部署说明

目录

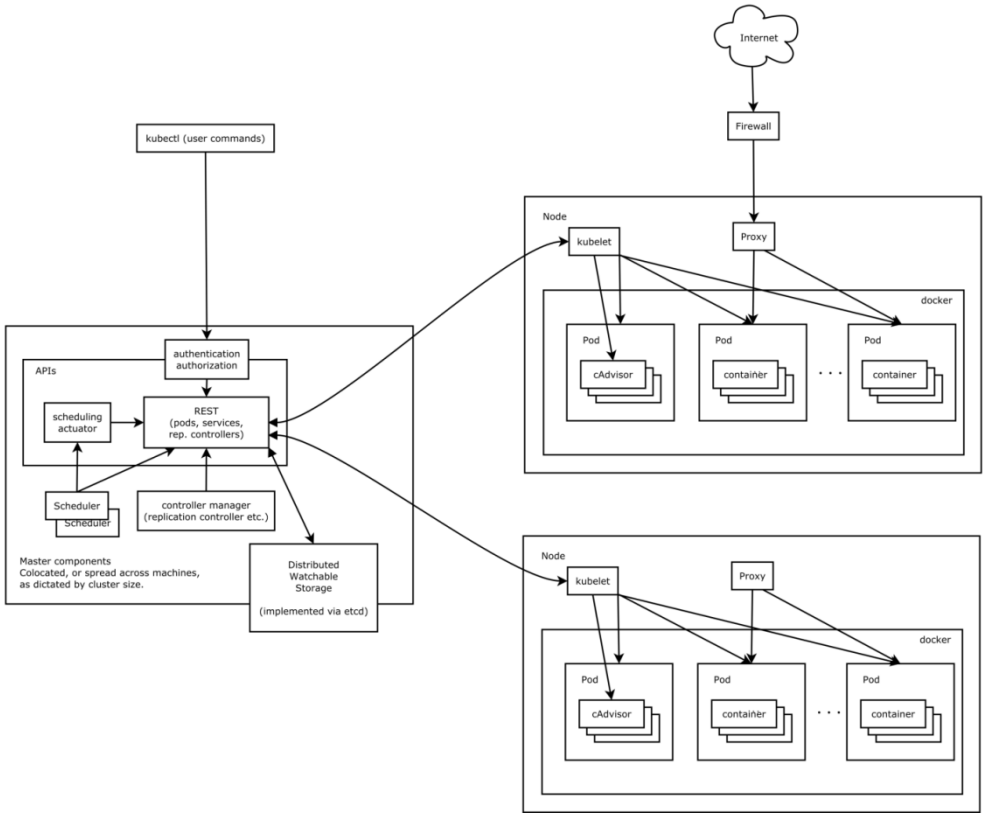
阅读说明：

1. 凡是加了灰底的区域，为操作指令输入内容，例如： ls kube*
2. 斜体字为补充的细节说明，例如：*参考链接、额外说明等*
3. 暗红色的字体，一般为错误消息，例如：命令结果错误、k8s 运行反馈错误等
4. 黄底标注的内容，为容易出错/忽略的细节，例如：提示、警告

K8S 简介

K8S 是 Kubernetes 的简称（以下全部使用简称），它是一个开源的，用于管理云平台中多个主机上的容器化的应用，它是大规模容器集群管理系统。k8s 的目标是让部署容器化的应用简单并且高效(powerful)，它提供了应用部署，规划，更新，维护的一种机制。K8S 一个核心的特点就是能够自主的管理容器来保证云平台中的容器按照用户的期望状态运行着（比如用户想让 apache 一直运行，用户不需要关心怎么去做，它会自动去监控，然后去重启、新建），k8s 也的提供了一系列管理工具，让用户能够方便的部署自己的应用。

K8S 对计算资源进行了更高层次的抽象，通过将容器进行细致的组合，将最终的应用服务交给用户。在 k8s 中所有的容器均在 Pod 中运行,一个 Pod 可以承载一个或者多个相关的容器,同一个 Pod 中的容器会部署在同一个物理机器上并且能够共享资源。一个 Pod 也可以包含零个或多个磁盘卷组（volumes）,这些卷组将会以目录的形式提供给一个容器，或者被所有 Pod 中的容器共享。对于用户创建的每个 Pod,系统会自动选择那个健康并且有足够容量的机器，然后创建类似容器的容器。当容器创建失败的时候，容器会被 node agent 自动的重启，这个 node agent 叫 **kubelet**。但是如果 Pod 失败或者机器，它不会自动的转移并且启动，除非用户定义了 replication controller(下图为 k8s 架构图)。



有关 k8s 的文档，可以访问 https://www.kubernetes.org.cn/k8s

预备工作

本文是基于 **k8s 1.15.3** 版本，通过 kubeadm 来完成所有集群安装，要求系统的内核版本不低于 3.10

```
[root@VMLinux ~]# uname -r
3.10.0-957.27.2.el7.x86_64
```

如果内核版本过低，将导致容器服务（docker）安装失败，本文所用的 OS 是 **CentOS-7.6.1810-x64**。

由于我们是要安装集群，所在至少要预备出两台机器，为了演示方便，我们预备两台 VM。通过网络桥接的方式，将两个 VM 接入局域网中，信息如下：

名称	IP	角色
vm-linux7(M)	192.168.31.194	Master node
vm-linux7(S)	192.168.31.195	Slave node

两台 VM 机器都是纯净的新 OS，我们先执行以下操作(以下所有操作，在每个 node 上都要执行，包括将来新加入 node，也要按这个说明处理)。

第一步：修改 host

修改每台服务器的 hostname:

`vi /etc/hostname`

主服务器上填 `master`

节点服务器上填 `node1`

修改每台服务器的 hosts

```
vi /etc/hosts
192.168.31.194 master
192.168.31.195 node1
```

重启计算机，使设置生效

`reboot -h NOW`

第二步：关闭 SELinux

先查看 SELinux 状态

sestatus

```
[root@VMLinux ~]# sestatus
SELinux status:                 disabled
```

如是要显示不是 disabled，则要修改/etc/selinux/config，使 SELINUX=enforcing 改为 disabled，如下:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of three values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected processes are protected.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

第三步：禁用交换分区

类似 ElasticSearch 集群一样，k8s 集群也需要关闭 swap 机制，这是因为 swap 会影响它的性能与稳定性。

查看一下 swap 分区状态，如果 swap 数值全为 0，则表示已禁用:

`free -h`

```
[root@VMLinux ~]# free -h
              total        used        free      shared  buff/cache   available
Mem:           1.8G        612M        106M         89M        1.1G        879M
Swap:          2.0G        4.3M        2.0G
```

如果没有全为 0，则需要关闭。通过 `swapoff -a` 可以临时关闭，但系统重启后会自动恢复，通过修改配置文件可永久关闭。编辑/etc/fstab，注释掉包含 swap 的那一行即可

```
#
# /etc/fstab
# Created by anaconda on Fri Sep  6 15:10:29 2019
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
/dev/mapper/centos-root /                       xfs     defaults        0 0
UUID=ffb89a8f-0446-4028-9673-3f2d5c3025b7 /boot                   xfs     defaults        0 0
/dev/mapper/centos-home /home                   xfs     defaults        0 0
#/dev/mapper/centos-swap swap                    swap    defaults        0 0
```

第四步：更新 yum

在后续的操作中，要经常使用 yum 安装程序包，所以要对 yum 作一次全面更新。

```
yum install -y epel-release
yum update
```

然后安装所需的软件包 yum-util，因为这个包提供了 yum-config-manager 工具：

```
yum install -y yum-utils
```

第五步：安装 docker

每台 node 都需要安装 docker，如果之前安装过 docker，需要先卸载旧版本。

```
yum list installed|grep docker
```

```
[root@VMLinux ~]# yum list installed|grep docker
docker.x86_64                       2:1.13.1-102.git7f2769b.el7.centos @extras
docker-client.x86_64                2:1.13.1-102.git7f2769b.el7.centos @extras
docker-common.x86_64                2:1.13.1-102.git7f2769b.el7.centos @extras
```

然后删除旧版本 docker

```
yum -y remove docker.x86_64 docker-client.x86_64 docker-common.x86_64
```

删除镜像/容器等

```
rm -rf /var/lib/docker
rm -rf /var/run/docker
```

在安装 docker 时，由于国内 GFW 的原因，直接安装 docker 可能会失败，所以一般都要添加国内镜像源：

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

然后执行一次清除缓存的操作，防止缓存中的版本冲突

```
yum clean all
```

在安装之前，可以看看线上所有的 docker 版本，例如

```
yum list docker-ce --showduplicates|sort -r
```

```
[root@VMLinux ~]# yum list docker-ce --showduplicates|sort -r
已加载插件：fastestmirror
可安装的软件包
 * updates: mirrors.zju.edu.cn
 * extras: mirror.bit.edu.cn
 * epel: hkg.mirror.rackspace.com
docker-ce.x86_64            3:19.03.2-3.el7                     docker-ce-stable
docker-ce.x86_64            3:19.03.1-3.el7                     docker-ce-stable
docker-ce.x86_64            3:19.03.0-3.el7                     docker-ce-stable
docker-ce.x86_64            3:18.09.9-3.el7                     docker-ce-stable
docker-ce.x86_64            3:18.09.8-3.el7                     docker-ce-stable
docker-ce.x86_64            3:18.09.7-3.el7                     docker-ce-stable
docker-ce.x86_64            3:18.09.6-3.el7                     docker-ce-stable
docker-ce.x86_64            3:18.09.5-3.el7                     docker-ce-stable
docker-ce.x86_64            3:18.09.4-3.el7                     docker-ce-stable
docker-ce.x86_64            3:18.09.3-3.el7                     docker-ce-stable
docker-ce.x86_64            3:18.09.2-3.el7                     docker-ce-stable
docker-ce.x86_64            3:18.09.1-3.el7                     docker-ce-stable
docker-ce.x86_64            3:18.09.0-3.el7                     docker-ce-stable
docker-ce.x86_64            18.06.3.ce-3.el7                    docker-ce-stable
docker-ce.x86_64            18.06.2.ce-3.el7                    docker-ce-stable
docker-ce.x86_64            18.06.1.ce-3.el7                    docker-ce-stable
docker-ce.x86_64            18.06.0.ce-3.el7                    docker-ce-stable
docker-ce.x86_64            18.03.1.ce-1.el7.centos             docker-ce-stable
docker-ce.x86_64            18.03.0.ce-1.el7.centos             docker-ce-stable
docker-ce.x86_64            17.12.1.ce-1.el7.centos             docker-ce-stable
docker-ce.x86_64            17.12.0.ce-1.el7.centos             docker-ce-stable
docker-ce.x86_64            17.09.1.ce-1.el7.centos             docker-ce-stable
docker-ce.x86_64            17.09.0.ce-1.el7.centos             docker-ce-stable
docker-ce.x86_64            17.06.2.ce-1.el7.centos             docker-ce-stable
docker-ce.x86_64            17.06.1.ce-1.el7.centos             docker-ce-stable
docker-ce.x86_64            17.06.0.ce-1.el7.centos             docker-ce-stable
docker-ce.x86_64            17.03.3.ce-1.el7                    docker-ce-stable
docker-ce.x86_64            17.03.2.ce-1.el7.centos             docker-ce-stable
docker-ce.x86_64            17.03.1.ce-1.el7.centos             docker-ce-stable
docker-ce.x86_64            17.03.0.ce-1.el7.centos             docker-ce-stable
Determining fastest mirrors
 * base: ftp.sjtu.edu.cn
```

用以下命令安装最新版本的 docker

```
sudo yum install docker-ce docker-ce-cli containerd.io  #默认安装最新版
```

或者要选择安装某个版本：

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

docker 安装完成后，使它以服务进程的方式启动

```
systemctl enable docker && systemctl start docker
```

接下来验证是否成功

```
docker version
```

```
[root@VMLinux ~]# docker version
Client: Docker Engine - Community
 Version:           19.03.2
 API version:       1.40
 Go version:        go1.12.8
 Git commit:        6a30dfc
 Built:             Thu Aug 29 05:28:55 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.2
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.8
  Git commit:       6a30dfc
  Built:            Thu Aug 29 05:27:34 2019
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.2.6
  GitCommit:        894b81a4b802e4eb2a91d1ce216b8817763c29fb
 runc:
  Version:          1.0.0-rc8
  GitCommit:        425e105d5a03fabd737a126ad93d62a9eeede87f
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
```

有 client 与 server 两部分的版本显示时，表示 docker 安装并启动成功。在后面拉取 docker 镜像时，从国外拉取会特别慢，所以要添加国内的 docker 镜像源。通过修改 daemon 配置文件 /etc/docker/daemon.json 来使用加速器

```
mkdir -p /etc/docker
cat <<EOF > /etc/docker/daemon.json
{
   "registry-mirrors": ["https://qqnn8qm9.mirror.aliyuncs.com"]
 }
EOF
```

然后重启 docker，使配置生效：

```
systemctl daemon-reload
systemctl restart docker
```

通过 docker run hello-world 验证是否成功，如果看到以内结果就表成 docker 环境全部完成了：

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:b8ba256769a0ac28dd126d584e0a2011cd2877f3f76e093a7ae560f2a5301c00
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

第六步：时间核对

所有 node 机器，都按以下指令核对时间：

```
yum install ntp
systemctl enable ntpd && systemctl start ntpd
ntpdate ntp1.aliyun.com
hwclock -w
```

第七步：预备国内的 k8s 源

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=http://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=0
repo_gpgcheck=0
gpgkey=http://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg
       http://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg
EOF
```

第八步：启用路由转发功能

第一次安装 k8s 时，必须打开路由转发功能，很多文章都没有写：

```
echo "1" > /proc/sys/net/ipv4/ip_forward
```

然后配置内核参数，将桥接的 ipv4 流量传递到 iptables 的链，即防止应用部署在 centOS 系统下，由于 iptables 被绕过而导致的路由错误。先调用命令加载内核模块

```
sudo modprobe br_netfilter && sudo modprobe ip_vs
```

然后设置参数

```
cat > /etc/sysctl.d/k8s.conf <<EOF
net.bridge.bridge-nf-call-ip6tables=1
net.bridge.bridge-nf-call-iptables=1
net.ipv4.ip_forward=1
net.ip4.tcp_tw_recycle=0
net.ipv6.conf.all.disable_ipv6=1
EOF
```

上面代码关闭 tcp_tw_recycle（它会与 k8s 的 NAT 冲突）和 IPV6 协议栈（防止触发 docker BUG），然后调用

```
sysctl --system
```

使配置生效，确认并保证的输出都是 1

```
cat /proc/sys/net/bridge/bridge-nf-call-ip6tables
cat /proc/sys/net/bridge/bridge-nf-call-iptables
```

第九步(可选)：启用 ipvs 功能

如果搭建的 k8s 集群使用 ipvs 实现网络路由（默认为 iptables），则需要按本节处理，否则可以跳过。由于 ipvs 已加入到了内核主干，所以要为 kube-proxy 开启 ipvs 的前提需要加载以内核模块：

```
ipvs、ip_vs_rr、ip_vs_wrr、ip_vs_sh、nf_conntrack_ipv4
```

在所有的节点机器上创建以下脚本文件

```
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
```

接下来执行脚本

```
chmod 755 /etc/sysconfig/modules/ipvs.modules
bash /etc/sysconfig/modules/ipvs.modules
lsmod | grep -e ip_vs -e nf_conntrack_ipv4
```

上面脚本创建了的/etc/sysconfig/modules/ipvs.modules 文件，保证在节点重启后能自动加载所需模块。

使用 `lsmod | grep -e ip_vs -e nf_conntrack_ipv4` 命令查看是否已经正确加载所需的内核模块。接

下来还要在各节点安装 ipset 包

```
yum install -y ipset
yum install -y ipvsadm
```

如果以上前提条件如果不满足，则即使 kube-proxy 的配置开启了 ipvs 模式，也会退回到 iptables 模式。

接下来还要调整配置文件

```
kubectl edit cm kube-proxy -n kube-system
```

在打开的编辑界面中，找到以下段落，然后将 mode 改为 ipvs

```
minSyncPeriod: 0s
      scheduler: ""
      syncPeriod: 30s
    kind: KubeProxyConfiguration
    metricsBindAddress: 127.0.0.1:10249
    mode: "ipvs"
    nodePortAddresses: null
```

默认情况下，mode 为空值（即 iptables 模式），改为 ipvs；scheduler 默认也是空（负载均衡算法为轮询）。

提示：对于已经运行 k8s 的各个节点，若要由 iptables 改为 ipvs，除了上面操作外，还要删除所有 kube-proxy

的 pod，再重启各节点的 kube-proxy pod：

```
kubectl delete pod xxx -n kube-system
kubectl get pod -n kube-system|grep kube-proxy|awk '{system("kubectl delete pod "$1" -n kube-system")}'
```

到此为止，所有的基本预备工作完成，重启计算机后，开始分别在各个 node 上做操作。

Master 安装 K8S

第一步：安装 k8s 组件

```
yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
```

```
已加载插件：fastestmirror
Loading mirror speeds from cached hostfile
 * base: ftp.sjtu.edu.cn
 * epel: hkg.mirror.rackspace.com
 * extras: mirror.bit.edu.cn
 * updates: mirrors.zju.edu.cn
kubernetes                                                              | 1.4 kB  00:00:00
正在解决依赖关系
--> 正在检查事务
---> 软件包 kubeadm.x86_64.0.1.15.3-0 将被 安装
--> 正在处理依赖关系 kubernetes-cni >= 0.7.5，它被软件包 kubeadm-1.15.3-0.x86_64 需要
--> 正在处理依赖关系 cri-tools >= 1.13.0，它被软件包 kubeadm-1.15.3-0.x86_64 需要
---> 软件包 kubectl.x86_64.0.1.15.3-0 将被 安装
---> 软件包 kubelet.x86_64.0.1.15.3-0 将被 安装
--> 正在检查事务
---> 软件包 cri-tools.x86_64.0.1.13.0-0 将被 安装
---> 软件包 kubernetes-cni.x86_64.0.0.7.5-0 将被 安装
--> 解决依赖关系完成

依赖关系解决

================================================================================================
 Package              架构              版本                 源                  大小
================================================================================================
正在安装:
 kubeadm              x86_64            1.15.3-0             kubernetes          8.9 M
 kubectl              x86_64            1.15.3-0             kubernetes          9.5 M
 kubelet              x86_64            1.15.3-0             kubernetes           22 M
为依赖而安装:
 cri-tools            x86_64            1.13.0-0             kubernetes          5.1 M
 kubernetes-cni       x86_64            0.7.5-0              kubernetes           10 M

事务概要
================================================================================================
安装  3 软件包 (+2 依赖软件包)

总下载量：55 M
安装大小：250 M
```

等待所有依赖项安装完成后，启动 Kubelet 服务

```
systemctl enable kubelet && systemctl start kubelet
```

```
[root@VMLinux ~]# systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; vendor preset: disabled)
  Drop-In: /usr/lib/systemd/system/kubelet.service.d
           └─ 10-kubeadm.conf
   Active: activating (auto-restart) (Result: exit-code) since 四 2019-09-12 16:07:08 CST; 856ms ago
     Docs: https://kubernetes.io/docs/
  Process: 11429 ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_KUBEADM_ARGS $KUBELET_EXTRA_ARGS (code=exited, status=255)
 Main PID: 11429 (code=exited, status=255)

9月 12 16:07:08 VMLinux systemd[1]: Unit kubelet.service entered failed state.
9月 12 16:07:08 VMLinux systemd[1]: kubelet.service failed.
```

可以看到，由于 k8s 未初化，所以 kubelet 不能启动，接下来我们初始化 master node。

通过 `kubectl version` 可查看版本信息 1.15.3

```
[root@VMLinux ~]# kubectl version
Client Version: version.Info{Major:"1", Minor:"15", GitVersion:"v1.15.3", GitCommit:"2d3c76f9091b6bec110a5e63777c332469e0cba2", GitTreeState:"clean", BuildDate:"2019-0
8-19T11:13:54Z", GoVersion:"go1.12.9", Compiler:"gc", Platform:"linux/amd64"}
The connection to the server localhost:8080 was refused - did you specify the right host or port?
```

第二步：初始化 master

方法一

```
kubeadm init --kubernetes-version=1.15.3 --apiserver-advertise-address=192.168.31.194
--image-repository registry.aliyuncs.com/google_containers --service-cidr=10.1.0.0/16
```

`--pod-network-cidr=10.244.0.0/16`

解释如下

1.15.3 是 kubectl 的版本
--image-repository 是指镜像仓库地址，默认是国外仓库，但由于 GFW 会下载失败
--apiserver-advertise-address 标红的 IP 地址填写本 master node 的 IP 地址
--pod-network-cidr、--service-cidr 设置 pod 的子网地址和 service 的子网地址

正常执行完成后，会提示以下信息

```
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane 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/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.31.194:6443 --token 0qxe73.mysqzzr2qd6mhkv8 \
    --discovery-token-ca-cert-hash sha256:72f0fb181707c308da87c713975580f1f9a5c3afeabde43d455b9035e72884bb
```

这说明集群初始化成功了，要记住最后一段代码，以后加入集群要经常使用它：

```
kubeadm join 192.168.31.194:6443 --token 0qxe73.mysqzzr2qd6mhkv8 \
    --discovery-token-ca-cert-hash
sha256:72f0fb181707c308da87c713975580f1f9a5c3afeabde43d455b9035e72884bb
```

为了安全性，我没有关闭防火墙，因此它提示防火墙正在工作中，需要开放 6443、10250 端口，那我们就

开放这两个端口即可：

```
[root@master ~]# kubeadm init --kubernetes-version=1.15.3 --apiserver-advertise-address=192.168.31.194 --image-repository registry.ali
vice-cidr=10.1.0.0/16 --pod-network-cidr=10.244.0.0/16
[init] Using Kubernetes version: v1.15.3
[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 IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended driver is "systemd". Please f
etes.io/docs/setup/cri/
        [WARNING SystemVerification]: this Docker version is not on the list of validated versions: 19.03.2. Latest validated version:
        [WARNING Hostname]: hostname "master" could not be reached
```

```
firewall-cmd --permanent --zone=public --add-port=6443/tcp
firewall-cmd --permanent --zone=public --add-port=10250/tcp
firewall-cmd --reload
```

方法二

如果网络正常，使用方法一应该就可以操作成功了；但是有时会遇到国内镜像不能下载成功，此时需要手动

用国内镜像安装（例如使用阿里云提供的 docker hub 进行安装）。首先输出 kubeadm 的默认配置

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

将其中 `imageRepository` 修改为

`registry.cnhangzhou.aliyuncs.com/google_containers`;serviceSubnet 部分设置为 *`10.244.0.0/16`*，也就是方法一中 pod-network-cidr= *10.244.0.0/16* 的参数，再将 advertiseAddress:1.2.3.4 改为 `192.168.31.194`（即本机 IP 地址）。

接下来进行镜像下载：

```
kubeadm config images pull --config kubeadm-init.yaml
```

最后使用修改后的配置进行初始化：

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

初始化失败处理

如果使用 kubeadm 初始化集群失败，或启动过程中卡在以下位置

[apiclient] Created API client, waiting for the control plane to become ready

可以使用 journalctl -t kubelet -S 'xxxxxxx'查看日志，发现如下错误

error: failed to run Kubelet: failed to create kubelet: misconfiguration: kubelet cgroup driver: "systemd"

这说明 cgroup-driver 参数与 docker 的不一致，需要修改 KUBE_CGROUP_ARGS="--cgroup-driver=systemd"为 KUBE_CGROUP_ARGS="--cgroup-driver= cgroupfs"，即：

```
vi /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
#Environment="KUBELET_CGROUP_ARGS=--cgroup-driver=systemd"
Environment="KUBELET_CGROUP_ARGS=--cgroup-driver=cgroupfs"

systemctl daemon-reload && systemctl restart kubelet
```

小结

不管是方法一还是方法二，如果操作失败，可以执行 `kubeadm reset` 重置，然后重新从第一步开始初始化。

输入 `kubectl get cs` 可以检测组件运行是否正常

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

输入 `kubectl get no` 可以查看集群节点

```
[root@master ~]# kubectl get no
The connection to the server localhost:8080 was refused - did you specify the right host or port?
```

如果出现上面提示，是因为 kubectl 这个命令需要使用 Kubernetes-admin 来运行，我们可以将主节点中

的`/etc/kubernetes/admin.conf` 复制到从节点机器相同目录下，然后配置环境变量；如果本身就是在

master node 上，则直接配置环境变量即可：

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

接着再运行 `kubectl get no`

```
[root@master ~]# kubectl get no
NAME     STATUS     ROLES    AGE   VERSION
master   NotReady   master   11m   v1.15.3
```

这里显示 NotReady 是因为还没有配置集群网络（pod network）。

第三步：安装 flannel

下载yaml文件(这个kube-flannel.yml文件里的flannel的镜像是0.11.0【`quay.io/coreos/flannel:v0.11.0-amd64`】)

```
curl -O
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

然后执行以下命令

```
kubectl create -f kube-flannel.yml
```

```
[root@master ~]# kubectl create -f kube-flannel.yml
podsecuritypolicy.policy/psp.flannel.unprivileged created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds-amd64 created
daemonset.apps/kube-flannel-ds-arm64 created
daemonset.apps/kube-flannel-ds-arm created
daemonset.apps/kube-flannel-ds-ppc64le created
daemonset.apps/kube-flannel-ds-s390x created
```

然后重启 kubelet 服务 `systemctl restart kubelet`，接下来查看节点状态

```
kubectl get nodes
```

```
[root@master ~]# kubectl get nodes
NAME     STATUS   ROLES    AGE   VERSION
master   Ready    master   42m   v1.15.3
```

查看 k8s 系统级的 pod 状态：`kubectl get pods -n kube-system`

```
[root@master ~]# kubectl get pods -n kube-system
NAME                              READY   STATUS    RESTARTS   AGE
coredns-bccdc95cf-755bv           1/1     Running   0          43m
coredns-bccdc95cf-zbw7d           1/1     Running   0          43m
etcd-master                       1/1     Running   0          42m
kube-apiserver-master             1/1     Running   0          43m
kube-controller-manager-master    1/1     Running   0          42m
kube-flannel-ds-amd64-hrpx8       1/1     Running   0          2m58s
kube-proxy-kpjdn                  1/1     Running   0          43m
kube-scheduler-master             1/1     Running   0          42m
```

也可以查看更详细的 pod 状态 `kubectl get pod -o wide -n kube-system`

```
[root@master ~]# kubectl get pod -o wide -n kube-system
NAME                              READY   STATUS    RESTARTS   AGE     IP              NODE     NOMINATED NODE   READINESS GATES
coredns-bccdc95cf-755bv           1/1     Running   0          44m     10.244.0.2      master   <none>           <none>
coredns-bccdc95cf-zbw7d           1/1     Running   0          44m     10.244.0.3      master   <none>           <none>
etcd-master                       1/1     Running   0          43m     192.168.31.194  master   <none>           <none>
kube-apiserver-master             1/1     Running   0          43m     192.168.31.194  master   <none>           <none>
kube-controller-manager-master    1/1     Running   0          43m     192.168.31.194  master   <none>           <none>
kube-flannel-ds-amd64-hrpx8       1/1     Running   0          3m52s   192.168.31.194  master   <none>           <none>
kube-proxy-kpjdn                  1/1     Running   0          44m     192.168.31.194  master   <none>           <none>
kube-scheduler-master             1/1     Running   0          43m     192.168.31.194  master   <none>           <none>
```

至此，Master 配置完成。

集群扩展

加入集群(Slave)

所有的 slave node，都是 k8s 的实际工作节点，将分担所有 pod 的计算任务。现在我们要把之前预备的机器 node1，IP 为 192.168.31.195，参照《预备工作》做好环境准备后，开始在 node1 上安装 k8s 组件

`yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes`

```
已加载插件：fastestmirror
Loading mirror speeds from cached hostfile
 * base: ftp.sjtu.edu.cn
 * epel: hkg.mirror.rackspace.com
 * extras: mirror.bit.edu.cn
 * updates: mirrors.zju.edu.cn
kubernetes                                                                                                                                                | 1.4 kB  00:00:00
正在解决依赖关系
--> 正在检查事务
---> 软件包 kubeadm.x86_64.0.1.15.3-0 将被 安装
--> 正在处理依赖关系 kubernetes-cni >= 0.7.5，它被软件包 kubeadm-1.15.3-0.x86_64 需要
--> 正在处理依赖关系 cri-tools >= 1.13.0，它被软件包 kubeadm-1.15.3-0.x86_64 需要
---> 软件包 kubectl.x86_64.0.1.15.3-0 将被 安装
---> 软件包 kubelet.x86_64.0.1.15.3-0 将被 安装
--> 正在检查事务
---> 软件包 cri-tools.x86_64.0.1.13.0-0 将被 安装
---> 软件包 kubernetes-cni.x86_64.0.0.7.5-0 将被 安装
--> 解决依赖关系完成

依赖关系解决

================================================================================================================================================================
 Package                                架构                                版本                                源                                大小
================================================================================================================================================================
正在安装:
 kubeadm                                x86_64                              1.15.3-0                            kubernetes                        8.9 M
 kubectl                                x86_64                              1.15.3-0                            kubernetes                        9.5 M
 kubelet                                x86_64                              1.15.3-0                            kubernetes                         22 M
为依赖而安装:
 cri-tools                              x86_64                              1.13.0-0                            kubernetes                        5.1 M
 kubernetes-cni                         x86_64                              0.7.5-0                             kubernetes                         10 M

事务概要
================================================================================================================================================================
安装  3 软件包 (+2 依赖软件包)

总下载量：55 M
安装大小：250 M
```

等待所有依赖项安装完成后，**不要启动 Kubelet 服务**!！接下来执行在 master node 成功后，最后一句的 join 指令，join 指令会自动管理 kubelet 服务：

```
kubeadm join 192.168.31.194:6443 --token 0qxe73.mysqzzr2qd6mhkv8 \
    --discovery-token-ca-cert-hash
sha256:72f0fb181707c308da87c713975580f1f9a5c3afeabde43d455b9035e72884bb
```

```
[root@node1 ~]# kubeadm join 192.168.31.194:6443 --token 0qxe73.mysqzzr2qd6mhkv8     --discovery-token-ca-cert-hash sha256:72f0fb181707c308da87c713975580f1f9a5c3afeabd
e43d455b9035e72884bb
[preflight] Running pre-flight checks
        [WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended driver is "systemd". Please follow the guide at https://kubern
etes.io/docs/setup/cri/
        [WARNING SystemVerification]: this Docker version is not on the list of validated versions: 19.03.2. Latest validated version: 18.09
        [WARNING Hostname]: hostname "node1" could not be reached
        [WARNING Hostname]: hostname "node1": lookup node1 on 192.168.31.1:53: no such host
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.15" 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
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

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 control-plane to see this node join the cluster.
```

看到上述结果时，说明这个 salve node 已经加入集群了！在 master node 上执行

```
kubectl get nodes
```

```
[root@master ~]# kubectl get no
NAME     STATUS     ROLES    AGE     VERSION
master   Ready      master   65m     v1.15.3
node1    NotReady   <none>   2m33s   v1.15.3
```

可以看到 node1 已在集群中，但是它的状态是 NotReady 状态，要等待一会，然后再执行即可正常：

```
[root@node1 ~]# kubectl get no
NAME      STATUS   ROLES    AGE     VERSION
master    Ready    master   70m     v1.15.3
node1     Ready    <none>   7m19s   v1.15.3
```

至此，一个 slave node 加入集群成功。有时候使用加入集群命令时，会提示错误

error execution phase preflight: couldn't validate the identity of the API Server: abort connecting to API servers after timeout of 5m0s
 # kubeadm join ……
error execution phase preflight: couldn't validate the identity of the API Server: abort connecting to API servers after timeout of 5m0

这是因为 master 节点给的 token 过期了，需要重新生成一次 token：

```
kubeadm token create
openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/null | openssl dgst -sha256 -hex | sed 's/^.* //'
```

执行后，会有一串新 token 显示出来，使用 kubeadm join 时，用新 token 即可。

离开集群

（提示：本节内容未经过亲自测试，是在网上摘抄而来）

若要从集群中移除某个 node，例如 slave3，可执行以下命令

```
kubectl get node
```

```
NAME               STATUS   ROLES    AGE
master.hanli.com   Ready    master   3d7h
slave1.hanli.com   Ready    <none>   3d7h
slave2.hanli.com   Ready    <none>   3d7h
slave3.hanli.com   Ready    <none>   3d7h
```

可以看到有三个 slave 节点一个 master 节点，再查看一下各 pod 的情况

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

```
NAME                     READY   STATUS    RESTARTS   AGE     IP           NODE
curl-66959f6557-r4crd    1/1     Running   1          6m32s   10.244.2.7   slave2.h
nginx-58db6fdb58-5wt7p   1/1     Running   0          3d6h    10.244.1.4   slave1.h
nginx-58db6fdb58-7qkfn   1/1     Running   0          3d6h    10.244.3.2   slave3.h
```

接下来我们要封锁要移除的 node(将节点标记为不可调度)

```
kubectl cordon node/slave3.hanli.com
```

驱逐这个 node 上的所有 pod，使这些 pod 全部平滑的转到其它 node 上

```
kubectl drain slave3.hanli.com --delete-local-data --force --ignore-daemonsets node/slave3.hanli.com
```

```
WARNING: Ignoring DaemonSet-managed pods: kube-flannel-ds-amd64-8hhsb, kube-proxy-6
pod/monitoring-grafana-8445c4b56d-j2wfl evicted
pod/nginx-58db6fdb58-7qkfn evicted
node/slave3.hanli.com evicted
```

此时再查看 node 状态显示如下

```
NAME              STATUS                     ROLES    AGE
master1.hanli.com   Ready                    master   17h
master2.hanli.com   Ready                    master   17h
master3.hanli.com   Ready                    master   16h
slave3.hanli.com    Ready,SchedulingDisabled   <none>   16h
```

然后就可以移除 slave3 节点了

```
kubectl delete node slave3.hanli.com
```

```
node "slave3.hanli.com" deleted
```

再次查看 node 信息，已经没有这个节点了，而再次查看 pod 信息时，会发现原来属于 slave3 的 pod 已经被调到其它 node 上了，例如：

```
NAME                    READY   STATUS    RESTARTS   AGE     IP           NODE
curl-66959f6557-r4crd   1/1     Running   1          8m34s   10.244.2.7   slave2.h
nginx-58db6fdb58-5wt7p  1/1     Running   0          3d6h    10.244.1.4   slave1.h
nginx-58db6fdb58-bhmcv  1/1     Running   0          55s     10.244.2.8   slave2.h
```

此时，还需要 slave3 节点执行后续操作，彻底删除节点资源

```
kubeadm reset
ifconfig cni0 down
ip link delete cni0
ifconfig flannel.1 down
ip link delete flannel.1
rm -rf /var/lib/cni/
```

如是要只是想维护 slave3 节点，则不要执行 delete 指令，在对 slave3 节点做完维护（如升级内核等），可以使用以下指令使 slave3 节点重回集群：

```
kubectl uncordon slave3.hanli.com
```

该指令执行后，这个节点会恢复可调度状态。

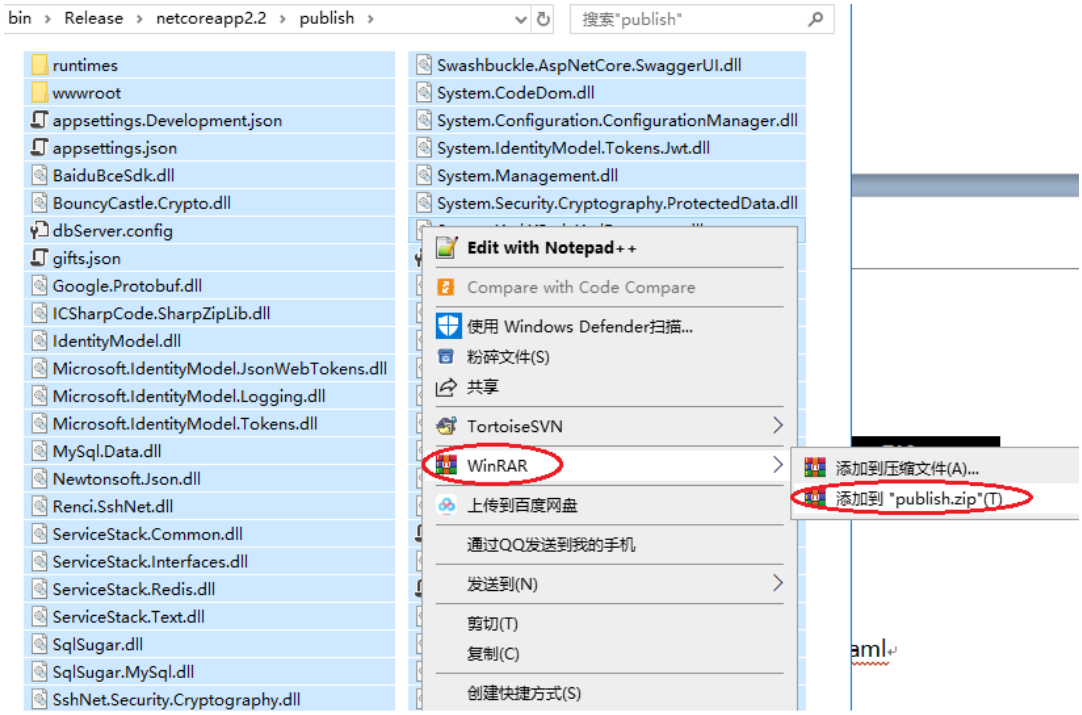
分布式应用部署

创建了 k8s 集群环境后，我们就可以在任何一个节点上部署自己的应用了。所有运行在 k8s pods 中的“应用”其实都是要基于 docker 容器化的。本节将从 0 开始说明如何构造一个 docker image，并将该 image 通过 yaml 的方式部署到 k8s 集群中。

制作 Docker 镜像

本例以一个.NET 的 web 应用为例（当然也可以是 java web），制作一个 image 镜像。由于本人的操作系统是 Windows10 HOME，无法启用 Hyper-V 在本机直接制作 docker image，只能将 web 站点传至 linux 机器中处理。以下是手动制作镜像全过程。

1. 将.net web 应用压缩为 zip 文件



2. 将 publish.zip 传送到 linux 主机中，假设放在当前登录用户的 home 目录中；
3. 新建一个 web 文件夹，将 publish.zip 移动到该文件夹；然后执行 `unzip publish.zip` 解压文件。如果提示 unzip 不是命令，则执行 `yum install unzip` 安装后，再执行 unzip 指令；
4. 删除 publish.zip 文件，然后返回到上级目录（即/home）；

5. 创建 Dockerfile，编写如下：

```
FROM mcr.microsoft.com/dotnet/core/aspnet:2.2
WORKDIR /app
EXPOSE 80
EXPOSE 443
COPY ./lxx_web/ ./
ENTRYPOINT ["dotnet",
"ZSPlatform.Lexuexi.WebApi.dll","server.urls=http://localhost:80/"]
```

解释如下

FROM 从 asp.net core 2.2 版本创建镜像
WORKDIR 定义工作目录为/app，此后所有指令都将基于/app 目录执行
EXPOSE 公开端口
COPY 将本机的网站内容，复制到镜像的 app 目录中
ENTRYPOINT 镜像入口

6. 创建镜像：`docker build -t lxx-web:1.0.0 --rm .`（将镜像创建为 lxx-web:1.0.0，并放在当前目录中）

```
[root@master data]# docker build -t lxx-web:1.0.0 --rm .
Sending build context to Docker daemon   86.7MB
Step 1/6 : FROM mcr.microsoft.com/dotnet/core/aspnet:2.2
2.2: Pulling from dotnet/core/aspnet
8f91359f1fff: Downloading [========>                                          ]  3.668MB/22.51MB
69ea005c1e27: Downloading [=========>                                         ]  3.242MB/17.69MB
afa80b070dee: Download complete
cf2efb63ce7a: Downloading [==>                                                ]  2.702MB/62.15MB
```

7. 试验镜像是否成功

   `docker run -ti --rm -p 88:80 lxx-web:1.0.0`

   或者

   `docker run -d --rm -p 88:80 lxx-web:1.0.0`

   -ti 是以控制台终端方式运行，-d 是以后台服务方式运行，-p 是指将本机的 88 端口与容器中的 80 端口做映射。无论哪一种模式运行，都可以获得正确的站点内容

   如果想替换容器内默认的站点，也可以通过加-v 数据卷映射，如下：

   `docker run -ti -v /data/lxx_web:/app -p 88:80 --rm lxx-web:1.0.0`

   其中-v 是指使用本地的站点目录替换容器中的 app 目录。

   示例使用 `curl http://localhost:88/home/getmachineid` 查看本机编码结果。

```
[root@master ~]# curl http://localhost:88/home/getmachineid
{"IsError":false,"ErrorCode":0,"StatusMessage":"A1D1-01BF-8509-E51C-48C9-09FC-4D68-8F2D","IsLogin":false,
```

8. 使用 docker images 可以看到刚刚制作的镜像

```
[root@master data]# docker images
REPOSITORY                                  TAG                 IMAGE ID            CREATED             SIZE
lxx-web                                     1.0.0               a3ea9cbd316b        8 seconds ago       297MB
```

复制镜像

由于我们的环境是集群，而这个镜像又是在本地创建的，那么在部署镜像时，创建的 pod 很有可能在 master/slave 都会有实例，每个 pod 实例创建时，要拉取这个镜像。若本地未找到，又会从公网拉取。我们的这个本地镜像根本没有上传到公共仓库，所以另一些 node 机器永远也不能拉取到镜像，会导致 pod 节点创建失败。要解决这个问题，我们要保证集群所有的 node 都能访问到 lxx-web image，有三种做法：

1.将 image push 到公共仓库；

2.搭建私有仓库，然后配置各个 node 的 docker 源；

3.使用本地镜像，将镜像向每个 node 复制一份；

为了演示方便，我们使用本地镜像的方法，通过以下手段，将 lxx-web 镜像向 node1 节点复制一份。

1)导出镜像

docker save imgname > path，例如：

```
docker save lxx-web:1.0.0 > ./lxx-web_1.0.0.tar
```

命令执行后，可以查看当前目录已经有了一个镜像文件

```
[root@master ~]# ll -h
总用量 289M
-rw-------. 1 root root 1.3K 9月   6 15:15 anaconda-ks.cfg
-rw-r--r--  1 root root  13K 9月  12 17:13 kube-flannel.yml
-rw-r--r--  1 root root 289M 9月  12 18:14 lxx-web_1.0.0.tar
```

2）接下来，通过 scp 将这个文件复制到目标机器中

```
scp lxx-web_1.0.0.tar root@192.168.31.195:~/lxx-web_1.0.0.tar
```

```
[root@master ~]# scp lxx-web_1.0.0.tar root@192.168.31.195:~/lxx-web_1.0.0.tar
root@192.168.31.195's password:
lxx-web_1.0.0.tar                                                                                   100%
```

3）到目标机器中，导入镜像

```
docker load --input ./lxx-web_1.0.0.tar
```

```
[root@node1 ~]# docker load --input ./lxx-web_1.0.0.tar
e9dc98463cd6: Loading layer [=========================================>          ] 46.24 MB/58.48 MB
```

待进度到 100%后，可通过 docker images 查看新导入的镜像。

部署镜像

接下来，利用 k8s 部署应用：创建 yaml 文件 lxx-web-rc.yaml。注意：

1.谷歌定义的 yaml 格式很严格,每个冒号后面都必须带有空格；

2.对每行的缩进也有严格要求，不能 TAB 与空格混用；

```
apiVersion: v1

kind: ReplicationController 或 Deployment（区别请参考这里）

metadata:
  name: lxx-web
spec:
  selector:
    app: lxx-web

  replicas: 2  # 节点数，设置为>1 个可以实现负载均衡效果

  template:
    metadata:
      labels:
        app: lxx-web
    spec:
      containers:
        - name: lxx-web
          image: lxx-web:1.0.0

          imagePullPolicy: IfNotPresent  #本地有镜像就不会去仓库拉取

          ports:
          - containerPort: 80
            protocol: TCP
          resources:
            requests:
              memory: "1024m"
              cpu: "1000m"
            limits:

              memory: "1Gi"    #限额内存使用不超过 1GB
```

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: lxx-web
spec:
  selector:
    app: lxx-web
  replicas: 2
  template:
    metadata:
      labels:
        app: lxx-web
    spec:
      containers:
        - name: lxx-web
          image: lxx-web:1.0.0
          imagePullPolicy: IfNotPresent
          ports:
          - containerPort: 80
            protocol: TCP
          resources:
            requests:
              memory: "1024m"
              cpu: "1000m"
            limits:
              memory: "1Gi"
```

上述配置中，resources 用于限定 pod 的资源，其中 requests 用于调度阶段，在调度 pod 时保证所有 pod 的 requests 总和，小于 node 提供的计算能力。也就是说，k8s 使用 requests 来设置各个容器需要的最小资源（requests.memory 不对应 docker，是作为 k8s 的调度依据）。

而 limits 用于限定容器在运行时期占用的资源，limits.cpu 会被转换为 docker 的-cpu-quota 参数，与 cgroup cpu.cfs_quota_us 功能相同，用来限制容器的最大 CPU 使用率。cpu 的单位使用 m（千分之一核）；limits.memory 被转为 docker 的-memory 参数，用于限制容器的最大内存，当容器申请内存超过 limits 时会被终止。

requests 与 limits 之间的关系，简单总结就是：requests 保证 pod 有足够的资源来运行，而 limits 是防止某个 pod 无限制的使用资源，而导致其它 pod 崩溃，所以两者之关的关系必须满足

```
0 <= requests <= limits
```

如果 limits=0 表示不对资源做任何限制。

待配置脚本编写后，执行以下命令创建一个 pod

`kubectl create -f lxx-web-rc.yaml`

如果提示

```
kubectl create -f lxx-web-rc.yaml
mapping values are not allowed in this context
```

或者

```
error: error validating "lxx-web-rc.yaml": error validating data: [found invalid field ports for v1.PodSpec, found invalid field -containerPort for v1.PodSpec, found invalid field -name for v1.PodSpec, found invalid field image for v1.PodSpec, found invalid field imagePullPolicy for v1.PodSpec]; if you choose to ignore these errors, turn validation off with --validate=false
```

那是因为没有严格遵守 yaml 的规范要求，要特别注意每个冒号后面必须有一个空格，并且缩进格式中 TAB 与空格不能混用。不同的缩进域，有专用的字段配置项，当某字段项出现在不合格的缩进域时，就会报 found invalid field xxx。

创建成功后中，我们可以查看 pod

```
[root@master data]# kubectl create -f lxx-web-rc.yaml
replicationcontroller/lxx-web created
```

在任何一个节点中执行 `kubectl get po` 可以得到 pod 的状态

```
[root@master data]# kubectl get po
NAME            READY   STATUS    RESTARTS   AGE
lxx-web-2dltn   1/1     Running   0          7s
lxx-web-wl648   1/1     Running   0          7s
```

ContainerCreating 提示正在创建中，这时可以查看创建日志 `kubectl describe po lxx-web-wl648`

```
NAME           READY   STATUS    RESTARTS   AGE
lxx-web-2dltn  1/1     Running   0          7s
lxx-web-wl648  1/1     Running   0          7s
[root@master data]# kubectl describe po lxx-web-wl648
Name:           lxx-web-wl648
Namespace:      default
Priority:       0
Node:           node1/192.168.31.195
Start Time:     Thu, 12 Sep 2019 18:34:31 +0800
Labels:         app=lxx-web
Annotations:    <none>
Status:         Running
IP:             10.244.1.5
Controlled By:  ReplicationController/lxx-web
Containers:
  lxx-web:
    Container ID:   docker://e48d5971f59926498b2397a69a3c62c9c8806d25497fc03f789fec6027e56b37
    Image:          lxx-web:1.0.0
    Image ID:       docker://sha256:a3ea9cbd316bld5c3c37edf376a4fb2bfa961ded51a18ab4c31d2e6a0156lb4c
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Thu, 12 Sep 2019 18:34:32 +0800
    Ready:          True
    Restart Count:  0
    Limits:
      memory:  1Gi
    Requests:
      cpu:        1
      memory:     1024m
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-nn9kj (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-nn9kj:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-nn9kj
    Optional:    false
QoS Class:       Burstable
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  48s   default-scheduler  Successfully assigned default/lxx-web-wl648 to node1
  Normal  Pulled     47s   kubelet, node1     Container image "lxx-web:1.0.0" already present on machine
  Normal  Created    47s   kubelet, node1     Created container lxx-web
  Normal  Started    47s   kubelet, node1     Started container lxx-web
```

如果在状态中，提示 redhat-cat.crt 不存在，我们先通过 ll 命令查看一下此文件是否在

`ll /etc/docker/certs.d/registry.access.redhat.com/redhat-ca.crt`

```
[root@VMLinux data]# ll /etc/docker/certs.d/registry.access.redhat.com/redhat-ca.crt
lrwxrwxrwx. 1 root root 27 9月   6 15:38 /etc/docker/certs.d/registry.access.redhat.com/redhat-ca.crt -> /etc/rhsm/ca/redhat-uep.pem
```

在结果中可以看到此文件是一个链接文件，它指向/etc/rhsm/ca/redhat-uep.pem，而这个文件确实不存在

```
[root@VMLinux data]# ls /etc/rhsm/ca/redhat-uep.pem
ls: 无法访问/etc/rhsm/ca/redhat-uep.pem: 没有那个文件或目录
```

我们需要安装一下 rhsm 这个软件：`yum install -y *rhsm*`

等待一段时间后，安装即可完成，此时再执行 ls 或 ll 查看 redhat-uep.pem 是否存在。我们发现，依然没有此文件，此时需要我们手动创建

```
touch /etc/rhsm/ca/redhat-uep.pem
```

创建之后，我们需要将刚才创建的 pod 删除，重新创建：

```
kubectl delete rc lxx-web
```

```
[root@VMLinux data]# kubectl delete rc lxx-web
replicationcontroller "lxx-web" deleted
```

```
kubectl create -f lxx-web-rc.yaml
```

等一段时间，再查看状态，会看到显示为 running，这表明各个 pod 已经在运行了。

```
[root@master data]# kubectl get po
NAME            READY   STATUS    RESTARTS   AGE
lxx-web-2dltn   1/1     Running   0          7s
lxx-web-wl648   1/1     Running   0          7s
```

发布服务

通过上面的部署操作完成后，还是无法访问应用的，我们还需要创建 service yaml

```
apiVersion: v1
kind: Service
metadata:
  name: lxx-web
spec:
  selector:
    app: lxx-web
  type: NodePort
  ports:
  - port: 8008
    targetPort: 80   #这个 port 必须与 RC container port 指定的数字相同
    nodePort: 30001  #节点暴露给外部的端口（范围必须为 30000-32767 之间）
    protocol: TCP
```

```
apiVersion: v1
kind: Service
metadata:
  name: lxx-web
spec:
  selector:
    app: lxx-web
  type: NodePort
  ports:
  - port: 8008
    targetPort: 80
    nodePort: 30001
```

保存后，执行创建指令 `kubectl create -f lxx-web-svc.yaml`

```
[root@master data]# kubectl create -f lxx-web-svc.yaml
service/lxx-web created
```

查看刚刚所有的节点 `kubectl get ep`

```
NAME         ENDPOINTS                       AGE
kubernetes   192.168.31.194:6443             4d21h
lxx-web      10.244.1.6:80,10.244.1.7:80     4d19h
```

可见 lxx-web 确实被创建了两个节点，被自动分配了 IP 地址。此时在任何一个 slave node 中使用

curl http://10.244.1.6/home/getmachineid 来验证其中之一的 pod 是否 ok。

```
[root@node1 ~]# curl http://10.244.1.6/home/getmachineid
{"IsError":false,"ErrorCode":0,"StatusMessage":"6CB0-1ADF-FBFA-8E4D-81B3-2028-B649-6353",'
[root@node1 ~]# curl http://10.244.1.7/home/getmachineid
{"IsError":false,"ErrorCode":0,"StatusMessage":"BD66-7B4A-3E06-F502-0B7E-A418-8182-4AC2",'
```

查看刚刚创建的服务 `kubectl get svc`

```
[root@master data]# kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.1.0.1       <none>        443/TCP          126m
lxx-web      NodePort    10.1.85.132    <none>        8008:30001/TCP   7s
```

这时，就可以通过 cluster ip:8008 访问站点了（在集群内网络访问）:

```
curl http://10.1.85.132:8008/home/getmachineid
```

通过多次访问上面的地址，会发现每次返回的机器 ID 不同，这说明集群 IP 已实现了负载均衡。

```
[root@node1 ~]# curl http://10.1.85.132:8008/home/getmachineid
{"IsError":false,"ErrorCode":0,"StatusMessage":"6CB0-1ADF-FBFA-8E4D-81B3-2028-B649-6353",'
 http://10.1.85.132:8008/home/getmachineid
{"IsError":false,"ErrorCode":0,"StatusMessage":"BD66-7B4A-3E06-F502-0B7E-A418-8182-4AC2",'
```

如果要在外部访问，则需要使用节点物理机 `IP:30001`，如下图:

```
curl http://192.168.31.195:30001/home/getmachineid
```

```
[root@node1 ~]# curl http://192.168.31.195:30001/home/getmachineid
{"IsError":false,"ErrorCode":0,"StatusMessage":"BD66-7B4A-3E06-F502-0B7E-A418-8182-4AC2",'
```

如果发现**服务访问失败,**无法通过 nodeip:nodeport 的方式访问节点服务，则需要设置一些配置

```
iptables -P FORWARD ACCEPT
```

如果在局域网的另一台计算机中，无法通过节点物理机 ip:30001 的方式访问，那是因为我们的 30001 端口

可能被防火墙屏蔽了，开放 30001 端口

```
firewall-cmd --permanent --zone=public --add-port=30001/tcp
firewall-cmd --reload
```

然后任何一台局域网电脑中，访问 `http://192.168.31.195:30001/home/getmachineid`

master 角色

默认情况下，k8s 为了保证 Master 的安全与高性能，只允许在 master 上调度一些管理所必须的 pods，但不允许把业务 pods 调度到 master 的。如果有时受限于资源 或 想把一些业务 pod 被调度到 master，需要人工设置 master 为可调度的 role。其原理就是在 master 上打一个污点 taints 操作如下：

```
kubectl taint node master-name node-role.kubernetes.io/master-
```

如果要使 master 恢复为默认不可调度，操作如下：

```
kubectl taint node master-name node-role.kubernetes.io/master=""
```

注意：不推荐在 master 调度 pods。

对外暴露 Services

在上一节分布式应用部署时，我们能通过 NodePort 的方式，可以使外部网络访问 lxx-web。本章节将系统性的解释 k8s 集群中各服务的对外公开方式（不止 NodePort 一种方式）。

本章节介绍的各个公开方式，还没有完全搞透，暂存记录，后面待续

kube-proxy 转发的两种模式

kube-proxy 是一个简单的网络代理与负载均衡器，负责 service 的代理实现，每个 servic 都会在所有 kube-proxy 节点上体现。具体来说，就是实现了内部从 pod 到 service 和外部的从 node port 指向 service 的访问。kube-proxy 在转发时有两种模式 Userspace 与 Iptables。userspace 是在用户空间通过 proxy 实现 LB 的代理服务，在 1.2 版本前是 k8s-proxy 的默认方式，效率不太高。在 1.2 版本之后，iptables 是默认方式，纯采用 iptables 来实现 LB，所有转发通过 Iptables 内核模块实现，而 proxy 只负责生成相应

的 iptables 规则。

使用 userspace 模式（1.2 版本前的默认模式），外部网络可以直接访问 cluster IP，使用 Iptables 模式，外部网络不能直接访问 cluster IP。

Service 的三种端口

Port

service 暴露在 cluster ip 上的端口，:port 是提供给集群内部客户访问 service 的入口。

NodePort

是 k8s 提供给集群外部客户访问 service 入口的一种方式。

TargetPort

targetPort 是 pod 上的端口，从 port 和 nodeport 上来的数据最终经过 kube-proxy 流入到后端 pod 的 targetPort 上进入容器。

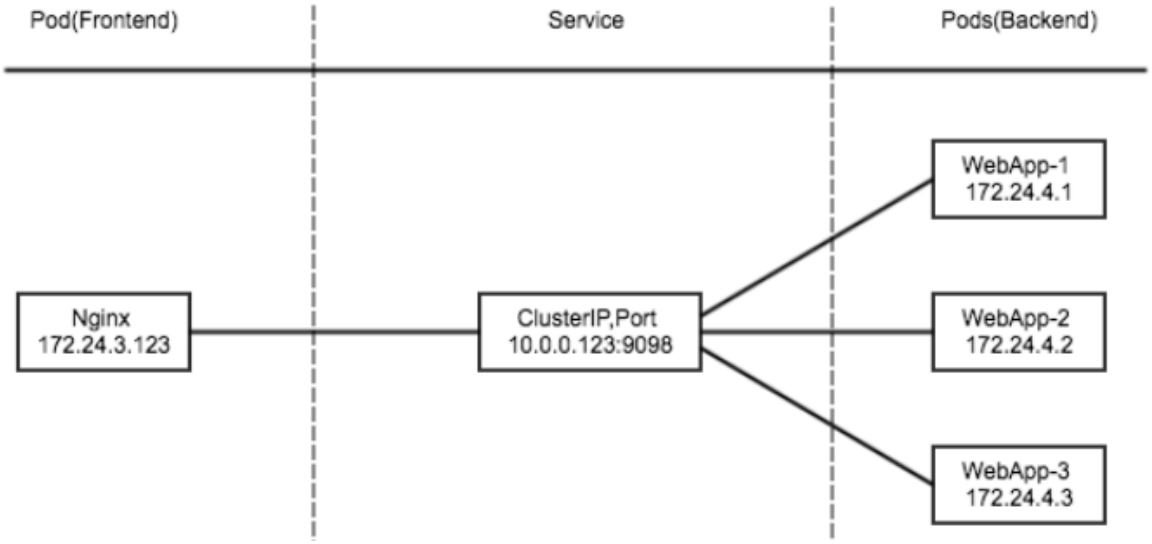
总的来说，port 与 nodePort 都是 service 的端口，前者暴露给集群内访问服务，后者暴露给集群外访问服务。从这两端口来的请求与数据，都会流入后端 Pod 的 targetPort，从而到达实际的容器。

转发后端服务的四种方式

在 k8s 中，所有的功能资源是以 service 方式对外公布，一个 service 可以由一个或多个 pod 提供支撑。每个 service 会被分配一个集群 IP（虚似 IP），这个集群 IP 的范围是通过 k8s API server 的启动参数 --service-cluster-ip-range=19.254.0.0/16 配置的。k8s 提供了 NodePort Service、 LoadBalancer 和 Ingress 对外发布 Service。

ClusterIP

此类型会提供一个集群内部的虚拟 IP（与 Pod 不在同一网段），以供集群内部 Pod 之间通信使用。



为了实现这种模式，k8s 是由几个组件协同工作的：

apiserver：在创建 service 时，api server 接收到请求后将数据存在 etcd 中；

kube-proxy：k8s 的每个节点都有该进程，负责实现 service 功能，并感知 service、pod 的变化，将变化信息写入本地的 Iptables 中；

iptables：使用 NAT 等技术将 virtualIP 的流量转至 endpoint 中。

NodePort

NodePort 除了使用 cluster ip 外，也将 service 的 port 映射到每个 node 的一个内部 port 上，映射的每个 node 的内部 port 都一样。为每个 node 暴露一个端口，通过 nodeip+nodeport 就可以访问这个服务，同时服务依然会有 cluster 类型的 ip+port。内部通过 cluster ip 方式访问，外部通过 nodeip:nodeport 方式访问。在生产环境时，不推荐使用 NodePort 公开服务。

LoadBalance

LB 在 NodePort 基础上，k8s 可以请求底层云平台创建一个负载均衡器，将每个 Node 作为后端，进行服

务分发，该模式需要底层云平台（例如 GCE）支持。**生产环境推荐方式。**

Ingress

Ingress 是一种 HTTP 方式的路由转发机制，由 Ingress controller 与 HTTP 代理服务器组合而成。它实时监控K8S API,实时更新HTTP代理服务器的转发规则。HTTP代理服务器有GCE Load-Balancer、HAProxy、Nginx 等方案。

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
name: my-ingress
spec:
rules:
- host: my.example.com
http:
paths:
- path: /app
backend:
serviceName: my-app
servicePort: 80
```

Ingress 定义中的.spec.rules 设置了转发规则，其中配置了一条规则，当 HTTP 请求的 host 为 my.example.com 且 path 为/app 时，转发到 Service my-app 的 80 端口；

```
#kubectl create -f my-ingress.yaml; kubectl get ingress my-ingress
NAME         RULE      BACKEND        ADDRESS
my-ingress     -
           my.example.com
           /app           my-app:80
```

当 Ingress 创建成功后,需要 Ingress Controller 根据 Ingress 的配置,设置 HTTP 代理服务器的转发策略,外部通过 HTTP 代理服务就可以访问到 Service。

比较好的参考：https://www.cnblogs.com/justmine/p/8991379.html

k8s web ui/api

外部访问集群状态信息

在安装 k8s 之后，可通过 主机 ip:8080 的方式访问 k8s 的 web api。但是如果遇到无法访问 或者

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

是因为/etc/kubernetes/apiserver 文件中的-insecure-bind-address 参数默认为 127.0.0.1，即 API-server 绑定的安全 IP 只有 127.0.0.1，相当于一个白名单，修改成如下值后，表示运行所有节点进行访问。

`-insecure-bind-address=0.0.0.0`

```
###
# kubernetes system config
#
# The following values are used to configure the kube-apiserver
#

# The address on the local server to listen to.
KUBE_API_ADDRESS="--insecure-bind-address=0.0.0.0"
```

然后访问 http://192.168.31.194:8080/


```
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/apps",
    "/apis/apps/v1beta1",
    "/apis/authentication.k8s.io",
    "/apis/authentication.k8s.io/v1beta1",
    "/apis/authorization.k8s.io",
    "/apis/authorization.k8s.io/v1beta1",
    "/apis/autoscaling",
    "/apis/autoscaling/v1",
    "/apis/batch",
    "/apis/batch/v1",
    "/apis/batch/v2alpha1",
    "/apis/certificates.k8s.io",
    "/apis/certificates.k8s.io/v1alpha1",
    "/apis/extensions",
    "/apis/extensions/v1beta1",
    "/apis/policy",
    "/apis/policy/v1beta1",
    "/apis/rbac.authorization.k8s.io",
    "/apis/rbac.authorization.k8s.io/v1alpha1",
    "/apis/storage.k8s.io",
    "/apis/storage.k8s.io/v1beta1",
    "/healthz",
    "/healthz/ping",
    "/healthz/poststarthook/bootstrap-controller",
    "/healthz/poststarthook/extensions/third-party-resources",
    "/healthz/poststarthook/rbac/bootstrap-roles",
    "/logs",
    "/metrics",
    "/swaggerapi/",
    "/ui/",
    "/version"
  ]
}
```


安装 dashboard

K8S 还提供了 dashboard （web ui）控制界面，专们用于管理 k8s 集群并查看集群状态。在默认情况下，安装 k8s 后是不包含 dashboard 的，需要专们创建它。安装 dashboard 之后，不用命令行，也可以完成集群部署与管理工作。

官方文档是最重要的参考资料：

https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

在官方文档中，创建 dashboard 的命令行为

```
kubectl create -f
https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended
```

`/kubernetes-dashboard.yaml`

然而，如果直接用官方的指令创建的话，会有很多问题。目前网上有人将 dashboard 镜像搞定了，可以直接用 docker 镜像，具体的部署过程可参考以下 URL：

https://www.jianshu.com/p/6f42ac331d8a
https://github.com/minminmsn/k8s1.13/blob/master/kubernetes-dashboard-amd64/Kubernetes1.13.1%E9%83%A8%E7%BD%B2Kuberneted-dashboard%20v1.10.1.md

RC 与 Deployment 区别

两者区别

Replication Controller 为 K8S 的一个核心内容，应用托管到 k8s 之后，需保证应用能持续运行，RC 就可以做到这一点，主要功能是：

- 确保 Pod 数量：rc 会确保 k8s 中有指定数量的 pod 在运行，如果少于限定数量会创建新的，反之会删掉多余的以保证 pod 数量不变；
- 确保 Pod 健康：当 pod 不健康、运行出错或无法提供服务时，rc 会杀死不健康的 pod，重新建新 pod；
- 弹性伸缩：在业务高峰或低峰时，可通过 rc 动态调整 pod 数量来提供资源利用率。同时配置相应的监控功能，定时自动从监控平台获取 rc 关联的 pod 整体资源使用情况，做到自动伸缩；
- 滚动升级：一种平滑的升级方式，通过逐步替换的策略，保证整体系统的稳定,在初始化升级的时候可以及时发现问题并解决，避免问题不断扩大；

Deployment 同样是 k8s 的一个核心内容，90%的功能与 RC 一致，可以看作是新一代 RC，它具备了 RC 之外的新特性：

- 具备 RC 的全部功能；
- 事件和状态查看：可以查看 deployment 的升级进度与状态；
- 回滚：当升级 pod 或相关参数时发现问题，可以利用回滚操作退回到上一个版本或指定的版本；
- 版本记录：每一次 deployment 操作，都能保存下来；
- 暂停/启动：对于每一次升级，可以随时暂停和启动；

- 多种升级方案：Recreate 删除已有 pod 重新建新 pod；RollingUpdate 滚动升级逐步替换；

Deployment 的示例文件

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: review-demo
  namespace: scm
  labels:
    app: review-demo
spec:
  replicas: 3
#  minReadySeconds: 60     #滚动升级时 60s 后认为该 pod 就绪
  strategy:
    rollingUpdate:  ##由于 replicas 为 3,则整个升级,pod 个数在 2-4 个之间
      maxSurge: 1      #滚动升级时会先启动 1 个 pod
      maxUnavailable: 1 #滚动升级时允许的最大 Unavailable 的 pod 个数
  template:
    metadata:
      labels:
        app: review-demo
    spec:
      terminationGracePeriodSeconds: 60 ##k8s 将会给应用发送 SIGTERM 信号，可以用来正确、优
雅地关闭应用,默认为 30 秒
      containers:
      - name: review-demo
        image: library/review-demo:0.0.1-SNAPSHOT
        imagePullPolicy: IfNotPresent
        livenessProbe: #kubernetes 认为该 pod 是存活的,不存活则需要重启
          httpGet:
            path: /health
            port: 8080
            scheme: HTTP
          initialDelaySeconds: 60 ## equals to the maximum startup time of the application
+ couple of seconds
          timeoutSeconds: 5
          successThreshold: 1
```

```
        failureThreshold: 5
      readinessProbe: #kubernetes 认为该 pod 是启动成功的
        httpGet:
          path: /health
          port: 8080
          scheme: HTTP
        initialDelaySeconds: 30 ## equals to minimum startup time of the application
        timeoutSeconds: 5
        successThreshold: 1
        failureThreshold: 5
      resources:
        # keep request = limit to keep this container in guaranteed class
        requests:
          cpu: 50m
          memory: 200Mi
        limits:
          cpu: 500m
          memory: 500Mi
      env:
        - name: PROFILE
          value: "test"
      ports:
        - name: http
          containerPort: 8080
```

几个重要参数说明

maxSurge 与 maxUnavailable

maxSurge: 1 表示滚动升级时会先启动 1 个 pod

maxUnavailable: 1 表示滚动升级时允许的最大 Unavailable 的 pod 个数

由于 replicas 为 3,则整个升级,pod 个数在 2-4 个之间

terminationGracePeriodSeconds

k8s 将会给应用发送 SIGTERM 信号,可以用来正确、优雅地关闭应用,默认为 30 秒。如果需要更优雅地关闭，则可以使用 k8s 提供的 pre-stop lifecycle hook 的配置声明，将会在发送 SIGTERM 之前执行。

livenessProbe 与 readinessProbe

livenessProbe 是 kubernetes 认为该 pod 是存活的，不存在则需要 kill 掉，然后再新启动一个，以达到 replicas 指定的个数。

readinessProbe 是 kubernetes 认为该 pod 是启动成功的，这里根据每个应用的特性，自己去判断，可以执行 command，也可以进行 httpGet。比如对于使用 java web 服务的应用来说，并不是简单地说 tomcat 启动成功就可以对外提供服务的，还需要等待 spring 容器初始化，数据库连接连接上等等。对于 spring boot 应用，默认的 actuator 带有 /health 接口，可以用来进行启动成功的判断。其中 readinessProbe.initialDelaySeconds 可以设置为系统完全启动起来所需的最少时间，livenessProbe.initialDelaySeconds 可以设置为系统完全启动起来所需的最大时间+若干秒。

这几个参数配置好了之后，基本就可以实现近乎无缝地平滑升级了。对于使用服务发现的应用来说，readinessProbe 可以去执行命令，去查看是否在服务发现里头应该注册成功了，才算成功。

在新版本的 Kubernetes 中建议使用 ReplicaSet 来取代 ReplicationCtronller。ReplicaSet 跟 ReplicationCtronller 没有本质的不同，只是名字不一样，并且 ReplicaSet 支持集合式的 selector。虽然 ReplicaSet 可以独立使用，但一般还是建议使用 Deployment 来自动管理 ReplicaSet，这样就无需担心跟其他机制的不兼容问题（比如 ReplicaSet 不支持 rolling-update 但 Deployment 支持）。

Deployment 的常用命令

查看部署状态

kubectl rollout status deployment/xxxx --namespace=xx

kubectl describe deployment/xxxx --namespace=xx

升级

kubectl set image deployment/xxx xxx=新版本 --namespace=xx

或者

kubectl edit deployment/xxx --namespace=xx

编辑 spec/template/spec/containers[0]/image 的值

终止升级

kubectl rollout pause deployment/xxx --namespace=xx

继续升级

kubectl rollout resume deployment/xxx --namespace=xx

回滚

kubectl rollout undo deployment/xxx --namespace=xx

查看 deployment 版本

kubectl rollout history deployments --namespace=xx

回滚到指定版本

kubectl rollout undo deployment/xxx --to-revision=x --namespace=xx