链滴

kubernetes 资源控制器

作者：zhhui
原文链接：https://ld246.com/article/1603184054350
来源网站：链滴

pod分类

- 自主式pod：pod一旦退出，该类型pod不会被创建，可以理解　无管理者
- 控制器管理pod：在控制器的生命周期中始终维持pod的副本数目

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的具体状态行为

控制器类型

- ReplicationController和ReplicaSet
- Deployment
- DaemonSet
- StateFulSet
- Job/CronJob
- Horizontal Pod Autoscaling

ReplicationController和ReplicaSet

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，如果有容异常退出，会自动创建新的Pod来代替，而如果异常多出来的容器会被自动回收；

在新版本的kubernetes中建议使用ReplicaSet来取代ReplicationController。ReplicaSet跟ReplicationController没有本质不同，只是名字不一样，并且ReplicaSet支持集合式的selector，通过标签进行制的

Deployment

Deployment为pod和ReplicaSet提供了一个声明式定义（declarative）方法，代替以前的ReplicationController来方便的管理应用。典型的应用场景：

- 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

命令式编程：它侧重于如何实现程序，就像我们刚接触编程时候那样，需要把程序的实现过程按照逻结果一步步写下来；（RS， 最优创建方法：apply）

声明式编程：它侧重于定义想要什么，然后告诉计算机/引擎去帮实现；（Deployment, 最优创建方：create）

Deployment与RS的关系

```
- Deployment(nginx-deployment)
  - RS(nginx-deployment-xxxx)
```

- Pod
- Pod

DaemonSet--类似于守护进程的方案

DaemonSet确保全部（或一些）Node上运行一个Pod的副本。当有Node加入集群时，也会为它们增一个Pod，当有Node从集群移除时，这些Pod也会被回收。删除DaemonSet将会删除它创建的所Pod

使用DaemonSet的一些典型用法：

- 运行集群存储daemon，例如在每个Node上运行 glusterd、ceph
- 在每个Node上运行日志收集daemon，例如 fluentd、logstash
- 在每个Node上运行监控daemon，例如 prometheus、collectd、datadog代理、New Relic代理

Job

Job负责批处理任务，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

CronJob

CronJob管理基于时间的Job，在特定时间循环创建Job，即：

- 在给定时间点只运行一次
- 周期性的在给定时间点运行

使用前提条件：当前使用的kubernetes集群的版本 >= 1.8（对于CronJob）；对于< 1.8版本的，启API Server时，通过传递选项 --runtime-config-batch/v2alphal-true可以开启batch/v2alphal API

典型用法：

- 在给定时间点调度Job运行
- 创建周期性运行的Job，例如：数据库备份、发送邮件

StatefulSet

StatefulSet作为Controller为Pod提供唯一的标识，它可以保证部署和scale的顺序

StatefulSet是为了解决有状态服务的问题（对应Deployment和ReplicaSet是为无状态服务而设计的）

其应用场景：

- 稳定的持久化存储，即Pod重新调度后还是能访问到相同的持久化数据，基于PVC来实现
- 稳定的网络标志，即Pod重新调度后其PodName和Hostname不变，基于Headless Service（即有cluster IP的service）来实现
- 有序部署、有序扩展，即Pod是有顺序的，在部署或者扩展的时候要依据定义的顺序依次进行（即0到N-1，在下一个Pod运行之前的Pod必须是Running后Ready状态），基于init containers(init C)实现
- 有序收缩、有序删除（即从N-1到0）

Horizontal Pod Autoscaling

应用的资源使用率通常都有高峰和低谷的时候，如何削峰填谷，提高集群的整体资源利用率，让servic中的Pod个数自动调整呢？这就有有赖于Horizontal Pod Autoscaling了，顾名思义使Pod水平自动放（以控制控制器为模板的控制器）

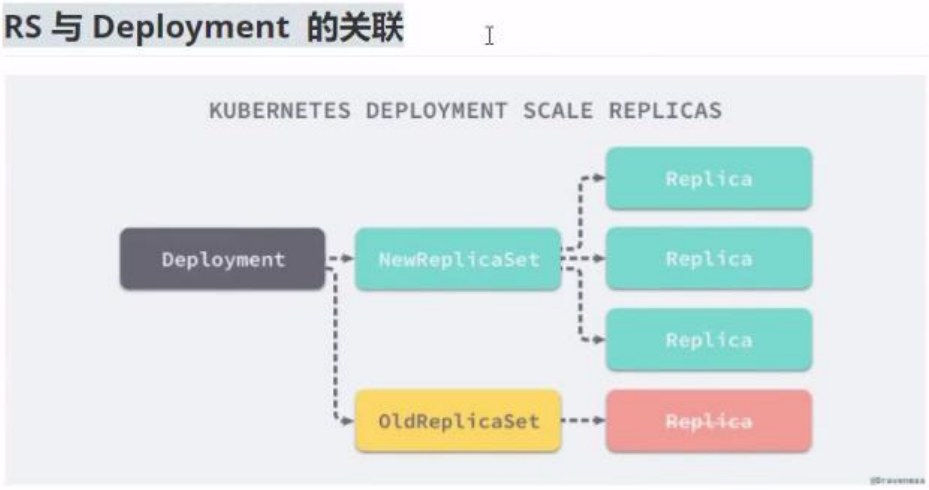
RS RC与Deployment关联

RC主要作用就是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。如果有容器异常退出，会动创建新的Pod来代替，异常退出的容器会进行回收

kubernetes官方建议使用RS（ReplicaSet）来RC进行部署，因为RS支持集合式的selector

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: frontend
spec:
  # 副本数
  replicas: 3
  # 选择标签
  selector:
    mathLabels:
      tier: frontend
  template:
    matadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: myapp
        image: test/myapp:v1
        env:
        - name: GET_HOSTS_FROM
          value: dns
        ports:
        - containerPort: 80
```

所有副本数目的监控都是以标签为基础进行监控



原文链接：kubernetes 资源控制器

示例

一、部署一个简单的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
```

创建deployment应用

```
kubectl apply -f nginx-deployment.yaml --record
```

--record参数可以记录命令，可以很方便的查看每次reversion的变化

二、扩容

```
kubectl scale deployment nginx-deployment --replicas=5
```

如果集群支持HPA，还可以为Deployment设置自动扩展

```
kubectl autoscale deployment nginx-deployment --min=3 --max=10 --cpu-percent=80
```

三、更新镜像

```
kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
```

四、回滚

```
kubectl rollout undo deployment/nginx-deployment
```

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，默认它会确保至少有比期望的Pod数少一个是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的一定数量pod。默认它会确保最多比期望的Pod数多一个的Pod是up（最多一个surge）

未来的kubernetes版本中，将从1-1变成25%-25%

```
kubectl describe deployments
```

Rollover（多个rollout并行）

假如创建一个有5个 nginx:1.7.9 replica的deployment，但是当还只有三个 nginx:1.7.9 的replica创出来的时候就开始含有5个 nginx:1.9.1 replica的Deployment。在这种情况下，Deployment会立即掉已经创建的3个 nginx:1.7.9的Pod，并开始创建 nginx:1.9.1的Pod。不会等到所有的5个 nginx:1.7.9 Pod都创建完成后才开始改变航道。

回退Deployment

```
kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
# 查看deployment回退是否完成
kubectl rollout status deployments nginx-deployment
kubectl get pods
# 查看历史版本
kubectl rollout history deployment/nginx-deployment
# 回滚到上一个版本
kubectl rollout undo deployment/nginx-deployment
# 回滚到指定历史版本
kubectl rollout undo deployment/nginx-deployment --to-revision=2 # 可以使用 --revision参指定某个历史版本
kubectl rollout pause deployment/nginx-deployment  # 暂停 deployment的更新
```

清理Policy

可以通过设置 .spec.revisionHistoryLimit项来指定deployment最多保留多少个revision历史记录。认的会保留所有的revision；如果将该项设置为0，deployment就不允许回退了

DaemonSet

DaemonSet确保全部Node上运行一个Pod的副本。

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: daemonset-example
  labels:
    app: daemonset
spec:
  selector:
    matchLabels:
      name: daemonset-example
    template:
      metadata:
        labels:
          name: daemonset-example
      spec:
        containers:
        - name: daemonset-example
          image: test/myapp:v1
```

Job

Job负责批处理任务，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

特殊说明：

- spec.template格式同Pod
- RestartPolicy仅支持Never或OnFailure
- 单个pod时，默认Pod成功运行后Job即结束
- .spec.completions标志Job结束需要成功运行的Pod个数，默认为1
- .spec.parallelism标志并运行的pod个数，默认为1
- .spec.activeDeadlineSenconds标志失败的Pod的重试最大时间，超过这个时间不会继续重试

模板

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    metadata:
      name: pi
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      resrartPolicy: Never
```

CronJob

Cron Job管理基于时间的Job，即：

- 在给定时间点只运行一次
- 周期性地在给定时间点运行

使用条件：kubernetes版本>=1.8

典型用法：

- 在给定时间点调度Job运行
- 创建周期性运行的Pod，例如：备份数据库、发送邮件

CronJob Spec

- .spec.schedule：调度，必须字段，指定任务运行周期，格式同Cron
- .spec.jobTemplate：Job模板，必须字段，指定需要运行的任务，格式同Job
- .spec.startingDeadlineSeconds：启动Job的期限（秒级别），该字段是可选的。如果因为任何原而错过了调度的时间，那么错过执行时间的Job将被认为是失败的。如果没有指定，则没有期限
- .spec.concurrencyPolicy：并发策略，该字段也是可选的。它指定了如何被处理Cron Job创建的Jo

并发执行。只允许指定下面策略中的一种：

- Allow：允许并发运行Job。（默认）
- Forbid：禁止并发运行，如果前一个还没有完成，则直接跳过下一个
- Replace：取消当前正在运行的Job，用一个新的来替换

注意：当前策略只能应用与同一个Cron Job创建的Job。如果存在多个Cron Job，它们创建的Job之总是允许并发运行

- .spec.suspend：挂起，该字段是可选的。如果设置为 true，后续所有执行都会被挂起。它对已经始执行的Job不起作用。默认为 false
- .spec.successfullJobsHistoryLimit和.spec.failedJobsHistoryLimit：历史限制，是可选字段。它指定了可以保留多少完成和失败的Job。默认情况下，它们分别设置为 3 和 1。设置限制的值为0，相类型的Job完成后将不会保留

模板

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: cronjob
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: cronjob
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Hello cronjob
          restartPolicy: OnFailure
```

查看CronJob

```
kubectl get cronjob
```

查看Job

```
kubectl get jobs
```

```
[root@master k8s-yaml]# kubectl create -f cronjob.yaml
cronjob.batch/cronjob created
[root@master k8s-yaml]# kubectl get cronjob
NAME      SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
cronjob   */1 * * * *   False     0        <none>          8s
[root@master k8s-yaml]# kubectl get job
NAME                  COMPLETIONS   DURATION   AGE
cronjob-1603212480    1/1           2s         36s
[root@master k8s-yaml]# kubectl get pod
NAME                      READY   STATUS      RESTARTS   AGE
cronjob-1603212480-ks2j2  0/1     Completed   0          43s
[root@master k8s-yaml]# kubectl logs cronjob-1603212480-ks2j2
Tue Oct 20 16:48:09 UTC 2020
Hello cronjob
[root@master k8s-yaml]#
```

删除CronJob

```
kubectl delete cronjob $(cronjob-name)
```

CronJob本身的限制：创建Job操作应该是 幂等的