k8s 部署和使用

Name ： 曲中岭
Email:zlingqu@126.com
Q Q :441869115

第一章 部署准备

1.1 目的

使用 k8s 搭建集群，实现相关功能。

1.2 规划

OS ：CentOS_7.5 x64
Host1 ：172.16.6.37(docker07),master 节点
Host2 ：172.16.6.38(docker08),node 节点
Host3 ：172.16.6.39(docker09),node 节点
Host4 ：172.16.6.40(docker10),node 节点
Docker-ce ：18.09.0

序号	类目	master 节点	node 节点	版本	安装方式
1	IP	172.16.6.37	172.16.6.38/39/40		
2	主机名	docker07	docker08/09/10		
3	docker	√	√	18.09.0	系统服务
4	kubeadm	√	√	v1.13.3	rpm
5	kubectl	√	√	v1.13.3	rpm
6	kubelet	√	√	v1.13.3	rpm
7	kube-proxy	√	√	v1.13.3	container
8	flannel	√	√	v1.13.3	container
9	pause	√	√	3.1	container
10	apiserver	√		v1.13.3	container
11	controller-manager	√		v1.13.3	container
12	scheduler	√		v1.13.3	container
13	etcd	√		3.2.24	container

pod 网络：10.244.0.0/16
service 网络：10.96.0.0/12
节点网络：172.20.0.0/16

1.3 k8s 的两种部署方式

方式 1

kubeadm 方式部署，k8s 可以把 k8s 自身的大部分应用管控起来，即运行于 pod 上，但是 kubelet 和 docker 不能这样实现自托管，这两个主机运行为守护进程，因此，只需要在所有主机都安装 kubelet 和 docker,构建 k8s 集群。相当于是自举。etcd 也是托管于 pod 上运行，使用 kubeadm 进行部署，安装过程相对简单。这些主件的 pod 一般为静态 pod (不属于 k8s 管理)，也可以运行为自托管的 pod.每个主机都要运行 flannel 这个主件，可以运行为 pod。flannel 为动态 pod。

kubeadm 的介绍可以查看如下链接

https://github.com/kubernetes/kubeadm/blob/master/docs/design/design_v1.10.md

安装步骤如下三步

1.master 和 node 安装 kubelet,kubeadm,docker

2.mster 节点: kubeadm init，集群初始化

3.nodes 节点: kubeadm join，node 节点加入集群

方式 2

手动配置，主节点和 node 都主要组件运行为系统级的守护进程，每一步都需要手动处理，如证书和配置过程都是用手动配置的。另外，这种方式在 github 上有 playbook 自动化实现

a).master 节点: 安装 apiserver,scheduler,controller-manager,etcd,flanel

b).node 节点: 安装 kublet,kub-proxy,docker(container engine),flannel,需要多个节点

c).etcd: 安装 etcd 存储服务器，建议配置为高可用

这种方式，可以到 https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG-1.11.md#downloads-for-v1112 下载相关的安装包，注意，master 或者 node 都是要安装 server 端的包。client 是交互时使用，也需要安装，不建议使用这种方式安装，有一定难度。

本文仅介绍使用 kubeadm 实现 k8s 集群安装

第二章 docker 安装

操作对象：所有节点

安装方法有很多，这里选择其中一种，rpm 方式。

2.1 安装

添加 docker 源：

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

或者使用国内阿里/清华的源：

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

yum-config-manager --add-repo https://mirrors.tuna.tsinghua.edu.cn/docker-ce/linux/centos/docker-ce.repo
```

如果找不到 yum-config-manager 命令
执行 yum install yum-utils

从指定源安装 docker-ce ：

```
yum install docker-ce --enablerepo=docker-ce-stable –y
```

安装指定版本

```
yum install docker-ce-18.06.3.ce
yum install docker-ce-18.03.1.ce

systemctl start docker
systemctl enable docker
```

查看是否开机运行：

```
systemctl list-unit-files|grep docker
```

有时候需要执行如下命令：

```
cat >> /usr/lib/systemd/system/docker.service << EOF
LimitNOFILE=1048576
LimitNPROC=1048576
EOF
```

2.2 确认

```
docker version
```

```
[root@docker07 ~]#
[root@docker07 ~]# docker version
Client:
 Version:           18.09.0
 API version:       1.39
 Go version:        go1.10.4
 Git commit:        4d60db4
 Built:             Wed Nov  7 00:48:22 2018
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          18.09.0
  API version:      1.39 (minimum version 1.12)
  Go version:       go1.10.4
  Git commit:       4d60db4
  Built:            Wed Nov  7 00:19:08 2018
  OS/Arch:          linux/amd64
  Experimental:     false
[root@docker07 ~]#
```

2.3 ubuntu 安装（补充）

方法有很多，这里只说一种。

```
curl -sSL https://get.docker.com/ | sh
service start docker
sysv-rc-conf --list|grep docker
update-rc.d  docker  start 90 3 4 5 . stop 20 0 1 2 6 .
sysv-rc-conf --list|grep docker
docker version
```

第三章 kubeadm 等安装

操作主机:所有

所有主机安装 kubeadm、kubectl、kubelet

3.1 添加源

这里使用阿里云，也可使用其他源。另外，需要提醒的是，这几个包有个特别的地方，就是在下载后重新组装成的 rpm，而不是直接下载 rpm，所以必须在线安装。

```
cat >> /etc/yum.repos.d/k8s.repo << EOF
[k8s]
name=aliyun_k8s
baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64/
enabled=1
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg
EOF
```

3.2 安装

```
yum install kubeadm
```

指定版本安装

```
yum -y install kubectl-1.12.2  kubelet-1.12.2  kubeadm-1.12.2
```

```
[root@docker09 ~]# yum install kubeadm
已加载插件：fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: mirrors.shu.edu.cn
 * extras: mirrors.aliyun.com
 * updates: mirrors.163.com
k8s
k8s/primary
k8s
正在解决依赖关系
--> 正在检查事务
---> 软件包 kubeadm.x86_64.0.1.13.3-0 将被 安装
--> 正在处理依赖关系 kubernetes-cni >= 0.6.0, 它被软件包 kubeadm-1.13
--> 正在处理依赖关系 kubelet >= 1.6.0, 它被软件包 kubeadm-1.13.3-0.x8
--> 正在处理依赖关系 kubectl >= 1.6.0, 它被软件包 kubeadm-1.13.3-0.x8
--> 正在处理依赖关系 cri-tools >= 1.11.0, 它被软件包 kubeadm-1.13.3-0
--> 正在检查事务
---> 软件包 cri-tools.x86_64.0.1.12.0-0 将被 安装
---> 软件包 kubectl.x86_64.0.1.13.3-0 将被 安装
---> 软件包 kubelet.x86_64.0.1.13.3-0 将被 安装
--> 正在处理依赖关系 socat, 它被软件包 kubelet-1.13.3-0.x86_64 需要
```

自动安装依赖 kubectl 、kubelet、 kubernetes-cni

```
 Package                                    架构                          版本                                          源
================================================================================================================================
正在安装:
 kubeadm                                    x86_64                        1.13.3-0                                      k8s
为依赖而安装:
 conntrack-tools                            x86_64                        1.4.4-4.el7                                   base
 cri-tools                                  x86_64                        1.12.0-0                                      k8s
 kubectl                                    x86_64                        1.13.3-0                                      k8s
 kubelet                                    x86_64                        1.13.3-0                                      k8s
 kubernetes-cni                             x86_64                        0.6.0-0                                       k8s
 libnetfilter_cthelper                      x86_64                        1.0.0-9.el7                                   base
 libnetfilter_cttimeout                     x86_64                        1.0.0-6.el7                                   base
 libnetfilter_queue                         x86_64                        1.0.2-2.el7_2                                 base
 socat                                      x86_64                        1.7.3.2-2.el7                                 base

事务概要
================================================================================================================================
安装  1 软件包 (+9 依赖软件包)
```

到这里可以查看都有哪些版本发布

https://github.com/kubernetes/kubernetes/releases

查看 k8s 和 docker 版本的对应关系：

第四章 部署和管理

操作对象：所有主机

4.1 环境准备

4.1.1 kubelet 加入开机启动

```
systemctl enable kubelet
```

```
[root@docker07 ~]# systemctl list-unit-files |grep kube
kubelet.service                                 disabled
[root@docker07 ~]#
[root@docker07 ~]# systemctl enable kubelet
Created symlink from /etc/systemd/system/multi-user.target.wants/kubelet.service to /etc/systemd/system/kubelet.service.
[root@docker07 ~]#
[root@docker07 ~]# systemctl list-unit-files |grep kube
kubelet.service                                 enabled
[root@docker07 ~]#
```

此时无法启动 kubelet，因为还未初始化完成，但需要将此服务加入开机启动

4.1.2 禁止 firewalld

```
systemctl stop firewalld
systemctl disable firewalld
```

4.1.3 调整内核参数

主要调整以下三个参数，并将其加入到/etc/rc.local 中。

```
cat >> /etc/rc.local << EOF
echo 1 > /proc/sys/net/bridge/bridge-nf-call-iptables
echo 1 > /proc/sys/net/bridge/bridge-nf-call-ip6tables
echo 1 > /proc/sys/net/ipv4/ip_forward
EOF
```

或者

```
cat >> /usr/lib/sysctl.d/00-system.conf    << EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-arptables = 0
net.ipv4.ip_forward = 1
vm.swappiness=0
EOF
```

然后使其生效：

```
sysctl -p
systemctl restart network
```

如果要使用 ipvs，还要执行如下内容

```
cat > /etc/sysconfig/modules/ipvs.modules <<EOF
#!/bin/bash
modprobe -- ip_vs
modprobe -- ip_vs_rr
modprobe -- ip_vs_wrr
modprobe -- ip_vs_sh
modprobe -- nf_conntrack_ipv4
EOF
chmod 755 /etc/sysconfig/modules/ipvs.modules && bash /etc/sysconfig/modules/ipvs.modules && lsmod | grep -e ip_vs -e nf_conntra
```

保证有如下内容，并生效

```
cat >> /etc/security/limits.conf << EOF
* soft noproc 65535
* hard noproc 65535
* soft nofile 65535
* hard nofile 65535
EOF
```

4.1.4 host 配置

```
cat >> /etc/hosts << EOF
172.16.6.37 docker07
172.16.6.38 docker08
172.16.6.39 docker09
172.16.6.40 docker10
EOF
```

4.1.5 忽略 swap 错误

k8s 默认不支持 swap，如果开启了会 error 报错，处理方式有两种

方法 1：禁止 swap

```
swapoff -a && sed -i '/swap/s&^&#&' /etc/fstab
```

方法 2：强制使用 swap

```
echo "KUBELET_EXTRA_ARGS=\"--fail-swap-on=false\"" > /etc/sysconfig/kubelet
```

并在初始化时添加如下参数

```
--ignore-preflight-errors=Swap
```

4.1.6 网络不通处理

初始化过程，默认会到 gcr.io/google_containers 站点拉取相关 k8s 的镜像信息，所需的镜像信息如 4.2.1 所列出。当前国内不能进行这些站点的访问，也就不能访问进行初始化安装。

解决方法 1：使用国外的代理服务器或则其他方法，使能够从该站点下载对应镜像

解决方法 2：使用 docker 官方的克隆镜像，方法如 4.4.2 所示。

本文档使用方法 2，方法 1 不再演示。

4.2 启动 master 节点

4.2.1 所需的镜像

```
[root@docker10 ~]# kubeadm config images list
k8s.gcr.io/kube-apiserver:v1.13.3
k8s.gcr.io/kube-controller-manager:v1.13.3
k8s.gcr.io/kube-scheduler:v1.13.3
k8s.gcr.io/kube-proxy:v1.13.3
k8s.gcr.io/pause:3.1
k8s.gcr.io/etcd:3.2.24
k8s.gcr.io/coredns:1.2.6
[root@docker10 ~]#
```

k8s.gcr.io/kube-apiserver:v1.13.3

k8s.gcr.io/kube-controller-manager:v1.13.3

k8s.gcr.io/kube-scheduler:v1.13.3

k8s.gcr.io/kube-proxy:v1.13.3

k8s.gcr.io/pause:3.1

k8s.gcr.io/etcd:3.2.24

k8s.gcr.io/coredns:1.2.6

注意 coredns、etcd 和 kube 模块的版本对应关系，可使用命令

kubectl config images list

查到类似如下信息

```
[root@master ~]# kubeadm config images list
I0325 22:31:50.367151    1345 version.go:236] remote version is much newer: v1.13.4; falling back to: stable-1.12
k8s.gcr.io/kube-apiserver:v1.12.6
k8s.gcr.io/kube-controller-manager:v1.12.6
k8s.gcr.io/kube-scheduler:v1.12.6
k8s.gcr.io/kube-proxy:v1.12.6
k8s.gcr.io/pause:3.1
k8s.gcr.io/etcd:3.2.24
k8s.gcr.io/coredns:1.2.2
```

4.2.2 拉取镜像

使用如下命令下载上述列出的镜像

docker pull mirrorgooglecontainers/kube-apiserver:v1.13.3

docker pull mirrorgooglecontainers/kube-controller-manager:v1.13.3

docker pull mirrorgooglecontainers/kube-scheduler:v1.13.3

docker pull mirrorgooglecontainers/kube-proxy:v1.13.3

docker pull mirrorgooglecontainers/pause:3.1

docker pull mirrorgooglecontainers/etcd:3.2.24

docker pull coredns/coredns:1.2.6

各模块还包含 64 位版本，比如 etcd 和 pause 可到如下页面查到。

https://hub.docker.com/r/mirrorgooglecontainers/etcd-amd64/tags

https://hub.docker.com/r/mirrorgooglecontainers/pause-amd64/tags

添加标签：

docker tag mirrorgooglecontainers/kube-apiserver:v1.13.3 k8s.gcr.io/kube-apiserver:v1.13.3

docker tag mirrorgooglecontainers/kube-controller-manager:v1.13.3 k8s.gcr.io/kube-controller-manager:v1.13.3

```
docker tag mirrorgooglecontainers/kube-scheduler:v1.13.3 k8s.gcr.io/kube-scheduler:v1.13.3
docker tag mirrorgooglecontainers/kube-proxy:v1.13.3 k8s.gcr.io/kube-proxy:v1.13.3
docker tag mirrorgooglecontainers/pause:3.1 k8s.gcr.io/pause:3.1
docker tag mirrorgooglecontainers/etcd:3.2.24 k8s.gcr.io/etcd:3.2.24
docker tag coredns/coredns:1.2.6 k8s.gcr.io/coredns:1.2.6
```

修改完成后，查看镜像

```
[root@docker07 ~]# docker images
REPOSITORY                                  TAG                 IMAGE ID            CREATED             SIZE
mirrorgooglecontainers/kube-apiserver       v1.13.3             fe242e556a99        2 weeks ago         181MB
k8s.gcr.io/kube-apiserver                   v1.13.3             fe242e556a99        2 weeks ago         181MB
mirrorgooglecontainers/kube-proxy           v1.13.3             98db19758ad4        2 weeks ago         80.3MB
k8s.gcr.io/kube-proxy                       v1.13.3             98db19758ad4        2 weeks ago         80.3MB
mirrorgooglecontainers/kube-controller-manager   v1.13.3        0482f6400933        2 weeks ago         146MB
k8s.gcr.io/kube-controller-manager          v1.13.3             0482f6400933        2 weeks ago         146MB
mirrorgooglecontainers/kube-scheduler       v1.13.3             3a6f709e97a0        2 weeks ago         79.6MB
k8s.gcr.io/kube-scheduler                   v1.13.3             3a6f709e97a0        2 weeks ago         79.6MB
coredns/coredns                             1.3.1               eb516548c180        5 weeks ago         40.3MB
k8s.gcr.io/coredns                          1.3.1               eb516548c180        5 weeks ago         40.3MB
mirrorgooglecontainers/etcd                 3.3.10              2c4adeb21b4f        2 months ago        258MB
k8s.gcr.io/etcd                             3.3.10              2c4adeb21b4f        2 months ago        258MB
coredns/coredns                             1.2.6               f59dcacceff4        3 months ago        40MB
k8s.gcr.io/coredns                          1.2.6               f59dcacceff4        3 months ago        40MB
mirrorgooglecontainers/etcd                 3.2.24              3cab8e1b9802        5 months ago        220MB
k8s.gcr.io/etcd                             3.2.24              3cab8e1b9802        5 months ago        220MB
k8s.gcr.io/pause                            3.1                 da86e6ba6ca1        14 months ago       742kB
mirrorgooglecontainers/pause                3.1                 da86e6ba6ca1        14 months ago       742kB
[root@docker07 ~]#
[root@docker07 ~]#
```

此时可以删除 mirrorgooglecontainers 相关的标签，我这里不再处理。

4.2.3 初始化集群

使用如下命令下载所需要的镜像，如果不下载，在 init 时自动下载

kubeadm config images pull --kubernetes-version=v1.12.2

使用如下命令初始化集群

```
kubeadm init --kubernetes-version=v1.13.3 --pod-network-cidr=10.244.0.0/16 --service-cidr=10.96.0.0/12 --ignore-preflight-errors=Swap
```

默认从 k8s.gcr.io 仓库拉取镜像，如果要从国内拉取，可以添加参数

```
--image-repository registry.aliyuncs.com/google_containers
```

已经要进行 4.1 步骤，否则会有如下几个警告信息，

```
[root@docker07 ~]#
[root@docker07 ~]# kubeadm init --kubernetes-version=v1.13.3 --pod-network-cidr=10.244.0.0/16 --service-cidr=10.96.0.0/12
[init] Using Kubernetes version: v1.13.3
[preflight] Running pre-flight checks
        [WARNING Firewalld]: firewalld is active, please ensure ports [6443 10250] are open or your cluster may not function correctly
        [WARNING SystemVerification]: this Docker version is not on the list of validated versions: 18.09.0. Latest validated version: 18.06
        [WARNING Hostname]: hostname "docker07" could not be reached
        [WARNING Hostname]: hostname "docker07": lookup docker07 on 1.2.4.8:53: no such host
        [WARNING Service-Kubelet]: kubelet service is not enabled, please run 'systemctl enable kubelet.service'
error execution phase preflight: [preflight] Some fatal errors occurred:
        [ERROR Swap]: running with swap on is not supported. Please disable swap
[preflight] If you know what you are doing, you can make a check non-fatal with `--ignore-preflight-errors=...`
[root@docker07 ~]#
[root@docker07 ~]#
```

其中第二个警告信息说，kubeadm 目前支持最高版本是 18.06，而我们安装的是 18.09，这个警告忽略即可。

```
[root@docker07 ~]#
[root@docker07 ~]# kubeadm init --kubernetes-version=v1.13.3 --pod-network-cidr=10.244.0.0/16 --service-cidr=10.96.0.0/12 --ignore-preflight-errors=Swap
[init] Using Kubernetes version: v1.13.3
[preflight] Running pre-flight checks
        [WARNING Swap]: running with swap on is not supported. Please disable swap
        [WARNING SystemVerification]: this Docker version is not on the list of validated versions: 18.09.0. Latest validated version: 18.06
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Activating the kubelet service
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [docker07 kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.loc
.96.0.1 172.16.6.37]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [docker07 localhost] and IPs [172.16.6.37 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [docker07 localhost] and IPs [172.16.6.37 127.0.0.1 ::1]
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
```

初始化完成后，如下提示：

```
Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

You can now join any number of machines by running the following on each node
as root:

  kubeadm join 172.16.6.37:6443 --token pzviwj.cii3jx0zg61d4gfb --discovery-token-ca-cert-hash sha256:23c

[root@docker07 ~]#
```

记录下上面这句话，用于 node 节点加入集群：

kubeadm join 172.16.6.37:6443 --token pzviwj.cii3jx0zg61d4gfb --discovery-token-ca-cert-hash sha256:23cbb3efbe8a2e2b73cc1442cf4b132b98511a451ffe14dacfe25b9594599c1a

kubeadmin init 主要做了以下工作：

- [init]：指定版本进行初始化操作
- [preflight] ：初始化前的检查和下载所需要的 Docker 镜像文件
- [kubelet-start] ：生成 kubelet 的配置文件"/var/lib/kubelet/config.yaml"，没有这个文件 kubelet 无法启动，所以初始化之前的 kubelet 实际上启动失败。
- [certificates]：生成 Kubernetes 使用的证书，存放在/etc/kubernetes/pki 目录中。
- [kubeconfig] ：生成 KubeConfig 文件，存放在/etc/kubernetes 目录中，组件之间通信需要使用对应文件。
- [control-plane]：使用/etc/kubernetes/manifest 目录下的 YAML 文件，安装 Master 组件。
- [etcd]：使用/etc/kubernetes/manifest/etcd.yaml 安装 Etcd 服务。
- [wait-control-plane]：等待 control-plan 部署的 Master 组件启动。
- [apiclient]：检查 Master 组件服务状态。
- [uploadconfig]：更新配置
- [kubelet]：使用 configMap 配置 kubelet。
- [patchnode]：更新 CNI 信息到 Node 上，通过注释的方式记录。
- [mark-control-plane]：为当前节点打标签，打了角色 Master，和不可调度标签，这样默认就不会使用 Master 节点来运行 Pod。
- [bootstrap-token]：生成 token 记录下来，后边使用 kubeadm join 往集群中添加节点时

会用到

- [addons]：安装附加组件 CoreDNS 和 kube-proxy

配置环境变量：

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

4.2.4 销毁集群

删除所有 worker 节点
kubelet delete node *****
重置集群
kubeadm reset
停止 master 上的 kubelet 服务
停止 master 上的 docker 服务
rm /etc/kubernetes/ -rf
rm /var/lib/kubelet/ -rf
rm /var/lib/etcd/ -rf
可以重新初始化

4.3 启动 worker 节点

4.3.1 安装必要包

```
docker pull mirrorgooglecontainers/kube-proxy:v1.13.3
docker pull mirrorgooglecontainers/pause:3.1
docker tag mirrorgooglecontainers/kube-proxy:v1.13.3 k8s.gcr.io/kube-proxy:v1.13.3
docker tag mirrorgooglecontainers/pause:3.1 k8s.gcr.io/pause:3.1
```

4.3.2 加入集群

使用如下语句在 node 节点上执行即可加入集群，我这里所用了 swap

```
kubeadm join 172.16.6.37:6443 --token pzviwj.cii3jx0zg61d4gfb --discovery-token-ca-cert-hash sha256:23cbb3efbe8a2e2b73cc1442cf4b132b98511a451ffe14dacfe25b9594599c1a --ignore-preflight-errors=Swap
```

如下图：docker08 加入集群：

```
[root@docker08 ~]# kubeadm join 172.16.6.37:6443 --token pzviwj.cii3jx0zg61d4gfb --discovery-token-ca-cert-hash sha256:23cbb3efbe8a2e2b73cc1442c
cfe25b9594599c1a --ignore-preflight-errors=Swap
[preflight] Running pre-flight checks
        [WARNING Swap]: running with swap on is not supported. Please disable swap
        [WARNING SystemVerification]: this Docker version is not on the list of validated versions: 18.09.0. Latest validated version: 18.06
[discovery] Trying to connect to API Server "172.16.6.37:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://172.16.6.37:6443"
[discovery] Requesting info from "https://172.16.6.37:6443" again to validate TLS against the pinned public key
[discovery] Cluster info signature and contents are valid and TLS certificate validates against pinned roots, will use API Server "172.16.6.37:6
[discovery] Successfully established connection with API Server "172.16.6.37:6443"
[join] Reading configuration from the cluster...
[join] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[kubelet] Downloading configuration for the kubelet from the "kubelet-config-1.13" ConfigMap in the kube-system namespace
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Activating the kubelet service
[tlsbootstrap] Waiting for the kubelet to perform the TLS Bootstrap...
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node API object "docker08" as an annotation

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.

[root@docker08 ~]#
```

如果使用 ansible，可使用如下语句一次性加入

```
[root@jenkins ansible-playbook]#
[root@jenkins ansible-playbook]#
[root@jenkins ansible-playbook]# ansible -i hosts k8s-node -m shell -a "kubeadm join 192.168.11.20:644
3 --token oh4fi8.1tnp7ow0p65hhnmv --discovery-token-ca-cert-hash sha256:c57eacc156efe97764dc546cbc8297
9185285c6eac32a2b3959b64d5d8348003"
192.168.11.21 | SUCCESS | rc=0 >>
[preflight] running pre-flight checks
[discovery] Trying to connect to API Server "192.168.11.20:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://192.168.11.20:6443"
```

ansible -i hosts k8s-node -m shell -a "kubeadm join 192.168.11.20:6443 --token oh4fi8.1tnp7ow0p65hhnmv --discovery-token-ca-cert-hash sha256:c57eacc156efe97764dc546cbc82979185285c6eac32a2b3959b64d5d8348003"

4.3.3 排错

如果出现如下错误

```
[root@docker09 ~]#
[root@docker09 ~]# kubeadm join 172.16.6.37:6443 --token pzviwj.cii3jx0zg61d4gfb --discovery-token-ca-cert-hash sh
cfe25b9594599c1a --ignore-preflight-errors=Swap
[preflight] Running pre-flight checks
        [WARNING Swap]: running with swap on is not supported. Please disable swap
        [WARNING SystemVerification]: this Docker version is not on the list of validated versions: 18.09.0. Lates
        [WARNING Service-Kubelet]: kubelet service is not enabled, please run 'systemctl enable kubelet.service'
[discovery] Trying to connect to API Server "172.16.6.37:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://172.16.6.37:6443"
[discovery] Requesting info from "https://172.16.6.37:6443" again to validate TLS against the pinned public key
[discovery] Cluster info signature and contents are valid and TLS certificate validates against pinned roots, will
[discovery] Successfully established connection with API Server "172.16.6.37:6443"
[join] Reading configuration from the cluster...
[join] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
unable to fetch the kubeadm-config ConfigMap: failed to get config map: Unauthorized
[root@docker09 ~]#
[root@docker09 ~]#
```

unable to fetch the kubeadm-config ConfigMap: failed to get config map: Unauthorized 是因为 token 过期了，默认有效期 24 小时。

解决方法：在 master 节点上，使用如下命令重新生产新的 token

另外，token 有效期 24 小时，如果过期使用如下命令重新生成

```
kubeadm token create --print-join-command
kubeadm token create
```

```
[root@docker07 ~]#
[root@docker07 ~]# kubeadm token create
e4l7o1.expvenkjvafg93yt
[root@docker07 ~]#
[root@docker07 ~]#
```

使用新的 token 重新加入集群，如下图

```
[root@docker09 ~]#
[root@docker09 ~]# kubeadm join 172.16.6.37:6443 --token e4l7o1.expvenkjvafg93yt --discovery-token-ca-cert-hash sha256:23cbb3efbe8a2e2b7
cfe25b9594599c1a --ignore-preflight-errors=Swap
[preflight] Running pre-flight checks
        [WARNING Swap]: running with swap on is not supported. Please disable swap
        [WARNING SystemVerification]: this Docker version is not on the list of validated versions: 18.09.0. Latest validated version: 1
        [WARNING Service-Kubelet]: kubelet service is not enabled, please run 'systemctl enable kubelet.service'
[discovery] Trying to connect to API Server "172.16.6.37:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://172.16.6.37:6443"
[discovery] Requesting info from "https://172.16.6.37:6443" again to validate TLS against the pinned public key
[discovery] Cluster info signature and contents are valid and TLS certificate validates against pinned roots, will use API Server "172.1
[discovery] Successfully established connection with API Server "172.16.6.37:6443"
[join] Reading configuration from the cluster...
[join] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[kubelet] Downloading configuration for the kubelet from the "kubelet-config-1.13" ConfigMap in the kube-system namespace
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Activating the kubelet service
[tlsbootstrap] Waiting for the kubelet to perform the TLS Bootstrap...
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node API object "docker09" as an annotation

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.
```

也可使用以下命令查看已经颁发的 token

```
kubeadm token list
```

如果出现以下错误:

请检查网络是否同，node 和 master 时间是否一致

Failed to request cluster info, will try again: [Get https://192.168.11.20:6443/api/v1/namespaces/kube-public/configmaps/cluster-info: x509: certificate has expired or is not yet valid]

```
[root@node1321 ~]#
[root@node1321 ~]#
[root@node1321 ~]# kubeadm join 192.168.11.20:6443 --token i6qmpa.qxyfeotj3sboo491 --discovery-token-c
a-cert-hash sha256:c57eacc156efe97764dc546cbc82979185285c6eac32a2b3959b64d5d8348003
[preflight] running pre-flight checks
        [WARNING RequiredIPVSKernelModulesAvailable]: the IPVS proxier will not be used, because the f
ollowing required kernel modules are not loaded: [ip_vs ip_vs_rr ip_vs_wrr ip_vs_sh] or no builtin ker
nel ipvs support: map[ip_vs:{} ip_vs_rr:{} ip_vs_wrr:{} ip_vs_sh:{} nf_conntrack_ipv4:{}]
you can solve this problem with following methods:
 1. Run 'modprobe -- ' to load missing kernel modules;
2. Provide the missing builtin kernel ipvs support

[discovery] Trying to connect to API Server "192.168.11.20:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://192.168.11.20:6443"
[discovery] Requesting info from "https://192.168.11.20:6443" again to validate TLS against the pinned
 public key
[discovery] Failed to request cluster info, will try again: [Get https://192.168.11.20:6443/api/v1/nam
espaces/kube-public/configmaps/cluster-info: x509: certificate has expired or is not yet valid]
[discovery] Failed to request cluster info, will try again: [Get https://192.168.11.20:6443/api/v1/nam
espaces/kube-public/configmaps/cluster-info: x509: certificate has expired or is not yet valid]
[discovery] Failed to request cluster info, will try again: [Get https://192.168.11.20:6443/api/v1/nam
```

4.3.4 删除 worker 节点

首先将 pod 调度走

```
kubectl drain --ignore-daemonsets docker08
```

等待所有的 pod 都被驱逐后，再删除

```
kubectl get pod –all-namespaces -o wide|grep docker08
kubectl delete node docker08
```

```
[root@docker07 ~]# kubectl delete node docker08
node "docker08" deleted
[root@docker07 ~]#
```

也可以在 worker 节点执行如下命令，直接离开集群

```
kubeadm reset
```

4.3.5 清除网络

当一个 node 从一个集群剔除，并开始加入另一个集群时，可能由于已有的网络，而报类似于如下的错误

```
.t" network: failed to set bridge addr: "cni0" already has an IP address different from 10.244.2.1/24
```

此时需要重置 k8s、停止 docker/kubelet、删除配置、清除网络、重启 docker，重新加入集群

```
kubeadm reset

清空规则:
iptables -F && iptables -t nat -F && iptables -t mangle -F && iptables -X

systemctl stop kubelet
systemctl stop docker
rm -rf /etc/kubernetes/
rm -rf /var/lib/cni/
rm -rf /var/lib/kubelet/*
rm -rf /etc/cni/
ifconfig cni0 down
ifconfig flannel.1 down
ifconfig docker0 down
ip link delete cni0
ip link delete flannel.1
rm -rf /var/run/flannel
systemctl start docker
```

4.4 配置网络

k8s 支持多种网络模型，比如

4.4.1 master 节点

使用如下语句安装：

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

如果出现以下错误，需要开启代理端口：

```
[root@master ~]# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
unable to recognize "https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml": Get http://localhost:8080/api?timeout=32s: dial tcp [::1]:8080: connect: connection refused
unable to recognize "https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml": Get http://localhost:8080/api?timeout=32s: dial tcp [::1]:8080: connect: connection refused
unable to recognize "https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml": Get http://localhost:8080/api?timeout=32s: dial tcp [::1]:8080: connect: connection refused
```

请先执行 5.1.1 的配置

4.4.2 node 节点（选做）

flannel 版本选择，查看如下：

https://quay.io/repository/coreos/flannel?tab=tags

使用如下命令下载镜像：

```
docker pull quay.io/coreos/flannel:v0.11.0-amd64
```

下载后，会被主节点调度，自动配置

如果网络是通的，则不需要这一步，worker 节点会自动下载 flannel.

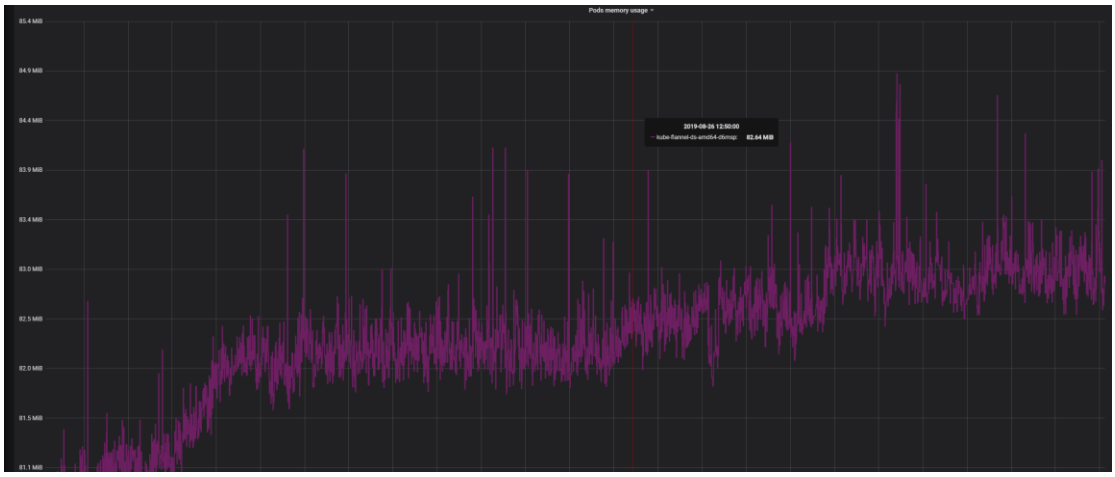
4.4.3 flannel 异常示例

实际使用中发现 flannel 的 pod 偶尔会挂掉，当重启多次之后就不再重启了，导致网络有异常。使用 grafana 的看板发现，flannel 的内存使用超过了 50M，而 flannel 默认的内存限制是 50M，所以调整资源限制后恢复正常。

使用 kubectl describe 或者/var/log/message 中可以看到如下类似内容：

```
be-system(e044427f-bf3e-11e9-979b-20040ff34f74)"
Aug 18 03:35:19 master3 kubelet: E0818 03:35:19.167565    6021 pod_workers.go:190] Error syncing pod e044427f-bf3e-11e9-979b-20040
ff34f74 ("kube-flannel-ds-amd64-4q8n6_kube-system(e044427f-bf3e-11e9-979b-20040ff34f74)"), skipping: failed to "StartContainer" fo
r "kube-flannel" with CrashLoopBackOff: "Back-off 5m0s restarting failed container=kube-flannel pod=kube-flannel-ds-amd64-4q8n6_ku
be-system(e044427f-bf3e-11e9-979b-20040ff34f74)"
Aug 18 03:35:33 master3 kubelet: E0818 03:35:33.167668    6021 pod_workers.go:190] Error syncing pod e044427f-bf3e-11e9-979b-20040
ff34f74 ("kube-flannel-ds-amd64-4q8n6_kube-system(e044427f-bf3e-11e9-979b-20040ff34f74)"), skipping: failed to "StartContainer" fo
r "kube-flannel" with CrashLoopBackOff: "Back-off 5m0s restarting failed container=kube-flannel pod=kube-flannel-ds-amd64-4q8n6_ku
be-system(e044427f-bf3e-11e9-979b-20040ff34f74)"
Aug 18 03:35:46 master3 kubelet: E0818 03:35:46.168269    6021 pod_workers.go:190] Error syncing pod e044427f-bf3e-11e9-979b-20040
ff34f74 ("kube-flannel-ds-amd64-4q8n6_kube-system(e044427f-bf3e-11e9-979b-20040ff34f74)"), skipping: failed to "StartContainer" fo
r "kube-flannel" with CrashLoopBackOff: "Back-off 5m0s restarting failed container=kube-flannel pod=kube-flannel-ds-amd64-4q8n6_ku
be-system(e044427f-bf3e-11e9-979b-20040ff34f74)"
Aug 18 03:35:59 master3 kubelet: E0818 03:35:59.167501    6021 pod_workers.go:190] Error syncing pod e044427f-bf3e-11e9-979b-20040
ff34f74 ("kube-flannel-ds-amd64-4q8n6_kube-system(e044427f-bf3e-11e9-979b-20040ff34f74)"), skipping: failed to "StartContainer" fo
r "kube-flannel" with CrashLoopBackOff: "Back-off 5m0s restarting failed container=kube-flannel pod=kube-flannel-ds-amd64-4q8n6_ku
be-system(e044427f-bf3e-11e9-979b-20040ff34f74)"
Aug 18 03:36:14 master3 kubelet: E0818 03:36:14.168097    6021 pod_workers.go:190] Error syncing pod e044427f-bf3e-11e9-979b-20040
ff34f74 ("kube-flannel-ds-amd64-4q8n6_kube-system(e044427f-bf3e-11e9-979b-20040ff34f74)"), skipping: failed to "StartContainer" fo
r "kube-flannel" with CrashLoopBackOff: "Back-off 5m0s restarting failed container=kube-flannel pod=kube-flannel-ds-amd64-4q8n6_ku
```

修改为 200M 内存限制后，如下图，内存使用率长期稳定在 80-85M 之间。



4.5 多 master 节点部署

4.5.1 规划

haproxy + keepalievd	192.168.11.19	192.168.12.19	192.168.13.19
k8s-master	192.168.11.20	192.168.12.20	192.168.13.20
k8s-worker	192.168.11.21	192.168.12.21	192.168.13.21
k8s-worker	192.168.11.22	192.168.12.22	192.168.13.22
k8s-worker	192.168.11.23	192.168.12.23	192.168.13.23

其中 VIP：192.168.11.3/24

4.5.2 keepalived 部署

在 192.168.11/12/13.19 上安装 keepalived

```
yum install keepalived
```

配置文件如下，三台中，不同的地方使用黄色背景标注
cat /etc/keepalived/keepalived.conf

```
! Configuration File for keepalived

global_defs {
    notification_email {
        acassen@firewall.loc
        failover@firewall.loc
        sysadmin@firewall.loc
    }
    notification_email_from Alexandre.Cassen@firewall.loc
    smtp_server 192.168.200.1
    smtp_connect_timeout 30
    router_id 1119
    vrrp_mcast_group4 224.0.100.100
}
```

```
vrrp_instance k8s_api_server {
    state MASTER
    interface eth0
    virtual_router_id 59
    priority 100
    advert_int 1

    authentication {
        auth_type PASS
        auth_pass 1111fdsfas
    }

    virtual_ipaddress {
        192.168.11.3/24 dev eth0 label eth0:0

    }
}
```

```
systemctl enable keepalived
systemctl start keepalived
```

4.5.3 haproxy 部署

在 192.168.11/12/13.19 上安装 haproxy

```
yum install haproxy
```

配置文件如下，三个节点配置完全相同

```
cat /etc/haproxy/haproxy.cfg
global
        chroot  /var/lib/haproxy
        daemon
        group haproxy
        user haproxy
        log 127.0.0.1 local2
        pidfile /var/lib/haproxy/haproxy.pid
        maxconn 20000
        spread-checks 3
        nbproc 8
        stats socket /var/lib/haproxy/stats
```

```
defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    option http-server-close
    option forwardfor       except 127.0.0.0/8
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 3000

frontend  k8s-api  *:6443
    mode                tcp
    default_backend     k8s-master
    option              tcplog

backend k8s-master
    mode        tcp
    balance     roundrobin
    option      tcplog
    server      master1120  192.168.11.20:6443 check maxconn 2000
    server      master1220  192.168.12.20:6443 check maxconn 2000
    server      master1320  192.168.13.20:6443 check maxconn 2000
```

```
systemctl enable haproxy
systemctl start haproxy
```

4.5.4 k8s 初始化

在 11.20 上，使用如下命令显示初始化配置并保存到文件中

```
kubeadm config print init-defaults > k8s-init.yaml
```

修改相关内容，其中红色部分是我修改过的。

```
apiVersion: kubeadm.k8s.io/v1beta1
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  token: abcdef.0123456789abcdef
  ttl: 24h0m0s
  usages:
  - signing
  - authentication
kind: InitConfiguration
localAPIEndpoint:
  advertiseAddress: 192.168.11.20
  bindPort: 6443
nodeRegistration:
  criSocket: /var/run/dockershim.sock
  name: master1120
  taints:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
---
apiServer:
  timeoutForControlPlane: 4m0s
apiVersion: kubeadm.k8s.io/v1beta1
certificatesDir: /etc/kubernetes/pki
clusterName: kubernetes
controlPlaneEndpoint: "192.168.11.3:6443"
controllerManager: {}
dns:
  type: CoreDNS
etcd:
  local:
    dataDir: /var/lib/etcd
imageRepository: k8s.gcr.io
#imageRepository: registry.aliyuncs.com/google_containers

kind: ClusterConfiguration
kubernetesVersion: v1.13.5
networking:
  dnsDomain: cluster.local
  podSubnet: 10.244.0.0/16
  serviceSubnet: 10.96.0.0/12
scheduler: {}
```

如果要使用 ipvs 而不是 iptables，此文件尾部还应添加如下内容

```
---
apiVersion: kubeproxy.config.k8s.io/v1alpha1
kind: KubeProxyConfiguration
mode: "ipvs"
```

使用如下命令初始化集群

```
kubeadm init --config k8s-init.yaml
```

4.5.5 添加 master

在 1120 上传递证书到其他 master

```
cat > k8s.sh << EOF
USER=root
IP="master1220 master1320"
for host in \${IP}; do
    ssh "\${USER}"@\$host "mkdir -p /etc/kubernetes/pki/etcd"
    scp /etc/kubernetes/pki/ca.* "\${USER}"@\$host:/etc/kubernetes/pki/
    scp /etc/kubernetes/pki/sa.* "\${USER}"@\$host:/etc/kubernetes/pki/
    scp /etc/kubernetes/pki/front-proxy-ca.* "\${USER}"@\$host:/etc/kubernetes/pki/
    scp /etc/kubernetes/pki/etcd/ca.* "\${USER}"@\$host:/etc/kubernetes/pki/etcd/
    scp /etc/kubernetes/admin.conf "\${USER}"@\$host:/etc/kubernetes/
done
EOF
```

sh k8s.sh

然后在 1220、1320 上节点上执行 join

```
kubeadm join 192.168.11.3:6443 --token abcdef.0123456789abcdef --discovery-token-ca-cert-hash sha256:ceef8a8b804a884fa16ab8e86c3eaa95aed01a95eae9285b68170bf56899f0d9 --experimental-control-plane
```

注意最后的--experimental-control-plane 参数，普通 node 加入集群不需要这个参数。

参考: https://kubernetes.io/docs/setup/independent/high-availability/

另外，token 有效期 24 小时，如果过期使用如下命令重新生成

```
kubeadm token create --print-join-command
```

也可使用以下命令查看已经颁发的 token

```
kubeadm token list
```

注意：1.15 版本之后，部分参数有变化，可参考 https://kubernetes.io/zh/docs/setup/production-environment/tools/kubeadm/high-availability/

4.5.6 添加 worker 节点

和单 master 一样，添加 worker 节点，并配置 flannel 网络。

4.5.7 观察

```
[root@master1120 ~]# kubectl get nodes
NAME          STATUS   ROLES    AGE   VERSION
master1120    Ready    master   54m   v1.13.5
master1220    Ready    master   53m   v1.13.5
master1320    Ready    master   52m   v1.13.5
node1121      Ready    <none>   49m   v1.13.5
node1122      Ready    <none>   50m   v1.13.5
node1123      Ready    <none>   50m   v1.13.5
node1221      Ready    <none>   41m   v1.13.5
node1222      Ready    <none>   40m   v1.13.5
node1223      Ready    <none>   41m   v1.13.5
node1321      Ready    <none>   40m   v1.13.5
node1322      Ready    <none>   40m   v1.13.5
node1323      Ready    <none>   40m   v1.13.5
[root@master1120 ~]#
```

```
NAME                                      READY   STATUS    RESTARTS   AGE
coredns-86c58d9df4-dwlgr                  1/1     Running   0          54m
coredns-86c58d9df4-r8ggq                  1/1     Running   0          54m
etcd-master1120                           1/1     Running   0          53m
etcd-master1220                           1/1     Running   0          53m
etcd-master1320                           1/1     Running   0          52m
kube-apiserver-master1120                 1/1     Running   0          53m
kube-apiserver-master1220                 1/1     Running   0          53m
kube-apiserver-master1320                 1/1     Running   0          52m
kube-controller-manager-master1120        1/1     Running   1          53m
kube-controller-manager-master1220        1/1     Running   0          53m
kube-controller-manager-master1320        1/1     Running   0          52m
```

```
kube-scheduler-master1120                 1/1     Running   1          53m
kube-scheduler-master1220                 1/1     Running   0          53m
kube-scheduler-master1320                 1/1     Running   0          52m
```

查看 etcd 健康状况:

```
docker run --rm -it \
--net host \
-v /etc/kubernetes:/etc/kubernetes quay.io/coreos/etcd etcdctl \
--cert-file /etc/kubernetes/pki/etcd/peer.crt \
--key-file /etc/kubernetes/pki/etcd/peer.key \
--ca-file /etc/kubernetes/pki/etcd/ca.crt \
--endpoints https://127.0.0.1:2379 cluster-health
```

```
[root@master1120 ~]# docker run --rm -it \
> --net host \
> -v /etc/kubernetes:/etc/kubernetes quay.io/coreos/etcd etcdctl \
> --cert-file /etc/kubernetes/pki/etcd/peer.crt \
> --key-file /etc/kubernetes/pki/etcd/peer.key \
> --ca-file /etc/kubernetes/pki/etcd/ca.crt \
> --endpoints https://127.0.0.1:2379 cluster-health
member 32225956760089 is healthy: got healthy result from https://192.168.12.20:2379
member 4ebbb444774b731c is healthy: got healthy result from https://192.168.11.20:2379
member aa54a44501678029 is healthy: got healthy result from https://192.168.13.20:2379
cluster is healthy
[root@master1120 ~]#
```

同时返回三个 etcd 集群的结果。

4.5.8 etcd 集群操作

查看集群成员:

```
docker run --rm -it --net host -v /etc/kubernetes:/etc/kubernetes quay.io/coreos/etcd etcdctl --cert-file /etc/kubernetes/pki/etcd/peer.crt --key-file /etc/kubernetes/pki/etcd/peer.key --ca-file /etc/kubernetes/pki/etcd/ca.crt --endpoints https://127.0.0.1:2379 member list
```

```
[root@master1220 ~]#
[root@master1220 ~]# docker run --rm -it --net host -v /etc/kubernetes:/etc/kubernetes quay.io/coreos/etcd etcdctl --cert-file
cd/peer.key --ca-file /etc/kubernetes/pki/etcd/ca.crt --endpoints https://127.0.0.1:2379 member list
32225956760089: name=master1220 peerURLs=https://192.168.12.20:2380 clientURLs=https://192.168.12.20:2379 isLeader=true
4ebbb444774b731c: name=master1120 peerURLs=https://192.168.11.20:2380 clientURLs=https://192.168.11.20:2379 isLeader=false
aa54a44501678029: name=master1320 peerURLs=https://192.168.13.20:2380 clientURLs=https://192.168.13.20:2379 isLeader=false
[root@master1220 ~]#
```

删除集群成员：
member remove {ID}

```
[root@master1220 ~]#
[root@master1220 ~]# docker run --rm -it --net host -v /etc/kubernetes:/etc/kubernetes quay.io/coreos/etcd etcdctl --cert-file /etc/kubernet
cd/peer.key --ca-file /etc/kubernetes/pki/etcd/ca.crt --endpoints https://127.0.0.1:2379 member remove 4ebbb444774b731c
Removed member 4ebbb444774b731c from cluster
[root@master1220 ~]#
[root@master1220 ~]# docker run --rm -it --net host -v /etc/kubernetes:/etc/kubernetes quay.io/coreos/etcd etcdctl --cert-file /etc/kubernet
cd/peer.key --ca-file /etc/kubernetes/pki/etcd/ca.crt --endpoints https://127.0.0.1:2379 member list
322259567600 89: name=master1220 peerURLs=https://192.168.12.20:2380 clientURLs=https://192.168.12.20:2379 isLeader=true
aa54a44501678029: name=master1320 peerURLs=https://192.168.13.20:2380 clientURLs=https://192.168.13.20:2379 isLeader=false
[root@master1220 ~]#
[root@master1220 ~]#
```

添加成员
member add master1120 https://192.168.11.20:2380

```
[root@master1220 manifests]#
[root@master1220 manifests]# docker run --rm -it --net host -v /etc/kubernetes:/etc/kubernetes quay.io/coreos/etcd etcdctl --cert-file /etc/kubernete
s/pki/etcd/peer.key --ca-file /etc/kubernetes/pki/etcd/ca.crt --endpoints https://127.0.0.1:2379 member add master1120 https://192.168.11.20:2380
Added member named master1120 with ID ba63f35307d0914b to cluster

ETCD_NAME="master1120"
ETCD_INITIAL_CLUSTER="master1220=https://192.168.12.20:2380,master1320=https://192.168.13.20:2380,master1120=https://192.168.11.20:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
[root@master1220 manifests]#
```

4.6 部署 web-ui

到这里可以看到版本对应关系：
https://github.com/kubernetes/dashboard/releases
比如我这里选择最新的 v1.10.1 版本，自动拉取的镜像是 k8s.gcr.io/kubernetes-dashboard-amd64:v1.10.1

如果网络不通，可到这里选择镜像版本
https://hub.docker.com/r/mirrorgooglecontainers/kubernetes-dashboard-amd64/tags
比如我选择最新的 1.10.1 版本

```
docker pull mirrorgooglecontainers/kubernetes-dashboard-amd64:v1.10.1
docker tag mirrorgooglecontainers/kubernetes-dashboard-amd64:v1.10.1 k8s.gcr.io/kubernetes-dashboard-amd64:v1.10.1
```

4.6.1 https-token

注意：只有 https 才可以使用 token 或者 kubeconfig 登陆。
执行如下语句即可安装完成

```
wget https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.0/src/deploy/recommended/kubernetes-dashboard.yaml
```

设定未 nodeport 方式
vim kubernetes-dashboard.yaml
如下图，增加如下两行

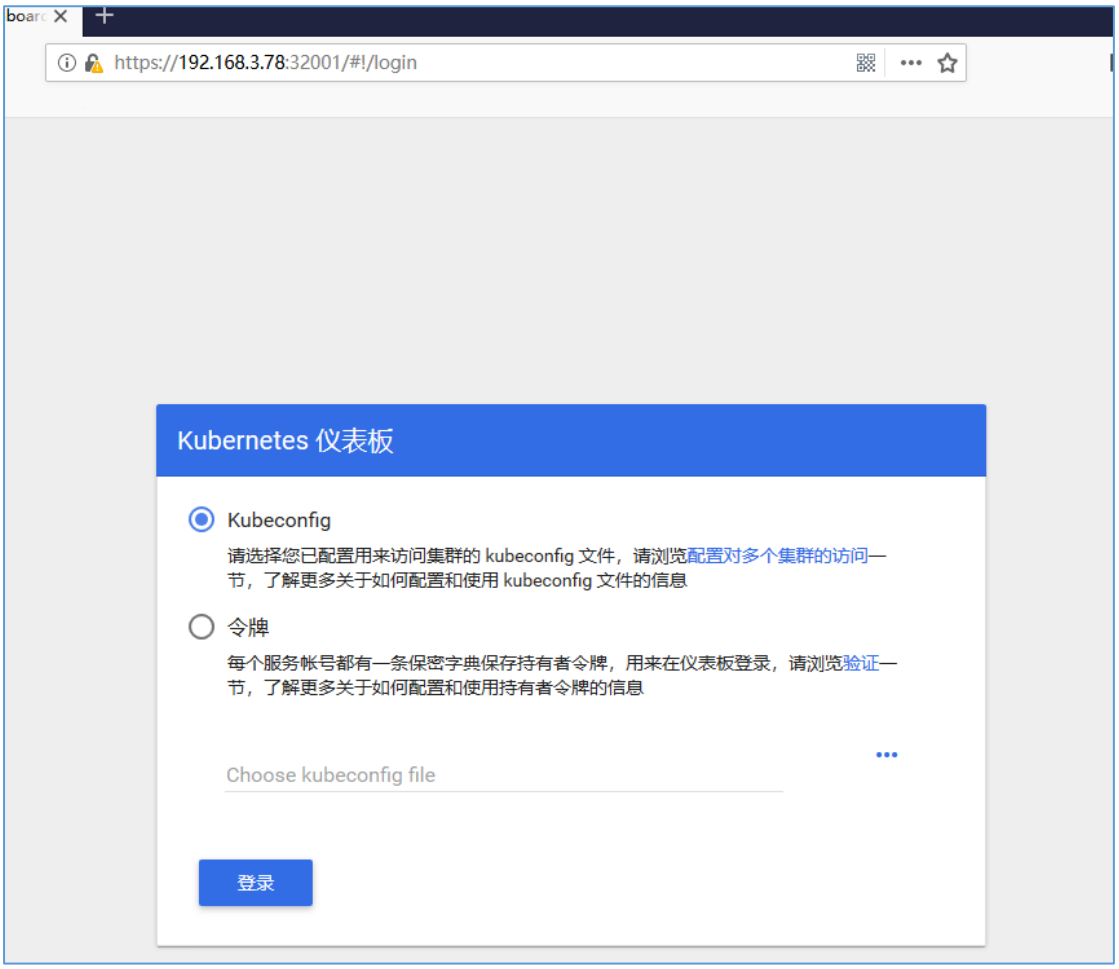
```
type: NodePort
nodePort: 32001
```

```yaml
kind: Service
apiVersion: v1
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  type: NodePort
  ports:
    - port: 443
      targetPort: 8443
      nodePort: 32001
  selector:
    k8s-app: kubernetes-dashboard
```

部署

```
kubectl apply -f kubernetes-dashboard.yaml
```

使用火狐浏览器访问：
https://192.168.3.78:32001/#!/login



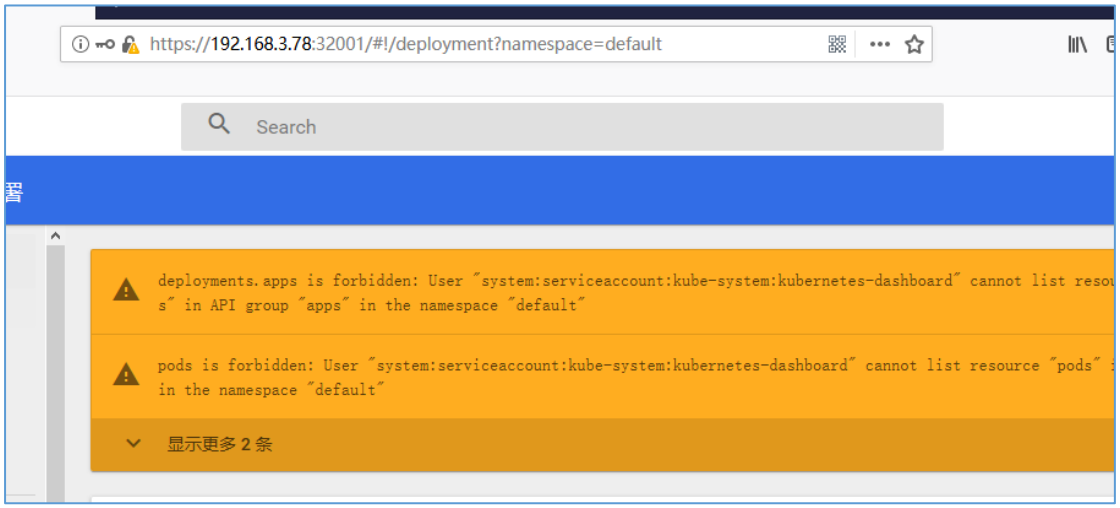
查看 token:

```
kubectl get secret $(kubectl get secret -n kube-system |awk '/^kubernetes-dashboard-token/{print $1}') -n kube-system -o jsonpath={.data.token}|base64 -d
```

或者

```
kubectl describe secret $(kubectl get secret -n kube-system |awk '/^kubernetes-dashboard-token/{print $1}') -n kube-system|awk '/^token/{print $2}'
```

官方文件中创建的 sa 只是为了初始化 dashboasd 的 pod，token 权限太小，如下图



下面创建权限更大的 sa，用于访问

创建 sa

```
kubectl apply -f kubernetes-dashboard-admin.yaml
```

yaml 文件内容：

cat kubernetes-dashboard-admin.yaml

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard-admin
  namespace: kube-system

---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: kubernetes-dashboard-admin
  labels:
    k8s-app: kubernetes-dashboard
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kubernetes-dashboard-admin
  namespace: kube-system
```

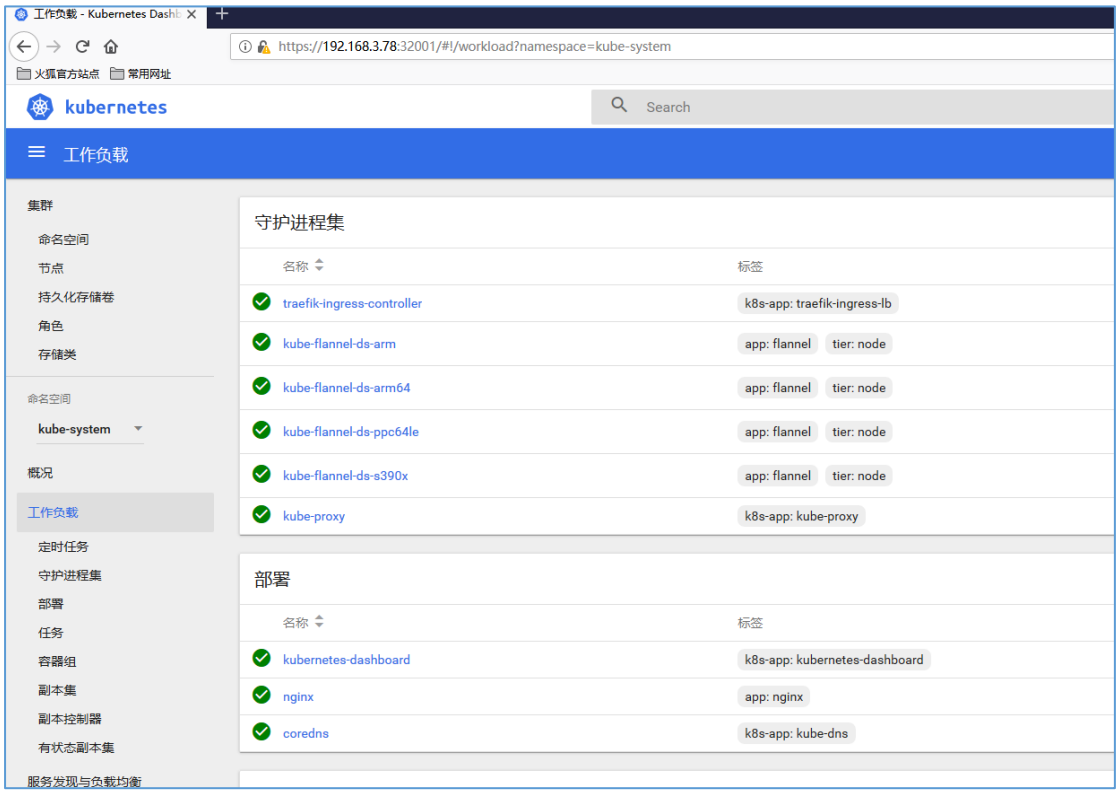
获取新的 token，这个 sa 和运行 dashbord 的 sa 可以不同，使用哪个 token 登陆，就获得那个 sa 的权限，比如上面我新建的是 sa 是 kubernetes-dashboard-admin

```
kubectl get secret $(kubectl get secret -n kube-system |awk '/^kubernetes-dashboard-admin-token/{print $1}') -n kube-system -o jsonpath={.data.token}|base64 -d
```

或者

```
kubectl describe secret $(kubectl get secret -n kube-system |awk '/^kubernetes-dashboard-admin-token/{print $1}') -n kube-system|awk '/^token/{print $2}'
```

使用新的 token，再次访问



4.6.2 https--kubeconfig

4.6.3 http

部署

```
kubectl apply -f kubernetes-dashboard.yaml
```

文件内容如下：

cat kubernetes-dashboard.yaml

```
# ------------------- Dashboard Service Account ------------------- #
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
---
# ------------------- Dashboard ClusterRoleBinding ------------------- #
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kubernetes-dashboard-cluster-role
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/cluster-service: "true"
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kubernetes-dashboard
  namespace: kube-system

---
# ------------------- Dashboard Deployment ------------------- #

kind: Deployment
apiVersion: apps/v1
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      k8s-app: kubernetes-dashboard
  template:
```

```
    metadata:
      labels:
        k8s-app: kubernetes-dashboard
    spec:
      containers:
      - name: kubernetes-dashboard
        image: k8s.gcr.io/kubernetes-dashboard-amd64:v1.10.1
        ports:
        - containerPort: 9090
          protocol: TCP
        volumeMounts:
          # Create on-disk volume to store exec logs
        - mountPath: /tmp
          name: tmp-volume
        livenessProbe:
          httpGet:
            path: /
            port: 9090
          initialDelaySeconds: 30
          timeoutSeconds: 30
      volumes:
      - name: tmp-volume
        emptyDir: {}
      serviceAccountName: kubernetes-dashboard
      # Comment the following tolerations if Dashboard must not be deployed on master
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule

---
# ------------------- Dashboard Service ------------------- #

kind: Service
apiVersion: v1
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  ports:
  - port: 80
    targetPort: 9090
  selector:
```

```
    k8s-app: kubernetes-dashboard
```

访问方式：http://***:32001,无需输入 kubeconfig 或者 token，直接访问。

4.6.4 使用 ingress

创建 ingress 规则：

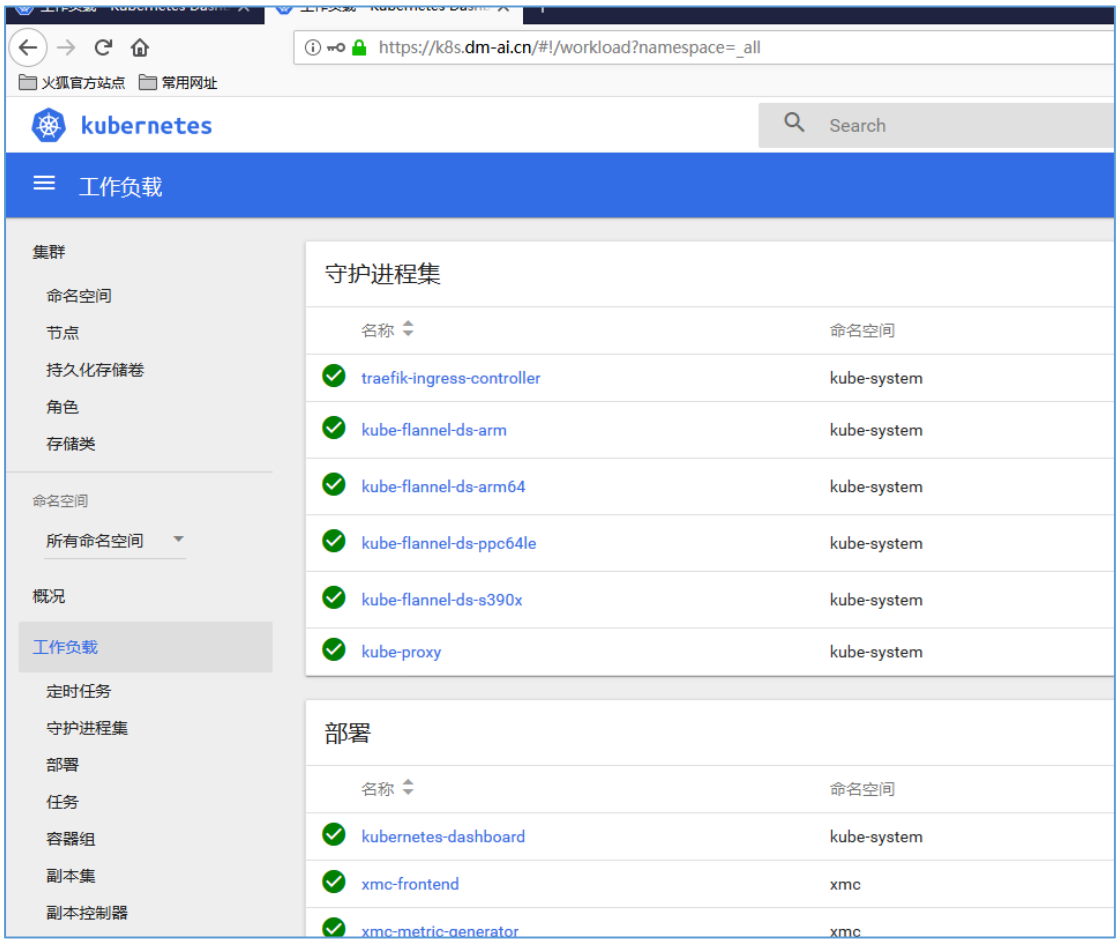
```
kubectl apply -f kubernetes-dashboard-ingress.yml
```

规则内容如下，其中红色部门标识使用 https 访问时的配置：

cat kubernetes-dashboard-ingress.yml

```
---
kind: Ingress
apiVersion: extensions/v1beta1
metadata:
  name: kubernetes-dashboard
  namespace: kube-system
  annotations:
    kubernetes.io/ingress.class: traefik
spec:
  #tls:
  #   - secretName: https-key-secret
  rules:
  - host: k8s.dm-ai.cn
    http:
      paths:
      - path: /
        backend:
          serviceName: kubernetes-dashboard
          #servicePort: 443
          servicePort: 9090
```

如果使用 https，同样要使用 token 或者 kubeconfig，效果如下图：



注意，使用了 ingress 后，dashboard 可不使用 nodeport 方式。

4.7 部署 helm

4.7.1 下载和配置 helm

下载地址：

https://github.com/helm/helm/releases

例如：

wget https://get.helm.sh/helm-v2.14.3-linux-amd64.tar.gz

tar xf helm-v2.14.3-linux-amd64.tar.gz

cp linux-amd64/helm /usr/local/bin/

cp linux-amd64/tiller /usr/local/bin/

4.7.2 安装 tiller

kubectl apply -f helm.yml

helm.yml 内容如下：

```
apiVersion: v1
```

```
kind: ServiceAccount
metadata:
  name: tiller
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: tiller-cluster-rule
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: tiller
  namespace: kube-system
```

默认源是 gcr.io，
如果网络不通，可以更好为国内源：

```
helm repo add stable https://kubernetes.oss-cn-hangzhou.aliyuncs.com/charts
helm repo update
```

查看当前使用的源：

```
helm repo list
```

初始化：

```
helm init
```

也可以指定版本初始化：

```
helm init -i gcr.io/kubernetes-helm/tiller:v2.14.3
```

初始化后，在 kube-system 下面会有一个 tiller 的 pod，可使用 helm version 查看版本

```
[root@master ~]#
[root@master ~]# kubectl get pod -nkube-system|grep tiller
tiller-deploy-779784fbd6-89h8t          1/1     Running   0          8m36s
[root@master ~]#
[root@master ~]#
[root@master ~]#
[root@master ~]# helm version
Client: &version.Version{SemVer:"v2.14.3", GitCommit:"0e7f3b6637f7af8fcfddb3d2941fcc7cbebb0085", GitTreeState:"clean"}
Server: &version.Version{SemVer:"v2.14.3", GitCommit:"0e7f3b6637f7af8fcfddb3d2941fcc7cbebb0085", GitTreeState:"clean"}
[root@master ~]#
[root@master ~]#
```

4.7.3 协助 tiller

```
helm reset
```

4.8 节点维护

标记节点不可用，标记后不会再有新的 pod 被调度到该节点

```
kubectl cordon node1123
```

节点标记为不可用，并迁移已经运行在此节点上的 pod

```
kubectl drain node1123
```

驱逐 pod，但忽略 daemonset 类型的

```
kubectl drain --ignore-daemonsets node1123
```

标记后效果如下：

```
[root@master1120 deployment]# kubectl get node
NAME          STATUS                     ROLES    AGE    VERSION
gpu6867       Ready                      <none>   25d    v1.15.5
master1120    Ready                      master   208d   v1.15.5
master1220    Ready                      master   234d   v1.15.5
master1320    Ready                      master   234d   v1.15.5
node1117      Ready                      <none>   175d   v1.15.5
node1118      Ready                      <none>   175d   v1.15.5
node1119      Ready                      <none>   175d   v1.15.5
node1121      Ready                      <none>   233d   v1.15.5
node1122      Ready                      <none>   212d   v1.15.5
node1123      Ready,SchedulingDisabled   <none>   233d   v1.15.5
node1124      Ready                      <none>   143d   v1.15.5
node1125      Ready                      <none>   143d   v1.15.5
node1126      Ready                      <none>   143d   v1.15.5
```

取消标记，取消后节点可被调度

```
kubectl uncordon node1123
```

4.9 证书管理

部分内容适用于 k8s-1.15 版本及之后

4.9.1 证书说明

	路径/etc/kubrnetes	上级证书	位于	有效	举例
etcd相关证书	pki/etcd/ca.key		etcd节点		
	pki/etcd/ca.crt	pki/etcd/ca.key	etcd节点，k8s-master节点	10年	Not Before: Jul 29 06:25:29 2019 GMT Not After : Jul 26 06:25:29 2029 GMT
	pki/etcd/peer.key		etcd节点		
	pki/etcd/peer.crt	pki/etcd/ca.key	etcd节点	1年	Not Before: Jul 29 06:25:29 2019 GMT Not After : Jul 28 06:25:33 2020 GMT
	pki/etcd/server.key		etcd节点		
	pki/etcd/server.crt	pki/etcd/ca.key	etcd节点	1年	Not Before: Jul 29 06:25:29 2019 GMT Not After : Jul 28 06:25:33 2020 GMT
	pki/etcd/healthcheck-client.key		etcd节点		
	pki/etcd/healthcheck-client.crt	pki/etcd/ca.key	etcd节点	1年	Not Before: Jul 29 06:25:29 2019 GMT Not After : Jul 28 06:25:34 2020 GMT
	pki/apiserver-etcd-client.key		k8s-master节点		
	pki/apiserver-etcd-client.crt	pki/etcd/ca.key	k8s-master节点	1年	Not Before: Jul 29 06:25:29 2019 GMT Not After : Jul 28 06:25:34 2020 GMT
master相关证书	pki/ca.key		k8s-master节点		
	pki/ca.crt	pki/ca.key	k8s-master节点	10年	Not Before: Jul 29 06:25:27 2019 GMT Not After : Jul 26 06:25:27 2029 GMT
	pki/apiserver.key		k8s-master节点		
	pki/apiserver.crt	pki/ca.key	k8s-master节点	1年	Not Before: Jul 29 06:25:27 2019 GMT Not After : Jul 28 06:25:27 2020 GMT
	pki/apiserver-kubelet-client.key		k8s-master节点		
	pki/apiserver-kubelet-client.crt	pki/ca.key	k8s-master节点	1年	Not Before: Jul 29 06:25:27 2019 GMT Not After : Jul 28 06:25:28 2020 GMT
	pki/front-proxy-ca.key		k8s-master节点		
	pki/front-proxy-ca.crt	pki/front-proxy-ca.key	k8s-master节点	1年	Not Before: Jul 29 06:25:35 2019 GMT Not After : Jul 26 06:25:35 2029 GMT
	pki/front-proxy-client.key		k8s-master节点		
	pki/front-proxy-client.crt	pki/ca.key	k8s-master节点	1年	Not Before: Jul 29 06:25:35 2019 GMT Not After : Jul 28 06:25:36 2020 GMT
	pki/sa.key		k8s-master节点		
	pki/sa.pub		k8s-master节点		

4.9.2 查看证书有效期

kubeadm alpha certs check-expiration

```
[root@master1 quzl]# kubeadm alpha certs check-expiration
CERTIFICATE                EXPIRES                  RESIDUAL TIME   EXTERNALLY MANAGED
admin.conf                 Feb 25, 2021 01:16 UTC   364d            no
apiserver                  Feb 25, 2021 01:16 UTC   364d            no
apiserver-etcd-client      Feb 25, 2021 01:16 UTC   364d            no
apiserver-kubelet-client   Feb 25, 2021 01:16 UTC   364d            no
controller-manager.conf    Feb 25, 2021 01:16 UTC   364d            no
etcd-healthcheck-client    Feb 25, 2021 01:16 UTC   364d            no
etcd-peer                  Feb 25, 2021 01:16 UTC   364d            no
etcd-server                Feb 25, 2021 01:16 UTC   364d            no
front-proxy-client         Feb 25, 2021 01:16 UTC   364d            no
scheduler.conf             Feb 25, 2021 01:16 UTC   364d            no
[root@master1 quzl]#
```

4.9.3 证书续期

kubeadm alpha certs renew all　更新所有证书有效期，延长到一年，无论当前还剩余多久，一共更新 7 个 crt 证书+三个配置文件

```
[root@master kubernetes]# kubeadm alpha certs renew all
certificate embedded in the kubeconfig file for the admin to use and for kubeadm itself renewed
certificate for serving the Kubernetes API renewed
certificate the apiserver uses to access etcd renewed
certificate for the API server to connect to kubelet renewed
certificate embedded in the kubeconfig file for the controller manager to use renewed
certificate for liveness probes to healtcheck etcd renewed
```

```
certificate for etcd nodes to communicate with each other renewed
certificate for serving etcd renewed
certificate for the front proxy client renewed
certificate embedded in the kubeconfig file for the scheduler manager to use renewed
```

也就是官网提供的这 10 项

kubeadm alpha certs renew

You can renew all Kubernetes certificates using the `all` subcommand or renew them selectively. For more details about certificate expiration and renewal see the certificate management documentation.

renew | all | admin.conf | apiserver-etcd-client | apiserver-kubelet-client
apiserver | controller-manager.conf | etcd-healthcheck-client | etcd-peer
etcd-server | front-proxy-client | scheduler.conf

https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm-alpha/

4.9.4 证书保存为 secret

使用如下命令，可以将证书保存到 secret，名字是 kubeadm-certs

```
[root@master1 quzl]# kubeadm init phase upload-certs --upload-certs

W0226 09:43:36.234529 2392542 version.go:98] could not fetch a Kubernetes version from the internet: unable to get URL "https://dl.k8s.io/release/stable-1.txt": Get https://storage.googleapis.com/kubernetes-release/release/stable-1.txt: net/http: request canceled while waiting for connection (Client.Timeout exceeded while awaiting headers)
W0226 09:43:36.235070 2392542 version.go:99] falling back to the local client version: v1.15.5
[upload-certs] Storing the certificates in Secret "kubeadm-certs" in the "kube-system" Namespace
[upload-certs] Using certificate key:
03a3d670edd627e6d9430e2d3c42d26585fcda3569ddad824ea37cfff30f6198
```

```
[root@master1 ~]# kubectl get secret kubeadm-certs -n kube-system
NAME            TYPE      DATA   AGE
kubeadm-certs   Opaque    8      84m
```

改 secret 中一共保存了 8 个证书文件，详情可以进去查看。

```
[root@master1 ~]# kubectl describe  secret kubeadm-certs -n kube-system
Name:         kubeadm-certs
Namespace:    kube-system
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
sa.pub:              479 bytes
ca.crt:              1053 bytes
ca.key:              1707 bytes
etcd-ca.crt:         1045 bytes
etcd-ca.key:         1703 bytes
front-proxy-ca.crt:  1066 bytes
front-proxy-ca.key:  1707 bytes
sa.key:              1703 bytes
[root@master1 ~]#
```

保存为 secret 后，可以在创建高可用集群的时候不用单独同步证书。

第五章 常用命令

5.1 get

集群启动后，在 master 节点上观察集群运行状态是否和规划相符

5.1.1 配置环境变量

输入以下语句

```
echo "export KUBECONFIG=/etc/kubernetes/admin.conf" >> ~/.bash_profile
source ~/.bash_profile
```

若不进行这一步，执行任何 kubectl 命令都将出现以下错误

```
[root@docker07 ~]#
[root@docker07 ~]# kubectl get nodes
The connection to the server localhost:8080 was refused - did you specify the right host or port?
[root@docker07 ~]#
[root@docker07 ~]#
```

5.1.2 get cs

查看组件状态

```
kubectl get componentstatus
kubectl get cs
```

```
[root@docker07 ~]# kubectl get componentstatus
NAME                 STATUS    MESSAGE             ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-0               Healthy   {"health": "true"}
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# kubectl get cs
NAME                 STATUS    MESSAGE             ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-0               Healthy   {"health": "true"}
[root@docker07 ~]#
[root@docker07 ~]#
```

5.1.3 get node

查看节点

```
[root@docker07 ~]# kubectl get nodes
NAME        STATUS     ROLES    AGE     VERSION
docker07    NotReady   master   2d23h   v1.13.3
docker08    NotReady   <none>   2d22h   v1.13.3
docker09    NotReady   <none>   80m     v1.13.3
docker10    NotReady   <none>   79m     v1.13.3
[root@docker07 ~]#
```

看到状态都是 NotReady 状态，因为未执行 4.4 步骤，执行后查看信息如下:

```
[root@docker07 ~]# kubectl get nodes
NAME        STATUS   ROLES    AGE     VERSION
docker07    Ready    master   3d1h    v1.13.3
docker08    Ready    <none>   3d1h    v1.13.3
docker09    Ready    <none>   3h23m   v1.13.3
docker10    Ready    <none>   3h22m   v1.13.3
[root@docker07 ~]#
```

显示标签

kubectl get nodes --show-labels

```
[root@node3 ~]#
[root@node3 ~]# kubectl get nodes --show-labels
NAME       STATUS   ROLES    AGE    VERSION   LABELS
node1      Ready    <none>   129d   v1.12.2   beta.kubernetes.
node100    Ready    <none>   101d   v1.12.2   beta.kubernetes.
node2      Ready    <none>   129d   v1.12.2   beta.kubernetes.
node3      Ready    master   129d   v1.12.2   beta.kubernetes.
node4      Ready    <none>   129d   v1.12.2   beta.kubernetes.
[root@node3 ~]#
[root@node3 ~]#
[root@node3 ~]#
```

查看各个节点的 pod 网络

kubectl get node "-o=custom-columns=NAME:.metadata.name,podCIDR:spec.podCIDR"

```
[root@master1120 xmc-model-serving]# kubectl get node "-o=custom-columns=NAME:.metadata.name,podCIDR:spec.podCIDR"
NAME        podCIDR
gpu0312     10.244.11.0/24
gpu6802     10.244.4.0/24
master1120  10.244.3.0/24
master1220  10.244.1.0/24
master1320  10.244.2.0/24
node1121    10.244.20.0/24
node1122    10.244.5.0/24
node1123    10.244.16.0/24
node1221    10.244.15.0/24
node1222    10.244.17.0/24
node1223    10.244.13.0/24
node1321    10.244.19.0/24
node1322    10.244.6.0/24
node1323    10.244.18.0/24
```

5.1.4 get ns

查看名称空间

kubectl get namespace

kubectl get ns

```
[root@docker07 ~]#
[root@docker07 ~]# kubectl get namespace
NAME          STATUS   AGE
default       Active   4d21h
kube-public   Active   4d21h
kube-system   Active   4d21h
[root@docker07 ~]#
[root@docker07 ~]# kubectl get ns
NAME          STATUS   AGE
default       Active   4d21h
kube-public   Active   4d21h
kube-system   Active   4d21h
[root@docker07 ~]#
```

可以看到一共有三个名称空间，

default：默认的

kube-public：公共的

kube-system：系统级别的

5.1.5 get deploy

查看资源

kubectl get deployments

kubectl get deploy

```
[root@docker07 ~]# kubectl get deployments
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
myapp    1/1     1            1           2m52s
nginx    3/3     3            3           24h
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
myapp-5d4d8c8458-wkskj    1/1     Running   0          2m56s
nginx-7cdbd8cdc9-6bxcj    1/1     Running   0          24h
nginx-7cdbd8cdc9-csb95    1/1     Running   0          24h
nginx-7cdbd8cdc9-qc5vh    1/1     Running   0          24h
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# kubectl get deployments -n kube-system
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
coredns   2/2     2            2           4d22h
[root@docker07 ~]#
```

5.1.6 get svc

查看 service

```
kubectl get service
kubectl get svc
```

```
[root@docker07 ~]# kubectl get services
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP        5d2h
myapp        NodePort    10.107.127.2   <none>        80:31274/TCP   4h
nginx        NodePort    10.103.94.20   <none>        80:30435/TCP   28h
[root@docker07 ~]#
[root@docker07 ~]# kubectl get services -n kube-system
NAME       TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)         AGE
kube-dns   ClusterIP   10.96.0.10    <none>        53/UDP,53/TCP   5d2h
[root@docker07 ~]#
```

从图中可以看到 cluster-ip，此 ip 即是 service 网络的 ip，在集群初始化时使用如下参数定义的网络

--service-cidr=10.96.0.0/12

图中 10.96、10.103、10.107 等都属于 10.96.0.0/12 网络。

需要说明的 kube-dns 的 IP：10.96.0.10 将作为所有 pod 中的默认 nameserver，以此来解析其他服务，服务之间的调用使用服务名。

使用如下命令

```
kubectl describe svc myapp
```

可以查看 service 的详情，包括很多信息，看下图

```
[root@docker07 ~]# kubectl describe svc myapp
Name:                     myapp
Namespace:                default
Labels:                   run=myapp
Annotations:              <none>
Selector:                 run=myapp
Type:                     ClusterIP
IP:                       10.96.245.240
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
Endpoints:                10.244.2.11:80,10.244.3.11:80,10.244.4.14:80
Session Affinity:         None
Events:                   <none>
[root@docker07 ~]#
[root@docker07 ~]# kubectl describe svc nginx
Name:                     nginx
Namespace:                default
Labels:                   run=nginx
Annotations:              <none>
Selector:                 run=nginx
Type:                     NodePort
IP:                       10.103.94.20
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30435/TCP
Endpoints:                10.244.2.2:80,10.244.3.2:80,10.244.3.6:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
[root@docker07 ~]#
```

5.1.7 get endpoints

查看 endpoints

k8s 创建 service 的同时，会自动创建跟 service 同名的 endpoints。

使用如下语句可查看详细信息，包括 endpoint 后端对应的 pod 的 IP 地址

```
kubectl get endpoints kube-dns -o yaml
```

```
[root@docker07 ~]# kubectl get endpoints kube-dns -o yaml -n kube-system
apiVersion: v1
kind: Endpoints
metadata:
  creationTimestamp: "2019-02-15T08:33:12Z"
  labels:
    k8s-app: kube-dns
    kubernetes.io/cluster-service: "true"
    kubernetes.io/name: KubeDNS
  name: kube-dns
  namespace: kube-system
  resourceVersion: "347640"
  selfLink: /api/v1/namespaces/kube-system/endpoints/kube-dns
  uid: 551e3a64-30fc-11e9-a2d0-000c29b30ea9
subsets:
- addresses:
  - ip: 10.244.0.2
    nodeName: docker07
    targetRef:
      kind: Pod
      name: coredns-86c58d9df4-2nwl7
      namespace: kube-system
```

使用 kubectl get pods 也可以看到对应的 pod 的 IP

5.1.8 get pod

查看 pods

```
kubectl get pods
```

默认查看 default 的 pod

```
[root@docker07 ~]#
[root@docker07 ~]# kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
nginx-7cdbd8cdc9-6bxcj        1/1     Running   0          24h
nginx-7cdbd8cdc9-csb95        1/1     Running   0          24h
nginx-7cdbd8cdc9-qc5vh        1/1     Running   0          24h
[root@docker07 ~]#
[root@docker07 ~]#
```

kubectl get pods -n kube-public -o wide

使用-n 查看指定名称空间的 pod

```
[root@docker07 ~]#
[root@docker07 ~]# kubectl get pods -n kube-system -o wide
NAME                                   READY   STATUS    RESTARTS   AGE     IP              NODE       NOMINATED NODE   READINESS GATES
coredns-86c58d9df4-2nwl7               1/1     Running   0          4d21h   10.244.0.2      docker07   <none>           <none>
coredns-86c58d9df4-5969g               1/1     Running   0          4d21h   10.244.0.3      docker07   <none>           <none>
etcd-docker07                          1/1     Running   0          4d21h   172.16.6.37     docker07   <none>           <none>
kube-apiserver-docker07                1/1     Running   0          4d21h   172.16.6.37     docker07   <none>           <none>
kube-controller-manager-docker07       1/1     Running   0          4d21h   172.16.6.37     docker07   <none>           <none>
kube-flannel-ds-amd64-2ffm8            1/1     Running   0          46h     172.16.6.40     docker10   <none>           <none>
kube-flannel-ds-amd64-fwwrr            1/1     Running   0          46h     172.16.6.38     docker08   <none>           <none>
kube-flannel-ds-amd64-kbc55            1/1     Running   0          46h     172.16.6.37     docker07   <none>           <none>
kube-flannel-ds-amd64-qp6b5            1/1     Running   0          46h     172.16.6.39     docker09   <none>           <none>
kube-proxy-dfr7w                       1/1     Running   0          2d      172.16.6.40     docker10   <none>           <none>
kube-proxy-dgtsm                       1/1     Running   0          4d21h   172.16.6.37     docker07   <none>           <none>
kube-proxy-l9b7l                       1/1     Running   0          2d      172.16.6.39     docker09   <none>           <none>
kube-proxy-zz2z6                       1/1     Running   0          4d21h   172.16.6.38     docker08   <none>           <none>
kube-scheduler-docker07                1/1     Running   0          4d21h   172.16.6.37     docker07   <none>           <none>
[root@docker07 ~]#
```

从这里可以看到整个集群的架构，这种部署方式就是将系统组件也作为 pod 运行。

5.1.9 get label

使用--show-labels 可以将标签也一并显示出来，如下图的 run=myapp 就是标签

```
[root@docker07 ~]# kubectl get pods --show-labels
NAME                      READY   STATUS    RESTARTS   AGE   LABELS
busybox                   1/1     Running   15         15h   <none>
myapp-5d4d8c8458-5w4jc    1/1     Running   0          16m   pod-template-hash=5d4d8c8458,run=myapp
myapp-5d4d8c8458-kz5vw    1/1     Running   0          16m   pod-template-hash=5d4d8c8458,run=myapp
myapp-5d4d8c8458-qj72g    1/1     Running   0          16m   pod-template-hash=5d4d8c8458,run=myapp
nginx-7cdbd8cdc9-6bxcj    1/1     Running   0          44h   pod-template-hash=7cdbd8cdc9,run=nginx
nginx-7cdbd8cdc9-csb95    1/1     Running   0          44h   pod-template-hash=7cdbd8cdc9,run=nginx
nginx-7cdbd8cdc9-hhrl6    1/1     Running   0          18h   pod-template-hash=7cdbd8cdc9,run=nginx
[root@docker07 ~]#
[root@docker07 ~]#
```

-L app　显示 app 的标签值

-l app,abc

-l app=myapp

-l app=myapp，abc=bcd

-l app!=xmc-backend-service

-l “app in (xmc-backend-service,bcd)”

使用-l 筛选

```
[root@master xmc]#
[root@master xmc]# kubectl get pod -n xmc
NAME                                      READY   STATUS    RESTARTS   AGE
dm-model-serving-student-6775fcc894-sxfkb   0/1     Pending   0          4h38m
redis-8474575d46-hb9wr                    1/1     Running   0          4d3h
xmc-backend-service-5bb6dd5f7f-mzsk5      1/1     Running   0          4d4h
xmc-data-collector-76788d79c7-ln5tl       1/1     Running   0          4d4h
xmc-data-stream-6576659869-jbnsh          1/1     Running   0          22m
xmc-frontend-7f4bf7d87-f2xr9              1/1     Running   0          4d3h
xmc-metric-generator-5768d98655-cp87z     1/1     Running   0          4h44m
xmc-storage-service-74fdff799f-nmtjk      1/1     Running   0          4d4h
[root@master xmc]#
[root@master xmc]# kubectl get pod -n xmc -L app
NAME                                      READY   STATUS    RESTARTS   AGE     APP
dm-model-serving-student-6775fcc894-sxfkb   0/1     Pending   0          4h38m   dm-model-serving-student
redis-8474575d46-hb9wr                    1/1     Running   0          4d3h    redis
xmc-backend-service-5bb6dd5f7f-mzsk5      1/1     Running   0          4d4h    xmc-backend-service
xmc-data-collector-76788d79c7-ln5tl       1/1     Running   0          4d4h    xmc-data-collector
xmc-data-stream-6576659869-jbnsh          1/1     Running   0          22m     xmc-data-stream
xmc-frontend-7f4bf7d87-f2xr9              1/1     Running   0          4d3h    xmc-frontend
xmc-metric-generator-5768d98655-cp87z     1/1     Running   0          4h44m   xmc-metric-generator
xmc-storage-service-74fdff799f-nmtjk      1/1     Running   0          4d4h    xmc-storage-service
[root@master xmc]#
[root@master xmc]#
[root@master xmc]#
[root@master xmc]# kubectl get pod -n xmc -l app=xmc-backend-service
NAME                                      READY   STATUS    RESTARTS   AGE
xmc-backend-service-5bb6dd5f7f-mzsk5      1/1     Running   0          4d4h
[root@master xmc]#
[root@master xmc]# kubectl get pod -n xmc -l app=xmc-backend-service --show-labels
NAME                                      READY   STATUS    RESTARTS   AGE     LABELS
xmc-backend-service-5bb6dd5f7f-mzsk5      1/1     Running   0          4d4h    app=xmc-backend-service,pod-template-hash=5bb6dd5f7f
[root@master xmc]#
```

```
[root@master xmc]#
[root@master xmc]# kubectl get pod -n xmc -l "app in (redis,bdd)"
NAME                      READY   STATUS    RESTARTS   AGE
redis-8474575d46-hb9wr    1/1     Running   0          4d3h
[root@master xmc]#
[root@master xmc]#
[root@master xmc]# kubectl get pod -n xmc -l "app notin (redis,bdd)"
NAME                                      READY   STATUS    RESTARTS   AGE
dm-model-serving-student-6775fcc894-sxfkb   0/1     Pending   0          4h48m
xmc-backend-service-5bb6dd5f7f-mzsk5        1/1     Running   0          4d4h
xmc-data-collector-76788d79c7-ln5tl         1/1     Running   0          4d4h
xmc-data-stream-6576659869-jbnsh            1/1     Running   0          32m
xmc-frontend-7f4bf7d87-f2xr9                1/1     Running   0          4d3h
xmc-metric-generator-5768d98655-cp87z       1/1     Running   0          4h54m
xmc-storage-service-74fdff799f-nmtjk        1/1     Running   0          4d4h
[root@master xmc]#
[root@master xmc]#
```

5.1.10 查看资源限制

查看资源限制:

从 deployment 查看:

```
kubectl get deploy --all-namespaces "-o=custom-columns=\
namespaces:.metadata.namespace,\
NAME:.metadata.name,\
mem_request:.spec.template.spec.containers[0].resources.requests.memory,\
mem_limit:.spec.template.spec.containers[0].resources.limits.memory,\
cpu_request:.spec.template.spec.containers[0].resources.requests.cpu,\
cpu_limit:.spec.template.spec.containers[0].resources.limits.cpu"
```

从 pod 查看:

```
kubectl get pod --all-namespaces "-o=custom-columns=\
namespaces:.metadata.namespace,\
NAME:.metadata.name,\
mem_request:.spec.containers[0].resources.requests.memory,\
mem_limit:.spec.containers[0].resources.limits.memory,\
cpu_request:.spec.containers[0].resources.requests.cpu,\
cpu_limit:.spec.containers[0].resources.limits.cpu"
```

5.2 describe

查看相信信息

5.2.1 describe deploy

```
kubectl describe deployments couchbase-server
```

5.2.2 describe node

```
kubectl describe node node2
```

5.2.3 describe ns

```
kubectl describe ns x2
```

5.3 logs

```
kubectl logs myapp -n quzl
```

如果一个 pod 中有多个容器，需要指明需要查看哪个容器，如下查看 busybox 容器日志

```
kubectl logs myapp –c myapp -n quzl
```

```
[root@master k8s-test]#
[root@master k8s-test]# kubectl logs myapp -c busybox -n quzl
[root@master k8s-test]#
[root@master k8s-test]# kubectl logs myapp  -n quzl
error from server (BadRequest): a container name must be specified for pod myapp, choose one of: [myapp busybox]
[root@master k8s-test]#
[root@master k8s-test]#
[root@master k8s-test]# kubectl logs myapp -c myapp -n quzl
10.244.0.0 - - [31/Mar/2019:20:58:43 +0000] "GET / HTTP/1.1" 200 87 "-" "curl/7.29.0" "-"
10.244.0.0 - - [31/Mar/2019:20:58:50 +0000] "GET / HTTP/1.1" 200 116 "-" "curl/7.29.0" "-"
10.244.0.0 - - [31/Mar/2019:20:58:52 +0000] "GET / HTTP/1.1" 200 116 "-" "curl/7.29.0" "-"
10.244.0.0 - - [31/Mar/2019:20:58:53 +0000] "GET / HTTP/1.1" 200 116 "-" "curl/7.29.0" "-"
10.244.0.0 - - [31/Mar/2019:20:58:54 +0000] "GET / HTTP/1.1" 200 116 "-" "curl/7.29.0" "-"
[root@master k8s-test]#
```

5.4 rollout

回滚

查看版本历史，序号越大，越新

```
kubectl rollout history deployment xmc-backend-service -n xmc
```

回滚，默认回滚到上一个版本

```
kubectl rollout undo deployment xmc-backend-service -n xmc
```

指定版本回滚

```
kubectl rollout undo deployment xmc-backend-service --to-revision=1 -n xmc
```

重启:

kubectl rollout restart deployment --namespace dmos

5.5 alpha

更新所有证书有效期，延长到一年，无论当前还剩余多久，一共更新 7 个 crt 证书+三个配置文件

```
[root@master kubernetes]# kubeadm alpha certs renew all
certificate embedded in the kubeconfig file for the admin to use and for kubeadm itself renewed
certificate for serving the Kubernetes API renewed
certificate the apiserver uses to access etcd renewed
```

```
certificate for the API server to connect to kubelet renewed
certificate embedded in the kubeconfig file for the controller manager to use renewed
certificate for liveness probes to healtcheck etcd renewed
certificate for etcd nodes to communicate with each other renewed
certificate for serving etcd renewed
certificate for the front proxy client renewed
certificate embedded in the kubeconfig file for the scheduler manager to use renewed
```

查看证书有效期，1.15 之后提供

```
[root@master kubernetes]# kubeadm alpha certs check-expiration
CERTIFICATE                EXPIRES                  RESIDUAL TIME    EXTERNALLY MANAGED
admin.conf                 Nov 09, 2020 14:10 UTC    364d            no
apiserver                  Nov 09, 2020 14:10 UTC    364d            no
apiserver-etcd-client      Nov 09, 2020 14:10 UTC    364d            no
apiserver-kubelet-client   Nov 09, 2020 14:10 UTC    364d            no
controller-manager.conf    Nov 09, 2020 14:10 UTC    364d            no
etcd-healthcheck-client    Nov 09, 2020 14:10 UTC    364d            no
etcd-peer                  Nov 09, 2020 14:10 UTC    364d            no
etcd-server                Nov 09, 2020 14:10 UTC    364d            no
front-proxy-client         Nov 09, 2020 14:10 UTC    364d            no
scheduler.conf             Nov 09, 2020 14:10 UTC    364d            no
```

第六章 测试

6.1 创建集群实例-nginx

6.1.1 创建 nginx 的 pod

```
kubectl get pod
kubectl run nginx --image=nginx --replicas=3
kubectl get pod
```

```
[root@docker07 ~]# kubectl get pod
No resources found.
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# kubectl run nginx --image=nginx --replicas=3
kubectl run --generator=deployment/apps.v1 is DEPRECATED and will be removed in a future v
deployment.apps/nginx created
[root@docker07 ~]#
[root@docker07 ~]# kubectl get pod
NAME                     READY   STATUS              RESTARTS   AGE
nginx-7cdbd8cdc9-6bxcj   0/1     ContainerCreating   0          5s
nginx-7cdbd8cdc9-csb95   0/1     ContainerCreating   0          5s
nginx-7cdbd8cdc9-qc5vh   0/1     ContainerCreating   0          5s
[root@docker07 ~]#
[root@docker07 ~]# kubectl get pod
NAME                     READY   STATUS              RESTARTS   AGE
nginx-7cdbd8cdc9-6bxcj   0/1     ContainerCreating   0          15s
nginx-7cdbd8cdc9-csb95   0/1     ContainerCreating   0          15s
nginx-7cdbd8cdc9-qc5vh   0/1     ContainerCreating   0          15s
[root@docker07 ~]# kubectl get pod
NAME                     READY   STATUS              RESTARTS   AGE
nginx-7cdbd8cdc9-6bxcj   0/1     ContainerCreating   0          103s
nginx-7cdbd8cdc9-csb95   0/1     ContainerCreating   0          103s
nginx-7cdbd8cdc9-qc5vh   0/1     ContainerCreating   0          103s
[root@docker07 ~]#
[root@docker07 ~]# kubectl get pod
NAME                     READY   STATUS    RESTARTS   AGE
nginx-7cdbd8cdc9-6bxcj   1/1     Running   0          3m40s
nginx-7cdbd8cdc9-csb95   1/1     Running   0          3m40s
nginx-7cdbd8cdc9-qc5vh   1/1     Running   0          3m40s
```

使用-o wide 参数可看到更加详细的信息

```
kubectl get pods -o wide
```

```
[root@docker07 ~]# kubectl get pods -o wide
NAME                     READY   STATUS    RESTARTS   AGE     IP           NODE       NOMINATED NODE   READINESS GATES
nginx-7cdbd8cdc9-6bxcj   1/1     Running   0          8m19s   10.244.3.2   docker10   <none>           <none>
nginx-7cdbd8cdc9-csb95   1/1     Running   0          8m19s   10.244.2.2   docker09   <none>           <none>
nginx-7cdbd8cdc9-qc5vh   1/1     Running   0          8m19s   10.244.1.2   docker08   <none>           <none>
[root@docker07 ~]#
```

从图中我们可以看到，每个 node 节点运行了一个 pod

6.1.2 创建 pod 的 service

pod 包含容器，着眼于多节点的服务，而服务访问的入口由 service 提供，service 提供类似于 lvs 类似的功能，做流量分发，使用如下命令创建 service

```
kubectl expose deployment nginx --port=80 --target-port=80 --type=NodePort
```

--type 的可选参数：

NodePort 节点 pod，节点会暴露端口到宿主机网卡，集群外部可以访问

ClusterIP 集群内部 pod

LoadBalancer

ExternalName

使用如下命令可查看端口监听情况：

```
kubectl get service
kubectl get svc
```

```
[root@docker07 ~]#
[root@docker07 ~]# kubectl expose deployment nginx --port=80 --target-port=80 --type=NodePort
service/nginx exposed
[root@docker07 ~]#
[root@docker07 ~]# kubectl get pods -o wide
NAME                     READY   STATUS    RESTARTS   AGE   IP           NODE       NOMINATED NODE   READINESS GATES
nginx-7cdbd8cdc9-6bxcj   1/1     Running   0          47m   10.244.3.2   docker10   <none>           <none>
nginx-7cdbd8cdc9-csb95   1/1     Running   0          47m   10.244.2.2   docker09   <none>           <none>
nginx-7cdbd8cdc9-qc5vh   1/1     Running   0          47m   10.244.1.2   docker08   <none>           <none>
[root@docker07 ~]#
[root@docker07 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP        3d22h
nginx        NodePort    10.103.94.20   <none>        80:30435/TCP   107s
```

其中 nginx 服务的类型（TYPE）是 NodePort，pod 所在宿主机将使用 30435 端口进行转发流量转发到 pod

6.1.3 访问 pod

使用以下命令

kubectl get service

kubectl get svc

可以查看对应的 cluster-ip 是：10.103.94.20，可通过此 IP 在安装了 flannel 的节点上进行访问：

```
[root@docker09 ~]# curl 10.103.94.20:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
```

我们看到还有一个 30435 端口， 该端口将监听在 node 节点上，如下图

```
[root@docker09 ~]#
[root@docker09 ~]# netstat -tnlp|grep 30435
tcp6       0      0 :::30435                :::*                    LISTEN      4151/kube-proxy
[root@docker09 ~]#
[root@docker09 ~]#
```

所以可以在集群外部通过该 node 节点的 30435 端口直接访问

定时重启 pod

crontab -l

* * * * export KUBECONFIG=/etc/kubernetes/admin.conf && kubectl delete pod $(kubectl get pod -n kube-system|grep tiller|awk '{print $1}') -n kube-system

6.2 创建另一个实例-myapp

6.2.1 创建 myapp

我这里使用 ikubernetes/myapp 镜像，可通过访问 http://**/hostname.html 返回主机名，通过 http://***返回版本，用于测试负载均衡和版本回退等各种功能。

```
kubectl run myapp --image=ikubernetes/myapp:v6 --replicas=3
kubectl expose deployment myapp --port=80 --target-port=80
kubectl get svc
```

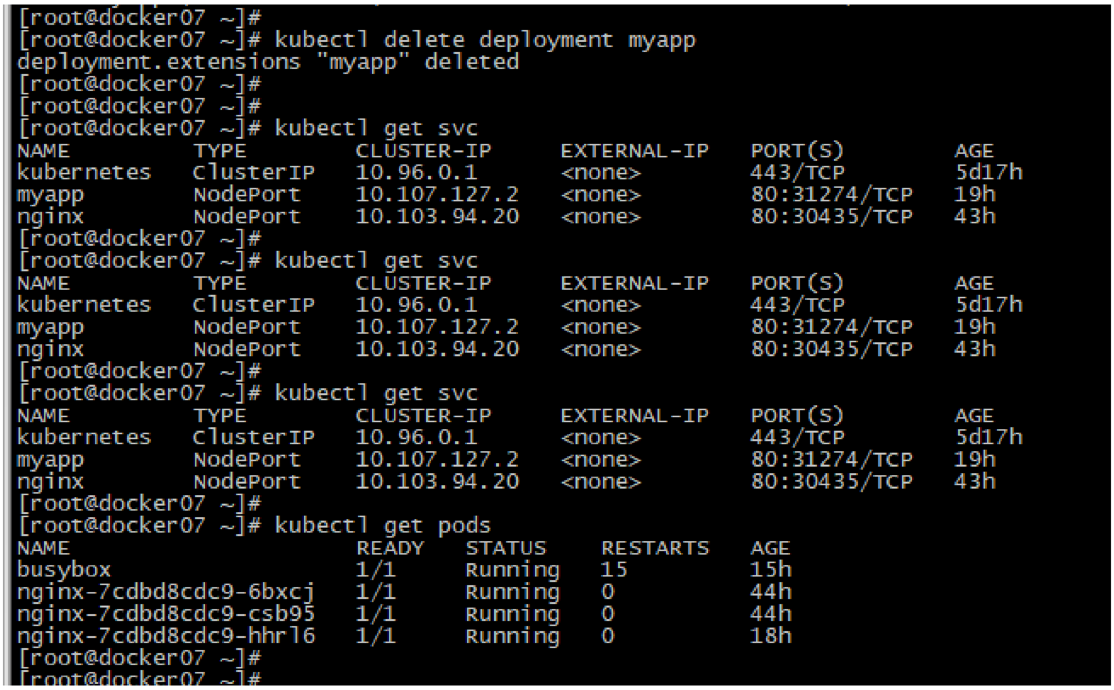
```
[root@docker07 ~]#
[root@docker07 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        5d18h
myapp        ClusterIP   10.96.245.240   <none>        80/TCP         2s
nginx        NodePort    10.103.94.20    <none>        80:30435/TCP   43h
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# kubectl get service
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        5d18h
myapp        ClusterIP   10.96.245.240   <none>        80/TCP         13s
nginx        NodePort    10.103.94.20    <none>        80:30435/TCP   43h
[root@docker07 ~]#
[root@docker07 ~]# kubectl get services
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        5d18h
myapp        ClusterIP   10.96.245.240   <none>        80/TCP         18s
nginx        NodePort    10.103.94.20    <none>        80:30435/TCP   43h
[root@docker07 ~]#
```

6.2.2 删除控制器

```
kubectl delete deployment myapp
```

```
[root@docker07 ~]# kubectl delete deployment myapp
deployment.extensions "myapp" deleted
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# kubectl get pods
NAME                     READY   STATUS        RESTARTS   AGE
myapp-5d4d8c8458-6rz5g   1/1     Terminating   0          12m
myapp-5d4d8c8458-c4lbz   1/1     Terminating   0          12m
myapp-5d4d8c8458-k78rl   0/1     Terminating   0          12m
nginx-7cdbd8cdc9-6bxcj   1/1     Running       0          24h
nginx-7cdbd8cdc9-csb95   1/1     Running       0          24h
nginx-7cdbd8cdc9-qc5vh   1/1     Running       0          24h
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
nginx-7cdbd8cdc9-6bxcj   1/1     Running   0          24h
nginx-7cdbd8cdc9-csb95   1/1     Running   0          24h
nginx-7cdbd8cdc9-qc5vh   1/1     Running   0          24h
[root@docker07 ~]#
```

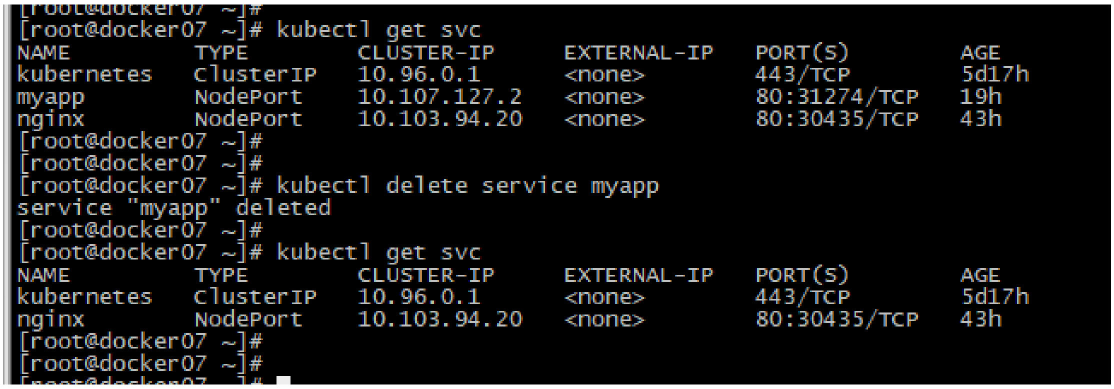
需要说明的是，及时删除了调度器，对应的 service 还是存在的



```
[root@docker07 ~]#
[root@docker07 ~]# kubectl delete deployment myapp
deployment.extensions "myapp" deleted
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        5d17h
myapp        NodePort    10.107.127.2    <none>        80:31274/TCP   19h
nginx        NodePort    10.103.94.20    <none>        80:30435/TCP   43h
[root@docker07 ~]#
[root@docker07 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        5d17h
myapp        NodePort    10.107.127.2    <none>        80:31274/TCP   19h
nginx        NodePort    10.103.94.20    <none>        80:30435/TCP   43h
[root@docker07 ~]#
[root@docker07 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        5d17h
myapp        NodePort    10.107.127.2    <none>        80:31274/TCP   19h
nginx        NodePort    10.103.94.20    <none>        80:30435/TCP   43h
[root@docker07 ~]#
[root@docker07 ~]# kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
busybox                  1/1     Running   15         15h
nginx-7cdbd8cdc9-6bxcj   1/1     Running   0          44h
nginx-7cdbd8cdc9-csb95   1/1     Running   0          44h
nginx-7cdbd8cdc9-hhrl6   1/1     Running   0          18h
[root@docker07 ~]#
[root@docker07 ~]#
```

6.2.3 删除 service

kubectl delete service myapp



```
[root@docker07 ~]#
[root@docker07 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        5d17h
myapp        NodePort    10.107.127.2    <none>        80:31274/TCP   19h
nginx        NodePort    10.103.94.20    <none>        80:30435/TCP   43h
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# kubectl delete service myapp
service "myapp" deleted
[root@docker07 ~]#
[root@docker07 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        5d17h
nginx        NodePort    10.103.94.20    <none>        80:30435/TCP   43h
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]#
```

6.2.4 删除 pod

kubectl delete pod myapp-5d4d8c8458-wkskj

```
[root@docker07 ~]#
[root@docker07 ~]# kubectl  get pods
NAME                      READY   STATUS    RESTARTS   AGE
myapp-5d4d8c8458-wkskj    1/1     Running   0          10m
nginx-7cdbd8cdc9-6bxcj    1/1     Running   0          24h
nginx-7cdbd8cdc9-csb95    1/1     Running   0          24h
nginx-7cdbd8cdc9-qc5vh    1/1     Running   0          24h
[root@docker07 ~]#
[root@docker07 ~]# kubectl delete myapp-5d4d8c8458-wkskj
error: resource(s) were provided, but no name, label selector, or --all flag specified
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# kubectl delete po myapp-5d4d8c8458-wkskj
pod "myapp-5d4d8c8458-wkskj" deleted
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# kubectl  get pods
NAME                      READY   STATUS              RESTARTS   AGE
myapp-5d4d8c8458-gn28c    0/1     ContainerCreating   0          11s
nginx-7cdbd8cdc9-6bxcj    1/1     Running             0          24h
nginx-7cdbd8cdc9-csb95    1/1     Running             0          24h
nginx-7cdbd8cdc9-qc5vh    1/1     Running             0          24h
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# kubectl  get pods
NAME                      READY   STATUS    RESTARTS   AGE
myapp-5d4d8c8458-gn28c    1/1     Running   0          29s
nginx-7cdbd8cdc9-6bxcj    1/1     Running   0          24h
nginx-7cdbd8cdc9-csb95    1/1     Running   0          24h
nginx-7cdbd8cdc9-qc5vh    1/1     Running   0          24h
[root@docker07 ~]#
```

从上图可以看出我们删除一个 pod，还会再启动一个 pod，这就是自愈。

强制删除 pod，加上参数：--grace-period=0 --force

```
kubectl delete pod myapp-5d4d8c8458-g9wkt --grace-period=0 --force
```

```
[root@docker07 ~]#
[root@docker07 ~]# kubectl delete pod myapp-5d4d8c8458-g9wkt --grace-period=0 --force
warning: Immediate deletion does not wait for confirmation that the running resource has been te
pod "myapp-5d4d8c8458-g9wkt" force deleted
[root@docker07 ~]#
[root@docker07 ~]#
```

6.2.5 删除 ns

如果 ns 一直处于 terminating 状态无法删除，使用如下命令找到是哪个资源没删除导致 ns 无法删除

```
kubectl api-resources --verbs=list --namespaced -o name | xargs -n 1 kubectl get --show-kind --ignore-not-found -n <namespace>
```

6.2.6 扩/缩容

```
kubectl scale --replicas=3 deployment/myapp
```

```
[root@docker07 ~]# kubectl get deployment
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
myapp    1/1     1            1           15m
nginx    3/3     3            3           24h
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# kubectl scale --replicas=3 deployment/myapp
deployment.extensions/myapp scaled
[root@docker07 ~]#
[root@docker07 ~]# kubectl get deployment
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
myapp    1/3     3            1           16m
nginx    3/3     3            3           24h
[root@docker07 ~]#
[root@docker07 ~]# kubectl get deployment
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
myapp    2/3     3            2           17m
nginx    3/3     3            3           24h
[root@docker07 ~]#
[root@docker07 ~]# kubectl get deployment
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
myapp    3/3     3            3           17m
nginx    3/3     3            3           24h
[root@docker07 ~]#
```

6.2.7 动态查看执行过程

使用如下命令可以动态查看过程，包括扩/缩容、版本更新、回退等

```
kubectl rollout status deployment myapp
```

```
[root@docker07 ~]#
[root@docker07 ~]# kubectl scale --replicas=10 deployment/myapp
deployment.extensions/myapp scaled
[root@docker07 ~]# kubectl rollout status deployment myapp
Waiting for deployment "myapp" rollout to finish: 5 of 10 updated replicas are available...
Waiting for deployment "myapp" rollout to finish: 6 of 10 updated replicas are available...
Waiting for deployment "myapp" rollout to finish: 7 of 10 updated replicas are available...
Waiting for deployment "myapp" rollout to finish: 8 of 10 updated replicas are available...
Waiting for deployment "myapp" rollout to finish: 9 of 10 updated replicas are available...
deployment "myapp" successfully rolled out
[root@docker07 ~]#
```

如图所示是进行节点扩容时的动态查看

6.2.8 滚动更新

Kubectl rollout undo deployment myapp　#回滚，默认回滚到上一个版本

使用如下命令进行滚动更新，更新版本：

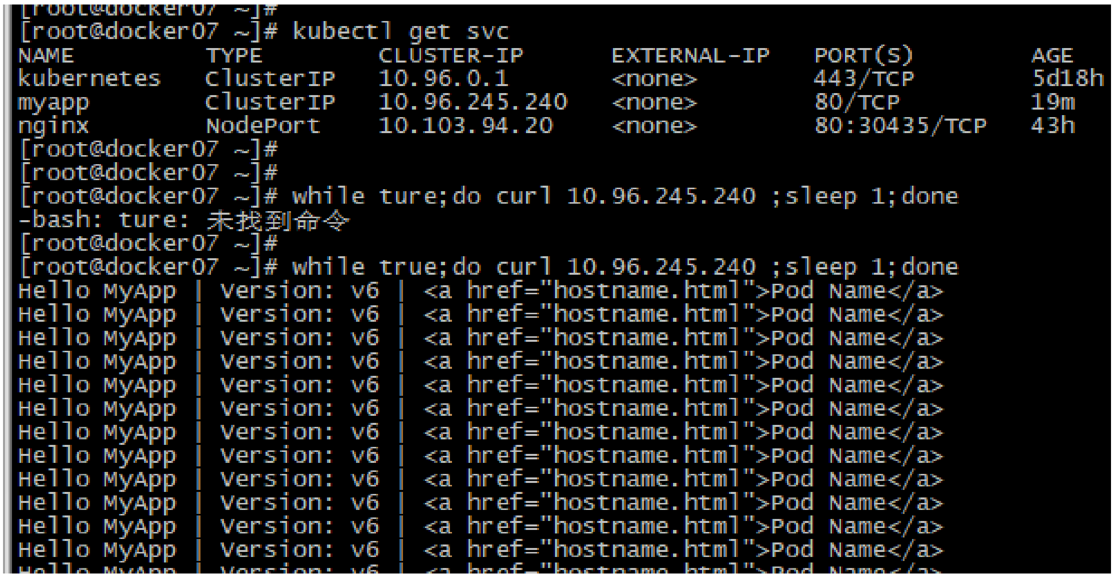
```
kubectl set image deployment/myapp myapp=ikubernetes/myapp:v5
```

```
[root@docker07 ~]#
[root@docker07 ~]# kubectl set image deployment/myapp myapp=ikubernetes/myapp:v5
deployment.extensions/myapp image updated
[root@docker07 ~]#
[root@docker07 ~]# kubectl rollout status deployment myapp
Waiting for deployment "myapp" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "myapp" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "myapp" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "myapp" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "myapp" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "myapp" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "myapp" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "myapp" rollout to finish: 1 old replicas are pending termination...
deployment "myapp" successfully rolled out
[root@docker07 ~]#
[root@docker07 ~]#
```

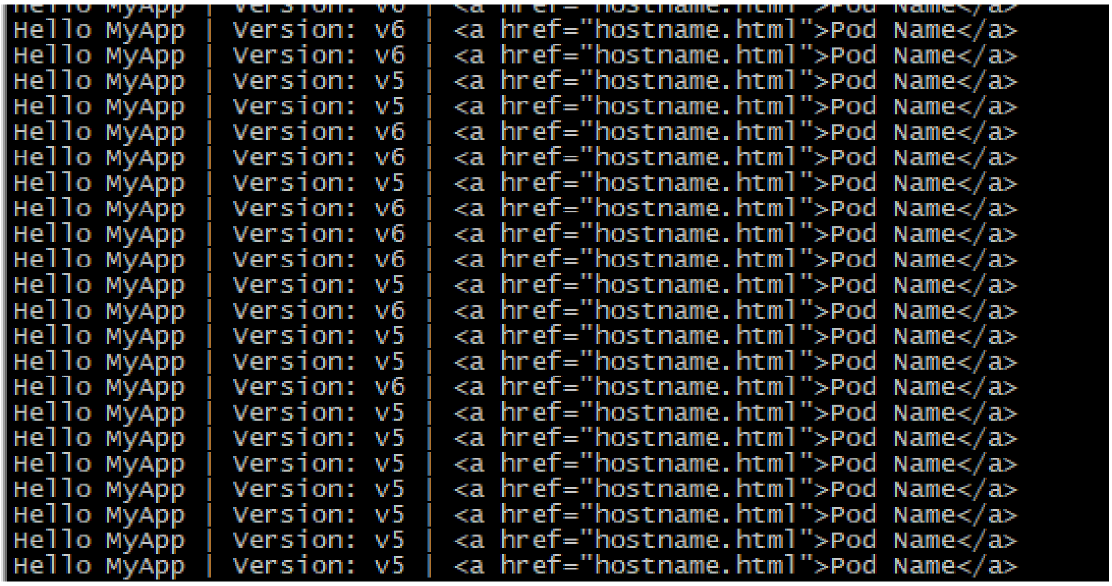
同时使用如下命令进行动态观察

```
kubectl rollout status deployment myapp
```

也可使用 curl 请求 service 的的方法进行观察：



```
[root@docker07 ~]#
[root@docker07 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        5d18h
myapp        ClusterIP   10.96.245.240   <none>        80/TCP         19m
nginx        NodePort    10.103.94.20    <none>        80:30435/TCP   43h
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# while ture;do curl 10.96.245.240 ;sleep 1;done
-bash: ture: 未找到命令
[root@docker07 ~]#
[root@docker07 ~]# while true;do curl 10.96.245.240 ;sleep 1;done
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
```

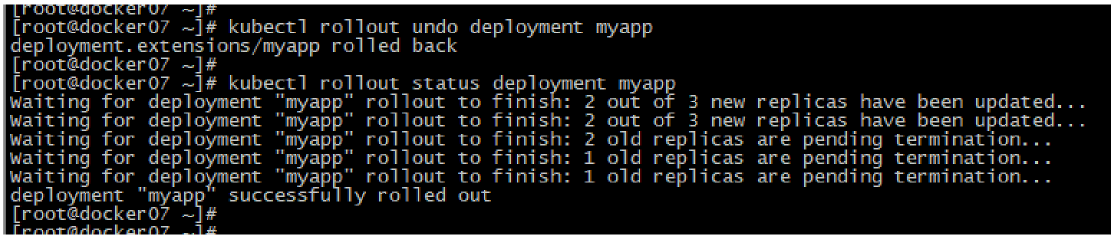


```
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
```

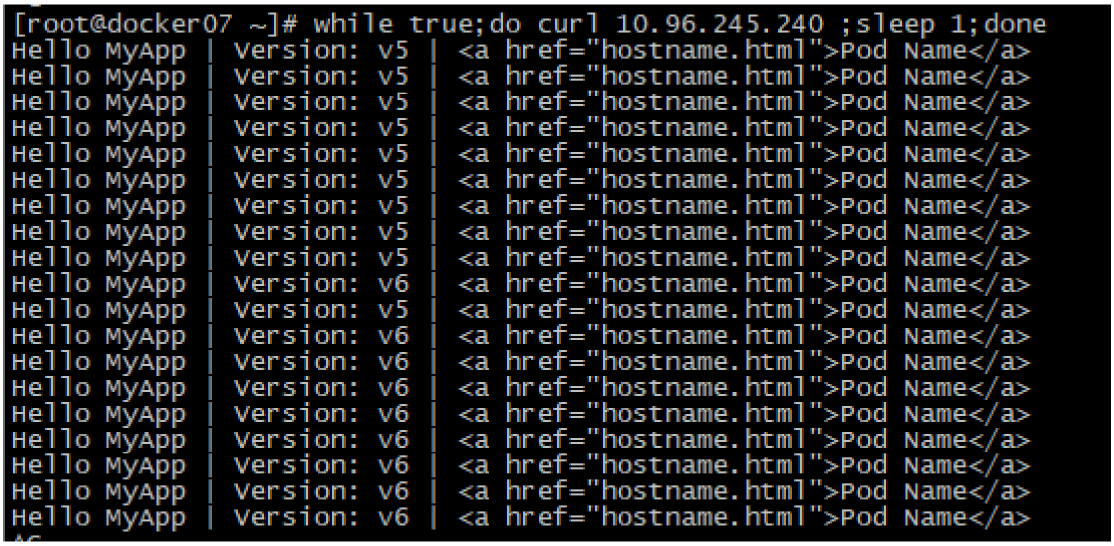
6.2.9　回退

使用如下命令进行版本回退，默认只回退到上一个版本：

```
kubectl rollout undo deployment myapp
```



```
[root@docker07 ~]#
[root@docker07 ~]# kubectl rollout undo deployment myapp
deployment.extensions/myapp rolled back
[root@docker07 ~]#
[root@docker07 ~]# kubectl rollout status deployment myapp
Waiting for deployment "myapp" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "myapp" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "myapp" rollout to finish: 2 old replicas are pending termination...
Waiting for deployment "myapp" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "myapp" rollout to finish: 1 old replicas are pending termination...
deployment "myapp" successfully rolled out
[root@docker07 ~]#
[root@docker07 ~]#
```



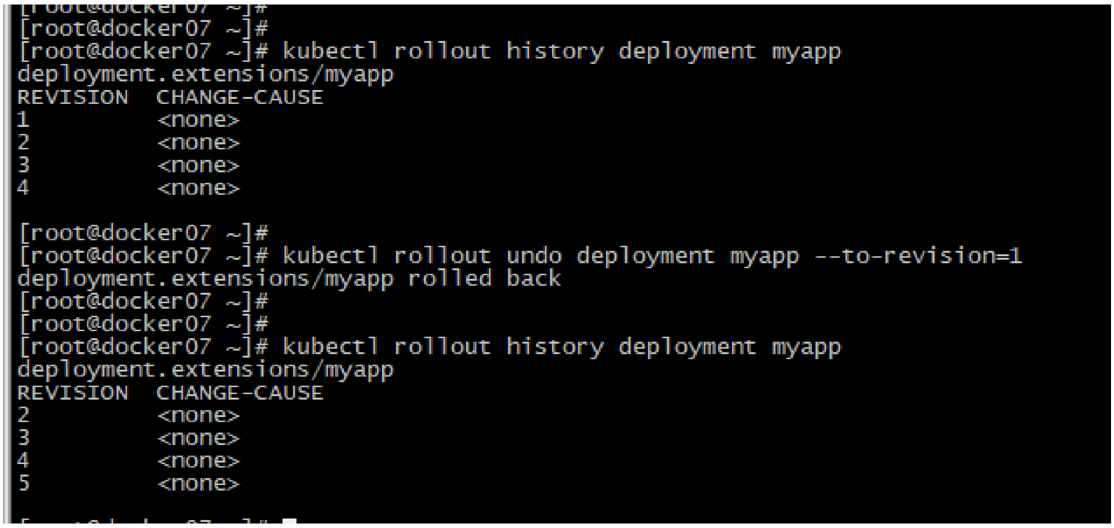
```
[root@docker07 ~]# while true;do curl 10.96.245.240 ;sleep 1;done
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v5 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
Hello MyApp | Version: v6 | <a href="hostname.html">Pod Name</a>
```

使用如下命令查看版本历史：

```
kubectl rollout history deployment myapp
```

使用如下命令指定版本回退：

```
kubectl rollout undo deployment myapp --to-revision=1
```



```
[root@docker07 ~]#
[root@docker07 ~]# kubectl rollout history deployment myapp
deployment.extensions/myapp
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
3         <none>
4         <none>

[root@docker07 ~]#
[root@docker07 ~]# kubectl rollout undo deployment myapp --to-revision=1
deployment.extensions/myapp rolled back
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# kubectl rollout history deployment myapp
deployment.extensions/myapp
REVISION  CHANGE-CAUSE
2         <none>
3         <none>
4         <none>
5         <none>
```

要理解下这里版本的关系，按照时间顺序排列，1 最早，比如我回退到 1 版本，那么 5 将取代 1，历史版本中不会保留 1，也就是说历史版本中不会有重复版本。

6.2.10 内部 dns 理解

验证一：

```
kubectl get svc -n kube-system
dig -t A myapp.default.svc.cluster.local +short @10.96.0.10
kubectl get svc
```

```
[root@docker07 ~]#
[root@docker07 ~]# kubectl get svc -n kube-system
NAME       TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)         AGE
kube-dns   ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP   5d18h
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# dig -t A myapp.default.svc.cluster.local +short @10.96.0.10
10.96.245.240
[root@docker07 ~]#
[root@docker07 ~]# dig -t A nginx.default.svc.cluster.local +short @10.96.0.10
10.103.94.20
[root@docker07 ~]#
[root@docker07 ~]#
[root@docker07 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        5d18h
myapp        ClusterIP   10.96.245.240   <none>        80/TCP         53m
nginx        NodePort    10.103.94.20    <none>        80:30435/TCP   44h
[root@docker07 ~]#
```

```
[root@docker07 ~]# kubectl get pod -n kube-system -o wide
NAME                                READY   STATUS    RESTARTS   AGE     IP
coredns-86c58d9df4-2nwl7            1/1     Running   0          5d18h   10.244.0.2
coredns-86c58d9df4-5969g            1/1     Running   0          5d18h   10.244.0.3
```

```
[root@docker07 ~]# dig -t A nginx.default.svc.cluster.local +short @10.244.0.2
10.103.94.20
[root@docker07 ~]# dig -t A nginx.default.svc.cluster.local +short @10.244.0.3
10.103.94.20
[root@docker07 ~]#
[root@docker07 ~]# dig -t A kubernetes.default.svc.cluster.local +short @10.244.0.3
10.96.0.1
[root@docker07 ~]#
```

如上图，内部 dns 也是通过一个 service 进行服务分发，我们使用 dig 进行测试，可通过服务名 myapp、nginx 等进行解析。

dns 的默认搜索域是.default.svc.cluster.local

验证二：

我这里新建一个 busybox 的 pod，进入 busybox，查看 dns 的配置，可以看到 dns 的配置以及默认的搜索域等

```
/ # hostname
busybox
/ #
/ # cat /etc/resolv.conf
nameserver 10.96.0.10
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
/ #
/ #
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
3: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450 qdisc noqueue
    link/ether 0a:58:0a:f4:04:0b brd ff:ff:ff:ff:ff:ff
    inet 10.244.4.11/24 scope global eth0
       valid_lft forever preferred_lft forever
/ #
```

使用如下命令测试，得到和测试一相同的结果：

```
nslookup -type=A kubernetes.default.svc.cluster.local 10.96.0.10
```

```
/ # nslookup -type=A myapp.default.svc.cluster.local 10.96.0.10
Server:         10.96.0.10
Address:        10.96.0.10:53

Name:   myapp.default.svc.cluster.local
Address: 10.96.245.240

/ #
/ # nslookup -type=A nginx.default.svc.cluster.local 10.96.0.10
Server:         10.96.0.10
Address:        10.96.0.10:53

Name:   nginx.default.svc.cluster.local
Address: 10.103.94.20

/ #
/ # nslookup -type=A kubernetes.default.svc.cluster.local 10.96.0.10
Server:         10.96.0.10
Address:        10.96.0.10:53

Name:   kubernetes.default.svc.cluster.local
Address: 10.96.0.1
```

6.2.11 flannel 网络理解

flannel 作为一个 pod 运行，管理对应节点的 iptables 规则或者 ipvs 规则，使用如下命令

Iptable –vnL –t nat

查看当前的规则

每一个 node 节点中的 pod 属于同一 10.244.0.0/16 网络的子网，比如我这台是 10.244.4.0/24，所以整个集群内部所有 pod 的 ip 不会重复。

```
Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
 1199 94359 KUBE-POSTROUTING  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes postrouting rules *
    0     0 MASQUERADE  all  --  *      !docker0  172.17.0.0/16        0.0.0.0/0
  171 10588 RETURN     all  --  *      *       10.244.0.0/16        10.244.0.0/16
    3   252 MASQUERADE  all  --  *      *       10.244.0.0/16        !224.0.0.0/4
    0     0 RETURN     all  --  *      *      !10.244.0.0/16        10.244.4.0/24
    0     0 MASQUERADE  all  --  *      *      !10.244.0.0/16        10.244.0.0/16
```

如下图：为集群中每一个 pod（不管是否在此 node 上）创建一个 DNAT 规则，保证集群中任何一个 node 可以和集群中任何一个 pod 通信。

```
Chain KUBE-SEP-463KM6RTTDV6MG7S (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       10.244.4.23          0.0.0.0/0
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            tcp to:10.244.4.23:80

Chain KUBE-SEP-4MQWJMQB3MP6HHHN (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       10.244.2.2           0.0.0.0/0
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            tcp to:10.244.2.2:80

Chain KUBE-SEP-6E7XQMQ4RAYOWTTM (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       10.244.0.3           0.0.0.0/0
    0     0 DNAT       udp  --  *      *       0.0.0.0/0            0.0.0.0/0            udp to:10.244.0.3:53

Chain KUBE-SEP-6YBLMPJKOUHDFBST (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       10.244.3.2           0.0.0.0/0
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            tcp to:10.244.3.2:80

Chain KUBE-SEP-7QJK43IC5KUCNJMY (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       10.244.3.18          0.0.0.0/0
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            tcp to:10.244.3.18:80
```

为集群内的公共服务，比如 dns 的 53 端口、master 的 6443 端口也是作为 pod 运行的，所以也创建有对应的 DNAT 规则

```
Chain KUBE-SEP-HJ5LOISWSGGQY4OL (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       172.16.6.37          0.0.0.0/0
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            tcp to:172.16.6.37:6443

Chain KUBE-SEP-IT2ZTR26TO4XFPTO (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       10.244.0.2           0.0.0.0/0
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            tcp to:10.244.0.2:53

Chain KUBE-SEP-U6YIJEITHRETXKES (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       10.244.3.6           0.0.0.0/0
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            tcp to:10.244.3.6:80

Chain KUBE-SEP-VJKWIWD27NYXSES2 (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       10.244.3.19          0.0.0.0/0
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            tcp to:10.244.3.19:80

Chain KUBE-SEP-YIL6JZP7A3QYXJU2 (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       10.244.0.2           0.0.0.0/0
    0     0 DNAT       udp  --  *      *       0.0.0.0/0            0.0.0.0/0            udp to:10.244.0.2:53

Chain KUBE-SEP-ZXMNUKOKXUTL2MK2 (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       10.244.0.3           0.0.0.0/0
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            tcp to:10.244.0.3:53
```

为集群中的 service 服务创建规则，如下图中的 10.103.94.20、10.96.245.240、10.96.0.10 都是属于 service 网络的

```
Chain KUBE-SERVICES (2 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  tcp  --  *      *       !10.244.0.0/16       10.103.94.20         /* default/nginx: cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-4N57TFCL4MD7ZTDA  tcp  --  *      *       0.0.0.0/0            10.103.94.20         /* default/nginx: cluster IP */ tcp dpt:80
    0     0 KUBE-MARK-MASQ  tcp  --  *      *       !10.244.0.0/16       10.96.245.240        /* default/myapp: cluster IP */ tcp dpt:80
    0     0 KUBE-SVC-NPJI2GAOYBRMPXVD  tcp  --  *      *       0.0.0.0/0            10.96.245.240        /* default/myapp: cluster IP */ tcp dpt:80
    0     0 KUBE-MARK-MASQ  tcp  --  *      *       !10.244.0.0/16       10.96.0.1            /* default/kubernetes:https cluster IP */ tcp dpt:443
    0     0 KUBE-SVC-NPX46M4PTMTKRN6Y  tcp  --  *      *       0.0.0.0/0            10.96.0.1            /* default/kubernetes:https cluster IP */ tcp dpt:443
    0     0 KUBE-MARK-MASQ  tcp  --  *      *       !10.244.0.0/16       10.96.0.10           /* kube-system/kube-dns:dns-tcp cluster IP */ tcp dpt:53
    0     0 KUBE-SVC-ERIFXISQEP7F7OF4  tcp  --  *      *       0.0.0.0/0            10.96.0.10           /* kube-system/kube-dns:dns-tcp cluster IP */ tcp dpt:53
    0     0 KUBE-MARK-MASQ  udp  --  *      *       !10.244.0.0/16       10.96.0.10           /* kube-system/kube-dns:dns cluster IP */ udp dpt:53
    0     0 KUBE-SVC-TCOU7JCQXEZGVUNU  udp  --  *      *       0.0.0.0/0            10.96.0.10           /* kube-system/kube-dns:dns cluster IP */ udp dpt:53
    0     0 KUBE-NODEPORTS  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes service nodeports; NOTE: this must be the last rule in this
chain */ ADDRTYPE match dst-type LOCAL
```

第七章 资源清单

7.1 资源

7.1.1 资源类型

k8s 中所有的内容都抽象为资源， 资源实例化之后，叫做对象。

类别	名称
工作负载型资源对象	Pod、 Replicaset 、ReplicationController 、Deployments 、StatefulSets 、Daemonset、 Job、CronJob
服务发现及负载均衡	Service、 Ingress
配置与存储	Volume、Persistent Volume、CSI 、 configmap、 secret
集群资源	Namespace 、Node 、Role ClusterRole 、RoleBinding 、ClusterRoleBinding
元数据资源	HPA、PodTemplate、LimitRang

查看有哪些资源

```
kubectl api-resources
```

7.1.2 资源版本

一共有 5 个一级字段。

1) apiVersion （group/version）

查看哪些

```
kubectl api-versions
```

```
[root@master ~]#
[root@master ~]# kubectl api-versions
admissionregistration.k8s.io/v1beta1
apiextensions.k8s.io/v1beta1
apiregistration.k8s.io/v1
apiregistration.k8s.io/v1beta1
apps/v1
apps/v1beta1
apps/v1beta2
authentication.k8s.io/v1
authentication.k8s.io/v1beta1
authorization.k8s.io/v1
authorization.k8s.io/v1beta1
autoscaling/v1
autoscaling/v2beta1
autoscaling/v2beta2
batch/v1
batch/v1beta1
certificates.k8s.io/v1beta1
coordination.k8s.io/v1beta1
events.k8s.io/v1beta1
extensions/v1beta1
networking.k8s.io/v1
policy/v1beta1
rbac.authorization.k8s.io/v1
rbac.authorization.k8s.io/v1beta1
scheduling.k8s.io/v1beta1
storage.k8s.io/v1
storage.k8s.io/v1beta1
v1
[root@master ~]#
```

2) kind

3) Metadata

name

namespace

labels

annotations

资源引用：/api/GROUP/VERSION/namespace/NAMESPACE/TYPE

4) Spec

期望的状态，disired state

5) status

7.1.3 资源限制

如下，使用 resource 字段

```
apiVersion: apps/v1
    kind: Deployment
    metadata:
      name: stress
      namespace: default
    spec:
      replicas: 1
```

```
    selector:
      matchLabels:
        app: stress
    template:
      metadata:
        labels:
          app: stress
      spec:
        containers:
        - image: polinux/stress
          name: stress
          command: ['sh', '-c', 'stress -c 20']
          resources:
            limits:
              cpu: 2 #限制 cpu 使用率 200%，也可写作 500m，表示 0.5 个 50%
              memory: 1Gi
            requests:
              cpu: 1
              memory: 500Mi #Mi 表示使用 1024 进制，M 表示 1000 进制
```

测试效果：

我这里是 4 核 CPU，启动 20 个线程测试，总的 cpu 使用率 200%左右，且可以在 4 个核上运行，总的使用率限制为 200%即可。

```
[root@node1 ~]# top
top - 17:58:45 up 89 days,  8:22,  1 user,  load average: 19.82, 13.43, 8.55
Tasks: 173 total,  21 running, 152 sleeping,   0 stopped,   0 zombie
%Cpu0  : 51.5 us,  0.0 sy,  0.0 ni, 47.8 id,  0.0 wa,  0.0 hi,  0.3 si,  0.3 st
%Cpu1  : 52.0 us,  0.3 sy,  0.0 ni, 47.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 50.7 us,  2.0 sy,  0.0 ni, 46.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.7 st
%Cpu3  : 51.2 us,  0.7 sy,  0.0 ni, 47.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.3 st
KiB Mem :  8009524 total,   516008 free,   788404 used,  6705112 buff/cache
KiB Swap:  2097148 total,  2049020 free,    48128 used.  5547640 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
12141 root      20   0     732     40      0 R  11.9  0.0   1:08.53 stress
12158 root      20   0     732     40      0 R  11.6  0.0   1:05.82 stress
12149 root      20   0     732     40      0 R  11.3  0.0   1:08.77 stress
12150 root      20   0     732     40      0 R  11.3  0.0   1:06.10 stress
12157 root      20   0     732     40      0 R  10.6  0.0   1:09.66 stress
12140 root      20   0     732     40      0 R  10.3  0.0   1:08.12 stress
12139 root      20   0     732     40      0 R   9.9  0.0   1:05.43 stress
12142 root      20   0     732     40      0 R   9.9  0.0   1:06.89 stress
12145 root      20   0     732     40      0 R   9.9  0.0   1:06.92 stress
12146 root      20   0     732     40      0 R   9.9  0.0   1:09.95 stress
12151 root      20   0     732     40      0 R   9.9  0.0   1:05.92 stress
12153 root      20   0     732     40      0 R   9.9  0.0   1:06.24 stress
12155 root      20   0     732     40      0 R   9.9  0.0   1:06.99 stress
12148 root      20   0     732     40      0 R   9.6  0.0   1:07.24 stress
12152 root      20   0     732     40      0 R   9.3  0.0   1:08.30 stress
12143 root      20   0     732     40      0 R   8.9  0.0   1:05.66 stress
12144 root      20   0     732     40      0 R   8.9  0.0   1:10.19 stress
12147 root      20   0     732     40      0 R   8.9  0.0   1:07.80 stress
12154 root      20   0     732     40      0 R   8.9  0.0   1:05.15 stress
12156 root      20   0     732     40      0 R   8.9  0.0   1:06.49 stress
```

7.1.4 查看定义的方式

```
kubectl explain svc
kubectl explain pod.metadate.namespace
kubectl explain svc.spec.type
```

如下，查看 deployment

```
[root@master ~]#
[root@master ~]# kubectl explain deployment
KIND:     Deployment
VERSION:  extensions/v1beta1

DESCRIPTION:
     DEPRECATED - This group version of Deployment is
     apps/v1beta2/Deployment. See the release notes f
     Deployment enables declarative updates for Pods

FIELDS:
   apiVersion   <string>
     APIVersion defines the versioned schema of this
     object. Servers should convert recognized schema
     value, and may reject unrecognized values. More
     https://git.k8s.io/community/contributors/devel/

   kind <string>
     Kind is a string value representing the REST res
     represents. Servers may infer this from the endp
     requests to. Cannot be updated. In CamelCase. Mo
     https://git.k8s.io/community/contributors/devel/

   metadata     <Object>
     Standard object metadata.

   spec <Object>
     Specification of the desired behavior of the Dep

   status       <Object>
     Most recently observed status of the Deployment.
```

会有多种字段类型：

1）<string>，表示字符串

2）<Object>，表示对象，比如

deployment.metadata

name: mongodb

namespace: x2

3）<[]Object>，表示对象列表，比如

pod.spec.containers

- name: myapp

  image: ikuberneters/myaqpp:v1

- name: busybox

  image: busybox

4）<[]string>，表示字符串列表，比如

pod.spec.containers.command

command:

- "/bin/sh"
- "-c"

- " ***"

5）<boolean>，表示布尔类型，比如

6）<integer>，整型，比如

deployment.spec.replicas：4

7）<map[string]string>，字符类型的 map，比如

pod.metadata.labels

labels:

  app: redis

  abc: bcd

7.1.5 查看是否属于 namespace

```
# In a namespace
kubectl api-resources --namespaced=true

# Not in a namespace
kubectl api-resources --namespaced=false
```

7.2 coredns

k8s 默认 dns 搜索域是：svc.cluster.local. cluster.local.

7.4.1 Service 类型

- Record：my-svc.my-namespace.svc.cluster.local，解析 IP 分为两种情况

1）普通 Service 解析为 Cluster IP，

2）Headless Service 解析为指定的 Pod IP 列表

比如，无头服务 service-apollo-meta-server-dev

```
[root@master apollo-env-dev]#
[root@master apollo-env-dev]# dig service-apollo-meta-server-dev.apollo.svc.cluster.local @10.96.0.10 +short
10.244.6.51
10.244.2.205
10.244.1.109
[root@master apollo-env-dev]#
[root@master apollo-env-dev]#
[root@master apollo-env-dev]# kubectl get svc -n apollo
NAME                              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
apollo-portal                     ClusterIP   10.101.56.134   <none>        80/TCP         75m
service-apollo-admin-server-dev   ClusterIP   10.100.39.64    <none>        80/TCP         177m
service-apollo-config-server-dev  NodePort    10.99.102.218   <none>        80:30002/TCP   3h13m
service-apollo-meta-server-dev    ClusterIP   None            <none>        80/TCP         3h13m
[root@master apollo-env-dev]#
```

可根据这个特征，找到某个 pod 对应的 IP 地址，如下图，服务必须是无头服务。

```
[root@master apollo-env-dev]# kubectl get pod -n apollo -o wide|grep statefulset
statefulset-apollo-config-server-dev-0          1/1     Running   0          3h15m   10.244.6.51    node6   <none>
statefulset-apollo-config-server-dev-1          1/1     Running   1          75m     10.244.1.109   node1   <none>
statefulset-apollo-config-server-dev-2          1/1     Running   0          3h13m   10.244.2.205   node2   <none>
[root@master apollo-env-dev]#
[root@master apollo-env-dev]#
[root@master apollo-env-dev]# dig statefulset-apollo-config-server-dev-0.service-apollo-meta-server-dev.apollo.svc.cluster.local @10.96.0.10 +short
10.244.6.51
[root@master apollo-env-dev]#
[root@master apollo-env-dev]#
[root@master apollo-env-dev]# dig statefulset-apollo-config-server-dev-1.service-apollo-meta-server-dev.apollo.svc.cluster.local @10.96.0.10 +short
10.244.1.109
[root@master apollo-env-dev]#
[root@master apollo-env-dev]# dig statefulset-apollo-config-server-dev-2.service-apollo-meta-server-dev.apollo.svc.cluster.local @10.96.0.10 +short
10.244.2.205
[root@master apollo-env-dev]#
[root@master apollo-env-dev]#
```

- SRV 格式：生成 _my-port-name._my-port-protocol.my-svc.my-namespace.svc.cluster.local

7.4.2 Pod 类型

A record：pod-ip-address.my-namespace.pod.cluster.local

如下：

```
[root@master apollo-env-dev]#
[root@master apollo-env-dev]# kubectl get pod -n xmc -o wide|grep data-collector
xmc-data-collector-57f459667b-4q6mg         1/1     Running   0          11d     10.244.6.233   node6     <none>
[root@master apollo-env-dev]#
[root@master apollo-env-dev]#
[root@master apollo-env-dev]# dig 10-244-6-233.xmc.pod.cluster.local @10.96.0.10 +short
10.244.6.233
[root@master apollo-env-dev]#
[root@master apollo-env-dev]#
```

7.3 NameSpace 资源

7.4.3 创建 ns

方法一：命令行创建

```
kubectl create ns foo
```

方法二：使用 yaml 文件创建

```
apiVersion: v1
kind: Namespace
metadata:
    name: x2
    labels:
        name: x2
```

```
apiVersion: v1
kind: Namespace
metadata:
    name: x2
```

```
apiVersion: v1
kind: Namespace
metadata:
  name: istio-system
  labels:
    istio-injection: disabled
```

```
apiVersion: v1
kind: Namespace
```

```
metadata:
  name: kubernetes-dashboard
```

7.4.4 删除 ns

命令行删除

```
kubectl create ns foo
```

如果删除不了，如下图

```
[root@master test]# kubectl get ns harbor
NAME      STATUS        AGE
harbor    Terminating   30d
[root@master test]#
```

```
[root@master test]# kubectl get all -n harbor
NAME                                                  READY   STATUS    RESTARTS   AGE
pod/harbor-harbor-clair-6557bcd8f7-t6kxd              0/1     Unknown   29         157m
pod/harbor-harbor-database-0                          1/1     Unknown   1          6h32m
pod/harbor-harbor-notary-signer-99f5d694c-8x96r       1/1     Unknown   24         157m
[root@master test]#
[root@master test]#
```

此时需要强制删除 pod

7.4 pod 资源

7.4.1 资源定义的完整格式

```
apiVersion: v1                          #必选，版本号，例如 v1,版本号必须可以用
kubectl api-versions 查询到 .
kind: Pod                               #必选，Pod
metadata:                               #必选，元数据
  name: string                          #必选，Pod 名称
  namespace: string                     #必选，Pod 所属的命名空间,默认为"default"
  labels:                               #自定义标签
    - name: string                      #自定义标签名字
  annotations:                          #为对象提供"元数据"，不能用于挑选资源对象，在某些场景下，必须添加
    - name: string
spec:                                   #必选，Pod 中容器的详细定义
  containers:                           #必选，Pod 中容器列表
  - name: string                        #必选，容器名称,需符合 RFC 1035 规范
    image: string                       #必选，容器的镜像名称
    imagePullPolicy: [ Always|Never|IfNotPresent ]  #获取镜像的策略 Alawys 表示下载镜像 IfnotPresent 表示优先使用本地镜像,否则下载镜像，Nerver 表示仅使用本地镜像
```

```
拉取最新 latest 镜像：默认是 always
拉取的不是 latest 镜像：默认是 IfnotPresent
    command: [string]                    #容器的启动命令列表，如不指定，使用打包时使用的启动命令
    args: [string]                       #容器的启动命令参数列表
#和 dockerfile 里面的 cmd、entpoints 有关，具体如何生效，见：https://kubernetes.io/docs/tasks/inject-data-application/define-command-argument-container/
    workingDir: string                   #容器的工作目录
    volumeMounts:                        #挂载到容器内部的存储卷配置
    - name: string                       #引用 pod 定义的共享存储卷的名称，需用 volumes[]部分定义的的卷名
      mountPath: string                  #存储卷在容器内 mount 的绝对路径，应少于 512 字符
      readOnly: boolean                  #是否为只读模式
    ports:                               #需要暴露的端口库号列表
    - name: string                       #端口的名称
      containerPort: int                 #容器需要监听的端口号
      hostPort: int                      #容器所在主机需要监听的端口号，默认与 Container 相同
      protocol: string                   #端口协议，支持 TCP 和 UDP，默认 TCP
    env:                                 #容器运行前需设置的环境变量列表
    - name: string                       #环境变量名称
      value: string                      #环境变量的值
    resources:                           #资源限制和请求的设置
      limits:                            #资源限制的设置
        cpu: string                      #Cpu 的限制，单位为 core 数，将用于 docker run --cpu-shares 参数
        memory: string                   #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: string                      #Cpu 请求，容器启动的初始可用数量
        memory: string                   #内存请求,容器启动的初始可用数量
    livenessProbe:                       #对 Pod 内各容器健康检查的设置，当探测无响应几次后将自动重启该容器，检查方法有 exec、httpGet 和 tcpSocket，对一个容器只需设置其中一种方法即可
      exec:                              #对 Pod 容器内检查方式设置为 exec 方式
        command: [string]                #exec 方式需要制定的命令或脚本
      httpGet:                           #对 Pod 内个容器健康检查方法设置为 HttpGet，需要制定 Path、port
        path: string
        port: number
        host: string
        scheme: string
        HttpHeaders:
```

```
          - name: string
            value: string
        tcpSocket:                          #对 Pod 内个容器健康检查方式设置为 tcpSocket 方式
            port: number
          initialDelaySeconds: 0            #容器启动完成后首次探测的时间，单位为秒
          timeoutSeconds: 0                 #对容器健康检查探测等待响应的超时时间，单位秒，默认 1 秒
          periodSeconds: 0                  #对容器监控检查的定期探测时间设置，单位秒，默认 10 秒一次
          successThreshold: 0
          failureThreshold: 0
          securityContext:
            privileged: false
      restartPolicy: [Always | Never | OnFailure] #Pod 的重启策略，Always 表示一旦不管以何种方式终止运行，kubelet 都将重启，OnFailure 表示只有 Pod 以非 0 退出码退出才重启，Nerver 表示不再重启该 Pod
      nodeSelector: obeject               #设置 NodeSelector 表示将该 Pod 调度到包含这个 label 的 node 上，以 key：value 的格式指定
      imagePullSecrets:                   #Pull 镜像时使用的 secret 名称，以 key：secretkey 格式指定
      - name: string
      hostNetwork: false                  #是否使用主机网络模式，默认为 false，如果设置为 true，表示使用宿主机网络
      volumes:                            #在该 pod 上定义共享存储卷列表
      - name: string                      #共享存储卷名称 （volumes 类型有很多种）
        emptyDir: {}                      #类型为 emtyDir 的存储卷，与 Pod 同生命周期的一个临时目录。为空值
        hostPath: string                  #类型为 hostPath 的存储卷，表示挂载 Pod 所在宿主机的目录
          path: string                    #Pod 所在宿主机的目录，将被用于同期中 mount 的目录
        secret:                           #类型为 secret 的存储卷，挂载集群与定义的 secre 对象到容器内部
          scretname: string
          items:
          - key: string
            path: string
        configMap:                        #类型为 configMap 的存储卷，挂载预定义的 configMap 对象到容器内部
          name: string
          items:
          - key: string
            path: string
```

7.4.2 一个 pod 包含一个容器

一般很少直接创建 pod，而是通过 pod 控制器创建的，所以这里举例使用 deployment 创建的 pod

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: quzl
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: nginx
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx
  namespace: quzl
spec:
   replicas: 2
   template:
     metadata:
       labels:
         app: nginx
     spec:
       containers:
       - name: nginx
         image: nginx
         ports:
         - containerPort: 80
         volumeMounts:
         - name: web-base
           mountPath: /usr/share/nginx/html
       volumes:
       - name: web-base
         hostPath:
           path: /data/nginx-html
```

7.4.3 一个 pod 包含多个容器

同时，这里还举例了共用存储卷。需要说明的是，虽然多个容器都位于同一个 pod，但是其文件系统并不是相同的。

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  namespace: quzl
spec:
  containers:
  - name: myapp
    image: ikubernetes/myapp:v1
    volumeMounts:
    - name: www-data
      mountPath: /usr/share/nginx/html
  - name: busybox
    image: busybox
    volumeMounts:
    - name: www-data
      mountPath: /data/www
    command:
    - "/bin/sh"
    - "-c"
    - "while true;do echo $(date) >> /data/www/index.html;sleep 10 ;done"
  volumes:
  - name: www-data
    emptyDir: {}
```

另外，对于单个 pod 包含多个容器的情况，当使用 log 命令时，需要使用-c 指明要查看的是哪个容器

```
[root@master k8s-test]#
[root@master k8s-test]# kubectl logs myapp -c busybox -n quzl
[root@master k8s-test]#
[root@master k8s-test]# kubectl logs myapp  -n quzl
Error from server (BadRequest): a container name must be specified for pod myapp, choose one of: [myapp busybox]
[root@master k8s-test]#
[root@master k8s-test]#
[root@master k8s-test]# kubectl logs myapp -c myapp -n quzl
10.244.0.0 - - [31/Mar/2019:20:58:43 +0000] "GET / HTTP/1.1" 200 87 "-" "curl/7.29.0" "-"
10.244.0.0 - - [31/Mar/2019:20:58:50 +0000] "GET / HTTP/1.1" 200 116 "-" "curl/7.29.0" "-"
10.244.0.0 - - [31/Mar/2019:20:58:52 +0000] "GET / HTTP/1.1" 200 116 "-" "curl/7.29.0" "-"
10.244.0.0 - - [31/Mar/2019:20:58:53 +0000] "GET / HTTP/1.1" 200 116 "-" "curl/7.29.0" "-"
10.244.0.0 - - [31/Mar/2019:20:58:54 +0000] "GET / HTTP/1.1" 200 116 "-" "curl/7.29.0" "-"
[root@master k8s-test]#
```

7.4.4 添加 host 解析

和 containers 并列的地方添加，deployment 中也是类似。

```
apiVersion: v1
kind: Pod
metadata:
  name: jenkinsTemplate
```

```
  namespace: devops
spec:
  hostAliases:
  - ip: 192.168.3.140
    hostnames:
    - sonar.ops.dm-ai.cn
  - ip: 192.168.3.221
    hostnames:
    - gitlab.dm-ai.cn
  containers:
```

创建后的 pod，自动添加到/etc/hosts 文件中

```
[root@master sonar]#
[root@master sonar]# kubectl get pod -n devops
NAME                              READY   STATUS    RESTARTS   AGE
sonar-scanner-55ccc4d45-w74nk     1/1     Running   0          5s
sonarqube-76bdddd5b-jmxnc         1/1     Running   0          6h49m
[root@master sonar]#
[root@master sonar]#
[root@master sonar]# kubectl exec -it sonar-scanner-55ccc4d45-w74nk sh -n devops
/sonar-scanner #
/sonar-scanner # cat /etc/hosts
# Kubernetes-managed hosts file.
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
fe00::0 ip6-mcastprefix
fe00::1 ip6-allnodes
fe00::2 ip6-allrouters
10.244.1.70     sonar-scanner-55ccc4d45-w74nk

# Entries added by HostAliases.
192.168.3.140   sonar.ops.dm-ai.cn
/sonar-scanner #
/sonar-scanner #
```

7.4.5 env

1）来源于 cm 类型的

```
        env:
          - name: SPECIAL_LEVEL_KEY
            valueFrom:
              configMapKeyRef:
                name: special-config
                key: special.how
```

此时的 cm 格式:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
```

```
  namespace: default
data:
  special.how: very
  special.type: charm
```

2）来源于 fieldRef 类型的，支持以下几种：
kubectl explain pod.spec.containers.env.valueFrom
metadata.name
metadata.namespace,
metadata.labels
metadata.annotations
spec.nodeName
spec.serviceAccountName
status.hostIP
status.podIP
举例如下：

```
        env:
          - name: POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: POD_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
```

3）直接定义

```
    env:
    - name: TZ
      value: Asia/Shanghai
```

7.4.6 envFrom

1）from cm 的

```
        envFrom:
        - configMapRef:
```

```
                name: tk-engine-video-extract
```

此时 cm 格式如下：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: tk-engine-image-process-teacher
  namespace: xmc-tk
data:
  NODE6: '192.168.3.30:9092'
  NODE7: '192.168.3.31:9092'
  NODE8: '192.168.3.32:9092'
```

2）from secret 的

和 from cm 类似

7.5 Service 资源

7.5.1 Service 的实现技术

userspace: 1.1 之前使用，效率很低

iptables: 1.10 之后加入

ipvs: 1.11 之后加入

启用 ipvs 方法：

1） 安装 kubelet 后，配置文件中指定使用 ipvs

```
echo "KUBE_PROXY_MODE=ipvs" >> /etc/sysconfig/kubelet
```

2） 保证开机时自动加载如下模块

ip_vs,ip_vs_rr,ip_vs_wrr,ip_vs_sh,nf_conntrack_ipv4

如果未使用 ipvs，使用 join 加入集群时，会有以下警告。

```
[root@node1323 ~]# kubeadm join 192.168.11.20:6443 --token 6dmsrk.s6vijhgnfh120j2h --discovery-token-c
a-cert-hash sha256:cbff9300b8b76710d35d00954cd0ee36ec73fc4f26afa492cce38feb7fe29664
[preflight] running pre-flight checks
        [WARNING RequiredIPVSKernelModulesAvailable]: the IPVS proxier will not be used, because the f
ollowing required kernel modules are not loaded: [ip_vs_wrr ip_vs_sh ip_vs ip_vs_rr] or no builtin ker
nel ipvs support: map[ip_vs_sh:{} nf_conntrack_ipv4:{} ip_vs:{} ip_vs_rr:{} ip_vs_wrr:{}]
you can solve this problem with following methods:
 1. Run 'modprobe -- ' to load missing kernel modules;
2. Provide the missing builtin kernel ipvs support

[discovery] Trying to connect to API Server "192.168.11.20:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://192.168.11.20:6443"
[discovery] Requesting info from "https://192.168.11.20:6443" again to validate TLS against the pinned
 public key
[discovery] Cluster info signature and contents are valid and TLS certificate validates against pinned
```

[WARNING RequiredIPVSKernelModulesAvailable]: the IPVS proxier will not be used, because the following required kernel modules are not loaded: [ip_vs_wrr ip_vs_sh ip_vs ip_vs_rr] or no builtin kernel ipvs support: map[ip_vs_sh:{} nf_conntrack_ipv4:{} ip_vs:{} ip_vs_rr:{} ip_vs_wrr:{}]

you can solve this problem with following methods:

1. Run 'modprobe -- ' to load missing kernel modules;
2. Provide the missing builtin kernel ipvs support

7.5.2 ClusterIP 类型的 svc

最常见的一种类型，由于集群内部通信，举例：

```
apiVersion: v1
kind: Service
metadata:
    name: redis
    namespace: default
spec:
    type: ClusterIP
    ports:
    - port: 6379
      targetPort: 6379
    selector:
      app: redis
      role: logstor
     #sessionAffinity: None #默认
     #sessionAffinity: ClientIP #启用会话保持，类似于 nginx 的 ip hash
```

7.5.3 NodePort 类型的 svc

举例：

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort //指定 service 类型
  selector:
    app: forme
  ports:
    - port: 80 // 供集群中其它 container 访问端口
      targetPort: 8020 //转向后端 pod 中 container 暴露的端口
      nodePort: 9090 //节点暴露的端口
```

默认的，nodePort 的范围是 30000-32767, k8s 会从中随机选择一个端口，可以通过修改 apiserver 的--service-node-port-range 的参数来修改默认范围，如：--service-node-port-range 8000-9000。

对于已经部署完成的 k8s 集群，也可以通过如下文件修改此参数：

/etc/kubernetes/manifests/kube-apiserver.yaml

- --service-node-port-range=0-32767

```
[root@master k8s-test]# !vim
vim /etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    scheduler.alpha.kubernetes.io/critical-pod: ""
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --authorization-mode=Node,RBAC
    - --advertise-address=192.168.3.78
    - --allow-privileged=true
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
    - --insecure-port=0
    - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-clie
    - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
    - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
    - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
    - --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key
    - --requestheader-allowed-names=front-proxy-client
    - --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
    - --requestheader-extra-headers-prefix=X-Remote-Extra-
    - --requestheader-group-headers=X-Remote-Group
    - --requestheader-username-headers=X-Remote-User
    - --secure-port=6443
    - --service-account-key-file=/etc/kubernetes/pki/sa.pub
    - --service-cluster-ip-range=10.96.0.0/12
    - --service-node-port-range=0-32767
    - --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
    - --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
    image: k8s.gcr.io/kube-apiserver:v1.12.2
```

修改后动态生效，自动更新此 pod。

更新后，可以使用其他端口，如下举例：

```
[root@master k8s-test]# vim aliy-nginx.yml
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
      nodePort: 200
  selector:
    app: nginx
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
   selector:
     matchLabels:
       app: nginx
   replicas: 1
   template:
     metadata:
       labels:
         app: nginx
     spec:
       imagePullSecrets:
       - name: aliyun-regsecret
       containers:
       - name: nginx
         image: registry.cn-shenzhen.aliyuncs.com/xmc2/nginx
~
```

```
[root@master k8s-test]# netstat -tnlp|grep "200\>"
tcp6       0      0 :::200                  :::*
[root@master k8s-test]#
```

访问路径:

clinent --> NodeIP:NodePort --> ClusterIP:ClusterPort --> PodIP:PodPort

如果单独访问一个 node,该 node 可能压力过大，所以可以使用 LocadBalance 方式，该类型参考 7.5.5

7.5.4 LoadBalancer 类型的 svc

service 在每台主机的 iptables/ipvs 规则内，访问任意一台 node 都可以到达 pod，所以应该在这些 node_ip 前加负载均衡器，如果工作在公有云，可以在公网使用 loadBalancerIP，操作公有云的负载均衡器即服务。

可以理解为 loadBalancerIP 增强了 NodePort 类型的 service ，在集群外部对每台 nodeip 进行负载均衡。

其实这种类型就是在每个 node 创建 ipvs/iptables 规则，供外部的负载均衡器再次调取，比如阿里云的 slb、腾讯云的 clb,自动修改 slb 规则。

可以理解为一种增强的 NodePort 类型。

如下是腾讯云上的一个示例

```
[root@VM_0_16_centos ~]#
[root@VM_0_16_centos ~]# kubectl get svc
NAME          TYPE           CLUSTER-IP     EXTERNAL-IP        PORT(S)        AGE
kubernetes    ClusterIP      10.3.255.1     <none>             443/TCP        6d8h
nginx-test    LoadBalancer   10.3.255.103   193.112.239.200    80:32377/TCP   2m11s
[root@VM_0_16_centos ~]#
[root@VM_0_16_centos ~]#
```

yaml 示例:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-test
  namespace: default
spec:
  clusterIP: 10.3.255.103
  externalTrafficPolicy: Cluster
  ports:
  - name: tcp-80-80
    nodePort: 32377
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    k8s-app: nginx-test
    qcloud-app: nginx-test
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 193.112.239.200
```

7.5.5 无头类型的 svc

无头 service,无 IP 的 cluserip 类型，经常用于 statefulset 类型的资源。此时流程是 pod-->service name --> pod name

定义时：ClusterIP: "None "

举例:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: elasticsearch
```

```
      chart: elasticsearch-1.28.5
      component: master
      heritage: Helm
      release: skywalking
    name: skywalking-elasticsearch-discovery
    namespace: skywalking
spec:
    clusterIP: None
    ports:
    - port: 9300
      protocol: TCP
      targetPort: transport
    selector:
      app: elasticsearch
      component: master
      release: skywalking
    sessionAffinity: None
    type: ClusterIP
```

使用 dig 命令进行解析，可以看到三条 A 记录

```
[root@master samples]# dig skywalking-elasticsearch-discovery.skywalking.svc.cluster.local @10.96.0.10

; <<>> DiG 9.9.4-RedHat-9.9.4-73.el7_6 <<>> skywalking-elasticsearch-discovery.skywalking.svc.cluster.local @10.96.0.10
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32517
;; flags: qr aa rd; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;skywalking-elasticsearch-discovery.skywalking.svc.cluster.local. IN A

;; ANSWER SECTION:
skywalking-elasticsearch-discovery.skywalking.svc.cluster.local. 5 IN A 10.244.12.215
skywalking-elasticsearch-discovery.skywalking.svc.cluster.local. 5 IN A 10.244.16.98
skywalking-elasticsearch-discovery.skywalking.svc.cluster.local. 5 IN A 10.244.15.147

;; Query time: 1 msec
;; SERVER: 10.96.0.10#53(10.96.0.10)
;; WHEN: Thu Nov 28 19:14:50 CST 2019
;; MSG SIZE  rcvd: 329

[root@master samples]#
[root@master samples]#
[root@master samples]# dig skywalking-elasticsearch-discovery.skywalking.svc.cluster.local @10.96.0.10 +short
10.244.15.147
10.244.12.215
10.244.16.98
[root@master samples]#
```

7.5.6 ExternalName 类型的 svc

ExternalName 是 Service 的特例，它没有 selector，也没有定义任何的端口和 Endpoint。 对于运行在集群外部的服务，它通过返回该外部服务的别名这种方式来提供服务。

```
kind: Service
apiVersion: v1
metadata:
    name: my-service
    namespace: prod
spec:
```

```
  type: ExternalName
  externalName: my.database.example.com
```

当查询主机 my-service.prod.svc.cluster.local 时，集群的 DNS 服务将返回一个值为 my.database.example.com 的 CNAME 记录。 访问这个服务的工作方式与其它的相同，唯一不同的是重定向发生在 DNS 层，而且不会进行代理或转发。 如果后续决定要将数据库迁移到 Kubernetes 集群中，可以启动对应的 Pod，增加合适的 Selector 或 Endpoint，修改 Service 的 type，完全不需要修改调用的代码，这样就完全解耦了。

7.5.7 自定义 endpoint 的 service

也可以将外部的服务器定位为内部的服务

```
apiVersion: v1
kind: Service
metadata:
  name: cp-service
spec:
  ports:
    - port: 80
---
kind: Endpoints
apiVersion: v1
metadata:
  name: cp-service
subsets:
  - addresses:
      - ip: 192.168.3.54
    ports:
      - port: 80


#下面的用于创建 ingress,如果是 k8s 内部其他模块调用该 service，则不需要这个 ingress，只需要上面两个即可，就像使用普通 service 一样进行使用即可。
---
kind: Ingress
apiVersion: extensions/v1beta1
metadata:
  name: cp-frontend
  annotations:
    kubernetes.io/ingress.class: traefik
    ingress.kubernetes.io/whitelist-source-range:
"98.152.216.154,98.152.216.155,98.152.216.156,98.152.216.157,98.152.216.158,192.168.0.0/16"
spec:
```

```
  rules:
  - host: cp.dm-ai.cn
    http:
      paths:
      - backend:
          serviceName: cp-service
          servicePort: 80
```

7.6 ingress

ingress 也是一种标准的 k8s 资源，是 k8s 的重要的四种附件(dns、dashboard、ingress、heapster)之一。ingress 用于解决普通 service 不能七层调度的问题，ingress 可以使用 nginx 或者 traefik 等创建，这里使用 traefik 创建。

7.6.1 创建 secret

```
kubectl create secret tls https-key-secret --key dm-ai.cn.key --cert dm-ai.cn.pem -n kube-system
```

或者如下，保存成文件，后续便于引用

```
kubectl create secret tls https-key-secret --key dm-ai.cn.key --cert dm-ai.cn.pem -n kube-system --dry-run -o yaml > ssl-secret.yaml
```

这里是将 https 正式创建为 secret，如果不使用 https，也可以不创建

7.6.2 创建 traefik

yaml 内容如下：

```
#创建 SA
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: traefik-ingress-controller-sa
  namespace: kube-system

#创建 clusterrole
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: traefik-ingress-controller-cr
```

```
rules:
  - apiGroups:
      - ""
    resources:
      - services
      - endpoints
      - secrets
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - extensions
    resources:
      - ingresses
    verbs:
      - get
      - list
      - watch

#创建 ClusterRoleBinding
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: traefik-ingress-controller-crb
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: traefik-ingress-controller-cr
subjects:
- kind: ServiceAccount
  name: traefik-ingress-controller-sa
  namespace: kube-system



#创建 cm，traefik 的配置文件
---
kind: ConfigMap
apiVersion: v1
metadata:
  name: traefik-conf
  namespace: kube-system
```

```
data:
  traefik.toml: |
    insecureSkipVerify = true
    defaultEntryPoints = ["http","https"]
    [entryPoints]
      [entryPoints.http]
      address = ":80"
      [entryPoints.https]
      address = ":443"
        [entryPoints.https.tls]
          [[entryPoints.https.tls.certificates]]
            CertFile = "/ssl/dm-ai.cn.pem"
            KeyFile = "/ssl/dm-ai.cn.key"

#创建 ingress controller by traefik
---
kind: DaemonSet
apiVersion: extensions/v1beta1
metadata:
  name: traefik-ingress-controller
  namespace: kube-system
  labels:
    k8s-app: traefik-ingress-lb
spec:
  template:
    metadata:
      labels:
        k8s-app: traefik-ingress-lb
        name: traefik-ingress-lb
    spec:
      serviceAccountName: traefik-ingress-controller-sa
      #指定在 master 上运行
      nodeSelector:
        ingress: "traefik"
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      terminationGracePeriodSeconds: 60
      #表示使用共享宿主机的网络名称空间，所以不需要 service 了
      hostNetwork: true
      containers:
      - image: traefik:alpine
        name: traefik-ingress-lb
        ports:
```

```
            - name: http
              containerPort: 80
              hostPort: 80
            - name: https
              containerPort: 443
              hostPort: 443
            - name: admin
              containerPort: 8080
              hostPort: 8080
            env:
            - name: TZ
              value: Asia/Shanghai
            securityContext:
              privileged: true
            args:
            - --configfile=/config/traefik.toml
            - -d
            - --web
            - --kubernetes
            volumeMounts:
            - mountPath: "/config"
              name: "config"
            - mountPath: "/ssl"
              name: "ssl-key"
          volumes:
          - name: config
            configMap:
              name: traefik-conf
          - name: ssl-key
            secret:
              secretName: https-key-secret

#web-ui 的 service 创建
---
apiVersion: v1
kind: Service
metadata:
  name: traefik-web-ui
  namespace: kube-system
spec:
  selector:
    k8s-app: traefik-ingress-lb
  ports:
  - port: 80
```

```
      targetPort: 8080


#创建 ingress 规则
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
   name: traefik-web-ui
   namespace: kube-system
   annotations:
      kubernetes.io/ingress.class: traefik
spec:
   rules:
   - host: ingress.k8s.dm-ai.cn
      http:
         paths:
         - backend:
               serviceName: traefik-web-ui
               servicePort: 80
```

7.6.3 创建 ingress 规则

以下是一个例子:

```
---
kind: Ingress
apiVersion: extensions/v1beta1
metadata:
  name: kubernetes-dashboard
  namespace: kube-system
  annotations:
     kubernetes.io/ingress.class: traefik
     ingress.kubernetes.io/ssl-redirect: "true"     #http 自动转向 https
spec:
#如果是 http，删除以下两行
  tls:
     - secretName: https-key-secret
  rules:
  - host: k8s.dm-ai.cn
     http:
        paths:
        - path: /
           backend:
```

```
            serviceName: kubernetes-dashboard
            servicePort: 443 #如果 http、https 端口可能不一样，注意修改
```

多个 path 示例：

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: rdac-api
  namespace: devops
  annotations:
    kubernetes.io/ingress.class: traefik
spec:
  rules:
  - host: dev.api-rdac.dm-ai.cn
    http:
      paths:
    - path: /api/ota/v1
        backend:
          serviceName: rdac-ota-service
          servicePort: 80
      - path: /api/license/v1
        backend:
          serviceName: rdac-license-manager
          servicePort: 80
```

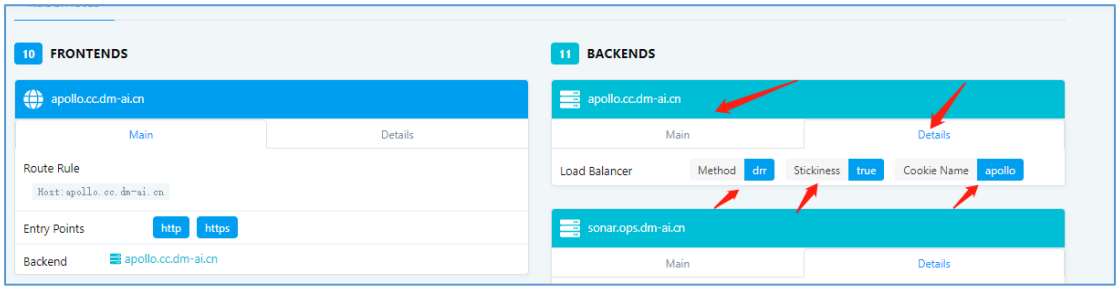
7.6.4 会话保持

traefik 的会话保持，不是在 ingress 规则中实现，而是在 service 的定义中实现的，现在是一个例子,注意红色字体部分：

```
kind: Service
apiVersion: v1
metadata:
  namespace: apollo
  name: apollo-portal
  annotations:
    traefik.ingress.kubernetes.io/affinity: "true"
    traefik.ingress.kubernetes.io/session-cookie-name: "apollo"
    traefik.ingress.kubernetes.io/load-balancer-method: drr
  labels:
    app: apollo-portal
spec:
  type: NodePort
```

```
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
      nodePort: 32221
  selector:
    app: apollo-portal
```

增加完成之后，可以在 ingress 的管理界面看到如下内容：



7.7 ConfigMap

7.6.5 创建

1）使用字符串创建

```
kubectl create configmap nginx-config --from-literal=nginx_port=80 --from-literal=srver_name=www.quzl.com -n quzl
```

查看：

```
[root@master k8s-test]#
[root@master k8s-test]# kubectl get configmap nginx-config -n quzl -o yaml
apiVersion: v1
data:
  nginx_port: "80"
  srver_name: www.quzl.com
kind: ConfigMap
metadata:
  creationTimestamp: 2019-03-31T23:57:00Z
  name: nginx-config
  namespace: quzl
  resourceVersion: "1010045"
  selfLink: /api/v1/namespaces/quzl/configmaps/nginx-config
  uid: aca0b768-5410-11e9-b7e8-5254009b4e6b
```

创建 pod 时引用：

```
        env:
        - name: PORT
          valueFrom:
            configMapKeyRef:
              name: nginx-config
              key: nginx_port
        - name: SERVER_NAME
          valueFrom:
            configMapKeyRef:
```

```
name: nginx-config
key: srver_name
```

```
      app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
        - name: web-base
          mountPath: /usr/share/nginx/html
        env:
        - name: PORT
          valueFrom:
            configMapKeyRef:
              name: nginx-config
              key: nginx_port
        - name: SERVER_NAME
          valueFrom:
            configMapKeyRef:
              name: nginx-config
              key: srver_name
```

创建 pod 后查看是否生效:

```
[root@master k8s-test]#
[root@master k8s-test]# kubectl exec -it nginx-5b8f5f47cb-72prr -n quzl -- printenv
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=nginx-5b8f5f47cb-72prr
TERM=xterm
PORT=80
SERVER_NAME=www.quzl.com
KUBERNETES_PORT_443_TCP_PROTO=tcp
NGINX_SERVICE_HOST=10.102.170.97
NGINX_PORT_80_TCP=tcp://10.102.170.97:80
NGINX_PORT_80_TCP_PROTO=tcp
```

2）使用文件创建，文件名就是 key，文件内容就是 value

```
kubectl create configmap nginx-www --from-file=www.conf
kubectl create configmap nginx-www --from-file=www.conf -n quzl
```

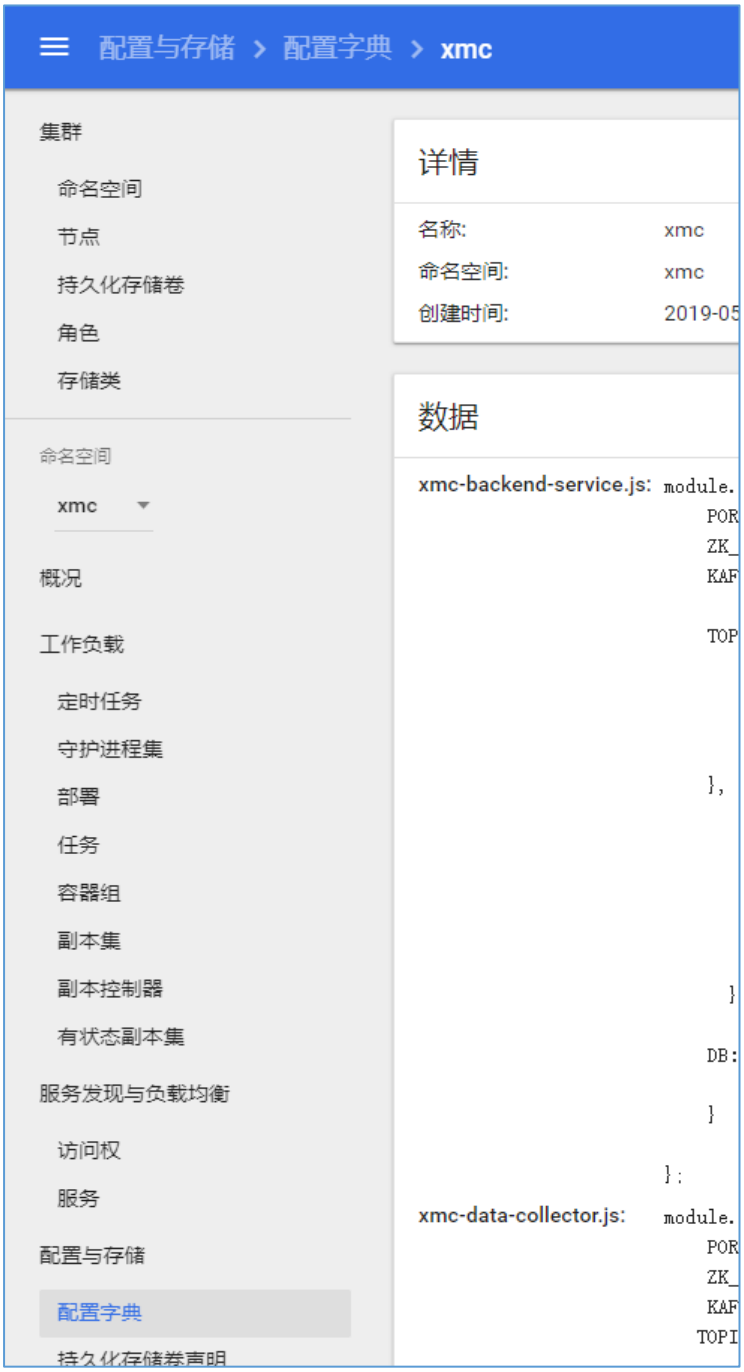
引用:

```
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
        - name: web-base
          mountPath: /usr/share/nginx/html
        - name: nginxconf
          mountPath: /etc/nginx/conf.d/
          readOnly: true
        env:
        - name: PORT
          valueFrom:
            configMapKeyRef:
              name: nginx-config
              key: nginx_port
        - name: SERVER_NAME
          valueFrom:
            configMapKeyRef:
              name: nginx-config
              key: srver_name
      volumes:
      - name: web-base
        hostPath:
          path: /data/nginx-html
      - name: nginxconf
        configMap:
          name: nginx-www
```

2）使用文件夹创建，文件名就是 key，文件内容就是 value
比如目录 dir 下面有 xmc-backend-service.config，xmc-data-collector.js 等

kubcel create cm xmc -n xmc --from-file=./dir

创建完成后的 cm 如下所示，cm 中有多条内容，可跟进文件名进行引用



yaml 文件中的引用如下，其中 xmc 就是 cm 的名字，xmc-backend-service.js 就是 dir 目录中的文件名

```
      value: Asia/Shanghai
    volumeMounts:
    - name: myconf
      mountPath: /usr/local/xmc-backend-service/config.js
      subPath: config.js
    ports:
    - containerPort: 31147
  volumes:
  - name: myconf
    configMap:
      name: xmc
      items:
      - key: xmc-backend-service.js
        path: config.js
```

更多参考:
https://blog.csdn.net/liukuan73/article/details/79492374

7.6.6 删除

kubecel delete cm *** -n ***

7.8 控制器

7.8.1 ReplicationController

7.8.2 ReplicaSet

7.8.3 Deployment

举例:

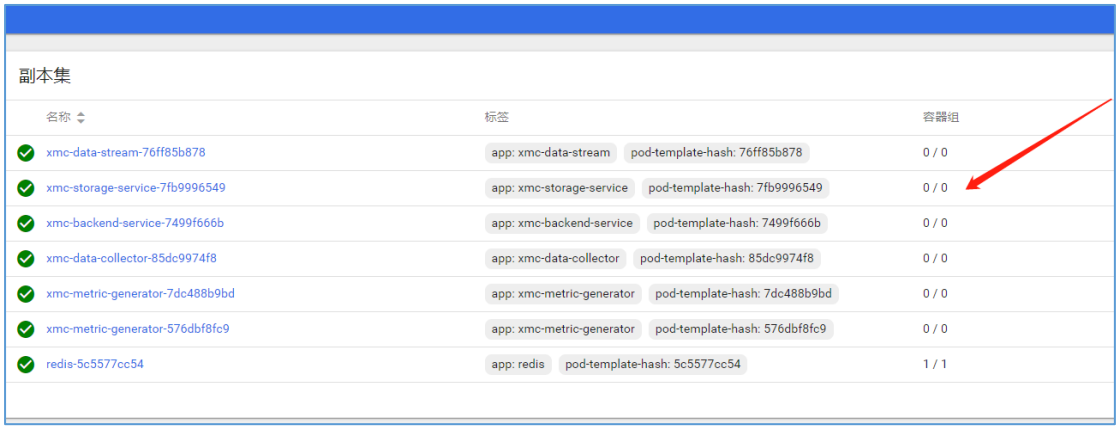
```
apiVersion: apps/v1
kind: Deployment
metadata:
  app: nginx-deployment
spec:
  selector:
    matchLabels:
```

```
        app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

以下配置项目，表示保留旧版本的 rs 数量，默认会保留 10 个。

```
deployment.spec.revisionHistoryLimit
```

每保留一个，这里就会多一个容器组为 0 的 rs



使用命令行也可以查看到

```
[root@master1120 ingress-demo]# kubectl get rs -n xmc
NAME                                  DESIRED   CURRENT   READY   AGE
dm-model-serving-student-586bdb7778   4         4         4       18d
dm-model-serving-teacher-6cccc79686   6         6         6       18d
redis-5c5577cc54                      1         1         1       41d
xmc-backend-service-7499f666b         0         0         0       31d
xmc-backend-service-8c87666fc         3         3         3       21d
xmc-backend-service-c5f8588d          0         0         0       26d
xmc-data-collector-75668744           3         3         3       26d
xmc-data-collector-85dc9974f8         0         0         0       31d
xmc-data-stream-6cd7cd8c7d            0         0         0       26d
xmc-data-stream-745d55cccf            3         3         3       26d
xmc-data-stream-76ff85b878            0         0         0       27d
xmc-metric-generator-576dbf8fc9       0         0         0       37d
xmc-metric-generator-78cc79c8db       3         3         3       26d
xmc-metric-generator-7dc488b9bd       0         0         0       31d
xmc-storage-service-65f959cb6b        3         3         3       24d
xmc-storage-service-7fb9996549        0         0         0       27d
xmc-storage-service-85879c789         0         0         0       26d
[root@master1120 ingress-demo]#
[root@master1120 ingress-demo]#
```

这种设计，就是为了回滚使用的，如果不想保留太多，可以设置这个参数

7.8.4 DaemontSet

可以保证每个 node 上只有一个 pod。下面是一个例子：

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: nvidia-device-plugin-daemonset-1.12
  namespace: kube-system
spec:
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      # Mark this pod as a critical add-on; when enabled, the critical add-on scheduler
      # reserves resources for critical add-on pods so that they can be rescheduled after
      # a failure.  This annotation works in tandem with the toleration below.
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ""
      labels:
        name: nvidia-device-plugin-ds
    spec:
      tolerations:
      # Allow this pod to be rescheduled while the node is in "critical add-ons only" mode.
      # This, along with the annotation above marks this pod as a critical add-on.
      - key: CriticalAddonsOnly
        operator: Exists
      - key: nvidia.com/gpu
        operator: Exists
        effect: NoSchedule
      containers:
      - image: nvidia/k8s-device-plugin:1.11
        name: nvidia-device-plugin-ctr
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop: ["ALL"]
        volumeMounts:
          - name: device-plugin
            mountPath: /var/lib/kubelet/device-plugins
      volumes:
        - name: device-plugin
          hostPath:
```

path: /var/lib/kubelet/device-plugins

7.8.5 Job

用的比较少

7.8.6 CronJob

用的比较少

7.8.7 Statefulset

7.9 node 资源

7.9.1 污点

1） 内置污点

- node.kubernetes.io/not-ready
  node 不是 ready 状态。对应于 node 的 condition ready=false.
- node.kubernetes.io/unreachable
  node controller 与 node 失联了。对应于 node 的 condition ready=unknown
- node.kubernetes.io/out-of-disk
  node 磁盘空间不足了
- node.kubernetes.io/network-unavailable
  node 的网断了
- node.kubernets.io/unschedulable
  node 不是可调度状态
- node.cloudprovider.kubernetes.io/uninitalized
  kubelet 是由外部云提供商提供的时候，刚开始的时候会打上这个污点来标记还未被使用。当 cloud-controller-manager 控制器初始化完这个 node，kubelet 会自动移除这个污点。

2） 添加污点

语法：
kubectl taint node [node] key=value:[effect]

value：可以为空
[effect] 可取值: [NoSchedule | PreferNoSchedule | NoExecute]

- NoSchedule: 一定不能被调度
- PreferNoSchedule: 尽量不要调度，已有 pod 不会被驱逐
- NoExecute: 不仅不会调度，还会驱逐不能容忍的 Pod

例如：

```
kubectl taint nodes gpu6802 hardware=gpu:NoSchedule
```

3） 查看污点

查看 node 的污点信息

```
kubectl get nodes "-o=custom-columns=NAME:.metadata.name,TAINT:.spec.taints"
```

4） 删除污点

只删除 key 的指定 effect，和添加命令的区别就是，没有 value，并增加的短横线

```
kubectl taint nodes node1 hardware:NoSchedule-
```

删除 key 的所有 effect

```
kubectl taint nodes node1 hardware-
```

5） 容忍污点

在 pod.spec.tolerations/deployment.spec.template.spec 等地方添加内容，如下是容忍 master 节点上的污点

```
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
```

```
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: traefik-ingress-controller
  namespace: kube-system
spec:
  replicas: 1
  template:
    metadata:
      labels:
        k8s-app: traefik-ingress-lb
        name: traefik-ingress-lb
    spec:
      serviceAccountName: traefik-ingress-controller-sa
      nodeSelector:
        node-role.kubernetes.io/master: ""
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
```

其他的匹配方式：

```
tolerations:
```

```
    - key: "key"
      operator: "Equal"
      value: "value"
      effect: NoSchedule
```

```
tolerations:
    - key: "key"
      value: "value"
      effect: NoSchedule
```

operator 的值为 Exists，只要存在 key2，不管 value 是啥

```
- key: "key2"
  operator: "Exists"
  effect: "NoSchedule"
```

容忍所有 NoSchedule，flannel 默认就是这种容忍方式

```
tolerations:
- operator: Exists
  effect: NoSchedule
```

key 也不指定，容忍所有的污点

```
tolerations:
- operator: "Exists"
```

7.9.2 节点选择器--标签

nodeSelect、nodeName 等，是一种强约束，只有满足才调度，都不满足，处于 pending 状态。

1） 添加标签

```
kubectl label node gpu6803 gpu=enable
```

```
kubectl label node master ingress=traefik
```

2） 查看标签

```
kubectl get nodes --show-labels
```

3） k8s 内置的标签：

- kubernetes.io/hostname

- failure-domain.beta.kubernetes.io/zone
- failure-domain.beta.kubernetes.io/region
- beta.kubernetes.io/instance-type
- kubernetes.io/os
- kubernetes.io/arch

4） 删除标签

key 后面加短横线即可

```
kubectl label node node1 disktype-
```

5） 选择标签

在 pod.spec.tolerations/deployment.spec.template.spec 等地方添加内容，如下是选择 master 节点：

```
      nodeSelector:
        node-role.kubernetes.io/master: ""
```

```
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: traefik-ingress-controller
  namespace: kube-system
spec:
  replicas: 1
  template:
    metadata:
      labels:
        k8s-app: traefik-ingress-lb
        name: traefik-ingress-lb
    spec:
      serviceAccountName: traefik-ingress-controller-sa
      nodeSelector:
        node-role.kubernetes.io/master: ""
      tolerations:
```

7.9.3 node 亲和性

一个示例如下：

```
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
```

```
            - matchExpressions:
                - key: beta.kubernetes.io/arch
                  operator: In
                  values:
                    - amd64
                    - ppc64le
                    - s390x
                - key: ingress
                  operator: In
                  values:
                    - istioIngress
        preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 2
            preference:
              matchExpressions:
                - key: beta.kubernetes.io/arch
                  operator: In
                  values:
                    - amd64
          - weight: 2
            preference:
              matchExpressions:
                - key: beta.kubernetes.io/arch
                  operator: In
                  values:
                    - ppc64le
          - weight: 2
            preference:
              matchExpressions:
                - key: beta.kubernetes.io/arch
                  operator: In
                  values:
                    - s390x
```

7.9.4 pod 亲和性

7.9.5 pod 反亲和性

示例如下：

```
    affinity:
      podAntiAffinity: #pod 反亲和性，防止多个 pod 运行在同一个节点上
        requiredDuringSchedulingIgnoredDuringExecution:
```

```
            - labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - media-access
              topologyKey: "kubernetes.io/hostname"
```

7.10 pv/pvc 资源

7.10.1 ceph-rbd 举例

其中 key，是经过 base64 加密的 rbd_key

```
apiVersion: v1
kind: Secret
metadata:
  name: cephrbd-secret
  namespace: gitlab
type: "kubernetes.io/rbd"
data:
  key: QVFENnJiNWNmSXk1R1JBQTRoaTd6N2tOOElydDVLRzRvMHFLd1E9PQo=
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gitlab-pv
  namespace: gitlab
spec:
  capacity:
    storage: 100Gi
  accessModes:
    - ReadWriteOnce
  rbd:
    monitors:
      - 172.16.68.23:6789
      - 172.16.68.24:6789
      - 172.16.68.25:6789
    pool: rbd
    user: rbd
    image: gitlab_k8s
```

```
      fsType: ext4
      secretRef:
        name: cephrbd-secret
      readOnly: false
  persistentVolumeReclaimPolicy: Retain
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: mypvc
  namespace: gitlab
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
  volumeName: gitlab-pv
```

另外各个 k8s 节点要按照 ceph-common，安装方式如下：

```
cat > /etc/yum.repos.d/ceph-mimic.repo << EOF
[ceph-mimic]
name=Ceph mimic packages
baseurl=https://mirrors.aliyun.com/ceph/rpm-mimic/el7/x86_64/
enabled=1
gpgcheck=0
EOF
```

mimic 表示 ceph 的版本

```
yum install ceph-common-13.2.5
```

7.10.2 cephfs 举例

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: xmc2-pv
  namespace: xmc2
spec:
  capacity:
    storage: 2Ti
  accessModes:
    - ReadWriteMany
  cephfs:
    monitors:
      - 172.16.68.23:6789
      - 172.16.68.24:6789
```

```
        - 172.16.68.25:6789
      path: /xmc2
      user: xmc2
      secretRef:
        name: cephfs-xmc2-secret
      readOnly: false
  persistentVolumeReclaimPolicy: Retain
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: mypvc
  namespace: xmc2
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 2Ti
  volumeName: xmc2-pv
```

7.11 StorageClass 资源

使用 nfs 创建 sc，需要修改红色字体部分：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: nfs-client-provisioner
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: nfs-client-provisioner-runner
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
```

```
      verbs: ["get", "list", "watch"]
   - apiGroups: [""]
      resources: ["events"]
      verbs: ["list", "watch", "create", "update", "patch"]
   - apiGroups: [""]
      resources: ["endpoints"]
      verbs: ["create", "delete", "get", "list", "watch", "patch", "update"]

---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
   name: run-nfs-client-provisioner
subjects:
   - kind: ServiceAccount
      name: nfs-client-provisioner
      namespace: default
roleRef:
   kind: ClusterRole
   name: nfs-client-provisioner-runner
   apiGroup: rbac.authorization.k8s.io
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
   name: course-nfs-storage
   #annotations:   创建为默认 sc
    #storageclass.kubernetes.io/is-default-class: true
provisioner: fuseim.pri/ifs
---
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
   name: nfs-client-provisioner
spec:
   replicas: 1
   strategy:
      type: Recreate
   template:
      metadata:
         labels:
            app: nfs-client-provisioner
      spec:
         serviceAccountName: nfs-client-provisioner
```

```
      containers:
        - name: nfs-client-provisioner
          image: quay.io/external_storage/nfs-client-provisioner:latest
          volumeMounts:
            - name: nfs-client-root
              mountPath: /persistentvolumes
          env:
            - name: PROVISIONER_NAME
              value: fuseim.pri/ifs
            - name: NFS_SERVER
              value: 192.168.3.35
            - name: NFS_PATH
              value: /databk/k8s-quzl/nfs-sc
      volumes:
        - name: nfs-client-root
          nfs:
            server: 192.168.3.35
            path: /databk/k8s-quzl/nfs-sc
```

第八章 k8s 进阶

8.1 真正的 0 宕机更新服务

前提:
以下二者都要配合 livenessProbe、readinessProbe 实现。
注意:
对于长连接，目前还没有更加有效的措施实现真正的 0 宕机，通常使用重试解决

8.1.1 优化更新策略

k8s 默认的滚动更新策略是:
maxSurge: 最多可以多出 25%的 pod，即新旧 pod 不能超过 125%，向上取整数，最小是 1
maxUnavailable：最大不可用 pod，即旧版本删除数量，默认 25%，向下取整，最小是 0

```
spec:
  strategy:
    rollingUpdate:
    maxSurge: 25%
    maxUnavailable: 25%
  type: RollingUpdate
template:
```

比如有 4 个 pod，旧版本是 v1、新版本是 v2，更更新过程是
4 个 v1 --> 3 个 v1 + 2 个 v2 --> 2 个 v1,3 个 v2 --> 1 个 v1，4 个 v2 --> 4 个 v2

存在的问题:
如果新版本的 pod 没有起来，将导致可用的 pod 低于目标值
极端情况下，比如只有 1 个副本，如果新 pod 没有正常起来，将导致服务不可用，一个删除中、一个创建中。建议使用如下策略，可保证至少有 4 个 pod 可用(最大不可用为 0)，更新过程变成:

4 个 v1 --> 4 个 v1,1 个创建中的 v2 --> 3 个 v1 + 1 个 v2 + 1 个创建中的 v2 --> 2 个 v1 + 2 个 v2 + 1 个创建中的 v2 --> 1 个 v1 + 3 个 v2 + 1 个创建中的 v2 --> 4 个 v2

```
spec:
  strategy:
    rollingUpdate:
    maxSurge: 1
    maxUnavailable: 0
  type: RollingUpdate
template:
```

8.1.2 添加 lifecycle 段落

添加 lifecycle 段落，具体原因讲起来比较复杂，可参考：
https://www.cnblogs.com/linuxk/p/9578211.html

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: zero-downtime
  labels:
    app: zero-downtime
spec:
  replicas: 3
  selector:
    matchLabels:
      app: zero-downtime
  template:
    spec:
      containers:
      - name: zero-downtime
        image: nginx
        livenessProbe:
          # ...
        readinessProbe:
          # ...
        lifecycle:
          preStop:
            exec:
              command: ["/bin/bash", "-c", "sleep 20"]
```

第九章 Jenkins CI/CD

到 k8s 的 master 节点上

cat /etc/kubernetes/admin.conf

此文件中有三个证书相关的字段，如下三条语句，使用 base64 解码，保存为三个文件

echo {certificate-authority-data} | base64 -d > /tmp/ca.crt

echo {client-certificate-data} | base64 -d > /tmp/client.crt

echo {client-key-data} | base64 -d > /tmp/client.key

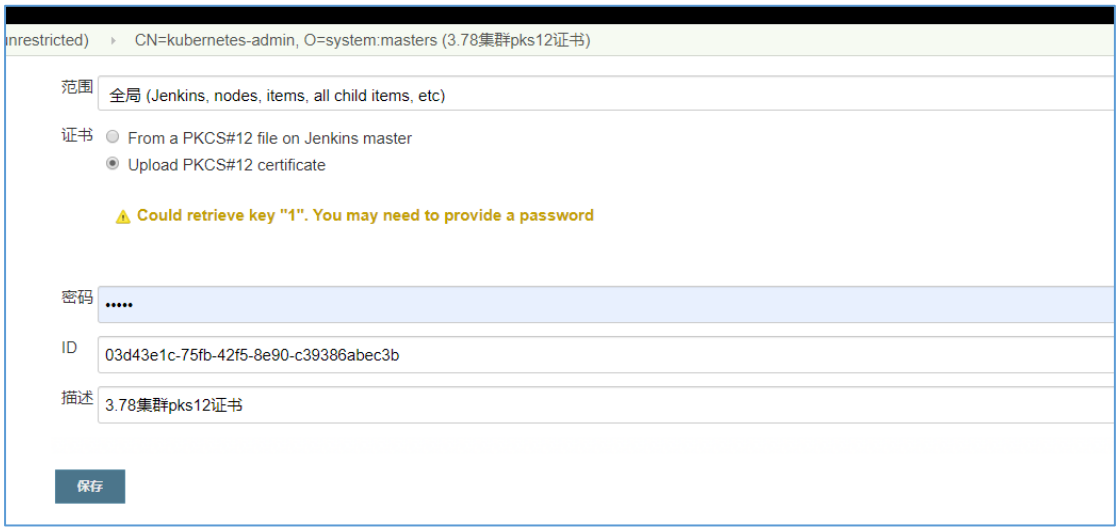
使用 openssl 进行格式转换

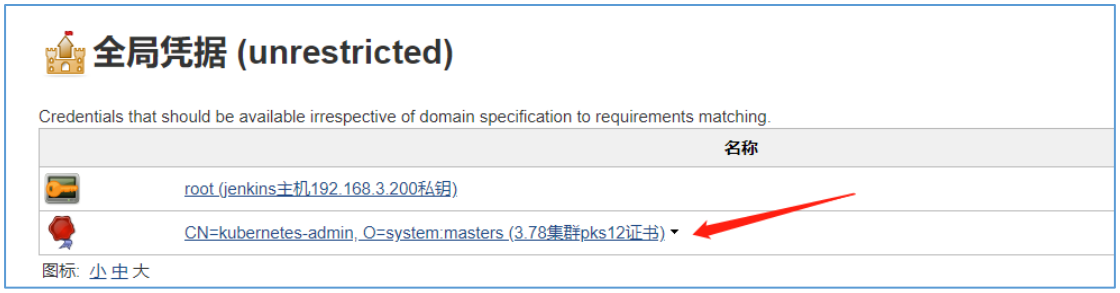
openssl pkcs12 -export -out /tmp/cert.pfx -inkey /tmp/client.key -in /tmp/client.crt -certfile /tmp/ca.crt

需要输入一个密码，建议不要留空。

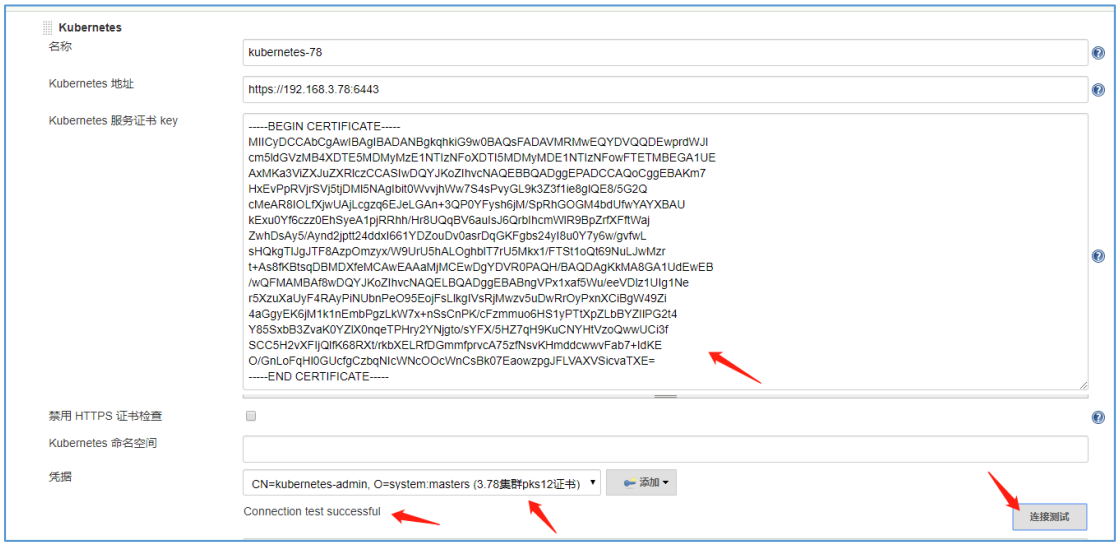
```
[root@master kubernetes]#
[root@master kubernetes]# openssl pkcs12 -export -out /tmp/cert.pfx -inkey /tmp/client.key -in /tmp/client.crt -certfile /tmp/ca.crt
Enter Export Password:
Verifying - Enter Export Password:
[root@master kubernetes]#
[root@master kubernetes]# ls /tmp/
ca.crt  cert.pfx  client.crt  client.key  hsperfdata_root
[root@master kubernetes]#
[root@master kubernetes]#
```

选择证书上传的方式，密码就是生成 pfx 证书时手动输入的密码





服务器证书 key，就是 ca.crt 内容，直接拷贝进去，凭证选择上一步添加的，使用连接测试，显示 success 即是成功添加。


Kubernetes
名称
kubernetes-78
Kubernetes 地址
https://192.168.3.78:6443
Kubernetes 服务证书 key
-----BEGIN CERTIFICATE-----
MIICyDCCAbCgAwIBAgIBADANBgkqhkiG9w0BAQsFADAVMRMwEQYDVQQDEwprdWJl
cm5ldGVzMB4XDTE5MDMyMzE1NTIzNFoXDTI5MDMyMDE1NTIzNFowFTETMBEGA1UE
AxMKa3ViZXJuZXRlczCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAKm7
HxEvPpRVjrSVj5tjDMI5NAgIbit0WvvjhVw7S4sPvyGL9k3Z3f1ie8gIQE8/5G2Q
cMeAR8IOLfXjwUAjLcgzq6EJeLGAn+3QP0YFysh6jM/SpRhGOGM4bdUfwYAYXBAU
kExu0Yf6czz0EhSyeA1pjRRhh/Hr8UQqBV6aulsJ6QrbIhcmWIR9BpZrfXFftWaj
ZwhDsAy5/Aynd2jptt24ddxI661YDZouDv0asrDqGKFgbs24yI8u0Y7y6w/gvfwL
sHQkgTIJgJTF8AzpOmzyx/W9UrU5hALOghbIT7rU5Mkx1/FTSt1oQt69NuLJwMzr
t+As8fKBtsqDBMDXfeMCAwEAAaMjMCEwDgYDVR0PAQH/BAQDAgKkMA8GA1UdEwEB
/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEBABngVPx1xaf5Wu/eeVDlz1UIg1Ne
r5XzuXaUyF4RAyPiNUbnPeO95EojFsLlkgIVsRjMwzv5uDwRrOyPxnXCiBgW49Zi
4aGgyEK6jM1k1nEmbPgzLkW7x+nSsCnPK/cFzmmuo6HS1yPTtXpZLbBYZIIPG2t4
Y85SxbB3ZvaK0YZIX0nqeTPHry2YNjgto/sYFX/5HZ7qH9KuCNYHtVzoQwwUCi3f
SCC5H2vXFIjQlfK68RXt/rkbXELRfDGmmfprvcA75zfNsvKHmddcwwvFab7+IdKE
O/GnLoFqHI0GUcfgCzbqNIcWNcOOcWnCsBk07EaowzpgJFLVAXVSicvaTXE=
-----END CERTIFICATE-----
禁用 HTTPS 证书检查
Kubernetes 命名空间
凭据
CN=kubernetes-admin, O=system:masters (3.78集群pks12证书)
添加
Connection test successful
连接测试