Kubernetes(K8s)-k8s资源清单

一、k8s资源指令

1、基础操作

```
#创建且运行一个pod
#deployment、rs、pod被自动创建
kubectl run my-nginx --image=nginx --port=80

#增加创建副本数量
kubectl scale deployment/my-nginx --replicas = 3

#添加service
#kubectl expose将RC、Service、Deployment或Pod作为新的Kubernetes Service公开。
 kubectl expose deployment/my-nginx --port=30000 --target-port=80

 #编辑service配置文件
 kubectl edit svc/my-nginx

#其他的基础指令
#查看集群中有几个Node
kubectl get nodes

# 查看pod
kubectl  get pods

# 查看服务详情信息
kubectl describe pod my-nginx-379829228-cwlbb

# 查看已部署
[root@jackhu ~]# kubectl  get deployments

# 删除pod
[root@jackhu ~]# kubectl delete pod my-nginx-379829228-cwlbb

# 删除部署的my-nginx服务。彻底删除pod
[root@jackhu ~]# kubectl delete deployment my-nginx
deployment "my-nginx" deleted

# 删除service服务
kubectl delete service my-nginx
```

2、命令手册

kubenetes命令手册，详情请查询下表：

类型	命令	描述
基础命令	create	通过文件名或标准输入创建资源
ecpose	将一个资源公开为一个新的Service	
run	在集群中运行一个特定的镜像	
set	在对象上设置特定的功能	
get	显示一个或多个资源	
explain	文档参考资料	
edit	使用默认的编辑器编辑一个资源	
delete	通过文件名，标准输入，资源名称或者标签选择器来删除资源	
部署命令	rollout	管理资源的发布
rolling-update	对给定的复制控制器滚动更新	
scale	扩容会缩容Pod数量，Deployment，ReplicaSet，RC或Job	
autoscale	创建一个自动选择扩容或缩容并设置Pod数量	
集群管理命令	certificate	修改证书资源
cluster-info	显示集群信息	
top	显示资源（CPU/Memory/Storage)使用，需要Heapster运行	
cordon	标记节点不可调	
uncordon	标记节点可调度	
drain	驱逐节点上的应用，准备下线维护	
taint	修改节点taint标记	
故障诊断和调试命令	describe	显示特定资源或资源组的详细信息
logs	在一个Pod中打印一个容器日志，如果Pod只有一个容器，容器名称是可选的	
attach	附加到一个运行的容器	
exec	执行命令到容器	
port-forward	转发一个或多个本地端口到一个pod	
proxy	运行一个proxy到Kubernetes API server	

类型	命令	描述
cp	拷贝文件或者目录到容器中	
auth	检查授权	
高级命令	apply	通过文件名或标准输入对资源应用配置
patch	使用补丁修改，更新资源的字段	
replace	通过文件名或标准输入替换一个资源	
convert	不同的API版本之间转换配置文件	
设置命令	label	更新资源上的标签
annotate	更新资源上的注释	
completion	用于实现kubectl工具自动补全	
其他命令	api-versions	打印受支持的API 版本
config	修改kubeconfig文件（用于访问API，比如配置认证信息）	
help	所有命令帮助	
plugin	运行一个命令插件	
version	打印客户端和服务版本信息	

二、资源清单

1、required

必须存在的属性【创建资源清单的时候没有这些属性的存在它是不允许被执行的】

参数名称	字段类型	说明
version	String	这里是指的是K8SAPI的版本，目前基本上是v1，可以用kubectl api-version命令查询
kind	String	这里指的是yam文件定义的资源类型和角色，比如：Pod
metadata	Object	元数据对象，固定值就写metadata
metadata.name	String	元数据对象的名字，这里由我们编写，比如命名Pod的名字
metadata.namespace	String	元数据对象的命名空间，由我们自身定义，如果不定义的话则默认是default名称空间
Spec	Object	详细定义对象，固定值就写Spec
spec.containers[]	List	这里是Spec对象的容器列表定义，是个列表
spec.containers[].name	String	这里定义容器的名字
spec.containers[].image	String	这里定义要用到的镜像名称

2、optional

主要属性【这些属性比较重要，如果不指定的话系统会自动补充默认值】

参数名称	字段类型	说明
spec.containers[].name	String	这里定义容器的名字
spec.containers[].image	String	这里定义要用到的镜像名称
spec.containers[].imagePullPolicy	String	定义镜像拉取策略，有Always、Never、IfNotPresent三个值可选（1）Always:意思是每次都尝试重新拉取镜像（2）Never:表示仅使用本地镜像（3）IfNotPresent:如果本地有镜像就使用本地镜像，没有就拉取在线镜像。上面三个值都没设置的话，默认是Always。
spec.containers[].command[]	List	指定容器启动命令，因为是数组可以指定多个，不指定则使用镜像打包时使用的启动命令。
spec.containers[].args[]	List	指定容器启动命令参数，因为是数组可以指定多个。
spec.containers[].workingDir	String	指定容器的工作目录，进入容器时默认所在的目录
spec.containers[].volumeMounts[]	List	指定容器内部的存储卷配置

参数名称	字段类型	说明
spec.containers[].volumeMounts[].name	String	指定可以被容器挂载的存储卷的名称
spec.containers[].volumeMounts[].mountPath	String	指定可以被容器挂载的存储卷的路径
spec.containers[].volumeMounts[].readOnly	String	设置存储卷路经的读写模式，true或者false，默认为读写模式
spec.containers[].ports[]	List	指定容器需要用到的端口列表
spec.containers[].ports[].name	String	指定端口名称
spec.containers[].ports[].containerPort	String	指定容器需要监听的端口号
spec.containers[].ports[].hostPort	String	指定容器所在主机需要监听的端口号，默认跟上面containerPort相同，注意设置了hostPort同一台主机无法启动该容器的相同副本（因为主机的端口号不能相同，这样会冲突)
spec.containers[].ports[].protocol	String	指定端口协议，支持TCP和UDP，默认值为 TCP
spec.containers[].env[]	List	指定容器运行前需设置的环境变量列表
spec.containers[].env[].name	String	指定环境变量名称
spec.containers[].env[].value	String	指定环境变量值
spec.containers[].resources	Object	指定资源限制和资源请求的值（这里开始就是设置容器的资源上限）
spec.containers[].resources.limits	Object	指定设置容器运行时资源的运行上限
spec.containers[].resources.limits.cpu	String	指定CPU的限制，单位为core数，将用于docker run --cpu-shares参数这里前面文章 Pod资源限制有讲过）
spec.containers[].resources.limits.memory	String	指定MEM内存的限制，单位为MIB、GiB
spec.containers[].resources.requests	Object	指定容器启动和调度时的限制设置
spec.containers[].resources.requests.cpu	String	CPU请求，单位为core数，容器启动时初始化可用数量

参数名称	字段类型	说明
spec.containers[].resources.requests.memory	String	内存请求，单位为MIB、GiB，容器启动的初始化可用数量

3、other

额外的的一些属性。

参数名称	字段类型	说明
spec.restartPolicy	String	定义Pod的重启策略，可选值为Always、OnFailure，默认值为Always。1.Always:Pod一旦终止运行，则无论容器是如何终止的，kubelet服务都将重启它。2.OnFailure:只有Pod以非零退出码终止时，kubelet才会重启该容器。如果容器正常结束（退出码为0），则kubelet将不会重启它。3.Never:Pod终止后，kubelet将退出码报告给Master，不会重启该Pod。
spec.nodeSelector	Object	定义Node的Label过滤标签，以key:value格式指定，选择node节点
去运行		
spec.imagePullSecrets	Object	定义pull镜像时使用secret名称，以name:secretkey格式指定
spec.hostNetwork	Boolean	定义是否使用主机网络模式，默认值为false。设置true表示使用宿主机网络，不使用docker0网桥，同时设置了true将无法在同一台宿主机上启动第二个副本。

查看资源有那些资源清单属性，使用以下命令

```
# 查询所有的资源清单资源
kubectl explain pod
# 查看属性说明
kubectl explain pod.apiVersion
```

4、资源清单格式

```
#如果没有给定group名称，那么默认为core，可以使用kubectlapi-versions命令获取当前k8s版本上所有的apiversion版本信息（每个版本可能不同）
apiVersion: group/apiversion
#资源类别
kind:
#资源元数据
metadata:
  name:
  namespace:
  lables:
  annotations: #主要目的是方便用户阅读查找
spec: #期望的状态（disired state）
status: #当前状态，本字段由Kubernetes自身维护，用户不能去定义
```

5、常用命令

```
#获取apiVersion版本信息
kubectl api-versions
#获取资源的apiVersion的版本信息(以pod为例)，该命令同时输出属性设置帮助文档
kubectl explain pod

# 字段配置格式说明
apiVersion <string> #表示字符串类型
metadata <Object> #表示需要嵌套多层字段
labels <map[string]string> #表示由k：v组成的映射
finalizers <[]string> #表示字串列表
ownerReferences <[]Object>#表示对象列表
hostPID <boolean> #布尔类型
priority <integer> #整型
name <string> -required- #如果类型后面接-required-，表示为必填字段

#通过yaml文件创建pod
kubectl create -f xxx.yaml

#使用 -o 参数 加 yaml，可以将资源的配置以yaml的格式输出出来，也可以使用json，输出为json格式
kubectl get pod {podName} -o yaml
```

三、部署实例

1、nginx

1）创建deployment

```
#tomcat服务部署
apiVersion: v1
kind: ReplicationController
metadata:
  name: myweb
spec:
```

```
  replicas: 2
  selector:
    app: myweb
  template:
    metadata:
      labels:
        app: myweb
    spec:
      containers:
        - name: myweb
          image: docker.io/kubeguide/tomcat-app:v1
          ports:
          - containerPort: 8080
          env:
          - name: MYSQL_SERVICE_HOST
            value: 'mysql'
          - name: MYSQL_SERVICE_PORT
            value: '3306'



#创建deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deploy
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
      release: stabel
  template:
    metadata:
      labels:
        app: myapp
        release: stabel
        env: test
    spec:
      containers:
      - name: myapp
        image: nginx:v1
        imagePullPolicy: IfNotPresent
        ports:
        - name: http
          containerPort: 80


#创建service服务
apiVersion: v1
kind: Service
metadata:
  name: myweb
  namespace: default
spec:
  type: ClusterIP
```

```
  selector:
    app: myapp
    release: stabel
  ports:
  - name: http
    port: 80
    targetPort: 80

#创建pod
kubectl create[apply] -f xx.yaml
#创建成功后，发现报错：因为在这个pod中创建了2个容器，但是此2个容器出现了端口冲突
#查看原因：
kubectl describe pod my-app
# 查询某个容器的日志
kubectl log my-app -c test
```

2）创建tomcat-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: myweb
spec:
  type: NodePort
  ports:
    - port: 8080
      targetPort: 8080
      nodePort: 30088
  selector:
    app: myweb
```

3、eureka部署

1) deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myweb-deployment
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myweb
      release: stabel
  template:
    metadata:
      labels:
        app: myweb
        release: stabel
        env: test
```

```
    spec:
      containers:
      - name: myweb
        image: myweb:v1
        imagePullPolicy: IfNotPresent
        ports:
        - name: http
          containerPort: 10086
```

2) svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: web
  namespace: default
spec:
  type: NodePort
  selector:
    app: myweb
    release: stabel
  ports:
  - name: http
    port: 80
    targetPort: 10086
```