

讲堂 > 深入剖析Kubernetes > 文章详情

## 18 | 深入理解StatefulSet (一) : 拓扑状态

2018-10-03 张磊



### 18 | 深入理解StatefulSet (一) : 拓扑状态

朗读人：张磊 11'52" | 5.45M

你好，我是张磊。今天我和你分享的主题是：深入理解 StatefulSet 之拓扑状态。

在上一篇文章中，我在结尾处讨论到了 Deployment 实际上并不足以覆盖所有的应用编排问题。

造成这个问题的根本原因，在于 Deployment 对应用做了一个简单化假设。

它认为，一个应用的所有 Pod，是完全一样的。所以，它们互相之间没有顺序，也无所谓运行在哪台宿主机上。需要的时候，Deployment 就可以通过 Pod 模板创建新的 Pod；不需要的时候，Deployment 就可以“杀掉”任意一个 Pod。

但是，在实际的场景中，并不是所有的应用都可以满足这样的要求。

尤其是分布式应用，它的多个实例之间，往往有依赖关系，比如：主从关系、主备关系。

还有就是数据存储类应用，它的多个实例，往往都会在本地的磁盘上保存一份数据。而这些实例一旦被杀掉，即便重建出来，实例与数据之间的对应关系也已经丢失，从而导致应用失败。

所以，这种实例之间有不对等关系，以及实例对外部数据有依赖关系的应用，就被称为“有状态应用”（Stateful Application）。

容器技术诞生后，大家很快发现，它用来封装“无状态应用”（Stateless Application），尤其是 Web 服务，非常好用。但是，一旦你想要用容器运行“有状态应用”，其困难程度就会直线上升。而且，这个问题解决起来，单纯依靠容器技术本身已经无能为力，这也就导致了很长一段时期内，“有状态应用”几乎成了容器技术圈子的“忌讳”，大家一听到这个词，就纷纷摇头。

不过，Kubernetes 项目还是成为了“第一个吃螃蟹的人”。

得益于“控制器模式”的设计思想，Kubernetes 项目很早就基于 Deployment 的基础上，扩展出了对“有状态应用”的初步支持。这个编排功能，就是：StatefulSet。

StatefulSet 的设计其实非常容易理解。它把真实世界里的应用状态，抽象为了两种情况：

1. 拓扑状态。这种情况意味着，应用的多个实例之间不是完全对等的关系。这些应用实例，必须按照某些顺序启动，比如应用的主节点 A 要先于从节点 B 启动。而如果你把 A 和 B 两个 Pod 删除掉，它们再次被创建出来时也必须严格按照这个顺序才行。并且，新创建出来的 Pod，必须和原来 Pod 的网络标识一样，这样原先的访问者才能使用同样的方法，访问到这个新 Pod。
2. 存储状态。这种情况意味着，应用的多个实例分别绑定了不同的存储数据。对于这些应用实例来说，Pod A 第一次读取到的数据，和隔了十分钟之后再次读取到的数据，应该是同一份，哪怕在此期间 Pod A 被重新创建过。这种情况最典型的例子，就是一个数据库应用的多个存储实例。

所以，StatefulSet 的核心功能，就是通过某种方式记录这些状态，然后在 Pod 被重新创建时，能够为新 Pod 恢复这些状态。

在开始讲述 StatefulSet 的工作原理之前，我就必须先为你讲解一个 Kubernetes 项目中非常实用的概念：Headless Service。

我在和你一起讨论 Kubernetes 架构的时候就曾介绍过，Service 是 Kubernetes 项目中用来将一组 Pod 暴露给外界访问的一种机制。比如，一个 Deployment 有 3 个 Pod，那么我就可以定义一个 Service。然后，用户只要能访问到这个 Service，它就能访问到某个具体的 Pod。

那么，这个 Service 又是如何被访问的呢？

第一种方式，是以 Service 的 VIP ( Virtual IP，即：虚拟 IP ) 方式。比如：当我访问 10.0.23.1 这个 Service 的 IP 地址时，10.0.23.1 其实就是一个 VIP，它会把请求转发到该 Service 所代理的某一个 Pod 上。这里的具体原理，我会在后续的 Service 章节中进行详细介绍。

第二种方式，就是以 Service 的 DNS 方式。比如：这时候，只要我访问 “my-svc.my-namespace.svc.cluster.local” 这条 DNS 记录，就可以访问到名叫 my-svc 的 Service 所代理的某一个 Pod。

而在第二种 Service DNS 的方式下，具体还可以分为两种处理方法：

第一种处理方法，是 Normal Service。这种情况下，你访问 “my-svc.my-namespace.svc.cluster.local” 解析到的，正是 my-svc 这个 Service 的 VIP，后面的流程就跟 VIP 方式一致了。

而第二种处理方法，正是 Headless Service。这种情况下，你访问 “my-svc.my-namespace.svc.cluster.local” 解析到的，直接就是 my-svc 代理的某一个 Pod 的 IP 地址。可以看到，这里的区别在于，Headless Service 不需要分配一个 VIP，而是可以直接以 DNS 记录的方式解析出被代理 Pod 的 IP 地址。

那么，这样的设计又有什么作用呢？

想要回答这个问题，我们需要从 Headless Service 的定义方式看起。

下面是一个标准的 Headless Service 对应的 YAML 文件：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
    - port: 80
      name: web
  clusterIP: None
  selector:
    app: nginx
```

[复制代码](#)

可以看到，所谓的 Headless Service，其实仍是一个标准 Service 的 YAML 文件。只不过，它的 clusterIP 字段的值是：None，即：这个 Service，没有一个 VIP 作为“头”。这也就是 Headless 的含义。所以，这个 Service 被创建后并不会被分配一个 VIP，而是会以 DNS 记录的方式暴露出它所代理的 Pod。

而它所代理的 Pod，依然是采用我在前面第 12 篇文章 [《牛刀小试：我的第一个容器化应用》](#) 中提到的 Label Selector 机制选择出来的，即：所有携带了 app=nginx 标签的 Pod，都会被这个 Service 代理起来。

然后关键来了。

当你按照这样的方式创建了一个 Headless Service 之后，它所代理的所有 Pod 的 IP 地址，都会被绑定一个这样格式的 DNS 记录，如下所示：

```
<pod-name>.<svc-name>.<namespace>.svc.cluster.local
```

[复制代码](#)

这个 DNS 记录，正是 Kubernetes 项目为 Pod 分配的唯一“可解析身份”（Resolvable Identity）。

有了这个“可解析身份”，只要你知道了一个 Pod 的名字，以及它对应的 Service 的名字，你就可以非常确定地通过这条 DNS 记录访问到 Pod 的 IP 地址。

那么，StatefulSet 又是如何使用这个 DNS 记录来维持 Pod 的拓扑状态的呢？

为了回答这个问题，现在我们就来编写一个 StatefulSet 的 YAML 文件，如下所示：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  serviceName: "nginx"
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
```

[复制代码](#)

```
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.9.1
    ports:
    - containerPort: 80
      name: web
```

这个 YAML 文件，和我们在前面文章中用到的 nginx-deployment 的唯一区别，就是多了一个 `serviceName=nginx` 字段。

这个字段的作用，就是告诉 StatefulSet 控制器，在执行控制循环（Control Loop）的时候，请使用 nginx 这个 Headless Service 来保证 Pod 的“可解析身份”。

所以，当你通过 `kubectl create` 创建了上面这个 Service 和 StatefulSet 之后，就会看到如下两个对象：

```
$ kubectl create -f svc.yaml
```

```
$ kubectl get service nginx
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nginx	ClusterIP	None	<none>	80/TCP	10s

```
$ kubectl create -f statefulset.yaml
```

```
$ kubectl get statefulset web
```

NAME	DESIRED	CURRENT	AGE
web	2	1	19s

[复制代码](#)

这时候，如果你手比较快的话，还可以通过 `kubectl` 的 `-w` 参数，即：Watch 功能，实时查看 StatefulSet 创建两个有状态实例的过程：

备注：如果手不够快的话，Pod 很快就创建完了。不过，你依然可以通过这个 StatefulSet 的 Events 看到这些信息。

```
$ kubectl get pods -w -l app=nginx
```

NAME	READY	STATUS	RESTARTS	AGE
web-0	0/1	Pending	0	0s

[复制代码](#)



web-0	0/1	Pending	0	0s
web-0	0/1	ContainerCreating	0	0s
web-0	1/1	Running	0	19s
web-1	0/1	Pending	0	0s
web-1	0/1	Pending	0	0s
web-1	0/1	ContainerCreating	0	0s
web-1	1/1	Running	0	20s

通过上面这个 Pod 的创建过程，我们不难看出，StatefulSet 给它所管理的所有 Pod 的名字，进行了编号，编号规则是：-。

而且这些编号都是从 0 开始累加，与 StatefulSet 的每个 Pod 实例一一对应，绝不重复。

更重要的是，这些 Pod 的创建，也是严格按照编号顺序进行的。比如，在 web-0 进入到 Running 状态、并且细分状态（Conditions）成为 Ready 之前，web-1 会一直处于 Pending 状态。

备注：Ready 状态再一次提醒了我们，为 Pod 设置 livenessProbe 和 readinessProbe 的重要性。

当这两个 Pod 都进入了 Running 状态之后，你就可以查看到它们各自唯一的“网络身份”了。

我们使用 `kubectl exec` 命令进入到容器中查看它们的 hostname：

```
$ kubectl exec web-0 -- sh -c 'hostname'
web-0
$ kubectl exec web-1 -- sh -c 'hostname'
web-1
```


[复制代码](#)

可以看到，这两个 Pod 的 hostname 与 Pod 名字是一致的，都被分配了对应的编号。接下来，我们再试着以 DNS 的方式，访问一下这个 Headless Service：

```
$ kubectl run -i --tty --image busybox dns-test --restart=Never --rm /bin/sh
```

[复制代码](#)

通过这条命令，我们启动了一个一次性的 Pod，因为 `--rm` 意味着 Pod 退出后就会被删除掉。然后，在这个 Pod 的容器里面，我们尝试用 `nslookup` 命令，解析一下 Pod 对应的 Headless Service：

 复制代码

```
$ kubectl run -i --tty --image busybox dns-test --restart=Never --rm /bin/sh

$ nslookup web-0.nginx

Server:      10.0.0.10
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local

Name:        web-0.nginx
Address 1: 10.244.1.7


$ nslookup web-1.nginx

Server:      10.0.0.10
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local

Name:        web-1.nginx
Address 1: 10.244.2.7
```

从 nslookup 命令的输出结果中，我们可以看到，在访问 web-0.nginx 的时候，最后解析到的，正是 web-0 这个 Pod 的 IP 地址；而当访问 web-1.nginx 的时候，解析到的则是 web-1 的 IP 地址。

这时候，如果你在另外一个 Terminal 里把这两个“有状态应用”的 Pod 删掉：


 复制代码

```
$ kubectl delete pod -l app=nginx

pod "web-0" deleted

pod "web-1" deleted
```

然后，再在当前 Terminal 里 Watch 一下这两个 Pod 的状态变化，就会发现一个有趣的现象：

 复制代码

```
$ kubectl get pod -w -l app=nginx
```

NAME	READY	STATUS	RESTARTS	AGE
web-0	0/1	ContainerCreating	0	0s


NAME	READY	STATUS	RESTARTS	AGE
web-0	1/1	Running	0	2s
web-1	0/1	Pending	0	0s
web-1	0/1	ContainerCreating	0	0s
web-1	1/1	Running	0	32s

可以看到，当我们把这两个 Pod 删除之后，Kubernetes 会按照原先编号的顺序，创建出了两个新的 Pod。并且，Kubernetes 依然为它们分配了与原来相同的“网络身份”：web-0.nginx 和 web-1.nginx。

通过这种严格的对应规则，StatefulSet 就保证了 Pod 网络标识的稳定性。

比如，如果 web-0 是一个需要先启动的主节点，web-1 是一个后启动的从节点，那么只要这个 StatefulSet 不被删除，你访问 web-0.nginx 时始终都会落在主节点上，访问 web-1.nginx 时，则始终都会落在从节点上，这个关系绝对不会发生任何变化。

所以，如果我们再用 nslookup 命令，查看一下这个新 Pod 对应的 Headless Service 的话：

 复制代码

```
$ kubectl run -i --tty --image busybox dns-test --restart=Never --rm /bin/sh
$ nslookup web-0.nginx
Server:      10.0.0.10
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local

Name:        web-0.nginx
Address 1: 10.244.1.8

$ nslookup web-1.nginx
Server:      10.0.0.10
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local

Name:        web-1.nginx
Address 1: 10.244.2.8
```

我们可以看到，在这个 StatefulSet 中，这两个新 Pod 的“网络标识”（比如：web-0.nginx 和 web-1.nginx），再次解析到了正确的 IP 地址（比如：web-0 Pod 的 IP 地址 10.244.1.8）。

通过这种方法，Kubernetes 就成功地将 Pod 的拓扑状态（比如：哪个节点先启动，哪个节点后启动），按照 Pod 的“名字 + 编号”的方式固定了下来。此外，Kubernetes 还为每一个 Pod 提供了一个固定并且唯一的访问入口，即：这个 Pod 对应的 DNS 记录。

这些状态，在 StatefulSet 的整个生命周期里都会保持不变，绝不会因为对应 Pod 的删除或者重新创建而失效。



不过，相信你也已经注意到了，尽管 `web-0.nginx` 这条记录本身不会变，但它解析到的 Pod 的 IP 地址，并不是固定的。这就意味着，对于“有状态应用”实例的访问，你必须使用 DNS 记录或者 `hostname` 的方式，而绝不应该直接访问这些 Pod 的 IP 地址。

## 总结

在今天这篇文章中，我首先和你分享了 `StatefulSet` 的基本概念，解释了什么是应用的“状态”。

紧接着，我为你分析了 `StatefulSet` 如何保证应用实例之间“拓扑状态”的稳定性。

如果用一句话来总结的话，你可以这么理解这个过程：

`StatefulSet` 这个控制器的主要作用之一，就是使用 Pod 模板创建 Pod 的时候，对它们进行编号，并且按照编号顺序逐一完成创建工作。而当 `StatefulSet` 的“控制循环”发现 Pod 的“实际状态”与“期望状态”不一致，需要新建或者删除 Pod 进行“调谐”的时候，它会严格按照这些 Pod 编号的顺序，逐一完成这些操作。

所以，`StatefulSet` 其实可以认为是对 `Deployment` 的改良。

与此同时，通过 `Headless Service` 的方式，`StatefulSet` 为每个 Pod 创建了一个固定并且稳定的 DNS 记录，来作为它的访问入口。

实际上，在部署“有状态应用”的时候，应用的每个实例拥有唯一并且稳定的“网络标识”，是一个非常重要的假设。

在下一篇文章中，我将会继续为你剖析 `StatefulSet` 如何处理存储状态。

## 思考题

你曾经运维过哪些有拓扑状态的应用呢（比如：主从、主主、主备、一主多从等结构）？你觉得这些应用实例之间的拓扑关系，能否借助这种为 Pod 实例编号的方式表达出来呢？如果不能，你觉得 Kubernetes 还应该为你提供哪些支持来管理这个拓扑状态呢？

感谢你的收听，欢迎你给我留言。



# 深入剖析 Kubernetes

Kubernetes 原来可以如此简单

张磊

Kubernetes 社区  
资深成员与项目维护者



版权归极客邦科技所有，未经许可不得转载

写留言

## 精选留言



asdf100

那么两种service模式应用场景通常有哪些？

2018-10-03

1



Dillion

在上面的例子中，web-0、web-1启动后，此时如果web-0挂了，那在创建web-0的过程中，web-1也会被重新创建一次么？？？也就是如果一个StatefulSet中只有某个Pod挂了，在重启它的时候，如何确保文中说的Pod启动顺序呢？

2018-10-03

0

作者回复

当然。任何pod的变化都会触发一次statefulset的滚动更新。

2018-10-03