

19 | 深入理解StatefulSet（二）：存储状态

2018-10-05 张磊

深入剖析Kubernetes

[进入课程 >](#)



讲述：张磊

时长 12:51 大小 5.89M



你好，我是张磊。今天我和你分享的主题是：深入理解 StatefulSet 之存储状态。

在上一篇文章中，我和你分享了 StatefulSet 如何保证应用实例的拓扑状态，在 Pod 删除和再创建的过程中保持稳定。

而在今天这篇文章中，我将继续为你解读 StatefulSet 对存储状态的管理机制。这个机制，主要使用的是一个叫作 **Persistent Volume Claim** 的功能。


在前面介绍 Pod 的时候，我曾提到过，要在一个 Pod 里声明 Volume，只要在 Pod 里加上 spec.volumes 字段即可。然后，你就可以在这个字段里定义一个具体类型的 Volume 了，比如：hostPath。

可是，你有没有想过这样一个场景：**如果你并不知道有哪些 Volume 类型可以用，要怎么办呢？**

更具体地说，作为一个应用开发者，我可能对持久化存储项目（比如 Ceph、GlusterFS 等）一窍不通，也不知道公司的 Kubernetes 集群里到底是怎么搭建出来的，我也自然不会编写它们对应的 Volume 定义文件。

所谓“术业有专攻”，这些关于 Volume 的管理和远程持久化存储的知识，不仅超越了开发者的知识储备，还会有暴露公司基础设施秘密的风险。

比如，下面这个例子，就是一个声明了 Ceph RBD 类型 Volume 的 Pod：

 复制代码

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: rbd
5  spec:
6    containers:
7      - image: kubernetes/pause
8        name: rbd-rw
9        volumeMounts:
10       - name: rbdpd
11         mountPath: /mnt/rbd
12  volumes:
13    - name: rbdpd
14      rbd:
15        monitors:
16          - '10.16.154.78:6789'
17          - '10.16.154.82:6789'
18          - '10.16.154.83:6789'
19        pool: kube
20        image: foo
21        fsType: ext4
22        readOnly: true
23        user: admin
24        keyring: /etc/ceph/keyring
25        imageformat: "2"
26        imagefeatures: "layering"
```


其一，如果不懂得 Ceph RBD 的使用方法，那么这个 Pod 里 Volumes 字段，你十有八九也完全看不懂。其二，这个 Ceph RBD 对应的存储服务器的地址、用户名、授权文件的位

置，也都被轻易地暴露给了全公司的所有开发人员，这是一个典型的信息被“过度暴露”的例子。

这也是为什么，在后来的演化中，Kubernetes 项目引入了一组叫作 **Persistent Volume Claim (PVC)** 和 **Persistent Volume (PV)** 的 API 对象，大大降低了用户声明和使用持久化 Volume 的门槛。

举个例子，有了 PVC 之后，一个开发人员想要使用一个 Volume，只需要简单的两步即可。

第一步：定义一个 PVC，声明想要的 Volume 的属性：


 复制代码

```
1 kind: PersistentVolumeClaim
2 apiVersion: v1
3 metadata:
4   name: pv-claim
5 spec:
6   accessModes:
7     - ReadWriteOnce
8   resources:
9     requests:
10       storage: 1Gi
```

可以看到，在这个 PVC 对象里，不需要任何关于 Volume 细节的字段，只有描述性的属性和定义。比如，`storage: 1Gi`，表示我想要的 Volume 大小至少是 1 GiB；`accessModes: ReadWriteOnce`，表示这个 Volume 的挂载方式是可读写，并且只能被挂载在一个节点上而非被多个节点共享。

备注：关于哪种类型的 Volume 支持哪种类型的 AccessMode，你可以查看 Kubernetes 项目官方文档中的[详细列表](#)。

第二步：在应用的 Pod 中，声明使用这个 PVC：

 复制代码

```
1 apiVersion: v1
2 kind: Pod
```

```


3 metadata:
4   name: pv-pod
5 spec:
6   containers:
7     - name: pv-container
8       image: nginx
9       ports:
10        - containerPort: 80
11          name: "http-server"
12       volumeMounts:
13        - mountPath: "/usr/share/nginx/html"
14          name: pv-storage
15   volumes:
16     - name: pv-storage
17       persistentVolumeClaim:
18         claimName: pv-claim

```

可以看到，在这个 Pod 的 Volumes 定义中，我们只需要声明它的类型是 `persistentVolumeClaim`，然后指定 PVC 的名字，而完全不必关心 Volume 本身的定义。

这时候，只要我们创建这个 PVC 对象，Kubernetes 就会自动为它绑定一个符合条件的 Volume。可是，这些符合条件的 Volume 又是从哪里来的呢？

答案是，它们来自于由运维人员维护的 PV (Persistent Volume) 对象。接下来，我们一起看一个常见的 PV 对象的 YAML 文件：

 复制代码

```

1 kind: PersistentVolume
2 apiVersion: v1
3 metadata:
4   name: pv-volume
5   labels:
6     type: local
7 spec:
8   capacity:
9     storage: 10Gi
10  rbd:
11    monitors:
12      - '10.16.154.78:6789'
13      - '10.16.154.82:6789'
14      - '10.16.154.83:6789'
15    pool: kube
16    image: foo
17    fsType: ext4

```

```
18     readOnly: true
19     user: admin
20     keyring: /etc/ceph/keyring
21     imageformat: "2"
22     imagefeatures: "layering"
```

可以看到，这个 PV 对象的 spec.rbd 字段，正是我们前面介绍过的 Ceph RBD Volume 的详细定义。而且，它还声明了这个 PV 的容量是 10 GiB。这样，Kubernetes 就会为我们刚刚创建的 PVC 对象绑定这个 PV。

所以，Kubernetes 中 PVC 和 PV 的设计，**实际上类似于“接口”和“实现”的思想**。开发者只要知道并会使用“接口”，即：PVC；而运维人员则负责给“接口”绑定具体的实现，即：PV。

这种解耦，就避免了因为向开发者暴露过多的存储系统细节而带来的隐患。此外，这种职责的分离，往往也意味着出现事故时可以更容易定位问题和明确责任，从而避免“扯皮”现象的出现。

而 PVC、PV 的设计，也使得 StatefulSet 对存储状态的管理成为了可能。我们还是以上一篇文章中用到的 StatefulSet 为例（你也可以借此再回顾一下[《深入理解 StatefulSet（一）：拓扑状态》](#)中的相关内容）：

 复制代码

```
1  apiVersion: apps/v1
2  kind: StatefulSet
3  metadata:
4    name: web
5  spec:
6    serviceName: "nginx"
7    replicas: 2
8    selector:
9      matchLabels:
10       app: nginx
11    template:
12      metadata:
13        labels:
14          app: nginx
15      spec:
16        containers:
17          - name: nginx
18            image: nginx:1.9.1
```

```
19     ports:
20     - containerPort: 80
21       name: web
22     volumeMounts:
23     - name: www
24       mountPath: /usr/share/nginx/html
25   volumeClaimTemplates:
26   - metadata:
27     name: www
28     spec:
29       accessModes:
30       - ReadWriteOnce
31       resources:
32         requests:
33           storage: 1Gi
```

这次，我们为这个 StatefulSet 额外添加了一个 volumeClaimTemplates 字段。从名字就可以看出来，它跟 Deployment 里 Pod 模板（PodTemplate）的作用类似。也就是说，凡是被这个 StatefulSet 管理的 Pod，都会声明一个对应的 PVC；而这个 PVC 的定义，就来自于 volumeClaimTemplates 这个模板字段。更重要的是，这个 PVC 的名字，会被分配一个与这个 Pod 完全一致的编号。

这个自动创建的 PVC，与 PV 绑定成功后，就会进入 Bound 状态，这就意味着这个 Pod 可以挂载并使用这个 PV 了。

如果你还是不太理解 PVC 的话，可以先记住这样一个结论：**PVC 其实就是一种特殊的 Volume**。只不过一个 PVC 具体是什么类型的 Volume，要在跟某个 PV 绑定之后才知道。关于 PV、PVC 更详细的知识，我会在容器存储部分做进一步解读。

当然，PVC 与 PV 的绑定得以实现的前提是，运维人员已经在系统里创建好了符合条件的 PV（比如，我们在前面用到的 pv-volume）；或者，你的 Kubernetes 集群运行在公有云上，这样 Kubernetes 就会通过 Dynamic Provisioning 的方式，自动为你创建与 PVC 匹配的 PV。


所以，我们在使用 kubectl create 创建了 StatefulSet 之后，就会看到 Kubernetes 集群里出现了两个 PVC：

```
2 $ kubectl get pvc -l app=nginx
3 NAME          STATUS    VOLUME                                     CAPACITY   ACCESSMODES
4 www-web-0     Bound    pvc-15c268c7-b507-11e6-932f-42010a800002  1Gi        RWO
5 www-web-1     Bound    pvc-15c79307-b507-11e6-932f-42010a800002  1Gi        RWO
```

可以看到，这些 PVC，都以 “<PVC 名字>-<StatefulSet 名字>-<编号>” 的方式命名，并且处于 Bound 状态。

我们前面已经讲到过，这个 StatefulSet 创建出来的所有 Pod，都会声明使用编号的 PVC。比如，在名叫 web-0 的 Pod 的 volumes 字段，它会声明使用名叫 www-web-0 的 PVC，从而挂载到这个 PVC 所绑定的 PV。

所以，我们就可以使用如下所示的指令，在 Pod 的 Volume 目录里写入一个文件，来验证一下上述 Volume 的分配情况：

 复制代码

```
1 $ for i in 0 1; do kubectl exec web-$i -- sh -c 'echo hello $(hostname) > /usr/share/nginx/
```

如上所示，通过 kubectl exec 指令，我们在每个 Pod 的 Volume 目录里，写入了一个 index.html 文件。这个文件的内容，正是 Pod 的 hostname。比如，我们在 web-0 的 index.html 里写入的内容就是 "hello web-0"。


此时，如果你在这个 Pod 容器里访问“http://localhost”，你实际访问到的就是 Pod 里 Nginx 服务器进程，而它会为你返回 /usr/share/nginx/html/index.html 里的内容。这个操作的执行方法如下所示：

 复制代码

```
1 $ for i in 0 1; do kubectl exec -it web-$i -- curl localhost; done
2 hello web-0
3 hello web-1
```


现在，关键来了。

如果你使用 `kubectl delete` 命令删除这两个 Pod，这些 Volume 里的文件会不会丢失呢？

 复制代码

```
1 $ kubectl delete pod -l app=nginx
2 pod "web-0" deleted
3 pod "web-1" deleted
```

可以看到，正如我们前面介绍过的，在被删除之后，这两个 Pod 会被按照编号的顺序被重新创建出来。而这时候，如果你在新创建的容器里通过访问“`http://localhost`”的方式去访问 web-0 里的 Nginx 服务：

 复制代码

```
1 # 在被重新创建出来的 Pod 容器里访问 http://localhost
2 $ kubectl exec -it web-0 -- curl localhost
3 hello web-0
```

就会发现，这个请求依然会返回：`hello web-0`。也就是说，原先与名叫 web-0 的 Pod 绑定的 PV，在这个 Pod 被重新创建之后，依然同新的名叫 web-0 的 Pod 绑定在了一起。对于 Pod web-1 来说，也是完全一样的情况。

这是怎么做到的呢？

其实，我和你分析一下 StatefulSet 控制器恢复这个 Pod 的过程，你就可以很容易理解了。

首先，当你把一个 Pod，比如 web-0，删除之后，这个 Pod 对应的 PVC 和 PV，并不会被删除，而这个 Volume 里已经写入的数据，也依然会保存在远程存储服务里（比如，我们在这个例子里用到的 Ceph 服务器）。

此时，StatefulSet 控制器发现，一个名叫 web-0 的 Pod 消失了。所以，控制器就会重新创建一个新的、名字还是叫作 web-0 的 Pod 来，“纠正”这个不一致的情况。

需要注意的是，在这个新的 Pod 对象的定义里，它声明使用的 PVC 的名字，还是叫作：www-web-0。这个 PVC 的定义，还是来自于 PVC 模板（volumeClaimTemplates），这是 StatefulSet 创建 Pod 的标准流程。

所以，在这个新的 web-0 Pod 被创建出来之后，Kubernetes 为它查找名叫 www-web-0 的 PVC 时，就会直接找到旧 Pod 遗留下来的同名的 PVC，进而找到跟这个 PVC 绑定在一起的 PV。

这样，新的 Pod 就可以挂载到旧 Pod 对应的那个 Volume，并且获取到保存在 Volume 里的数据。

通过这种方式，Kubernetes 的 StatefulSet 就实现了对应用存储状态的管理。

看到这里，你是不是已经大致理解了 StatefulSet 的工作原理呢？现在，我再为你详细梳理一下吧。

首先，StatefulSet 的控制器直接管理的是 Pod。这是因为，StatefulSet 里的不同 Pod 实例，不再像 ReplicaSet 中那样都是完全一样的，而是有了细微区别的。比如，每个 Pod 的 hostname、名字等都是不同的、携带了编号的。而 StatefulSet 区分这些实例的方式，就是通过在 Pod 的名字里加上事先约定好的编号。

其次，Kubernetes 通过 Headless Service，为这些有编号的 Pod，在 DNS 服务器中生成带有同样编号的 DNS 记录。只要 StatefulSet 能够保证这些 Pod 名字里的编号不变，那么 Service 里类似于 web-0.nginx.default.svc.cluster.local 这样的 DNS 记录也就不会变，而这条记录解析出来的 Pod 的 IP 地址，则会随着后端 Pod 的删除和再创建而自动更新。这当然是 Service 机制本身的能力，不需要 StatefulSet 操心。

最后，StatefulSet 还为每一个 Pod 分配并创建一个同样编号的 PVC。这样，Kubernetes 就可以通过 Persistent Volume 机制为这个 PVC 绑定上对应的 PV，从而保证了每一个 Pod 都拥有一个独立的 Volume。

在这种情况下，即使 Pod 被删除，它所对应的 PVC 和 PV 依然会保留下来。所以当这个 Pod 被重新创建出来之后，Kubernetes 会为它找到同样编号的 PVC，挂载这个 PVC 对应的 Volume，从而获取到以前保存在 Volume 里的数据。

这么一看，原本非常复杂的 StatefulSet，是不是也很容易理解了呢？

总结

在今天这篇文章中，我为你详细介绍了 StatefulSet 处理存储状态的方法。然后，以此为基础，我为你梳理了 StatefulSet 控制器的工作原理。

从这些讲述中，我们不难看出 StatefulSet 的设计思想：StatefulSet 其实就是一种特殊的 Deployment，而其独特之处在于，它的每个 Pod 都被编号了。而且，这个编号会体现在 Pod 的名字和 hostname 等标识信息上，这不仅代表了 Pod 的创建顺序，也是 Pod 的重要网络标识（即：在整个集群里唯一的、可被的访问身份）。

有了这个编号后，StatefulSet 就使用 Kubernetes 里的两个标准功能：Headless Service 和 PV/PVC，实现了对 Pod 的拓扑状态和存储状态的维护。

实际上，在下一篇文章的“有状态应用”实践环节，以及后续的讲解中，你就会逐渐意识到，StatefulSet 可以说是 Kubernetes 中作业编排的“集大成者”。

因为，几乎每一种 Kubernetes 的编排功能，都可以在编写 StatefulSet 的 YAML 文件时被用到。

思考题

在实际场景中，有一些分布式应用的集群是这么工作的：当一个新节点加入到集群时，或者老节点被迁移后重建时，这个节点可以从主节点或者其他从节点那里同步到自己所需要的数据。

在这种情况下，你认为是否还有必要将这个节点 Pod 与它的 PV 进行一对一绑定呢？（提示：这个问题的答案根据不同的项目是不同的。关键在于，重建后的节点进行数据恢复和同步的时候，是不是一定需要原先它写在本地磁盘里的数据）

感谢你的收听，欢迎你给我留言，也欢迎分享给更多的朋友一起阅读。

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



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 18 | 深入理解StatefulSet (一)：拓扑状态

下一篇 20 | 深入理解StatefulSet (三)：有状态应用实践

精选留言 (35)

写留言



虎虎

2018-10-05

12

老师，由于无法追问，恳请能够回复详细一些。提前谢过了。

pvc的使用方式当中，对于由运维人员去创建pv，我始终有些疑问。

首先，一个pv对应一个pvc。如果实际绑定的pvc小于pv声明的存储大小，会造成存储的浪费吗？

其次，运维人员事先要创建多少个，以及多大容量的pv呢？因为并不清楚开发人员将来...

展开

作者回复: 手动创建PV就是有这些问题，要不然为啥推崇storageclass呢。可以自己编写external provisioner来代替你自己写pv啊，跟你用storageclass一样。



pepezzz...

2019-02-21

👍 6

原文中的PV创建不成功。

spec.accessModes: Required value

unknown field "imagefeatures" in io.k8s.api.core.v1.RBDPersistentVolumeSource

unknown field "imageformat" in io.k8s.api.core.v1.RBDPersistentVolumeSource

修改后如下：...

展开 ▾



程序修行

2018-12-03

👍 6

老师能每次给出本次讲解全套的实现命令吗？貌似每次复现都会花很多时间，还不知道为什么。这次就是不知道pv该怎么创建，为什么总是创建失败。

展开 ▾



浅行

2019-01-07

👍 4

老师这个里面创建pv需要有ceph存储支持，大家做实验可以搞一个rook-ceph，两步创建一个ceph集群，很好用，官网：<https://rook.io/>。

展开 ▾



Terry Hu

2018-11-28

👍 3

由于没有ceph server，pv用hostPath方式，在master上创建了index.html，pv pvc都创建好了，bound上了，登陆容器后在/usr/share/nginx/html目录下死活找不到index.html，搞了一个小时。突然猛醒原来hostPath指的是worker节点的path.....哎，蠢啊

展开 ▾

作者回复: 吃一堑长一智



kyleqian

2018-10-14

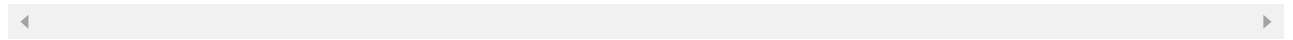
👍 3

对于有些应用，比如关系数据库，它保存数据文件的位置必须严格符合POSIX接口，远程

文件系统例如NFS对于类似锁定这样的操作支持的不好，即使是sqlite官方文档也不推荐用NFS。这种情况下，数据库应用好像只能用本地硬盘或者iSCSI的存储盘，这不就等同必须把重启的StatefulSet的Pod每次调度到同一个机器上才行吗？因为那个机器硬盘上的文件不会自动传输到其它机器上。是不是可以这么理解？

展开 ▾

作者回复: 关系数据库也可以做replication同步数据啊。pv跟机器没有绑定关系，除非用的是local pv，这时候调度器会保证pod的调度正确。



北卡

2018-10-06

👍 2

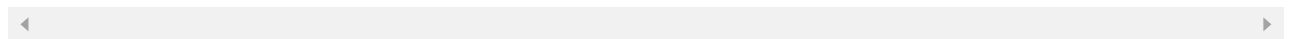
asdf100的问题应该是：一个集群中可能有很多个pv, pvc是如何找到他对应的pv的？

试着回答一下：

1、 指定了storageClassName的pv, 只有指定了同样storageClassName的pvc才能绑定到它上面。...

展开 ▾

作者回复: 容量是重要的匹配条件之一



kevinsu

2019-03-27

👍 1

rook-ceph-mon的IP地址正确获取方式来自查阅rook官方文档，kubectl -n rook-ceph get service，非常重要！！

展开 ▾



哈哼

2019-02-25

👍 1

rolling uodate 咋搞？

展开 ▾



不知所措

2018-12-11

👍 1

老师你好，statefulset 如果 不同的pod ，需要不同的配置，
比如说 zk集群，每个集群的myid 都是不同的，比如mysql集群每个主机的serverid 也是不同的，这种的怎么处理呢？

展开 ▾

作者回复: 用operator



Adam

2018-12-04

👍 1

老师写得很赞，多做几遍实验后就明白怎么使用pv，pvc了。需要提前创建好pv确实有点麻烦。这一步得自动完成才完美。

展开 ▾



LQ

2018-10-17

👍 1

在一个 Pod 里面有两个容器，容器 a 里自己实现一个 fuse filesystem，将远程的文件（比如 hdfs 上的数据）mount 到该容器里，另外一个容器 b 通过什么方式能读取到容器 a 里 mount 的数据呢，通过 PV 吗？有啥解决方案没？

展开 ▾

作者回复: 你让a通过挂载传播到宿主机上，然后b挂载宿主机目录试试。



我来也

2018-10-13

👍 1

表示 nginx:1.9.1 里并没有 curl 命令.

通过 `for i in 0 1; do kubectl exec -it web-$i -- curl localhost; done` 无法获取到期待的结果.

通过

不知道其他小伙伴尝试过没有....

展开 ▾

作者回复: 没有的话可以装一个嘛.....



yulibaoz...

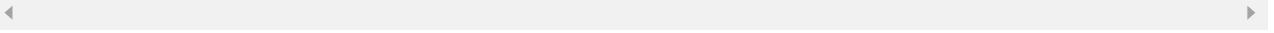
2018-10-10

👍 1

storage: 1Gi , 表示我想要的 Volume 大小至少是 1 GB ; 这里是笔误么 ? 1Gi不等于1GB, 如果这里是1G应该更符合语境

展开 ▾

作者回复: 是的



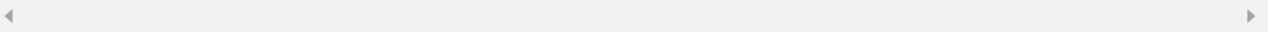
silver

2018-10-10

👍 1

所以大体是Pod与PVC绑定 , PVC与PV绑定 , 所以完成了Pod与PV的绑定?

作者回复: 是



虎虎

2018-10-08

👍 1

有必要。可以减少同步的数据量。

展开 ▾



Mr.Cling

2019-05-10

👍

如果创建了两个存储类型和存储空间都一样的PV , pvc是怎么决定绑定哪一个pv呢 ? 随机选择一个吗 ? 可不可能通过类似label的筛选 ?

展开 ▾



kevinsu

2019-05-09

👍

我用ubuntu完全去按照之前的安装方法搞的环境 , 但是到持久化存储这块老是有这个错 : Warning FailedMount 15s (x8 over 79s) kubelet, izj6cbsxfhzowfxz65471sz MountVolume.WaitForAttach failed for volume "pv-volume" : fail to check rbd image status with: (executable file not found in \$PATH), rbd output: ()

超级郁闷

展开 ▾



honnkyou

2019-05-09



各位rook部署成功，但关联PVC的Pod还是无法创建成功不知各位有没有什么好的方法。
rook 相关的pod 情况如下。

===

```
kubectll -n rook - cephgetpods - owideNAMEREADYSTATUSRESTA
```

```
cat pv-volume.yaml
```

```
apiVersion: v1
```

```
kind: PersistentVolume
```

```
metadata:...
```

```
kubectll getpodmeb - 0NAMEREADYSTATUSRESTARTSweb - 00/1
```

```
:
```

event 的结果如下。

```
= monitors:
```

```
$ kubectl describe pod web-0...
```

展开 ▾



kevinsu

2019-05-05



第二次实验遇到一个恶心的问题，MountVolume.WaitForAttach failed for volume "pv-volume" : fail to check rbd image status with: (executable file not found in \$PATH),
rbd output: ()