22.DaemonSet

在每个节点上运行pod的一个副本

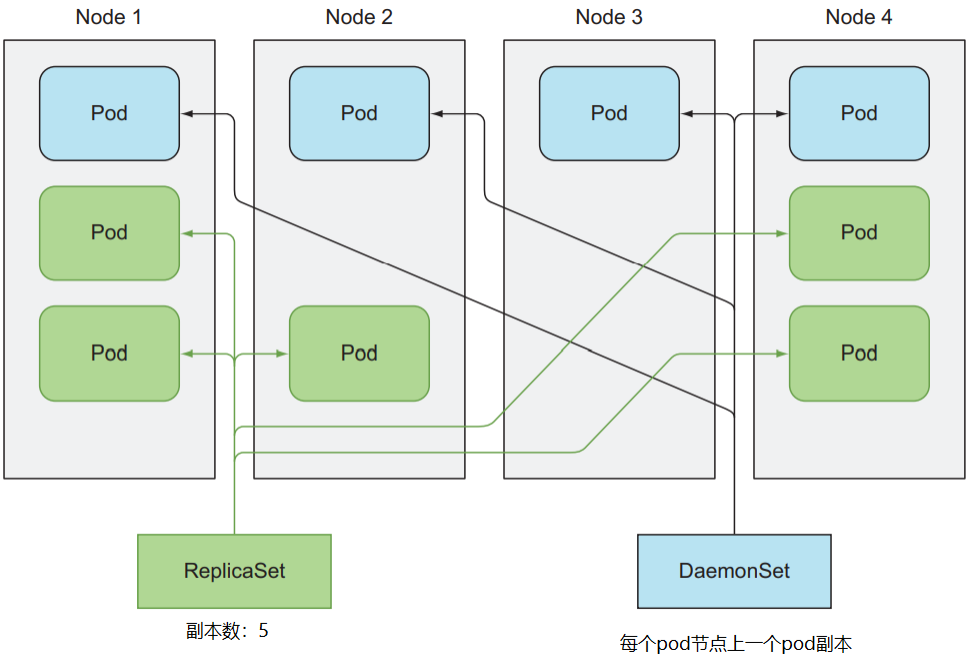
在某些节点上运行pod的副本

创建DaemonSet

ReplicationController和ReplicaSet一样，都是用于运行指定数量的pod，这些pod会部署到Kubernetes集群中的任何位置。但是有些情况下（如有些与基础设施相关的pod需要执行系统级别的操作），我们希望能在集群中的每个节点上都运行某个pod的副本。
例如，我们希望在每个节点上都运行一个日志收集器和一个资源监控器；
例如，Kubernetes自身的kube-proxy进程需要在所有节点上运行，以便service能正常工作。
在没有使用Kubernetes时，类似这种进程通常会在节点启动的时候通过系统初始化脚本或者systemd守护进程来启动。如果使用了Kubernetes，在Kubernetes节点上，我们仍然可以使用systemd守护进程来运行系统进程，但是那样的话就不能利用Kubernetes提供的功能了。

在每个节点上运行pod的一个副本

想要在集群中的所有节点上运行pod的副本，我们需要创建一个DaemonSet对象。DaemonSet创建的pod不是随机分散到集群中的节点上，而是指定了目标节点并且绕过了Kubernetes调度器，其他方面跟ReplicationController和ReplicaSet类似。
DaemonSet确保创建与节点数量相同的pod，并将每个pod部署到各自的节点上。
如下图所示，DaemonSet使每个节点只运行一个pod副本，而ReplicaSet则根据副本数将5个pod副本随机分散到3个节点上：



与ReplicaSet和ReplicationController不同，DaemonSet没有而且也不需要有期望副本数的概念，因为它的工作是确保与它的pod选择器匹配的pod在每个节点上运行。
如果某个节点故障了，DaemonSet并不会导致Kubernetes创建一个新的pod。但是当集群中加入了一个新的节点时，DaemonSet就会立即部署一个新的pod到该节点上。如果不小心删除了某个节点上DaemonSet管理的所有pod，DaemonSet会基于pod模板创建新的pod。

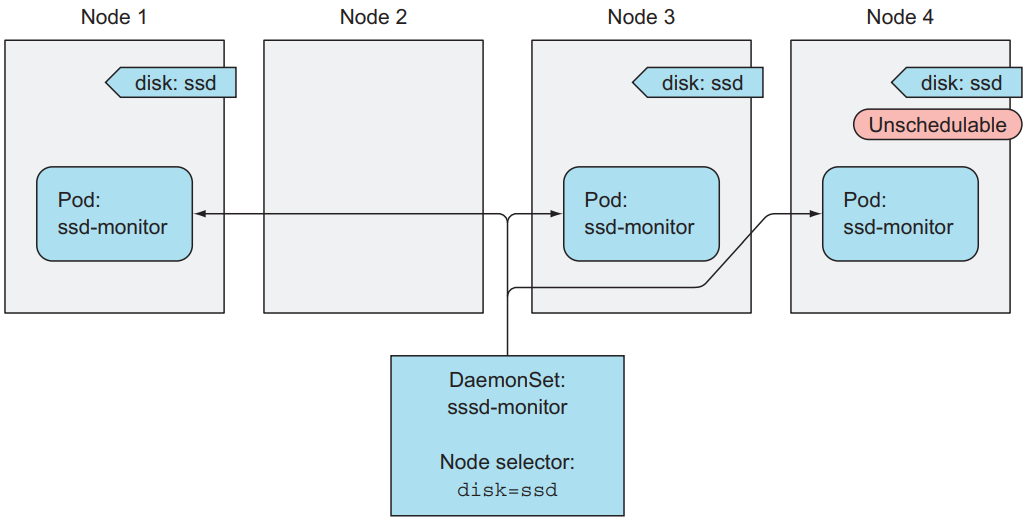
在某些节点上运行pod的副本

DaemonSet会将pod部署到集群中的所有节点上，除非我们特别指定pod只需要在某一部分节点上运行。
要实现这种功能，我们可以在DaemonSet定义文件的pod模板中指定nodeSelector属性。
在之前的章节中，我们使用节点选择器将一个pod部署到某个特定的节点上。DaemonSet中的节点选择器与此类似，它定义了Pod的目标部署节点。
在后面的学习中，我们还会了解到如何将一个节点设置成不可调度（unschedulable），从而防止pod部署到这个节点上。但是即便如此，DaemonSet还是会部署pod到这种节点上，因为只有调度器（Scheduler）才会使用unschedulable属性，而DaemonSet管理的pod会完全忽略调度器。这通常也是我们所希望达到的效果，因为DaemonSet旨在运行系统服务，即使在不可调度的节点上，系统服务也需要运行。

创建DaemonSet

现在我们假定有一个名为ssd-monitor的守护进程，它需要在所有包含固态驱动器（SSD）的节点上运行。我们可以创建一个DaemonSet，它会在所有标记为具有SSD的节点上运行这个守护进程。假设集群

管理员已经给所有的这种节点添加了disk=ssd标签，因此我们可以创建一个DaemonSet，它的节点选择器只会选择具有这个标签的节点，如下图所示：



首先我们需要制作并上传一个镜像：

Dockerfile内容如下：

FROM busybox

ENTRYPOINT while true; do echo 'SSD OK'; sleep 5; done

```
FROM busybox
ENTRYPOINT while true; do echo 'SSD OK'; sleep 5; done
```

```
[root@dhr-demo test02]# docker build -t ssd-monitor .
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM busybox
 ---> dc3bacd8b5ea
Step 2/2 : ENTRYPOINT while true; do echo 'SSD OK'; sleep 5; done
 ---> Running in 0bd6d66884f1
Removing intermediate container 0bd6d66884f1
 ---> 85e0fded1c04
Successfully built 85e0fded1c04
Successfully tagged ssd-monitor:latest
```

构建好的镜像为registry.cn-shanghai.aliyuncs.com/boogoo-ns/ssd-monitor:1.0

```
[root@dhr-demo test02]# docker images
REPOSITORY                                                TAG                 IMAGE ID            CREATED             SIZE
ssd-monitor                                               latest              85e0fded1c04        10 minutes ago      1.23MB
registry.cn-shanghai.aliyuncs.com/boogoo-ns/ssd-monitor   1.0                 85e0fded1c04        10 minutes ago      1.23MB
busybox                                                   latest              dc3bacd8b5ea        9 days ago          1.23MB
test1                                                     latest              0d510a193a68        12 days ago         660MB
test2                                                     latest              0d510a193a68        12 days ago         660MB
registry.cn-shanghai.aliyuncs.com/david-ns01/test1        1.0                 0d510a193a68        12 days ago         660MB
registry.cn-shanghai.aliyuncs.com/david-ns01/test2        1.0                 0d510a193a68        12 days ago         660MB
registry.cn-shanghai.aliyuncs.com/david-ns01/test1        <none>              773641d7cb99        4 weeks ago         660MB
```

然后创建一个运行模拟ssd-monitor进程的DaemonSet，该进程每5秒会打印“SSD OK”到标准输出。

创建DaemonSet的YAML文件：

vim ssd-monitor-daemonset.yaml

内容如下：

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: ssd-monitor
spec:
  selector:
    matchLabels:
      app: ssd-monitor
  template:
    metadata:
      labels:
        app: ssd-monitor
    spec:
      nodeSelector:
        disk: ssd
      containers:
      - name: main
        image: registry.cn-shanghai.aliyuncs.com/boogoo-ns/ssd-monitor:1.0
```

上面的文件定义了一个DaemonSet，它会运行一个包含单个容器的pod，这个容器基于我们刚刚创建的容器镜像。

创建DaemonSet后，每个包含disk=label标签的节点都会运行这个pod的一个实例。

执行如下命令创建DaemonSet：

```
kubectl create –f ssd-monitor-daemonset.yaml
```

```
[david@dhr-demo ~]$ kubectl create -f ssd-monitor-daemonset.yaml
daemonset.apps/ssd-monitor created
[david@dhr-demo ~]$
```

查看创建的DaemonSet:

kubectl get ds

```
[david@dhr-demo ~]$ kubectl get ds
NAME          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
ssd-monitor   0         0         0       0            0           disk=ssd        22s
[david@dhr-demo ~]$
```

由于我们还未为节点添加disk=ssd标签，所以创建DaemonSet后并没有创建pod。

kubectl get po

```
[david@dhr-demo root]$ kubectl get po
No resources found in default namespace.
[david@dhr-demo root]$
```

当前环境只有一个minikube节点，我们来为其添加disk=ssd标签：

kubectl label node minikube disk=ssd

```
[david@dhr-demo root]$ kubectl get nodes --show-labels
NAME       STATUS   ROLES    AGE   VERSION   LABELS
minikube   Ready    master   28d   v1.19.0   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,disk=ssd,gpu=true,kubernetes
stname=minikube,kubernetes.io/os=linux,minikube.k8s.io/commit=23aa1eb200a03ae5883dd9d453d4daf3e0f59668,minikube.k8s.io/name=minikub
20_11_07T16_47_08_0700,minikube.k8s.io/version=v1.13.0,node-role.kubernetes.io/master=
[david@dhr-demo root]$
```

此时，DaemonSet应该已经为我们创建了一个pod：

```
[david@dhr-demo root]$ kubectl get po
NAME                READY   STATUS    RESTARTS   AGE
ssd-monitor-h6wsg   1/1     Running   0          45s
[david@dhr-demo root]$
```

现在，假设我们不小心将disk=ssd换成了disk=hdd，来看看会发生什么情况：

kubectl label node minikube disk=hdd --overwrite

```
[david@dhr-demo root]$ kubectl label node minikube disk=hdd --overwrite
node/minikube labeled
[david@dhr-demo root]$ kubectl get po
NAME                READY   STATUS        RESTARTS   AGE
ssd-monitor-h6wsg   1/1     Terminating   0          5m45s
[david@dhr-demo root]$
```

可以看到pod正在被终止，最终会被删除掉。

需要注意的是，删除DaemonSet也会删除相应的pod。