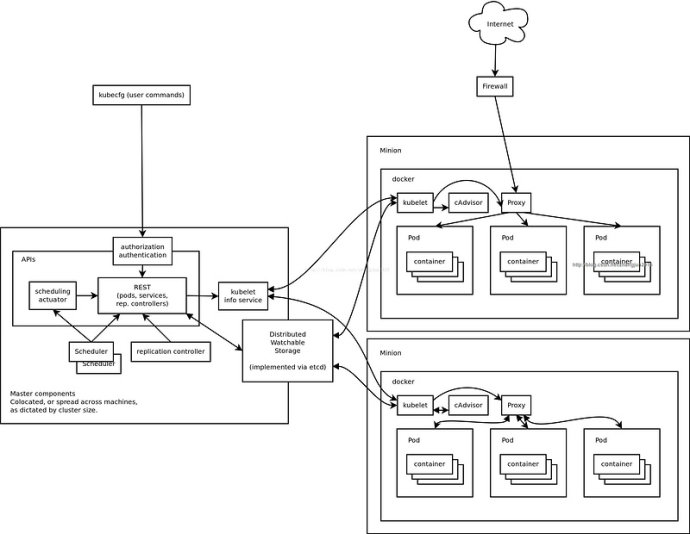
**Kubernetes-1.4.x集群**

**官方网站:**

<http://kubernetes.io/>

<https://github.com/kubernetes/kubernetes>

<https://access.redhat.com/documentation/en/red-hat-enterprise-linux-atomic-host/7/single/getting-started-with-containers>

[](http://photo.blog.sina.com.cn/showpic.html#blogid=8ea8e9d50102ww8m&url=http://album.sina.com.cn/pic/002BYBFzzy74suIeUFb69)

参考文档:

<https://segmentfault.com/a/1190000002886795>

**环境:**

**CentOS7.3**

**etcd-3.0.4**

**flannel-0.6.1**

**docker-1.12.1**

**kubernetes-1.4.1**

**kubernetes-dashboard-1.4.0**

Minion1+Etcd1:192.168.8.101

Minion2+Etcd2:192.168.8.102

Minion3+Etcd3:192.168.8.103

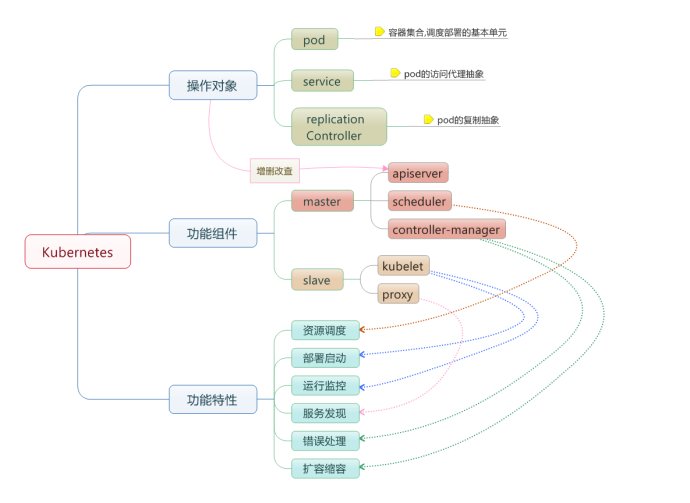
Master1:192.168.8.201

**一.组件安装**

<http://kubernetes.io/docs/getting-started-guides/scratch/>

<http://kubernetes.io/docs/admin/cluster-components/>

* etcd
* A container runner, one of:
  + docker
  + rkt
* Kubernetes
  + kubelet
  + kube-proxy
  + kube-apiserver
  + kube-controller-manager
  + kube-scheduler

You will run docker, kubelet, and kube-proxy outside of a container, the same way you would run any system daemon, so you just need the bare binaries. For etcd, kube-apiserver, kube-controller-manager, and kube-scheduler, we recommend that you run these as containers, so you need an image to be built.[](http://photo.blog.sina.com.cn/showpic.html#blogid=8ea8e9d50102ww8m&url=http://album.sina.com.cn/pic/002BYBFzzy74srdY6vF53)

**1.etcd**

**a.容器外运行**

请参看[Etcd集群](http://blog.sina.com.cn/s/blog_8ea8e9d50102x6im.html)

http://192.168.8.101:2379,http://192.168.8.102:2379,http://192.168.8.103:2379

或

**b.容器内运行**

<https://quay.io/repository/coreos/etcd?tag=latest>

kubernets官方建议使用kubernets二进制包里经过充分测试的etcd版本

docker pull quay.io/coreos/etcd

docker tag quay.io/coreos/etcd etcd

docker rmi -f quay.io/coreos/etcd

docker run -tid --restart=always \

    -p 2379:2379 \

    -p 2380:2380 \

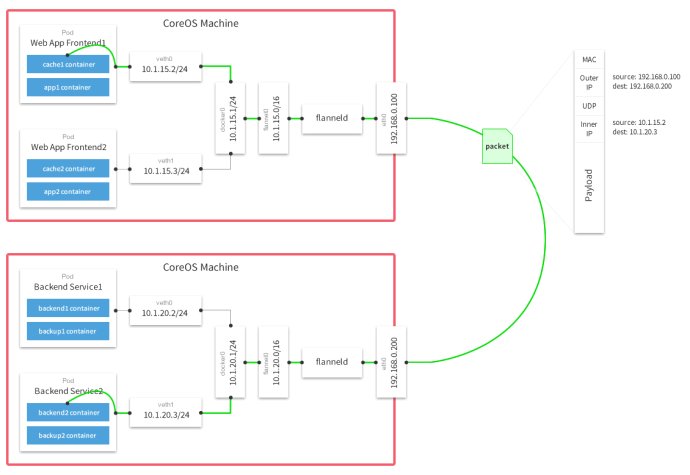
    --net=host \

    --name etcd \

    etcd /usr/local/bin/etcd --name etcd --data-dir /opt/etcd

**提示:**本实验采用容器外Etcd集群

**2.flannel(所有Minion节点)**

[](http://photo.blog.sina.com.cn/showpic.html#blogid=8ea8e9d50102ww8m&url=http://album.sina.com.cn/pic/002BYBFzzy74tRzLzKR3d)

<https://github.com/coreos/flannel/releases/>

flannel是coreos在docker跨主机组网上的一个解决方案

**i.安装flannel**

curl -sL <https://github.com/coreos/flannel/releases/download/v0.6.1/flannel-v0.6.1-linux-amd64.tar.gz>|tar -xvf - --gzip

mv flanneld mk-docker-opts.sh /usr/local/bin/

**ii.配置桥接网段**

etcdctl set /coreos.com/network/config  '{ "Network": "**10.1.0.0/16**", "Backend": { "Type": "vxlan", "VNI": 1 } }'

**iii.启动flanneld(所有Minion节点)**

flanneld -iface=eth0 -subnet-file=/etc/profile.d/flanneld.env -etcd-endpoints=http://192.168.8.101:2379,http://192.168.8.102:2379,http://192.168.8.103:2379

[root@node1 ~]# ifconfig

eth0: flags=4163  mtu 1500

        inet 192.168.8.101  netmask 255.255.255.0  broadcast 192.168.8.255

        ether 52:54:00:07:00:01  txqueuelen 1000  (Ethernet)

        RX packets 34410  bytes 3536207 (3.3 MiB)

        RX errors 0  dropped 0  overruns 0  frame 0

        TX packets 32932  bytes 5034105 (4.8 MiB)

        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

**flannel.1**: flags=4163  mtu 1450

        inet **10.1.71.0**  netmask 255.255.0.0  broadcast 0.0.0.0

        ether b6:9b:94:d7:fc:09  txqueuelen 0  (Ethernet)

        RX packets 0  bytes 0 (0.0 B)

        RX errors 0  dropped 0  overruns 0  frame 0

        TX packets 0  bytes 0 (0.0 B)

        TX errors 0  dropped 2 overruns 0  carrier 0  collisions 0

**iv.systemd管控**

**1.创建flanneld.service**

cat >/lib/systemd/system/flanneld.service <<'HERE'

[Unit]

Description=Flannel Server

After=network.target

After=network-online.target

Wants=network-online.target

[Service]

Type=notify

EnvironmentFile=/etc/flanneld.conf

ExecStart=/usr/local/bin/flanneld -subnet-file=/etc/profile.d/flanneld.env -etcd-endpoints=${FLANNELD\_ETCD\_ENDPOINTS}

Restart=on-failure

LimitNOFILE=1000000

[Install]

WantedBy=multi-user.target

HERE

**2.创建主配置文件flanneld.conf**

cat >/etc/flanneld.conf <<HERE

FLANNELD\_ETCD\_ENDPOINTS=http://192.168.8.101:2379,http://192.168.8.102:2379,http://192.168.8.103:2379

HERE

**3.测试systemd启动flanneld**

[root@node4 ~]# systemctl enable flanneld

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

[root@node4 ~]# systemctl restart flanneld

[root@node4 ~]# systemctl status flanneld

**●** flanneld.service - Flannel Server

   Loaded: loaded (/usr/lib/systemd/system/flanneld.service; enabled; vendor preset: disabled)

   Active: **active (running)** since 三 2016-08-31 14:12:13 CST; 12s ago

 Main PID: 2449 (flanneld)

   CGroup: /system.slice/flanneld.service

           └─2449 /usr/local/bin/flanneld -subnet-file=/etc/profile.d/flanneld.env -etcd-endpoint...

8月 31 14:12:13 node4.example.com flanneld[2449]: I0831 14:12:13.759557 02449 local\_manager.go...ng

8月 31 14:12:13 node4.example.com flanneld[2449]: I0831 14:12:13.771634 02449 manager.go:246] ...24

8月 31 14:12:13 node4.example.com systemd[1]: Started Flannel Server.

8月 31 14:12:13 node4.example.com flanneld[2449]: I0831 14:12:13.772516 02449 network.go:58] W...es

8月 31 14:12:13 node4.example.com flanneld[2449]: I0831 14:12:13.772545 02449 network.go:66] W...es

8月 31 14:12:13 node4.example.com flanneld[2449]: I0831 14:12:13.789447 02449 network.go:153] ...ts

8月 31 14:12:13 node4.example.com flanneld[2449]: I0831 14:12:13.789467 02449 device.go:163] c... 3

8月 31 14:12:13 node4.example.com flanneld[2449]: I0831 14:12:13.789578 02449 network.go:160] ...4b

8月 31 14:12:13 node4.example.com flanneld[2449]: I0831 14:12:13.789615 02449 network.go:160] ...01

8月 31 14:12:13 node4.example.com flanneld[2449]: I0831 14:12:13.789620 02449 network.go:160] ...0c

Hint: Some lines were ellipsized, use -l to show in full.

**3.docker(所有Minion节点)**

请参看[CentOS6/7 docker安装](http://blog.sina.com.cn/s/blog_8ea8e9d50102ww8w.html)

**注意:**

1.集群服务中的docker要监听在tcp端口，**-H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock**

2.需要指定桥接网络

修改docker.service启动参数

**EnvironmentFile=/etc/profile.d/flanneld.env**

ExecStart=/usr/bin/dockerd --registry-mirror http://192.168.8.254:5000 --insecure-registry 192.168.8.254:5000 -H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock **--bip=${FLANNEL\_SUBNET} --mtu=${FLANNEL\_MTU}**

ip link set dev docker0 down

brctl delbr docker0 #yum -y install bridge-utils

systemctl daemon-reload

systemctl restart docker

systemctl enable docker

**提示:** docker指定flannel桥接网络后，docker宿主机上的容器就可以实现跨主机通信，实际上是修改了docker原生docker0的地址

[root@node1 ~]# ifconfig docker0

docker0: flags=4099  mtu 1500

        inet **10.1.71.1**  netmask 255.255.255.0  broadcast 0.0.0.0

        ether 02:42:fb:d7:89:1e  txqueuelen 0  (Ethernet)

        RX packets 0  bytes 0 (0.0 B)

        RX errors 0  dropped 0  overruns 0  frame 0

        TX packets 0  bytes 0 (0.0 B)

        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@node2 ~]# ifconfig docker0

docker0: flags=4099  mtu 1500

        inet **10.1.68.1**  netmask 255.255.255.0  broadcast 0.0.0.0

        ether 02:42:ae:25:8e:7d  txqueuelen 0  (Ethernet)

        RX packets 0  bytes 0 (0.0 B)

        RX errors 0  dropped 0  overruns 0  frame 0

        TX packets 0  bytes 0 (0.0 B)

        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@node3 ~]# ifconfig docker0

docker0: flags=4099  mtu 1500

        inet **10.1.44.1**  netmask 255.255.255.0  broadcast 0.0.0.0

        ether 02:42:a1:53:39:7c  txqueuelen 0  (Ethernet)

        RX packets 0  bytes 0 (0.0 B)

        RX errors 0  dropped 0  overruns 0  frame 0

        TX packets 0  bytes 0 (0.0 B)

        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@node3 ~]# ping 10.1.68.1

PING 10.1.68.1 (10.1.68.1) 56(84) bytes of data.

64 bytes from 10.1.68.1: icmp\_seq=1 ttl=64 time=1.82 ms

64 bytes from 10.1.68.1: icmp\_seq=2 ttl=64 time=0.733 ms

**4.kuberetes**

**二进制包安装(约1G)**

curl -sSL <https://github.com/kubernetes/kubernetes/releases/download/v1.4.1/kubernetes.tar.gz> -o kubernetes.tar.gz

tar xvf kubernetes.tar.gz

cp kubernetes/platforms/linux/amd64/kubectl /usr/bin

chmod +x /usr/bin/kubectl

tar xvf kubernetes/server/kubernetes-server-linux-amd64.tar.gz -C /opt

[root@node1 ~]# cd /opt/kubernetes/server/bin/

[root@node1 bin]# ls

federation-apiserver                      kube-controller-manager.tar

federation-apiserver.docker\_tag           kubectl

federation-apiserver.tar                  kube-dns

federation-controller-manager             kubelet

federation-controller-manager.docker\_tag  kubemark

federation-controller-manager.tar         kube-proxy

hyperkube                                 kube-proxy.docker\_tag

kube-apiserver                            kube-proxy.tar

kube-apiserver.docker\_tag                 kube-scheduler

kube-apiserver.tar                        kube-scheduler.docker\_tag

kube-controller-manager                   kube-scheduler.tar

kube-controller-manager.docker\_tag

**提示:**kubernetes二进制包直接提供了kube-apiserver, kube-controller-manager, and kube-scheduler等docker image,导入后即可使用

docker load -i kube-apiserver.tar

docker load -i kube-controller-manager.tar

docker load -i kube-scheduler.tar

[root@node1 bin]# docker images

REPOSITORY                                         TAG                                IMAGE ID            CREATED             SIZE

etcd                                               latest                             dce3ed2412be        2 weeks ago         43.3 MB

gcr.io/google\_containers/kube-apiserver            d0247a2195e3bf420e64d887acec323c   d0ed2fe3ef25        3 weeks ago         110.8 MB

gcr.io/google\_containers/kube-controller-manager   e40cae03d66549ce387642a893c76bbf   3a99d6aaabad        3 weeks ago         100.6 MB

gcr.io/google\_containers/kube-scheduler            1d39a72473ede8ceda23bfa4aca8bd33   468f18f0d101        3 weeks ago         60.08 MB

或

**源码安装**

**1.安装或升级go**

curl -sSL <http://www.golangtc.com/static/go/1.6.2/go1.6.2.linux-amd64.tar.gz> -o o1.6.2.linux-amd64.tar.gz

tar -xvf go1.6.2.linux-amd64.tar.gz -C /opt

sudo cat >>/etc/profile <<'HERE'

export GOROOT=/opt/go

export GOPATH=/var/tmp/go

export PATH=$GOROOT/bin:$PATH

HERE

source /etc/profile

提示:主要设置GOROOT(安装路径),GOPATH(go项目的存放位置，自定义)

**root@router:~#**go version

go version go1.6.2 linux/amd64

**2.源码安装**

git clone <https://github.com/kubernetes/kubernetes.git>

cd kubernetes

make release

**提示:**编译后的打包文件放在kubernetes/\_output/release-tars,但因国内网络问题,正常情况下无法编译通过

Step 1 : FROM gcr.io/google\_containers/kube-cross:v1.6.2-2

Get https://gcr.io/v1/\_ping: dial tcp 173.194.72.82:443: getsockopt: **connection refused**

**二.配置kubernetes**

**禁用firewalld,启用iptables(所有节点)**

systemctl disable firewalld

systemctl stop firewalld

yum -y install iptables-services

systemctl enable iptables

systemctl start iptables

iptables-save >/etc/sysconfig/iptables

**A.Master组件(kube-apiserver,kube-scheduler,kube-controller-manager)**

**Master1:192.168.8.201**

**1.kube-apiserver**

<http://kubernetes.io/docs/admin/high-availability/kube-apiserver.yaml>

<http://kubernetes.io/docs/user-guide/kubectl-overview/>

<http://kubernetes.io/docs/admin/high-availability/>

/opt/kubernetes/server/bin/kube-apiserver \

--advertise-address=192.168.8.201 \

--insecure-bind-address=0.0.0.0 \

--insecure-port=8080 \

--log-dir=/opt/kubernetes/server/log \

--allow\_privileged=false \

--service-cluster-ip-range=**10.1.0.0/16** \

--service-node-port-range=30000-32767 \

--etcd-servers=http://192.168.8.101:2379,http://192.168.8.102:2379,http://192.168.8.103:2379

**2.kube-scheduler**

<http://kubernetes.io/docs/admin/high-availability/kube-scheduler.yaml>

/opt/kubernetes/server/bin/kube-scheduler \

--address=0.0.0.0 \

--port=10251 \

--log-dir=/opt/kubernetes/server/log \

--master=http://192.168.8.201:8080

**3.kube-controller-manager**

<http://kubernetes.io/docs/admin/high-availability/kube-controller-manager.yaml>

/opt/kubernetes/server/bin/kube-controller-manager \

--address=0.0.0.0 \

--port=10252 \

--log-dir=/opt/kubernetes/server/log \

--master=http://192.168.8.201:8080

**B.Minion组件(kubelet,kube-proxy)**

**Minion1:192.168.8.101**

**Minion2:192.168.8.102**

**Minion3:192.168.8.103**

**1.kubelet**

/opt/kubernetes/server/bin/kubelet \

--node-ip=**192.168.8.101** \

--address=0.0.0.0 \

--port=10250 \

--log-dir=/opt/kubernetes/server/log \

--api-servers=192.168.8.201:8080 \

--configure-cbr0=**false**

**提示:** Minion节点只需修改对应--node-ip即可

**2.kube-proxy**

/opt/kubernetes/server/bin/kube-proxy \

--bind-address=0.0.0.0 \

--log-dir=/opt/kubernetes/server/log \

--master=http://192.168.8.201:8080

**说明:**>=kubernetes-1.2.0 的版本，proxy分发方式是优先采用性能更好效率更高的iptables，当iptables不可用时会采用userspace方式。这也是前面禁用firewalld启用iptables的原因

**问题:**CentOS7.3启动kube-proxy时报**missing br-netfilter module**

照理说，如果br-netfilter没加载的话，iptables分发不生效，换言之，当访问ClusterIP的时候无法正确访问到后端pod的资源。

I0831 15:29:17.507564    2887 server.go:202] Using iptables Proxier.

I0831 15:29:17.507951    2887 proxier.go:209] **missing br-netfilter module or unset br-nf-call-iptables; proxy may not work as intended**

I0831 15:29:17.507991    2887 server.go:215] Tearing down userspace rules.

I0831 15:29:17.529334    2887 conntrack.go:40] Setting nf\_conntrack\_max to 131072

I0831 15:29:17.529752    2887 conntrack.go:57] Setting conntrack hashsize to 32768

I0831 15:29:17.530187    2887 conntrack.go:62] Setting nf\_conntrack\_tcp\_timeout\_established to 86400

**解决:**

但查下来，确定为误报，为什么这么说？

[root@node1 ~]# uname -r

3.10.0-327.el7.x86\_64

[root@node1 ~]# grep 'CONFIG\_BRIDGE\_NETFILTER' /boot/config-3.10.0-327.el7.x86\_64

**CONFIG\_BRIDGE\_NETFILTER**=y

[root@node1 ~]# sysctl -a|grep 'nf-call-iptables'

net.bridge.bridge-**nf-call-iptables** = 1

因为<http://ebtables.netfilter.org/documentation/bridge-nf.html>

Since Linux kernel 3.18-rc1, you have to modprobe br\_netfilter to enable bridge-netfilter.

但CentOS7.3 是直接将br\_netfilter编译进了内核，并且br-nf-call-iptables功能也是开启的。k8s官方issue上也有讨论这个问题，详见

<https://github.com/kubernetes/kubernetes/issues/23385>

However, kubelet prints a warning highlighting the absence of br-netfilter:

1:58.462930 18042 proxier.go:205] missing br-netfilter module or unset br-nf-call-iptables; proxy may not work as intended

This warning seems to be incorrect.

The check that triggers the warning is in Newproxier, located in proxier.go:

if \_, err := os.Stat("/sys/module/br\_netfilter"); os.IsNotExist(err) {

warnBrNetfilter = true

}

**三.查看kubernetes集群状态**

kubectl管理命令需要在Master端执行，当然，也可以通过-s来指定api server主机

[root@node4 ~]# **kubectl version**

Client Version: version.Info{Major:"1", Minor:"4", GitVersion:"v1.4.1", GitCommit:"a16c0a7f71a6f93c7e0f222d961f4675cd97a46b", GitTreeState:"clean", BuildDate:"2016-09-26T18:16:57Z", GoVersion:"go1.6.3", Compiler:"gc", Platform:"linux/amd64"}

Client Version: version.Info{Major:"1", Minor:"4", GitVersion:"v1.4.1", GitCommit:"a16c0a7f71a6f93c7e0f222d961f4675cd97a46b", GitTreeState:"clean", BuildDate:"2016-09-26T18:16:57Z", GoVersion:"go1.6.3", Compiler:"gc", Platform:"linux/amd64"}

[root@node4 ~]# **kubectl get componentstatuses**

NAME                 STATUS    MESSAGE              ERROR

controller-manager   Healthy   ok

scheduler            Healthy   ok

etcd-1               Healthy   {"health": "true"}

etcd-0               Healthy   {"health": "true"}

etcd-2               Healthy   {"health": "true"}

[root@node4 ~]# **kubectl get nodes**

NAME                STATUS    AGE

node1.example.com   Ready     13d

node2.example.com   Ready     2h

node3.example.com   Ready     2h

**提示:**如果这里看到节点状态为**NotReady**，请检查是否有参数错误，个人之前一直是NotReady,原因是在不知道确切意思的情况下误将--configure-cbr0(默认为false)设置为true

[root@node4 ~]# curl -s http://192.168.8.201:8080/api

{

  "kind": "APIVersions",

  "versions": [

    "v1"

  ],

  "serverAddressByClientCIDRs": [

    {

      "clientCIDR": "0.0.0.0/0",

      "serverAddress": "192.168.8.201:6443"

    }

  ]

}

**四.运行容器pods**

**方式一:直接命令行**

[root@node4 ~]# **kubectl run web --image=python3 --replicas=5 "python3 -m http.server 8080"**

deployment "web" created

[root@node4 ~]# **kubectl get pods**

NAME                  READY     STATUS              RESTARTS   AGE

web-799709087-2dzex   0/1       **ContainerCreating**   0          6s

web-799709087-8uyir   0/1       ContainerCreating   0          6s

web-799709087-9hqiw   0/1       ContainerCreating   0          6s

web-799709087-joh1u   0/1       ContainerCreating   0          6s

web-799709087-zwczj   0/1       ContainerCreating   0          6s

[root@node4 ~]# **kubectl get deployment**

NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE

web       5         5         5            0           21s

[root@node4 ~]# **kubectl describe pods web-799709087-2dzex**

Name: web-799709087-2dzex

Namespace: default

Node: node3.example.com/192.168.8.103

Start Time: Sun, 28 Aug 2016 17:42:43 +0800

Labels: pod-template-hash=799709087

run=web

Status: Pending

IP:

Controllers: ReplicaSet/web-799709087

Containers:

  web:

    Container ID:

    Image: python3

    Image ID:

    Port:

    Args:

      python3 -m http.server 8080

    State: Waiting

      Reason: ContainerCreating

    Ready: False

    Restart Count: 0

    Environment Variables:

Conditions:

  Type Status

  Initialized True

  Ready False

  PodScheduled True

No volumes.

QoS Tier: BestEffort

Events:

  FirstSeen LastSeen Count From SubobjectPath Type Reason Message

  --------- -------- ----- ---- ------------- -------- ------ -------

  46s 46s 1 {default-scheduler } Normal Scheduled Successfully assigned web-799709087-2dzex to node3.example.com

  1s 1s 1 {kubelet node3.example.com} Warning FailedSync Error syncing pod, skipping: failed to "StartContainer" for "POD" with ErrImagePull: "**image pull failed for gcr.io/google\_containers/pause-amd64:3.0**, this may be because there are no credentials on this request.  details: (Error response from daemon: {"message":"Get https://gcr.io/v1/\_ping: dial tcp 173.194.72.82:443: i/o timeout"})"

**注意:** 上面的操作后直接卡在**ContainerCreating**上，而实际上我事先在所有minion节点上早将python3的image 从本地仓库中pull了下来，但k8s依赖pause镜像**gcr.io/google\_containers/pause-amd64:3.0**，不同的k8s版本依赖的pause版本不一样

**解决办法:**

**A.VPN**

请自行翻墙

**B.伪装(所有Minion节点)**

docker pull docker.io/kubernetes/pause

docker tag docker.io/kubernetes/pause gcr.io/google\_containers/pause-amd64:3.0

docker rmi -f docker.io/kubernetes/pause

道理很简单，先在minion节点本地准备好依赖的pause镜像，版本名称请与k8s依赖保持一致

以nginx再重新部署一次

[root@node4 ~]# kubectl delete deployment web

deployment "web" deleted

[root@node4 ~]# **kubectl run nginx --image=nginx --replicas=2**

deployment "nginx" created

[root@node4 ~]# kubectl get pods

NAME                     READY     STATUS              RESTARTS   AGE

nginx-3137573019-tza59   0/1       ContainerCreating   0          2s

nginx-3137573019-xro4m   0/1       ContainerCreating   0          2s

[root@node4 ~]# kubectl get pods **-o wide**

NAME                     READY     STATUS    RESTARTS   AGE       IP          NODE

nginx-3137573019-0yuta   1/1       **Running**   0          18s       10.1.68.2   node2.example.com

nginx-3137573019-fun4v   1/1       Running   0          18s       10.1.44.2   node3.example.com

ok，己成功运行了容器nginx

**提示:** 输出支持json|yaml|wide等格式

**方式二:配置文件(yaml,json)**

方便长期维护与跟踪，建议使用配置文件方式来运行pods

<http://kubernetes.io/docs/user-guide/deployments/>

cat >nginx.yaml <<HERE

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

  name: nginx-deployment

spec:

  replicas: 3

  template:

    metadata:

      labels:

        app: nginx

    spec:

      containers:

      - name: nginx

        image: 192.168.8.254:5000/nginx

        ports:

        - containerPort: 80

        - containerPort: 443

HERE

[root@node4 ~]# kubectl create -f nginx.yaml

deployment "nginx-deployment" created

[root@node4 ~]# kubectl rollout status deployments nginx-deployment

deployment nginx-deployment successfully rolled out

[root@node4 ~]# kubectl get deployment

NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE

nginx              2         2         2            2           14m

nginx-deployment   3         3         3            2           23s

[root@node4 ~]# kubectl get pods

NAME                                READY     STATUS    RESTARTS   AGE

nginx-3137573019-0yuta              1/1       Running   0          15m

nginx-3137573019-fun4v              1/1       Running   0          15m

nginx-deployment-2445923563-az8ma   1/1       Running   0          28s

nginx-deployment-2445923563-bqlwd   1/1       Running   0          28s

nginx-deployment-2445923563-vz9l3   1/1       Running   0          28s

[root@node4 ~]# kubectl get rs

NAME                          DESIRED   CURRENT   AGE

nginx-3137573019              2         2         15m

nginx-deployment-2445923563   3         3         30s

**回滚**

<http://kubernetes.io/docs/user-guide/rolling-updates/>

[root@node4 ~]# kubectl rollout undo deployment/nginx-deployment

deployment "nginx-deployment" skipped rollback (DeploymentRollbackRevisionNotFound: Unable to find last revision.)

[root@node4 ~]# kubectl rollout undo deployment/nginx-deployment --to-revision=2

deployment "nginx-deployment" skipped rollback (DeploymentRollbackRevisionNotFound: Unable to find the revision to rollback to.)

**更新**

将pods拉伸至5，只需修改配置文件中的replicas: 5

[root@node4 ~]# kubectl **replace** -f nginx.yaml

deployment "nginx-deployment" replaced

[root@node4 ~]# kubectl get deployment

NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE

nginx              2         2         2            2           27m

nginx-deployment   5         5         5            5           13m

[root@node4 ~]# kubectl get pods

NAME                                READY     STATUS    RESTARTS   AGE

nginx-3137573019-0yuta              1/1       Running   0          27m

nginx-3137573019-fun4v              1/1       Running   0          27m

nginx-deployment-2445923563-az8ma   1/1       Running   0          13m

nginx-deployment-2445923563-bqlwd   1/1       Running   0          13m

nginx-deployment-2445923563-dr9dx   1/1       Running   0          29s

nginx-deployment-2445923563-s9vpy   1/1       Running   0          29s

nginx-deployment-2445923563-vz9l3   1/1       Running   0          13m

**0宕机在线维护Minion主机(cordon,drain,uncordon)**

场景:假如node1.example.com这台主机需要维护，但上面有容器正在运行

[root@node4 ~]# kubectl get pods -o wide

NAME                                READY     STATUS    RESTARTS   AGE       IP          NODE

nginx-3137573019-0yuta              1/1       Running   0          38m       10.1.68.2   node2.example.com

nginx-3137573019-fun4v              1/1       Running   0          38m       10.1.44.2   node3.example.com

nginx-deployment-2445923563-4f2p6   1/1       Running   0          2m        10.1.71.5   **node1.example.com**

nginx-deployment-2445923563-az8ma   1/1       Running   0          24m       10.1.71.4   **node1.example.com**

nginx-deployment-2445923563-g0wkh   1/1       Running   0          2m        10.1.44.4   node3.example.com

nginx-deployment-2445923563-mf0kf   1/1       Running   0          2m        10.1.68.3   node2.example.com

nginx-deployment-2445923563-vz9l3   1/1       Running   0          24m       10.1.44.3   node3.example.com

[root@node4 ~]# kubectl get nodes

NAME                STATUS    AGE

node1.example.com   Ready     15d

node2.example.com   Ready     2d

node3.example.com   Ready     2d

**1.将要维护的Minion节点标识为SchedulingDisabled**

有新部署的时候不会部署到该Minion节点

[root@node4 ~]# kubectl **cordon** node1.example.com

node "node1.example.com" cordoned

[root@node4 ~]# kubectl get nodes

NAME                STATUS                     AGE

node1.example.com   Ready,**SchedulingDisabled**   15d

node2.example.com   Ready                      2d

node3.example.com   Ready                      2d

**2.迁移要维护Minion节点上的容器**

[root@node4 ~]# kubectl **drain** node1.example.com

node "node1.example.com" already cordoned

pod "nginx-deployment-2445923563-4f2p6" deleted

pod "nginx-deployment-2445923563-az8ma" deleted

pod "kubernetes-dashboard-3825951078-a9o82" deleted

pod "busybox-49452825-ldfpw" deleted

node "node1.example.com" drained

[root@node4 ~]# kubectl get pods -o wide

NAME                                READY     STATUS    RESTARTS   AGE       IP          NODE

nginx-3137573019-0yuta              1/1       Running   0          41m       10.1.68.2   node2.example.com

nginx-3137573019-fun4v              1/1       Running   0          41m       10.1.44.2   node3.example.com

nginx-deployment-2445923563-3pwle   1/1       Running   0          22s       10.1.68.4   node2.example.com

nginx-deployment-2445923563-41jqn   1/1       Running   0          22s       10.1.44.5   node3.example.com

nginx-deployment-2445923563-g0wkh   1/1       Running   0          4m        10.1.44.4   node3.example.com

nginx-deployment-2445923563-mf0kf   1/1       Running   0          4m        10.1.68.3   node2.example.com

nginx-deployment-2445923563-vz9l3   1/1       Running   0          26m       10.1.44.3   node3.example.com

**3.维护完成后，撤销SchedulingDisabled 标识**

[root@node4 ~]# kubectl **uncordon** node1.example.com

node "node1.example.com" uncordoned

[root@node4 ~]# kubectl get nodes

NAME                STATUS    AGE

node1.example.com   Ready     15d

node2.example.com   Ready     2d

node3.example.com   Ready     2d

**创建rc(Replication-Controller)**

<http://kubernetes.io/docs/user-guide/replicasets/>

<http://kubernetes.io/docs/user-guide/replication-controller/>

**说明:** rs是下一代的rc，默认创建的Deployment会以rs方式呈现，这也是很多示例中用get rc时看不到任何信息的原因

Replica Set is the **next-generation Replication Controller**. The only difference between a *Replica Set* and a [*Replication Controller*](http://kubernetes.io/docs/user-guide/replication-controller/)right now is the selector support. Replica Set supports the new set-based selector requirements as described in the [labels user guide](http://kubernetes.io/docs/user-guide/labels/#label-selectors) whereas a Replication Controller only supports equality-based selector requirements.

cat >rc-nginx.yaml <<HERE

apiVersion: v1

kind: **ReplicationController**

metadata:

  name: rc-nginx

spec:

  replicas: 3

  selector:

    app: nginx

  template:

    metadata:

      name: nginx

      labels:

        app: nginx

    spec:

      containers:

      - name: nginx

        image: 192.168.8.254:5000/nginx

        ports:

        - containerPort: 80

        - containerPort: 443

HERE

[root@node4 ~]# kubectl create -f rc-nginx.yaml

replicationcontroller "rc-nginx" created

[root@node4 ~]# kubectl get rc

NAME       DESIRED   CURRENT   AGE

rc-nginx   3         3         12s

[root@node4 ~]# kubectl describe rc

Name: rc-nginx

Namespace: default

Image(s): nginx

Selector: app=nginx

Labels: app=nginx

Replicas: 3 current / 3 desired

Pods Status: 3 Running / 0 Waiting / 0 Succeeded / 0 Failed

No volumes.

Events:

  FirstSeen LastSeen Count From SubobjectPath Type Reason Message

  --------- -------- ----- ---- ------------- -------- ------ -------

  1m 1m 1 {replication-controller } Normal SuccessfulCreate Created pod: rc-nginx-56jy0

  1m 1m 1 {replication-controller } Normal SuccessfulCreate Created pod: rc-nginx-9xf3r

  1m 1m 1 {replication-controller } Normal SuccessfulCreate Created pod: rc-nginx-qh49y

再创建一个redis pods

cat >redis.yaml <<HERE

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

  name: redis

spec:

  replicas: 3

  template:

    metadata:

      labels:

        app: redis

    spec:

      containers:

      - name: redis

        image: 192.168.8.254:5000/redis

        ports:

        - containerPort: 6379

HERE

[root@node4 ~]# kubectl create -f redis.yaml

deployment "redis" created

[root@node4 ~]# kubectl get pods

NAME                     READY     STATUS    RESTARTS   AGE

rc-nginx-5lcku           1/1       Running   0          22m

rc-nginx-ffzu1           1/1       Running   0          22m

rc-nginx-mcaxg           1/1       Running   0          22m

redis-3972576797-3s64o   1/1       Running   0          21s

redis-3972576797-q7b0k   1/1       Running   0          21s

redis-3972576797-qc9xf   1/1       Running   0          21s

[root@node4 ~]# kubectl get rs

NAME               DESIRED   CURRENT   AGE

redis-3972576797   3         3         33s

[root@node4 ~]# kubectl get rc

NAME       DESIRED   CURRENT   AGE

rc-nginx   3         3         23m

**删除rc**

[root@node4 ~]# kubectl delete rc rc-nginx

replicationcontroller "rc-nginx" deleted

或

[root@node4 ~]# kubectl delete rc/rc-nginx

replicationcontroller "rc-nginx" deleted

**五.定义service**

<http://kubernetes.io/docs/user-guide/services/>

cat >nginx-service.json <<HERE

{

    "kind": "Service",

    "apiVersion": "v1",

    "metadata": {

        "name": "nginx-service"

    },

    "spec": {

        "selector": {

            "app": "nginx"

        },

        "ports": [

            {

                "name": "http",

                "protocol": "TCP",

                "port": 80,

                "targetPort": 80

            },

            {

                "name": "https",

                "protocol": "TCP",

                "port": 443,

                "targetPort": 443

            }

        ]

    }

}

HERE

[root@node4 ~]# kubectl create -f nginx-service.json

service "nginx-service" created

[root@node4 ~]# kubectl **describe svc nginx-service**

Name: nginx-service

Namespace: default

Labels:

Selector: app=nginx

Type: ClusterIP

IP: **10.1.177.130**

Port: http **80**/TCP

Endpoints: 10.1.43.4:80,10.1.56.3:80,10.1.79.2:80

Port: https **443**/TCP

Endpoints: 10.1.43.4:443,10.1.56.3:443,10.1.79.2:443

Session Affinity: None

No events.

[root@node4 ~]# kubectl get svc nginx-service

NAME         CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE

nginx-service   10.1.177.130           80/TCP,443/TCP   1d

[root@node4 ~]# kubectl get ep nginx-service

NAME         ENDPOINTS                                               AGE

nginx-service   10.1.43.4:443,10.1.56.3:443,10.1.79.2:443 + 3 more...   1d

**说明:**一个Service可以有多个Pods同时工作，，类似负载均衡，当访问Service时，请求会被重定向到其中的一个Pod。但k8s目前采用的是iptables端口映射方式，而Docker官方最新的swarm(>=docker-engine-1.12.0)给我们下了一济猛料，直接采用lvs做负载，光从调度算法上看iptables就逊色得多。

如上显示，10.1.177.130是Service的虚拟地址，映射关系如下

*10.1.177.130:80 -> 10.1.43.4:80,10.1.56.3:80,10.1.79.2:80*

*10.1.177.130:443 -> 10.1.43.4:443,10.1.56.3:80,10.1.79.2:443*

**注:**需要注意的是，Service中的ClusterIP是无法ping通的，但在Minion桥接网络内访问80/443端口时可以访问到对应的资源。可以在创建Service的时候直接指定在--service-cluster-ip-range=**10.1.0.0/16**范围内的合法ClusterIP

实际测试中遇到这样一样问题，当pods运行在某一Minion上时，无法在该Minion上通过ClusterIP访问到pods

比如:当nginx只有一个pods并且运行在Minion2上，此时，在Minion1,Minion3上可以通过ClusterIP访问到对应资源，而在Minion2上则无法通过ClusterIP访问。不知道朋友们是否也遇到过同样的问题，还有待进一步研究

[root@**node1** ~]# curl -I 10.1.177.130

HTTP/1.1 200 OK

Server: nginx/1.11.3

Date: Wed, 31 Aug 2016 08:57:39 GMT

Content-Type: text/html

Content-Length: 612

Last-Modified: Tue, 26 Jul 2016 14:54:48 GMT

Connection: keep-alive

ETag: "579779b8-264"

Accept-Ranges: bytes

[root@node1 ~]# iptables -t nat -S|grep 10.1.177.130

-A KUBE-SERVICES -d **10.1.177.130**/32 -p tcp -m comment --comment "default/my-service:http cluster IP" -m tcp --dport 80 -j KUBE-SVC-I37Z43XJW6BD4TLV

-A KUBE-SERVICES -d **10.1.177.130**/32 -p tcp -m comment --comment "default/my-service:https cluster IP" -m tcp --dport 443 -j KUBE-SVC-CKR3WBBWMIGA5XGG

当有新Service定义时，所有的Minion节点上都会生成对应的iptables条目

**六.kubernetes-dashboard**

<https://github.com/kubernetes/dashboard/releases>

<https://github.com/kubernetes/kube-ui>

Deprecated Web UI for Kubernetes; please try dashboard instead

**说明:**从1.2版本开始, kubernetes提供配套的web UI可视化工具早期的UI项目为kube-ui，该项目基于NodeJS，已经不再维护，转而由新项目dashboard取代，这里只是简单提下这两者的关系

kubernetes-dashboard是以pods的形式运行的，所以需要先准备好镜像，请自行先准备好kubernetes-dashboard镜像

**1.准备kubernetes-dashboard镜像**

**a.可以在Minion节点上事先离线下载好镜像**

docker pull gcr.io/google\_containers/kubernetes-dashboard-amd64:v1.4.0

或

**b.配置文件中指定镜像URL**

将己下好的kubernetes-dashboard上传到私有仓库，再在配置文件kubernetes-dashboard.yaml中指定镜像url，官方最新配置文件为<https://rawgit.com/kubernetes/dashboard/master/src/deploy/kubernetes-dashboard.yaml>

kind: Deployment

apiVersion: extensions/v1beta1

metadata:

  labels:

    app: kubernetes-dashboard

    version: v1.4.0

  name: kubernetes-dashboard

  namespace: kube-system

spec:

  replicas: **1**

  selector:

    matchLabels:

      app: kubernetes-dashboard

  template:

    metadata:

      labels:

        app: kubernetes-dashboard

    spec:

      containers:

      - name: kubernetes-dashboard

        image: **192.168.8.254:5000/kubernetes-dashboard**

        imagePullPolicy: Always

        ports:

        - containerPort: 9090

          protocol: TCP

        args:

**- --apiserver-host=http://192.168.8.201:8080**

        livenessProbe:

          httpGet:

            path: /

            port: 9090

          initialDelaySeconds: 30

          timeoutSeconds: 30

---

kind: Service

apiVersion: v1

metadata:

  labels:

    app: kubernetes-dashboard

  name: kubernetes-dashboard

  namespace: kube-system

spec:

  type: NodePort

  ports:

  - port: 80

    targetPort: 9090

  selector:

    app: kubernetes-dashboard

如上，只需修改镜像URL和apiserver URL即可

**提示:** kubectl默认查看的是default命名空间的内容，而kubernetes-dashboard在kube-system命名空间，所以操作时必须指定命名空间**--namespace=kube-system**

**2.创建deployment和service**

[root@node4 ~]# kubectl create -f kubernetes-dashboard.yaml

deployment "kubernetes-dashboard" created

You have exposed your service on an external port on all nodes in your

cluster.  If you want to expose this service to the external internet, you may

need to set up firewall rules for the service port(s) (tcp:31653) to serve traffic.

See http://releases.k8s.io/release-1.3/docs/user-guide/services-firewalls.md for more details.

service "kubernetes-dashboard" created

**3.查看kubernetes-dashboard状态**

[root@node4 ~]# kubectl get pods **--namespace=kube-system**

NAME                                    READY     STATUS    RESTARTS   AGE

kubernetes-dashboard-2950980434-1d82j   1/1       Running   0          8s

kubernetes-dashboard-2950980434-3v5lz   1/1       Running   0          8s

**删除kubernetes-dashboard**

[root@node4 ~]# kubectl delete deployment kubernetes-dashboard --namespace=kube-system

deployment "kubernetes-dashboard" deleted

[root@node4 ~]# kubectl delete service kubernetes-dashboard --namespace=kube-system

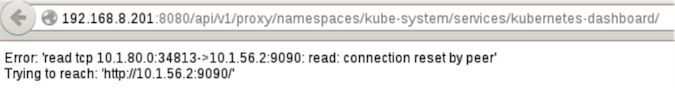
service "kubernetes-dashboard" deleted

**4.访问kubernetes-dashboard**

**i.通过Master转发**

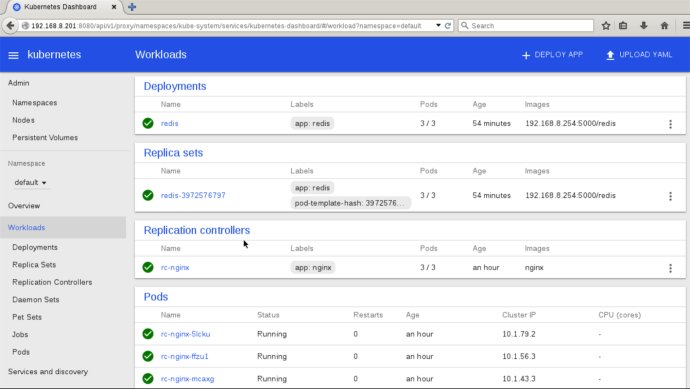
短链接:http://192.168.8.201:8080**/ui**

长链接:http://192.168.8.201:8080/api/v1/proxy/namespaces/kube-system/services/kubernetes-dashboard/

[](http://photo.blog.sina.com.cn/showpic.html#blogid=8ea8e9d50102ww8m&url=http://album.sina.com.cn/pic/002BYBFzzy74stT4MkE70)

**注意:**这种方式需要Master和Minion桥接网络互通，最简单的是在Master上也启一个flannel，如果没有路由，会报如上错误。

flanneld -iface=eth0 -subnet-file=/etc/profile.d/flannel.env -etcd-endpoints=http://192.168.8.101:2379,http://192.168.8.102:2379,http://192.168.8.103:2379

[](http://photo.blog.sina.com.cn/showpic.html#blogid=8ea8e9d50102ww8m&url=http://album.sina.com.cn/pic/002BYBFzzy74stbjgEr83)  
[](http://photo.blog.sina.com.cn/showpic.html#blogid=8ea8e9d50102ww8m&url=http://album.sina.com.cn/pic/002BYBFzzy74suuVm1D15)

**2.ClusterIP**

10.1.124.152

[root@node4 ~]# kubectl get **svc** --namespace=kube-system

NAME                   CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE

kubernetes-dashboard   **10.1.124.152**          80/TCP    38s

**3.pods**

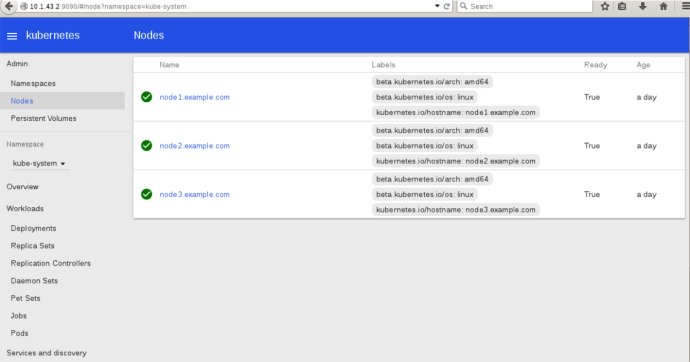
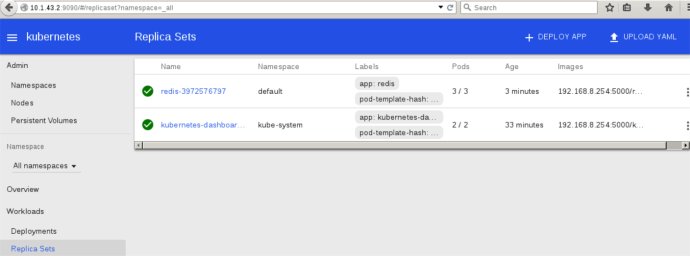
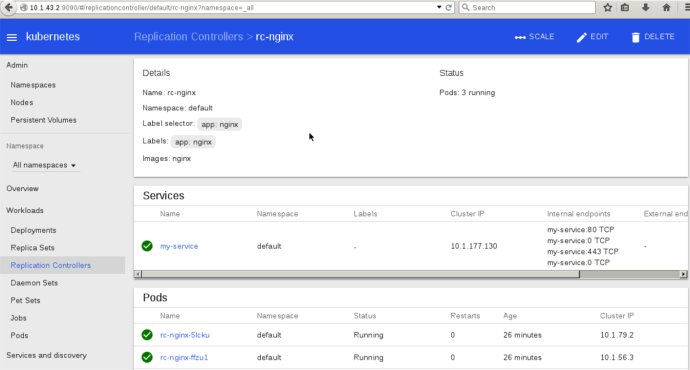
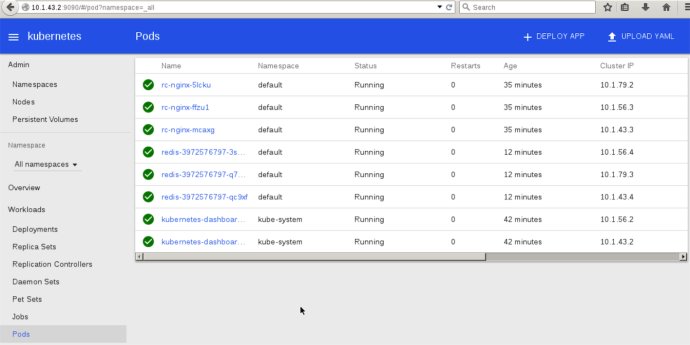
[root@node4 ~]# kubectl get **ep** --namespace=kube-system

NAME                   ENDPOINTS                       AGE

kubernetes-dashboard   **10.1.43.2:9090,10.1.56.2:9090**   30s

10.1.43.2:9090

10.1.56.2:9090

[](http://photo.blog.sina.com.cn/showpic.html#blogid=8ea8e9d50102ww8m&url=http://album.sina.com.cn/pic/002BYBFzzy74spPCR662e)  
  
[](http://photo.blog.sina.com.cn/showpic.html#blogid=8ea8e9d50102ww8m&url=http://album.sina.com.cn/pic/002BYBFzzy74spPG5L63c)  
  
[](http://photo.blog.sina.com.cn/showpic.html#blogid=8ea8e9d50102ww8m&url=http://album.sina.com.cn/pic/002BYBFzzy74spPJBsh15)  
[](http://photo.blog.sina.com.cn/showpic.html#blogid=8ea8e9d50102ww8m&url=http://album.sina.com.cn/pic/002BYBFzzy74sqkHS9qc8)