第十二章　使用 kubectl 管理 Kubernetes 容器平台

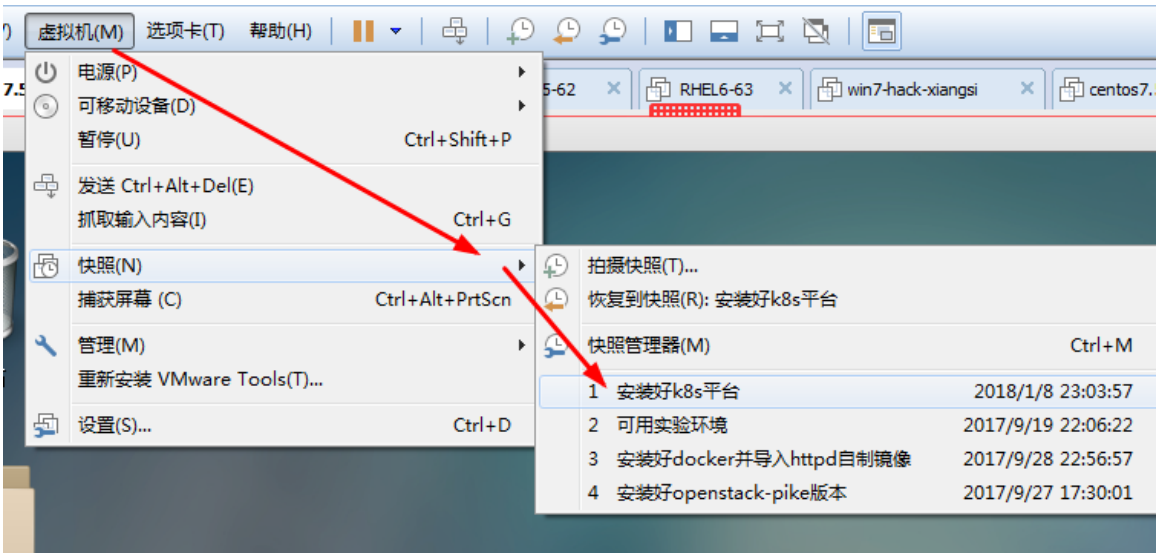
本节所讲内容：

- 12.1　kubectl 概述
- 12.2　kubectl 创建和删除一个 pod 相关操作
- 12.3　yaml 语法规则
- 12.4　kubectl create 加载 yaml 文件生成 deployment 设备资源
- 12.5　kubectl 其他常用命令和参数说明
- 12.6　使用 kubectl 管理集群中 deployment 资源和 service 服务

12.1　kubectl 概述

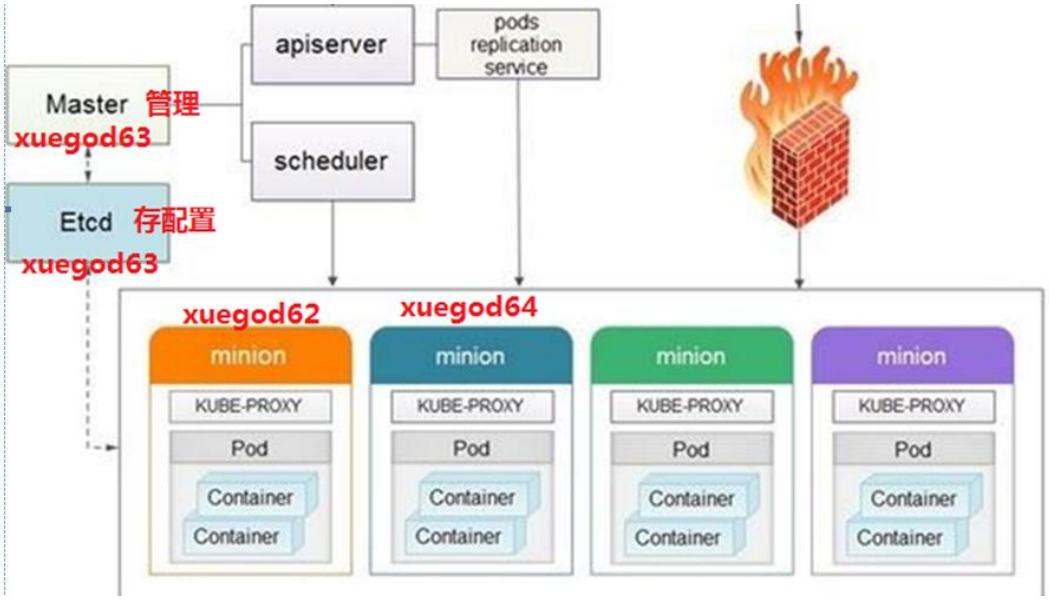
kubectl 是一个用于操作 kubernetes 集群的命令行接口，通过利用 kubectl 的各种命令可以实现各种功能。

实验环境： 在之前搭建好 K8S 集群的基础，做今天的实验。还原 3 台机器到 k8s 的快照



12.1.1　kubernetes 实验环境：使用之前已经安装好 kubernetes 集群平台

1、实验拓扑图



当前集群构成

一主两个 minion 结的 Kubernetes 集群

类型	Hostname	IP
Master/etcd	master	192.168.1.63
Node	node1	192.168.1.62
Node	node2	192.168.1.64

启动相关服务：

[root@xuegod63 ~]# systemctl restart kube-apiserver kube-controller-manager kube-scheduler flanneld

[root@master ~]# kubectl get nodes

NAME STATUS AGE

node1 Ready 50d

node2 Ready 50d

2、查看版本：

[root@master ~]# kubectl version

Client Version: version.Info{Major:"1", Minor:"5", GitVersion:"v1.5.2", GitCommit:"269f928217957e7126dc87e6adfa82242bfe5b1e", GitTreeState:"clean", BuildDate:"2017-07-03T15:31:10Z", GoVersion:"go1.7.4", Compiler:"gc", Platform:"linux/amd64"}

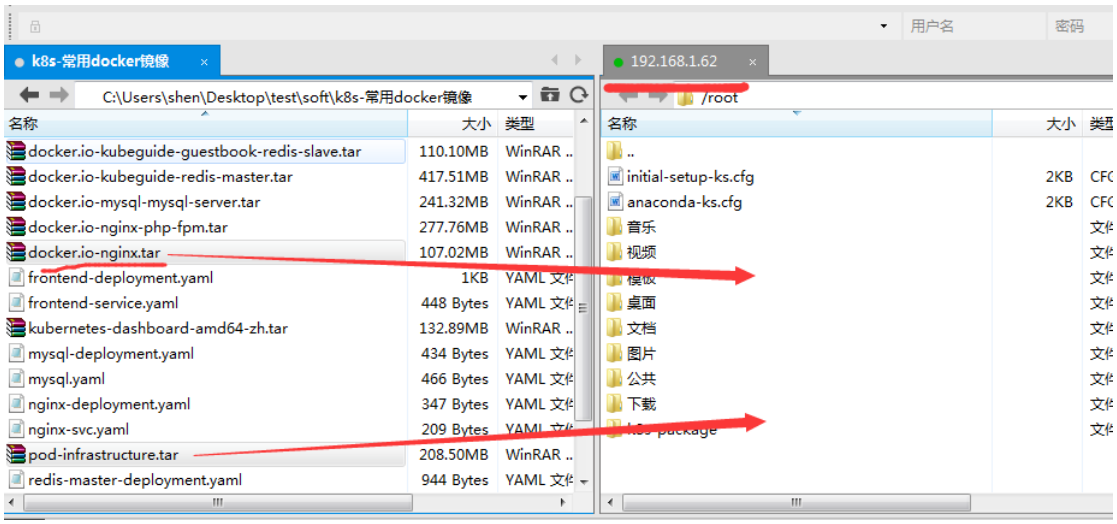
Server Version: version.Info{Major:"1", Minor:"5", GitVersion:"v1.5.2", GitCommit:"269f928217957e7126dc87e6adfa82242bfe5b1e", GitTreeState:"clean", BuildDate:"2017-07-03T15:31:10Z", GoVersion:"go1.7.4", Compiler:"gc", Platform:"linux/amd64"}

12.2 kubectl 创建和删除一个 pod 相关操作

命令	说明
run	在集群上运行一个 pod
create	使用文件或者标准输入的方式创建一个 pod
delete	使用文件或者标准输入以及资源名称或者标签选择器来删除某个 pod

12.2.1 在集群上运行一个镜像

提前将："C:\Users\shen\Desktop\test\soft\k8s-常用 docker 镜像" 中镜像：
docker.io-nginx.tar 和 pod-infrastructure.tar
上传到 node1 和 node2 上并导入镜像。
登录 node1，开始上传：



[root@node1 ~]# scp docker.io-nginx.tar pod-infrastructure.tar 192.168.1.64:/root/
如果 node1 和 node2 服务器上没有 docker.io-nginx.tar 和 pod-infrastructure.tar 镜像，后期使用时会自动在 dockerhub 上下载此镜像，这样比较慢，所以我们提前上传到服务器上。
[root@node1 ~]# docker load -i docker.io-nginx.tar
[root@node1 ~]# docker load -i pod-infrastructure.tar
[root@node2 ~]# docker load -i docker.io-nginx.tar
[root@node2 ~]# docker load -i pod-infrastructure.tar

12.2.2 kubectl run 语法

kubectl run 和 docker run 一样，kubectl run 能将一个 pod 运行起来。
语法：
kubectl run NAME --image=image [--env="key=value"] [--port=port] [--replicas=replicas]
replica ['replɪkə] 副本
开始启动 pod：
[root@master ~]# kubectl run nginx --image=docker.io/nginx --replicas=1 --port=9000

deployment "nginx" created

注：使用 docker.io/nginx 镜像 ，--port=暴露容器端口 9000 ，设置副本数 1

注： 如果 docker.io/nginx 镜像没有，那么 node1 和 node2 会自动在 dockerhub 上下载。也可以改成自己的私有仓库地址：--image=192.168.1.63:5000/nginx:1.12

kubectl run 之后，kubernetes 创建了一个 deployment

查看 Deployment

```
[root@master ~]# kubectl  get deployment
NAME       DESIRED    CURRENT    UP-TO-DATE    AVAILABLE    AGE
nginx      1          1          1             0            29s
```

查看生成的 pod，kubernetes 将容器运行在 pod 中以方便实施卷和网络共享等管理

```
[root@master ~]#  kubectl get pod
NAME                          READY      STATUS      RESTARTS    AGE
nginx-3431840295-m9xcs    1/1        Running    0           3m
```

pods 常见的状态：

1、ContainerCreating #容器创建

2、ImagePullBackOff #从后端把镜像拉取到本地

注：如果这里 pod 没有正常运行，都是因为 docker hub 没有连接上，导致镜像没有下载成功，这时，可以在 node 节点上把相关镜像手动上传一下或把 docker 源换成阿里云的。

3、terminating ['tɜ:mɪneɪtɪŋ] #终止 。当删除 pod 时的状态

4、Running 正常运行状态

12.2.3 使用 kubectl delete 删除创建的对象

1、删除 pod

```
[root@master ~]#    kubectl delete pod nginx-3431840295-m9xcs
pod "nginx-3431840295-m9xcs" deleted
```

可以看到刚刚生成的 nginx pod 正在结束(Terminating)，随之一个新的 nginx pod 正在创建，这是正是 replicas 为 1 的作用，平台会一直保证有一个副本在运行。

```
[root@master ~]#    kubectl get pod
NAME                          READY      STATUS                RESTARTS    AGE
nginx-3431840295-sfhlv    0/1        ContainerCreating    0           10s
```

过 2 分钟左右，再次确认，发现已经运行

```
[root@master ~]#    kubectl get pod
NAME                          READY      STATUS      RESTARTS    AGE
nginx-3431840295-sfhlv    1/1        Running    0           1m
```

2、删除 deployment

直接删除 pod 触发了 replicas 的确保机制，所以我需要直接删除 deployment

```
[root@master ~]# kubectl delete deployment nginx
deployment "nginx" deleted
[root@master ~]# kubectl   get pod    #pod 也被删除了
No resources found.
```

12.3 yaml 语法规则

YAML 语言（发音 /ˈjæməl/ ）的设计目标，就是方便人类读写。它实质上是一种通用的数据串行化格式。

12.3.1 yaml 语法规则

1、yaml 配置文件常见单词：

kind: 同类，类型 ； apiVersion API 版本 ； metadata 元数据 ；

spec: 规格，说明书（定义具体参数） ； replicas [ˈreplɪkəs] ： 副本

selector [sɪˈlektə(r)] 选择器 ； template [ˈtempleɪt] 模板

2、yaml 语法的基本语法规则如下：

1、大小写敏感

2、使用缩进表示层级关系

3、缩进时不允许使用 Tab 键，只允许使用空格。

4、缩进的空格数目不重要，只要相同层级的元素左侧对齐即可

5、# 表示注释，从这个字符一直到行尾，都会被解析器忽略。

6、在 yaml 里面，连续的项目（如：数组元素、集合元素）通过减号“-”来表示，map 结构里面的键值对（key/value）用冒号“:”来分割。

3、YAML 支持的数据结构有三种。

对象：键值对的集合，又称为映射（mapping）/ 哈希（hashes） / 字典（dictionary）

数组：一组按次序排列的值，又称为序列（sequence） / 列表（list）

纯量（scalars）：单个的、不可再分的值

4、数据结构--对象

对象的一组键值对，使用冒号结构表示。

例 1：animal 代表 pets # pet [pet] 宠物

```
animal: pets
```

Yaml 也允许另一种写法，将所有键值对写成一个行内对象。

例 2：hash 对象中包括 name 和 foo

```
hash:
    name: Steve
    foo: bar
```

或

```
hash: { name: Steve, foo: bar }
```

5、数组

一组连词线开头的行，构成一个数组。

```
- Cat
- Dog
- Goldfish
```

转为 JavaScript 如下。

```
[ 'Cat', 'Dog', 'Goldfish' ]
```

数据结构的子成员是一个数组，则可以在该项下面缩进一个空格。数组中还有数组。

-
  - Cat
  - Dog
  - Goldfish

转为 JavaScript 如下。

[['Cat', 'Dog', 'Goldfish']]

数组也可以采用行内表示法。

animal: [Cat, Dog]

转为 JavaScript 如下。

{ animal: ['Cat', 'Dog'] }

6、复合结构

对象和数组可以结合使用，形成复合结构。

例：编写一个包括 BAT 基本信息的 bat.yaml 配置文件

```
[root@master ~]# vim   bat.yaml    #写入以下内容
bat:
 website:
   baidu: http://www.baidu.com
   qq: http://www.qq.com
   ali:
     - http://www.taobao.com
     - http://www.tmall.com
 ceo:
   yanhongli: 李彦宏
   huatengma: 马化腾
   yunma: 马云
```

注：格式如下

```
对象 ：
   对象：
       对象：键值
       对象：
         - 数组
         - 数组
```

7、纯量

纯量是最基本的、不可再分的值。如：字符串、布尔值、整数、浮点数、Null、时间、日期

例：数值直接以字面量的形式表示。

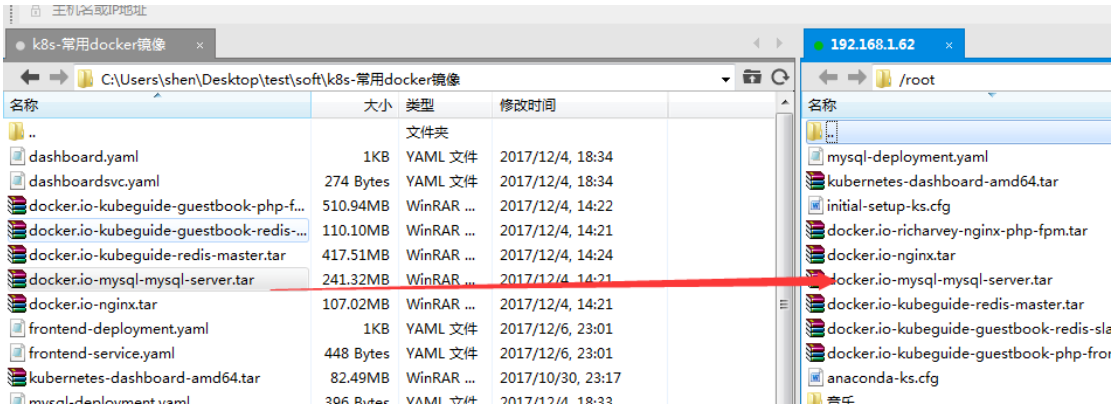
number: 12.30

12.4 kubectl create 加载 yaml 文件生成 deployment

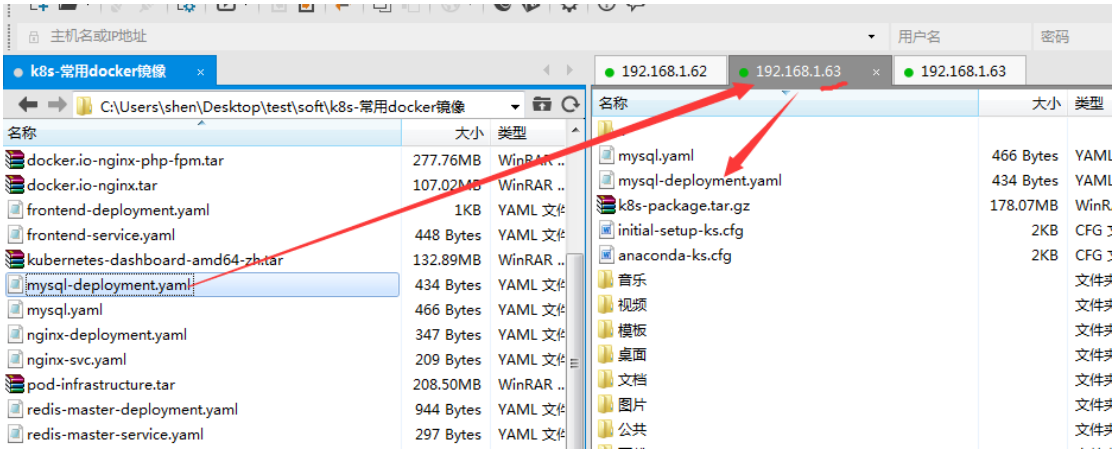
使用 kubectl run 在设定很复杂的需求时，需要非常长的一条语句，也很容易出错，也没法保存。所以更多场景下会使用 yaml 或者 json 文件。

12.4.1 生成 mysql-deployment.yaml 文件：

上传 mysql 服务器镜像 docker.io-mysql-mysql-server.tar 到 node1 和 node2 上



上传 mysql-deployment.yaml 到 xuegod63 管理结点



[root@node1 ~]# scp docker.io-mysql-mysql-server.tar 192.168.1.64:/root/

[root@node1 ~]# docker load -i docker.io-mysql-mysql-server.tar

[root@node2 ~]# docker load -i docker.io-mysql-mysql-server.tar

[root@master ~]# vim mysql-deployment.yaml #写入以下内容

```
kind: Deployment
#使用 deployment 创建一个 pod 资源 ,旧的 k8s 版本可以使用 kind: ReplicationController 来创建 pod
apiVersion: extensions/v1beta1
metadata:
  name: mysql      #deployment 的名称，全局唯一
spec:
  replicas: 1      # Pod 副本期待数量，1 表示只运行一个 pod，里面一个容器
  template:        #根据此模板创建 Pod 的副本（实例）
    metadata:
      labels:            #符合目标的 Pod 拥有此标签。默认和 name 的值一样
        name: mysql     #定义 Pod 的名字是 mysql
    spec:
      containers:        # Pod 中容器的定义部分
      - name: mysql     #容器的名称
        image: docker.io/mysql/mysql-server     #容器对应的 Docker Image 镜像
        imagePullPolicy: IfNotPresent     #默认值为 :imagePullPolicy: Always 一直从外网，
```

下载镜像，不用使用本地的。

#其他镜像下载策略参数说明：

#IfNotPresent ：如果本地存在镜像就优先使用本地镜像。这样可以直接使用本地镜像，加快启动速度。Present [ˈpreznt] 目前;现在

#Never：直接不再去拉取镜像了，使用本地的；如果本地不存在就报异常了。

```
        ports:
        - containerPort: 3306      #容器暴露的端口号
          protocol: TCP
        env:                       #注入到容器的环境变量
          - name: MYSQL_ROOT_PASSWORD      #设置 mysql root 的密码
            value: "hello123"
```

注： mysql-deployment.yaml 文件结构：

Deployment 的定义

pod 的定义

容器的定义

12.4.2 使用 mysql-deployment.yaml 创建和删除 mysql 资源

```
[root@master yamls]# kubectl create -f mysql-deployment.yaml
deployment "mysql" created
```

注：当一个目录下，有多个 yaml 文件的时候，使用 kubectl create -f 目录 的方式一下全部创建

```
[root@master tmp]# kubectl create -f yamls/
deployment "mysql" created
deployment " mysql 1" created
```

12.4.3 使用 get 参数查看 pod 详细信息：

```
[root@master ~]# kubectl   get pod
NAME                        READY      STATUS      RESTARTS    AGE
mysql-3417104986-918z7    1/1         Running    0            55s
[root@master ~]# kubectl get deployment
NAME        DESIRED    CURRENT    UP-TO-DATE    AVAILABLE    AGE
mysql       1          1          1             1            1m
```

加上-o wide 参数可以查看更详细的信息,比如看到此 pod 在哪个 node 上运行，此 pod 的集群 IP 是多少也被一并显示了

```
[root@master ~]# kubectl   get pod -o wide
NAME                        READY      STATUS      RESTARTS    AGE        IP              NODE
mysql-3417104986-918z7    1/1         Running    0            2m         10.255.92.2     node1
```

注 :10.255.92.2 这个IP地址是flannel 中定义的网段中的一个IP地址。pod通过这个IP和master进行通信

测试：

```
[root@master ~]# ping 10.255.92.2   #可以 ping 通
```

[root@master ~]# ping 10.255.92.1 #ping node1 上的 docker0 的地址，也可以通的

总结： master，node1，pod，docker，container 它们之间通信都是使用 flannel 分配的地址。也就是通过 flannel 隧道把物理上分开的主机和容器，连接在一个局域网中了。

回顾：flannel 地址的配置

设置 etcd 网络

[root@xuegod63 ~]# etcdctl mkdir /k8s/network #创建一个目录/ k8s/network 用于存储 flannel 网络信息

[root@xuegod63 ~]# etcdctl set /k8s/network/config '{"Network": "10.255.0.0/16"}'

#给/k8s/network/config 赋一个字符串的值 '{"Network": "10.255.0.0/16"}'

在这里配置的。最终是存储 etcd 中的

在 node1 查看运行 mysql docker 实例：

[root@node1 ~]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

485d9692424d docker.io/mysql/mysql-server "/entrypoint.sh mysql" 3 minutes ago Up 3 minutes (healthy) k8s_mysql.3b8a259e_mysql-3417104986-918z7_default_1df811f5-f610-11e7-b2c1-000c290283db_8ddfbcf6

总结：get 命令能够确认的信息类别：

deployments (缩写 deploy)

events (缩写 ev)

namespaces (缩写 ns)

nodes (缩写 no)

pods (缩写 po)

replicasets (缩写 rs)

replicationcontrollers (缩写 rc)

services (缩写 svc)

12.4.4 使用 describe 查看 k8s 中详细信息

describe [dɪˈskraɪb] 描述

语法： kubectl describe pod pod 名字

语法： kubectl describe node node 名字

语法： kubectl describe deployment deployment 名字

1、使用 describe 查看 pod 的详细描述信息

[root@master yamls]# kubectl describe pod mysql-3417104986-bscx9

通过这个可以查看创建 pod 时报的错误信息，可以看到以下错误：

```
rror syncing pod, skipping: failed to "StartContainer" for "mysql" with ErrImagePull
: "image pull failed for docker.io/mysql/mysql-server:latest, this may be because th
ere are no credentials on this request.  details: (error pulling image configuration
: Get https://dseasb33srnrn.cloudfront.net/registry-v2/docker/registry/v2/blobs/sha2
56/f9/f92f0896ed95f3f0cbb4c72a35b6c59dfc3c0ec4b8f991243f469962f6174fc0/data?Expires=
1522722433&Signature=YvR7XXB2Ue-Vx2exPwZpr27v9DM-5klOT1cKhxgSlOTspTPFEv5OMRIOT~fK5dh
c6ojMyyMsM6l~4cJ7ANPYOmr8nIgl2tBxkz2YwPhzGTf5Jdp8P~3OtA7ksLEKwa9ljyANPB5TePjUszM9cku
5V5TOPU1l2yWVa-6861fvmiI_&Key-Pair-Id=APKAJECH5M7VWIS5YZ6Q: net/http: TLS handshake
timeout)"
```

注：在下载镜像时，因为从外网下载，可能会报错。 所以我们可以提前把镜像上传到所有 node 节点上。

2、使用 describe 查看 node 的详细描述信息

[root@master ~]# kubectl describe node node1 #查看详细信息

3、使用 describe 查看 deployment 的详细描述信息

[root@master ~]# kubectl describe deployment mysql

12.5 kubectl 其他常用命令和参数说明

命令	说明
logs	取得 pod 中容器的 log 信息
exec	在 pod 中执行一条命令
cp	从容器拷出或向容器拷入文件
attach	Attach 到一个运行中的容器上，实时查看容器消息

实验环境：先生成一个 mysql 资源：

[root@master yamls]# kubectl create -f /root /mysql-deployment.yaml

#这个命令在之前已经生成了。这里就不用执行了。

12.5.1 kubectl logs

类似于 docker logs，使用 kubectl logs 能够取出 pod 中镜像的 log，也是故障排除时候的重要信息

[root@master ~]# kubectl get pod

[root@master yamls]# kubectl logs mysql-3417104986-2z1wt

[Entrypoint] MySQL Docker Image 5.7.20-1.1.2

[Entrypoint] Initializing database

[Entrypoint] Database initialized

Warning: Unable to load '/usr/share/zoneinfo/iso3166.tab' as time zone. Skipping it.

。。。

12.5.2 kubectl exec

exec 命令用于到 pod 中执行一条命令，到 mysql 的镜像中执行 cat /etc/my.cnf 命令

[root@master ~]# kubectl get pod

[root@master ~]# kubectl exec mysql-3417104986-918z7 cat /etc/my.cnf

例 2：使用参数 exec -it ，直接登 pod 上中

[root@master ~]# kubectl exec -it mysql-478535978-1dnm2 bash

bash-4.2# exit

12.5.3 kubectl cp

用于从容器中拷出 hosts 文件到物理机的/tmp 下

[root@master ~]# kubectl cp mysql-3417104986-2z1wt:/etc/hosts /tmp/hosts

error: unexpected EOF #报错：在文件结尾处，发生意外

unexpected [ˌʌnɪkˈspektɪd] 意外的 EOF：End Of File

排错方法

[root@master ~]# kubectl cp --help

Copy files and directories to and from containers.
Examples:
  # !!!Important Note!!!
  # Requires that the 'tar' binary is present in your container #发现想要使用 kubectl cp 你的容器实例中必须有 tar 库
  # image. If 'tar' is not present, 'kubectl cp' will fail. #如果镜像中 tar 命令不存在，那么 kubectl cp 将失败

安装 mysql pod 中安装 tar 命令：
[root@master ~]# kubectl exec -it mysql-3417104986-35rn7 bash
bash-4.2# yum install tar -y

在 pod 中创建一个文件 message.log
bash-4.2# echo "this is a message from xuegod.cn" > /tmp/message.log
bash-4.2# cat /tmp/message.log
this is a message from xuegod.cn
bash-4.2# exit

拷贝出来并确认
[root@master ~]# kubectl cp mysql-478535978-1dnm2:/tmp/message.log message.log
tar: Removing leading `/' from member names
[root@master ~]# cat message.log
this is a message from mysql-478535978-1dnm2

更改 message.log 并拷贝回 pod
[root@master ~]# echo " I am mk！！！" >>message.log
[root@master ~]# kubectl cp message.log mysql-478535978-1dnm2:/tmp/message.log
确认更改后的信息
[root@master ~]# kubectl exec mysql-478535978-1dnm2 cat /tmp/message.log
this is a message from mysql-478535978-1dnm2
I am mk！！！

12.5.4 kubectl attach

kubectl attach 用于取得 pod 中容器的实时信息，可以持续不断实时的取出消息。像 tail -f /var/log/messages 动态查看日志的作用。
kubectl logs 是一次取出所有消息，像 cat /etc/passwd
attach [əˈtætʃ] 贴上 附上
[root@master ~]# kubectl attach mysql-3417104986-35rn7

注： 到现在，所创建 nginx 和 mysql 都只是 deployment 设备硬件资源，并没有对应 service 服务，所以现在还不能直接在外网进行访问 nginx 和 mysql 服务。

12.6 使用 kubectl 管理集群中 deployment 资源和 service 服务

kubectl 相关命令参数如下：

kubectl edit 编辑服务器侧资源

kubectl replace 替换，使用 yaml 配置文件来替换正在运行中的配置参数

kubectl patch 部分更新资源相关信息

kubectl apply 使用文件或者标准输入更改配置信息

kubectl scale 重新设定 Deployment/ReplicaSet/RC/Job 的 size

kubectl autoscale Deployment/ReplicaSet/RC 的自动扩展设定

kubectl cordon 设定 node 不可使用

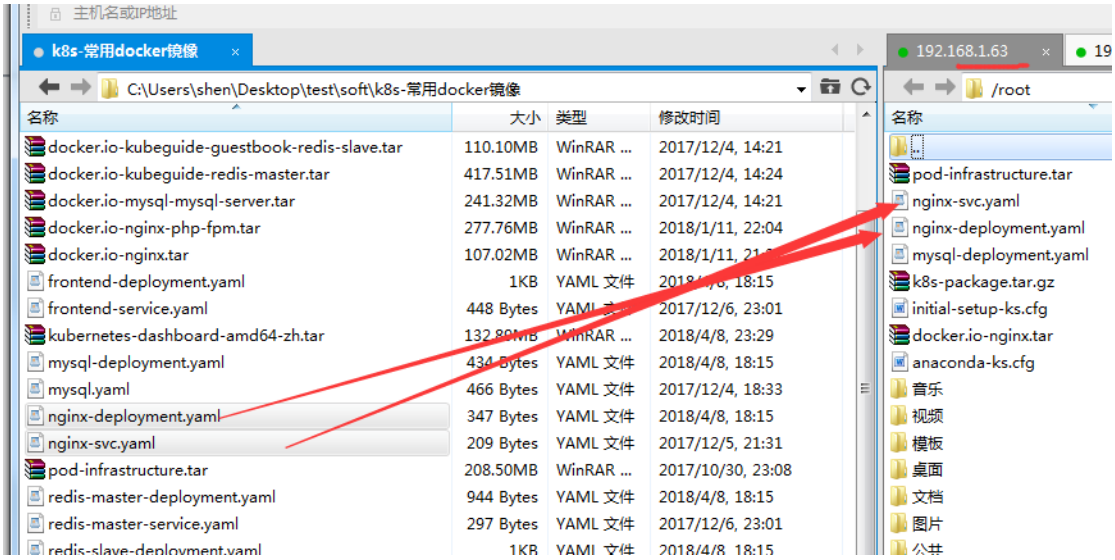
kubectl uncordon 设定 node 可以使用

kubectl drain 设定 node 进入维护模式

12.6.1 生成 nginx-deployment.yaml 资源和 nginx-svc.yaml 服务的配置文件

上传 docker.io-nginx.tar 并导入所有 node，这个操作，在之前已经作过了。这里不需要做了。

上传 nginx-deployment.yaml 资源和 nginx-svc.yaml 到 xuegod63 上。



使用 Deployment 方式启动 nginx 的 pod 和 service

[root@master ~]# vim /root/nginx-deployment.yaml #上传到 linux

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: nginx
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: nginx
    spec:
      containers:
      - name: nginx
        image:  docker.io/nginx:latest
        imagePullPolicy: IfNotPresent
```

```
        ports:
        - containerPort: 80
          protocol: TCP
[root@master ~]# vim   /root/nginx-svc.yaml      #创建 service
kind: Service
apiVersion: v1
metadata:
  name: nginx
  labels:
    name: nginx
spec:
  type: NodePort
  ports:
  - protocol: TCP
      nodePort: 31001    #后期用户可以通过 node 节点上这个端口访问 nginx，物理机的公网上的端口
    targetPort: 80    #指定是 nginx docker 容器的端口
      port: 80    # pod 端口号定义
  selector:
    name: nginx
```

注：服务端口的定义：

```
    nodePort: 31001
    port: 80    # pod 端口号定义
    targetPort: 80    #指定是 nginx docker 容器的端口
```

12.6.2 创建 deployment 和 serveice

```
[root@master ~]#   kubectl create -f nginx-deployment.yaml
[root@master ~]#   kubectl create -f nginx-svc.yaml
```

查看创建的： deployment、service、pod

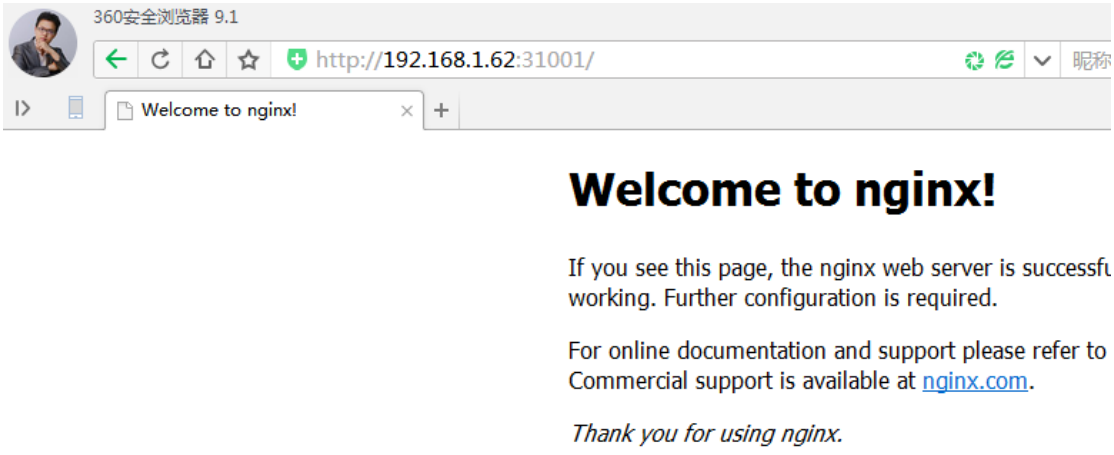
```
[root@master ~]# kubectl get deploy
NAME          DESIRED    CURRENT    UP-TO-DATE    AVAILABLE    AGE
nginx         1          1          1             1            28m
[root@master ~]#   kubectl get service
```

或：

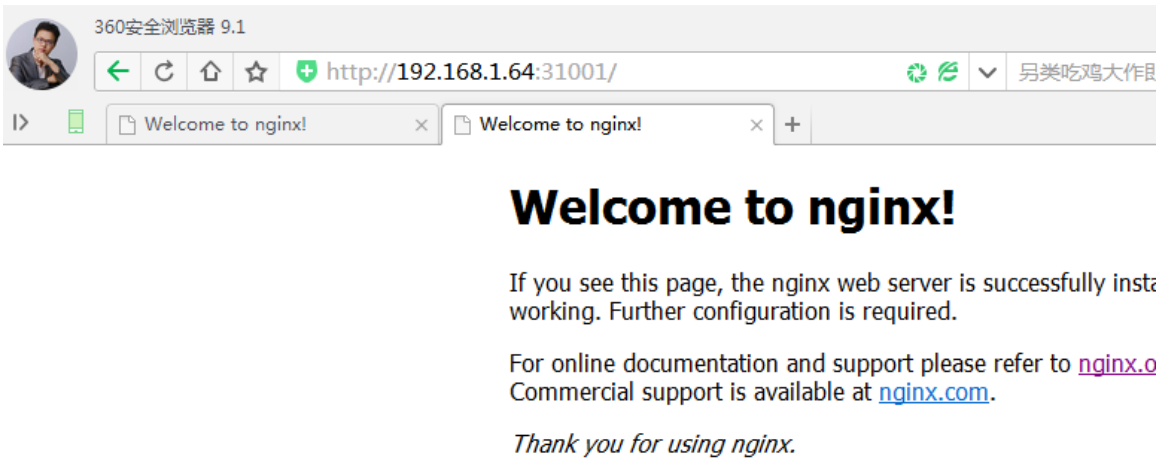
```
[root@master ~]# kubectl   get svc
NAME          CLUSTER-IP       EXTERNAL-IP    PORT(S)          AGE
kubernetes    10.254.0.1       <none>         443/TCP          54d
nginx         10.254.250.54    <nodes>        80:31001/TCP     3m
[root@master ~]# kubectl get pod -o wide    #查看详细信息
NAME                        READY     STATUS    RESTARTS   AGE       IP            NODE
nginx-1277935750-7j2p8      1/1       Running   0          34s       10.255.9.6    node1
```

注：发现 nginx 容器中的 80 端口已经映射到 node（Minion 物理机）节点的 31001 端口上了。可以查看到 nginx 服务已经运行在 node1 上。

http://192.168.1.62:31001/



尽管 ngixn 的 pod 是在 node1 运行的，但我们去访问任意 node，都可以正常访问 nginx 的。已经做了负载均衡。所以在 xueogd64 上也可以成功访问 web 服务。



修改一下默认主页：

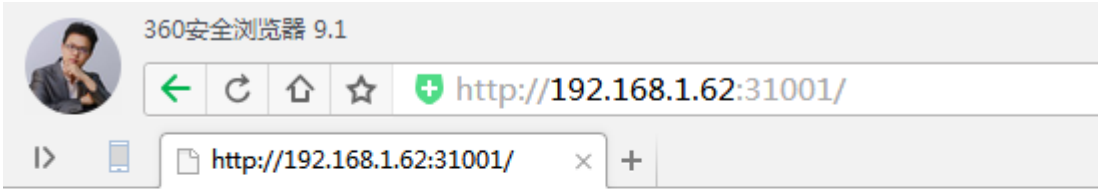
方法 1：连接到 nginx pod 中：

[root@master ~]# kubectl exec -it nginx-1277935750-7j2p8 bash

root@nginx-1277935750-7j2p8:/# cd /usr/share/nginx/html/

root@nginx-1277935750-7j2p8:/usr/share/nginx/html# echo xuegod.cn > index.html

访问：http://192.168.1.62:31001/



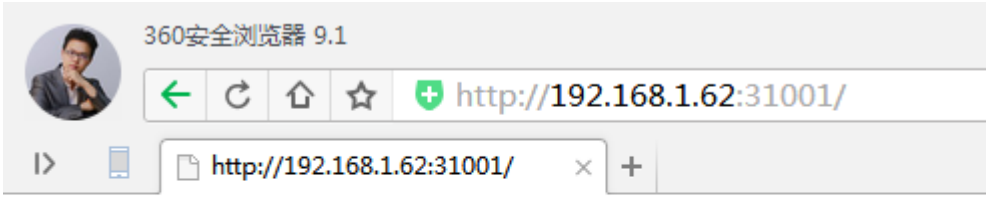
xuegod. cn

方法 2：

[root@master ~]# echo master.xuegod.cn > index.html

[root@master ~]# kubectl cp index.html nginx-1277935750-7j2p8:/usr/share/nginx/html/index.html

访问： http://192.168.1.62:31001/



master. xuegod. cn

以下维护 k8s 的命令，大家自己操作一下或了解一下就可以了

12.6.3 kubectl edit

例 1：查看 service 值。 -o 参数指定输出的消息为 yaml 类型

[root@master ~]# kubectl get service |grep nginx #查看 nginx 的 deployment 具体参数：

nginx 172.200.229.212 <nodes> 80:31001/TCP 2m

[root@master ~]# kubectl get service nginx -o yaml #查看 nginx 的 service 的配置文件

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: 2017-06-30T04:50:44Z
  labels:
    name: nginx
  name: nginx
  namespace: default
  resourceVersion: "77068"
  selfLink: /api/v1/namespaces/default/services/nginx
  uid: ad45612a-5d4f-11e7-91ef-000c2933b773
spec:
  clusterIP: 172.200.229.212
```

```
  ports:
  - nodePort: 31001    #查看到当前商品为 31001
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    name: nginx
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

edit 这条命令用于编辑服务器上的资源。

例 1：改端口 31001 为 31002

```
[root@master ~]# kubectl edit service nginx
改：19    - nodePort: 31001
为：19    - nodePort: 31002
```

编辑之后确认结果发现，此服务端口已经改变

```
[root@master ~]# kubectl get service
NAME         CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes   172.200.0.1      <none>        443/TCP        1d
nginx        172.200.229.212  <nodes>       80:31002/TCP   8m
```

修改后，发现 nginx 可以正常访问，而之前的端口已经不通

http://192.168.1.62:31002/

master.xuegod.cn

注：edit 编辑修改配置文件时，不需要停止服务。改完后立即生效

12.6.4 kubectl replace

replace 就是替换，我们使用上个例子中的 service 的 port，把它改为 31003

```
[root@master ~]# kubectl get service nginx -o yaml > nginx_replace.yaml
[root@master ~]# vim   nginx_replace.yaml
改：    - nodePort: 31002
为：    - nodePort: 31003
```

注：不能直接改成 31001

不然后：报错：

[root@master ~]# kubectl replace -f nginx_replace.yaml

The Service "nginx" is invalid: spec.ports[0].nodePort: Invalid value: 31001: provided port is already allocated

执行 replace 命令

[root@master ~]# kubectl replace -f nginx_forreplace.yaml

service "nginx" replaced

确认结果，确认之后发现 port 确实重新变成了 31003

[root@master ~]# kubectl get service

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	172.200.0.1	<none>	443/TCP	1d
nginx	10.254.64.119	<nodes>	80:31003/TCP	33m

12.6.5 kubectl patch

当修改一部分设定时，使用 patch 很方便。比如：给 pod 换个 image 镜像。

patch [pætʃ] 补丁

查看当前 port 中使用的 nginx 镜像是否能解析 php：

[root@master ~]# kubectl exec -it nginx-1277935750-w81tc bash

root@nginx-1277935750-5rsb4:/# nginx -v

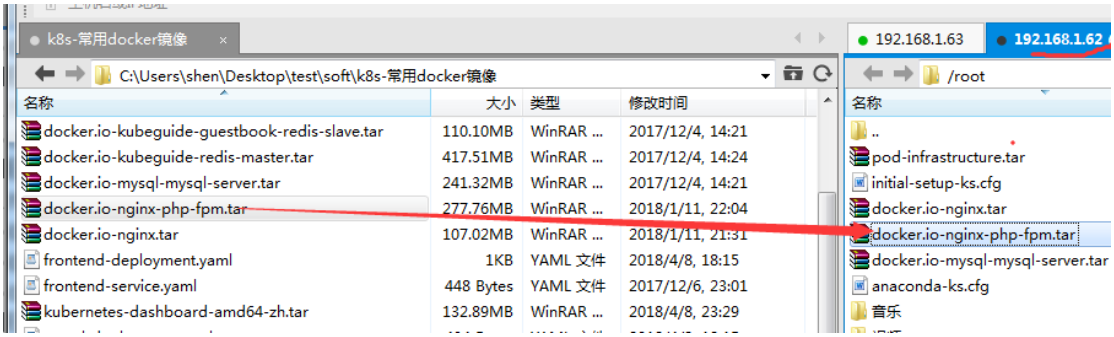
nginx version: nginx/1. 13.8

root@nginx-1277935750-5rsb4:/# php #发现不支持 php

bash: php: command not found

执行 patch 进行替换，替换成可以支持 php 的 nginx docker 镜像

上传 docker.io-nginx-php-fpm.tar 镜像到所有 node 上并导入。先传到 node1 上



[root@node1 ~]# scp docker.io-nginx-php-fpm.tar 192.168.1.64:/root/

[root@node1 ~]# docker load -i docker.io-nginx-php-fpm.tar

[root@node2 ~]# docker load -i docker.io-nginx-php-fpm.tar

开始替换镜像：

[root@master ~]# kubectl patch pod nginx-1277935750-5rsb4 -p '{"spec":{"containers":[{"name":"nginx","image":"docker.io/richarvey/nginx-php-fpm:latest"}]}}'

"nginx-1277935750-5rsb4" patched

排错经验："image":"docker.io/richarvey/nginx-php-fpm:latest" 镜像名和双引号之间不能用空格，否则会显示：

[root@master ~]# kubectl get pod

NAME	READY	STATUS	RESTARTS	AGE
nginx-1277935750-5rsb4	0/1	InvalidImageName	1	1d

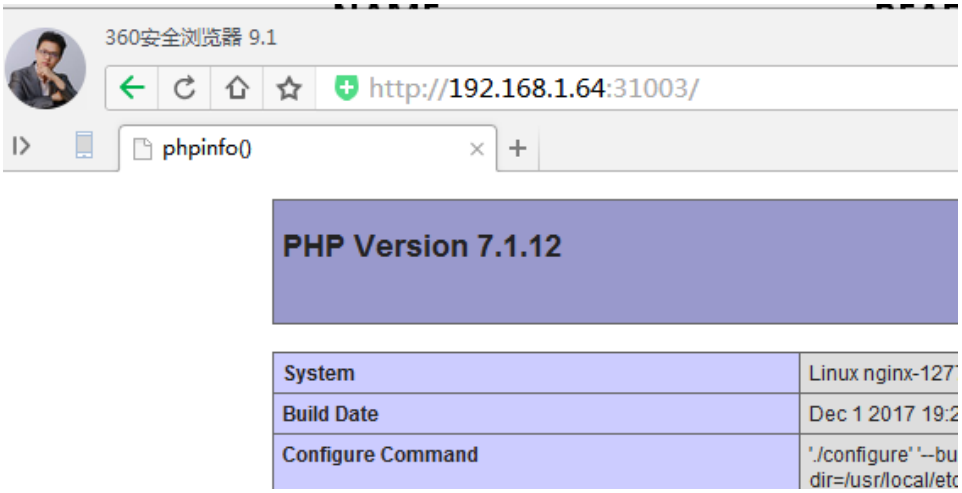
查看结果

[root@master ~]# kubectl exec nginx-1277935750-5rsb4 -it bash

bash-4.3# php -v

PHP 7.1.12 (cli) (built: Dec 1 2017 19:26:10) (NTS)

测试访问：



12.6.6 kubectl apply

apply 命令是用来使用文件或者标准输入来更改配置信息。

[root@master ~]# vim nginx-svc.yaml

改： nodePort: 31001

为： nodePort: 31004

执行 apply 命令，执行设定文件可以在运行状态修改 port 信息

[root@master ~]# kubectl apply -f nginx-svc.yaml

service "nginx" configured

[root@master ~]# kubectl get svc

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	10.254.0.1	<none>	443/TCP	56d
nginx	10.254.250.54	<nodes>	80:31004/TCP	1d

已经改变。

12.6.7 kubectl scale 、

scale [skeɪl] 规模

scale 命令用于横向扩展，是 kubernetes 或者 swarm 这类容器编辑平台的重要功能之一

实现环境： 之前已经设定 nginx 的 replica 副本为 1。

查看 nginx 现在在哪个结点上运行：

[root@master ~]# kubectl get pod -o wide

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
nginx-1277935750-5rsb4	1/1	Running	2	1d	10.255.53.2	node1

经过确认此 pod 在 node1 上运行

执行 scale 命令，使用 scale 命令进行横向扩展，将原本为 1 的副本提高到 3。

```
[root@master ~]# kubectl scale --current-replicas=1 --replicas=3 deployment/nginx
deployment "nginx" scaled
[root@master ~]# kubectl get deploy
NAME          DESIRED    CURRENT    UP-TO-DATE    AVAILABLE    AGE
nginx          3          3          3             1            1d
[root@master ~]# kubectl get pod -o wide   #查都在哪些结点上运行
```

访问：http://192.168.1.64:31004/ 和 http://192.168.1.62:31004/ 都可以访问。

12.6.8 kubectl autoscale

autoscale 命令用于自动扩展确认，跟 scale 不同的是前者还是需要手动执行，而 autoscale 则会根据负载进行调解。而这条命令则可以对 Deployment/ReplicaSet/RC 进行设定，通过最小值和最大值的指定进行设定。

```
[root@master ~]# kubectl autoscale deployment nginx --min=2 --max=5
deployment "nginx" autoscaled、、、
```

当然使用还会有一些限制，比如当前 3 个，设定最小和最大值为 2 的，会报以一下错误：

```
[root@master ~]# kubectl autoscale deployment nginx --min=2 --max=2
Error from server (AlreadyExists): horizontalpodautoscalers.autoscaling "nginx" already
exists
```

12.6.9 kubectl cordon 与 uncordon

cordon [ˈkɔ:dn] 封锁 警戒线

在实际维护的时候会出现某个 node 坏掉，或者做一些处理，暂时不能让生成的 pod 在此 node 上运行，需要通知 kubernetes 让其不要创建过来，这条命令就是 cordon，uncordon 则是取消这个要求。

之前横向扩展到 3 个副本，发现利用 roundrobin 策略每个 node 上运行起来了一个 pod

执行 cordon 命令

设定 node2 上不再运行新的 pod 实例，使用 get node 确认，其状态显示 SchedulingDisabled。

```
[root@master ~]# kubectl cordon node2
node "node2" cordoned
[root@master ~]# kubectl get nodes -o wide
NAME        STATUS                        AGE         EXTERNAL-IP
node1       Ready                         56d         <none>
node2       Ready,SchedulingDisabled      56d         <none>
```

执行 scale 命令，再次执行横向扩展命令，看是否会有 pod 漂到 node2 这台机器上，结果发现只有之前的一个 pod，再没有新的 pod 漂过去。

```
[root@master ~]# kubectl scale --replicas=6 deployment/nginx
deployment "nginx" scaled
[root@master ~]# kubectl get pod -o wide
```

注：发现新扩展的 pod 都在 node1 运行了，没有在 node2 上运行，说明封锁 node2 成功了。另外，我们指定扩展到 6 个 pod，但是只运行了 5 个，因为前面执行 autoscale 时，指定最多可以扩展到 5 个。如下图：

```
[root@master ~]# kubectl get pod -o wide
NAME                     READY     STATUS              RESTARTS   AGE       IP             NODE
mysql-3417104986-918z7   1/1       Running             2          1d        10.255.92.2    node1
nginx-1277935750-0s9kw   1/1       Running             0          8m        10.255.92.3    node1
nginx-1277935750-9341j   0/1       ContainerCreating   0          12s       <none>         node1
nginx-1277935750-cnm2w   1/1       Running             0          8m        10.255.72.3    node2
nginx-1277935750-qw90l   1/1       Running             0          12s       10.255.92.5    node1
nginx-1277935750-tcxhl   1/1       Running             0          12s       10.255.92.4    node1
nginx-1277935750-w81tc   1/1       Running             1          1h        10.255.72.2    node2
[root@master ~]#
```

执行 uncordon 解除封锁命令

使用 uncordon 命令解除对 node2 机器的限制，通过 get node 确认状态也已经正常。

[root@master ~]# kubectl uncordon node2

node "node2" uncordoned

[root@master ~]# kubectl get nodes -o wide

NAME STATUS AGE EXTERNAL-IP

node1 Ready 56d <none>

node2 Ready 56d <none> [root@master ~]#

执行 scale 命令

再次执行 scale 命令，发现有新的 pod 可以创建到 134node 上了。

[root@master ~]# kubectl scale --replicas=10 deployment nginx

12.6.10 kubectl drain 命令 [dreɪn] 排水

用于对某个 node 结点进行维护。

drain 两个作用：

1. 设定此 node 不可以使用（cordon）

2. evict 驱逐 pod 到他正常的 node 节点上

evict [ɪˈvɪkt] 驱逐

事前准备

[root@master ~]# kubectl delete deployment nginx #删除之前的 nginx 部署

将 nginx 的副本设定为 4，发现 node1 和 node2 各启动了两个 pod。

[root@master ~]# kubectl create -f /root/nginx-deployment.yaml

[root@master ~]# kubectl scale --replicas=4 deployment nginx

deployment "nginx" scaled

[root@master ~]# kubectl get pod -o wide

```
[root@master ~]# kubectl get pods -o wide
NAME                     READY     STATUS    RESTARTS   AGE       IP             NODE
mysql-3417104986-918z7   1/1       Running   2          1d        10.255.92.2    node1
nginx-1277935750-4mhk1   1/1       Running   0          10s       10.255.92.3    node1
nginx-1277935750-68h64   1/1       Running   0          19s       10.255.72.2    node2
nginx-1277935750-7j3z2   1/1       Running   0          10s       10.255.92.4    node1
nginx-1277935750-jk823   1/1       Running   0          10s       10.255.72.3    node2
[root@master ~]#
```

执行 drain 命令，让 node2 不可用，并把 node2 上已经运行 pod 驱逐到其他 node1 上:

[root@master ~]# kubectl drain node2

node "node2" cordoned

pod "nginx-1277935750-hc555" evicted

pod "nginx-1277935750-h82dc" evicted

node "node2" drained

结果确认

把 node2 上的 pod 删除后，k8s 会根据 replicas 的机制，在清退 pod 和设定 node 时，在其他结点上生成新的 pod。这时就可以对 node2 节点进行维护了，维护好后，重新 uncordon 即可。

[root@master ~]# kubectl get pod -o wide

```
[root@master ~]#  kubectl get pods -o wide
NAME                     READY     STATUS    RESTARTS   AGE       IP             NODE
mysql-3417104986-918z7   1/1       Running   2          1d        10.255.92.2    node1
nginx-1277935750-4mhk1   1/1       Running   0          1m        10.255.92.3    node1
nginx-1277935750-7j3z2   1/1       Running   0          1m        10.255.92.4    node1
nginx-1277935750-kbhrr   1/1       Running   0          13s       10.255.92.6    node1
nginx-1277935750-kgz0v   1/1       Running   0          13s       10.255.92.5    node1
```

[root@master ~]# kubectl get nodes -o wide

NAME	STATUS	AGE	EXTERNAL-IP
node1	Ready	56d	<none>
node2	Ready,SchedulingDisabled	56d	<none>

等修改好后，现在恢复可用：

[root@master ~]# kubectl uncordon node2

总结：

12.1 kubectl 概述

12.2 kubectl 创建和删除一个 pod 相关操作

12.3 yaml 语法规则

12.4 kubectl create 加载 yaml 文件生成 deployment 设备资源

12.5 kubectl 其他常用命令和参数说明

12.6 使用 kubectl 管理集群中 service 服务和 deployment 资源