Kubernetes(K8s)-k8s资源清单

一、资源控制器

1、什么是控制器?

Kubernetes中内建了很多controller (控制器) ,这些相当于一个状态机,用来控制Pod的具体状态和行为

Pod 的分类

- 自主式 Pod：Pod 退出了，此类型的 Pod 不会被创建
- 控制器管理的 Pod：在控制器的生命周期里，始终要维持 Pod 的副本数目

2、常用控制器

- ReplicationController（旧版本）
- ReplicaSet
- Deployment
- DaemonSet
- Job/Cronjob

3、自主式pod

```
# 自主式的pod: 单独定义一个pod,这个没有没有副本控制器管理，也没有对应deployment
#init-pod.yaml
apiVersion: v1
kind: Pod
metadata:
 name: init-pod
 labels:
   app: myapp
spec:
 containers:
 - name: myapp
   image: hub.kaikeba.com/library/myapp:v1
#注意事项
#总结
#k8s资源对象（所有的k8s管理的资源，都叫做资源对象），都可以独立存在，但是需要根据相应原理，需求结合使用。
```

4、RC&RS

ReplicationController (RC)用来确保容器应用的副本数始终保持在用户定义的副本数,即如果有容器异常退出,会自动创建新的Pod来替代;而如果异常多出来的容器也会自动回收;

在新版本的Kubernetes中建议使用Replicaset来取代ReplicationController. ReplicaSet跟ReplicationController没有本质的不同,只是名字不一样,并且ReplicaSet支持集合式的selector;

```yaml
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
        - name: Java-nginx
          image: hub.kaikeba.com/library/myapp:v1
          env:
            - name: GET_HOSTS_FROM
              value: dns
          ports:
            - containerPort: 80
```

5、Deployment

Deployment为Pod和ReplicaSet提供了一个声明式定义(declarative)方法,用来替代以前的ReplicationController来方便的管理应用。典型的应用场景包括;

- 定义Deployment来创建Pod和ReplicaSet
- 滚动升级和回滚应用
- 扩容和缩容
- 暂停和继续Deployment

```yaml
#1)、部署一简单的Nginx应用
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

Deployment更新策略

Deployment可以保证在升级时只有一定数量的Pod是down的。默认的,它会确保至少有比期望的Pod数量少一个是up状态(最多一个不可用)

Deployment同时也可以确保只创建出超过期望数量的一定数量的Pod,默认的,它会确保最多比期望的Pod数量多一个的Pod是up的(最多1个surge)

未来的Kuberentes版本中,将从1-1变成25%-25%

```
kubect1 describe deployments
```

6、DaemonSet

```
#确保只运行一个副本，运行在集群中每一个节点上。（也可以部分节点上只运行一个且只有一个pod副本，如监控ssd硬盘）
# kubectl explain ds
# vim  filebeat.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: my-deamon
  namespace: default
  labels:
    app: daemonset
spec:
  selector:
    matchLabels:
      app: my-daemonset
  template:
    metadata:
      labels:
        app: my-daemonset
    spec:
      containers:
      - name: daemon-app
        image: hub.kaikeba.com/library/myapp:v1
```

7、Job

`Job`负责处理任务，即仅执行一次的任务，它保证批处理任务的一个或多个`Pod`成功结束。而`CronJob`则就是在`Job`上加上了时间调度。

```
# 我们用Job这个资源对象来创建一个任务，我们定一个Job来执行一个倒计时的任务，定义YAML文件：
apiVersion: batch/v1
kind: Job
metadata:
  name: job-demo
spec:
  template:
    metadata:
      name: job-demo
    spec:
```

```
      restartPolicy: Never
      containers:
      - name: counter
        image: busybox
        command:
        - "bin/sh"
        - "-c"
        - "for i in 9 8 7 6 5 4 3 2 1; do echo $i; done"

# 创建
kubectl apply -f xx.yaml
# 查询日志
kubectl logs
```

注意 Job 的 RestartPolicy 仅支持 Never 和 OnFailure 两种，不支持 Always，我们知道 Job 就相当于来执行一个批处理任务，执行完就结束了，如果支持 Always 的话是不是就陷入了死循环了？

8、cronJob

CronJob 其实就是在 Job 的基础上加上了时间调度，我们可以：在给定的时间点运行一个任务，也可以周期性地在给定时间点运行。这个实际上和我们 Linux 中的 crontab 就非常类似了。

一个 CronJob 对象其实就对应中 crontab 文件中的一行，它根据配置的时间格式周期性地运行一个 Job，格式和 crontab 也是一样的。

crontab 的格式如下：

> 分 时 日 月 星期 要运行的命令 第1列分钟0～59 第2列小时0～23） 第3列日1～31 第4列月1～12 第5列星期0～7（0和7表示星期天） 第6列要运行的命令

```
# 现在，我们用CronJob来管理我们上面的Job任务
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: cronjob-demo
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          restartPolicy: OnFailure
          containers:
          - name: hello
            image: busybox
            args:
            - "bin/sh"
            - "-c"
            - "for i in 9 8 7 6 5 4 3 2 1; do echo $i; done"

# 创建cronjob
kubctl apply -f xx.yaml
# 查询cronjob
kubectl get cronjob
# 查询jon ,cronjon会循环多个job
```

```
kubectl get job
# 实时监控查询job
kubectl get job -w
```

我们这里的`Kind`是`CronJob`了，要注意的是`.spec.schedule`字段是必须填写的，用来指定任务运行的周期，格式就和`crontab`一样，另外一个字段是`.spec.jobTemplate`,用来指定需要运行的任务，格式当然和`Job`是一致的。还有一些值得我们关注的字段`.spec.successfulJobsHistoryLimit`