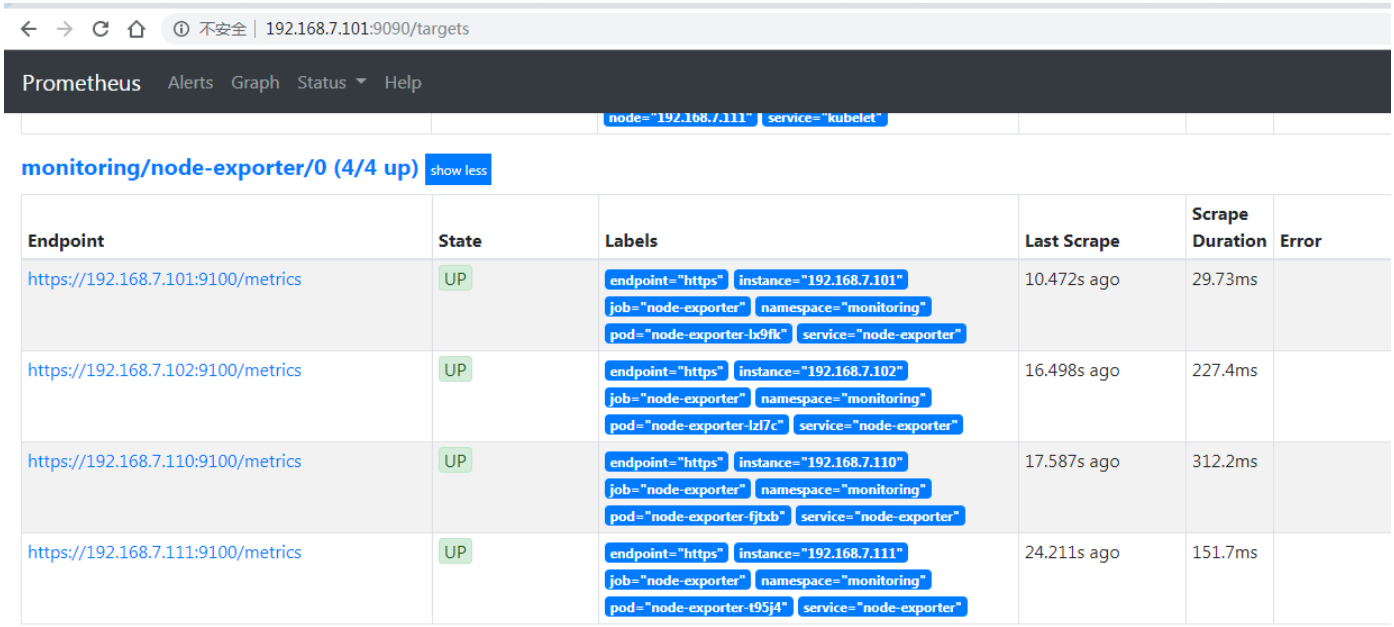
https://songjiayang.gitbooks.io/prometheus/content/promql/summary.html #ProQL简介

6.1.3：prometheus 安装方式：

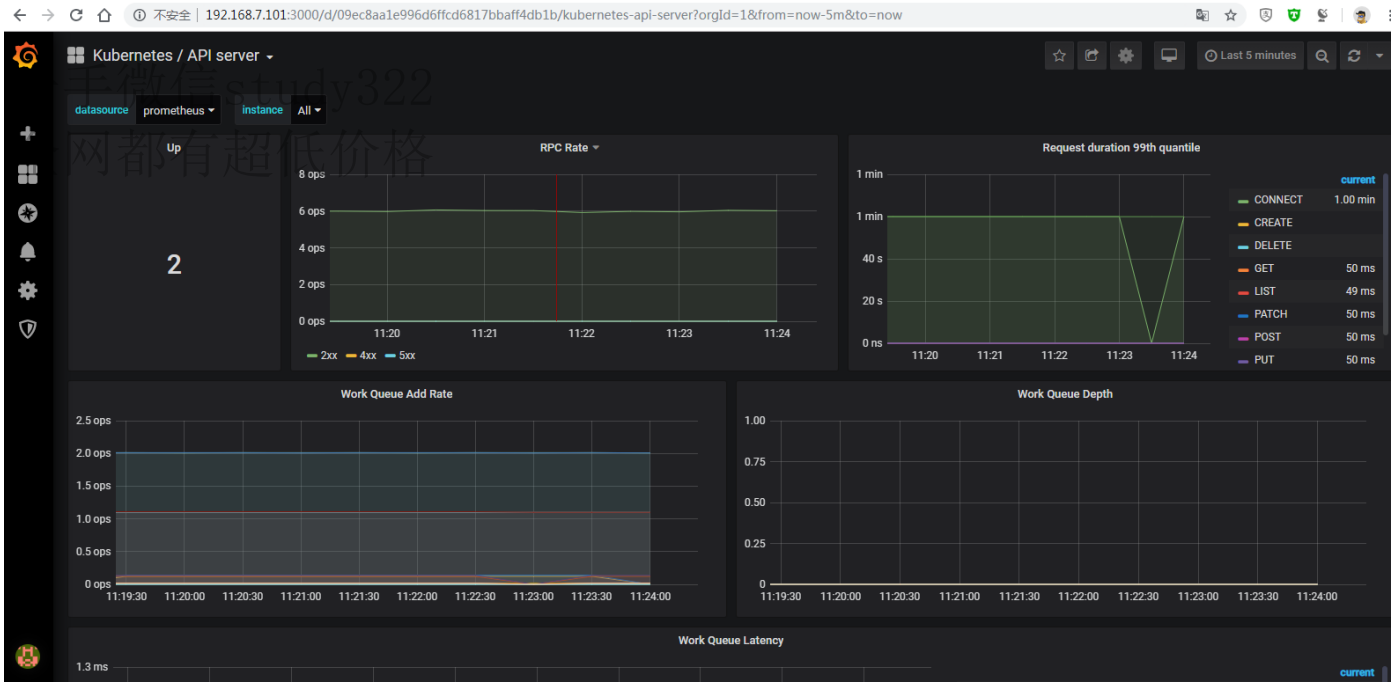
```
https://prometheus.io/download/ #官方二进制下载及安装，prometheus server的监听端口为9090
https://prometheus.io/docs/prometheus/latest/installation/ #docker镜像直接启动
https://github.com/coreos/kube-prometheus #operator部署
~# apt install prometheus #使用apt或者yum安装
```

```
root@k8s-master1:~/kube-prometheus/manifests# pwd
/root/kube-prometheus/manifests
root@k8s-master1:~/kube-prometheus/manifests# kubectl  get pod -n monitoring
NAME                                  READY   STATUS    RESTARTS   AGE
alertmanager-main-0                   2/2     Running   0          13m
alertmanager-main-1                   2/2     Running   0          13m
alertmanager-main-2                   2/2     Running   0          13m
grafana-598f68c777-ttbnr              1/1     Running   0          13m
kube-state-metrics-5c9bd5465b-jd8zw   3/3     Running   0          13m
node-exporter-fjtxb                   2/2     Running   0          13m
node-exporter-lx9fk                   2/2     Running   0          13m
node-exporter-lzl7c                   2/2     Running   0          13m
node-exporter-t95j4                   2/2     Running   0          13m
prometheus-adapter-74fc6495d7-w7h2k   1/1     Running   0          13m
prometheus-k8s-0                      3/3     Running   1          12m
prometheus-k8s-1                      3/3     Running   1          12m
prometheus-operator-658766d58-vb6xm   1/1     Running   0          13m
root@k8s-master1:~/kube-prometheus/manifests#
```

```
#kubectl  port-forward --help
# kubectl --namespace monitoring port-forward --address 0.0.0.0 svc/prometheus-k8s 9090:9090
```




```
#kubectl --namespace monitoring port-forward --address 0.0.0.0 svc/grafana 3000:3000
```



基于NodePort暴露服务：

grafana：

```
# pwd
/root/kube-prometheus/manifests

# vim grafana-service.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: grafana
  name: grafana
  namespace: monitoring
spec:
  type: NodePort
  ports:
  - name: http
    port: 3000
    targetPort: http
    nodePort: 33000
  selector:
    app: grafana
# kubectl apply  -f  grafana-service.yaml
```

prometheus：

```
# pwd
/root/kube-prometheus/manifests
```

```
# vim prometheus-service.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    prometheus: k8s
  name: prometheus-k8s
  namespace: monitoring
spec:
  type: NodePort
  ports:
  - name: web
    port: 9090
    targetPort: web
    nodePort: 39090
  selector:
    app: prometheus
    prometheus: k8s
  sessionAffinity: ClientIP
```

6.1.3.1：二进制方式安装：

```
# pwd
/usr/local/src

# tar xvf prometheus-2.24.1.linux-amd64.tar.gz
# ln -sv /apps/prometheus-2.24.1.linux-amd64 /apps/prometheus
'/apps/prometheus' -> '/apps/prometheus-2.24.1.linux-amd64'
# cd /apps/prometheus
# ll
prometheus.yml #配置文件
prometheus #prometheus服务可执行程序
promtool  #测试工具，用于检测配置prometheus配置文件、检测metrics数据等
  ./promtool check config prometheus.yml
    Checking prometheus.yml
    SUCCESS: 0 rule files found
```

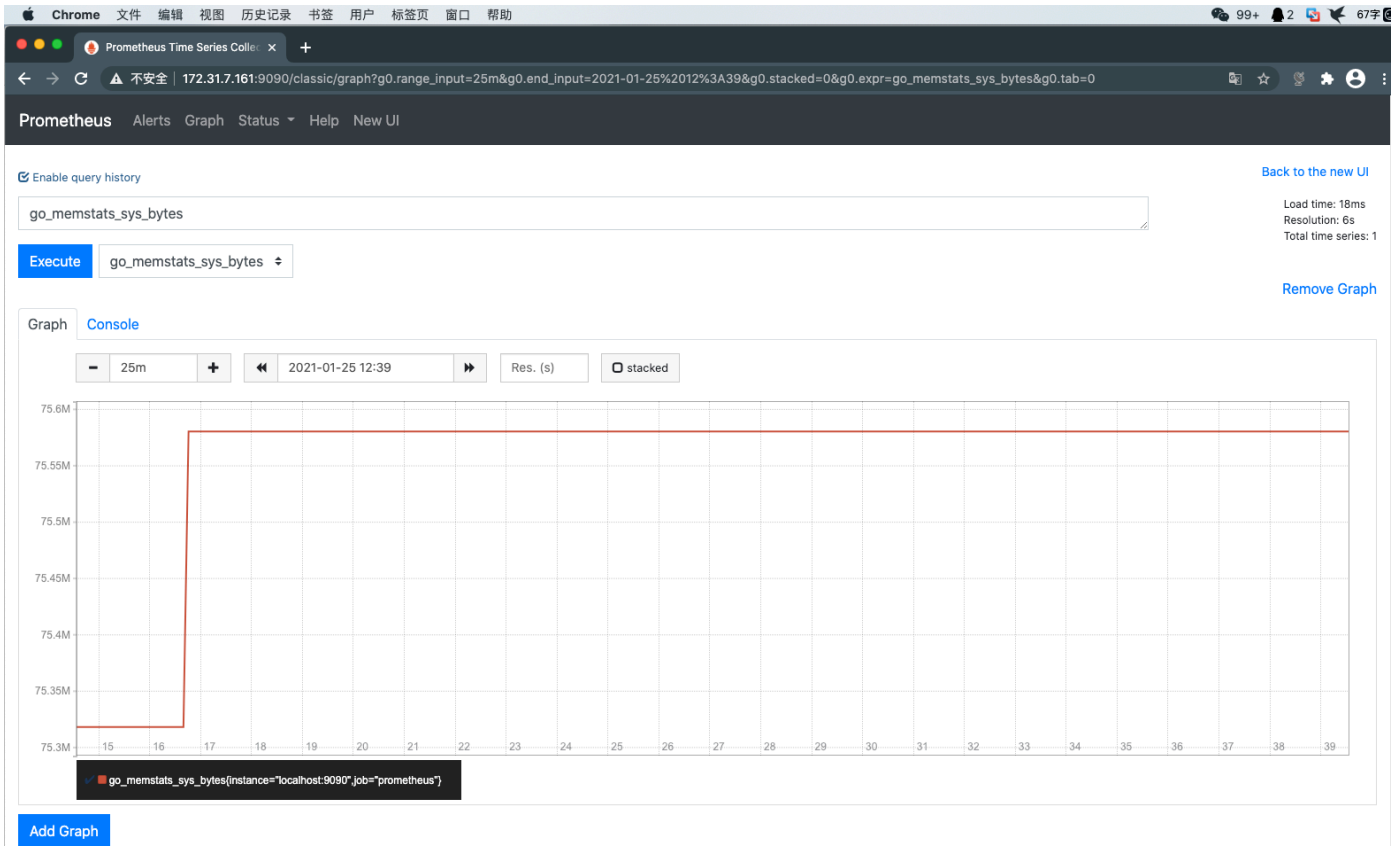
6.1.3.2：创建prometheus启动脚本：

```
# vim /etc/systemd/system/prometheus.service
[Unit]
Description=Prometheus Server
Documentation=https://prometheus.io/docs/introduction/overview/
After=network.target

[Service]
Restart=on-failure
WorkingDirectory=/apps/prometheus/
ExecStart=/apps/prometheus/prometheus    --config.file=/apps/prometheus/prometheus.yml

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

6.1.3.3：启动prometheus服务：

```
# systemctl   daemon-reload &&  systemctl  restart prometheus && systemctl  enable prometheus
```

6.1.3.4：访问prometheus web界面：



6.1.4：node exporter：

各node节点安装node_exporter，用于收集各k8s node节点上的监控指标数据，默认监听端口为9100

6.1.4.1：二进制方式安装node exporter：

```
# mkdir  /apps
# cd /apps/

# tar xvf node_exporter-1.0.1.linux-amd64.tar.gz
# ln -sv /apps/node_exporter-1.0.1.linux-amd64 /apps/node_exporter
# cd /apps/node_exporter
node_exporter #可执行程序
```

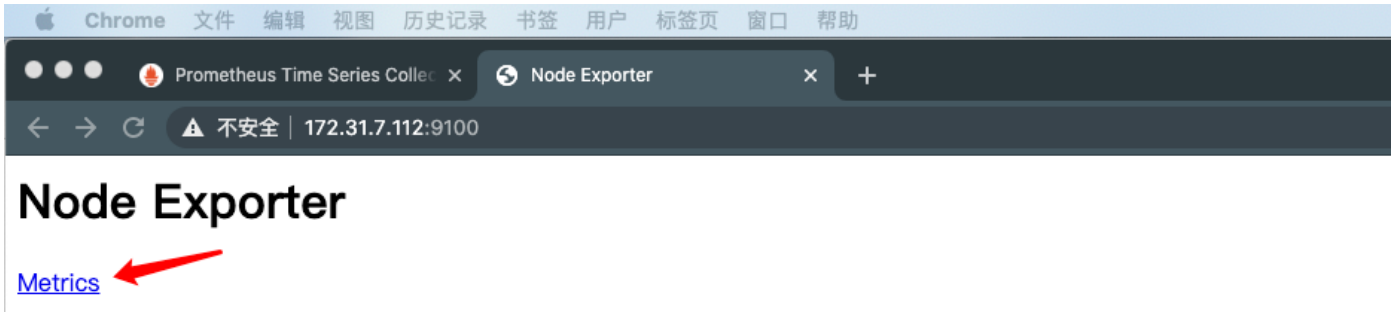
6.1.4.2：创建node exporter启动脚本：

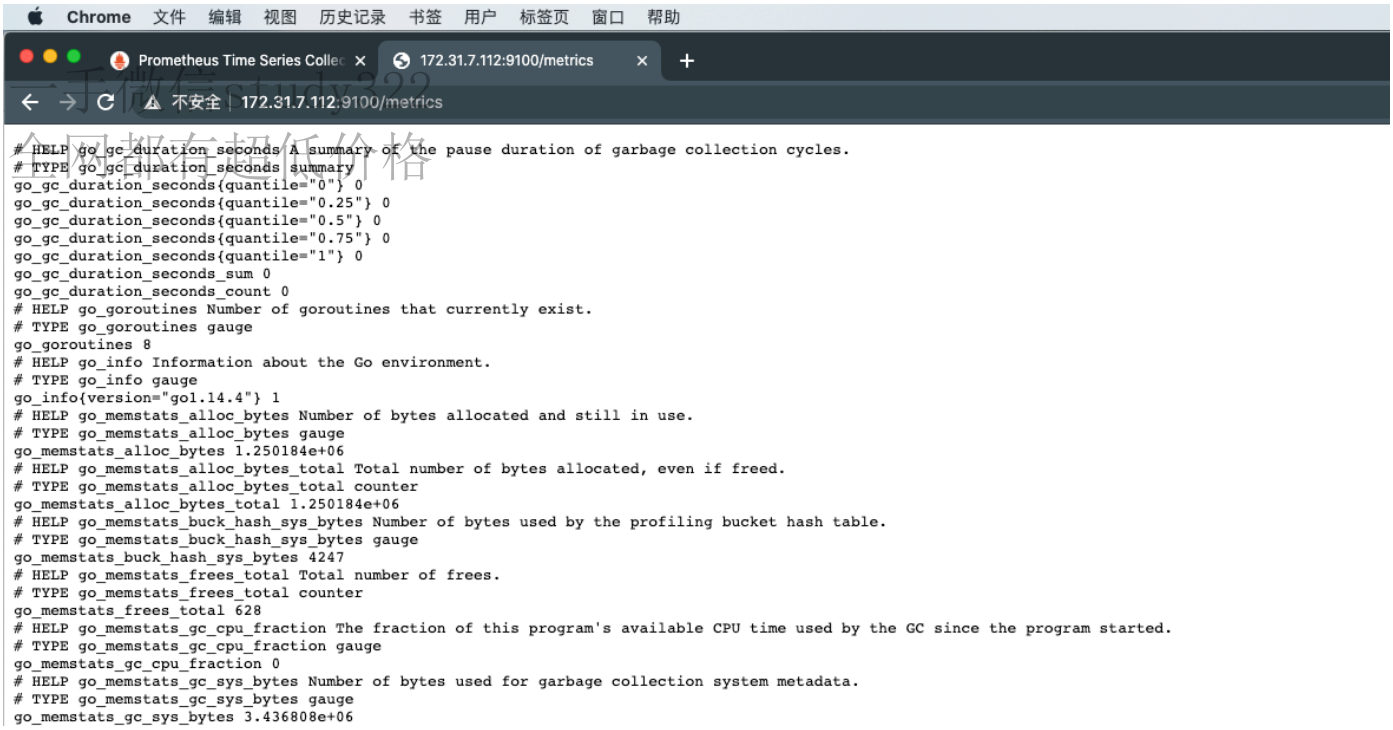
```
# vim   /etc/systemd/system/node-exporter.service
[Unit]
Description=Prometheus Node Exporter
After=network.target

[Service]
ExecStart=/apps/node_exporter/node_exporter

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

6.1.4.3：启动node exporter服务：

```
# systemctl  daemon-reload  && systemctl  restart  node-exporter && systemctl  enable
node-exporter.service
```

6.1.4.4：访问node exporter web界面：





6.1.5：prometheus采集node 指标数据：

配置prometheus通过node exporter采集 监控指标数据

6.1.5.1：prometheus默认配置文件：

```
# my global config
global: #
  scrape_interval:  15s # Set the scrape interval to every 15 seconds. Default is every
1 minute. #数据收集间隔时间，如果不配置默认为一分钟
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1
minute. #规则扫描间隔时间，如果不配置默认为一分钟
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting: #报警通知配置
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global
'evaluation_interval'.
rule_files: #规则配置
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs: #数据采集目标配置
```

```
  # 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: ['localhost:9090']
```

6.1.5.2：prometheus收集node数据：

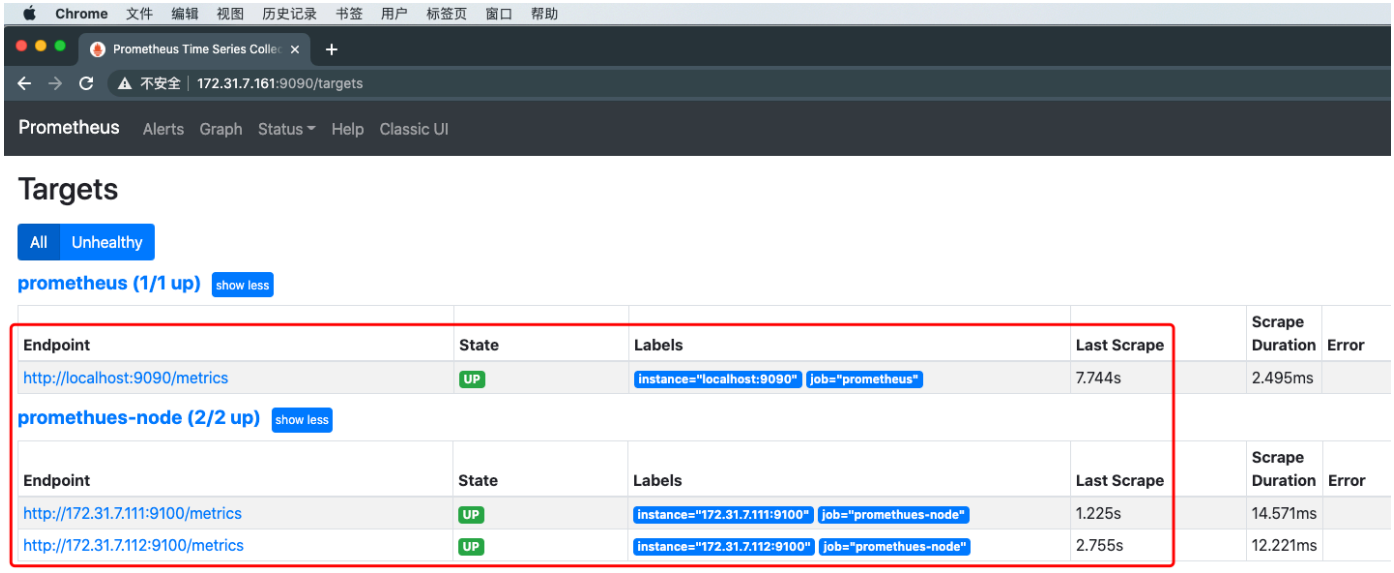
```
root@prometheus-server:~# cat  /apps/prometheus/prometheus.yml
# my global config
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is
every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1
minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global
'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "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: ['localhost:9090']
  - job_name: 'promethues-node'
    static_configs:
      - targets: ['172.31.7.111:9100','172.31.7.112:9100']
```

```
root@prometheus-server:/apps/prometheus# systemctl  restart prometheus.service
```

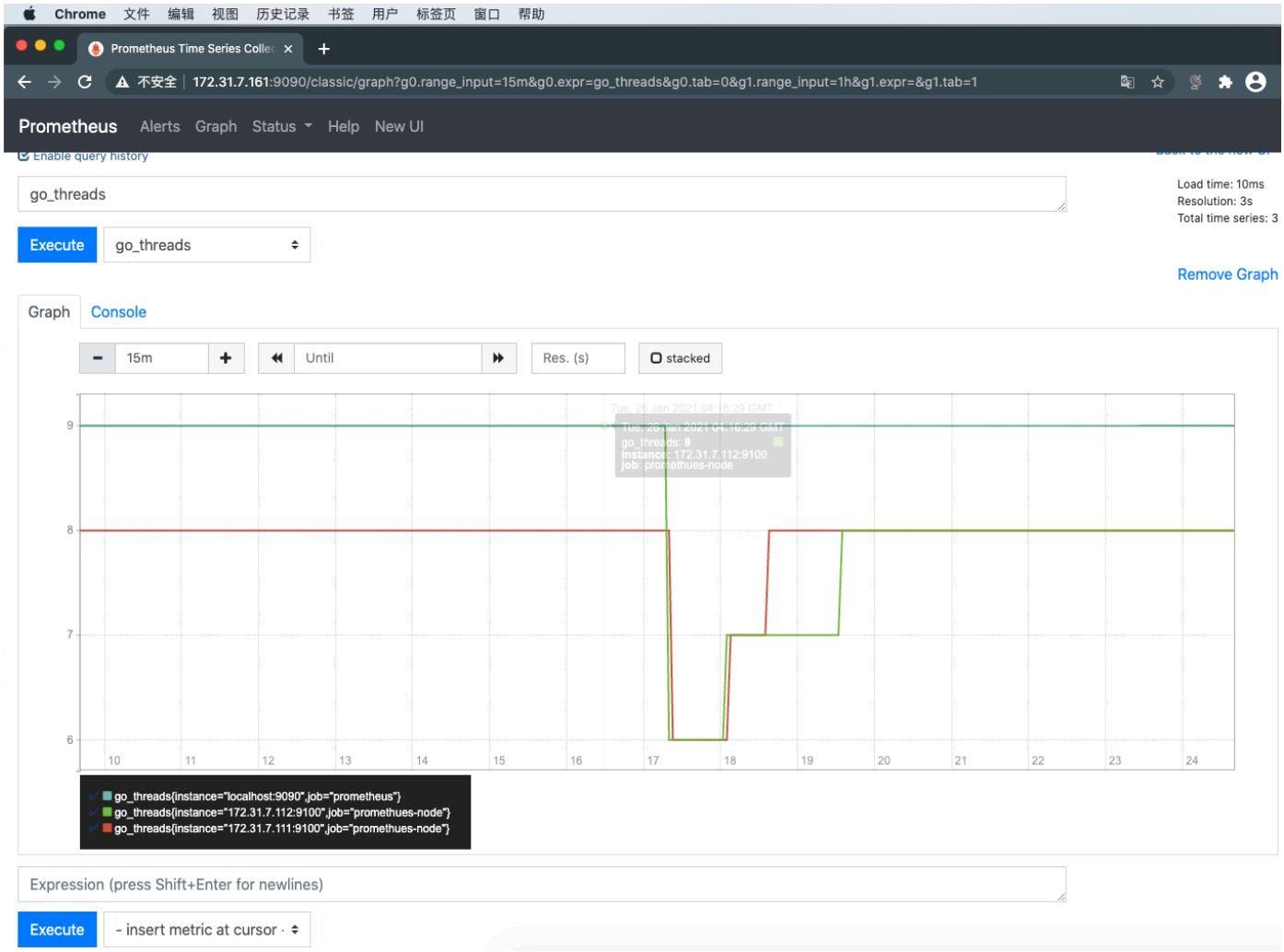
6.1.5.2：重启prometheus服务：

```
# systemctl  restart prometheus
```

6.1.5.3：prometheus验证node节点状态：



6.1.5.4：prometheus验证node节点监控数据：



6.1.6：PromQL简介：

Prometheus提供一个函数式的表达式语言PromQL (Prometheus Query Language)，可以使用户实时地查找和聚合时间序列数据，表达式计算结果可以在图表中展示，也可以在Prometheus表达式浏览器中以表格形式展示，或者作为数据源, 以HTTP API的方式提供给外部系统使用

6.1.6.1：PromQL基本查询：

```
node_memory_MemTotal_bytes #查询node节点总内存大小
node_memory_MemFree_bytes #查询node节点剩余可用内存
node_memory_MemTotal_bytes{instance="172.31.7.111:9100"} #查询指定节点的总内存
node_memory_MemFree_bytes{instance="172.31.7.111:9100"} #查询指定节点的可用内存

node_disk_io_time_seconds_total{device="sda"} #查询指定磁盘的每秒磁盘io
node_filesystem_free_bytes{device="/dev/sda1",fstype="xfs",mountpoint="/"} #查看指定磁盘的磁盘剩余空间

# HELP node_load1 1m load average. #CPU负载
# TYPE node_load1 gauge
node_load1 0.1
# HELP node_load15 15m load average.
# TYPE node_load15 gauge
node_load15 0.17
```

```
# HELP node_load5 5m load average.
# TYPE node_load5 gauge
node_load5 0.13
```

6.1.6.2：PromQL数据类型：

瞬时向量(instant vector):是一组时间序列，每个时间序列包含单个数据样本，比如node_memory_MemTotal_bytes查询当前剩余内存就是一个瞬时向量，该表达式的返回值中只会包含该时间序列中的最新的一个样本值，而相应的这样的表达式称之为瞬时向量表达式。
范围向量(range vector):是指在任何一个时间范围内，抓取的所有度量指标数据.比如最近一天的网卡流量趋势图。
标量(scalar)：是一个浮点数类型的数据值，使用node_load1获取到时一个瞬时向量，但是可用使用内置函数scalar()将瞬时向量转换为标量。
字符串(string):字符串类型的数据，目前使用较少

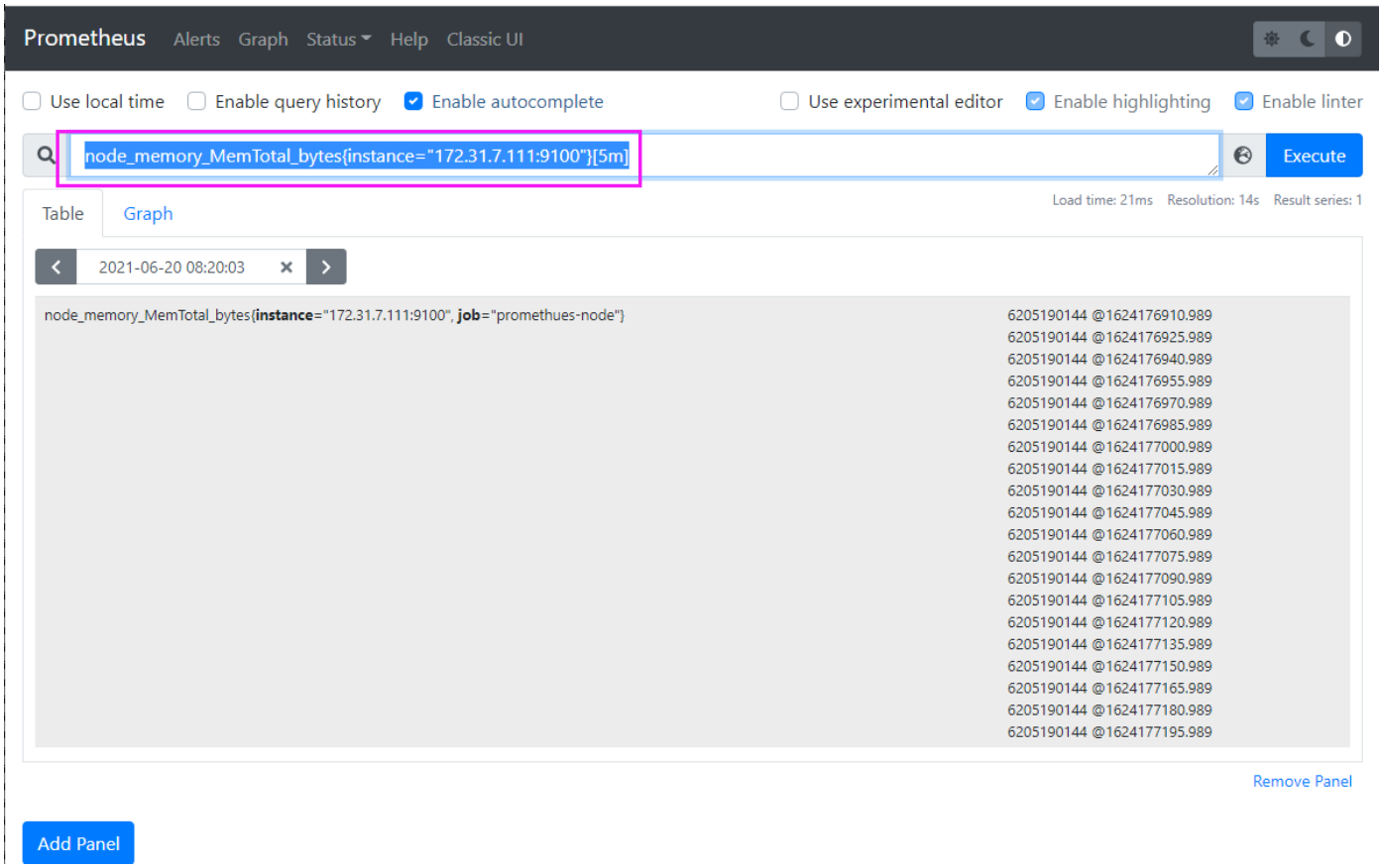
6.1.6.3：PromQL匹配器：

= :选择与提供的字符串完全相同的标签。
!= :选择与提供的字符串不相同的标签。
=~ :选择正则表达式与提供的字符串（或子字符串）相匹配的标签。
!~ :选择正则表达式与提供的字符串（或子字符串）不匹配的标签。

```
#查询格式<metric name>{<label name>=<label value>, ...}
node_load1{instance="172.31.7.111:9100"}
node_load1{job="promethues-node"}

node_load1{job="promethues-node",instance="172.31.7.111:9100"}
node_load1{job="promethues-node",instance!="172.31.7.111:9100"}
```

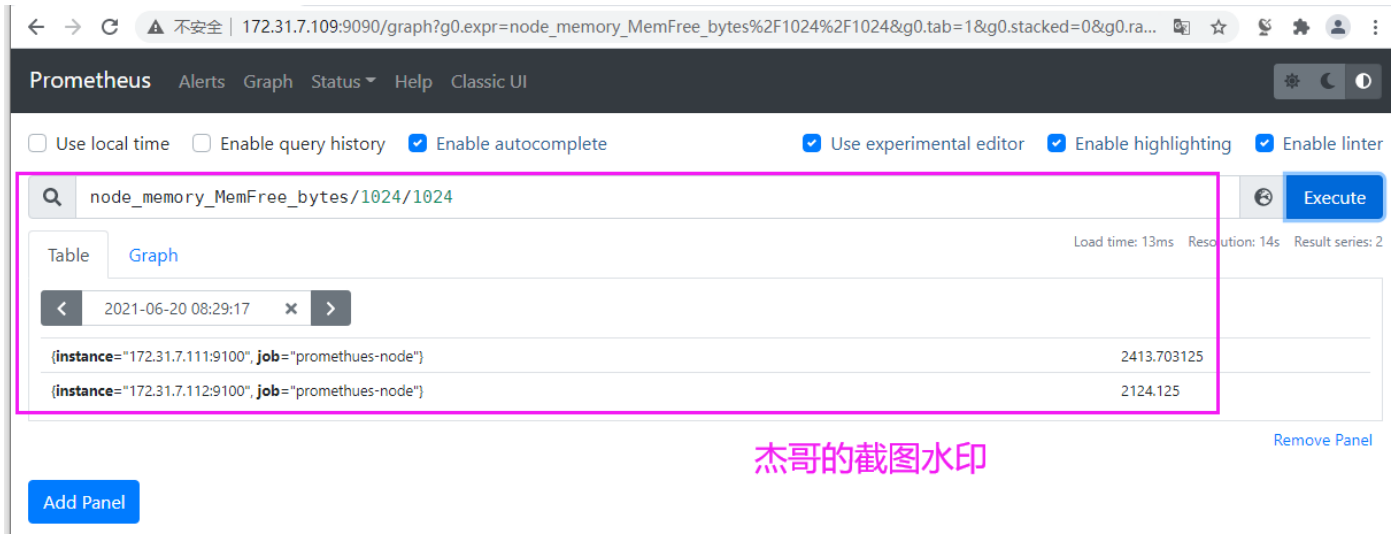
6.1.6.4：PromQL时间范围：

```
s - 秒
m - 分钟
h - 小时
d - 天
w - 周
y - 年

node_memory_MemTotal_bytes{} # 瞬时向量表达式，选择当前最新的数据
node_memory_MemTotal_bytes{}[5m] # 区间向量表达式，选择以当前时间为基准，5分钟内的数据
node_memory_MemTotal_bytes{instance="172.31.7.111:9100"}[5m]
```



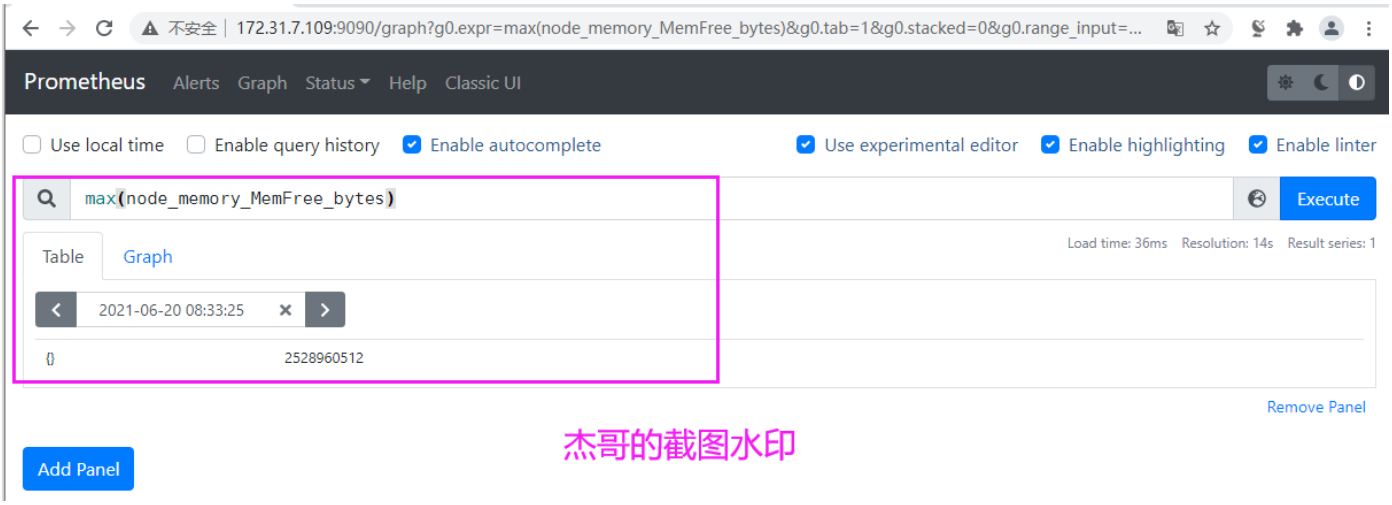
6.1.6.5：PromQL运算符：

```
+ 加法
- 减法
* 乘法
/ 除法
% 模
^ 幂等

node_memory_MemFree_bytes/1024/1024 #将内存进行单位转换
node_disk_read_bytes_total{device="sda"} + node_disk_written_bytes_total{device="sda"}
#计算磁盘每秒读写数据量
```



6.1.6.6：PromQL聚合运算：

```
sum （求和）
min （最小值）
max （最大值）
avg （平均值）
stddev （标准差）
stdvar （标准差异）
count （计数）
count_values （对 value 进行计数）
bottomk （样本值最小的 k 个元素）
topk （样本值最大的k个元素）
quantile （分布统计）

max(node_memory_MemFree_bytes) #获5分钟内的最大值
sum(http_requests_total) #计算http_requests_total最近的请求总量
```



6.2：Grafana：

https://grafana.com/docs/ #官方安装文档

调用prometheus的数据，进行更专业的可视化

6.2.1：安装grafana：

安装版本： v7.3.7

```
# pwd
/usr/local/src

# apt-get install -y adduser libfontconfig1
# dpkg -i grafana_<VERSION>_amd64.deb
# apt --fix-broken install -y
```

6.2.2：配置文件：

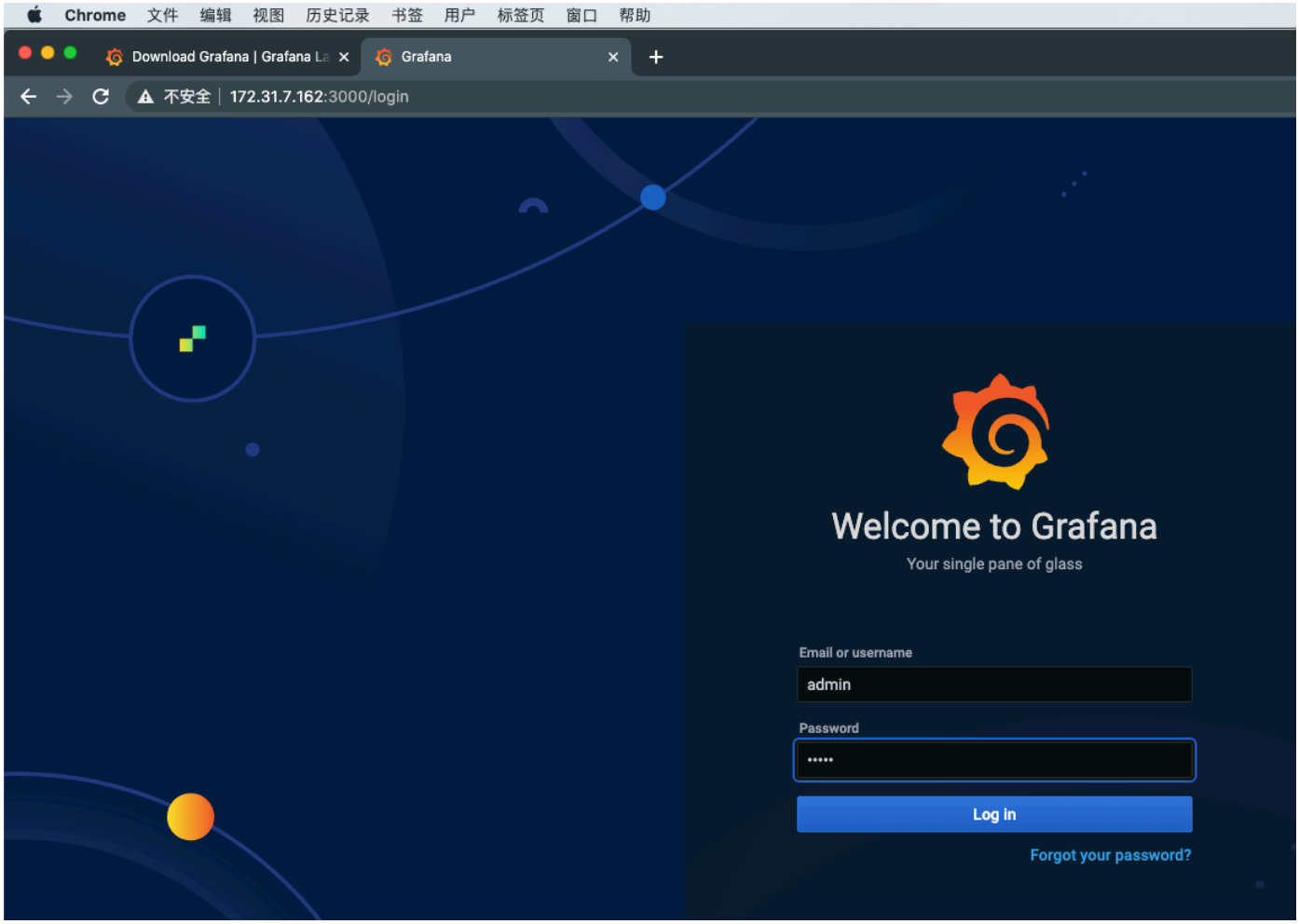
```
# vim /etc/grafana/grafana.ini
[server]
# Protocol (http, https, socket)
protocol = http

# The ip address to bind to, empty will bind to all interfaces
http_addr = 0.0.0.0

# The http port  to use
http_port = 3000
```

6.2.3：启动grafana：

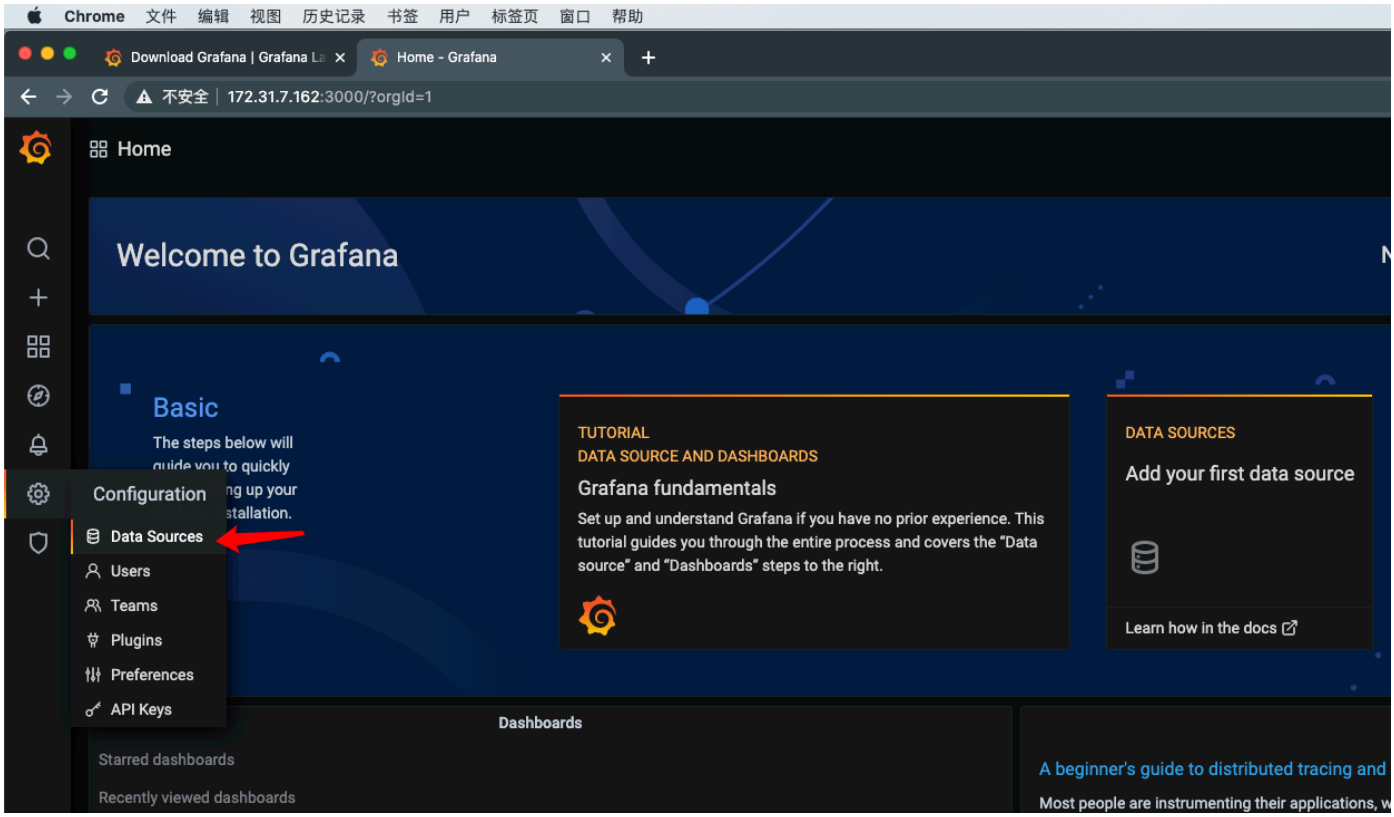
```
# systemctl  start grafana-server.service
# systemctl  enable grafana-server.service
```

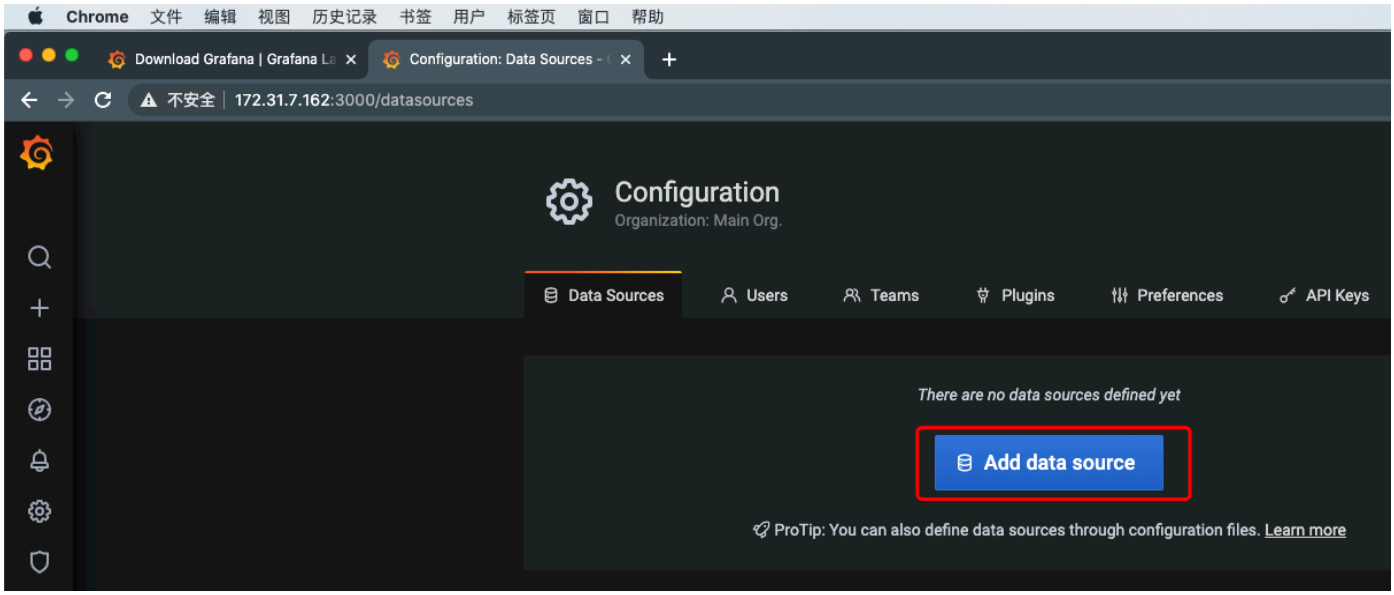
6.2.4：grafana web界面：

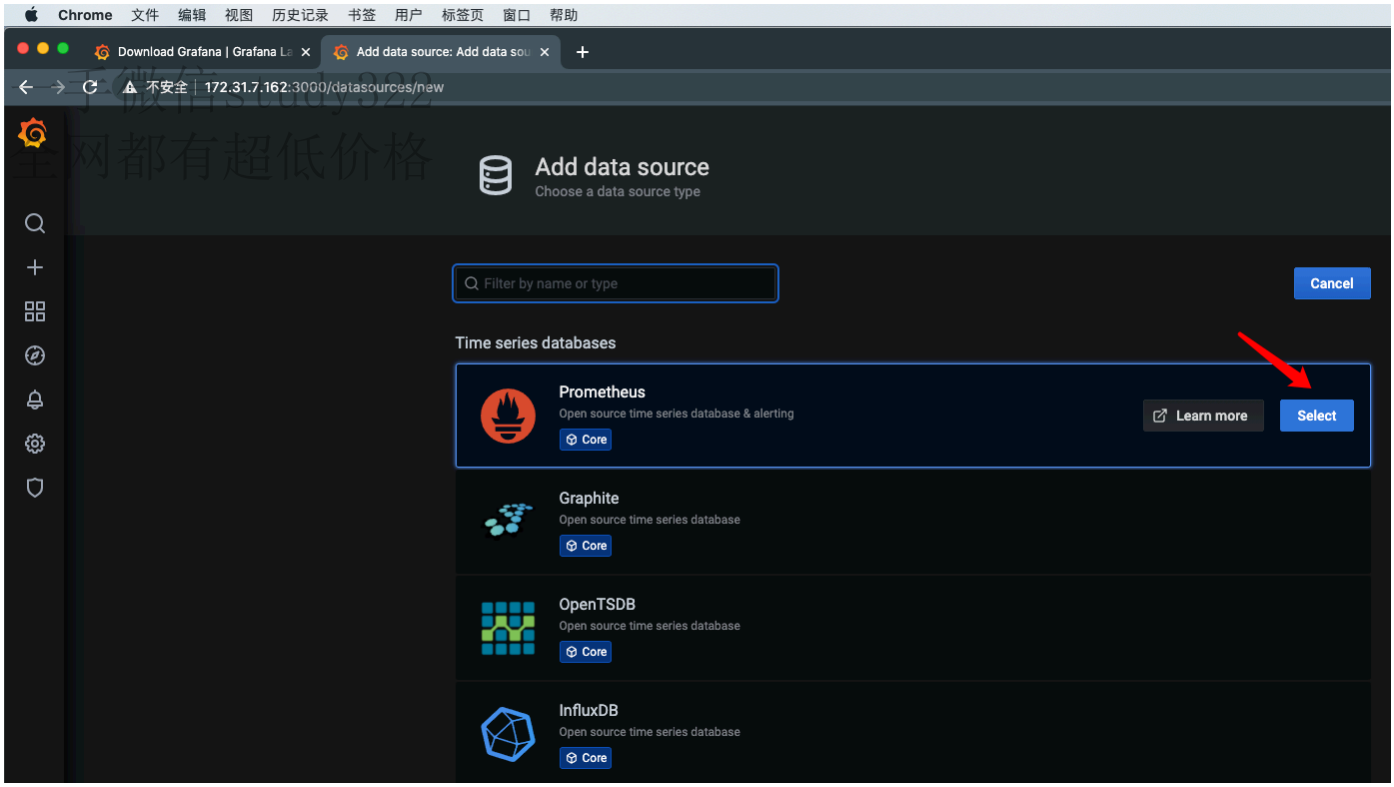
6.2.4.1：登录界面：

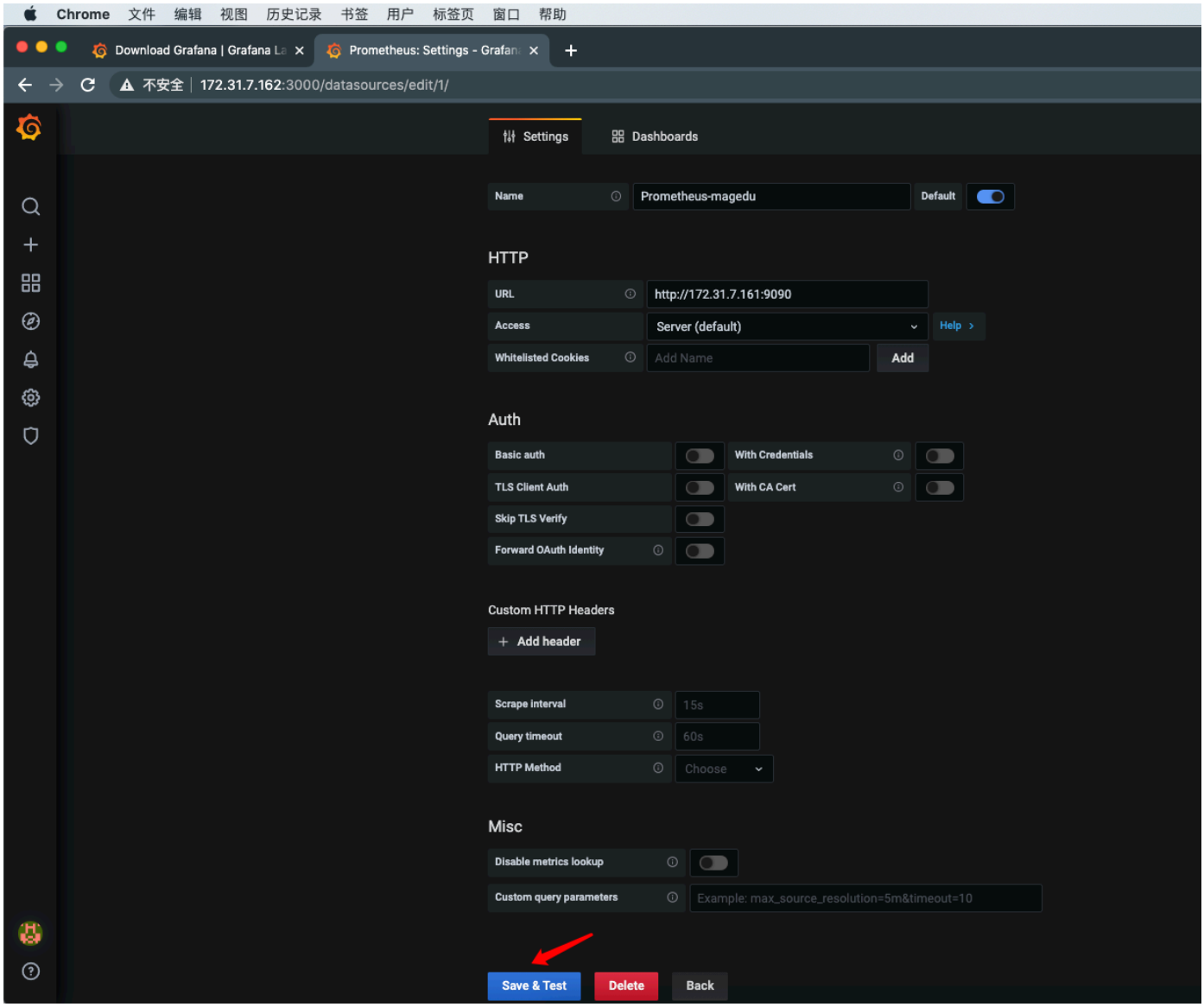


6.2.4.2：添加prometheus数据源：






Chrome 文件 编辑 视图 历史记录 书签 用户 标签页 窗口 帮助
Download Grafana | Grafana La
Add data source: Add data sou
不安全 | 172.31.7.162:3000/datasources/new
Add data source
Choose a data source type
Filter by name or type
Cancel
Time series databases
Prometheus
Open source time series database & alerting
Core
Learn more
Select
Graphite
Open source time series database
Core
OpenTSDB
Open source time series database
Core
InfluxDB
Open source time series database
Core

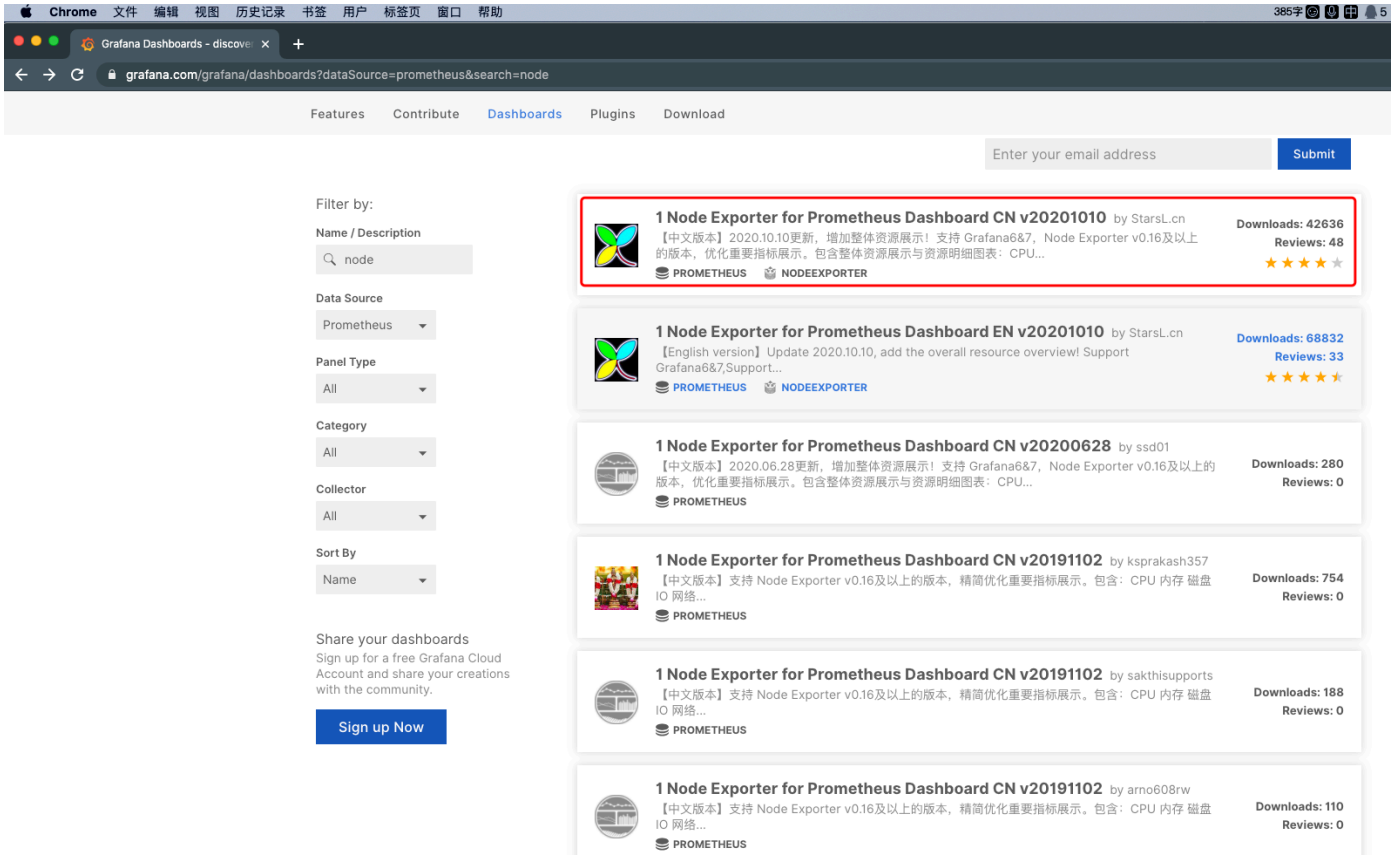


6.4.3：import模板：

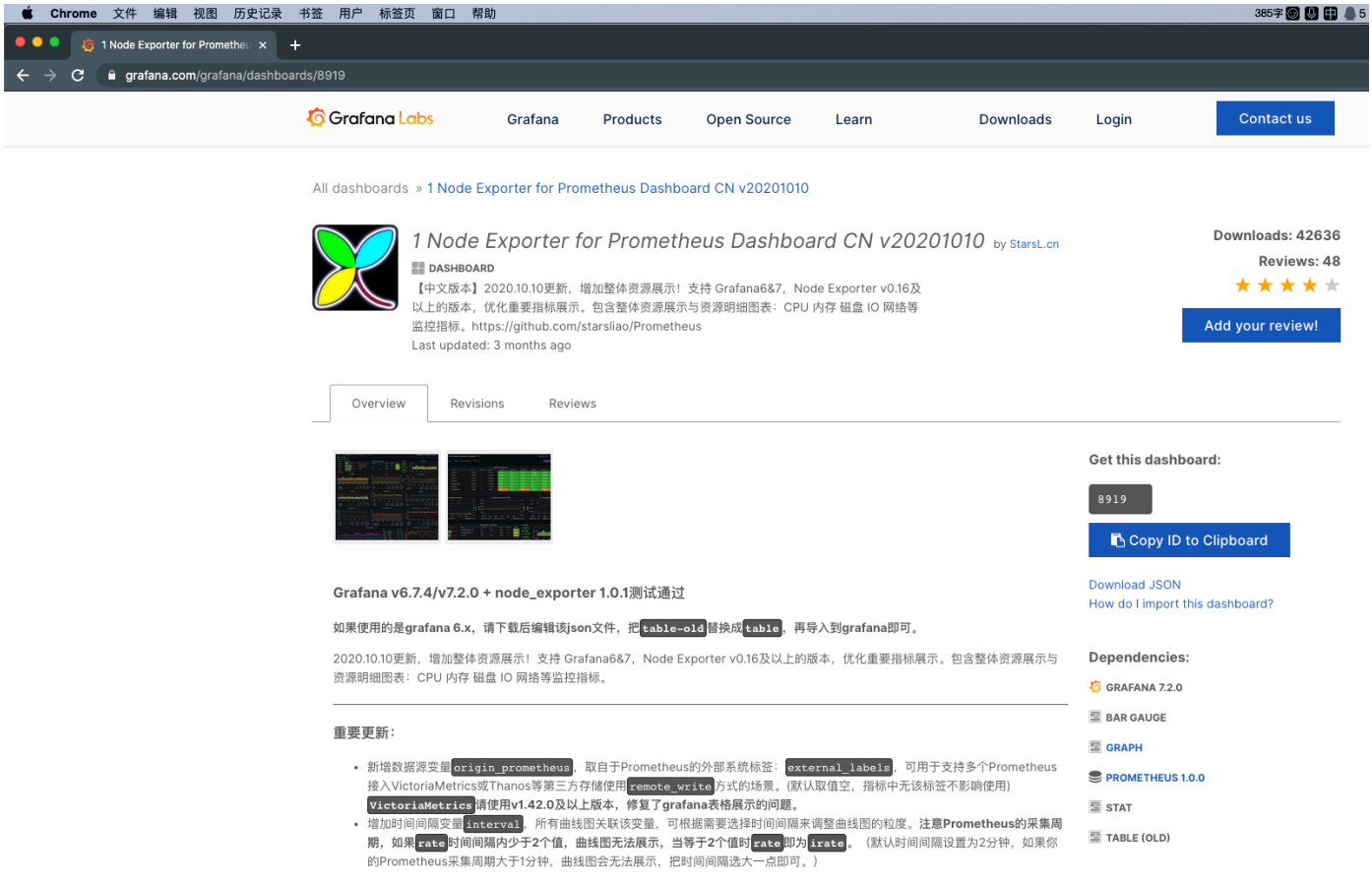
导入模板查看web

6.4.3.1：模板下载地址：

模板可以在左侧栏匹配或搜索相关名称

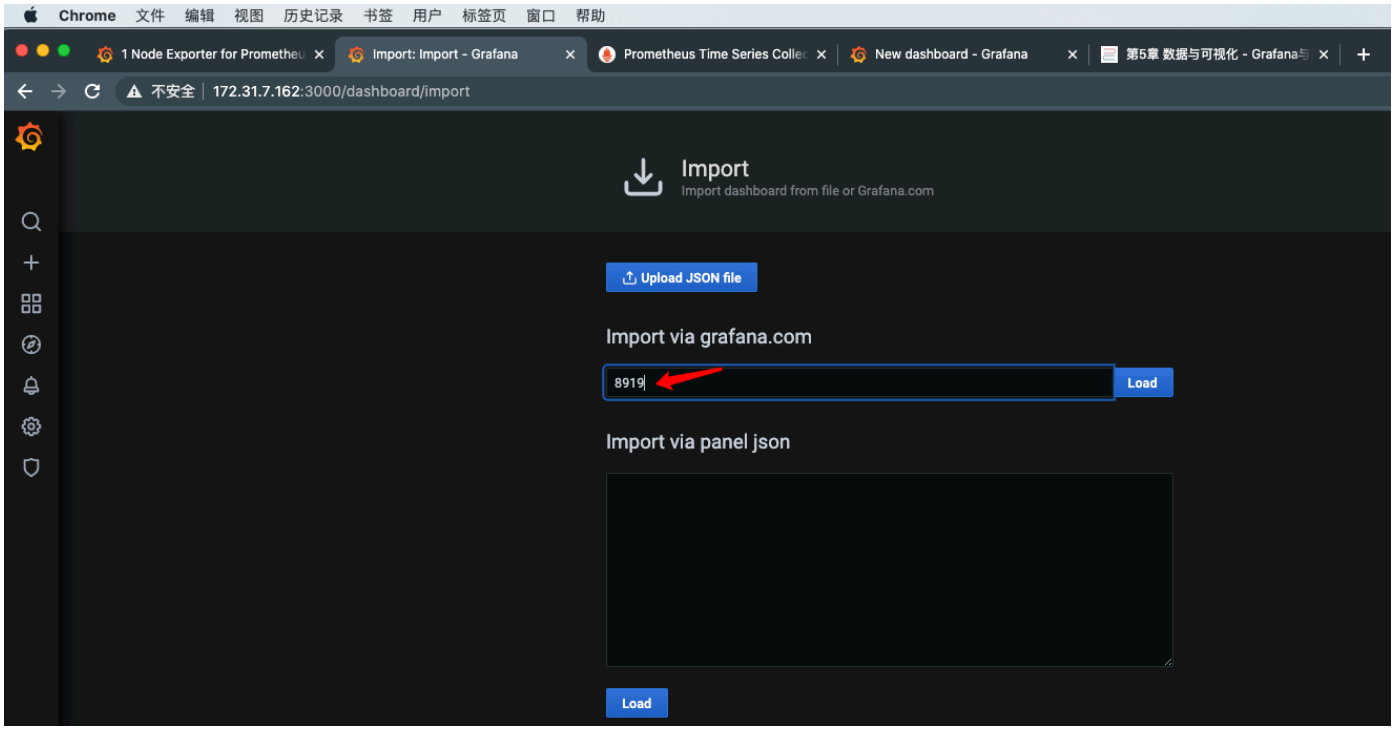

Chrome 文件 编辑 视图 历史记录 书签 用户 标签页 窗口 帮助
Grafana Dashboards - discover
grafana.com/grafana/dashboards?dataSource=prometheus&search=node
Features Contribute Dashboards Plugins Download
Enter your email address
Submit
Filter by:
Name / Description
node
Data Source
Prometheus
Panel Type
All
Category
All
Collector
All
Sort By
Name
Share your dashboards
Sign up for a free Grafana Cloud Account and share your creations with the community.
Sign up Now
1 Node Exporter for Prometheus Dashboard CN v20201010 by StarsL.cn
【中文版本】2020.10.10更新，增加整体资源展示！支持 Grafana6&7，Node Exporter v0.16及以上的版本，优化重要指标展示。包含整体资源展示与资源明细图表：CPU...
PROMETHEUS NODEEXPORTER
Downloads: 42636
Reviews: 48
1 Node Exporter for Prometheus Dashboard EN v20201010 by StarsL.cn
【English version】Update 2020.10.10, add the overall resource overview! Support Grafana6&7,Support...
PROMETHEUS NODEEXPORTER
Downloads: 68832
Reviews: 33
1 Node Exporter for Prometheus Dashboard CN v20200628 by ssd01
【中文版本】2020.06.28更新，增加整体资源展示！支持 Grafana6&7，Node Exporter v0.16及以上的版本，优化重要指标展示。包含整体资源展示与资源明细图表：CPU...
PROMETHEUS
Downloads: 280
Reviews: 0
1 Node Exporter for Prometheus Dashboard CN v20191102 by ksprakash357
【中文版本】支持 Node Exporter v0.16及以上的版本，精简优化重要指标展示。包含：CPU 内存 磁盘 IO 网络...
PROMETHEUS
Downloads: 754
Reviews: 0
1 Node Exporter for Prometheus Dashboard CN v20191102 by sakthisupports
【中文版本】支持 Node Exporter v0.16及以上的版本，精简优化重要指标展示。包含：CPU 内存 磁盘 IO 网络...
PROMETHEUS
Downloads: 188
Reviews: 0
1 Node Exporter for Prometheus Dashboard CN v20191102 by arno608rw
【中文版本】支持 Node Exporter v0.16及以上的版本，精简优化重要指标展示。包含：CPU 内存 磁盘 IO 网络...
PROMETHEUS
Downloads: 110
Reviews: 0

6.4.3.2：模板详细信息：

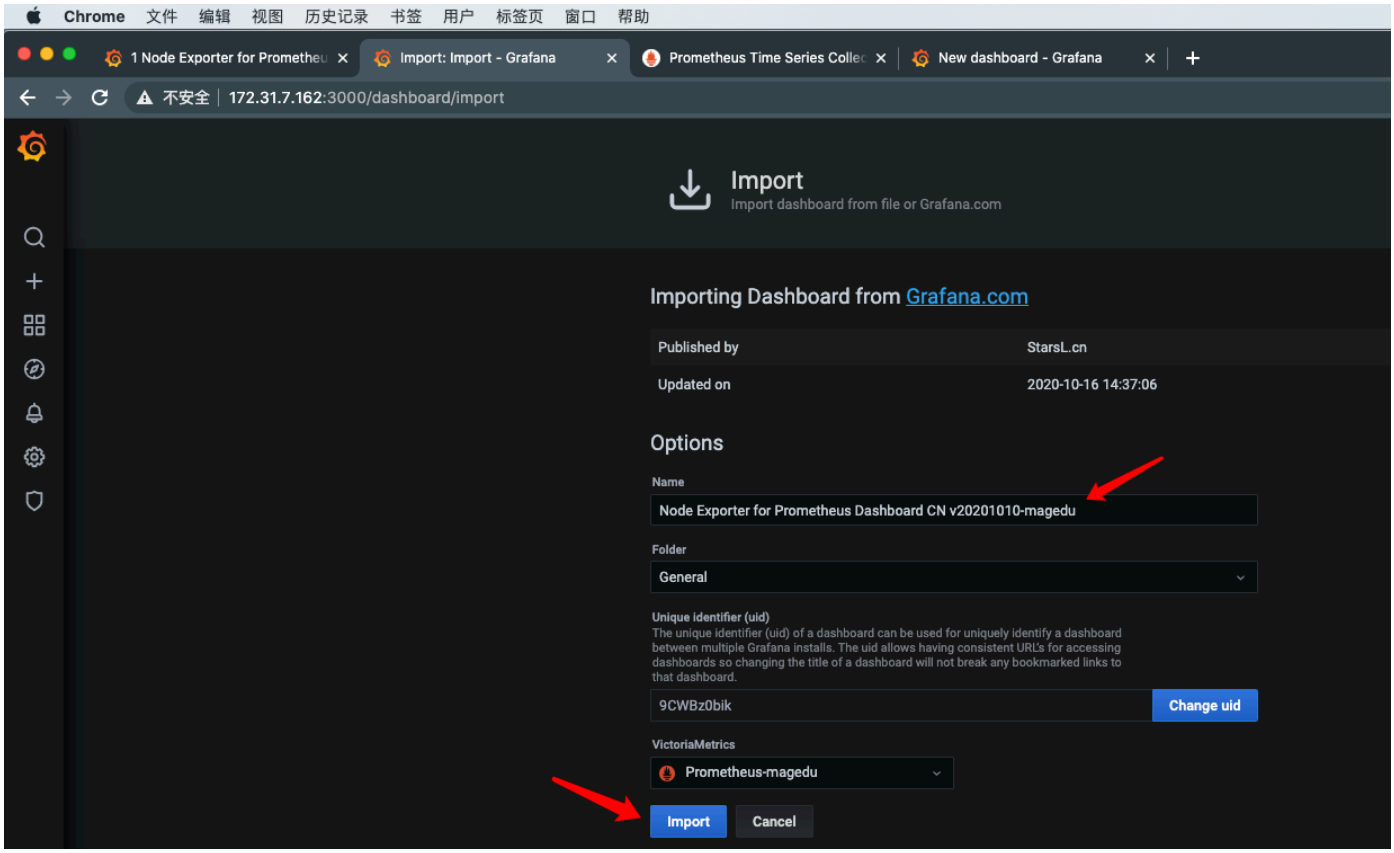

Chrome 文件 编辑 视图 历史记录 书签 用户 标签页 窗口 帮助
1 Node Exporter for Prometheus
grafana.com/grafana/dashboards/8919
Grafana Labs Grafana Products Open Source Learn Downloads Login Contact us
All dashboards » 1 Node Exporter for Prometheus Dashboard CN v20201010
1 Node Exporter for Prometheus Dashboard CN v20201010 by StarsL.cn
DASHBOARD
【中文版本】2020.10.10更新，增加整体资源展示！支持 Grafana6&7，Node Exporter v0.16及以上的版本，优化重要指标展示。包含整体资源展示与资源明细图表：CPU 内存 磁盘 IO 网络等监控指标。https://github.com/starsliao/Prometheus
Last updated: 3 months ago
Downloads: 42636
Reviews: 48
Add your review!
Overview Revisions Reviews
Grafana v6.7.4/v7.2.0 + node_exporter 1.0.1测试通过
如果使用的是grafana 6.x，请下载后编辑该json文件，把table-old替换成table，再导入到grafana即可。
2020.10.10更新，增加整体资源展示！支持 Grafana6&7，Node Exporter v0.16及以上的版本，优化重要指标展示。包含整体资源展示与资源明细图表：CPU 内存 磁盘 IO 网络等监控指标。
重要更新：
新增数据源变量origin_prometheus，取自于Prometheus的外部系统标签：external_labels，可用于支持多个Prometheus接入VictoriaMetrics或Thanos等第三方存储使用remote_write方式的场景。(默认取值空，指标中无该标签不影响使用)
VictoriaMetrics请使用v1.42.0及以上版本，修复了grafana表格展示的问题。
增加时间间隔变量interval，所有曲线图关联该变量，可根据需要选择时间间隔来调整曲线图的粒度。注意Prometheus的采集周期，如果rate时间间隔内少于2个值，曲线图无法展示，当等于2个值时rate即为irate。（默认时间间隔设置为2分钟，如果你的Prometheus采集周期大于1分钟，曲线图会无法展示，把时间间隔选大一点即可。）
Get this dashboard:
8919
Copy ID to Clipboard
Download JSON
How do I import this dashboard?
Dependencies:
GRAFANA 7.2.0
BAR GAUGE
GRAPH
PROMETHEUS 1.0.0
STAT
TABLE (OLD)

6.4.3.3：下载模板：

6.4.3.4：通过模板ID导入：



6.4.3.5：确认模板信息：



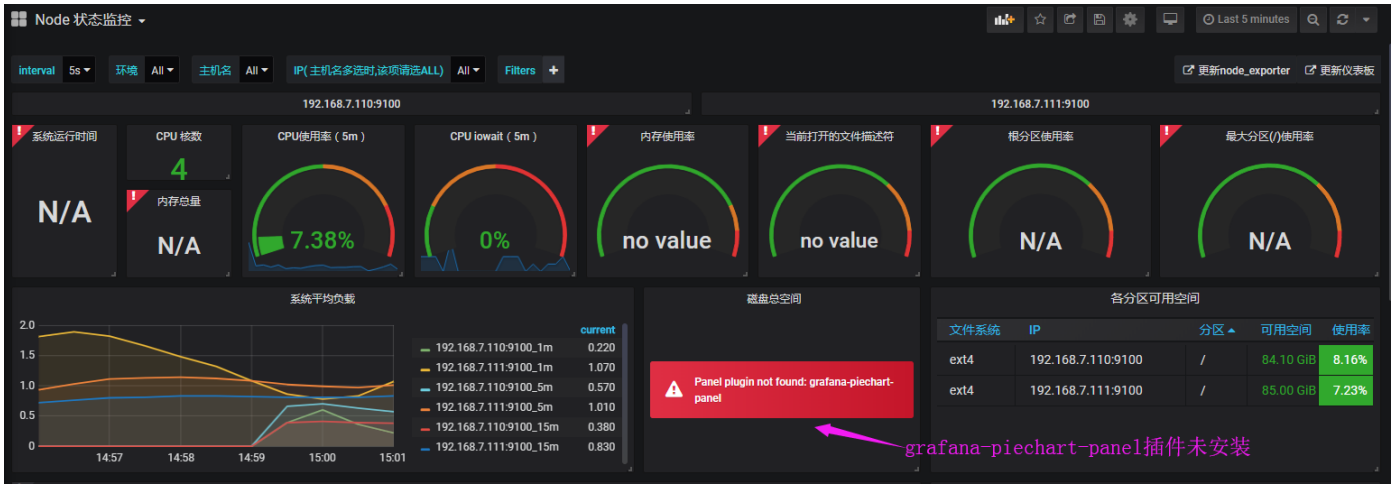
6.4.3.6：验证图形信息：

饼图插件未安装，需要提前安装

https://grafana.com/grafana/plugins/grafana-piechart-panel

```
在线安装：
# grafana-cli plugins install grafana-piechart-panel

离线安装：
# pwd
/var/lib/grafana/plugins

# unzip grafana-piechart-panel-v1.3.8-0-g4f34110.zip
# mv grafana-piechart-panel-4f34110 grafana-piechart-panel
# systemctl  restart grafana-server
```

未安装饼图插件：



已安装饼图插件：

其他监控项图形：

6.3：监控pod资源：

cadvisor由谷歌开源，cadvisor不仅可以搜集一台机器上所有运行的容器信息，还提供基础查询界面和http接口，方便其他组件如Prometheus进行数据抓取，cAdvisor可以对节点机器上的资源及容器进行实时监控和性能数据采集，包括CPU使用情况、内存使用情况、网络吞吐量及文件系统使用情况。

k8s 1.12之前cadvisor集成在node节点的上kubelet服务中，从1.12版本开始分离为两个组件，因此需要在node节点单独部署cadvisor。

https://github.com/google/cadvisor

6.3.1：cadvisor镜像准备：

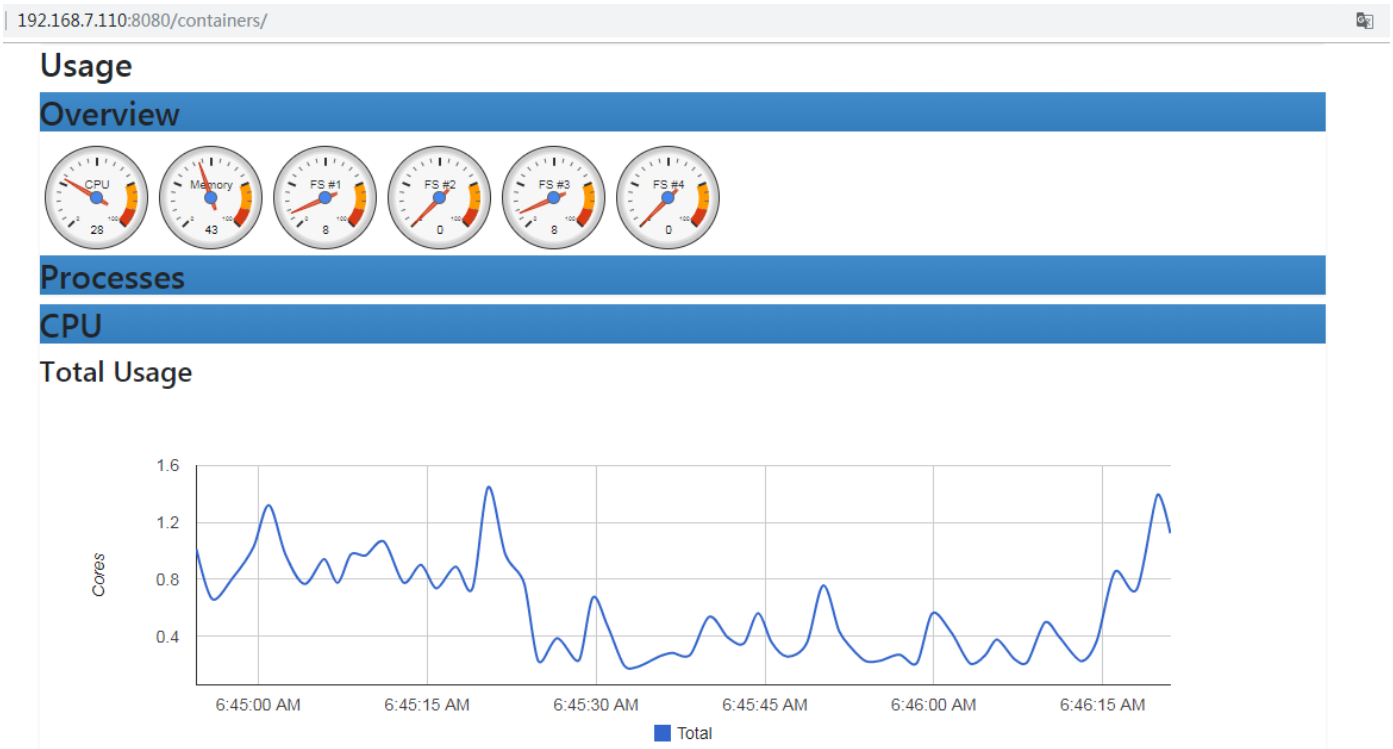
```
# docker load -i  cadvisor-v0.38.7.tar.gz
# docker tag gcr.io/cadvisor/cadvisor:v0.38.7
harbor.magedu.net/linux46/cadvisor:v0.38.7
# docker push harbor.magedu.net/linux46/cadvisor:v0.38.7
```

6.3.2：启动cadvisor容器：

```
# docker run -it -d  \
  --volume=/:/rootfs:ro \
  --volume=/var/run:/var/run:ro \
  --volume=/sys:/sys:ro \
  --volume=/var/lib/docker/:/var/lib/docker:ro \
  --volume=/dev/disk/:/dev/disk:ro \
  --publish=8080:8080 \
  --detach=true \
  --name=cadvisor \
  --privileged \
  --device=/dev/kmsg \
  harbor.magedu.net/linux46/cadvisor:v0.38.7
```

6.3.3：验证cadvisor web界面：

访问node节点的cadvisor监听端口：http://192.168.7.110:8080/



6.3.4：prometheus采集cadvisor数据：

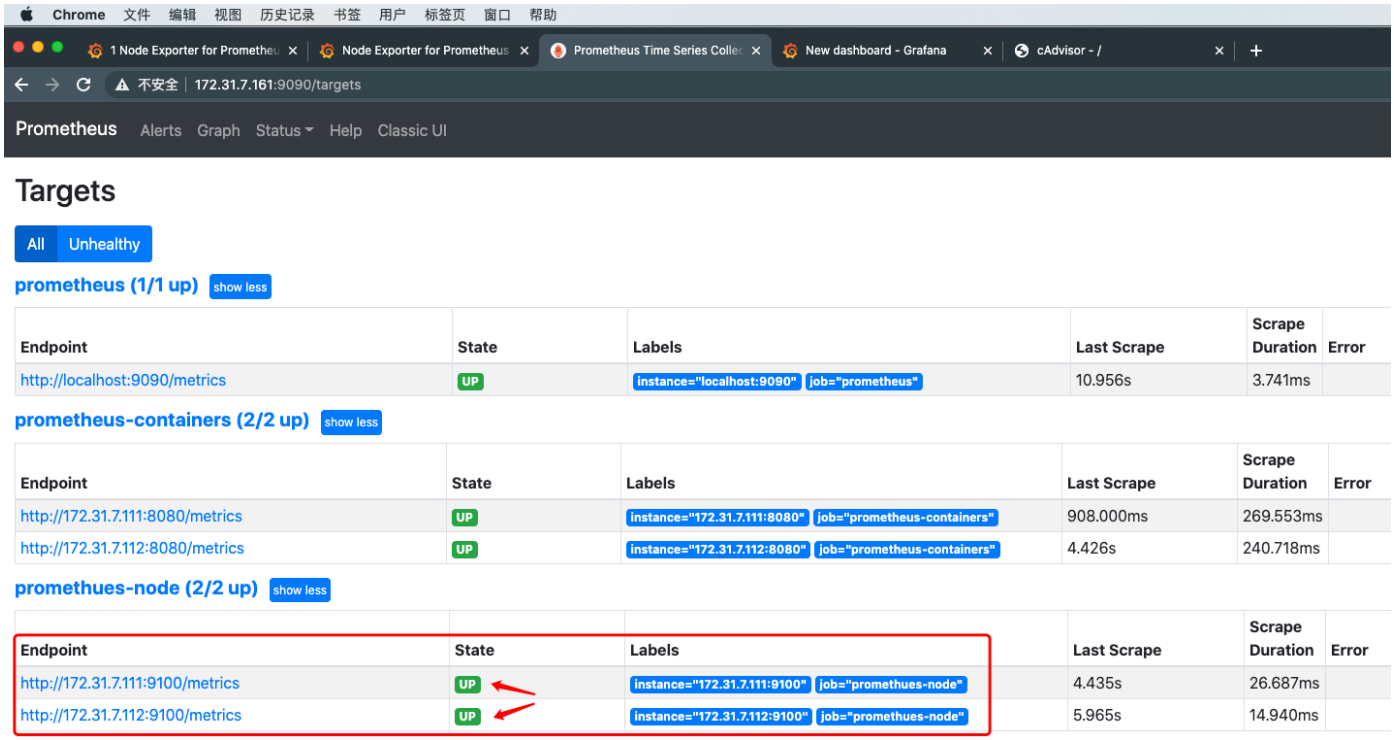
```
root@k8s-master2:~# vim /usr/local/prometheus/prometheus.yml
# my global config
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is
every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1
minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global
'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "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: ['localhost:9090']
  - job_name: 'promethues-node'
    static_configs:
      - targets: ['172.31.7.111:9100','172.31.7.112:9100']

  - job_name: 'prometheus-containers'
    static_configs:
      - targets: ["172.31.7.111:8080","172.31.7.112:8080"]
```

6.3.5：重启prometheus：

```
root@k8s-master2:~# systemctl  restart prometheus
```

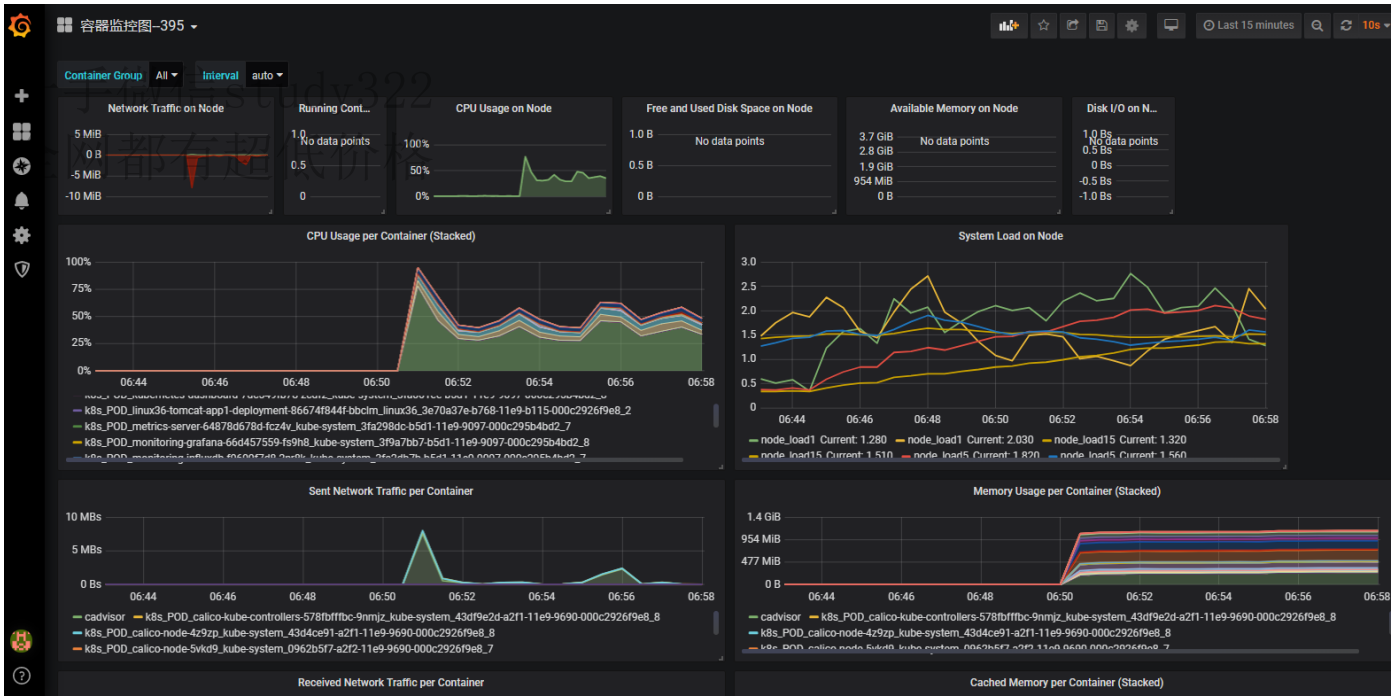
6.3.6：验证prometheus数据：



6.3.7：grafana添加pod监控模板：

395 893 容器模板ID

395模板：

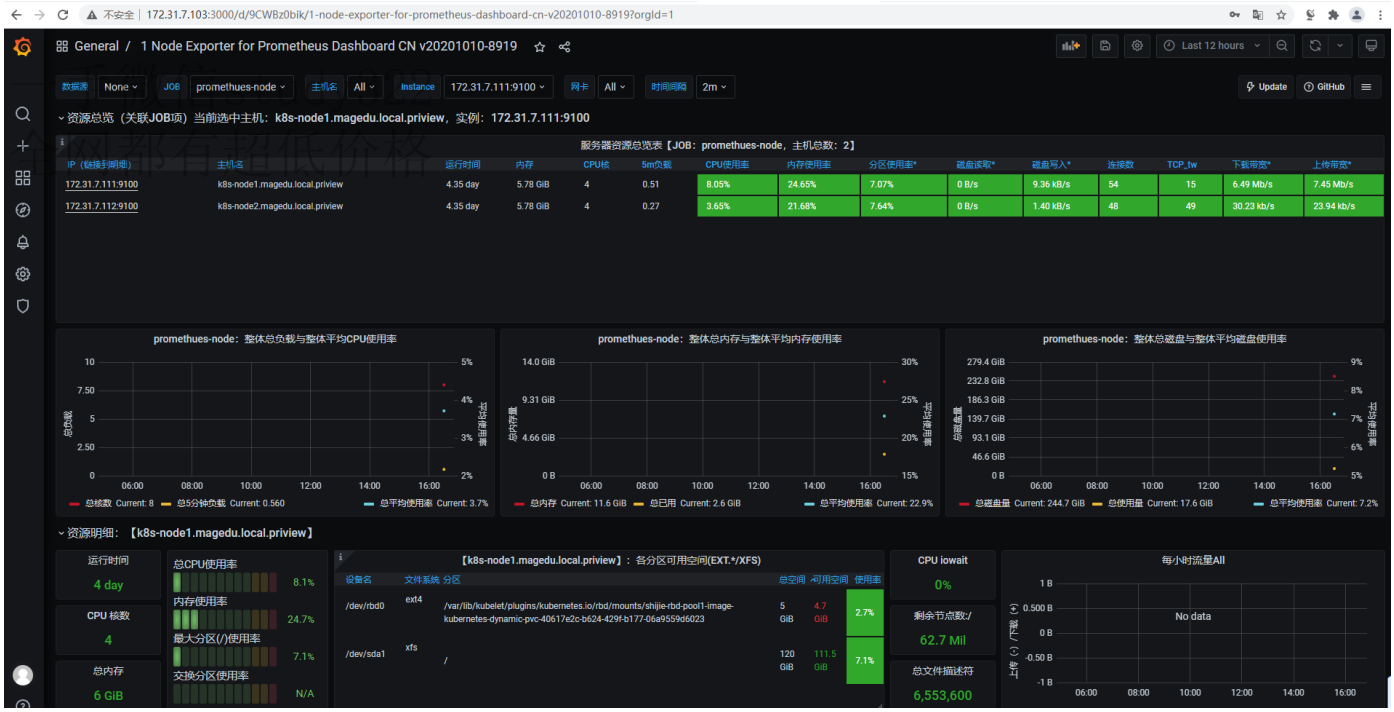


893模板：

修改模板变量

修改模板key的名称


```
8919
```

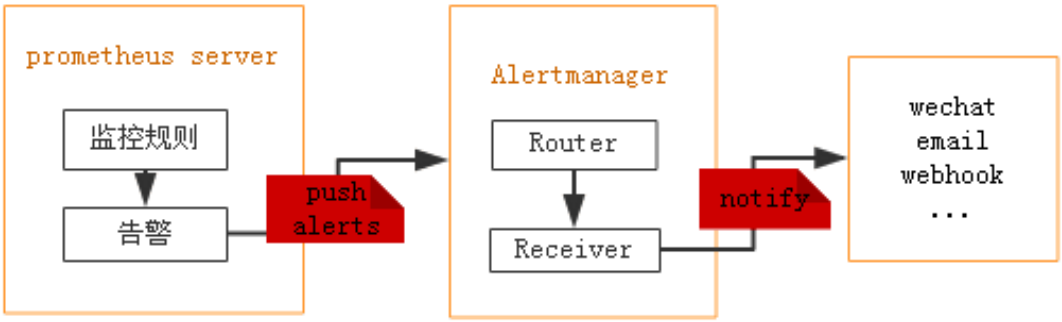


6.4：prometheus报警设置：

prometheus触发一条告警的过程：

prometheus--->触发阈值--->超出持续时间--->alertmanager--->分组|抑制|静默--->媒体类型--->邮件|钉钉|微信等。

分组(group)：将类似性质的警报合并为单个通知，比如网络通知、主机通知、服务通知。
静默(silences)：是一种简单的特定时间静音的机制，例如：服务器要升级维护可以先设置这个时间段告警静默。
抑制(inhibition)：当警报发出后，停止重复发送由此警报引发的其他警报即合并一个故障引起的多个报警事件，可以消除冗余告警



6.4.1：下载并报警组件alertmanager：

```
# pwd
/usr/local/src

# tar xvf alertmanager-0.21.0.freebsd-amd64.tar.gz

# ln -sv /apps/alertmanager-0.21.0.freebsd-amd64 /apps/alertmanager
```

```
# cat /etc/systemd/system/alertmanager.service
[Unit]
Description=Prometheus Server
Documentation=https://prometheus.io/docs/introduction/overview/
After=network.target

[Service]
Restart=on-failure
WorkingDirectory=/apps/alertmanager
ExecStart=/apps/alertmanager/alertmanager

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

6.4.2：配置alertmanager：

https://prometheus.io/docs/alerting/configuration/ #官方配置文档

```
global:
  smtp_from:       #发件人邮箱地址
  smtp_smarthost: #邮箱smtp地址。
  smtp_auth_username:   #发件人的登陆用户名，默认和发件人地址一致。
  smtp_auth_password:   #发件人的登陆密码，有时候是授权码。
  smtp_require_tls:     #是否需要tls协议。默认是true。

  wechart_api_url:      #企业微信API 地址。
  wechart_api_secret:  #企业微信API secret
  wechat_api_corp_id:   #企业微信corp id信息。

  resolve_timeout:   #在指定时间内没有产生新的事件就发送恢复通知
```

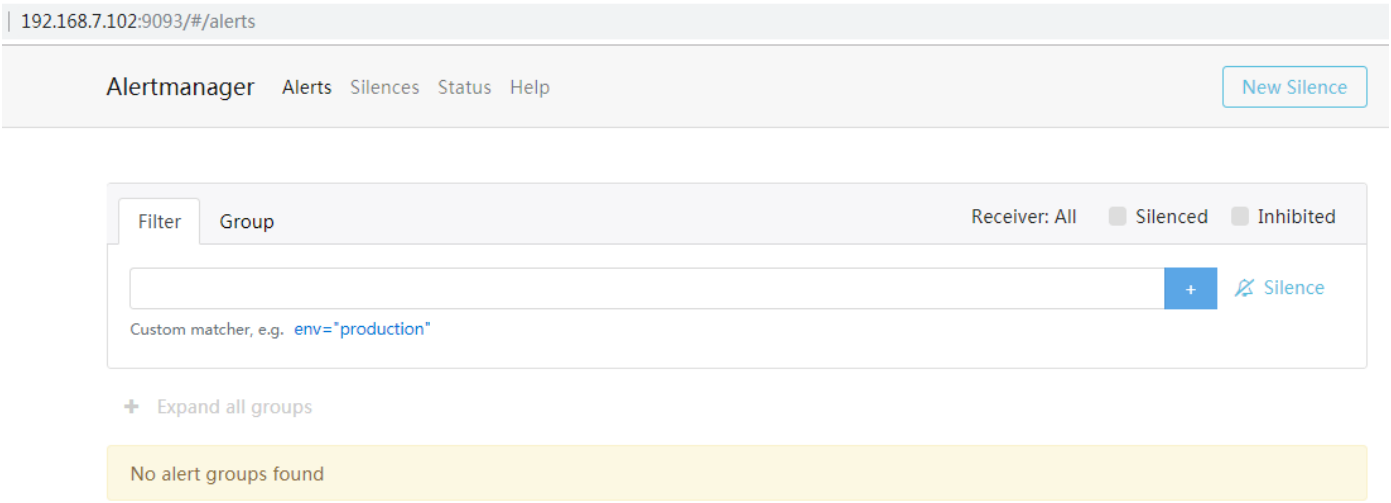
```
# pwd
/apps/alertmanager

# cat alertmanager.yml
global:
  resolve_timeout: 5m
  smtp_smarthost: 'smtp.qq.com:465'
  smtp_from: '2973707860@qq.com'
  smtp_auth_username: '2973707860@qq.com'
  smtp_auth_password: 'udwthyyxtstcdhcj'
  smtp_hello: '@qq.com'
  smtp_require_tls: false


route: #route用来设置报警的分发策略
```

```
  group_by: ['alertname']  #采用哪个标签来作为分组依据
  group_wait: 10s #组告警等待时间。也就是告警产生后等待10s，如果有同组告警一起发出
  group_interval: 10s #两组告警的间隔时间
  repeat_interval: 2m #重复告警的间隔时间，减少相同邮件的发送频率
  receiver: 'web.hook' #设置接收人
receivers:
- name: 'web.hook'
  #webhook_configs:
  #- url: 'http://127.0.0.1:5001/'
  email_configs:
    - to: '2973707860@qq.com'

inhibit_rules: #抑制的规则
  - source_match: #源匹配级别，当匹配成功发出通知，但是其他的通知将被抑制
      severity: 'critical' #
    target_match:
      severity: 'warning'
    equal: ['alertname', 'dev', 'instance']
```

6.4.3：启动alertmanager服务：

```
# systemctl  restart  alertmanager.service

#验证alertmanager的9093端口已经监听
# lsof  -i:9093
COMMAND      PID USER   FD   TYPE  DEVICE SIZE/OFF NODE NAME
alertmana 127083 root    6u  IPv6 8581566      0t0  TCP *:9093 (LISTEN)
```

alertmanager dashboard截图



6.4.4：配置prometheus报警规则：

```
# cd  /usr/local/prometheus

# vim prometheus.yml
# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
       - 192.168.7.102:9093  #alertmanager地址

# Load rules once and periodically evaluate them according to the global
'evaluation_interval'.
rule_files:
  - "/apps/prometheus/rule.yml" #指定规则文件
  # - "second_rules.yml"
```

6.4.5：创建报警规则文件：

```
# pwd
/usr/local/prometheus

# cat /usr/local/prometheus/rule-linux36.yml
groups:
  - name: alertmanager_pod.rules
    rules:
    - alert: Pod_all_cpu_usage
      expr: (sum by(name)(rate(container_cpu_usage_seconds_total{image!=""}[5m]))*100)
> 1
      for: 2m
      labels:
        severity: critical
        service: pods
      annotations:
        description: 容器 {{ $labels.name }} CPU 资源利用率大于 10% , (current value is {{
$value }})
        summary: Dev CPU 负载告警

    - alert: Pod_all_memory_usage
      #expr: sort_desc(avg by(name)(irate(container_memory_usage_bytes{name!=""}
[5m]))*100) > 10  #内存大于10%
      expr: sort_desc(avg by(name)(irate(node_memory_MemFree_bytes {name!=""}[5m]))) >
2   #内存大于2G
      for: 2m
      labels:
        severity: critical
      annotations:
```

```
        description: 容器 {{ $labels.name }} Memory 资源利用率大于 2G , (current value is
{{ $value }})
        summary: Dev Memory 负载告警

    - alert: Pod_all_network_receive_usage
      expr: sum by (name)
(irate(container_network_receive_bytes_total{container_name="POD"}[1m])) > 1
      for: 2m
      labels:
        severity: critical
      annotations:
        description: 容器 {{ $labels.name }} network_receive 资源利用率大于 50M , (current
value is {{ $value }})

    - alert: pod内存可用大小
      expr: node_memory_MemFree_bytes > 1 #故意写错的
      for: 2m
      labels:
        severity: critical
      annotations:
        description: 容器可用内存小于100k
```

6.4.6：报警规则验证：

```
# pwd
/usr/local/prometheus

#验证报警规则设置：
#  ./promtool check rules  rule-linux36.yml  #监测rule规则文件是否正确
Checking rule-linux36.yml
  SUCCESS: 3 rules found
```

6.4.7：重启prometheus：

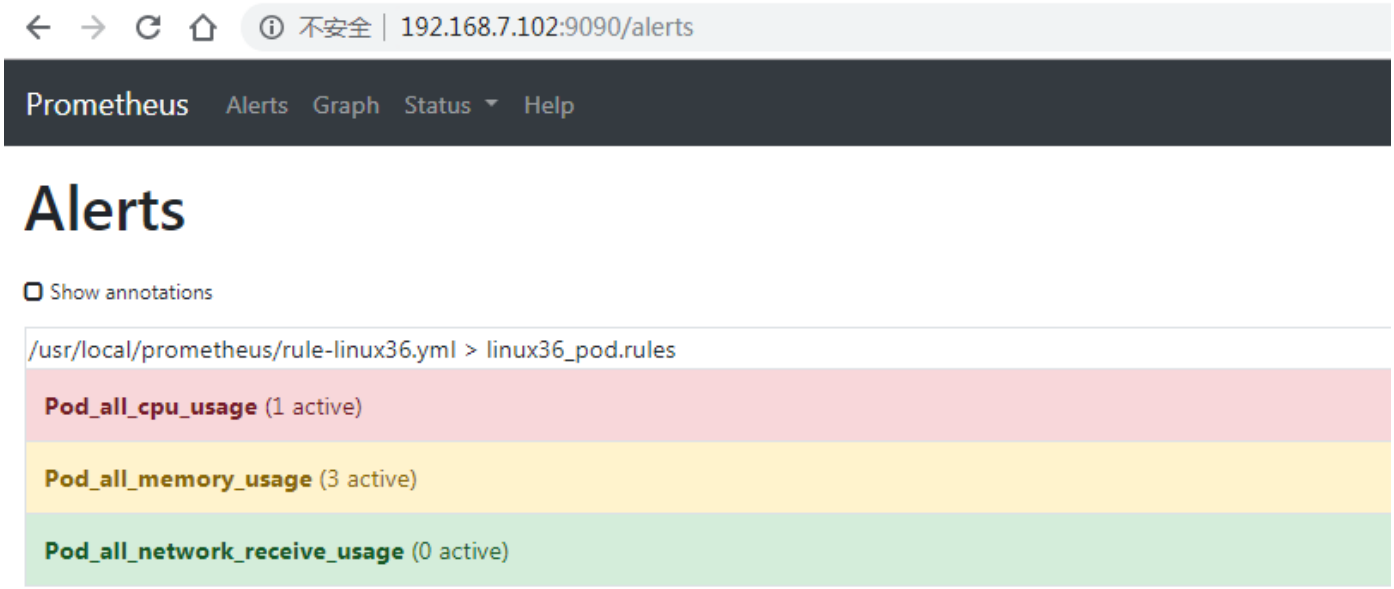
```
# systemctl  restart prometheus
```

6.4.8：验证报警规则匹配：

```
# pwd
/usr/local/alertmanager

# ./amtool alert --alertmanager.url=http://192.168.7.102:9093
Alertname          Starts At                Summary
Pod_all_cpu_usage  2019-08-07 07:39:04 CST  Dev CPU 负载告警
```

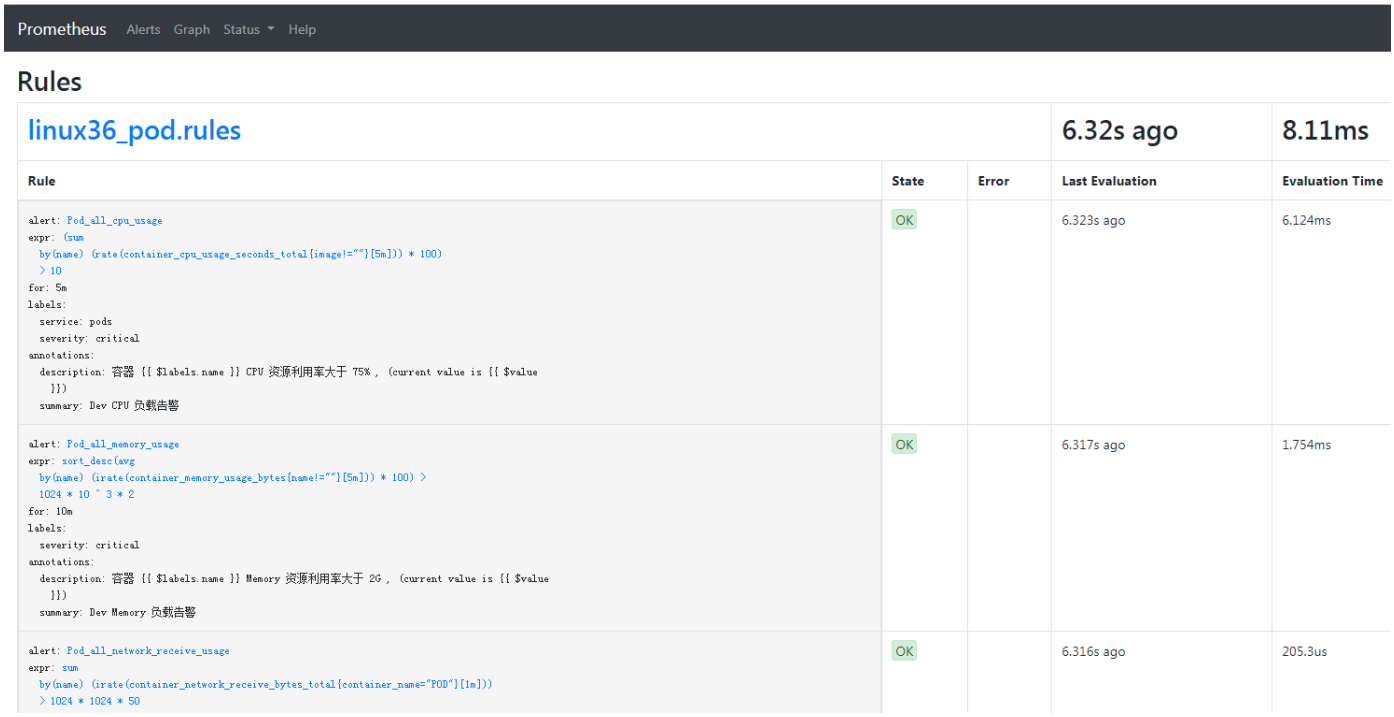
6.4.9：prometheus首页状态：

promethus报警状态
- Inactive：没有异常。
- Pending：已触发阈值，但未满足告警持续时间（即rule中的for字段）
- Firing：已触发阈值且满足条件并发送至alertmanager

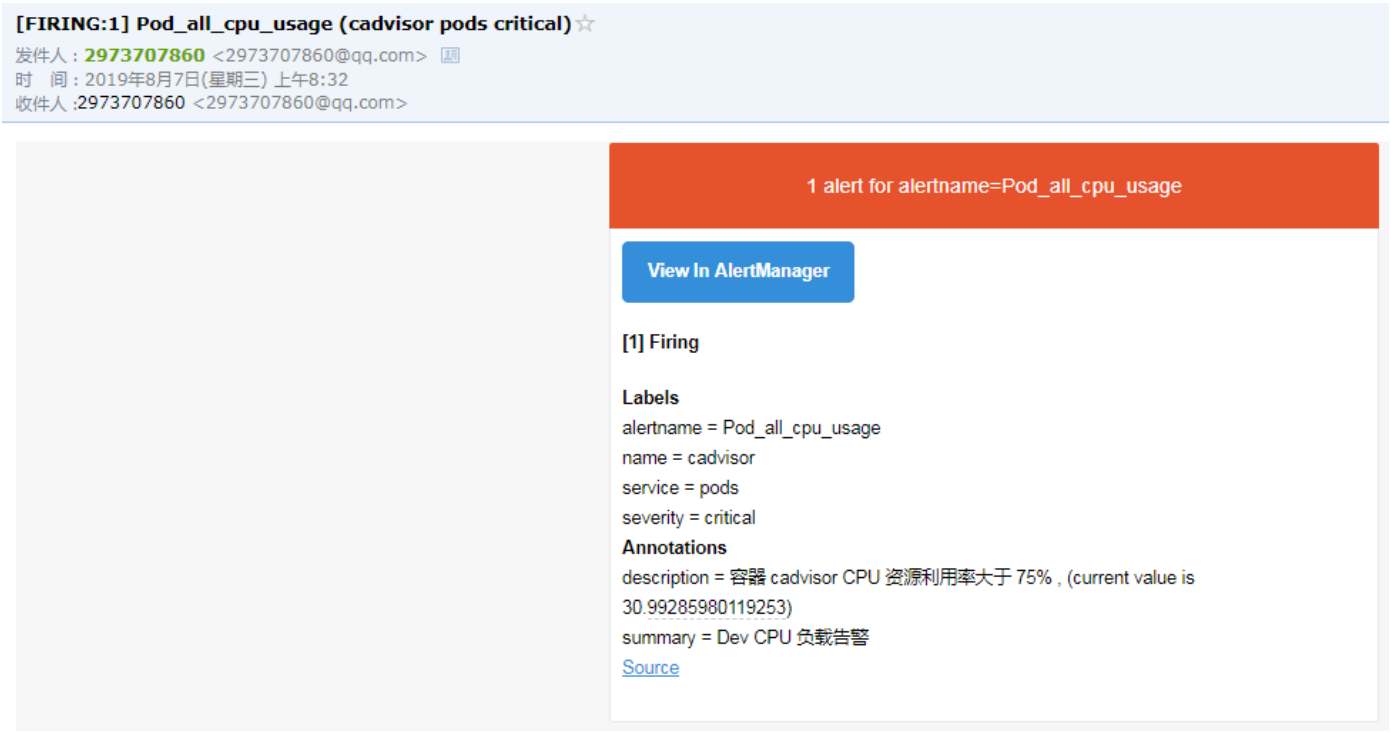


6.4.10：prometheus web界面验证报警规则：

status-rules



6.4.11：验证收到的报警邮件：

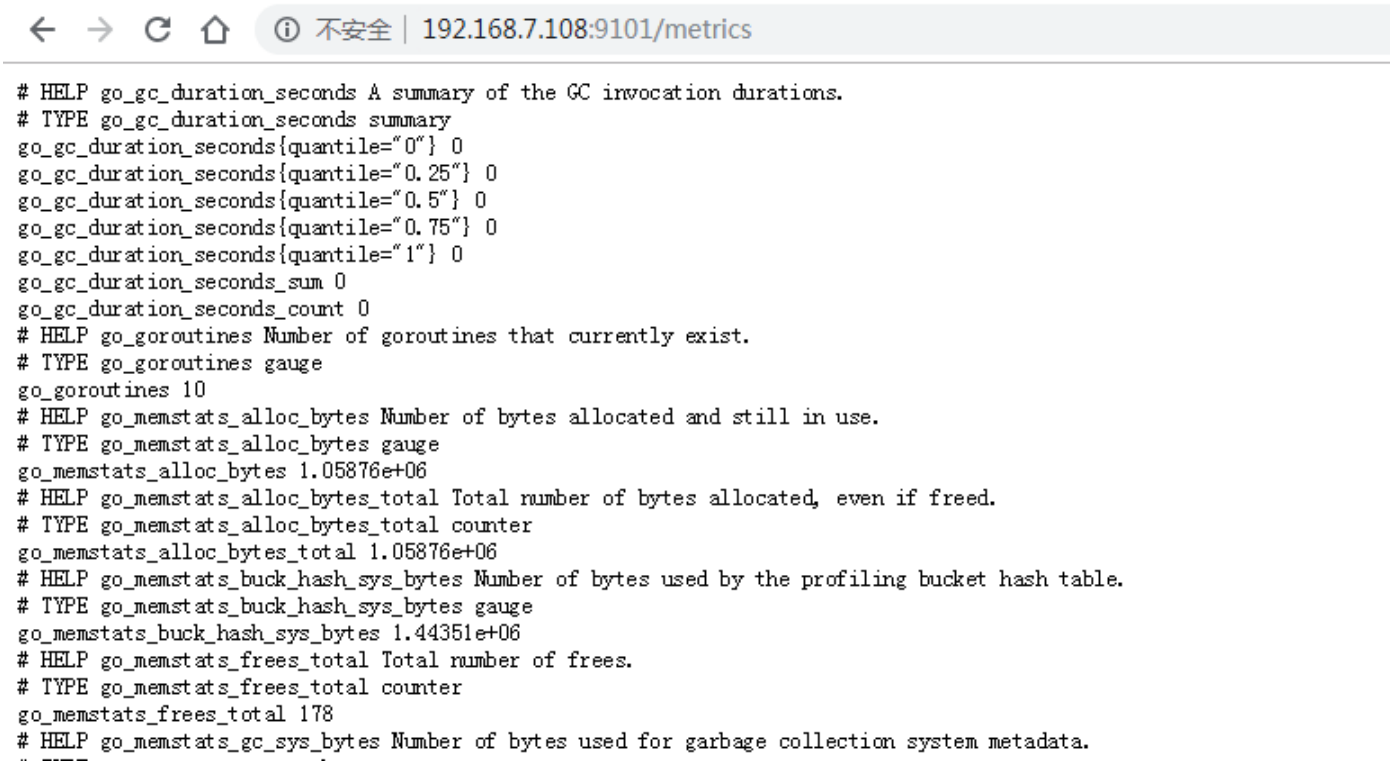


6.5：prometheus监控haproxy：

6.5.1：部署haproxy_exporter：

```
# pwd
/usr/local/src

# tar xvf haproxy_exporter-0.9.0.linux-amd64.tar.gz
# ln -sv /usr/local/src/haproxy_exporter-0.9.0.linux-amd64 /usr/local/haproxy_exporter

# cd /usr/local/haproxy_exporter
# ./haproxy_exporter --haproxy.scrape-uri=unix:/run/haproxy/admin.sock
# ./haproxy_exporter --haproxy.scrape-
uri="http://haadmin:q1w2e3r4ys@127.0.0.1:9999/haproxy-status;csv" &
```

6.5.2：验证web界面数据：



6.5.3：prometheus server端添加haproxy数据采集：

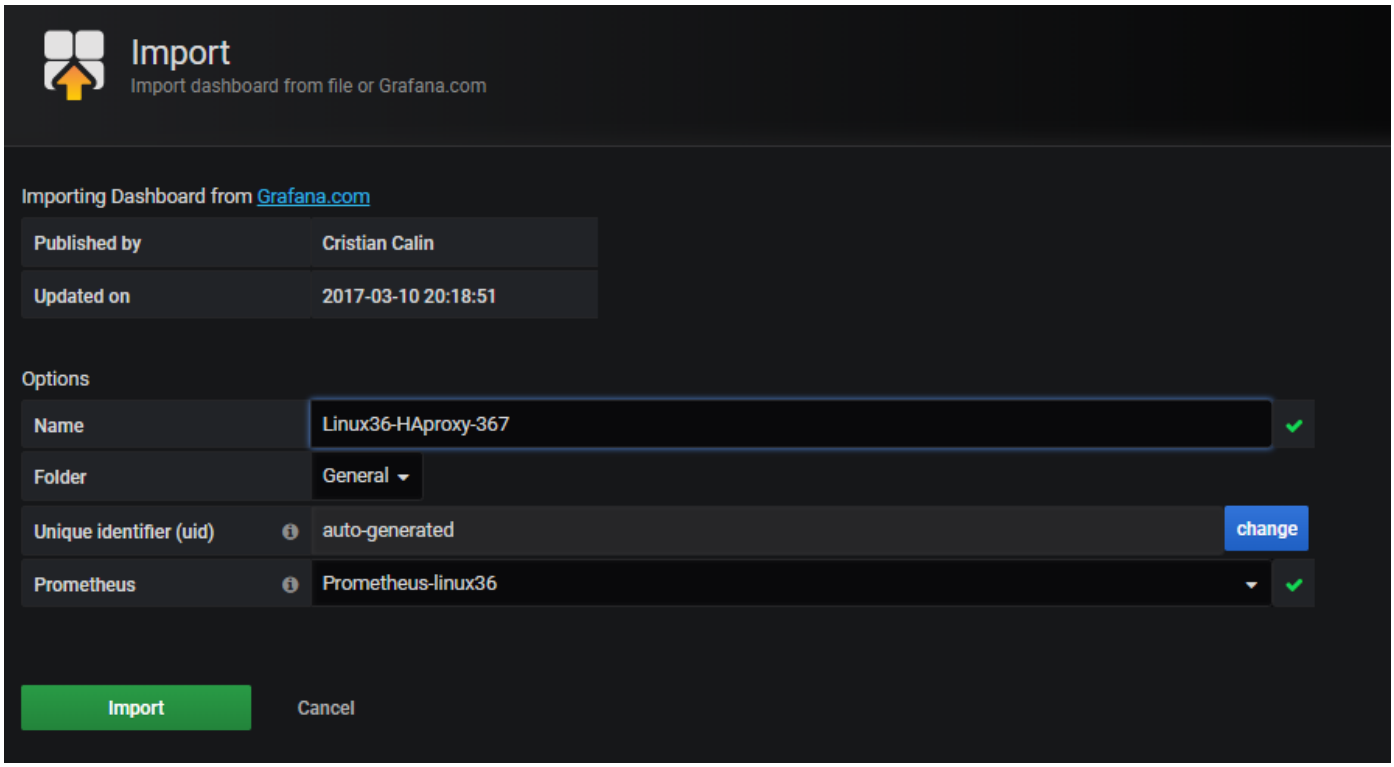
```
# vim /usr/local/prometheus/prometheus.yml

# cd /usr/local/prometheus/
#  grep -v "#"  prometheus.yml  | grep -v "^$"
global:
alerting:
  alertmanagers:
  - static_configs:
    - targets: ["192.168.7.102:9093"]
rule_files:
  - "/usr/local/prometheus/rule-linux36.yml"
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
    - targets: ['localhost:9090']
  - job_name: 'promethues-node'
    static_configs:
      - targets: ['192.168.7.110:9100','192.168.7.111:9100']
  - job_name: 'prometheus-containers'
    static_configs:
      - targets: ["192.168.7.110:8080","192.168.7.111:8080"]
  - job_name: 'prometheus-haproxy'
    static_configs:
      - targets: ["192.168.7.108:9101"]
```

6.5.4：重启prometheus：

```
# systemctl  restart prometheus
```

6.5.5：grafana添加模板：

367 2428



6.5.6：验证haproxy监控数据：

6.6：prometheus监控nginx：

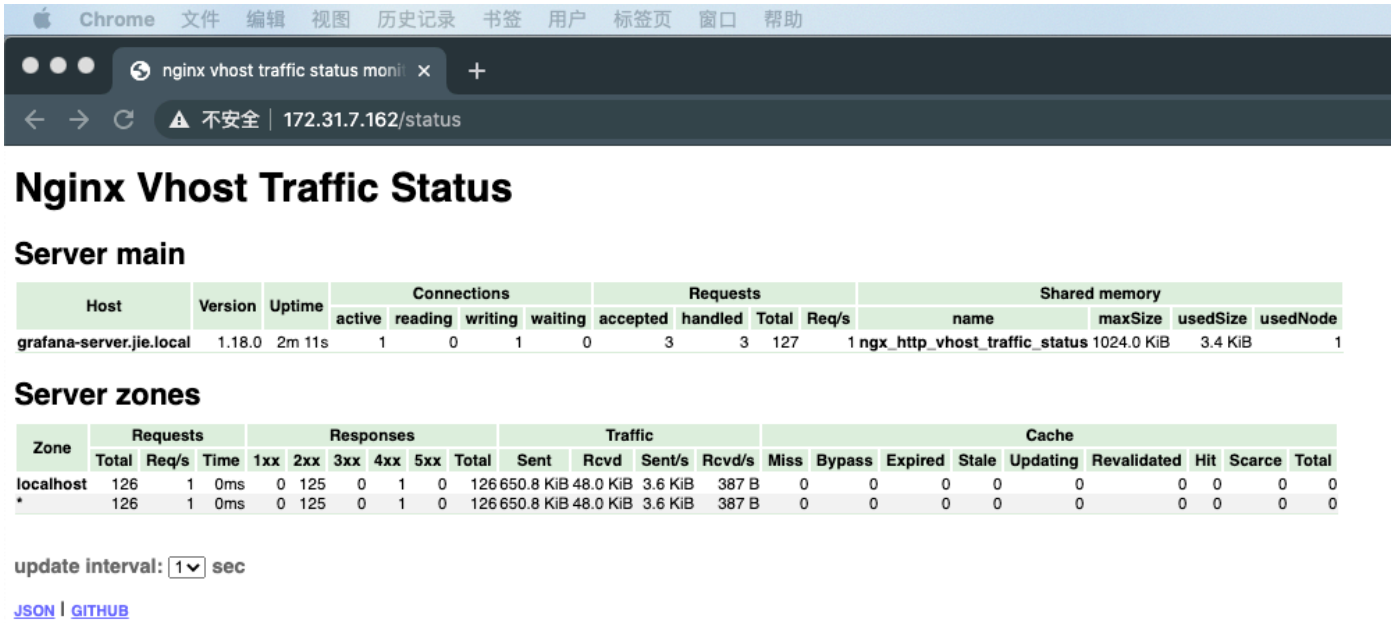
通过prometheus监控nginx

需要在编译安装nginx的时候添加nginx-module-vts模块，github地址：https://github.com/vozlt/nginx-module-vts

6.6.1：编译安装nginx：


```
# git clone https://github.com/vozlt/nginx-module-vts.git
# wget http://nginx.org/download/nginx-1.18.0.tar.gz
# tar xvf nginx-1.18.0.tar.gz
# cd nginx-1.18.0/
# ./configure --prefix=/apps/nginx --add-module=/usr/local//src/nginx-module-vts/
# cd /apps/nginx/conf/

# vim nginx.conf
    #gzip  on;
vhost_traffic_status_zone; #
    server {
        listen       80;
        server_name  localhost;

        #charset koi8-r;

        #access_log  logs/host.access.log  main;

        location / {
            root   html;
            index  index.html index.htm;
        }

location /status {
     vhost_traffic_status_display;
     vhost_traffic_status_display_format html;
    }
}

root@grafana-server:/apps/nginx/conf# /apps/nginx/sbin/nginx  -t
nginx: the configuration file /apps/nginx/conf/nginx.conf syntax is ok
nginx: configuration file /apps/nginx/conf/nginx.conf test is successful
root@grafana-server:/apps/nginx/conf# /apps/nginx/sbin/nginx
```

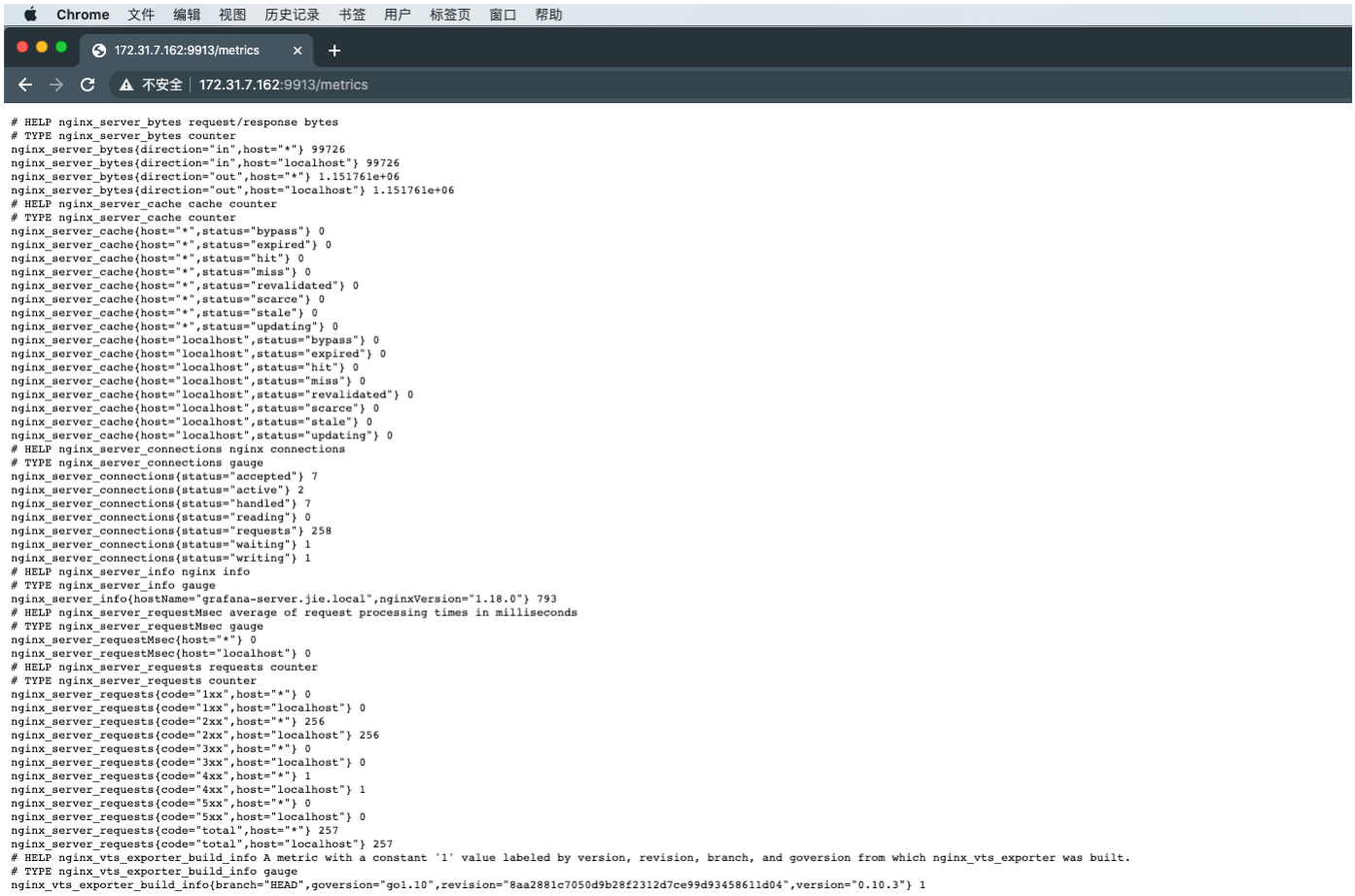
6.6.2：验证状态页：



6.6.3：安装nginx exporter：

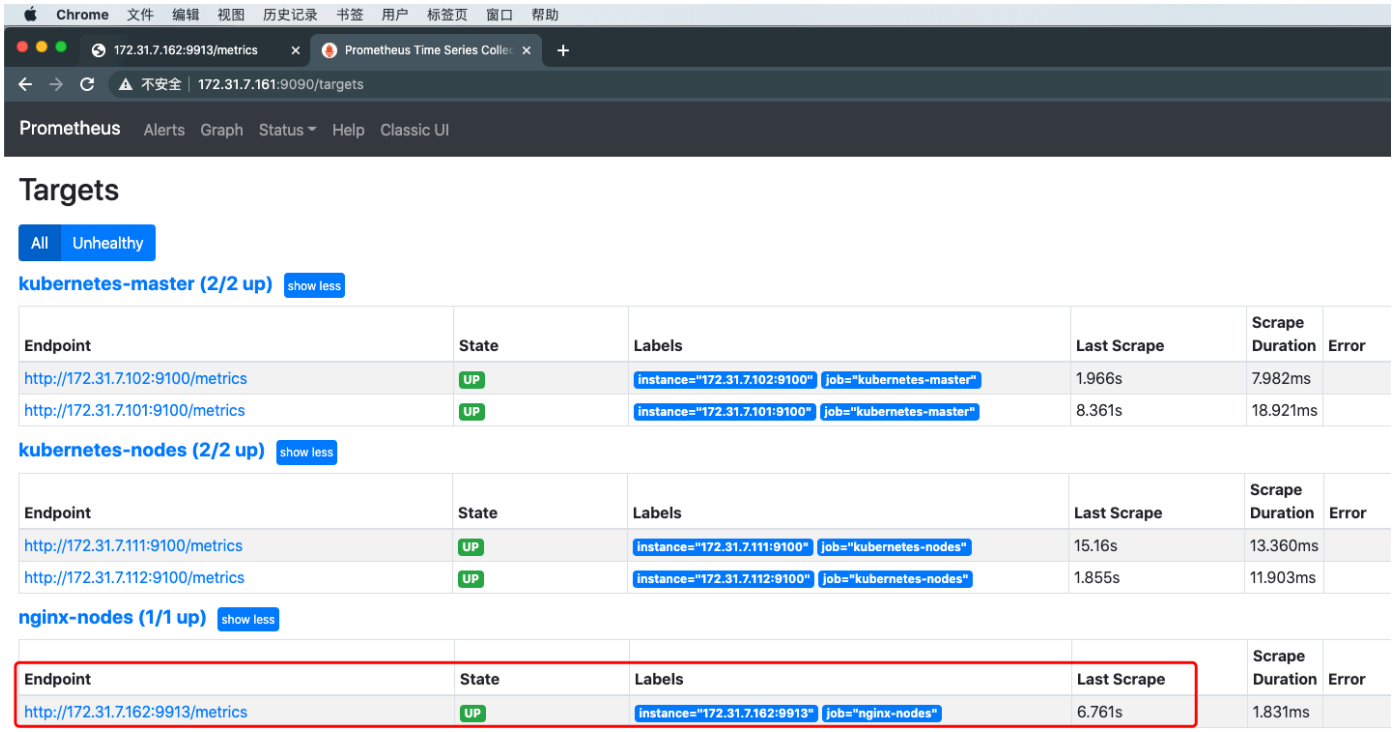
```
# wget https://github.com/hnlq715/nginx-vts-exporter/releases/download/v0.10.3/nginx-vts-exporter-0.10.3.linux-amd64.tar.gz
# tar xvf nginx-vts-exporter-0.10.3.linux-amd64.tar.gz
# ln -sv /apps/nginx-vts-exporter-0.10.3.linux-amd64 /apps/nginx-vts-exporter
'/apps/nginx-vts-exporter' -> '/apps/nginx-vts-exporter-0.10.3.linux-amd64'

# cd /apps/nginx-vts-exporter
# ./nginx-vts-exporter  -nginx.scrape_uri http://172.31.7.162/status/format/json
```

6.6.4：验证nginx exporter数据：



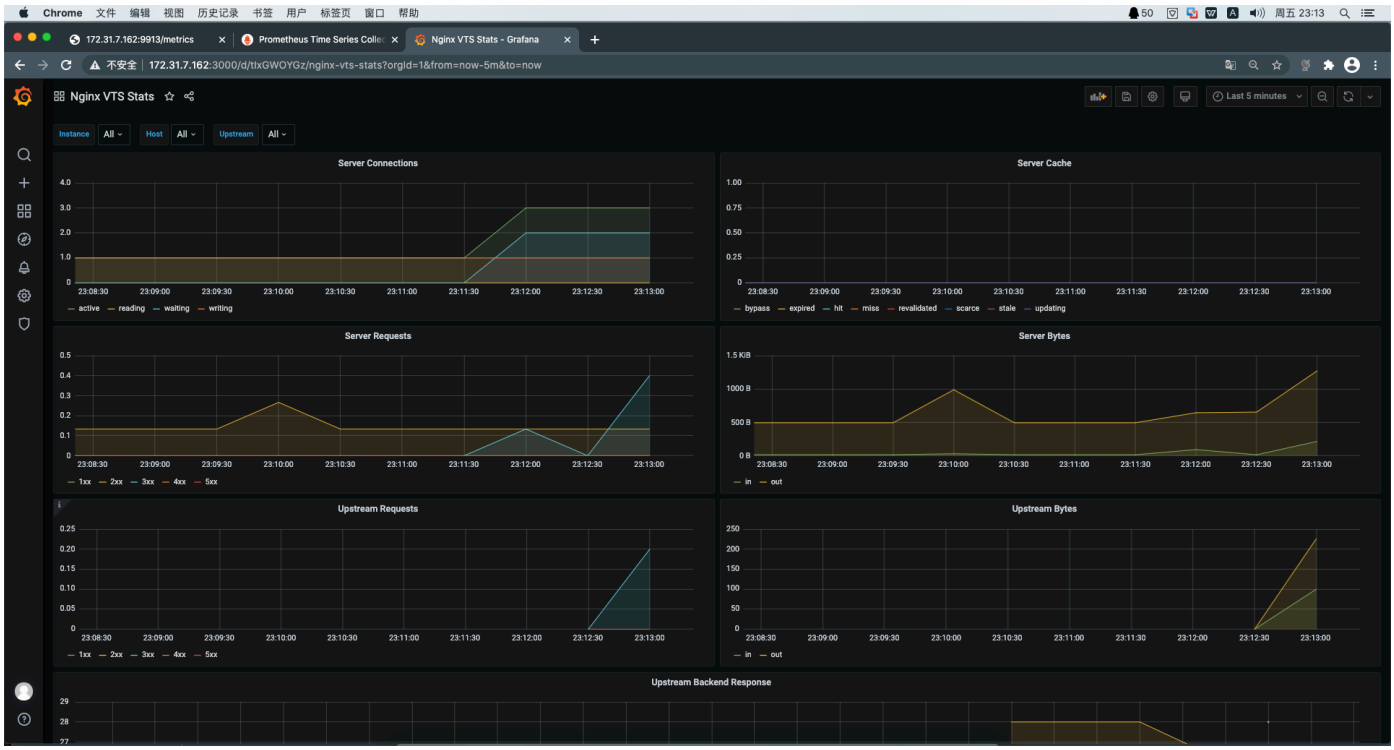
6.6.5：prometheus配置:

```
~# vim /apps/prometheus/prometheus.yml
  - job_name: 'nginx-nodes'
    static_configs:
    - targets: ['172.31.7.162:9913']
# systemctl  restart prometheus.service
```



6.6.6：grafana关联模板：

https://grafana.com/grafana/dashboards/2949



https://developers.dingtalk.com/document/app/how-to-call-apis

https://github.com/timonwong/prometheus-webhook-dingtalk

```
curl 'https://oapi.dingtalk.com/robot/send?
access_token=ba217ad9d1b70a9c8b10a88f46bb5e0e04fbcfd36bf80d22ee7cd99c79648dad' \
  -H 'Content-Type: application/json' \
  -d '{"msgtype": "text",
    "text": {
       "content": "业务运维报警机器人-Node01 is down"
    }
   }'
```

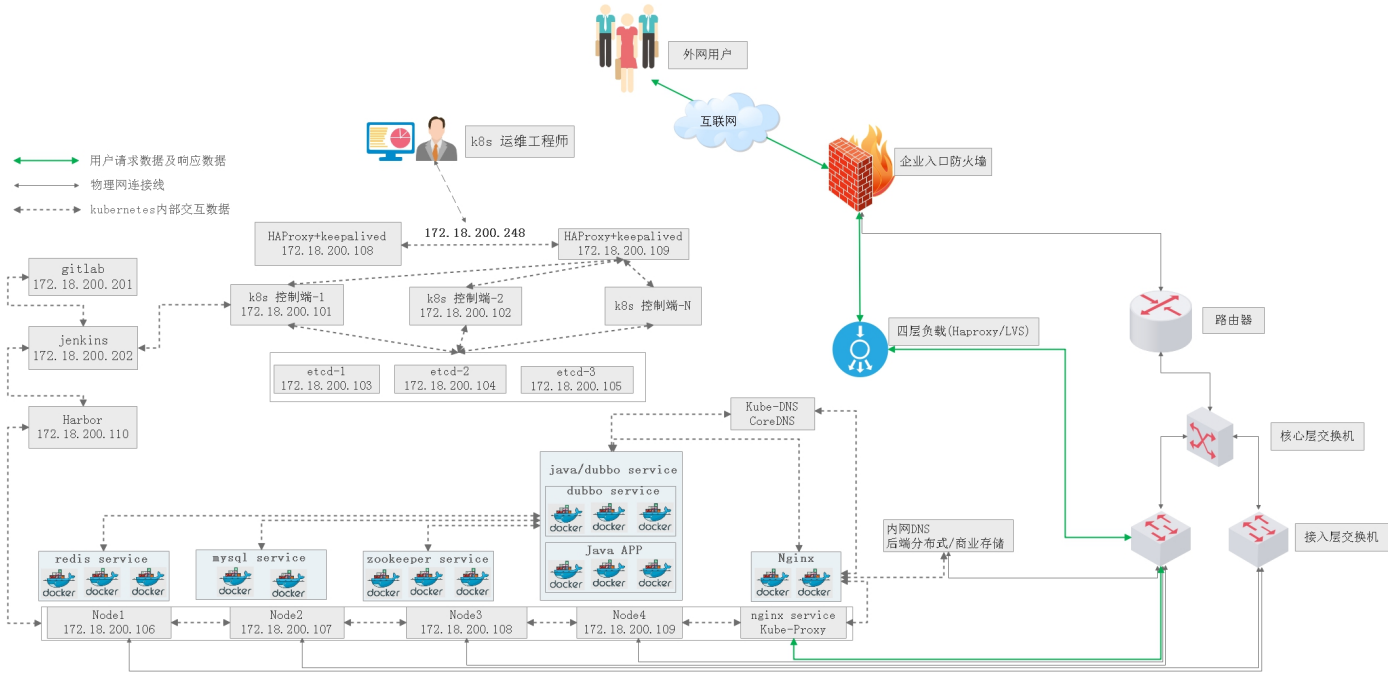
七.k8s:实战案例:

http://www.uml.org.cn/yunjisuan/201703162.asp?artid=19131 #京东从openstack切换为K8S

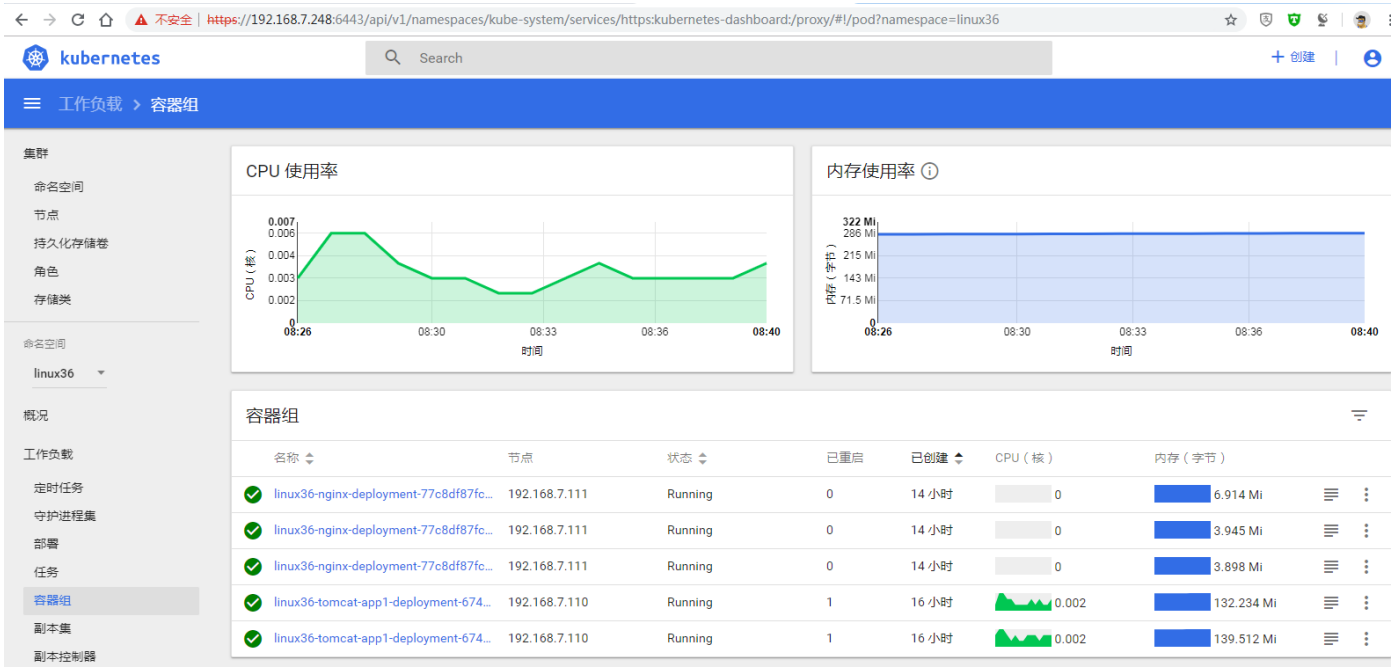
7.1：K8S高可用:

基于HAProxy+Keepalived实现高可用k8s集群环境、实现K8S版本升级、calico与flannel网络通信、kube DNS与CoreDNS、Dashboard。

7.1.1：高可用K8S基础环境:



7.1.2：当前运行状态：



7.2：动静分离web站点：

以下服务要求全部运行在K8S环境内，主要介绍在k8s中运行目前比较常见的主流服务和架构，如基于Nginx+Tomcat的动静分离架构、基于PVC实现的Zookeeper集群和Redis服务，基于PVC+StatefulSet实现的MySQL主从架构，运行java应用，K8S中基于Nginx+PHP+MySQL实现的WordPress的web站点，如何在k8s中基于Zookeeper运行微服务等，以及K8S的CI与CD、日志收集分析展示与prometheus+grafana实现pod监控与报警等。

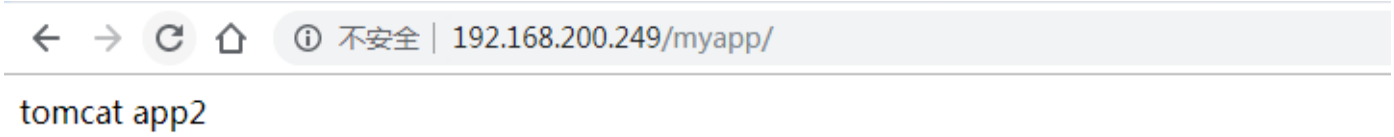
7.2.1：Nginx+Tomcat实动静分离web站点：

基于Nginx+Tomcat+NFS实现通过域名转发动态请求到Tomcat Pod的动静分离架构，要求能够通过负载均衡的VIP访问到k8s集群中运行的Nginx+Tomcat+NFS中的web页面。

```
root@s1:~# kubectl  exec -it linux35-nginx-deployment-768f9477bd-nlkr4 bash -n linux35
[root@linux35-nginx-deployment-768f9477bd-nlkr4 /]# df -TH
Filesystem                              Type      Size  Used Avail Use% Mounted on
overlay                                 overlay   105G  5.0G   94G   6% /
tmpfs                                   tmpfs      68M     0   68M   0% /dev
tmpfs                                   tmpfs     1.1G     0  1.1G   0% /sys/fs/cgroup
/dev/mapper/ubuntu--vg-root             ext4      105G  5.0G   94G   6% /etc/hosts
shm                                     tmpfs      68M     0   68M   0% /dev/shm
tmpfs                                   tmpfs     1.1G   13k  1.1G   1% /run/secrets/kubernetes.io/serviceaccount
192.168.200.102:/data/linux35/static    nfs4      105G  3.1G   96G   4% /usr/local/nginx/html/webapp/static
192.168.200.102:/data/linux35/img       nfs4      105G  3.1G   96G   4% /usr/local/nginx/html/webapp/images
tmpfs                                   tmpfs     1.1G     0  1.1G   0% /proc/acpi
tmpfs                                   tmpfs     1.1G     0  1.1G   0% /proc/scsi
tmpfs                                   tmpfs     1.1G     0  1.1G   0% /sys/firmware
[root@linux35-nginx-deployment-768f9477bd-nlkr4 /]#
```

```
http {
    include       mime.types;
    default_type  application/octet-stream;
    sendfile        on;
```

```
    keepalive_timeout  65;
upstream  tomcat_webserver {
        server  linux35-tomcat-app1-spec.linux35.svc.linux35.local:80;
        server  linux35-tomcat-app2-spec.linux35.svc.linux35.local:80;
}
    server {
        listen       80;
        server_name  localhost;
        location / {
            root   html;
            index  index.html index.htm;
        }
        location /webapp {
            root   html;
            index  index.html index.htm;
        }
        location /myapp {
             proxy_pass  http://tomcat_webserver;
             proxy_set_header   Host    $host;
       proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
             proxy_set_header X-Real-IP $remote_addr;
        }
        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }
    }
}
```



7.3：PV及PVC实战案例：

默认情况下容器中的磁盘文件是非持久化的，对于运行在容器中的应用来说面临两个问题，第一：当容器挂掉kubelet将重启启动它时，文件将会丢失；第二：当Pod中同时运行多个容器，容器之间需要共享文件时，Kubernetes的Volume解决了这两个问题。

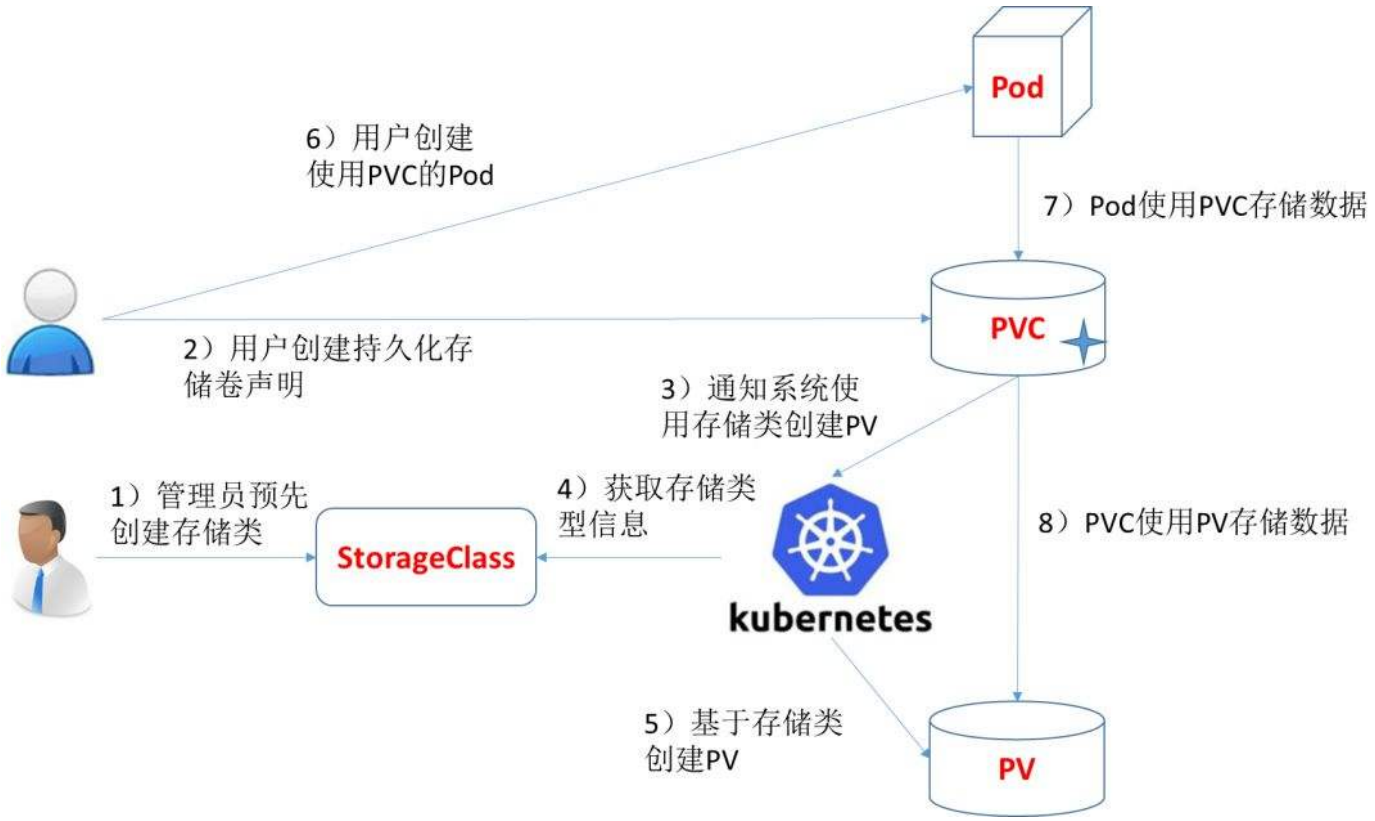
https://v1-14.docs.kubernetes.io/zh/docs/concepts/storage/ #官方文档

PersistentVolume（PV）是集群中已由管理员配置的一段网络存储，集群中的存储资源就像一个node节点是一个集群资源，PV是诸如卷之类的卷插件，但是具有独立于使用PV的任何单个pod的生命周期， 该API对象捕获存储的实现细节，即NFS，iSCSI或云提供商特定的存储系统，PV是由管理员添加的的一个存储的描述，是一个全局资源即不隶属于任何namespace，包含存储的类型，存储的大小和访问模式等，它的生命周期独立于Pod，例如当使用它的Pod销毁时对PV没有影响。

PersistentVolumeClaim（PVC）是用户存储的请求，它类似于pod，Pod消耗节点资源，PVC消耗存储资源， 就像pod可以请求特定级别的资源（CPU和内存），PVC是namespace中的资源，可以设置特定的空间大小和访问模式。

kubernetes 从1.0版本开始支持PersistentVolume和PersistentVolumeClaim。

PV是对底层网络存储的抽象，即将网络存储定义为一种存储资源，将一个整体的存储资源拆分成多份后给不同的业务使用。
PVC是对PV资源的申请调用，就像POD消费node节点资源一样,pod是通过PVC将数据保存至PV，PV在保存至存储。



PersistentVolume参数：

```
# kubectl  explain  PersistentVolume

Capacity:  #当前PV空间大小，kubectl explain  PersistentVolume.spec.capacity

accessModes ：访问模式，#kubectl  explain  PersistentVolume.spec.accessModes
  ReadWriteOnce – PV只能被单个节点以读写权限挂载，RWO
  ReadOnlyMany – PV以可以被多个节点挂载但是权限是只读的,ROX
  ReadWriteMany – PV可以被多个节点是读写方式挂载使用,RWX

persistentVolumeReclaimPolicy #删除机制即删除存储卷卷时候，已经创建好的存储卷由以下删除操作：
#kubectl  explain  PersistentVolume.spec.persistentVolumeReclaimPolicy
```

```
  Retain –  删除PV后保持原装，最后需要管理员手动删除
  Recycle – 空间回收，及删除存储卷上的所有数据(包括目录和隐藏文件),目前仅支持NFS和hostPath
  Delete –  自动删除存储卷

volumeMode #卷类型，kubectl  explain  PersistentVolume.spec.volumeMode
  定义存储卷使用的文件系统是块设备还是文件系统，默认为文件系统

mountOptions #附加的挂载选项列表，实现更精细的权限控制
  ro  #等
```

官方提供的基于各后端存储创建的PV支持的访问模式

https://kubernetes.io/docs/concepts/storage/persistent-volumes/#mount-options

Volume Plugin	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
AWSElasticBlockStore	✓	-	-
AzureFile	✓	✓	✓
AzureDisk	✓	-	-
CephFS	✓	✓	✓
Cinder	✓	-	-
CSI	depends on the driver	depends on the driver	depends on the driver
FC	✓	✓	-
Flexvolume	✓	✓	depends on the driver
Flocker	✓	-	-
GCEPersistentDisk	✓	✓	-
Glusterfs	✓	✓	✓
HostPath	✓	-	-
iSCSI	✓	✓	-
Quobyte	✓	✓	✓
NFS	✓	✓	✓
RBD	✓	✓	-
VsphereVolume	✓	-	- (works when pods are collocated)
PortworxVolume	✓	-	✓
ScaleIO	✓	✓	-
StorageOS	✓	-	-

PersistentVolumeClaim创建参数：

```
#kubectl  explain  PersistentVolumeClaim.

accessModes ：PVC 访问模式，#kubectl  explain  PersistentVolumeClaim.spec.volumeMode
  ReadWriteOnce – PVC只能被单个节点以读写权限挂载，RWO
  ReadOnlyMany – PVC以可以被多个节点挂载但是权限是只读的,ROX
  ReadWriteMany – PVC可以被多个节点是读写方式挂载使用,RWX

resources:  #定义PVC创建存储卷的空间大小

selector:   #标签选择器，选择要绑定的PV
  matchLabels #匹配标签名称
  matchExpressions #基于正则表达式匹配

volumeName #要绑定的PV名称
```

```
volumeMode #卷类型
    定义PVC使用的文件系统是块设备还是文件系统，默认为文件系统
```

7.3.1：实战案例之zookeeper集群：

基于PV和PVC作为后端存储，实现zookeeper集群

7.3.1.1：下载JDK镜像：

```
# docker pull elevy/slim_java:8
# docker tag elevy/slim_java:8  harbor.magedu.net/linux36/harbor.magedu.net/slim_java:8
# docker push harbor.magedu.net/linux36/harbor.magedu.net/slim_java:8
```

7.3.1.2：zookeeper镜像准备：

https://www.apache.org/dist/zookeeper #官方程序包下载官方网址

7.3.1.2.1：构建zookeeper镜像：

```
# pwd
/opt/k8s-data/dockerfile/linux36/zookeeper

# chmod  a+x *.sh
# chmod  a+x bin/*.sh
# bash build-command.sh  2019-08-04_09_41_30
# docker push  harbor.magedu.net/linux36/zookeeper:2019-08-04_09_41_30
```

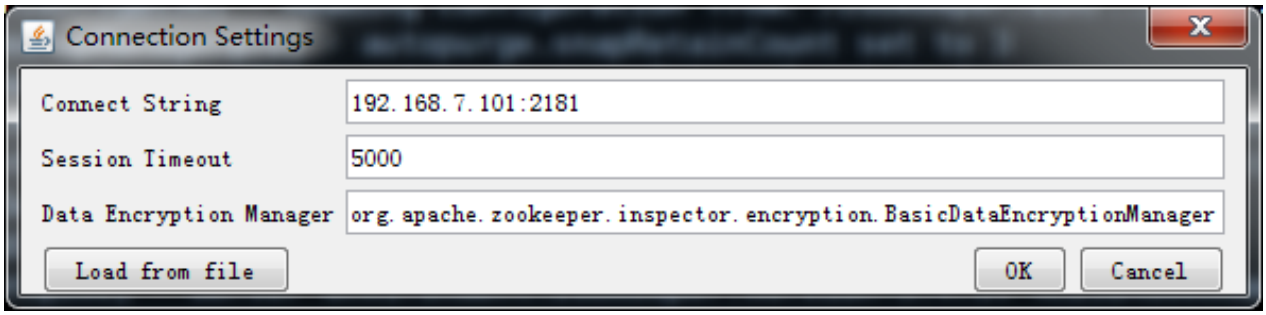
```
OK: 28 MiB in 33 packages
fetch http://dl-cdn.alpinelinux.org/alpine/v3.6/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.6/community/x86_64/APKINDEX.tar.gz
(1/2) Installing readline (6.3.008-r5)
(2/2) Installing bash (4.3.48-r1)
Executing bash-4.3.48-r1.post-install
Executing busybox-1.26.2-r5.trigger
Executing glibc-bin-2.26-r0.trigger
OK: 29 MiB in 35 packages
2019-08-04 01:51:12 URL:http://mirrors.tuna.tsinghua.edu.cn/apache/zookeeper/zookeeper-3.4.14/zookeeper-3.4.14.tar.gz [37676320/37676320] -> "/tmp/zk.tgz" [1]
2019-08-04 01:51:14 URL:https://www.apache.org/dist/zookeeper/zookeeper-3.4.14/zookeeper-3.4.14.tar.gz.asc [836/836] -> "/tmp/zk.tgz.asc" [1]
2019-08-04 01:51:16 URL:https://dist.apache.org/repos/dist/release/zookeeper/KEYS [55198/55198] -> "/tmp/KEYS" [1]
gpg: Signature made Wed Mar  6 18:47:14 2019 UTC
gpg:                using RSA key FFE35B7F15DFA1BA
gpg: Good signature from "Andor Molnar <andor@apache.org>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 3F7A 1D16 FA42 17B1 DC75  E1C9 FFE3 5B7F 15DF A1BA
WARNING: Ignoring APKINDEX.84815163.tar.gz: No such file or directory
WARNING: Ignoring APKINDEX.24d64ab1.tar.gz: No such file or directory
(1/16) Purging .build-deps (0)
(2/16) Purging ca-certificates (20161130-r3)
Executing ca-certificates-20161130-r3.post-deinstall
(3/16) Purging gnupg (2.1.20-r1)
(4/16) Purging pinentry (1.0.0-r0)
Executing pinentry-1.0.0-r0.post-deinstall
(5/16) Purging tar (1.32-r0)
(6/16) Purging wget (1.20.3-r0)
(7/16) Purging libksba (1.3.4-r0)
(8/16) Purging libassuan (2.4.3-r0)
(9/16) Purging libgcrypt (1.7.10-r0)
(10/16) Purging libgpg-error (1.27-r0)
```

7.3.1.2.2：测试zookeeper镜像：

```
# docker run -it --rm  -p 2181:2181 harbor.magedu.net/linux36/zookeeper:2019-08-
04_09_41_30
```

```
2019-08-04 01:52:36,085 [myid:] - INFO  [main:Environment@100] - Server environment:host.name=b6b16c4d083e
2019-08-04 01:52:36,085 [myid:] - INFO  [main:Environment@100] - Server environment:java.version=1.8.0_144
2019-08-04 01:52:36,085 [myid:] - INFO  [main:Environment@100] - Server environment:java.vendor=Oracle Corporation
2019-08-04 01:52:36,086 [myid:] - INFO  [main:Environment@100] - Server environment:java.home=/usr/lib/jvm/java-8-oracle
2019-08-04 01:52:36,086 [myid:] - INFO  [main:Environment@100] - Server environment:java.class.path=/zookeeper/bin/../zookeeper-server/target/classes:/zookeeper/bin/../build/
classes:/zookeeper/bin/../zookeeper-server/target/lib/*.jar:/zookeeper/bin/../build/lib/*.jar:/zookeeper/bin/../lib/slf4j-log4j12-1.7.25.jar:/zookeeper/bin/../lib/slf4j-api-1
.7.25.jar:/zookeeper/bin/../lib/netty-3.10.6.Final.jar:/zookeeper/bin/../lib/log4j-1.2.17.jar:/zookeeper/bin/../lib/jline-0.9.94.jar:/zookeeper/bin/../lib/audience-annotation
s-0.5.0.jar:/zookeeper/bin/../zookeeper-3.4.14.jar:/zookeeper/bin/../zookeeper-server/src/main/resources/lib/*.jar:/zookeeper/bin/../conf:
2019-08-04 01:52:36,087 [myid:] - INFO  [main:Environment@100] - Server environment:java.library.path=/usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/usr/lib
2019-08-04 01:52:36,087 [myid:] - INFO  [main:Environment@100] - Server environment:java.io.tmpdir=/tmp
2019-08-04 01:52:36,089 [myid:] - INFO  [main:Environment@100] - Server environment:java.compiler=<NA>
2019-08-04 01:52:36,089 [myid:] - INFO  [main:Environment@100] - Server environment:os.name=Linux
2019-08-04 01:52:36,090 [myid:] - INFO  [main:Environment@100] - Server environment:os.arch=amd64
2019-08-04 01:52:36,090 [myid:] - INFO  [main:Environment@100] - Server environment:os.version=4.15.0-54-generic
2019-08-04 01:52:36,091 [myid:] - INFO  [main:Environment@100] - Server environment:user.name=root
2019-08-04 01:52:36,091 [myid:] - INFO  [main:Environment@100] - Server environment:user.home=/root
2019-08-04 01:52:36,092 [myid:] - INFO  [main:Environment@100] - Server environment:user.dir=/zookeeper
2019-08-04 01:52:36,097 [myid:] - INFO  [main:ZooKeeperServer@836] - tickTime set to 2000
2019-08-04 01:52:36,097 [myid:] - INFO  [main:ZooKeeperServer@845] - minSessionTimeout set to -1
2019-08-04 01:52:36,097 [myid:] - INFO  [main:ZooKeeperServer@854] - maxSessionTimeout set to -1
2019-08-04 01:52:36,154 [myid:] - INFO  [main:ServerCnxnFactory@117] - Using org.apache.zookeeper.server.NIOServerCnxnFactory as server connection factory
2019-08-04 01:52:36,170 [myid:] - INFO  [main:NIOServerCnxnFactory@89] - binding to port 0.0.0.0/0.0.0.0:2181
```

7.3.1.2.3：测试客户端连接zookeeper：




```
2019-08-04 01:52:36,085 [myid:] - INFO  [main:Environment@100] - Server environment:host.name=b6b16c4d083e
2019-08-04 01:52:36,085 [myid:] - INFO  [main:Environment@100] - Server environment:java.version=1.8.0_144
2019-08-04 01:52:36,085 [myid:] - INFO  [main:Environment@100] - Server environment:java.vendor=Oracle Corporation
2019-08-04 01:52:36,086 [myid:] - INFO  [main:Environment@100] - Server environment:java.home=/usr/lib/jvm/java-8-oracle
2019-08-04 01:52:36,086 [myid:] - INFO  [main:Environment@100] - Server environment:java.class.path=/zookeeper/bin/../zookeeper-server/target/classes:/zookeeper/bin/../build/
classes:/zookeeper/bin/../zookeeper-server/target/lib/*.jar:/zookeeper/bin/../build/lib/*.jar:/zookeeper/bin/../lib/slf4j-log4j12-1.7.25.jar:/zookeeper/bin/../lib/slf4j-api-1
.7.25.jar:/zookeeper/bin/../lib/netty-3.10.6.Final.jar:/zookeeper/bin/../lib/log4j-1.2.17.jar:/zookeeper/bin/../lib/jline-0.9.94.jar:/zookeeper/bin/../lib/audience-annotation
s-0.5.0.jar:/zookeeper/bin/../zookeeper-3.4.14.jar:/zookeeper/bin/../zookeeper-server/src/main/resources/lib/*.jar:/zookeeper/bin/../conf:
2019-08-04 01:52:36,087 [myid:] - INFO  [main:Environment@100] - Server environment:java.library.path=/usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/usr/lib
2019-08-04 01:52:36,087 [myid:] - INFO  [main:Environment@100] - Server environment:java.io.tmpdir=/tmp
2019-08-04 01:52:36,089 [myid:] - INFO  [main:Environment@100] - Server environment:java.compiler=<NA>
2019-08-04 01:52:36,089 [myid:] - INFO  [main:Environment@100] - Server environment:os.name=Linux
2019-08-04 01:52:36,090 [myid:] - INFO  [main:Environment@100] - Server environment:os.arch=amd64
2019-08-04 01:52:36,090 [myid:] - INFO  [main:Environment@100] - Server environment:os.version=4.15.0-54-generic
2019-08-04 01:52:36,091 [myid:] - INFO  [main:Environment@100] - Server environment:user.name=root
2019-08-04 01:52:36,091 [myid:] - INFO  [main:Environment@100] - Server environment:user.home=/root
2019-08-04 01:52:36,092 [myid:] - INFO  [main:Environment@100] - Server environment:user.dir=/zookeeper
2019-08-04 01:52:36,097 [myid:] - INFO  [main:ZooKeeperServer@836] - tickTime set to 2000
2019-08-04 01:52:36,097 [myid:] - INFO  [main:ZooKeeperServer@845] - minSessionTimeout set to -1
2019-08-04 01:52:36,097 [myid:] - INFO  [main:ZooKeeperServer@854] - maxSessionTimeout set to -1
2019-08-04 01:52:36,154 [myid:] - INFO  [main:ServerCnxnFactory@117] - Using org.apache.zookeeper.server.NIOServerCnxnFactory as server connection factory
2019-08-04 01:52:36,170 [myid:] - INFO  [main:NIOServerCnxnFactory@89] - binding to port 0.0.0.0/0.0.0.0:2181
```


7.3.1.3：k8s 运行zookeeper服务：

通过yaml文件将zookeeper集群服务运行k8s环境

7.3.1.3.1：yaml文件准备：

yaml目录结构

```
# pwd
/opt/k8s-data/yaml/linux36/zookeeper

# tree
.
├── pv
│   ├── zookeeper-persistentvolumeclaim.yaml
│   └── zookeeper-persistentvolume.yaml
└── zookeeper.yaml

1 directory, 3 files
```

7.3.1.3.2：创建PV：

```
# pwd
/opt/k8s-data/yaml/linux36/zookeeper/pv

# kubectl  apply  -f  zookeeper-persistentvolume.yaml
persistentvolume/zookeeper-datadir-pv-1 created
persistentvolume/zookeeper-datadir-pv-2 created
persistentvolume/zookeeper-datadir-pv-3 created
```

7.3.1.3.3：验证PV：

```
root@k8s-master1:/opt/k8s-data/yaml/linux36/zookeeper/pv# kubectl  get pv
NAME                    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                              STORAGECLASS   REASON   AGE
zookeeper-datadir-pv-1   20Gi       RWO            Retain           Bound    linux36/zookeeper-datadir-pvc-1                           3m30s
zookeeper-datadir-pv-2   20Gi       RWO            Retain           Bound    linux36/zookeeper-datadir-pvc-2                           3m30s
zookeeper-datadir-pv-3   20Gi       RWO            Retain           Bound    linux36/zookeeper-datadir-pvc-3                           3m30s
root@k8s-master1:/opt/k8s-data/yaml/linux36/zookeeper/pv#
```

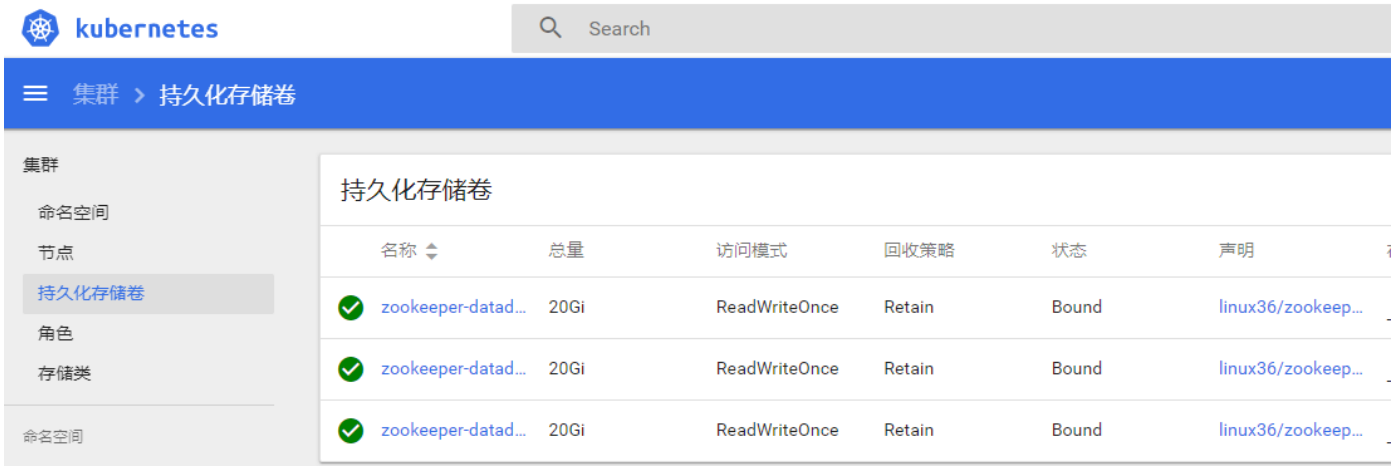
7.3.1.3.4：创建PVC：

```
# kubectl  apply  -f  zookeeper-persistentvolumeclaim.yaml
persistentvolumeclaim/zookeeper-datadir-pvc-1 created
persistentvolumeclaim/zookeeper-datadir-pvc-2 created
persistentvolumeclaim/zookeeper-datadir-pvc-3 created
```

7.3.1.3.5：验证PVC：

```
root@k8s-master1:/opt/k8s-data/yaml/linux36/zookeeper/pv# kubectl  get pvc -n linux36
NAME                     STATUS   VOLUME                   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
zookeeper-datadir-pvc-1   Bound    zookeeper-datadir-pv-1   20Gi       RWO                           22s
zookeeper-datadir-pvc-2   Bound    zookeeper-datadir-pv-2   20Gi       RWO                           22s
zookeeper-datadir-pvc-3   Bound    zookeeper-datadir-pv-3   20Gi       RWO                           22s
root@k8s-master1:/opt/k8s-data/yaml/linux36/zookeeper/pv#
```

7.3.1.3.6：dashborad 验证存储卷：



7.3.1.3.7：运行zookeeper集群：

```
# pwd
/opt/k8s-data/yaml/linux36/zookeeper

# kubectl  apply  -f zookeeper.yaml
service/zookeeper created
service/zookeeper1 created
service/zookeeper2 created
service/zookeeper3 created
deployment.extensions/zookeeper1 created
deployment.extensions/zookeeper2 created
deployment.extensions/zookeeper3 created
```

7.3.1.3.8：验证zookeeper集群：

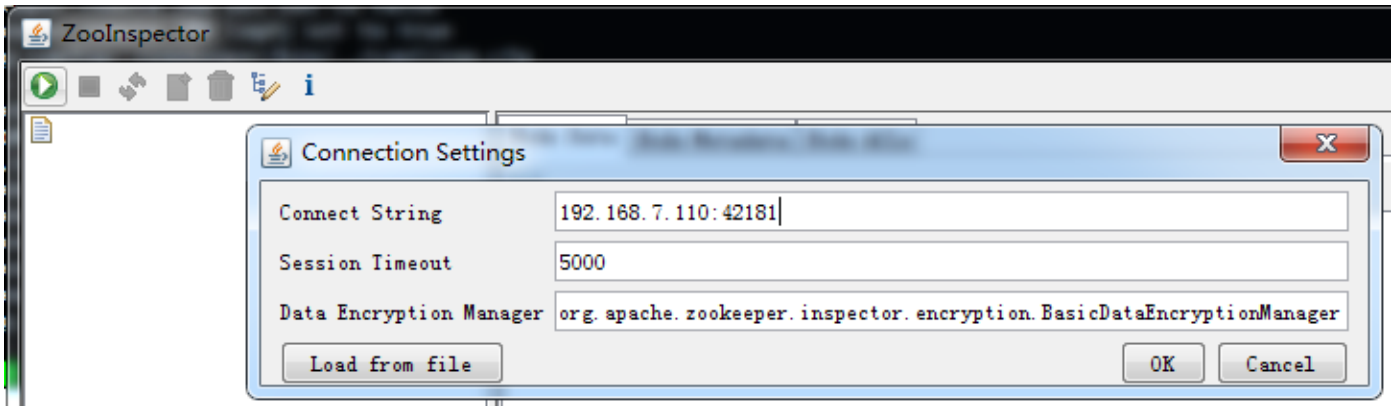
```
# kubectl  get pod -n linux36
NAME                                          READY   STATUS    RESTARTS   AGE
。。。。。。。。。。。。
zookeeper1-5f458745bf-lxsww                   1/1     Running   0          2m18s
zookeeper2-5844958588-9gvjh                   1/1     Running   0          2m18s
zookeeper3-7b646949f-w52cr                    1/1     Running   0          2m18s

# kubectl  exec -it zookeeper2-5844958588-9gvjh  sh -n linux36
```

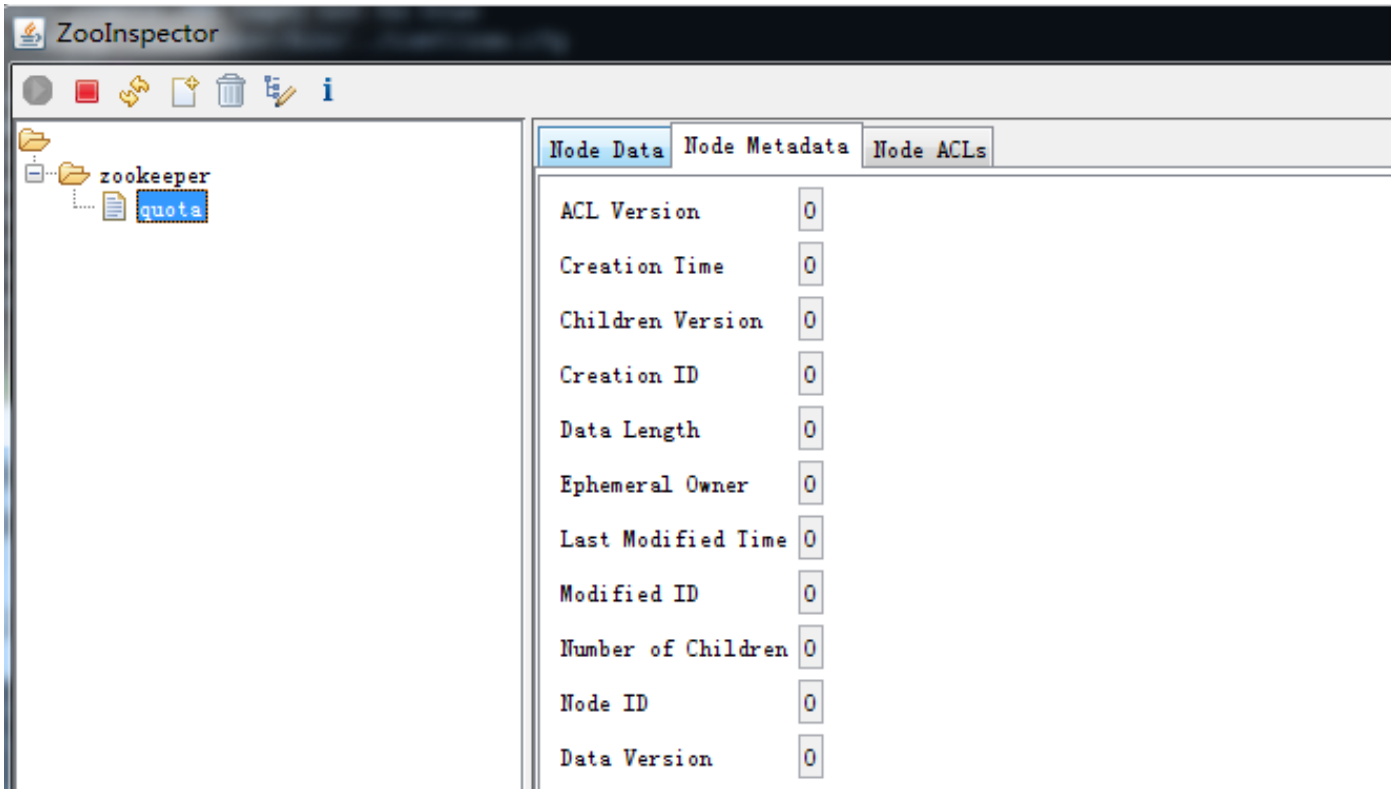
```
root@k8s-master1:/opt/k8s-data/yaml/linux36/zookeeper# kubectl  exec -it zookeeper1-5f458745bf-lxsww sh -n linux36
/ # /zookeeper/bin/zkServer.sh  status
ZooKeeper JMX enabled by default
ZooKeeper remote JMX Port set to 9010
ZooKeeper remote JMX authenticate set to false
ZooKeeper remote JMX ssl set to false
ZooKeeper remote JMX log4j set to true
Using config: /zookeeper/bin/../conf/zoo.cfg
Mode: follower
/ # exit
root@k8s-master1:/opt/k8s-data/yaml/linux36/zookeeper# kubectl  exec -it zookeeper2-5844958588-9gvjh sh -n linux36
/ # /zookeeper/bin/zkServer.sh  status
ZooKeeper JMX enabled by default
ZooKeeper remote JMX Port set to 9010
ZooKeeper remote JMX authenticate set to false
ZooKeeper remote JMX ssl set to false
ZooKeeper remote JMX log4j set to true
Using config: /zookeeper/bin/../conf/zoo.cfg
Mode: leader
/ # exit
root@k8s-master1:/opt/k8s-data/yaml/linux36/zookeeper# kubectl  exec -it zookeeper3-7b646949f-r4f2k  sh -n linux36
/ # /zookeeper/bin/zkServer.sh  status
ZooKeeper JMX enabled by default
ZooKeeper remote JMX Port set to 9010
ZooKeeper remote JMX authenticate set to false
ZooKeeper remote JMX ssl set to false
ZooKeeper remote JMX log4j set to true
Using config: /zookeeper/bin/../conf/zoo.cfg
Mode: follower
/ #
```

进入到容器验证zookeeper集群是否正常

删除期中一个pod，比如删除leader，验证在其余两个pod是否能自动 选举出新的leader，然后验证删除的pod重建后是否以follower的身份加入到zookeeper集群中。

7.3.1.3.9：验证从外部访问zookeeper：





7.3.2：实战案例之Redis 服务：

在k8s环境中运行redis服务

7.3.2.1：构建redis镜像：

7.3.2.1.1：镜像文件：

```
# pwd
/opt/k8s-data/dockerfile/linux36/redis

# tree
.
├── build-command.sh
├── Dockerfile
├── redis-4.0.14.tar.gz
├── redis.conf
└── run_redis.sh

0 directories, 5 file
```

7.3.2.1.2：构建镜像：

```
# chmod  a+x *.sh
# bash build-command.sh  2019-08-04_13-30-30
```

```
make[1]: Leaving directory `/usr/local/src/redis-4.0.14/src'
mkdir: created directory '/data'
mkdir: created directory '/data/redis-data'
Removing intermediate container 1167074f8736
 ---> 03abb621792a
Step 5/8 : ADD redis.conf /usr/local/redis/redis.conf
 ---> c0e8ea5b3790
Step 6/8 : ADD run_redis.sh /usr/local/redis/run_redis.sh
 ---> f06c9e68f9a0
Step 7/8 : EXPOSE 6379
 ---> Running in 99ba933c0359
Removing intermediate container 99ba933c0359
 ---> dccb14bece2a
Step 8/8 : CMD ["/usr/local/redis/run_redis.sh"]
 ---> Running in c85fa4b01110
Removing intermediate container c85fa4b01110
 ---> b44648090f0c
Successfully built b44648090f0c
Successfully tagged harbor.magedu.net/linux36/redis:2019-08-04_13-30-30
The push refers to repository [harbor.magedu.net/linux36/redis]
ee702de1add1: Pushed
82caf7c25eea: Pushed
e28cc56d2869: Pushed
```

7.3.2.1.3：测试redis 镜像：

```
#docker run -it --rm  -p6379:6379 harbor.magedu.net/linux36/redis:2019-08-04_13-30-30
```

7.3.2.2：运行redis服务：

基于PV/PVC保存数据，实现k8s中运行Redis服务

7.3.2.2.1：创建PV与PVC:

```
# pwd
/opt/k8s-data/yaml/linux36/redis/pv

# kubectl  apply  -f .
persistentvolume/redis-datadir-pv-1 created
persistentvolumeclaim/redis-datadir-pvc-1 created
```

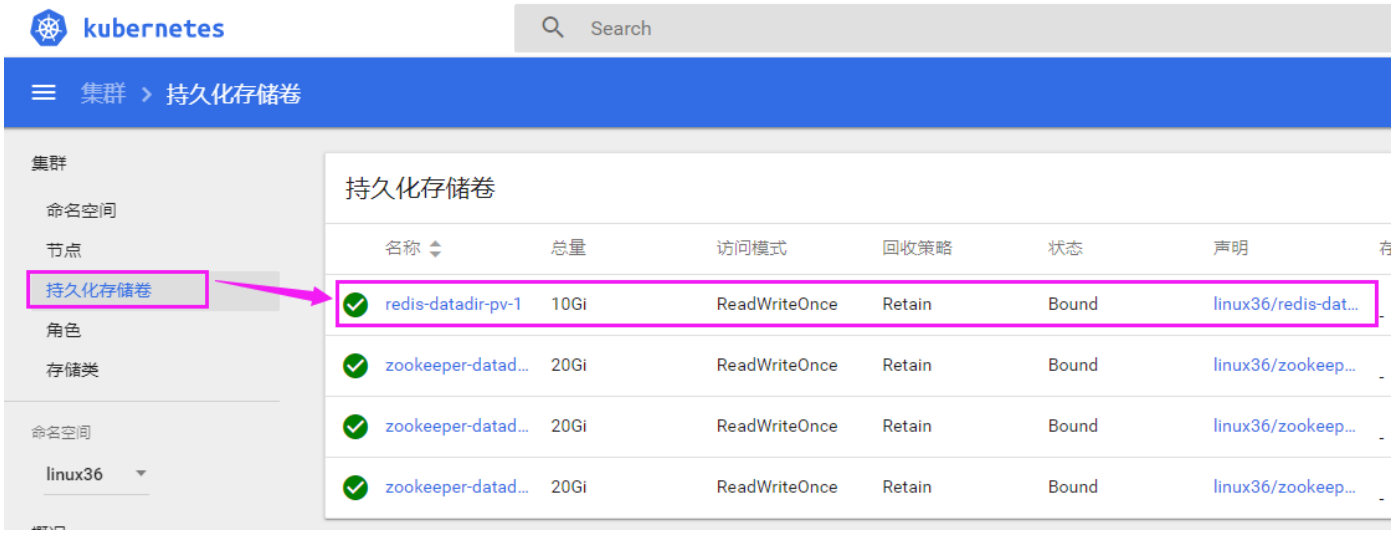
7.3.2.2.2：验证PV与PVC:

验证redis的PV与PVC已经处于bond状态

命令行验证：

```
root@k8s-master1:/opt/k8s-data/yaml/linux36/redis/pv# kubectl  get pv -n linux36
NAME                  CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                              STORAGECLASS   REASON   AGE
redis-datadir-pv-1    10Gi       RWO            Retain           Bound    linux36/redis-datadir-pvc-1                                 58s
zookeeper-datadir-pv-1   20Gi       RWO            Retain           Bound    linux36/zookeeper-datadir-pvc-1                             178m
zookeeper-datadir-pv-2   20Gi       RWO            Retain           Bound    linux36/zookeeper-datadir-pvc-2                             178m
zookeeper-datadir-pv-3   20Gi       RWO            Retain           Bound    linux36/zookeeper-datadir-pvc-3                             178m
root@k8s-master1:/opt/k8s-data/yaml/linux36/redis/pv#
root@k8s-master1:/opt/k8s-data/yaml/linux36/redis/pv# kubectl  get pvc -n linux36
NAME                  STATUS   VOLUME                  CAPACITY   ACCESS MODES   STORAGECLASS   AGE
redis-datadir-pvc-1   Bound    redis-datadir-pv-1      10Gi       RWO                           63s
zookeeper-datadir-pvc-1   Bound    zookeeper-datadir-pv-1   20Gi       RWO                           176m
zookeeper-datadir-pvc-2   Bound    zookeeper-datadir-pv-2   20Gi       RWO                           176m
zookeeper-datadir-pvc-3   Bound    zookeeper-datadir-pv-3   20Gi       RWO                           176m
root@k8s-master1:/opt/k8s-data/yaml/linux36/redis/pv#
```

dashboard验证：



7.3.1.2.3：运行Redis服务:

```
# pwd
/opt/k8s-data/yaml/linux36/redis

# tree
.
├── pv
│   ├── redis-persistentvolumeclaim.yaml
│   └── redis-persistentvolume.yaml
└── redis.yaml

1 directory, 3 files

# kubectl  apply  -f redis.yaml
deployment.extensions/deploy-devops-redis created
service/srv-devops-redis created
```

7.3.1.2.4：外部客户端访问redis：

```
# redis-cli  -h 192.168.7.110 -p 36379 -a 123456
192.168.7.110:36379> set key1 value1
OK
192.168.7.110:36379> set key2 value2
OK
```

```
root@k8s-harbor1:/usr/local/src/harbor# redis-cli  -h 192.168.7.110 -p 36379 -a 123456
192.168.7.110:36379> info
# Server
redis_version:4.0.14
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:d1ce9e1b29c005a4
redis_mode:standalone
os:Linux 4.15.0-54-generic x86_64
arch_bits:64
multiplexing_api:epoll
atomicvar_api:atomic-builtin
gcc_version:4.8.5
process_id:7
run_id:9a242777749e5e60ce35aa11efb44d0f8cd8dc25
tcp_port:6379
```

7.3.1.2.5：验证PVC存储卷数据：

验证nfs服务器Redis的快照数据：

```
root@k8s-ha1:~# ll  /data/k8sdata/linux36/redis-datadir-1
total 12
drwxr-xr-x 2 root root 4096 Aug  4 14:29 ./
drwxr-xr-x 8 root root 4096 Aug  4 14:09 ../
-rw-r--r-- 1 root root  124 Aug  4 14:29 dump.rdb
```

进入到容器里验证redis数据

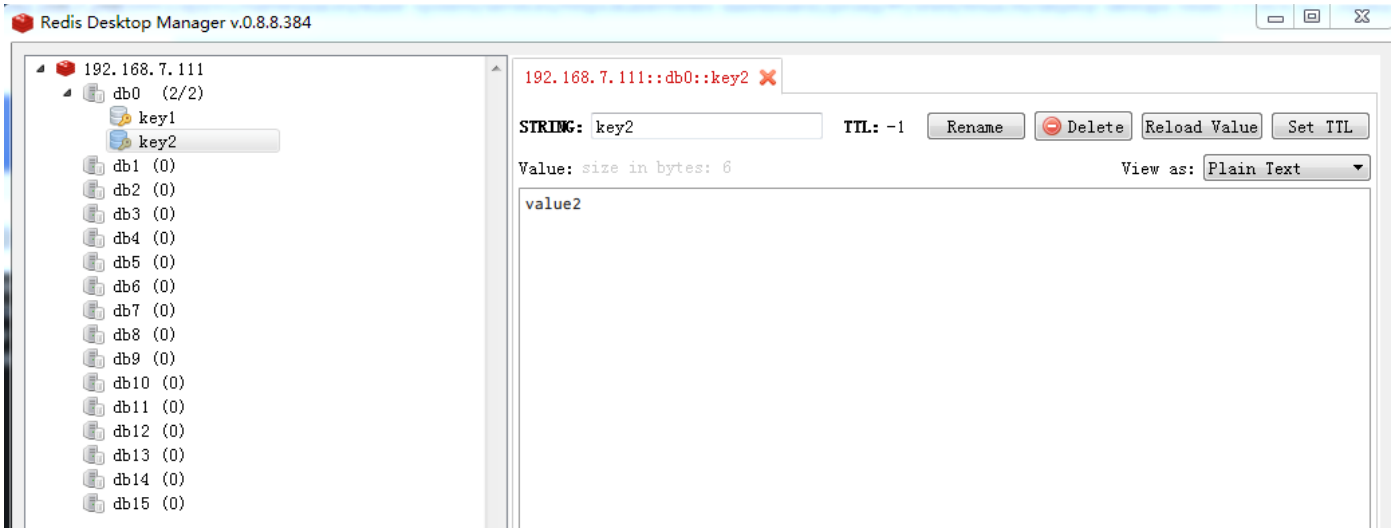
```
# ll /data/redis-data/
total 4
-rw-r--r-- 1 root root 124 Aug  4 14:29 dump.rdb
```

命令行 redis-container ▾ 在 deploy-devops-redis-775fcfc5c5-2q5kx

```
[root@deploy-devops-redis-775fcfc5c5-2q5kx /]# ll /data/redis-data/
total 4
-rw-r--r-- 1 root root 124 Aug  4 14:29 dump.rdb
[root@deploy-devops-redis-775fcfc5c5-2q5kx /]# df -TH
Filesystem                                          Type     Size  Used Avail Use% Mounted on
overlay                                             overlay  105G  6.0G   93G   7% /
tmpfs                                               tmpfs     68M     0   68M   0% /dev
tmpfs                                               tmpfs    2.1G     0  2.1G   0% /sys/fs/cgroup
192.168.7.108:/data/k8sdata/linux36/redis-datadir-1 nfs4     105G  3.8G   96G   4% /data/redis-data
/dev/mapper/ubuntu--vg-root                         ext4     105G  6.0G   93G   7% /etc/hosts
shm                                                 tmpfs     68M     0   68M   0% /dev/shm
tmpfs                                               tmpfs    2.1G   13k  2.1G   1% /run/secrets/kubernetes.io/serviceaccount
tmpfs                                               tmpfs    2.1G     0  2.1G   0% /proc/acpi
tmpfs                                               tmpfs    2.1G     0  2.1G   0% /proc/scsi
tmpfs                                               tmpfs    2.1G     0  2.1G   0% /sys/firmware
[root@deploy-devops-redis-775fcfc5c5-2q5kx /]#
```

7.3.2.2.6：验证Redis服务：

从k8s集群外的环境通过nodeport访问k8s环境中的的redis服务，或者从其他pod访问测试



7.3.2.2.7：验证Redis数据高可用：

删除redis的pod，然后重新创建pod验证新生成的pod中是否有之前的数据，可能有丢失数据的几率，取决于是否开启AOF或者dump数据的功能及设置：

```
# pwd
/opt/k8s-data/yaml/linux36/redis

# kubectl  delete -f  redis.yaml
deployment.extensions "deploy-devops-redis" deleted
service "srv-devops-redis" deleted

# kubectl  apply  -f  redis.yaml
deployment.extensions/deploy-devops-redis created
service/srv-devops-redis created
```

然后重新验证之前的key是也就是之前的数据是否还存在

7.3.3：实战案例之MySQL 主从架构：

https://kubernetes.io/zh/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume/

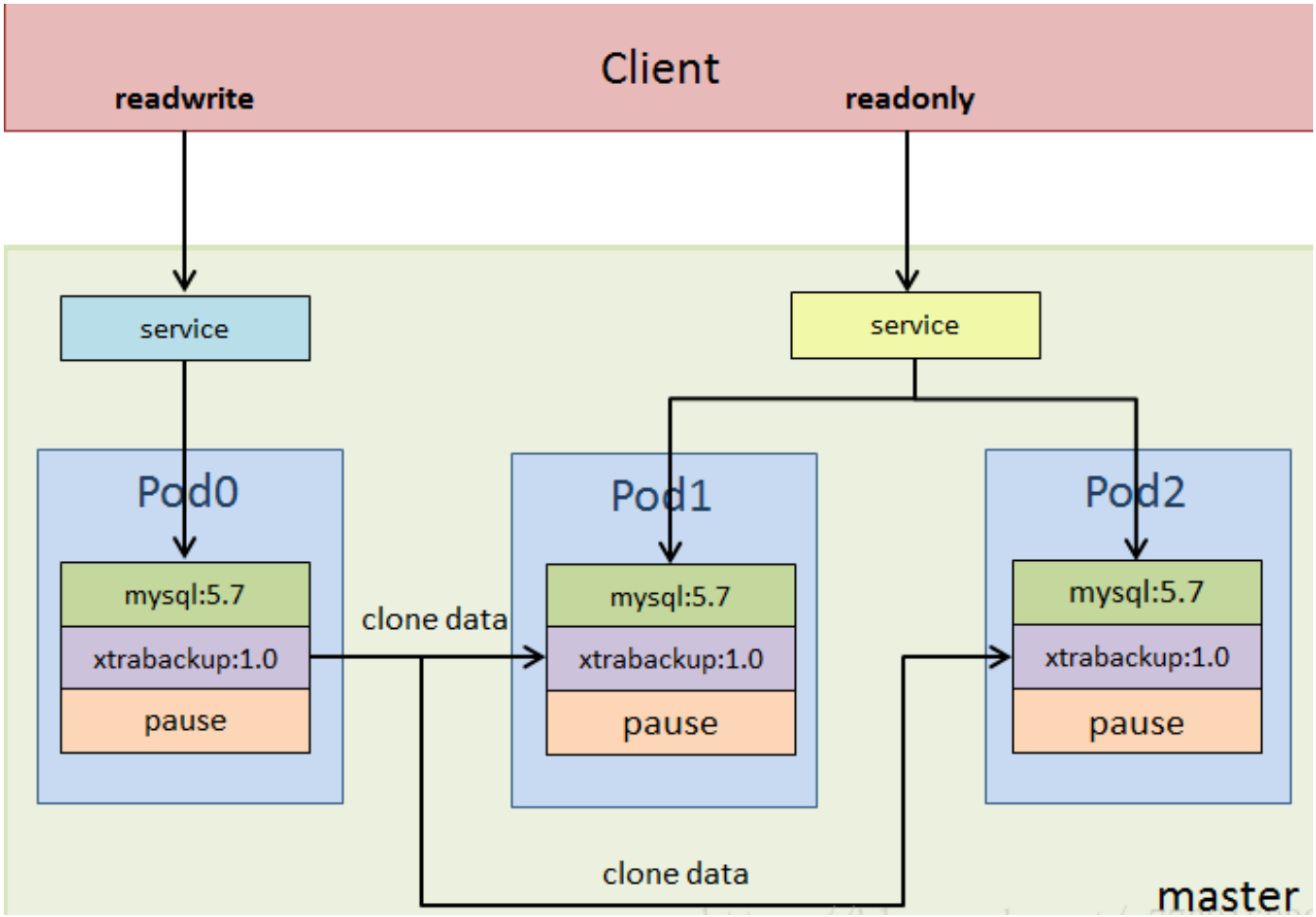
https://www.kubernetes.org.cn/statefulset

基于StatefulSet实现：

https://kubernetes.io/zh/docs/tasks/run-application/run-replicated-stateful-application/

Pod调度运行时，如果应用不需要任何稳定的标示、有序的部署、删除和扩展，则应该使用一组无状态副本的控制器来部署应用，例如 Deployment 或 ReplicaSet更适合无状态服务需求，而StatefulSet适合管理所有有状态的服务，比如MySQL、MongoDB集群等。

基于StatefulSet 实现的MySQL 一主多从架构



StatefulSet本质上是Deployment的一种变体，在v1.9版本中已成为GA版本，它为了解决有状态服务的问题，它所管理的Pod拥有固定的Pod名称，启停顺序，在StatefulSet中，Pod名字称为网络标识(hostname)，还必须要用到共享存储。
在Deployment中，与之对应的服务是service，而在StatefulSet中与之对应的headless service，headless service，即无头服务，与service的区别就是它没有Cluster IP，解析它的名称时将返回该Headless Service对应的全部Pod的Endpoint列表。

StatefulSet 特点：
-> 给每个pod分配固定且唯一的网络标识符
-> 给每个pod分配固定且持久化的外部存储
-> 对pod进行有序的部署和扩展
-> 对pod进有序的删除和终止
-> 对pod进有序的自动滚动更新

7.3.3.1：StatefulSet的组成部分：

Headless Service：用来定义Pod网络标识(DNS domain)，指的是短的serfvice(丢失了domainname)。
StatefulSet：定义具体应用，有多少个Pod副本，并为每个Pod定义了一个域名。
volumeClaimTemplates： 存储卷申请模板，创建PVC，指定pvc名称大小，将自动创建pvc，且pvc必须由存储类供应。

7.3.3.2：镜像准备：

https://github.com/docker-library/ #github下载地址

基础镜像准备：

```
#准备xtrabackup镜像
# docker pull registry.cn-hangzhou.aliyuncs.com/hxpdocker/xtrabackup:1.0
# docker tag registry.cn-hangzhou.aliyuncs.com/hxpdocker/xtrabackup:1.0
harbor.magedu.net/linux36xtrabackup:1.0
# docker push harbor.magedu.net/linux36/xtrabackup:1.0

#准备mysql 镜像
# docker pull mysql:5.7
# docker tag mysql:5.7 harbor.magedu.net/baseimages/mysql:5.7
# docker push harbor.magedu.net/baseimages/mysql:5.7
```

7.3.3.3：创建PV：

pvc会自动基于PV创建，只需要有多个可用的PV即可，PV数量取决于计划启动多少个mysql pod，本次创建5个PV，也就是最多启动5个mysql pod。

```
# pwd
/opt/k8s-data/yaml/linux36/mysql

# tree
.
├── mysql-configmap.yaml
├── mysql-services.yaml
├── mysql-statefulset.yaml
└── pv
    └── mysql-persistentvolume.yaml

1 directory, 4 files

#数据目录
~# mkdir mysql-datadir-1
~# mkdir mysql-datadir-2
~# mkdir mysql-datadir-3
~# mkdir mysql-datadir-4
~# mkdir mysql-datadir-5
~# mkdir mysql-datadir-clone

#创建PV
# kubectl  apply  -f pv/
persistentvolume/mysql-datadir-1 created
persistentvolume/mysql-datadir-2 created
persistentvolume/mysql-datadir-3 created
persistentvolume/mysql-datadir-4 created
```

```
persistentvolume/mysql-datadir-5 created
```

7.3.3.4：验证PV：

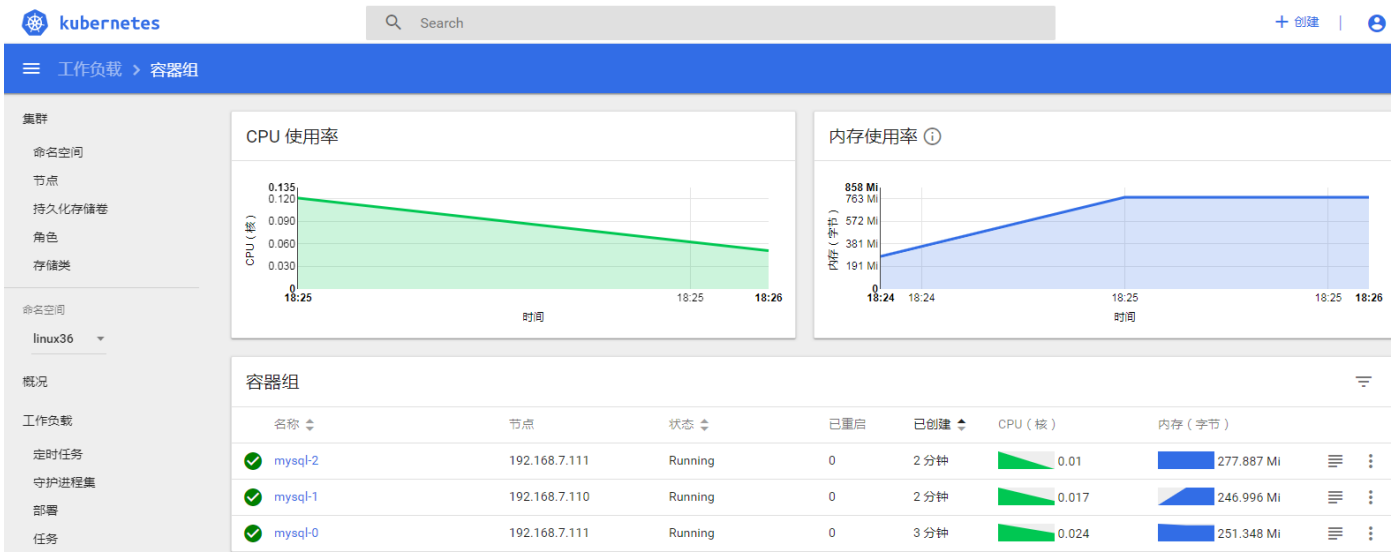
```
root@k8s-master1:/opt/k8s-data/yaml/linux36/mysql/pv# kubectl  get pv
NAME                  CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM                       STORAGECLASS   REASON   AGE
mysql-datadir-pv-1    100Gi      RWO            Retain           Available                                                    23s
mysql-datadir-pv-2    100Gi      RWO            Retain           Available                                                    23s
mysql-datadir-pv-3    100Gi      RWO            Retain           Available                                                    23s
mysql-datadir-pv-4    100Gi      RWO            Retain           Available                                                    23s
mysql-datadir-pv-5    100Gi      RWO            Retain           Available                                                    23s
```

7.3.3.5：运行mysql服务：

创建mysql pod：

```
# pwd
/opt/k8s-data/yaml/linux36/mysql

# ll
total 28
drwxr-xr-x 3 root root 4096 Aug  4 18:22 ./
drwxr-xr-x 7 root root 4096 Aug  4 18:19 ../
-rw-r--r-- 1 root root  364 Aug  4 17:34 mysql-configmap.yaml
-rw-r--r-- 1 root root  529 Aug  4 17:38 mysql-services.yaml
-rw-r--r-- 1 root root 5510 Aug  4 17:53 mysql-statefulset.yaml
drwxr-xr-x 2 root root 4096 Aug  4 18:20 pv/

# kubectl  create -f . -n linux36
configmap/mysql created
service/mysql created
service/mysql-read created
statefulset.apps/mysql created
```

7.3.3.6：验证MySQL Pod状态：



7.3.3.7：验证MySQL主从同步是否正常：

```
root@k8s-master1:/opt/k8s-data/yaml/linux36/mysql# kubectl  get pod -n linux36
NAME      READY   STATUS    RESTARTS   AGE
mysql-0   2/2     Running   0          2m57s
mysql-1   2/2     Running   0          12m
mysql-2   2/2     Running   0          12m
root@k8s-master1:/opt/k8s-data/yaml/linux36/mysql#
root@k8s-master1:/opt/k8s-data/yaml/linux36/mysql# kubectl  exec -it mysql-2 sh -n linux36
Defaulting container name to mysql.
Use 'kubectl describe pod/mysql-2 -n linux36' to see all of the containers in this pod.
# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 448
Server version: 5.7.27 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show slave status\G;
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: mysql-0.mysql
                  Master_User: root
                  Master_Port: 3306
                Connect_Retry: 10
              Master_Log_File: mysql-0-bin.000004
          Read_Master_Log_Pos: 154
               Relay_Log_File: mysql-2-relay-bin.000004
                Relay_Log_Pos: 371
        Relay_Master_Log_File: mysql-0-bin.000004
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
```

进入Pod验证MySQL主从同步是否正常

确认IO线程和SQL线程都为yes

7.3.3.8：高可用测试：

分别删除pod中的master与slave节点，验证MySQL服务最终能否恢复到正常状态。

7.3.3.8.1：删除MySQL Master：

```
# kubectl get pod -n linux36
NAME      READY   STATUS    RESTARTS   AGE
mysql-0   2/2     Running   0          8m57s
mysql-1   2/2     Running   0          18m
mysql-2   2/2     Running   0          18m

# kubectl  delete pod mysql-0 -n linux36
pod "mysql-0" deleted
```

```
root@k8s-master1:/opt/k8s-data/yaml/linux36/mysql# kubectl get pod -n linux36
NAME      READY   STATUS    RESTARTS   AGE
mysql-0   2/2     Running   0          8m57s
mysql-1   2/2     Running   0          18m
mysql-2   2/2     Running   0          18m
root@k8s-master1:/opt/k8s-data/yaml/linux36/mysql# kubectl  delete pod mysql-0 -n linux36
pod "mysql-0" deleted
root@k8s-master1:/opt/k8s-data/yaml/linux36/mysql# kubectl get pod -n linux36
NAME      READY   STATUS    RESTARTS   AGE
mysql-0   2/2     Running   0          14s
mysql-1   2/2     Running   0          19m
mysql-2   2/2     Running   0          19m
root@k8s-master1:/opt/k8s-data/yaml/linux36/mysql#
```

7.3.3.8.2：删除MySQL Slave：

```
# kubectl get pod -n linux36
NAME      READY   STATUS    RESTARTS   AGE
mysql-0   2/2     Running   0          14s
mysql-1   2/2     Running   0          19m
mysql-2   2/2     Running   0          19m

# kubectl  delete pod mysql-1 -n linux36
pod "mysql-1" deleted
```

```
root@k8s-master1:/opt/k8s-data/yaml/linux36/mysql# kubectl  delete pod mysql-1 -n linux36
pod "mysql-1" deleted
root@k8s-master1:/opt/k8s-data/yaml/linux36/mysql# kubectl get pod -n linux36
NAME      READY   STATUS    RESTARTS   AGE
mysql-0   2/2     Running   0          94s
mysql-1   2/2     Running   0          20s
mysql-2   2/2     Running   0          21m
root@k8s-master1:/opt/k8s-data/yaml/linux36/mysql#
```

7.3.4：运行java应用：

基于java命令，运行java war包或jar包，本次以jenkins.war 包部署方式为例，且要求jenkins的数据保存至外部存储(NFS或者PVC)，其他java应用看实际需求是否需要将数据保存至外部存储。

7.3.4.1：镜像目录文件：

```
# pwd
/opt/k8s-data/dockerfile/linux36/jenkins

# tree
.
├── build-command.sh
├── Dockerfile
├── jenkins-2.164.3.war
└── run_jenkins.sh

0 directories, 4 files
```

7.3.4.2：构建Jenkins镜像：

```
# bash build-command.sh
```

```
Step 6/6 : CMD ["/bin/run_jenkins.sh"]
 ---> Running in efc32aabcde6
Removing intermediate container efc32aabcde6
 ---> 966c8eb26d8f
Successfully built 966c8eb26d8f
Successfully tagged harbor.magedu.net/linux36/jenkins:v2.164.3
镜像制作完成，即将上传至Harbor服务器
The push refers to repository [harbor.magedu.net/linux36/jenkins]
0e3e975ab197: Pushed
74bc98d2ebed: Pushed
1562b98079f5: Mounted from linux36/tomcat-app1
942d70ea7025: Mounted from linux36/tomcat-app1
5884c2842480: Mounted from linux36/tomcat-app1
6ed64e45cdd3: Mounted from linux36/redis
68851c61210d: Mounted from linux36/redis
4551a1f33298: Mounted from linux36/redis
b071b2606076: Mounted from linux36/redis
d69483a6face: Mounted from linux36/redis
v2.164.3: digest: sha256:2c15147077ee441d04bd764f5099dcad227f16da5da6eaf4da1a36b723e8c00f size: 2421
镜像上传完成
root@k8s-master1:/opt/k8s-data/dockerfile/linux36/jenkins#
```

7.3.4.3：验证Jenkins镜像：

```
# docker run -it --rm  -p 8088:8080 harbor.magedu.net/linux36/jenkins:v2.164.3
```

```
root@k8s-master1:/opt/k8s-data/dockerfile/linux36/jenkins# docker run -it --rm  -p 8088:8080 harbor.magedu.net/linux36/jenkins:v2.164.3
Running from: /apps/jenkins/jenkins-2.164.3.war
Aug 04, 2019 6:56:15 PM org.eclipse.jetty.util.log.Log initialized
INFO: Logging initialized @779ms to org.eclipse.jetty.util.log.JavaUtilLog
Aug 04, 2019 6:56:15 PM winstone.Logger logInternal
INFO: Beginning extraction from war file
Aug 04, 2019 6:56:17 PM org.eclipse.jetty.server.handler.ContextHandler setContextPath
WARNING: Empty contextPath
Aug 04, 2019 6:56:17 PM org.eclipse.jetty.server.Server doStart
INFO: jetty-9.4.z-SNAPSHOT; built: 2018-08-30T13:59:14.071Z; git: 27208684755d94a92186989f695db2d7b21ebc51; jvm 1.8.0_212-b10
Aug 04, 2019 6:56:18 PM org.eclipse.jetty.webapp.StandardDescriptorProcessor visitServlet
INFO: NO JSP Support for /, did not find org.eclipse.jetty.jsp.JettyJspServlet
Aug 04, 2019 6:56:18 PM org.eclipse.jetty.server.session.DefaultSessionIdManager doStart
INFO: DefaultSessionIdManager workerName=node0
Aug 04, 2019 6:56:18 PM org.eclipse.jetty.server.session.DefaultSessionIdManager doStart
INFO: No SessionScavenger set, using defaults
Aug 04, 2019 6:56:18 PM org.eclipse.jetty.server.session.HouseKeeper startScavenging
INFO: node0 Scavenging every 660000ms
Jenkins home directory: /root/.jenkins found at: $user.home/.jenkins
Aug 04, 2019 6:56:19 PM org.eclipse.jetty.server.handler.ContextHandler doStart
INFO: Started w.@2f2bf0e2{Jenkins v2.164.3,/,file:///apps/jenkins/jenkins-data/,AVAILABLE}{/apps/jenkins/jenkins-data}
Aug 04, 2019 6:56:19 PM org.eclipse.jetty.server.AbstractConnector doStart
INFO: Started ServerConnector@16b062e2{HTTP/1.1,[http/1.1]}{0.0.0.0:8080}
Aug 04, 2019 6:56:19 PM org.eclipse.jetty.server.Server doStart
```

7.3.4.4：创建PV/PVC：

需要两个PVC，一个保存jenkins的数据，一个保存.jenkins的数据。

```
root@k8s-ha1:/data/linux36# ll  /data/k8sdata/linux36/jenkins-data
root@k8s-ha1:/data/linux36# ll /data/k8sdata/linux36/jenkins-root-data

# kubectl  apply  -f pv/
persistentvolume/jenkins-datadir-pv created
persistentvolume/jenkins-root-datadir-pv created
persistentvolumeclaim/jenkins-datadir-pvc created
persistentvolumeclaim/jenkins-root-data-pvc created
```

7.3.4.5：验证PV/PVC：

```
# kubectl  get pv -n linux36
NAME CAPACITY ACCESS MODES RECLAIM POLICY STATUS  CLAIM STORAGECLASS   REASON   AGE
jenkins-datadir-pv 100Gi   RWO   Retain    Bound  linux36/jenkins-datadir-pvc    15s
jenkins-root-datadir-pv 100Gi RWO  Retain  Bound  linux36/jenkins-root-data-pvc  14s

root@k8s-master1:/opt/k8s-data/yaml/linux36/jenkins# kubectl  get pvc  -n linux36
NAME           STATUS  VOLUME      CAPACITY   ACCESS MODES   STORAGECLASS   AGE
jenkins-datadir-pvc     Bound    jenkins-datadir-pv        100Gi      RWO      28s
jenkins-root-data-pvc   Bound    jenkins-root-datadir-pv   100Gi      RWO     28s
```

7.3.4.6：yaml文件：

```
# pwd
/opt/k8s-data/yaml/linux36/jenkins
root@k8s-master1:/opt/k8s-data/yaml/linux36/jenkins# tree
.
├── jenkins.yaml
└── pv
    ├── jenkins-persistentvolumeclaim.yaml
    └── jenkins-persistentvolume.yaml

1 directory, 3 files
```

7.3.4.7：k8s运行jenkins服务：

```
# pwd
/opt/k8s-data/yaml/linux36/jenkins

# kubectl  apply  -f jenkins.yaml
deployment.extensions/linux36-jenkins-deployment created
service/linux36-jenkins-service created
```

7.3.4.8：验证pod：

```
# kubectl  get pods -n  linux36
NAME                                          READY   STATUS    RESTARTS   AGE
linux36-jenkins-deployment-74c4b55576-ssxdv   1/1     Running   0          50s

# kubectl  describe pods linux36-jenkins-deployment-74c4b55576-ssxdv  -n linux36
Events:
  Type    Reason     Age   From                     Message
  ----    ------     ----  ----                     -------
  Normal  Scheduled  60s   default-scheduler        Successfully assigned
linux36/linux36-jenkins-deployment-74c4b55576-ssxdv to 192.168.7.111
  Normal  Pulling    58s   kubelet, 192.168.7.111  pulling image
"harbor.magedu.net/linux36/jenkins:v2.164.3"
  Normal  Pulled     55s   kubelet, 192.168.7.111  Successfully pulled image
"harbor.magedu.net/linux36/jenkins:v2.164.3"
  Normal  Created    54s   kubelet, 192.168.7.111  Created container
  Normal  Started    54s   kubelet, 192.168.7.111  Started container
```

7.3.4.9：验证web访问jenkins：

7.3.5：k8s 示例之WordPress:

LNMP案例之基于Nginx+PHP实现WordPress博客站点，要求Nginx+PHP运行在同一个Pod的不同容器，MySQL运行与default的namespace并可以通过service name增删改查数据库。

https://cn.wordpress.org/

https://cn.wordpress.org/download/releases/

wordpress-5.0.2-zh_CN.tar.gz

7.3.5.1：准备PHP镜像：

7.3.5.1.1：官方PHP镜像：

https://hub.docker.com/

```
# docker pull php:5.6.40-fpm
# docker tag php:5.6.40-fpm  harbor.magedu.net/linux36/php:5.6.40-fpm
# docker push harbor.magedu.net/linux36/php:5.6.40-fpm
```

7.3.5.1.2：自制PHP镜像：

```
# pwd
/opt/k8s-data/dockerfile/linux36/wordpress/php
# tree
.
├── build-command.sh
├── Dockerfile
├── run_php.sh
└── www.conf

0 directories, 4 files

# bash build-command.sh  v1
```

```
Successfully built a65cf15c02be
Successfully tagged harbor.magedu.net/linux36/wordpress-php-5.6:v1
镜像制作完成，即将上传至Harbor服务器
The push refers to repository [harbor.magedu.net/linux36/wordpress-php-5.6]
71ecbf3bc7db: Pushed
6619cb09a873: Pushed
ca56761ddf59: Pushed
6ed64e45cdd3: Mounted from linux36/jenkins
68851c61210d: Mounted from linux36/jenkins
4551a1f33298: Mounted from linux36/jenkins
b071b2606076: Mounted from linux36/jenkins
d69483a6face: Mounted from linux36/jenkins
v1: digest: sha256:0b35f7fa7f2933a0f21f60e50bb1934c251879d59970c33fc9803dd2d6e50e5b size: 2000
镜像上传完成
root@k8s-master1:/opt/k8s-data/dockerfile/linux36/wordpress/php#
```

7.3.5.2：准备Nginx镜像：

```
# pwd
/opt/k8s-data/dockerfile/linux36/wordpress/nginx

# tree
.
├── build-command.sh
├── Dockerfile
├── index.html
├── nginx.conf
└── run_nginx.sh

0 directories, 5 files

# bash build-command.sh  v1
```

7.3.5.3：运行WordPress站点：

使用官方PHP镜像运行PHP环境，WordPress页面文件保存在后端存储NFS服务器。

7.3.5.3.1：运行WordPress：

```
# pwd
/opt/k8s-data/yaml/linux36/wordpress

# kubectl  apply -f .
deployment.extensions/wordpress-app-deployment created
service/wordpress-app-spec created
```

7.3.5.3.2：创建PHP测试页：

准备WordPress页面文件并更改权限为指定用户

```
# pwd
/data/k8sdata/linux36/wordpress

# cat test.php
<?php
     phpinfo();
?>

linux36# chown  2001.2001 wordpress/ -R
```

7.3.5.3.3：访问PHP测试页：

192.168.7.110:30031/test.php

PHP Version 5.6.40

System	Linux wordpress-app-deployment-ff5775d99-szl62 4.15.0-54-generic #58-Ubuntu SMP Mon Jun 24 10:55:24 UTC 2019 x86_64
Build Date	Jan 23 2019 00:14:48
Configure Command	'./configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-curl' '--with-libedit' '--with-openssl' '--with-zlib' '--with-libdir=lib/x86_64-linux-gnu' '--enable-fpm' '--with-fpm-user=www-data' '--with-fpm-group=www-data' '--disable-cgi' 'build_alias=x86_64-linux-gnu' 'CFLAGS=-fstack-protector-strong '-fpic' '-fpie' '-O2'' 'LDFLAGS=-Wl,-O1 '-Wl,--hash-style=both' '-pie'' 'CPPFLAGS=-fstack-protector-strong '-fpic' '-fpie' '-O2''
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	(none)
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	(none)
PHP API	20131106
PHP Extension	20131226
Zend Extension	220131226
Zend Extension Build	API220131226,NTS
PHP Extension Build	API20131226,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled

7.3.5.4：初始化WordPress站点：

使用k8s中运行的mysql服务，作为mysql服务器

7.3.5.4.1：k8s中MySQL创建数据库：

```
# kubectl  exec -it mysql-0 sh -n linux36
.....
mysql> CREATE DATABASE wordpress;
Query OK, 1 row affected (0.01 sec)

mysql>  GRANT ALL PRIVILEGES ON wordpress.* TO "wordpress"@"%"  IDENTIFIED BY
"wordpress";
Query OK, 0 rows affected, 1 warning (0.01 sec)
```

7.3.5.4.2：k8s中测试MySQL连接：

```
# mysql -uwordpress -hmysql-0.mysql  -pwordpress
```

```
[root@linux36-jenkins-deployment-74c4b55576-ssxdv /]# mysql -uwordpress -hmysql-0.mysql  -pwordpress
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 3735
Server version: 5.7.27-log MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| wordpress          |
+--------------------+
2 rows in set (0.03 sec)

MySQL [(none)]>
```

7.3.5.4.3：通过web界面初始化数据库：

192.168.7.110:30031/wp-admin/setup-config.php?step=1&language=zh_CN

请在下方填写您的数据库连接信息。如果您不确定，请联系您的服务提供商。

数据库名	wordpress	希望将WordPress安装到的数据库名称。
用户名	wordpress	您的数据库用户名。
密码	wordpress	您的数据库密码。
数据库主机	mysql-0.mysql	如果localhost不能用，您通常可以从网站服务提供商处得到正确的信息。
表前缀	wp_	如果您希望在同一个数据库安装多个WordPress，请修改前缀。

提交

请在下方填写您的数据库连接信息。如果您不确定，请联系您的服务提供商。

数据库名	wordpress	希望将WordPress安装到的数据库名称。
用户名	wordpress	您的数据库用户名。
密码	wordpress	您的数据库密码。
数据库主机	mysql-0.mysql.default.svc.mage	如果localhost不能用，您通常可以从网站服务提供商处得到正确的信息。
表前缀	wp_	如果您希望在同一个数据库安装多个WordPress，请修改前缀。

提交

如果个mysql pod不在同一个namespace，那么就写对方service全称，以实现夸namespace解析和访问。

192.168.7.110:30031/wp-admin/setup-config.php?step=2

不错。您完成了安装过程中重要的一步，WordPress现在已经可以连接数据库了。如果您准备好了的话，现在就...

现在安装

192.168.7.110:30031/wp-admin/install.php?language=zh_CN

欢迎

欢迎使用著名的WordPress五分钟安装程序！请简单地填写下面的表格，来开始使用这个世界上最具扩展性、最强大的个人信息发布平台。

需要信息

您需要填写一些基本信息。无需担心填错，这些信息以后可以再次修改。

站点标题 马哥教育Linux架构师

用户名 admin

用户名只能含有字母、数字、空格、下划线、连字符、句号和"@"符号。

密码 y7$daQ#FimYShhxUz7 隐藏

强

重要： 您将需要此密码来登录，请将其保存在安全的位置。

您的电子邮件 2973707860@qq.com

请仔细检查电子邮件地址后再继续。

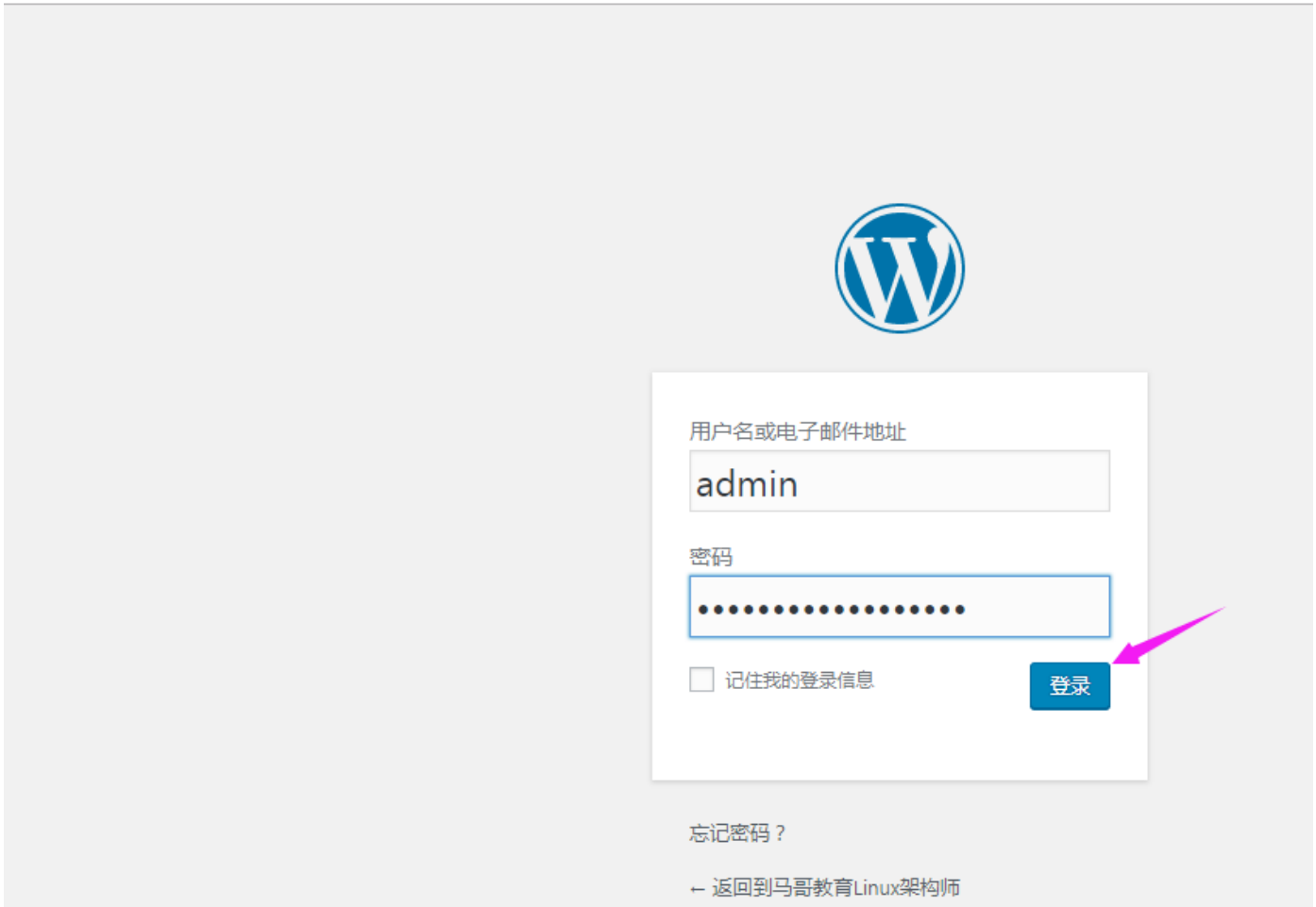
对搜索引擎的可见性 建议搜索引擎不索引本站点

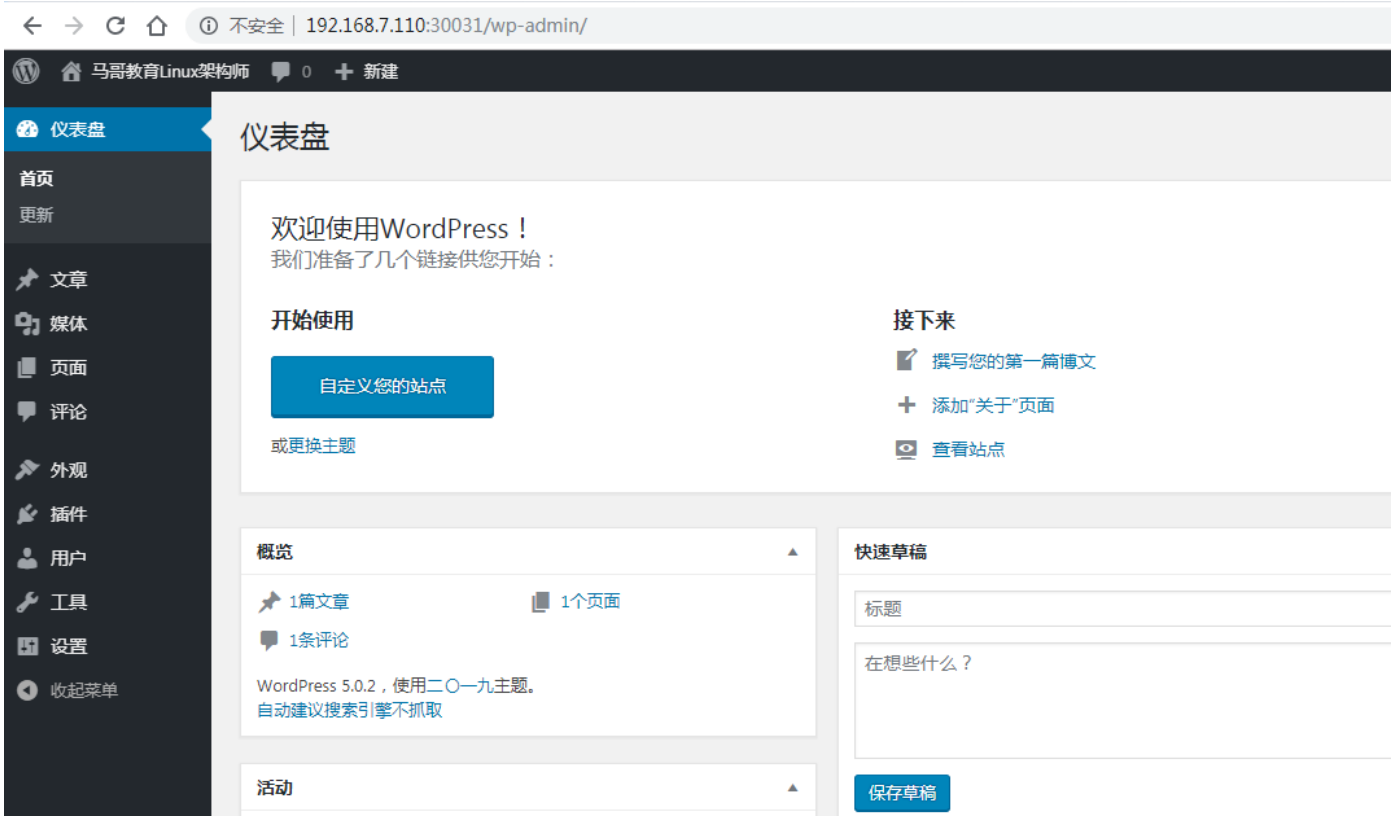
搜索引擎将本着自觉自愿的原则对待WordPress提出的请求。并不是所有搜索引擎都会遵守这类请求。

安装WordPress

192.168.7.110:30031/wp-admin/install.php?step=2

192.168.7.110:30031/wp-login.php





7.3.5.4.4：WordPress数据库连接配置：

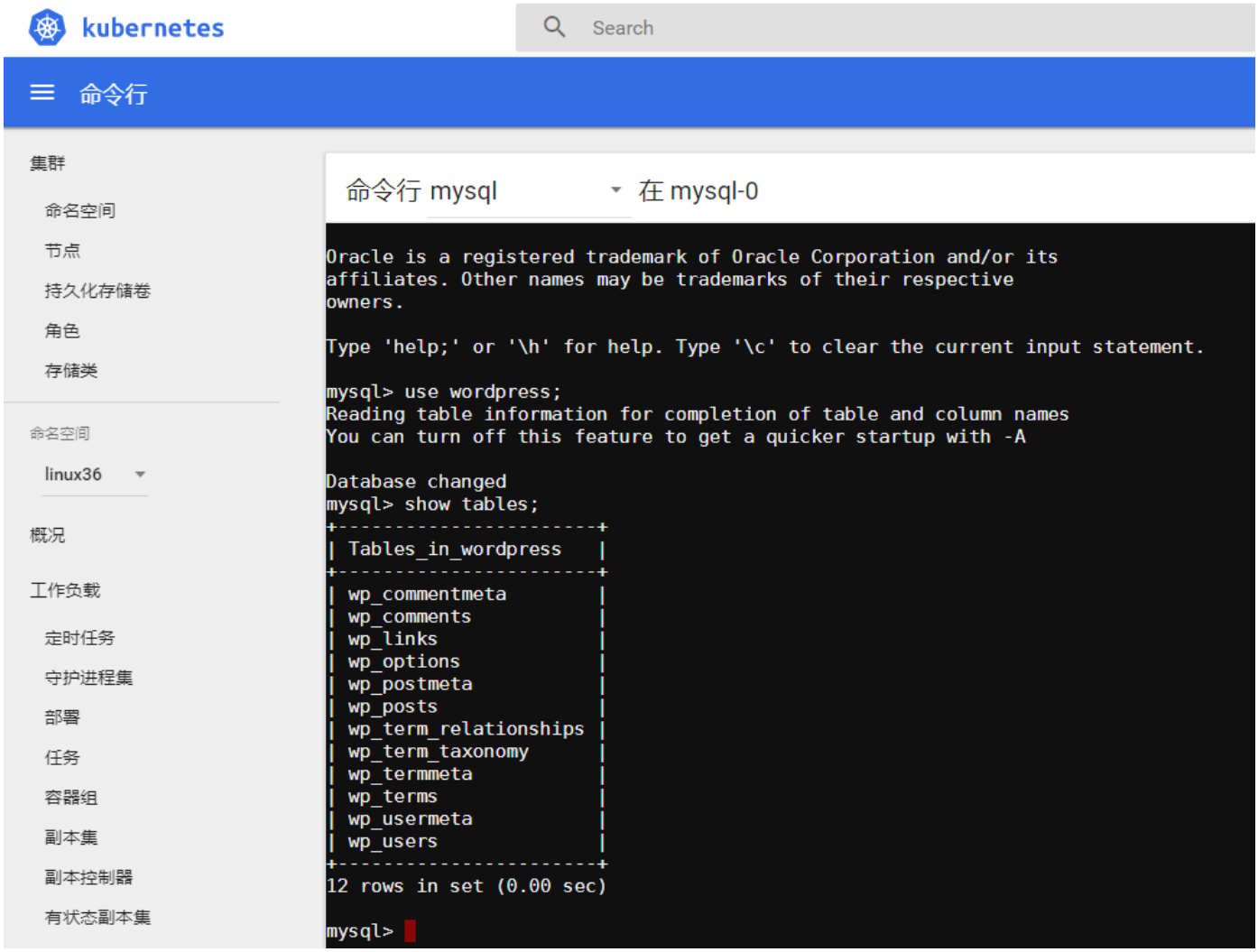
```
~ wordpress# cp wp-config-sample.php  wp-config.php

# cat  wp-config.php
<?php
/**
 * WordPress基础配置文件。
 *
 * 这个文件被安装程序用于自动生成wp-config.php配置文件，
 * 您可以不使用网站，您需要手动复制这个文件，
 * 并重命名为“wp-config.php”，然后填入相关信息。
 *
 * 本文件包含以下配置选项：
 *
 * * MySQL设置
 * * 密钥
 * * 数据库表名前缀
 * * ABSPATH
 *
 * @link https://codex.wordpress.org/zh-cn:%E7%BC%96%E8%BE%91_wp-config.php
 *
 * @package WordPress
 */

// ** MySQL 设置 - 具体信息来自您正在使用的主机 ** //
/** WordPress数据库的名称 */
define('DB_NAME', 'wordpress');
```

```
/** MySQL数据库用户名 */
define('DB_USER', 'wordpress');

/** MySQL数据库密码 */
define('DB_PASSWORD', 'wordpress');

/** MySQL主机 */
define('DB_HOST', 'mysql-0.mysql');

/** 创建数据表时默认的文字编码 */
define('DB_CHARSET', 'utf8');

/** 数据库整理类型。如不确定请勿更改 */
define('DB_COLLATE', '');

/**#@+
 * 身份认证密钥与盐。
..............................
```

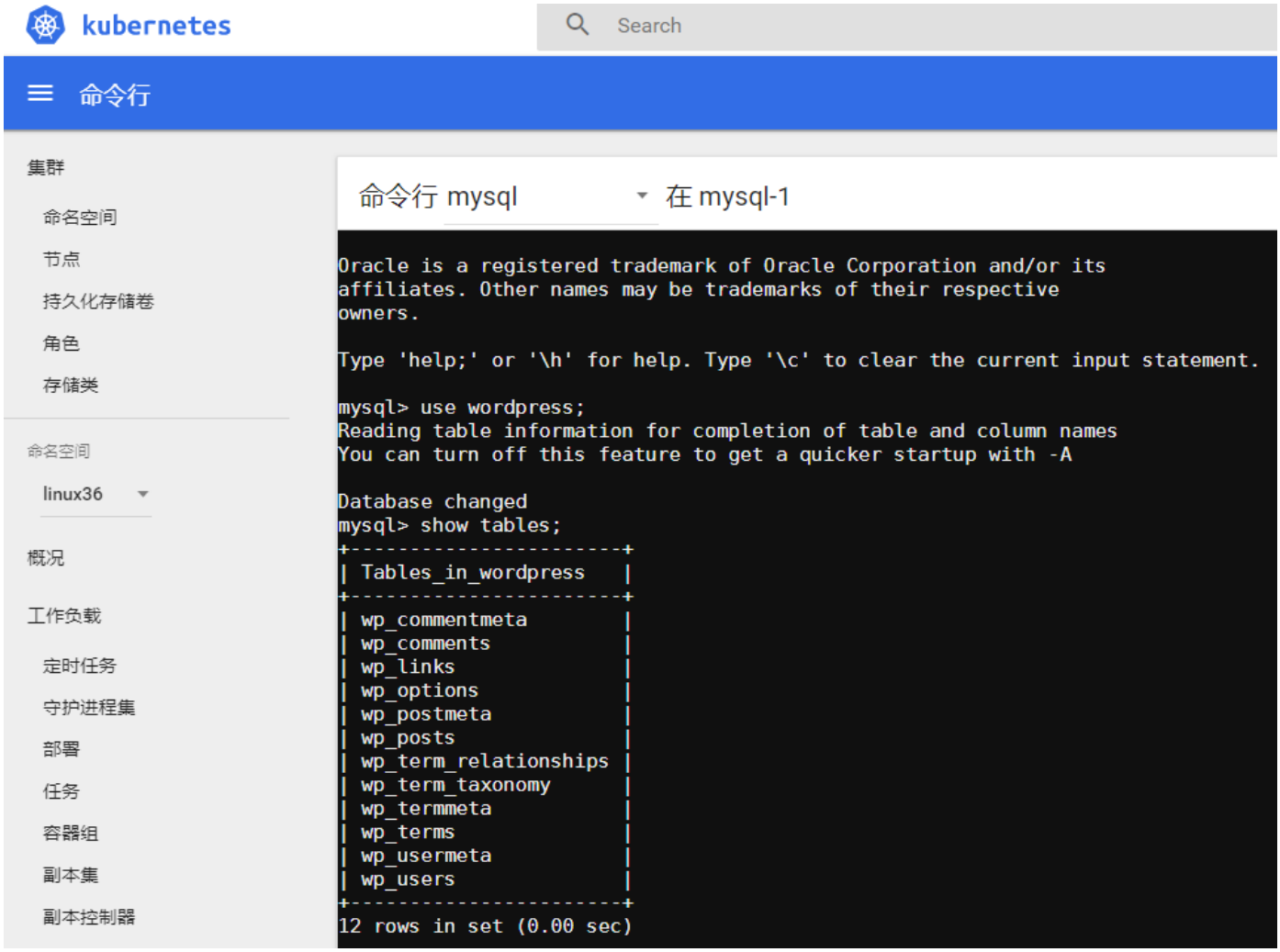
7.3.5.5：验证k8s中mysql 数据：

分别验证k8s中MySQL主库和从库是否数据

7.3.5.5.1：master数据：



7.3.5.5.2：slave数据库：



7.3.6：运行dubbo服务：

运行dubbo生成者与消费者示例。

dubbo 源码编译：

官方网站：http://dubbo.apache.org/zh-cn/

```
# mvn clean package -Dmaven.skip.test=true
```

7.3.6.1：运行provider：

7.3.6.1.1：准备镜像：

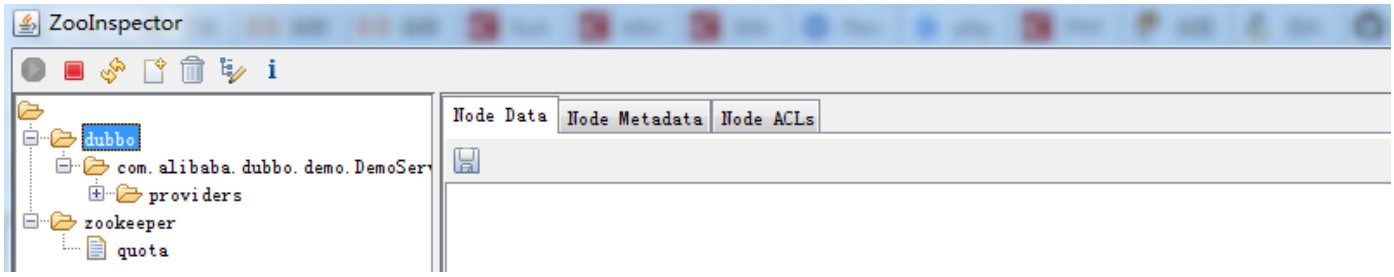
```
# pwd
/opt/k8s-data/dockerfile/linux36/dubbo/provider

# ll
total 10068
drwxr-xr-x 3 root root     4096 Aug  4 22:48 ./
drwxr-xr-x 5 root root     4096 Aug  4 22:47 ../
-rw-r--r-- 1 root root      145 Aug  4 22:47 build-command.sh
```

```
-rw-r--r-- 1 root root      349 Aug  4 22:47 Dockerfile
drwxr-xr-x 5 root root     4096 Aug  4 22:47 dubbo-demo-provider-2.1.5/
-rw-r--r-- 1 root root 10281793 Aug  4 22:47 dubbo-demo-provider-2.1.5-assembly.tar.gz
-rw-r--r-- 1 root root      313 Aug  4 22:47 run_java.sh

# chmod   a+x dubbo-demo-provider-2.1.5/bin/*.sh
# chmod   ./*.sh

# vim dubbo-demo-provider-2.1.5/conf/dubbo.properties #修改zookeeper地址
# bash build-command.sh #构建provider镜像
```

7.3.1.6.2：运行provider服务：

```
# pwd
/opt/k8s-data/yaml/linux36/dubbo/provider

# kubectl  apply -f provider.yaml
deployment.extensions/linux36-provider-deployment created
service/linux36-provider-spec created
```

7.3.1.6.3：zookeeper验证provider注册：



7.3.6.2：运行consumer：

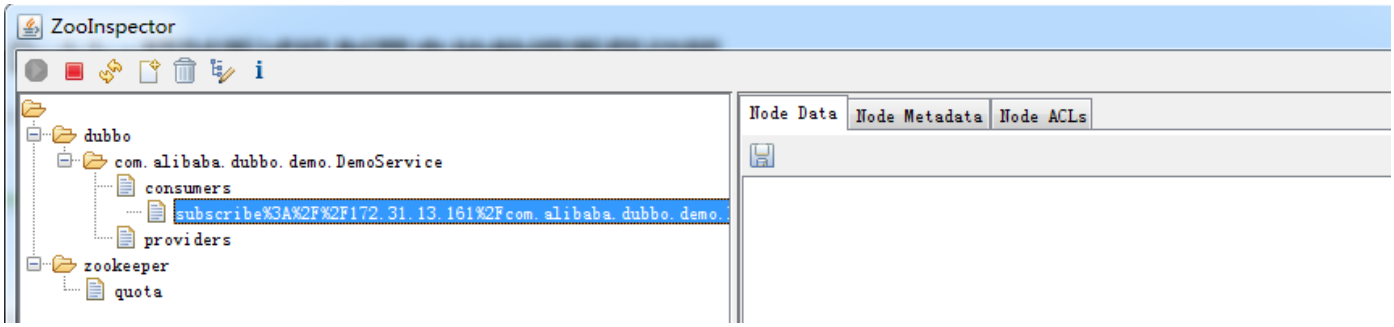
7.3.6.2.1：准备consumer镜像：

```
# pwd
/opt/k8s-data/dockerfile/linux36/dubbo/consumer

# chmod  a+x ./*.sh
# chmod  a+x dubbo-demo-consumer-2.1.5/bin/*.sh

# bash build-command.sh
```

7.3.6.2.2：运行consumer服务：

```
 pwd
/opt/k8s-data/yaml/linux36/dubbo/consumer

# kubectl  apply  -f .
deployment.extensions/linux36-consumer-deployment created
service/linux36-consumer-spec created
```

7.3.6.2.3：zookeeper验证消费者：

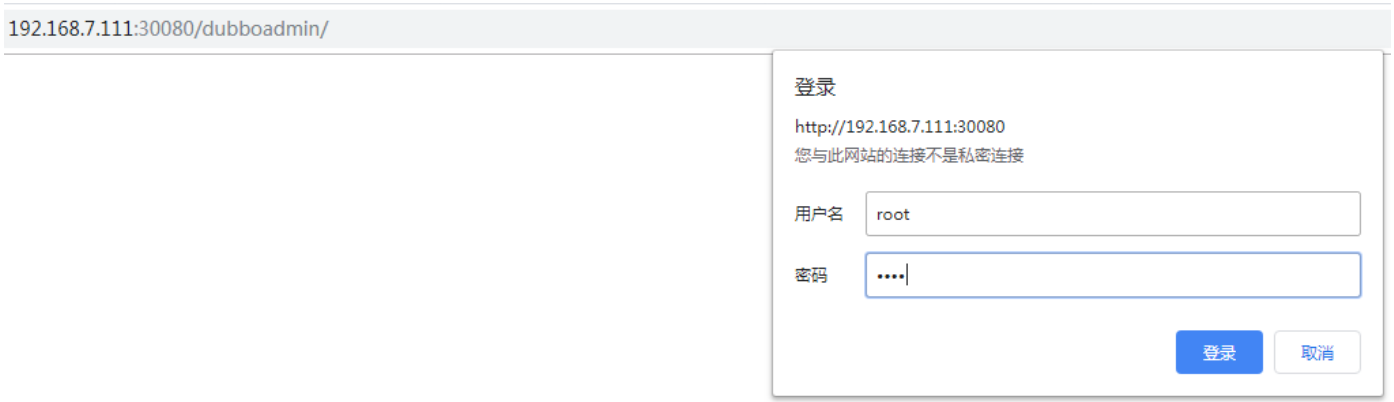


7.3.6.3：运行dubbo admin：

7.3.6.3.1：镜像准备：

```
# pwd
/opt/k8s-data/dockerfile/linux36/dubbo/dubboadmin

# vim dubboadmin/WEB-INF/dubbo.properties
# zip -r  dubboadmin.war dubboadmin/*
# bash build-command.sh
```

7.3.6.3.3：运行服务：



验证生产者与消费者：

当前pod运行状态：

```
root@k8s-master1:/opt/k8s-data/yaml/linux36/dubbo/dubboadmin# kubectl  get pods -n linux36
NAME                                          READY   STATUS    RESTARTS   AGE
linux36-consumer-deployment-c578cc67c-gvgj7   1/1     Running   0          20m
linux36-dubboadmin-deployment-5dfb77f566-4wqtx   1/1     Running   0          3m58s
linux36-jenkins-deployment-74c4b55576-ssxdv   1/1     Running   1          5h18m
linux36-provider-deployment-647fc9cdb8-kxw8n   1/1     Running   0          22m
mysql-0                                       2/2     Running   2          5h51m
mysql-1                                       2/2     Running   2          5h50m
mysql-2                                       2/2     Running   2          6h10m
wordpress-app-deployment-6f4d9bd6c-zfbdl      2/2     Running   2          3h36m
zookeeper1-5f458745bf-wtwph                   1/1     Running   0          23m
zookeeper2-5844958588-k6gm7                   1/1     Running   0          23m
zookeeper3-7b646949f-mjwvs                    1/1     Running   0          23m
root@k8s-master1:/opt/k8s-data/yaml/linux36/dubbo/dubboadmin#
```

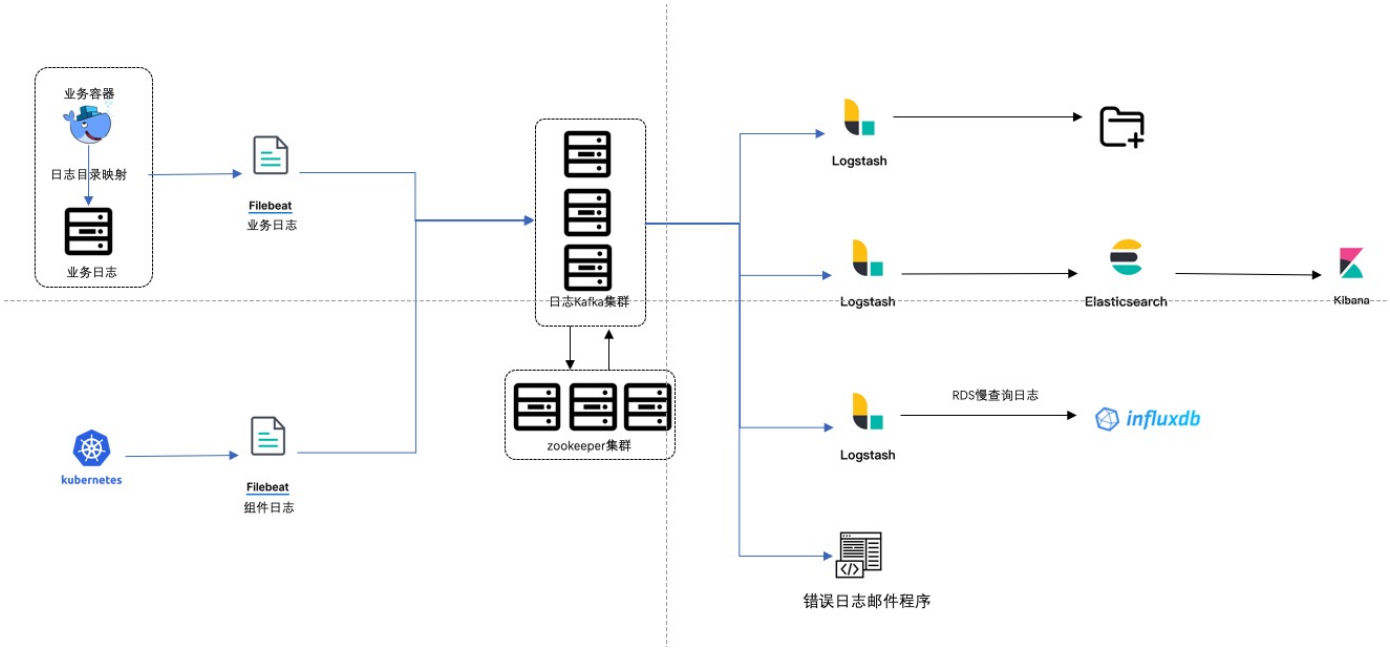
7.4：日志收集：

实现pod中日志收集之至ELK，自定义字段数据格式转换、排序、基于日志实现pod自愈、自动扩容等

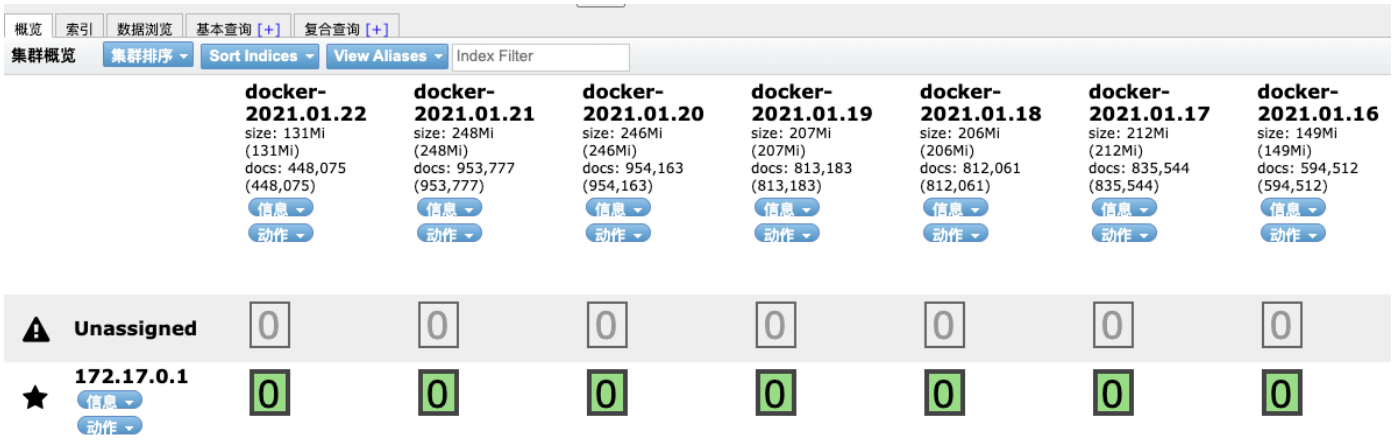
见ELK 日志收集。

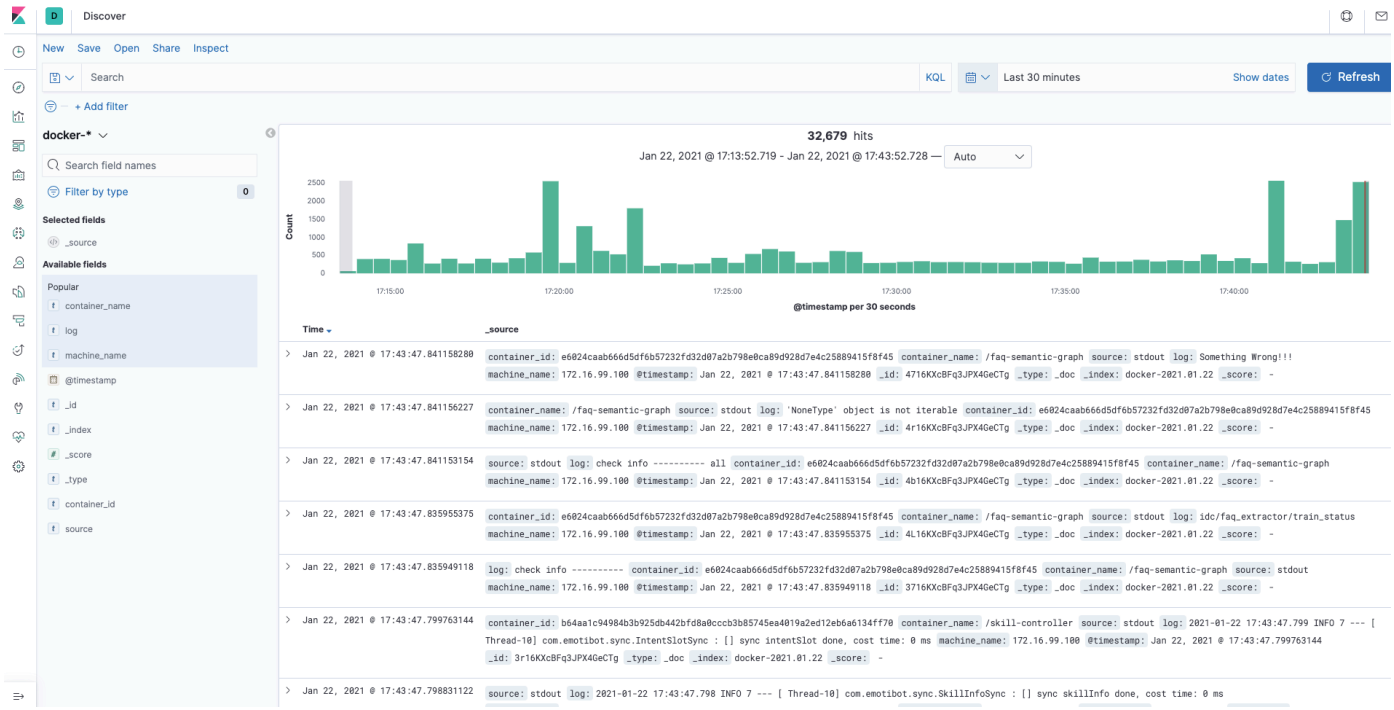
见ELK 日志收集。

https://kubernetes.io/zh/docs/concepts/cluster-administration/logging/




```
#cat build-command.sh

#!/bin/bash
#echo "nameserver 223.6.6.6" > /etc/resolv.conf
#echo "192.168.7.248 k8s-vip.example.com" >> /etc/hosts

/usr/share/filebeat/bin/filebeat -c /etc/filebeat/filebeat.yml -path.home
/usr/share/filebeat -path.config /etc/filebeat -path.data /var/lib/filebeat -path.logs
/var/log/filebeat  &
su - tomcat -c "/apps/tomcat/bin/catalina.sh start"
tail -f /etc/hosts

filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /apps/tomcat/logs/catalina.out
  fields:
    type: tomcat-catalina
filebeat.config.modules:
  path: ${path.config}/modules.d/*.yml
  reload.enabled: false
setup.template.settings:
  index.number_of_shards: 1
setup.kibana:
output.redis:
  hosts: ["172.31.2.105:6379"]
  key: "k8s-magedu-app1"
  db: 1
  timeout: 5
  password: "123456"
```


概览
索引
数据浏览
基本查询 [+]
复合查询 [+]
集群概览
集群排序
Sort Indices
View Aliases
Index Filter
docker-2021.01.22
size: 131Mi (131Mi)
docs: 448,075 (448,075)
docker-2021.01.21
size: 248Mi (248Mi)
docs: 953,777 (953,777)
docker-2021.01.20
size: 246Mi (246Mi)
docs: 954,163 (954,163)
docker-2021.01.19
size: 207Mi (207Mi)
docs: 813,183 (813,183)
docker-2021.01.18
size: 206Mi (206Mi)
docs: 812,061 (812,061)
docker-2021.01.17
size: 212Mi (212Mi)
docs: 835,544 (835,544)
docker-2021.01.16
size: 149Mi (149Mi)
docs: 594,512 (594,512)
信息
动作
Unassigned
0
172.17.0.1


Discover
New
Save
Open
Share
Inspect
Search
KQL
Last 30 minutes
Show dates
Refresh
+ Add filter
docker-*
Search field names
Filter by type
Selected fields
_source
Available fields
Popular
container_name
log
machine_name
@timestamp
_id
_index
_score
_type
container_id
source
32,679 hits
Jan 22, 2021 @ 17:13:52.719 - Jan 22, 2021 @ 17:43:52.728 —
Auto
Count
@timestamp per 30 seconds
Time
_source

Elasticsearch http://172.16.99.100:9201/ 连接 **EFKcluster** **集群健康值: yellow (12 of 19)** 信息

概览 | 索引 | 数据浏览 | 基本查询 [+] | 复合查询 [+]

数据浏览 刷新

所有索引

索引

- .apm-agent-configuration
- .apm-custom-link
- .async-search
- .kibana_1
- .kibana_task_manager_1
- docker-2021.01.16
- docker-2021.01.17
- docker-2021.01.18
- docker-2021.01.19
- docker-2021.01.20
- docker-2021.01.21
- docker-2021.01.22

类型

- _doc

字段

- action.actionTypeId
- action.name
- action.secrets
- action_task_params.actionId
- action_task_params.apiKey
- agent_name
- alert.actions.actionRef
- alert.actions.actionTypeId
- alert.actions.group
- alert.alertTypeId
- alert.apiKey
- alert.apiKeyOwner
- alert.consumer
- alert.createdAt
- alert.createdBy
- alert.enabled
- alert.muteAll
- alert.mutedInstanceIds
- alert.name
- alert.schedule.interval
- alert.scheduledTaskId
- alert.tags
- alert.throttle
- alert.updatedBy

查询 1 个分片中用的 1 个. 10000 命中. 耗时 0.062 秒

_index	_type	_id	_score ▲	container_name	source	log
docker-2021.01.19	_doc	HSzwF3cBYmS_V2B093v6	1	/re-engine	stderr	2021-01-19 08:00:00.319 INFO MainProcess --- [MainThread] web.middleware : event: request_begin, uri: http://0.0.0.0:15006/health_c
docker-2021.01.19	_doc	HizwF3cBYmS_V2B093v6	1	/re-engine	stderr	2021-01-19 08:00:00.319 INFO MainProcess --- [MainThread] web.middleware : event: output: {"status": "OK"}
docker-2021.01.19	_doc	HyzwF3cBYmS_V2B093v6	1	/mood	stderr	2021-01-19 08:00:00.320 INFO MainProcess --- [MainThread] web.middleware : event: request_begin, GET http://0.0.0.0:21103/health_
docker-2021.01.19	_doc	ICzwF3cBYmS_V2B093v6	1	/mood	stderr	2021-01-19 08:00:00.320 INFO MainProcess --- [MainThread] web.middleware : event: request_end
docker-2021.01.19	_doc	ISzwF3cBYmS_V2B093v6	1	/mood	stderr	2021-01-19 08:00:00.321 INFO MainProcess --- [MainThread] web.middleware : event: request_span, diff: 0:00:00.000489
docker-2021.01.19	_doc	IizwF3cBYmS_V2B093v6	1	/re-engine	stderr	2021-01-19 08:00:00.319 INFO MainProcess --- [MainThread] web.middleware : event: request_span, diff: 0:00:00.000441
docker-2021.01.19	_doc	IyzwF3cBYmS_V2B093v6	1	/mood	stderr	2021-01-19 08:00:00.321 INFO MainProcess --- [MainThread] web.middleware : event: output: {"status": "ok"}
docker-2021.01.19	_doc	JCzwF3cBYmS_V2B093v6	1	/model-scheduler	stdout	2021-01-19 08:00:00.395 - [INFO] - [app.modelScheduler.logic.ModelScheduleUtil.onPredictStatsFlushTimer():150] 统计数据准点入库[20
docker-2021.01.19	_doc	JSzwF3cBYmS_V2B093v6	1	/model-scheduler	stdout	2021-01-19 08:00:00.395 - [INFO] - [app.modelScheduler.logic.ModelScheduleUtil.onPredictStatsFlushTimer():153] 统计数据准点入库[20
docker-2021.01.19	_doc	JizwF3cBYmS_V2B093v6	1	/model-scheduler	stdout	2021-01-19 08:00:00.458 - [INFO] - [app.modelScheduler.logic.ModelStatsUtil.save2db():251] 本次 更新 345 个模型的 预测请求统计数据
docker-2021.01.19	_doc	JyzwF3cBYmS_V2B093v6	1	/model-scheduler	stdout	2021-01-19 08:00:00.459 - [INFO] - [app.modelScheduler.logic.ModelScheduleUtil.onPredictStatsFlushTimer():157] 统计数据准点入库[20
docker-2021.01.19	_doc	KCzwF3cBYmS_V2B093v6	1	/model-scheduler	stdout	2021-01-19 08:00:00.513 - [INFO] - [app.modelScheduler.logic.ModelStatsUtil.save2db():251] 本次 更新 345 个模型的 预测请求统计数据
docker-2021.01.19	_doc	KSzwF3cBYmS_V2B093v6	1	/model-scheduler	stdout	2021-01-19 08:00:00.514 - [INFO] - [app.modelScheduler.logic.ModelScheduleUtil.onModelCntStatsFlushTimer():172] 统计数据准点入库[
docker-2021.01.19	_doc	KizwF3cBYmS_V2B093v6	1	/model-scheduler	stdout	2021-01-19 08:00:00.516 - [INFO] - [app.modelScheduler.logic.ModelCountingUtil.save2db():154] 2021-01-19 07:59:00.0 ~ 2021-01-
docker-2021.01.19	_doc	KyzwF3cBYmS_V2B093v6	1	/model-scheduler	stdout	2021-01-19 08:00:00.516 - [INFO] - [app.modelScheduler.logic.ModelScheduleUtil.onModelCntStatsFlushTimer():175] 统计数据准点入库[
docker-2021.01.19	_doc	LCzwF3cBYmS_V2B093v6	1	/model-scheduler	stdout	2021-01-19 08:00:00.518 - [INFO] - [app.modelScheduler.logic.ModelCountingUtil.save2db():154] 2021-01-19 07:00:00.0 ~ 2021-01-
docker-2021.01.19	_doc	LSzwF3cBYmS_V2B093v6	1	/model-scheduler	stdout	2021-01-19 08:00:00.518 - [INFO] - [app.modelScheduler.logic.ModelScheduleUtil.onModelCntStatsFlushTimer():179] 统计数据准点入库[
docker-2021.01.19	_doc	LizwF3cBYmS_V2B093v6	1	/model-scheduler	stdout	2021-01-19 08:00:00.520 - [INFO] - [app.modelScheduler.logic.ModelCountingUtil.save2db():154] 2021-01-19 00:00:00.0 ~ 2021-01-
docker-2021.01.19	_doc	LyzwF3cBYmS_V2B093v6	1	/model-scheduler	stdout	2021-01-19 08:00:00.521 - [INFO] - [app.modelScheduler.logic.ModelScheduleUtil.delExpiredStatsFromDb():289] 循环删除过期统计数据[
docker-2021.01.19	_doc	MCzwF3cBYmS_V2B093v6	1	/model-scheduler	stdout	2021-01-19 08:00:00.545 - [INFO] - [app.modelScheduler.logic.ModelScheduleUtil.delExpiredStatsFromDb():294] 循环删除过期统计数据[
docker-2021.01.19	_doc	MSzwF3cBYmS_V2B093v6	1	/model-scheduler	stdout	2021-01-19 08:00:00.683 - [INFO] - [app.modelScheduler.logic.ModelScheduleUtil.delExpiredStatsFromDb():299] 循环删除过期统计数据[
docker-2021.01.19	_doc	MizwF3cBYmS_V2B093v6	1	/intent-engine	stderr	2021-01-19 08:00:01.006 INFO MainProcess --- [MainThread] web.middleware : event: request_begin, uri: http://0.0.0.0:15001/health_c
docker-2021.01.19	_doc	MyzwF3cBYmS_V2B093v6	1	/intent-engine	stderr	2021-01-19 08:00:01.007 INFO MainProcess --- [MainThread] web.middleware : event: output: {"status": "OK"}
docker-2021.01.19	_doc	NCzwF3cBYmS_V2B093v6	1	/intent-engine	stderr	2021-01-19 08:00:01.007 INFO MainProcess --- [MainThread] web.middleware : event: request_span, diff: 0:00:00.000484
docker-2021.01.19	_doc	NSzwF3cBYmS_V2B093v6	1	/emotion-sentence-embedding	stderr	2021-01-19 08:00:01.011 INFO MainProcess --22- [MainThread] web.middleware : event: request_begin, GET http://localhost/health_che
docker-2021.01.19	_doc	NizwF3cBYmS_V2B093v6	1	/emotion-sentence-embedding	stderr	2021-01-19 08:00:01.012 INFO MainProcess --22- [MainThread] web.middleware : event: request_end
docker-2021.01.19	_doc	NyzwF3cBYmS_V2B093v6	1	/emotion-sentence-embedding	stderr	2021-01-19 08:00:01.012 INFO MainProcess --22- [MainThread] web.middleware : event: request_span, diff: 0:00:00.000421
docker-2021.01.19	_doc	OCzwF3cBYmS_V2B093v6	1	/keyword-engine	stderr	2021-01-19 08:00:01.015 INFO MainProcess --- [MainThread] web.middleware : event: request_begin, uri: http://0.0.0.0:15005/health_c
docker-2021.01.19	_doc	OSzwF3cBYmS_V2B093v6	1	/keyword-engine	stderr	2021-01-19 08:00:01.016 INFO MainProcess --- [MainThread] web.middleware : event: output: {"status": "OK"}
docker-2021.01.19	_doc	OizwF3cBYmS_V2B093v6	1	/keyword-engine	stderr	2021-01-19 08:00:01.016 INFO MainProcess --- [MainThread] web.middleware : event: request_span, diff: 0:00:00.000604
docker-2021.01.19	_doc	OyzwF3cBYmS_V2B093v6	1	/keyword-engine	stderr	2021-01-19 08:00:01.471 INFO MainProcess --- [Thread-1] web.handlers.tracker : [UPDATER] get new load_map.
docker-2021.01.19	_doc	PCzwF3cBYmS_V2B093v6	1	/intent-engine	stderr	2021-01-19 08:00:01.472 INFO MainProcess --- [Thread-1] web.handlers.tracker : [UPDATER] get new load_map.
docker-2021.01.19	_doc	PSzwF3cBYmS_V2B093v6	1	/re-engine	stderr	2021-01-19 08:00:01.621 INFO MainProcess --- [Thread-1] web.handlers.tracker : [UPDATER] get new load_map.
docker-2021.01.19	_doc	PizwF3cBYmS_V2B093v6	1	/emotion-engine	stderr	2021-01-19 08:00:01.866 INFO MainProcess --- [Thread-1] web.handlers.tracker : [UPDATER] get new load_map.
docker-2021.01.19	_doc	PyzwF3cBYmS_V2B093v6	1	/skill-controller	stdout	2021-01-19 08:00:02.001 INFO 7 --- [Thread-10] com.emotibot.sync.IntentInfoSync : [] sync intent done, cost time: 1 ms
docker-2021.01.19	_doc	QCzwF3cBYmS_V2B093v6	1	/skill-controller	stdout	2021-01-19 08:00:02.003 INFO 7 --- [Thread-10] com.emotibot.sync.MaterialSync : [] sync material done, cost time: 1 ms
docker-2021.01.19	_doc	QSzwF3cBYmS_V2B093v6	1	/skill-controller	stdout	2021-01-19 08:00:02.004 INFO 7 --- [Thread-10] com.emotibot.sync.SkillInfoSync : [] sync skillInfo done, cost time: 0 ms
docker-2021.01.19	_doc	QizwF3cBYmS_V2B093v6	1	/skill-controller	stdout	2021-01-19 08:00:02.004 INFO 7 --- [Thread-10] com.emotibot.sync.IntentSlotSync : [] sync intentSlot done, cost time: 0 ms
docker-2021.01.19	_doc	nyzxF3cBYmS_V2B0KHtO	1	/fc	stdout	update data time from 2021-01-12 to 2021-01-26
docker-2021.01.19	_doc	oCzxF3cBYmS_V2B0KHtO	1	/skill-controller	stdout	2021-01-19 08:00:13.671 INFO 7 --- [Timer-4] com.emotibot.sync.IntentSlotSync : [] sync intentSlot done, cost time: 0 ms