35 容器化守护进程 DaemonSet

更新时间：2020-10-19 09:54:02

> 能够生存下来的物种,并不是那些最强壮的,也不是那些最聪明的,而是那些对变化作出快速反应的。——达尔文

在 Linux 系统中，有一种进程叫守护进程，英文是 daemon，这是一类在后台运行的特殊进程，用户执行特殊的系统任务。比如我们在 Linux 系统中，很多以 d 结尾的进程都是守护进程。

在 Kubernetes 中的 DaemonSet 严格意义上来说和守护进程关系其实不大。DaemonSet 的主要作用是用来控制 Daemon Pod。那么什么是 Daemon Pod 呢？Daemon Pod 具有如下一些特性：

- 这个 Pod 运行在 Kubernetes 集群中的每一个节点（Node）上；
- 每个节点上只能运行一个 Deamon Pod 实例；
- 当有新的节点（Node）加入到 Kubernetes 集群时，Daemon Pod 会被自动拉起；
- 当有旧节点被删除时，其上运行的 Daemon Pod 也将被删除。

DaemonSet 的典型应用场景如下：

- 在集群每个节点上启动一个存储守护进程，比如 `glusted` 或者 `ceph`；
- 在每个节点上启动一个日志收集进程，比如 `fluentd` 或者 `filebeat`；
- 在集群的每个节点上面启动监控的守护进程，比如 Prometheus 的 `node-exporter`。

1. 创建 DaemonSet

我们可以创建一个描述 DaemonSet 的 yaml 文件，下面是一个简单的例子。

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-app
  labels:
    k8s-app: fluentd
spec:
  selector:
    matchLabels:
      name: fluentd-app
  template:
    metadata:
      labels:
        name: fluentd-app
    spec:
      containers:
      - name: fluentd
        image: fluentd:v2.5.2
        resources:
          limits:
            cpu: 100m
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
```

简单介绍一下其中的重要部分：

- `kind`：指定 DaemonSet；
- `.spec.template`：是 Pod 模板，对应的 DaemonSet 启动的 Pod 的信息描述；
- `.spec.selector`：用来和 Pod 匹配的 selector，需要和 `.spec.template` 中描诉的 Pod 的 label 匹配上。从 Kubernetes 1.8 版本之后，这个字段必须指定。`.spec.selector` 支持两种：

  `matchLabels`：和 Pod 的 label 进行匹配。

  `matchExpression`：更加灵活的匹配，支持集合匹配，Operator 包括 `In` 和 `NotIn`。下面是一个简单的 matchExpression 示例，表示：

  ```yaml
  - matchExpressions:
    - key: kubernetes.io/e2e-az-name
      operator: In
      values:
      - e2e-az1
      - e2e-az2
  ```

2. 使用 DaemonSet

同样的，我们可以通过 `kubectl apply` 创建 DaemonSet。

```
$ kubectl apply -f fluentd.yaml -n imooc
daemonset.apps/fluentd-app configured
```

apply 成功之后，我们可以查看一下集群中的 Pod，如下，因为我们集群只有三个 worker 节点，所以一共有三个 Pod。

```
$ kubectl get po -n imooc
NAME                    READY   STATUS              RESTARTS   AGE
fluentd-app-6ml24       0/1     ContainerCreating   0          8s
fluentd-app-6sxz9       0/1     ContainerCreating   0          8s
fluentd-app-fknkb       0/1     ContainerCreating   0          8s
```

我们再来看一下集群中 DaemonSet 对象。

```
$ kubectl get daemonset -n imooc
NAME         DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
fluentd-app  3         3         3       3            3           <none>          4m26s
```

从 DaemonSet 对象的简略描述信息中可以看到该 DaemonSet 控制的 Pod 的状态：

- DESIRED：期望运行的 Pod 实例的个数；
- CURRENT：当前运行的 Pod 实例的个数；
- READY：状态 ready 的 Pod 实例的个数；
- …

我们再通过 `kubectl describe ds` 查看一下 DaemonSet 的明细信息，没错，这里的 ds 是 DaemonSet 的缩写。

```
$ kubectl describe ds fluentd-app -n imooc
Name:           fluentd-app
Selector:       name=fluentd-app
Node-Selector:  <none>
Labels:         k8s-app=fluentd
Annotations:    deprecated.daemonset.template.generation: 2
                kubectl.kubernetes.io/last-applied-configuration:
                  {"apiVersion":"apps/v1","kind":"DaemonSet","metadata":{"annotations":{},"labels":{"k8s-app":"fluentd"},"name":"fluentd-app","namespace":"i...
Desired Number of Nodes Scheduled: 3
Current Number of Nodes Scheduled: 3
Number of Nodes Scheduled with Up-to-date Pods: 3
Number of Nodes Scheduled with Available Pods: 3
Number of Nodes Misscheduled: 0
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  name=fluentd-app
  Containers:
   fluentd:
    Image:      fluentd
    Port:       <none>
    Host Port:  <none>
    Limits:
      cpu:     100m
      memory:  200Mi
    Requests:
      cpu:        100m
      memory:     200Mi
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From                  Message
  ----    ------            ----  ----                  -------
  Normal  SuccessfulCreate  6m24s  daemonset-controller  Created pod: fluentd-app-6sxz9
  Normal  SuccessfulCreate  6m24s  daemonset-controller  Created pod: fluentd-app-6ml24
  Normal  SuccessfulCreate  6m24s  daemonset-controller  Created pod: fluentd-app-fknkb
```

从这个输出里面我们可以看到几点信息：

- DaemonSet 的基本信息，包括名字，label 等；

- Pod 的调度情况；
- Pod 模板，也就是 Pod Template；
- Events：主要包括创建 pod 的事件；

下面我们看一下 DaemonSet 的自动拉起功能的特性。

为了展示自动拉起，很简单，我们只要删除 DaemonSet 之前拉起的 Pod，然后观察有没有新的 Pod 创建出来即可。

```
$ kubectl get pods -n imooc
NAME                          READY   STATUS    RESTARTS   AGE
fluentd-app-6ml24             1/1     Running   0          35m
fluentd-app-6sxz9             1/1     Running   0          35m
fluentd-app-fknkb             1/1     Running   0          35m
nginx-deployment-57f49c59d-8dzn4   1/1     Running   0          20h
nginx-deployment-57f49c59d-9jvrp   1/1     Running   0          20h
nginx-deployment-57f49c59d-m57sr   1/1     Running   0          20h
$ kubectl delete pods fluentd-app-6ml24 -n imooc
pod "fluentd-app-6ml24" deleted
$ kubectl get pods -n imooc
NAME                          READY   STATUS    RESTARTS   AGE
fluentd-app-2xjmg             1/1     Running   0          12s
fluentd-app-6sxz9             1/1     Running   0          36m
fluentd-app-fknkb             1/1     Running   0          36m
nginx-deployment-57f49c59d-8dzn4   1/1     Running   0          20h
nginx-deployment-57f49c59d-9jvrp   1/1     Running   0          20h
nginx-deployment-57f49c59d-m57sr   1/1     Running   0          20h
```

如上所示，在老的 Pod `fluentd-app-6ml24` 被删除之后，新的 Pod `fluentd-app-2xjmg` 立刻就被创建出来了。

DaemonSet 在新创建的 Kubernetes 的 Node 节点上自动创建的特性，这里就不再展示了。

虽然 DaemonSet 默认会在所有的节点上启动相同的 Pod，但是有时候我们还是希望只在某些指定的节点上面运行 Pod。对于这个问题有两种解决方案：

- 指定 `.spec.template.spec.nodeSelector` ，DaemonSet 将在能够与 Node Selector 匹配的节点上创建 Pod。
- 指定 `.spec.template.spec.affinity` ，然后 DaemonSet 将在能够与 nodeAffinity 匹配的节点上创建 Pod。

nodeSelector 示例

我们首先给某个节点打上特定的 label，使用命令 `kubectl labels` 。

```
$ kubectl label nodes <node-name> <label_key>=<value>
```

然后在 DaemonSet 的 yaml 文件中指定 nodeSelector。

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-app
  labels:
    k8s-app: fluentd
spec:
  selector:
    matchLabels:
      name: fluentd-app
  template:
    metadata:
      labels:
        name: fluentd-app
    spec:
      nodeSelector:
      <key>: <value>
      containers:
      - name: fluentd
        image: fluentd:v2.5.2
        resources:
          limits:
            cpu: 100m
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
```

nodeAffinity 示例

nodeAffinity 目前支持 4 种策略，分别是：

requiredDuringSchedulingIgnoredDuringExecution：表示 Pod 必须部署到满足条件的节点上，如果没有满足条件的节点，就不停重试。

requiredDuringSchedulingRequiredDuringExecution：类似 `requiredDuringSchedulingIgnoredDuringExecution`，不过如果节点标签发生了变化，不再满足pod指定的条件，则重新选择符合要求的节点。

preferredDuringSchedulingIgnoredDuringExecution：表示优先部署到满足条件的节点上，如果没有满足条件的节点，就忽略这些条件，按照正常逻辑部署。

preferredDuringSchedulingIgnoredDuringExecution：表示优先部署到满足条件的节点上，如果没有满足条件的节点，就忽略这些条件，按照正常逻辑部署。其中RequiredDuringExecution表示如果后面节点标签发生了变化，满足了条件，则重新调度到满足条件的节点。

下面我们以 requiredDuringSchedulingIgnoredDuringExecution 举例，看一下 DaemonSet 的一个示例 yaml。

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-app
  labels:
    k8s-app: fluentd
spec:
  selector:
    matchLabels:
      name: fluentd-app
  template:
    metadata:
      labels:
        name: fluentd-app
    spec:
      affinity:
     nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
       nodeSelectorTerms:
       - matchExpressions:
         - key: <label-name>
           operator: In
           values:
           - <value1>
           - <value2>
      containers:
      - name: fluentd
        image: fluentd:v2.5.2
        resources:
          limits:
            cpu: 100m
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
```

3. 更新 DaemonSet

如果 Node 节点的 label 发生改变，DaemonSet 会立刻根据节点的新 label 来做选择并调度 Pod，对于满足标签选择器的节点会将 Pod 调度上去，对于不满足标签选择器的节点则会删除上面的 Pod。

删除 DaemonSet 的时候，如果选择了参数 `--cascade=false` 则会保留之前 DaemonSet 创建出来的 Pod。然后可以创建具有不同模板的新 DaemonSet。具有不同模板的新 DaemonSet 将能够通过标签匹配并识别所有已经存在的 Pod。 如果有任何 Pod 需要替换，则 DaemonSet 根据它的 `updateStrategy` 来替换。

4. DaemonSet 工作原理

DaemonSet 的工作原理核心问题是要弄懂如何保证每个 Node 上有且只有一个被管理的 Pod。

这个好解决，我们只要拿到 Node 列表，然后检查每个 Node 节点上是不是运行指定的 label 的 Pod 就行了。而这正好是 DaemonSet Controller 做的事情，关于 Kubernetes 的控制器我们前面有介绍过，控制器会不断的检查状态是不是预期的，如果不是预期的就做一些处理。对于 DaemonSet Controller 这里遍历所有的 Node，然后状态会有如下几种情况：

- 没有指定 label 的 Pod 在运行，则需要在这个 Node 节点上创建一个这样的 Pod；
- 有指定 label 的 Pod 在运行，但是数量不是 1 个，可能是 2 个或者 3 个，则需要将多余的 Pod 删除；
- 正好有一个指定 label 的 Pod 在运行，这个是预期的行为，不做处理。

那么如何在新创建出来的 Node 创建新的 Pod 呢？或者说怎么将 Pod 调度到指定 Node 上呢？还记得我们之前 Pod 使用那章介绍的亲和性吗？是的，没错，这里使用的就是亲和性调度。

亲和性调度里面有一个是 nodeAffinity，就是用来将 Pod 调度到指定的 Node 节点上的。下面是一个简单的例子。

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: node-name
            operator: In
            values:
            - <new node name>
```

关于这个例子有几点需要说明的是：

- requiredDuringSchedulingIgnoredDuringExecution：每次调度的时候才考虑这个亲和性条件，如果之后 Node 节点的信息发生变更，并不会影响之前运行的 Pod。
- nodeSelectorTerms：具体的筛选条件，我们这里使用的 matchExpression，通过 node name 来进行比对选择。

看到这里，我们应该明白了，DaemonSet 的控制器在新的 Node 几点上创建 Pod 的时候，只需要加上类似这样一个 nodeAffinity 定义，然后在 select 选项里面通过新 Node 节点的名字进行匹配即可。

}

34 配置管理：ConfigMap 和 Secret

36 Kubernetes ReplicationController 和 ReplicaSet 介绍