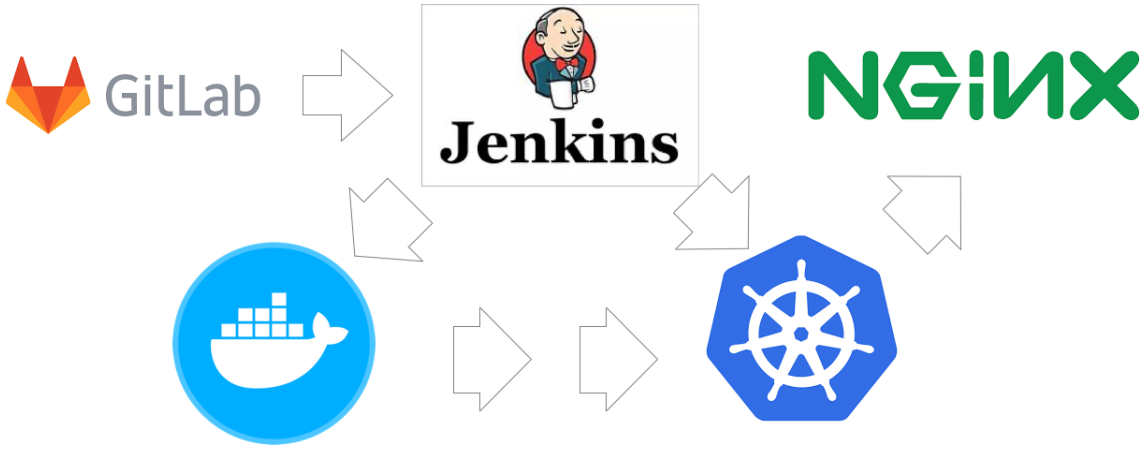
讲师(老司机)

容器虚拟化技术和自动化部署



K8S-pod控制器进阶

简介

Controller Manager 由 kube-controller-manager 和 cloud-controller-manager 组成， 是 Kubernetes 的大脑， 它通过 apiserver 监控整个集群的状态， 并确保集群处于预期的工作状态。

kube-controller-manager 由一系列的控制器组成

```
1 Replication Controller
2 Node Controller
3 CronJob Controller
4 DaemonSet Controller
5 Deployment Controller
6 Endpoint Controller
7 Garbage Collector
8 Namespace Controller
9 Job Controller
10 Pod AutoScaler
11 RelicaSet
12 Service Controller
13 ServiceAccount Controller
14 StatefulSet Controller
15 Volume Controller
16 Resource quota Controller
```

cloud-controller-manager 在 Kubernetes 启用 Cloud Provider 的时候才需要， 用来配合云服务提供商的控制， 也包括一系列的控制器

```
1 Node Controller
2 Route Controller
3 Service Controller
```

从v1.6开始，cloud provider已经经历了几次重大重构，以便在不修改Kubernetes核心代码的同时构建自定义的云服务商支持

常见Pod控制器及含义

1. ReplicaSet:适合无状态的服务部署

   用户创建指定数量的pod副本数量，确保pod副本数量符合预期状态，并且支持滚动式自动扩容和缩容功能。
   ReplicaSet主要三个组件组成：
   （1）用户期望的pod副本数量
   （2）标签选择器，判断哪个pod归自己管理
   （3）当现存的pod数量不足，会根据pod资源模板进行新建
   帮助用户管理无状态的pod资源，精确反应用户定义的目标数量，但是RelicaSet不是直接使用的控制器，而是使用Deployment。

2. deployment：适合无状态的服务部署

   工作在ReplicaSet之上，用于管理无状态应用，目前来说最好的控制器。支持滚动更新和回滚功能，还提供声明式配置。

3. StatefullSet：适合有状态的服务部署。需要学完存储卷后进行系统学习。

4. DaemonSet：一次部署，所有的node节点都会部署，例如一些典型的应用场景：

   - 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 Node Exporter

   用于确保集群中的每一个节点只运行特定的pod副本，通常用于实现系统级后台任务。比如ELK服务
   特性：服务是无状态的
   服务必须是守护进程

5. Job：一次性的执行任务。 只要完成就立即退出，不需要重启或重建。

6. Cronjob：周期性的执行任务。 周期性任务控制，不需要持续后台运行。

使用镜像

```
演示pod的控制器升级使用：
docker pull nginx:1.17.10-alpine
docker pull nginx:1.18.0-alpine
docker pull nginx:1.19.2-alpine
```

replication Controller控制器

replication controller简称RC，是kubernetes系统中的核心概念之一，简单来说，它其实定义了一个期望的场景，即声明某种pod的副本数量在任意时刻都复合某个预期值，所以RC的定义包含以下部分：

- pod期待的副本数量
- 用于筛选目标pod的Label Selector
- 当pod的副本数量小于期望值时，用于创建新的pod的pod模板（template）

ReplicaSet

ReplicationController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。

虽然ReplicaSet可以独立使用，但一般还是建议使用 Deployment 来自动管理ReplicaSet，这样就无需担心跟其他机制的不兼容问题（比如ReplicaSet不支持rolling-update但Deployment支持）。

ReplicaSet模板说明

```yaml
apiVersion: apps/v1   #api版本定义
kind: ReplicaSet   #定义资源类型为ReplicaSet
metadata:   #元数据定义
    name: myapp
    namespace: default
spec:   #ReplicaSet的规格定义
    replicas: 2   #定义副本数量为2个
    selector:       #标签选择器，定义匹配pod的标签
        matchLabels:
            app: myapp
            release: canary
    template:   #pod的模板定义
        metadata:   #pod的元数据定义
            name: myapp-pod     #自定义pod的名称
            labels:     #定义pod的标签，需要和上面定义的标签一致，也可以多出其他标签
                app: myapp
                release: canary
                environment: qa
        spec:   #pod的规格定义
            containers:   #容器定义
            - name: myapp-container   #容器名称
              image: nginx:1.17.10-alpine   #容器镜像
              ports:   #暴露端口
              - name: http
                containerPort: 80
```

可以通过kubectl命令行方式获取更加详细信息

```
kubectl explain rs
kubectl explain rs.spec
kubectl explain rs.spec.template.spec
```

部署ReplicaSet

controller/replicasetdemo.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: replicasetdemo
  labels:
    app: replicasetdemo
spec:
  replicas: 3
  template:
    metadata:
      name: replicasetdemo
      labels:
        app: replicasetdemo
    spec:
      containers:
        - name: replicasetdemo
          image: nginx:1.17.10-alpine
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
      restartPolicy: Always
  selector:
    matchLabels:
      app: replicasetdemo

```

运行ReplicaSet

```
运行ReplicaSet
kubectl apply -f replicasetdemo.yml

查看rs控制器
kubectl get rs

查看pod信息
kubectl get pod
查看pod详细信息
```

```
kubectl describe pod replicasetdemo-7fdd7b5f67-5gzfg

测试controller控制器下的pod删除、重新被controller控制器拉起
kubectl delete pod --all
kubectl get pod

修改pod的副本数量:通过命令行方式
kubectl scale replicaset replicasetdemo --replicas=8
kubectl get rs

修改pod的副本数量:通过资源清单方式
kubectl edit replicasets.apps replicasetdemo
kubectl get rs


显示pod的标签
kubectl get pod --show-labels
修改pod标签(label)
kubectl label pod replicasetdemo-652lc app=lagou --overwrite=True
再次显示pod的标签：发现多了一个pod，原来的rs中又重新拉起一个pod，说明rs是通过label去管理pod
kubectl get pod --show-labels


删除rs
kubectl delete rs replicasetdemo
```

总结

kubectl命令行工具适用于RC的绝大部分命令同样适用于ReplicaSet，此外，我们当前很少单独适用ReplicaSet，它主要被Deployment这个更高层的资源对象所使用，从而形成一整套Pod创建，删除，更新的编排机制，我们在使用Deployment时无需关心它是如何维护和创建ReplicaSet的，这一切都是自动发生的

最后，总结一下RC（ReplicaSet）的一些特性和作用：

- 在绝大多数情况下，我们通过定义一个RC实现Pod的创建及副本数量的自动控制
- 在RC里包括完整的Pod定义模板
- RC通过Label Selector机制实现对Pod副本的自动控制
- 通过改变RC里的Pod副本数量，可以实现Pod的扩容和缩容
- 通过改变RC里Pod模板中的镜像版本，可以实现滚动升级

Deployment

Deployment是kubernetes在1.2版本中引入的新概念，用于更好的解决Pod的编排问题，为此，Deployment在内部使用了ReplicaSet来实现目的，我们可以把Deployment理解为ReplicaSet的一次升级，两者的相似度超过90%

Deployment的使用场景有以下几个：

- 创建一个Deployment对象来生成对应的ReplicaSet并完成Pod副本的创建
- 检查Deployment的状态来看部署动作是否完成（Pod副本数量是否达到了预期的值）
- 更新Deployment以创建新的Pod（比如镜像升级）
- 如果当前Deployment不稳定，可以回滚到一个早先的Deployment版本
- 暂停Deployment以便于一次性修改多个PodTemplateSpec的配置项，之后在恢复Deployment，进行新的发布
- 扩展Deployment以应对高负载
- 查看Deployment的状态，以此作为发布是否成功的标志
- 清理不在需要的旧版本ReplicaSet

Deployment模板说明

可以通过kubectl命令行方式获取更加详细信息

```
kubectl explain deploy
kubectl explain deploy.spec
kubectl explain deploy.spec.template.spec
```

部署Deployment

除了API生命与Kind类型有区别，Deployment的定义与Replica Set的定义很类似。

controller/deploymentdemo.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploymentdemo1
  labels:
    app: deploymentdemo1
spec:
  replicas: 10
  template:
    metadata:
      name: deploymentdemo1
      labels:
        app: deploymentdemo1
    spec:
      containers:
        - name: deploymentdemo1
          image: nginx:1.17.10-alpine
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
      restartPolicy: Always
  selector:
    matchLabels:
      app: deploymentdemo1
```

运行Deployment

```
kubectl apply -f deploymentdemo.yml

查看deployment
kubectl get rs
查看rs:deployment名称+hashcode码组成

查看pod
kubectl get pod
```

镜像更新升级

命令行方式

```
升级nginx镜像版本为1.18.0
kubectl set image deployment deploymentdemo1 deploymentdemo1=nginx:1.18.0-alpine

查看pod升级情况
kubectl get pods -w

进去某一个pod内部，查看nginx升级版本信息
kubectl exec -it deploymentdemo1-df6bc5d4c-flc7b sh
nginx -v
exit
```

yml文件方式

```
升级nginx镜像版本为1.19.2-alpine
kubectl edit deployments.apps deploymentdemo1

查看pod升级情况
kubectl get pods -w

进去某一个pod内部，查看nginx升级版本信息
kubectl exec -it deploymentdemo1-584f6b54dd-4l62t sh
nginx -v
exit
```

Deployment扩容

命令行方式

```
kubectl scale deployment deploymentdemo1 --replicas=15

kubectl get pods
```

yml文件方式

```
kubectl edit deployments.apps deploymentdemo1

kubectl get pods
```

滚动更新

概述

微服务部署：蓝绿部署、滚动部署、灰度发布、金丝雀发布。

1. 蓝绿部署是不停老版本，部署新版本然后进行测试，确认OK，将流量切到新版本，然后老版本同时也升级到新版本。 蓝绿部署无需停机，并且风险较小。
2. 滚动发布：一般是取出一个或者多个服务器停止服务，执行更新，并重新将其投入使用。周而复始，直到集群中所有的实例都更新成新版本。 这种部署方式相对于蓝绿部署，更加节约资源——它不需要运行两个集群、两倍的实例数。我们可以部分部署，例如每次只取出集群的20%进行升级。
3. 灰度发布是指在黑与白之间，能够平滑过渡的一种发布方式。AB test就是一种灰度发布方式，让一部分用户继续用A，一部分用户开始用B，如果用户对B没有什么反对意见，那么逐步扩大范围，把所有用户都迁移到B上面来。灰度发布可以保证整体系统的稳定，在初始灰度的时候就可以发现、调整问题，以保证其影响度，而我们平常所说的金丝雀部署也就是灰度发布的一种方式。

金丝雀发布

Deployment控制器支持自定义控制更新过程中的滚动节奏，如“暂停(pause)”或“继续(resume)”更新操作。比如等待第一批新的Pod资源创建完成后立即暂停更新过程，此时，仅存在一部分新版本的应用，主体部分还是旧的版本。然后，再筛选一小部分的用户请求路由到新版本的Pod应用，继续观察能否稳定地按期望的方式运行。确定没问题之后再继续完成余下的Pod资源滚动更新，否则立即回滚更新操作。这就是所谓的金丝雀发布（Canary Release）

```
更新deployment的nginx:1.18.0-alpine版本，并配置暂停deployment
kubectl set image deployment deploymentdemo1 deploymentdemo1=nginx:1.18.0-alpine && kubectl rollout pause deployment deploymentdemo1


观察更新状态
kubectl rollout status deployments deploymentdemo1

监控更新的过程，可以看到已经新增了一个资源，但是并未按照预期的状态去删除一个旧的资源，就是因为使用了pause暂停命令
kubectl get pods -l app=deploymentdemo1 -w

确保更新的pod没问题了，继续更新
kubectl rollout resume deploy  deploymentdemo1

查看最后的更新情况
kubectl get pods -l app=deploymentdemo1 -w
```

```
16
17 进去某一个pod内部，查看nginx更新版本信息
18 kubectl exec -it deploymentdemo1-df6bc5d4c-flc7b sh
19 nginx -v
```

Deployment版本回退

默认情况下，kubernetes 会在系统中保存前两次的 Deployment 的 rollout 历史记录，以便可以随时回退（您可以修改`revision history limit`来更改保存的revision数）。

注意： 只要 Deployment 的 rollout 被触发就会创建一个 revision。也就是说当且仅当 Deployment 的 Pod template（如`.spec.template`）被更改，例如更新template 中的 label 和容器镜像时，就会创建出一个新的 revision。

其他的更新，比如扩容 Deployment 不会创建 revision——因此我们可以很方便的手动或者自动扩容。这意味着当您回退到历史 revision 时，只有 Deployment 中的 Pod template 部分才会回退。

rollout常见命令

子命令	功能说明
history	查看rollout操作历史
pause	将提供的资源设定为暂停状态
restart	重启某资源
resume	将某资源从暂停状态恢复正常
status	查看rollout操作状态
undo	回滚前一rollout

history操作

```
1 kubectl rollout history deployment deploymentdemo1
```

status操作

```
1 kubectl rollout status deployment deploymentdemo1
```

undo操作

```
回滚版本信息
kubectl rollout undo deployment deploymentdemo1

查看pod回滚情况
kubectl get pods -w

进去某一个pod内部，查看nginx回滚版本信息
kubectl exec -it deploymentdemo1-df6bc5d4c-flc7b sh
nginx -v
```

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 ）
Kuberentes 版本v1.17.5中，从1-1变成25%-25%

```
kubectl describe deployments.apps deploymentdemo1

查看到属性：
RollingUpdateStrategy:  25% max unavailable, 25% max surge

```

总结

Deployment为Pod和Replica Set（下一代Replication Controller）提供声明式更新。

只需要在 Deployment 中描述想要的目标状态是什么，Deployment controller 就会帮您将 Pod 和 ReplicaSet 的实际状态改变到您的目标状态。也可以定义一个全新的 Deployment 来创建 ReplicaSet 或者删除已有的 Deployment 并创建一个新的来替换。

Replicas（副本数量）：

.spec.replicas 是可以选字段，指定期望的pod数量，默认是1。

Selector（选择器）：

.spec.selector是可选字段，用来指定 label selector ，圈定Deployment管理的pod范围。如果被指定， .spec.selector 必须匹配 .spec.template.metadata.labels，否则它将被API拒绝。如果 .spec.selector 没有被指定， .spec.selector.matchLabels 默认是.spec.template.metadata.labels。

在Pod的template跟.spec.template不同或者数量超过了.spec.replicas规定的数量的情况下，Deployment会杀掉label跟selector不同的Pod。

Pod Template（Pod模板）：

.spec.template 是 .spec中唯一要求的字段。

.spec.template 是 pod template. 它跟 Pod有一模一样的schema，除了它是嵌套的并且不需要 apiVersion 和 kind字段。

另外为了划分Pod的范围，Deployment中的pod template必须指定适当的label（不要跟其他controller重复了，参考selector）和适当的重启策略。

.spec.template.spec.restartPolicy 可以设置为 Always , 如果不指定的话这就是默认配置。

strategy（更新策略）：

.spec.strategy 指定新的Pod替换旧的Pod的策略。 .spec.strategy.type 可以是"**Recreate**"或者是"**RollingUpdate**"。**"RollingUpdate"是默认值。**

Recreate： 重建式更新，就是删一个建一个。类似于ReplicaSet的更新方式，即首先删除现有的Pod对象，然后由控制器基于新模板重新创建新版本资源对象。

rollingUpdate：滚动更新，简单定义 更新期间pod最多有几个等。可以指定`maxUnavailable`和`maxSurge`来控制 rolling update 进程。

maxSurge：`.spec.strategy.rollingUpdate.maxSurge`是可选配置项，用来指定可以超过期望的Pod数量的最大个数。该值可以是一个绝对值（例如5）或者是期望的Pod数量的百分比（例如10%）。当`MaxUnavailable`为0时该值不可以为0。通过百分比计算的绝对值向上取整。默认值是1。

例如，该值设置成30%，启动rolling update后新的ReplicatSet将会立即扩容，新老Pod的总数不能超过期望的Pod数量的130%。旧的Pod被杀掉后，新的ReplicaSet将继续扩容，旧的ReplicaSet会进一步缩容，确保在升级的所有时刻所有的Pod数量和不会超过期望Pod数量的130%。

maxUnavailable：`.spec.strategy.rollingUpdate.maxUnavailable`是可选配置项，用来指定在升级过程中不可用Pod的最大数量。该值可以是一个绝对值（例如5），也可以是期望Pod数量的百分比（例如10%）。通过计算百分比的绝对值向下取整。 如果`.spec.strategy.rollingUpdate.maxSurge`为0时，这个值不可以为0。默认值是1。

例如，该值设置成30%，启动rolling update后旧的ReplicatSet将会立即缩容到期望的Pod数量的70%。新的Pod ready后，随着新的ReplicaSet的扩容，旧的ReplicaSet会进一步缩容确保在升级的所有时刻可以用的Pod数量至少是期望Pod数量的70%。

rollbackTo：

`.spec.rollbackTo`是一个可以选配置项，用来配置Deployment回退的配置。设置该参数将触发回退操作，每次回退完成后，该值就会被清除。

revision：`.spec.rollbackTo.revision`是一个可选配置项，用来指定回退到的revision。默认是0，意味着回退到上一个revision。

progressDeadlineSeconds：

`.spec.progressDeadlineSeconds`是可选配置项，用来指定在系统报告Deployment的failed progressing——表现为resource的状态中`type=Progressing`、`Status=False`、`Reason=ProgressDeadlineExceeded`前可以等待的Deployment进行的秒数。Deployment controller会继续重试该Deployment。未来，在实现了自动回滚后， deployment controller在观察到这种状态时就会自动回滚。

如果设置该参数，该值必须大于`.spec.minReadySeconds`。

paused：

`.spec.paused`是可以可选配置项，boolean值。用来指定暂停和恢复Deployment。Paused和没有paused的Deployment之间的唯一区别就是，所有对paused deployment中的PodTemplateSpec的修改都不会触发新的rollout。Deployment被创建之后默认是非paused。

DaemonSet

DaemonSet 确保全部Node 上运行一个 Pod 的副本。当有 Node 加入集群时，也会为他们新增一个 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有 Pod。

在每一个node节点上只调度一个Pod，因此无需指定replicas的个数，比如：

- 在每个node上都运行一个日志采集程序，负责收集node节点本身和node节点之上的各个Pod所产生的日志
- 在每个node上都运行一个性能监控程序，采集该node的运行性能数据

DaemonSet模板说明

可以通过kubectl命令行方式获取更加详细信息

```
kubectl explain daemonset
kubectl explain daemonset.spec
kubectl explain daemonset.spec.template.spec
```

部署DaemonSet

controller/daemonsetdemo.yml

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: demonsetdemo
  labels:
    app: demonsetdemo
spec:
  template:
    metadata:
      name: demonsetdemo
      labels:
        app: demonsetdemo
    spec:
      containers:
        - name: demonsetdemo
          image: nginx:1.17.10-alpine
          imagePullPolicy: IfNotPresent
```

```
      restartPolicy: Always
  selector:
    matchLabels:
      app: demonsetdemo
```

运行DaemonSet

```
运行demonset
kubectl apply -f demonsetdemo.yml

查看pod详细信息：只有工作节点创建pod，master节点并不会创建。
kubectl get pod -o wide
```

DaemonSet的滚动更新

DaemonSet有两种更新策略类型：

- OnDelete：这是向后兼容性的默认更新策略。使用 `OnDelete` 更新策略，在更新DaemonSet模板后，只有在手动删除旧的DaemonSet pod时才会创建新的DaemonSet pod。这与Kubernetes 1.5或更早版本中DaemonSet的行为相同。
- RollingUpdate：使用 `RollingUpdate` 更新策略，在更新DaemonSet模板后，旧的DaemonSet pod将被终止，并且将以受控方式自动创建新的DaemonSet pod。

Job

- 一次性执行任务，类似Linux中的job
- 应用场景：如离线数据处理，视频解码等业务

使用镜像

```
docker pull perl:slim
```

部署Job

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
        - name: pi
          image: perl:slim
          command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(6000)"]
      restartPolicy: Never
  backoffLimit: 4
```

backoffLimit说明

```
.spec.backoffLimit用于设置Job的容错次数，默认值为6。当Job运行的Pod失败次数到达.spec.backoffLimit次时，Job Controller不再新建Pod，直接停止运行这个Job，将其运行结果标记为Failure。另外，Pod运行失败后再次运行的时间间隔呈递增状态，例如10s，20s，40s。。。
```

运行Job

```
运行job
kubectl apply -f jobdemo.yml

查看pod日志
kubectl logs -f pi-7nrtv

删除job
kubectl delete -f jobdemo.yml
```

StatefulSet

在kubernetes系统中，Pod的管理对象RC，Deployment，DaemonSet和Job都面向无状态的服务，但现实中有很多服务时有状态的，比如一些集群服务，例如mysql集群，集群一般都会有这四个特点：

1. 每个节点都是有固定的身份ID，集群中的成员可以相互发现并通信
2. 集群的规模是比较固定的，集群规模不能随意变动
3. 集群中的每个节点都是有状态的，通常会持久化数据到永久存储中
4. 如果磁盘损坏，则集群里的某个节点无法正常运行，集群功能受损

如果你通过RC或Deployment控制Pod副本数量来实现上述有状态的集群，就会发现第一点是无法满足的，因为Pod名称和ip是随机产生的，并且各Pod中的共享存储中的数据不能都动，因此StatefulSet在这种情况下就派上用场了，那么StatefulSet具有以下特性：

- StatefulSet里的每个Pod都有稳定，唯一的网络标识，可以用来发现集群内的其它成员，假设，StatefulSet的名称为lagou，那么第1个Pod叫lagou-0，第2个叫lagou-1，以此类推
- StatefulSet控制的Pod副本的启停顺序是受控的，操作第N个Pod时，前N-1个Pod已经是运行且准备状态

- StatefulSet里的Pod采用稳定的持久化存储卷，通过PV或PVC来实现，删除Pod时默认不会删除与StatefulSet相关的存储卷（为了保证数据的安全）

StatefulSet除了要与PV卷捆绑使用以存储Pod的状态数据，还要与Headless，Service配合使用，每个StatefulSet定义中都要生命它属于哪个Handless Service，Handless Service与普通Service的关键区别在于，它没有Cluster IP