第 2 章

Kubernetes 实践指南

本章将从 Kubernetes 的系统安装开始，逐步介绍 Kubernetes 的服务相关配置、命令行工具 kubectl 的使用详解，然后通过大量案例实践对 Kubernetes 最核心的容器和微服务架构的概念和用法进行详细说明。

2.1 Kubernetes 安装与配置

2.1.1 安装 Kubernetes

Kubernetes 系统由一组可执行程序组成，用户可以通过 GitHub 上的 Kubernetes 项目页下载编译好的二进制包，或者下载源代码并编译后进行安装。

安装 Kubernetes 对软件和硬件的系统要求如表 2.1 所示。

表 2.1 安装 Kubernetes 对软件和硬件的系统要求

软 硬 件	最 低 配 置	推 荐 配 置
CPU 和内存	Master：至少 1 core 和 2GB 内存 Node：至少 1 core 和 2GB 内存	Master：2 core 和 4GB 内存 Node：由于要运行 Docker，所以应根据需要运行的容器数量进行调整
Linux 操作系统	基于 x86_64 架构的各种 Linux 发行版本，包括 Red Hat Linux、CentOS、Fedora、Ubuntu 等，Kernel 版本要求在 3.10 及以上。 也可以在谷歌的 GCE（Google Compute Engine）或者 Amazon 的 AWS（Amazon Web Service）云平台上进行安装	Red Hat Linux 7 CentOS 7

续表

软 硬 件	最 低 配 置	推 荐 配 置
Docker	1.9 版本及以上 下载和安装说明见 https://www.docker.com	1.12 版本
etcd	2.0 版本及以上 下载和安装说明见 https://github.com/coreos/etcd/releases	3.0 版本

最简单的安装方法是使用 yum install kubernetes 命令完成 Kubernetes 集群的安装，但仍需修改各组件的启动参数，才能完成 Kubernetes 集群的配置。

本章以二进制文件和手工配置启动参数的形式进行安装，对每个组件的配置进行详细说明。

从 Kubernetes 官网下载编译好的二进制包，如图 2.1 所示，下载地址为 https://github.com/kubernetes/kubernetes/releases。本书基于 Kubernetes 1.3 版本进行说明。

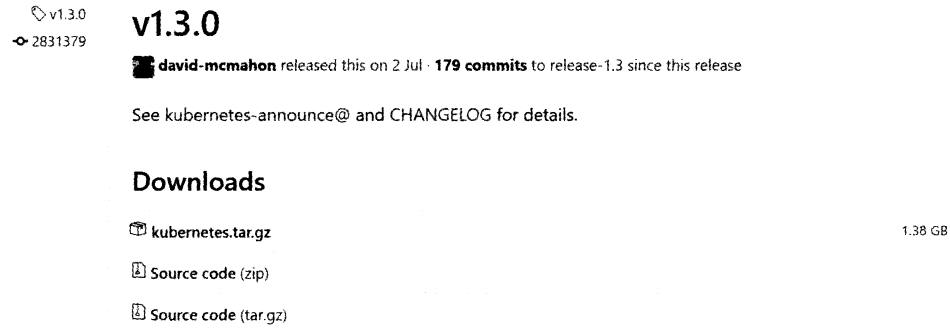

图 2.1 GitHub 上 Kubernetes 的下载页面

在压缩包 kubernetes.tar.gz 内包含了 Kubernetes 的服务程序文件、文档和示例。

解压缩后，server 子目录中的 kubernetes-server-linux-amd64.tar.gz 文件包含了 Kubernetes 需要运行的全部服务程序文件。服务程序文件列表如表 2.2 所示。

表 2.2 服务程序文件列表

文 件 名	说 明
hyperkube	总控程序，用于运行其他 Kubernetes 程序
kube-apiserver	apiserver 主程序
kube-apiserver.docker_tag	apiserver docker 镜像的 tag
kube-apiserver.tar	apiserver docker 镜像文件
kube-controller-manager	controller-manager 主程序
kube-controller-manager.docker_tag	controller-manager docker 镜像的 tag
kube-controller-manager.tar	controller-manager docker 镜像文件
kubectl	客户端命令行工具

续表

文　件　名	说　　明
kubelet	kubelet 主程序
kube-proxy	proxy 主程序
kube-scheduler	scheduler 主程序
kube-scheduler.docker_tag	scheduler docker 镜像的 tag
kube-scheduler.tar	scheduler docker 镜像文件

Kubernetes Master 节点安装部署 etcd、kube-apiserver、kube-controller-manager、kube-scheduler 服务进程。我们使用 kubectl 作为客户端与 Master 进行交互操作，在工作 Node 上仅需部署 kubelet 和 kube-proxy 服务进程。Kubernetes 还提供了一个“all-in-one”的 hyperkube 程序来完成对以上服务程序的启动。

2.1.2　配置和启动 Kubernetes 服务

Kubernetes 的服务都可以通过直接运行二进制文件加上启动参数完成。为了便于管理，常见的做法是将 Kubernetes 服务程序配置为 Linux 的系统开机自动启动的服务。

本节以 CentOS Linux 7 为例，使用 Systemd 系统完成 Kubernetes 服务的配置。其他 Linux 发行版的服务配置请参考相关的系统管理手册。

需要注意的是，CentOS Linux 7 默认启动了 firewalld——防火墙服务，而 Kubernetes 的 Master 与工作 Node 之间会有大量的网络通信，安全的做法是在防火墙上配置各组件需要相互通信的端口号，具体要配置的端口号详见 2.1.6 节中各服务监听的端口号说明。在一个安全的内部网络环境中可以关闭防火墙服务：

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

将 Kubernetes 的可执行文件复制到/usr/bin（如果复制到其他目录，则将 systemd 服务文件中的文件路径修改正确即可），然后对服务进行配置。

在下面的服务启动参数说明中主要介绍最重要的启动参数，每个服务的启动参数还有很多，详见 2.1.6 节的完整说明。有兴趣的读者可以尝试修改它们，以观察服务运行的不同效果。

1. Master 上的 etcd、kube-apiserver、kube-controller-manager、kube-scheduler 服务

1）etcd 服务

etcd 服务作为 Kubernetes 集群的主数据库，在安装 Kubernetes 各服务之前需要首先安装和启动。

从 GitHub 官网下载 etcd 发布的二进制文件，将 etcd 和 etcdctl 文件复制到/usr/bin 目录。

设置 systemd 服务文件/usr/lib/systemd/system/etcd.service：

```
[Unit]
Description=Etcd Server
After=network.target

[Service]
Type=simple
WorkingDirectory=/var/lib/etcd/
EnvironmentFile=-/etc/etcd/etcd.conf
ExecStart=/usr/bin/etcd

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

其中 WorkingDirectory（/var/lib/etcd/）表示 etcd 数据保存的目录，需要在启动 etcd 服务之前进行创建。

配置文件/etc/etcd/etcd.conf 通常不需要特别的参数设置（详细的参数配置内容参见官方文档），etcd 默认将监听在 http://127.0.0.1:2379 地址供客户端连接。

配置完成后，通过 systemctl start 命令启动 etcd 服务。同时，使用 systemctl enable 命令将服务加入开机启动列表中：

```
# systemctl daemon-reload
# systemctl enable etcd.service
# systemctl start etcd.service
```

通过执行 etcdctl cluster-health，可以验证 etcd 是否正确启动：

```
# etcdctl cluster-health
member ce2a822cea30bfca is healthy: got healthy result from http://127.0.0.1:2379
cluster is healthy
```

2）kube-apiserver 服务

将 kube-apiserver 的可执行文件复制到/usr/bin 目录。

编辑 systemd 服务文件/usr/lib/systemd/system/kube-apiserver.service，内容如下：

```
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=etcd.service
Wants=etcd.service

[Service]
EnvironmentFile=/etc/kubernetes/apiserver
```

```
ExecStart=/usr/bin/kube-apiserver $KUBE_API_ARGS
Restart=on-failure
Type=notify
LimitNOFILE=65536

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

配置文件/etc/kubernetes/apiserver 的内容包括了 kube-apiserver 的全部启动参数，主要的配置参数在变量 KUBE_API_ARGS 中指定。

```
# cat /etc/kubernetes/apiserver
KUBE_API_ARGS="--etcd_servers=http://127.0.0.1:2379
--insecure-bind-address=0.0.0.0 --insecure-port=8080
--service-cluster-ip-range=169.169.0.0/16 --service-node-port-range=1-65535
--admission_control=NamespaceLifecycle,LimitRanger,SecurityContextDeny,ServiceAc
count,ResourceQuota --logtostderr=false --log-dir=/var/log/kubernetes --v=2"
```

对启动参数的说明如下。

- --etcd_servers：指定 etcd 服务的 URL。
- --insecure-bind-address：apiserver 绑定主机的非安全 IP 地址，设置 0.0.0.0 表示绑定所有 IP 地址。
- --insecure-port：apiserver 绑定主机的非安全端口号，默认为 8080。
- --service-cluster-ip-range：Kubernetes 集群中 Service 的虚拟 IP 地址段范围，以 CIDR 格式表示，例如 169.169.0.0/16，该 IP 范围不能与物理机的真实 IP 段有重合。
- --service-node-port-range：Kubernetes 集群中 Service 可映射的物理机端口号范围，默认为 30000～32767。
- --admission_control：Kubernetes 集群的准入控制设置，各控制模块以插件的形式依次生效。
- --logtostderr：设置为 false 表示将日志写入文件，不写入 stderr。
- --log-dir：日志目录。
- --v：日志级别。

3）kube-controller-manager 服务

kube-controller-manager 服务依赖于 kube-apiserver 服务。

```
# cat /usr/lib/systemd/system/kube-controller-manager.service
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=kube-apiserver.service
Requires=kube-apiserver.service
```

```
[Service]
EnvironmentFile=/etc/kubernetes/controller-manager
ExecStart=/usr/bin/kube-controller-manager $KUBE_CONTROLLER_MANAGER_ARGS
Restart=on-failure
LimitNOFILE=65536

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

配置文件/etc/kubernetes/controller-manager 的内容包括了 kube-controller-manager 的全部启动参数，主要的配置参数在变量 KUBE_CONTROLLER_MANAGER_ARGS 中指定。

```
# cat /etc/kubernetes/controller-manager
KUBE_CONTROLLER_MANAGER_ARGS="--master=http://192.168.18.3:8080
--logtostderr=false --log-dir=/var/log/kubernetes --v=2"
```

对启动参数的说明如下。

- **--master**：指定 apiserver 的 URL 地址。
- **--logtostderr**：设置为 false 表示将日志写入文件，不写入 stderr。
- **--log-dir**：日志目录。
- **--v**：日志级别。

4）kube-scheduler 服务

kube-scheduler 服务也依赖于 kube-apiserver 服务。

```
# cat /usr/lib/systemd/system/kube-controller-manager.service
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=kube-apiserver.service
Requires=kube-apiserver.service

[Service]
EnvironmentFile=/etc/kubernetes/scheduler
ExecStart=/usr/bin/kube-scheduler $KUBE_SCHEDULER_ARGS
Restart=on-failure
LimitNOFILE=65536

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

配置文件/etc/kubernetes/scheduler 的内容包括了 kube-scheduler 的全部启动参数，主要的配置参数在变量 KUBE_SCHEDULER_ARGS 中指定。

```
# cat /etc/kubernetes/scheduler
```

```
    KUBE_SCHEDULER_ARGS="--master=http://192.168.18.3:8080 --logtostderr=false
--log-dir=/var/log/kubernetes --v=2"
```

对启动参数的说明如下。

- **--master**：指定 apiserver 的 URL 地址。
- **--logtostderr**：设置为 false 表示将日志写入文件，不写入 stderr。
- **--log-dir**：日志目录。
- **--v**：日志级别。

配置完成后，执行 systemctl start 命令按顺序启动这 3 个服务。同时，使用 systemctl enable 命令将服务加入开机启动列表中。

```
# systemctl daemon-reload
# systemctl enable kube-apiserver.service
# systemctl start kube-apiserver.service
# systemctl enable kube-controller-manager
# systemctl start kube-controller-manager
# systemctl enable kube-scheduler
# systemctl start kube-scheduler
```

通过 systemctl status <service_name> 来验证服务的启动状态，“running”表示启动成功。

到此，Master 上所需的服务就全部启动完成了。

2. Node 上的 kubelet、kube-proxy 服务

在工作 Node 节点上需要预先安装好 Docker Daemon 并且正常启动。Docker 的安装详见 http://www.docker.com 的说明。

1）kubelet 服务

kubelet 服务依赖于 Docker 服务。

```
# cat /usr/lib/systemd/system/kubelet.service
[Unit]
Description=Kubernetes Kubelet Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=docker.service
Requires=docker.service

[Service]
WorkingDirectory=/var/lib/kubelet
EnvironmentFile=/etc/kubernetes/kubelet
ExecStart=/usr/bin/kubelet $KUBELET_ARGS
Restart=on-failure
```

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

其中 WorkingDirectory 表示 kubelet 保存数据的目录，需要在启动 kubelet 服务之前进行创建。

配置文件/etc/kubernetes/kubelet 的内容包括了 kubelet 的全部启动参数，主要的配置参数在变量 KUBELET_ARGS 中指定。

```
# cat /etc/kubernetes/kubelet
KUBELET_ARGS="--api-servers=http://192.168.18.3:8080
--hostname-override=192.168.18.3 --logtostderr=false
--log-dir=/var/log/kubernetes --v=2"
```

对启动参数的说明如下。

- **--api-servers**：指定 apiserver 的 URL 地址，可以指定多个。
- **--hostname-override**：设置本 Node 的名称。
- **--logtostderr**：设置为 false 表示将日志写入文件，不写入 stderr。
- **--log-dir**：日志目录。
- **--v**：日志级别。

2）kube-proxy 服务

kube-proxy 服务依赖于 network 服务。

```
[Unit]
Description=Kubernetes Kube-Proxy Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target
Requires=network.service

[Service]
EnvironmentFile=/etc/kubernetes/proxy
ExecStart=/usr/bin/kube-proxy $KUBE_PROXY_ARGS
Restart=on-failure
LimitNOFILE=65536

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

配置文件/etc/kubernetes/proxy 的内容包括了 kube-proxy 的全部启动参数，主要的配置参数在变量 KUBE_PROXY_ARGS 中指定。

```
# cat /etc/kubernetes/proxy
KUBE_PROXY_ARGS="--master=http://192.168.18.3:8080 --logtostderr=false
--log-dir=/var/log/kubernetes --v=2"
```

对启动参数的说明如下。

- **--master**：指定 apiserver 的 URL 地址。
- **--logtostderr**：设置为 false 表示将日志写入文件，不写入 stderr。
- **--log-dir**：日志目录。
- **--v**：日志级别。

配置完成后，通过 systemctl 启动 kubelet 和 kube-proxy 服务：

```
# systemctl daemon-reload
# systemctl enable kubelet.service
# systemctl start kubelet.service
# systemctl enable kube-proxy
# systemctl start kube-proxy
```

kubelet 默认采用向 Master 自动注册本 Node 的机制，在 Master 上查看各 Node 的状态，状态为 Ready 表示 Node 已经成功注册并且状态为可用。

```
# kubectl get nodes
NAME             STATUS     AGE
192.168.18.3      Ready      1m
```

等所有 Node 的状态都为 Ready 之后，一个 Kubernetes 集群就启动完成了。接下来就可以创建 Pod、RC、Service 等资源对象来部署 Docker 容器应用了。

2.1.3　Kubernetes 集群的安全设置

1. 基于 CA 签名的双向数字证书认证方式

在一个安全的内网环境中，Kubernetes 的各个组件与 Master 之间可以通过 apiserver 的非安全端口 http://apiserver:8080 进行访问。但如果 apiserver 需要对外提供服务，或者集群中的某些容器也需要访问 apiserver 以获取集群中的某些信息，则更安全的做法是启用 HTTPS 安全机制。Kubernetes 提供了基于 CA 签名的双向数字证书认证方式和简单的基于 HTTP BASE 或 TOKEN 的认证方式，其中 CA 证书方式的安全性最高。本节先介绍以 CA 证书的方式配置 Kubernetes 集群，要求 Master 上的 kube-apiserver、kube-controller-manager、kube-scheduler 进程及各 Node 上的 kubelet、kube-proxy 进程进行 CA 签名双向数字证书安全设置。

基于 CA 签名的双向数字证书的生成过程如下。

（1）为 kube-apiserver 生成一个数字证书，并用 CA 证书进行签名。

（2）为 kube-apiserver 进程配置证书相关的启动参数，包括 CA 证书（用于验证客户端证书

的签名真伪）、自己的经过 CA 签名后的证书及私钥。

（3）为每个访问 Kubernetes API Server 的客户端（如 kube-controller-manager、kube-scheduler、kubelet、kube-proxy 及调用 API Server 的客户端程序 kubectl 等）进程生成自己的数字证书，也都用 CA 证书进行签名，在相关程序的启动参数里增加 CA 证书、自己的证书等相关参数。

1）设置 kube-apiserver 的 CA 证书相关的文件和启动参数

使用 OpenSSL 工具在 Master 服务器上创建 CA 证书和私钥相关的文件：

```
# openssl genrsa -out ca.key 2048
# openssl req -x509 -new -nodes -key ca.key -subj "/CN=yourcompany.com" -days 5000 -out ca.crt
# openssl genrsa -out server.key 2048
```

注意：生成 ca.crt 时，-subj 参数中“/CN”的值通常为域名。

准备 **master_ssl.cnf** 文件，该文件用于 x509 v3 版本的证书。在该文件中主要需要设置 Master 服务器的 hostname（k8s-master）、IP 地址（192.168.18.3），以及 Kubernetes Master Service 的虚拟服务名称（kubernetes.default 等）和该虚拟服务的 ClusterIP 地址（169.169.0.1）。

master_ssl.cnf 文件的示例如下：

```
[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = kubernetes
DNS.2 = kubernetes.default
DNS.3 = kubernetes.default.svc
DNS.4 = kubernetes.default.svc.cluster.local
DNS.5 = k8s-master
IP.1 = 169.169.0.1
IP.2 = 192.168.18.3
```

基于 master_ssl.cnf 创建 server.csr 和 server.crt 文件。在生成 server.csr 时，-subj 参数中“/CN”指定的名字需为 Master 所在的主机名。

```
# openssl req -new -key server.key -subj "/CN=k8s-master" -config master_ssl.cnf -out server.csr
# openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -days 5000 -extensions v3_req -extfile master_ssl.cnf -out server.crt
```

全部执行完后会生成 6 个文件：ca.crt、ca.key、ca.srl、server.crt、server.csr、server.key。

将这些文件复制到一个目录中（例如/var/run/kubernetes/），然后设置 kube-apiserver 的三个启动参数“--client-ca-file”“--tls-cert-file”和“--tls-private-key-file”，分别代表 CA 根证书文件、服务端证书文件和服务端私钥文件：

```
--client_ca_file=/var/run/kubernetes/ca.crt
--tls-private-key-file=/var/run/kubernetes/server.key
--tls-cert-file=/var/run/kubernetes/server.crt
```

同时，可以关掉非安全端口 8080，设置安全端口为 443（默认为 6443）：

```
--insecure-port=0
--secure-port=443
```

最后重启 kube-apiserver 服务。

2）设置 kube-controller-manager 的客户端证书、私钥和启动参数

```
$ openssl genrsa -out cs_client.key 2048
$ openssl req -new -key cs_client.key -subj "/CN=k8s-node-1" -out cs_client.csr
$ openssl x509 -req -in cs_client.csr -CA ca.crt -CAkey ca.key -CAcreateserial
-out cs_client.crt -days 5000
```

其中，在生成 cs_client.crt 时，-CA 参数和-CAkey 参数使用的是 apiserver 的 ca.crt 和 ca.key 文件。然后将这些文件复制到一个目录中（例如/var/run/kubernetes/）。

接下来创建/etc/kubernetes/kubeconfig 文件（kube-controller-manager 与 kube-scheduler 共用），配置客户端证书等相关参数，内容如下：

```
apiVersion: v1
kind: Config
users:
- name: controllermanager
  user:
    client-certificate: /var/run/kubernetes/cs_client.crt
    client-key: /var/run/kubernetes/cs_client.key
clusters:
- name: local
  cluster:
    certificate-authority: /var/run/kubernetes/ca.crt
contexts:
- context:
    cluster: local
    user: controllermanager
  name: my-context
current-context: my-context
```

然后，设置 kube-controller-manager 服务的启动参数，注意，--master 的地址为 HTTPS 安全服务地址，不使用非安全地址 http://192.168.18.3:8080。

```
--master=https://192.168.18.3:443
--service_account_private_key_file=/var/run/kubernetes/server.key
--root-ca-file=/var/run/kubernetes/ca.crt
--kubeconfig=/etc/kubernetes/kubeconfig
```

重启 kube-controller-manager 服务。

3）设置 kube-scheduler 启动参数

kube-scheduler 复用上一步 kube-controller-manager 创建的客户端证书，配置启动参数：

```
--master=https://192.168.18.3:443
--kubeconfig=/etc/kubernetes/kubeconfig
```

重启 kube-scheduler 服务。

4）设置每台 Node 上 kubelet 的客户端证书、私钥和启动参数

首先复制 kube-apiserver 的 ca.crt 和 ca.key 文件到 Node 上，在生成 kubelet_client.crt 时-CA 参数和-CAkey 参数使用的是 apiserver 的 ca.crt 和 ca.key 文件。在生成 kubelet_client.csr 时-subj 参数中的“/CN”设置为本 Node 的 IP 地址。

```
$ openssl genrsa -out kubelet_client.key 2048
$ openssl req -new -key kubelet_client.key -subj "/CN=192.168.18.4" -out
kubelet_client.csr
$ openssl x509 -req -in kubelet_client.csr -CA ca.crt -CAkey ca.key
-CAcreateserial -out kubelet_client.crt -days 5000
```

将这些文件复制到一个目录中（例如/var/run/kubernetes/）。

接下来创建/etc/kubernetes/kubeconfig 文件（kubelet 和 kube-proxy 进程共用），配置客户端证书等相关参数，内容如下：

```
apiVersion: v1
kind: Config
users:
- name: kubelet
  user:
    client-certificate: /etc/kubernetes/ssl_keys/kubelet_client.crt
    client-key: /etc/kubernetes/ssl_keys/kubelet_client.key
clusters:
- name: local
  cluster:
    certificate-authority: /etc/kubernetes/ssl_keys/ca.crt
contexts:
- context:
    cluster: local
    user: kubelet
  name: my-context
current-context: my-context
```

然后，设置 kubelet 服务的启动参数：

```
--api_servers=https://192.168.18.3:443
--kubeconfig=/etc/kubelet/kubeconfig
```

最后重启 kubelet 服务。

5）设置 kube-proxy 的启动参数

kube-proxy 复用上一步 kubelet 创建的客户端证书，配置启动参数：

```
--master=https://192.168.18.3:443
--kubeconfig=/etc/kubernetes/kubeconfig
```

重启 kube-proxy 服务。

至此，一个基于 CA 的双向数字证书认证的 Kubernetes 集群环境就搭建完成了。

6）设置 kubectl 客户端使用安全方式访问 apiserver

在使用 kubectl 对 Kubernetes 集群进行操作时，默认使用非安全端口 8080 对 apiserver 进行访问，也可以设置为安全访问 apiserver 的模式，需要设置 3 个证书相关的参数“—certificate-authority”“--client-certificate”和“--client-key”，分别表示用于 CA 授权的证书、客户端证书和客户端密钥。

- --certificate-authority：使用为 kube-apiserver 生成的 ca.crt 文件。
- --client-certificate：使用为 kube-controller-manager 生成的 cs_client.crt 文件。
- --client-key：使用为 kube-controller-manager 生成的 cs_client.key 文件。

同时，指定 apiserver 的 URL 地址为 HTTPS 安全地址（例如 https://k8s-master:443），最后输入需要执行的子命令，即可对 apiserver 进行安全访问了：

```
    # kubectl --server=https://k8s-master:443
--certificate-authority=/etc/kubernetes/ssl_keys/ca.crt
--client-certificate=/etc/kubernetes/ssl_keys/cs_client.crt
--client-key=/etc/kubernetes/ssl_keys/cs_client.key get nodes
    NAME          STATUS    AGE
    k8s-node-1    Ready     1h
```

2. 基于 HTTP BASE 或 TOKEN 的简单认证方式

除了基于 CA 的双向数字证书认证方式，Kubernetes 也提供了基于 HTTP BASE 或 TOKEN 的简单认证方式。各组件与 apiserver 之间的通信方式仍然采用 HTTPS，但不使用 CA 数字证书。

采用基于 HTTP BASE 或 TOKEN 的简单认证方式时，API Server 对外暴露 HTTPS 端口，客户端提供用户名、密码或 Token 来完成认证过程。需要说明的是，kubectl 命令行工具比较特殊，它同时支持 CA 双向认证与简单认证两种模式与 *apiserver* 通信，其他客户端组件只能配置

为双向安全认证或非安全模式与 apiserver 通信。

基于 HTTP BASE 认证的配置过程如下。

（1）创建包括用户名、密码和 UID 的文件 basic_auth_file，放置在合适的目录中，例如/etc/kuberntes 目录。需要注意的是，这是一个纯文本文件，用户名、密码都是明文。

```
# vi /etc/kubernetes/basic_auth_file
admin,admin,1
system,system,2
```

（2）设置 kube-apiserver 的启动参数“--basic_auth_file”，使用上述文件提供安全认证：

```
--secure-port=443
--basic_auth_file=/etc/kubernetes/basic_auth_file
```

然后，重启 API Server 服务。

（3）使用 kubectl 通过指定的用户名和密码来访问 API Server：

```
# kubectl --server=https://192.168.18.3:443 --username=admin --password=admin
--insecure-skip-tls-verify=true get nodes
```

基于 TOKEN 认证的配置过程如下。

（1）创建包括用户名、密码和 UID 的文件 token_auth_file，放置在合适的目录中，例如/etc/kuberntes 目录。需要注意的是，这是一个纯文本文件，用户名、密码都是明文。

```
$ cat /etc/kubernetes/token_auth_file
admin,admin,1
system,system,2
```

（2）设置 kube-apiserver 的启动参数“--token_auth_file”，使用上述文件提供安全认证：

```
--secure-port=443
--token_auth_file=/etc/kubernetes/token_auth_file
```

然后，重启 API Server 服务。

（3）用 curl 验证和访问 API Server：

```
$ curl -k --header "Authorization:Bearer admin" https://192.168.18.3:443/version
{
  "major": "1",
  "minor": "3",
  "gitVersion": "v1.3.3",
  "gitCommit": "c6411395e09da356c608896d3d9725acab821418",
  "gitTreeState": "clean",
  "buildDate": "2016-07-22T20:22:25Z",
  "goVersion": "go1.6.2",
  "compiler": "gc",
  "platform": "linux/amd64"
}
```

2.1.4　Kubernetes 的版本升级

Kubernetes 的版本升级需要考虑到当前集群中正在运行的容器不受影响。应对集群中的各 Node 逐个进行隔离，然后等待在其上运行的容器全部执行完成，再更新该 Node 上的 kubelet 和 kube-proxy 服务，将全部 Node 都更新完成后，最后更新 Master 的服务。

- 通过官网获取最新版本的二进制包 kubernetes.tar.gz，解压缩后提取服务二进制文件。
- 逐个隔离 Node，等待在其上运行的全部容器工作完成，更新 kubelet 和 kube-proxy 服务文件，然后重启这两个服务。
- 更新 Master 的 kube-apiserver、kube-controller-manager、kube-scheduler 服务文件并重启。

2.1.5　内网中的 Kubernetes 相关配置

Kubernetes 在能够访问 Internet 网络的环境中使用起来非常方便，一方面在 docker.io 和 gcr.io 网站中已经存在了大量官方制作的 Docker 镜像，另一方面 GCE、AWS 提供的云平台已经很成熟了，用户通过租用一定的空间来部署 Kubernetes 集群也很简便。

但是，许多企业内部由于安全性的原因无法访问 Internet。对于这些企业就需要通过创建一个内部的私有 Docker Registry，并修改一些 Kubernetes 的配置，来启动内网中的 Kubernetes 集群。

1. Docker Private Registry（私有 Docker 镜像库）

使用 Docker 提供的 Registry 镜像创建一个私有镜像仓库。

详细的安装步骤请参考 Docker 的官方文档 https://docs.docker.com/registry/deploying/。

2. kubelet 配置

由于在 Kubernetes 中是以 Pod 而不是 Docker 容器为管理单元的，在 kubelet 创建 Pod 时，还通过启动一个名为 google_containers/pause 的镜像来实现 Pod 的概念。

该镜像存在于谷歌镜像库 http://gcr.io 中，需要通过一台能够连上 Internet 的服务器将其下载，导出文件，再 push 到私有 Docker Registry 中去。

之后，可以给每台 Node 的 kubelet 服务的启动参数加上--pod_infra_container_image 参数，指定为私有 Docker Registry 中 pause 镜像的地址。例如：

```
    # cat /etc/kubernetes/kubelet
    KUBELET_ARGS="--api-servers=http://192.168.18.3:8080
--hostname-override=192.168.18.3 --log-dir=/var/log/kubernetes --v=2
```

```
--pod_infra_container_image=gcr.io/google_containers/pause-amd64:3.0"
```

如果该镜像无法下载，则也可以从 Docker Hub 上进行下载：

```
# docker pull kubeguide/google_containers/pause-amd64:3.0
```

修改 kubelet 配置文件中的 —pod_infra_container_image 参数如下：

```
--pod_infra_container_image=kubeguide/google_containers/pause-amd64:3.0
```

然后重启 kubelet 服务：

```
# systemctl restart kubelet
```

通过以上设置就在内网环境中搭建了一个企业内部的私有容器云平台。

2.1.6 Kubernetes 核心服务配置详解

我们在 2.1.2 节对 Kubernetes 各服务启动进程的关键配置参数进行了简要说明，实际上 Kubernetes 的每个服务都提供了许多可配置的参数。这些参数涉及安全性、性能优化及功能扩展（Plugin）等方方面面。全面理解和掌握这些参数的含义和配置，无论对于 Kubernetes 的生产部署还是日常运维都有很好的帮助。

每个服务的可用参数都可以通过运行“cmd --help”命令进行查看，其中 cmd 为具体的服务启动命令，例如 kube-apiserver、kube-controller-manager、kube-scheduler、kubelet、kube-proxy 等。另外，也可以通过在命令的配置文件（例如/etc/kubernetes/kubelet 等）中添加“--参数名=参数取值”的语句来完成对某个参数的配置。

本节将对 Kubernetes 所有服务的参数进行全面介绍，为了方便学习和查阅，对每个服务的参数用一个小节进行详细说明。

1. 公共配置参数

公共配置参数适用于所有服务，如表 2.3 所示的参数可用于 kube-apiserver、kube-controller-manager、kube-scheduler、kubelet、kube-proxy。本节对这些参数进行统一说明，不再在每个服务的参数列表中列出。

表 2.3 公共配置参数表

参数名和取值示例	说　明
--log-backtrace-at=:0	记录日志每到“file:行号”时打印一次 stack trace
--log-dir=	日志文件路径
--log-flush-frequency=5s	设置 flush 日志文件的时间间隔
--logtostderr=true	设置为 true 则表示将日志输出到 stderr，不输出到日志文件

续表

参数名和取值示例	说　明
--alsologtostderr=false	设置为 true 则表示将日志输出到文件的同时输出到 stderr
--stderrthreshold=2	将该 threshold 级别之上的日志输出到 stderr
--v=0	glog 日志级别
--vmodule=	glog 基于模块的详细日志级别
--version=[false]	设置为 true 则将打印版本信息然后退出

2. kube-apiserver 启动参数

对 kube-apiserver 启动参数的详细说明如表 2.4 所示。

表 2.4　对 kube-apiserver 启动参数的详细说明

参数名和取值示例	说　明
--admission-control="AlwaysAdmit"	对发送给 API Server 的任何请求进行准入控制，配置为一个“准入控制器”的列表，多个准入控制器时以逗号分隔。多个准入控制器将按顺序对发送给 API Server 的请求进行拦截和过滤，若某个准入控制器不允许该请求通过，则 API Server 拒绝此调用请求。可配置的准入控制器如下。 ◎ AlwaysAdmit：允许所有请求。 ◎ AlwaysPullImages：在启动容器之前总是去下载镜像，相当于在每个容器的配置项 imagePullPolicy=Always。 ◎ AlwaysDeny：禁止所有请求，一般用于测试。 ◎ DenyExecOnPrivileged：它会拦截所有想在 privileged container 上执行命令的请求。如果你的集群支持 privileged container，你又希望限制用户在这些 privileged container 上执行命令，那么强烈推荐你使用它。 ◎ ServiceAccount：这个 plug-in 将 serviceAccounts 实现了自动化，如果你想要使用 ServiceAccount 对象，那么强烈推荐你使用它。 ◎ SecurityContextDeny：这个插件将使用了 SecurityContext 的 Pod 中定义的选项全部失效。SecurityContext 在 container 中定义了操作系统级别的安全设定（uid、gid、capabilities、SELinux 等）。 ◎ ResourceQuota：用于配额管理目的，作用于 Namespace 上，它会观察所有的请求，确保在 namespace 上的配额不会超标。推荐在 admission control 参数列表中这个插件排最后一个。 ◎ LimitRanger：用于配额管理，作用于 Pod 与 Container 上，确保 Pod 与 Container 上的配额不会超标。 ◎ NamespaceExists（已过时）：对所有请求校验 namespace 是否已存在，如果不存在则拒绝请求。已合并至 NamespaceLifecycle。 ◎ NamespaceAutoProvision（已过时）：对所有请求校验 namespace，如果不存在则自动创建该 namespace，推荐使用 NamespaceLifecycle。

续表

参数名和取值示例	说　明
--admission-control="AlwaysAdmit"	◎ NamespaceLifecycle：如果尝试在一个不存在的 namespace 中创建资源对象，则该创建请求将被拒绝。当删除一个 namespace 时，系统将会删除该 namespace 中的所有对象，包括 Pod、Service 等。 如果启用多种准入选项，则建议加载的顺序是： --admission-control=NamespaceLifecycle,LimitRanger,SecurityContextDeny,ServiceAccount,ResourceQuota
--admission-control-config-file=""	与准入控制规则相关的配置文件
--advertise-address=<nil>	用于广播给集群的所有成员自己的 IP 地址，不指定该地址将使用“—bind-address”定义的 IP 地址
--allow-privileged=false	如果设置为 true，则 Kubernetes 将允许在 Pod 中运行拥有系统特权的容器应用，与 docker run --privileged 的功效相同
--apiserver-count=1	集群中运行的 API Server 数量
--authentication-token-webhook-cache-ttl=2m0s	将 webhook token authenticator 返回的响应保存在缓存内的时间，默认为两分钟
--authentication-token-webhook-config-file=""	Webhook 相关的配置文件，将用于 token authentication
--authorization-mode="AlwaysAllow"	到 API Server 的安全访问的认证模式列表，以逗号分隔，可选值包括：AlwaysAllow、AlwaysDeny、ABAC、Webhook、RBAC
--authorization-policy-file=""	当--authorization-mode 设置为 ABAC 时使用的 csv 格式的授权配置文件
--authorization-rbac-super-user=""	当--authorization-mode 设置为 RBAC 时使用的超级用户名，使用该用户名可以不进行 RBAC 认证
--authorization-webhook-cache-authorized-ttl=5m0s	将 webhook authorizer 返回的“已授权”响应保存在缓存内的时间，默认为 5 分钟。
--authorization-webhook-cache-unauthorized-ttl=30s	将 webhook authorizer 返回的“未授权”响应保存在缓存内的时间，默认为 30 秒
--authorization-webhook-config-file=""	当--authorization-mode 设置为 webhook 时使用的授权配置文件
--basic-auth-file=""	设置该文件用于通过 HTTP 基本认证的方式访问 API Server 的安全端口
--bind-address=0.0.0.0	Kubernetes API Server 在本地址的 6443 端口开启安全的 HTTPS 服务，默认为 0.0.0.0
--cert-dir="/var/run/kubernetes"	TLS 证书所在的目录，默认为/var/run/kubernetes。如果设置了--tls-cert-file 和 --tls-private-key-file，则该设置将被忽略
--client-ca-file=""	如果指定，则该客户端证书将被用于认证过程
--cloud-config=""	云服务商的配置文件路径，不配置则表示不使用云服务商的配置文件
--cloud-provider=""	云服务商的名称，不配置则表示不使用云服务商
--cors-allowed-origins=[]	CORS（跨域资源共享）设置允许访问的源域列表，用逗号分隔，并可使用正则表达式匹配子网。如果不指定，则表示不启用 CORS
--delete-collection-workers=1	启动 DeleteCollection 的工作线程数，用于提高清理 namespace 的效率
--deserialization-cache-size=50000	设置内存中缓存的 JSON 对象的个数

续表

参数名和取值示例	说　明
--enable-garbage-collector[=false]	设置为 true 表示启用垃圾回收器。必须与 kube-controller-manager 的该参数设置为相同的值
--enable-swagger-ui[=false]	设置为 true 表示启用 swagger ui 网页，可通过 API Server 的 URL/swagger-ui 访问
--etcd-cafile=""	到 etcd 安全连接使用的 SSL CA 文件
--etcd-certfile=""	到 etcd 安全连接使用的 SSL 证书文件
--etcd-keyfile=""	到 etcd 安全连接使用的 SSL key 文件
--etcd-prefix="/registry"	在 etcd 中保存 Kubernetes 集群数据的根目录名，默认为/registry
--etcd-quorum-read[=false]	设置为 true 表示启用 quorum read 机制
--etcd-servers=[]	以逗号分隔的 etcd 服务 URL 列表，etcd 服务以 http://ip:port 格式表示
--etcd-servers-overrides=[]	按资源覆盖 etcd 服务的设置，以逗号分隔。单个覆盖格式为：group/resource#servers，其中 servers 格式为 http://ip:port，以分号分隔
--event-ttl=1h0m0s	Kubernetes API Server 中各种事件（通常用于审计和追踪）在系统中保存的时间，默认为 1 小时
--experimental-keystone-url=""	设置 keystone 鉴权插件地址，实验用
--external-hostname=""	用于生成该 Master 的对外 URL 地址，例如用于 swagger api 文档中的 URL 地址。
--insecure-bind-address=127.0.0.1	绑定的不安全 IP 地址，与--insecure-port 共同使用，默认为 localhost。设置为 0.0.0.0 表示使用全部网络接口
--insecure-port=8080	提供非安全认证访问的监听端口，默认为 8080。应在防火墙中进行配置，以使得外部客户端不可以通过非安全端口访问 API Server
--ir-data-source="influxdb"	设置 InitialResources 使用的数据源，可配置项包括 influxdb、gcm
--ir-dbname="k8s"	InitialResources 所需指标保存在 influxdb 中的数据库名称，默认为 k8s
--ir-hawkular=""	设置 Hawkular 的 URL 地址
--ir-influxdb-host="localhost:8080/api/v1/proxy/namespaces/kube-system/services/monitoring-influxdb:api"	InitialResources 所需指标所在 influxdb 的 URL 地址，默认为 localhost:8080/api/v1/proxy/namespaces/kube-system/services/monitoring-influxdb:api
--ir-namespace-only[=false]	设置为 true 表示从相同的 namespace 内的数据进行估算
--ir-password="root"	连接 influxdb 数据库的密码
--ir-percentile=90	InitialResources 进行资源估算时的采样百分比，实验用
--ir-user="root"	连接 influxdb 数据库的用户名
--kubelet-certificate-authority=""	用于 CA 授权的 cert 文件路径
--kubelet-client-certificate=""	用于 TLS 的客户端证书文件路径
--kubelet-client-key=""	用于 TLS 的客户端 key 文件路径
--kubelet-https[=true]	指定 kubelet 是否使用 HTTPS 连接
--kubelet-timeout=5s	kubelet 执行操作的超时时间
--kubernetes-service-node-port=0	设置 Master 服务是否使用 NodePort 模式，如果设置，则 Master 服务将映射到物理机的端口号；设置为 0 表示以 ClusterIP 的形式启动 Master 服务

续表

参数名和取值示例	说　明
--long-running-request-regexp="(/|^)((watch|proxy)(/|$)\|(logs?\|portforward\|exec\|attach)/?$)"	以正则表达式配置哪些需要长时间执行的请求不会被系统进行超时处理
--master-service-namespace="default"	设置 Master 服务所在的 namespace，默认为 default
--max-connection-bytes-per-sec=0	设置为非 0 的值表示限制每个客户端连接的带宽为 xx 字节/秒，目前仅用于需要长时间执行的请求
--max-requests-inflight=400	同时处理的最大请求数量，默认为 400，超过该数量的请求将被拒绝。设置为 0 表示无限制
--min-request-timeout=1800	最小请求处理超时时间，单位为秒，默认为 1800 秒，目前仅用于 watch request handler，其将会在该时间值上加一个随机时间作为请求的超时时间
--oidc-ca-file=""	该文件内设置鉴权机构，OpenID Server 的证书将被其中一个机构进行验证。如果不设置，则将使用主机的 root CA 证书
--oidc-client-id=""	OpenID Connect 的客户端 ID，在 oidc-issuer-url 设置时必须设置
--oidc-groups-claim=""	定制的 OpenID Connect 用户组声明的设置，以字符串数组的形式表示，实验用
--oidc-issuer-url=""	OpenID 发行者的 URL 地址，仅支持 HTTPS scheme，用于验证 OIDC JSON Web Token（JWT）
--oidc-username-claim="sub"	OpenID claim 的用户名，默认为“sub”，实验用
--profiling=true	打开性能分析，可以通过 <host>:<port>/debug/pprof/地址查看程序栈、线程等系统信息
--repair-malformed-updates[=true]	设置为 true 表示服务器将尽可能修复无效或格式错误的 update request，以通过正确性校验，例如在一个 update request 中将一个已存在的 UID 值设置为空
--runtime-config=	一组 key=value 用于运行时的配置信息。apis/<groupVersion>/<resource> 可用于打开或关闭对某个 API 版本的支持。api/all 和 api/legacy 特别用于支持所有版本的 API 或支持旧版本的 API
--secure-port=6443	设置 API Server 使用的 HTTPS 安全模式端口号，设置为 0 表示不启用 HTTPS
--service-account-key-file=""	包含 PEM-encoded x509 RSA 公钥和私钥的文件路径，用于验证 Service Account 的 token。不指定则使用--tls-private-key-file 指定的文件
--service-account-lookup[=false]	设置为 true 时，系统会到 etcd 验证 ServiceAccount token 是否存在
--service-cluster-ip-range=<nil>	Service 的 Cluster IP（虚拟 IP）池，例如 169.169.0.0/16，这个 IP 地址池不能与物理机所在的网络重合
--service-node-port-range=	Service 的 NodePort 能使用的主机端口号范围，默认为 30000～32767，包括 30000 和 32767
--ssh-keyfile=""	如果指定，则通过 SSH 使用指定的秘钥文件对 Node 进行访问
--ssh-user=""	如果指定，则通过 SSH 使用指定的用户名对 Node 进行访问
--storage-backend=""	设置持久化存储类型，可选项为 etcd2（默认）、etcd3
--storage-media-type="application/json"	持久化存储中的保存格式，默认为 application/json。某些资源类型只能使用 application/json 格式进行保存，将忽略这个参数的设置

续表

参数名和取值示例	说　明
--storage-versions="apps/v1alpha1,authentication.k8s.io/v1beta1,authorization.k8s.io/v1beta1,autoscaling/v1,batch/v1,componentconfig/v1alpha1,extensions/v1beta1,policy/v1alpha1,rbac.authorization.k8s.io/v1alpha1,v1"	持久化存储的资源版本号，以分组形式标记，例如"group1/version1,group2/version2,..."
--tls-cert-file=""	包含 x509 证书的文件路径，用于 HTTPS 认证
--tls-private-key-file=""	包含 x509 与 tls-cert-file 对应的私钥文件路径
--token-auth-file=""	用于访问 API Server 安全端口的 token 认证文件路径
--watch-cache[=true]	设置为 true 表示将 watch 进行缓存
--watch-cache-sizes=[]	设置各资源对象 watch 缓存大小的列表，以逗号分隔，每个资源对象的设置格式为 resource#size，当 watch-cache 设置为 true 时生效

3. kube-controller-manager 启动参数

对 kube-controller-manager 启动参数的详细说明如表 2.5 所示。

表 2.5　对 kube-controller-manager 启动参数的详细说明

参数名和取值示例	说　明
--address=0.0.0.0	监听的主机 IP 地址，默认为 0.0.0.0 表示使用全部网络接口
--allocate-node-cidrs=false	设置为 true 表示使用云服务商为 Pod 分配的 CIDRs，仅用于公有云
--cloud-config=""	云服务商的配置文件路径，仅用于公有云
--cloud-provider=""	云服务商的名称，仅用于公有云
--cluster-cidr=<nil>	集群中 Pod 的可用 CIDR 范围
--cluster-name="kubernetes"	集群的名称，默认为 kubernetes
--concurrent-deployment-syncs=5	设置允许的并发同步 deployment 对象的数量，值越大表示同步操作越快，但将会消耗更多的 CPU 和网络资源
--concurrent-endpoint-syncs=5	设置并发执行 Endpoint 同步操作的数量，值越大表示同步操作越快，但将会消耗更多的 CPU 和网络资源
--concurrent-rc-syncs=5	并发执行 RC 同步操作的协程数，值越大表示同步操作越快，但将会消耗更多的 CPU 和网络资源
--concurrent-namespace-syncs=2	设置允许的并发同步 namespace 对象的数量，值越大表示同步操作越快，但将会消耗更多的 CPU 和网络资源
--concurrent-rc-syncs=5	设置允许的并发同步 replication controller 对象的数量，值越大表示同步操作越快，但将会消耗更多的 CPU 和网络资源
--concurrent-replicaset-syncs=5	设置允许的并发同步 replica set 对象的数量，值越大表示同步操作越快，但将会消耗更多的 CPU 和网络资源

续表

参数名和取值示例	说　明
--concurrent-resource-quota-syncs=5	设置允许的并发同步 resource quota 对象的数量，值越大表示更快地进行同步操作，但将会消耗更多的 CPU 和网络资源
--configure-cloud-routes[=true]	设置为 true 表示使用 allocate-node-cidrs 进行 CIDRs 的分配，仅用于公有云
--controller-start-interval=0	启动各个 controller manager 的时间间隔，默认为 0 秒
--daemonset-lookup-cache-size=1024	DaemonSet 的查询缓存大小，默认为 1024。值越大表示 DaemonSet 响应越快，内存消耗也越大
--deleting-pods-burst=10	如果一个 Node 节点失败，则会批量删除在上面运行的 Pod 实例的信息，此值定义了突发最大删除的 Pod 的数量，与 deleting-pods-qps 一起作为调度中的限流因子
--deleting-pods-qps=0.1	当 Node 失效时，每秒删除其上的多少个 Pod 实例
--deployment-controller-sync-period=30s	同步 deployments 的时间间隔，默认为 30 秒
--enable-dynamic-provisioning[=true]	设置为 true 表示启用动态 provisioning（需环境支持）
--enable-garbage-collector[=false]	设置为 true 表示启用垃圾回收机制，必须与 kube-apiserver 的该参数设置为相同的值
--enable-hostpath-provisioner[=false]	设置为 true 表示启用 hostPath PV provisioning 机制，仅用于测试，不可用于多 Node 的集群环境
--flex-volume-plugin-dir="/usr/libexec/kubernetes/kubelet-plugins/volume/exec/"	设置 flex volume 插件应搜索其他第三方 volume 插件的全路径
--horizontal-pod-autoscaler-sync-period=30s	Pod 自动扩容器的 Pod 数量的同步时间间隔，默认为 30 秒
--kube-api-burst=30	发送到 API Server 的每秒的请求数量，默认为 30
--kube-api-content-type="application/vnd.kubernetes.protobuf"	发送到 API Server 的请求内容类型
--kube-api-qps=20	与 API Server 通信的 QPS 值，默认为 20
--kubeconfig=""	kubeconfig 配置文件路径，在配置文件中包括 Master 地址信息及必要的认证信息
--leader-elect[=false]	设置为 true 表示进行 leader 选举，用于多个 Master 组件的高可用部署
--leader-elect-lease-duration=15s	leader 选举过程中非 leader 等待选举的时间间隔，默认为 15 秒，当 leader-elect=true 时生效
--leader-elect-renew-deadline=10s	leader 选举过程中在停止 leading 角色之前再次 renew 的时间间隔，应小于或等于 leader-elect-lease-duration，默认为 10 秒，当 leader-elect=true 时生效
--leader-elect-retry-period=2s	leader 选举过程中在获取 leader 角色和 renew 之间的等待时间，默认为两秒，当 leader-elect=true 时生效
--master=""	API Server 的 URL 地址，设置后不再使用 kubeconfig 中设置的值
--min-resync-period=12h0m0s	最小重新同步的时间间隔，实际重新同步的时间为 MinResyncPeriod（默认为 12 小时）到 2×MinResyncPeriod（默认 24 小时）之间的一个随机数
--namespace-sync-period=5m0s	namespace 生命周期更新的同步时间间隔，默认为 5 分钟
--node-cidr-mask-size=24	Node CIDR 的子网掩码设置，默认为 24

续表

参数名和取值示例	说　明
--node-monitor-grace-period=40s	监控 Node 状态的时间间隔，默认为 40 秒，超过该设置时间后，controller-manager 会把 Node 标记为不可用状态。此值的设置有如下要求： 它应该被设置为 kubelet 汇报的 Node 状态时间间隔（参数—node-status-update-frequency=10s）的 *N* 倍，*N* 为 kubelet 状态汇报的重试次数
--node-monitor-period=5s	同步 NodeStatus 的时间间隔，默认为 5 秒
--node-startup-grace-period=1m0s	Node 启动的最大允许时间，超过此时间无响应则会标记 Node 为不可用状态（启动失败），默认为 1 分钟
--node-sync-period=10s	Node 信息发生变化时（例如新 Node 加入集群）controller-manager 同步各 Node 信息的时间间隔，默认为 10 秒
--pod-eviction-timeout=5m0s	在发现一个 Node 失效以后，延迟一段时间，在超过这个参数指定的时间后，删除此 Node 上的 Pod，默认为 5 分钟
--port=10252	controller-manager 监听的主机端口号，默认为 10252
--profiling=true	打开性能分析，可以通过<host>:<port>/debug/pprof/地址查看程序栈、线程等系统运行信息
--pv-recycler-increment-timeout-nfs=30	使用 nfs scrubber 的 Pod 每增加 1Gi 空间在 ActiveDeadlineSeconds 上增加的时间，默认为 30 秒
--pv-recycler-minimum-timeout-hostpath=60	使用 hostPath recycler 的 Pod 的最小 ActiveDeadlineSeconds 秒数，默认为 60 秒。实验用
--pv-recycler-minimum-timeout-nfs=300	使用 nfs recycler 的 Pod 的最小 ActiveDeadlineSeconds 秒数，默认为 300 秒
--pv-recycler-pod-template-filepath-hostpath=""	使用 hostPath recycler 的 Pod 的模板文件全路径，仅用于实验
--pv-recycler-pod-template-filepath-nfs=""	使用 nfs recycler 的 Pod 的模板文件全路径
--pv-recycler-timeout-increment-hostpath=30	使用 hostPath scrubber 的 Pod 每增加 1Gi 空间在 ActiveDeadlineSeconds 上增加的时间，默认为 30 秒。实验用
--pvclaimbinder-sync-period=15s	同步 PV 和 PVC（容器声明的 PV）的时间间隔
--replicaset-lookup-cache-size=4096	设置 replica sets 查询缓存的大小，默认为 4096，值越大表示查询操作越快，但将会消耗更多的内存
--replication-controller-lookup-cache-size=4096	设置 replication controller 查询缓存的大小，默认为 4096，值越大表示查询操作越快，但将会消耗更多的内存
--resource-quota-sync-period=5m0s	resource quota 使用信息同步的时间间隔，默认为 5 分钟
--root-ca-file=""	根 CA 证书文件路径，将被用于 Service Account 的 token secret 中
--service-account-private-key-file=""	用于给 Service Account token 签名的 PEM-encoded RSA 私钥文件路径
--service-cluster-ip-range=""	Service 的 IP 范围
--service-sync-period=5m0s	同步 service 与外部 load balancer 的时间间隔，默认为 5 分钟
--terminated-pod-gc-threshold=12500	设置可保存的终止 Pod 的数量，超过该数量，垃圾回收器将开始进行删除操作。设置为不大于 0 的值表示不启用该功能

4. kube-scheduler 启动参数

对 kube-scheduler 启动参数的详细说明如表 2.6 所示。

表 2.6　对 kube-scheduler 启动参数的详细说明

参数名和取值示例	说　明
--address=0.0.0.0	监听的主机 IP 地址，默认为 0.0.0.0 表示使用全部网络接口
--algorithm-provider="DefaultProvider"	设置调度算法，默认为 DefaultProvider
--failure-domains="kubernetes.io/hostname,failure-domain.beta.kubernetes.io/zone,failure-domain.beta.kubernetes.io/region"	表示 Pod 调度时的亲和力参数。在调度 Pod 时，如果两个 Pod 有相同的亲和力参数，那么这两个 Pod 会被调度到相同的 Node 上；如果两个 Pod 有不同的亲和力参数，那么这两个 Pod 不会被调度到相同的 Node 上
--hard-pod-affinity-symmetric-weight=1	表示 Pod 调度规则亲和力的权重值，取值范围为 0～100。RequiredDuringScheduling 亲和性是非对称的，但对每一个 RequiredDuringScheduling 亲和性都存在一个对应的隐式 PreferredDuringScheduling 亲和性规则。该设置表示隐式 PreferredDuringScheduling 亲和性规则的权重值，默认为 1
--kube-api-burst=100	发送到 API Server 的每秒请求数量，默认为 100
--kube-api-content-type="application/vnd.kubernetes.protobuf"	发送到 API Server 的请求内容类型
--kube-api-qps=50	与 API Server 通信的 QPS 值，默认为 50
--kubeconfig=""	kubeconfig 配置文件路径，在配置文件中包括 Master 的地址信息及必要的认证信息
--leader-elect[=false]	设置为 true 表示进行 leader 选举，用于多个 Master 组件的高可用部署
--leader-elect-lease-duration=15s	leader 选举过程中非 leader 等待选举的时间间隔，默认为 15 秒，当 leader-elect=true 时生效
--leader-elect-renew-deadline=10s	leader 选举过程中在停止 leading 角色之前再次 renew 的时间间隔，应小于或等于 leader-elect-lease-duration，默认为 10 秒，当 leader-elect=true 时生效
--leader-elect-retry-period=2s	leader 选举过程中获取 leader 角色和 renew 之间的等待时间，默认为两秒，当 leader-elect=true 时生效
--master=""	API Server 的 URL 地址，设置后不再使用 kubeconfig 中设置的值
--policy-config-file=""	调度策略（scheduler policy）配置文件的路径
--port=10251	scheduler 监听的主机端口号，默认为 10251
--profiling=true	打开性能分析，可以通过 <host>:<port>/debug/pprof/地址查看栈、线程等系统运行信息
--scheduler-name="default-scheduler"	调度器名称，用于选择哪些 Pod 将被该调度器进行处理，选择的依据是 Pod 的 annotation 设置，包含 key='scheduler.alpha.kubernetes.io/name'的 annotation

5. kubelet 启动参数

对 kubelet 启动参数的详细说明如表 2.7 所示。