

讲堂 > 深入剖析Kubernetes > 文章详情

29 | PV、PVC体系是不是多此一举？从本地持久化卷谈起

2018-10-29 张磊



29 | PV、PVC体系是不是多此一举？从本地持久化卷谈起

朗读人：张磊 15'44" | 7.21M

你好，我是张磊。今天我和你分享的主题是：PV、PVC 体系是不是多此一举？从本地持久化卷谈起。

在上一篇文章中，我为你详细讲解了 PV、PVC 持久化存储体系在 Kubernetes 项目中的设计和实现原理。而在文章最后的思考题中，我为你留下了这样一个讨论话题：像 PV、PVC 这样的用法，是不是有“过度设计”的嫌疑？

比如，我们公司的运维人员可以像往常一样维护一套 NFS 或者 Ceph 服务器，根本不必学习 Kubernetes。而开发人员，则完全可以靠“复制粘贴”的方式，在 Pod 的 YAML 文件里填上 Volumes 字段，而不需要去使用 PV 和 PVC。

实际上，如果只是为了职责划分，PV、PVC 体系确实不见得比直接在 Pod 里声明 Volumes 字段有什么优势。

不过，你有没有想过这样一个问题，如果[Kubernetes 内置的 20 种持久化数据卷实现](#)，都没办法满足你的容器存储需求时，该怎么办？

这个情况乍一听起来有点不可思议。但实际上，凡是鼓捣过开源项目的读者应该都有所体会，“不能用”“不好用”“需要定制开发”，这才是落地开源基础设施项目的三大常态。

而在持久化存储领域，用户呼声最高的定制化需求，莫过于支持“本地”持久化存储了。

也就是说，用户希望 Kubernetes 能够直接使用宿主机上的本地磁盘目录，而不依赖于远程存储服务，来提供“持久化”的容器 Volume。

这样做的好处很明显，由于这个 Volume 直接使用本地磁盘，尤其是 SSD 盘，它的读写性能相比于大多数远程存储来说，要好得多。这个需求对本地物理服务器部署的私有 Kubernetes 集群来说，非常常见。

所以，Kubernetes 在 v1.10 之后，就逐渐依靠 PV、PVC 体系实现了这个特性。这个特性的名字叫作：Local Persistent Volume。

不过，首先需要明确的是，**Local Persistent Volume 并不适用于所有应用**。事实上，它的适用范围非常固定，比如：高优先级的系统应用，需要在多个不同节点上存储数据，并且对 I/O 较为敏感。典型的应用包括：分布式数据存储比如 MongoDB、Cassandra 等，分布式文件系统比如 GlusterFS、Ceph 等，以及需要在本地磁盘上进行大量数据缓存的分布式应用。

其次，相比于正常的 PV，一旦这些节点宕机且不能恢复时，Local Persistent Volume 的数据就可能丢失。这就要求**使用 Local Persistent Volume 的应用必须具备数据备份和恢复的能力**，允许你把这些数据定时备份在其他位置。

接下来，我就为你深入讲解一下这个特性。

不难想象，**Local Persistent Volume 的设计，主要面临两个难点**。

第一个难点在于：如何把本地磁盘抽象成 PV。

可能你会说，Local Persistent Volume，不就等同于 hostPath 加 NodeAffinity 吗？

比如，一个 Pod 可以声明使用类型为 Local 的 PV，而这个 PV 其实就是一个 hostPath 类型的 Volume。如果这个 hostPath 对应的目录，已经在节点 A 上被事先创建好了。那么，我只需要再给这个 Pod 加上一个 nodeAffinity=nodeA，不就可以使用这个 Volume 了吗？

事实上，**你绝不应该把一个宿主机上的目录当作 PV 使用**。这是因为，这种本地目录的存储行为完全不可控，它所在的磁盘随时都可能被应用写满，甚至造成整个宿主机宕机。而且，不同的本地目录之间也缺乏哪怕最基础的 I/O 隔离机制。

所以，一个 Local Persistent Volume 对应的存储介质，一定是一块额外挂载在宿主机的磁盘或者块设备（“额外”的意思是，它不应该是宿主机根目录所使用的主硬盘）。这个原则，我们可以称为“一个 PV 一块盘”。

第二个难点在于：调度器如何保证 Pod 始终能被正确地调度到它所请求的 Local Persistent Volume 所在的节点上呢？

造成这个问题的原因在于，对于常规的 PV 来说，Kubernetes 都是先调度 Pod 到某个节点上，然后，再通过“两阶段处理”来“持久化”这台机器上的 Volume 目录，进而完成 Volume 目录与容器的绑定挂载。

可是，对于 Local PV 来说，节点上可供使用的磁盘（或者块设备），必须是运维人员提前准备好的。它们在不同节点上的挂载情况可以完全不同，甚至有的节点可以没这种磁盘。

所以，这时候，调度器就必须能够知道所有节点与 Local Persistent Volume 对应的磁盘的关联关系，然后根据这个信息来调度 Pod。

这个原则，我们可以称为“**在调度的时候考虑 Volume 分布**”。在 Kubernetes 的调度器里，有一个叫作 VolumeBindingChecker 的过滤条件专门负责这个事情。在 Kubernetes v1.11 中，这个过滤条件已经默认开启了。

基于上述讲述，**在开始使用 Local Persistent Volume 之前，你首先需要在集群里配置好磁盘或者块设备**。在公有云上，这个操作等同于给虚拟机额外挂载一个磁盘，比如 GCE 的 Local SSD 类型的磁盘就是一个典型例子。


而在我们部署的私有环境中，你有两种办法来完成这个步骤。

- 第一种，当然就是给你的宿主机挂载并格式化一个可用的本地磁盘，这也是最常规的操作；
- 第二种，对于实验环境，你其实可以在宿主机上挂载几个 RAM Disk（内存盘）来模拟本地磁盘。

接下来，我会使用第二种方法，在我们之前部署的 Kubernetes 集群上进行实践。

首先，在名叫 node-1 的宿主机上创建一个挂载点，比如 /mnt/disks；**然后**，用几个 RAM Disk 来模拟本地磁盘，如下所示：

```
1 # 在 node-1 上执行
2 $ mkdir /mnt/disks
3 $ for vol in vol1 vol2 vol3; do
4     mkdir /mnt/disks/$vol
5     mount -t tmpfs $vol /mnt/disks/$vol
6 done
```

 复制代码

需要注意的是，如果你希望其他节点也能支持 Local Persistent Volume 的话，那就需要为它们也执行上述操作，并且确保这些磁盘的名字（vol1、vol2 等）都不重复。

接下来，我们就可以为这些本地磁盘定义对应的 PV 了，如下所示：

```

1 apiVersion: v1
2 kind: PersistentVolume
3 metadata:
4   name: example-pv
5 spec:
6   capacity:
7     storage: 5Gi
8   volumeMode: Filesystem
9   accessModes:
10  - ReadWriteOnce
11  persistentVolumeReclaimPolicy: Delete
12  storageClassName: local-storage
13  local:
14    path: /mnt/disks/vol1
15  nodeAffinity:
16    required:
17      nodeSelectorTerms:
18      - matchExpressions:
19        - key: kubernetes.io/hostname
20          operator: In
21          values:
22        - node-1

```

复制代码

可以看到，这个 PV 的定义里：local 字段，指定了它是一个 Local Persistent Volume；而 path 字段，指定的正是这个 PV 对应的本地磁盘的路径，即：/mnt/disks/vol1。

当然了，这也就意味着如果 Pod 要想使用这个 PV，那它就必须运行在 node-1 上。所以，在这个 PV 的定义里，需要有一个 nodeAffinity 字段指定 node-1 这个节点的名字。这样，调度器在调度 Pod 的时候，就能够知道一个 PV 与节点的对应关系，从而做出正确的选择。**这正是 Kubernetes 实现“在调度的时候就考虑 Volume 分布”的主要方法。**

接下来，我们就可以使用 kubectl create 来创建这个 PV，如下所示：

```

1 $ kubectl create -f local-pv.yaml
2 persistentvolume/example-pv created
3
4 $ kubectl get pv
5 NAME          CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS    CLAIM          STORAGECLASS
6 example-pv    5Gi       RWO           Delete          Available  local-storage


```

复制代码

可以看到，这个 PV 创建后，进入了 Available（可用）状态。

而正如我在上一篇文章里所建议的那样，使用 PV 和 PVC 的最佳实践，是你要创建一个 StorageClass 来描述这个 PV，如下所示：

```
1 kind: StorageClass
2 apiVersion: storage.k8s.io/v1
3 metadata:
4   name: local-storage
5 provisioner: kubernetes.io/no-provisioner
6 volumeBindingMode: WaitForFirstConsumer
```

 复制代码

这个 StorageClass 的名字，叫作 local-storage。需要注意的是，在它的 provisioner 字段，我们指定的是 no-provisioner。这是因为 Local Persistent Volume 目前尚不支持 Dynamic Provisioning，所以它没办法在用户创建 PVC 的时候，就自动创建出对应的 PV。也就是说，我们前面创建 PV 的操作，是不可以省略的。

与此同时，这个 StorageClass 还定义了一个 volumeBindingMode=WaitForFirstConsumer 的属性。它是 Local Persistent Volume 里一个非常重要的特性，即：**延迟绑定**。

我们知道，当你提交了 PV 和 PVC 的 YAML 文件之后，Kubernetes 就会根据它们俩的属性，以及它们指定的 StorageClass 来进行绑定。只有绑定成功后，Pod 才能通过声明这个 PVC 来使用对应的 PV。

可是，如果你使用的是 Local Persistent Volume 的话，就会发现，这个流程根本行不通。

比如，现在你有一个 Pod，它声明使用的 PVC 叫作 pvc-1。并且，我们规定，这个 Pod 只能运行在 node-2 上。

而在 Kubernetes 集群中，有两个属性（比如：大小、读写权限）相同的 Local 类型的 PV。

其中，第一个 PV 的名字叫作 pv-1，它对应的磁盘所在的节点是 node-1。而第二个 PV 的名字叫作 pv-2，它对应的磁盘所在的节点是 node-2。

假设现在，Kubernetes 的 Volume 控制循环里，首先检查到了 pvc-1 和 pv-1 的属性是匹配的，于是就将它们俩绑定在一起。

然后，你用 kubectl create 创建了这个 Pod。

这时候，问题就出现了。

调度器看到，这个 Pod 所声明的 pvc-1 已经绑定了 pv-1，而 pv-1 所在的节点是 node-1，根据“调度器必须在调度的时候考虑 Volume 分布”的原则，这个 Pod 自然会被调度到 node-1

上。

可是，我们前面已经规定过，这个 Pod 根本不允许运行在 node-1 上。所以。最后的结果就是，这个 Pod 的调度必然会失败。

这就是为什么，在使用 Local Persistent Volume 的时候，我们必须想办法推迟这个“绑定”操作。

那么，具体推迟到什么时候呢？

答案是：推迟到调度的时候。

所以说，StorageClass 里的 volumeBindingMode=WaitForFirstConsumer 的含义，就是告诉 Kubernetes 里的 Volume 控制循环（“红娘”）：虽然你已经发现这个 StorageClass 关联的 PVC 与 PV 可以绑定在一起，但请不要现在就执行绑定操作（即：设置 PVC 的 VolumeName 字段）。

而要等到第一个声明使用该 PVC 的 Pod 出现在调度器之后，调度器再综合考虑所有的调度规则，当然也包括每个 PV 所在的节点位置，来统一决定，这个 Pod 声明的 PVC，到底应该跟哪个 PV 进行绑定。

这样，在上面的例子里，由于这个 Pod 不允许运行在 pv-1 所在的节点 node-1，所以它的 PVC 最后会跟 pv-2 绑定，并且 Pod 也会被调度到 node-2 上。


所以，通过这个延迟绑定机制，原本实时发生的 PVC 和 PV 的绑定过程，就被延迟到了 Pod 第一次调度的时候在调度器中进行，从而保证了这个**绑定结果不会影响 Pod 的正常调度**。

当然，在具体实现中，调度器实际上维护了一个与 Volume Controller 类似的控制循环，专门负责为那些声明了“延迟绑定”的 PV 和 PVC 进行绑定工作。

通过这样的设计，这个额外的绑定操作，并不会拖慢调度器的性能。而当一个 Pod 的 PVC 尚未完成绑定时，调度器也不会等待，而是会直接把这个 Pod 重新放回到待调度队列，等到下一个调度周期再做处理。

在明白了这个机制之后，我们就可以创建 StorageClass 了，如下所示：

```
1 $ kubectl create -f local-sc.yaml
2 storageclass.storage.k8s.io/local-storage created
```

 复制代码

接下来，我们只需要定义一个非常普通的 PVC，就可以让 Pod 使用到上面定义好的 Local Persistent Volume 了，如下所示：

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

[复制代码](#)

可以看到，这个 PVC 没有任何特别的地方。唯一需要注意的是，它声明的 `storageClassName` 是 `local-storage`。所以，将来 Kubernetes 的 Volume Controller 看到这个 PVC 的时候，不会为它进行绑定操作。

现在，我们来创建这个 PVC：

```
1 $ kubectl create -f local-pvc.yaml
2 persistentvolumeclaim/example-local-claim created
3
4 $ kubectl get pvc
5 NAME                STATUS    VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS    AGE
6 example-local-claim Pending                                local-storage  7s
```

[复制代码](#)

可以看到，尽管这个时候，Kubernetes 里已经存在了一个可以与 PVC 匹配的 PV，但这个 PVC 依然处于 Pending 状态，也就是等待绑定的状态。

然后，我们编写一个 Pod 来声明使用这个 PVC，如下所示：

```
1 kind: Pod
2 apiVersion: v1
3 metadata:
4   name: example-pv-pod
5 spec:
6   volumes:
7     - name: example-pv-storage
8     persistentVolumeClaim:
9       claimName: example-local-claim
10  containers:
11    - name: example-pv-container
12      image: nginx
13      ports:
14        - containerPort: 80
15          name: "http-server"
16      volumeMounts:
```

[复制代码](#)

```

17     - mountPath: "/usr/share/nginx/html"
18     name: example-pv-storage

```

这个 Pod 没有任何特别的地方，你只需要注意，它的 volumes 字段声明要使用前面定义的、名叫 example-local-claim 的 PVC 即可。

而我们一旦使用 kubectl create 创建这个 Pod，就会发现，我们前面定义的 PVC，会立刻变成 Bound 状态，与前面定义的 PV 绑定在了一起，如下所示：

```

1 $ kubectl create -f local-pod.yaml
2 pod/example-pv-pod created
3
4 $ kubectl get pvc
5 NAME                STATUS    VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   A
6 example-local-claim  Bound    example-pv  5Gi        RWO            local-storage  C

```

复制代码

也就是说，在我们创建的 Pod 进入调度器之后，“绑定”操作才开始进行。

这时候，我们可以尝试在这个 Pod 的 Volume 目录里，创建一个测试文件，比如：

```

1 $ kubectl exec -it example-pv-pod -- /bin/sh
2 # cd /usr/share/nginx/html
3 # touch test.txt

```

复制代码

然后，登录到 node-1 这台机器上，查看一下它的 /mnt/disks/vol1 目录下的内容，你就可以看到刚刚创建的这个文件：

```

1 # 在 node-1 上
2 $ ls /mnt/disks/vol1
3 test.txt

```

复制代码

而如果你重新创建这个 Pod 的话，就会发现，我们之前创建的测试文件，依然被保存在这个持久化 Volume 当中：

```

1 $ kubectl delete -f local-pod.yaml
2
3 $ kubectl create -f local-pod.yaml
4
5 $ kubectl exec -it example-pv-pod -- /bin/sh
6 # ls /usr/share/nginx/html
7 # touch test.txt

```

复制代码

这就说明，像 Kubernetes 这样构建出来的、基于本地存储的 Volume，完全可以提供容器持久化存储的功能。所以，像 StatefulSet 这样的有状态编排工具，也完全可以通过声明 Local 类型的 PV 和 PVC，来管理应用的存储状态。

需要注意的是，我们上面手动创建 PV 的方式，即 Static 的 PV 管理方式，在删除 PV 时需要按如下流程执行操作：

1. 删除使用这个 PV 的 Pod；
2. 从宿主机移除本地磁盘（比如，umount 它）；
3. 删除 PVC；
4. 删除 PV。

如果不按照这个流程的话，这个 PV 的删除就会失败。

当然，由于上面这些创建 PV 和删除 PV 的操作比较繁琐，Kubernetes 其实提供了一个 Static Provisioner 来帮助你管理这些 PV。

比如，我们现在的所有磁盘，都挂载在宿主机的 /mnt/disks 目录下。

那么，当 Static Provisioner 启动后，它就会通过 DaemonSet，自动检查每个宿主机的 /mnt/disks 目录。然后，调用 Kubernetes API，为这些目录下面的每一个挂载，创建一个对应的 PV 对象出来。这些自动创建的 PV，如下所示：

```

1 $ kubectl get pv
2 NAME          CAPACITY   ACCESSMODES  RECLAIMPOLICY  STATUS   CLAIM   STOR
3 local-pv-ce05be60  1024220Ki  RW0          Delete         Available
4
5 $ kubectl describe pv local-pv-ce05be60
6 Name:  local-pv-ce05be60
7 ...
8 StorageClass:  local-storage
9 Status:  Available
10 Claim:
11 Reclaim Policy:  Delete
12 Access Modes:  RW0
13 Capacity:  1024220Ki
14 NodeAffinity:
15   Required Terms:
16     Term 0:  kubernetes.io/hostname in [node-1]
17 Message:
18 Source:
19   Type:  LocalVolume (a persistent volume backed by local storage on a node)
20   Path:  /mnt/disks/vol1

```

复制代码

这个 PV 里的各种定义，比如 StorageClass 的名字、本地磁盘挂载点的位置，都可以通过 provisioner 的[配置文件指定](#)。当然，provisioner 也会负责前面提到的 PV 的删除工作。

而这个 provisioner 本身，其实也是一个我们前面提到过的[External Provisioner](#)，它的部署方法，在[对应的文档里](#)有详细描述。这部分内容，就留给你课后自行探索了。

总结

在今天这篇文章中，我为你详细介绍了 Kubernetes 里 Local Persistent Volume 的实现方式。

可以看到，正是通过 PV 和 PVC，以及 StorageClass 这套存储体系，这个后来新添加的持久化存储方案，对 Kubernetes 已有用户的影响，几乎可以忽略不计。作为用户，你的 Pod 的 YAML 和 PVC 的 YAML，并没有任何特殊的改变，这个特性所有的实现只会影响到 PV 的处理，也就是由运维人员负责的那部分工作。

而这，正是这套存储体系带来的“解耦”的好处。

其实，Kubernetes 很多看起来比较“繁琐”的设计（比如“声明式 API”，以及我今天讲解的“PV、PVC 体系”）的主要目的，都是希望为开发者提供更多的“可扩展性”，给使用者带来更多的“稳定性”和“安全感”。这两个能力的高低，是衡量开源基础设施项目水平的重要标准。

思考题

正是由于需要使用“延迟绑定”这个特性，Local Persistent Volume 目前还不能支持 Dynamic Provisioning。你是否能说出，为什么“延迟绑定”会跟 Dynamic Provisioning 有冲突呢？

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。



The banner features a light blue background. On the left, there is a logo for '极客时间' (Geek Time) in orange and white. Below the logo, the title '深入剖析 Kubernetes' is written in large, bold, blue font. Underneath the title, the subtitle 'Kubernetes 原来可以如此简单' is written in a smaller, blue font. At the bottom left, the author's name '张磊' is displayed in bold, followed by his credentials 'Kubernetes 社区 资深成员与项目维护者' in a smaller font. On the right side of the banner, there is a portrait of a young man with glasses and a white t-shirt, smiling.

©版权归极客邦科技所有，未经许可不得转载

上一篇 28 | PV、PVC、StorageClass, 这些到底在说啥?

下一篇 30 | 编写自己的存储插件: FlexVolume与CSI

写留言

精选留言



realc

👍 2

知其然，知其所以然。很多教程教材，就跟上学时学校直接灌给我们一样，要让我们去硬啃。不如老师这课程，一个知识点，一个功能的来龙去脉、前世今生都给讲的清清楚楚的。

2018-10-29



liuchjlu

👍 1

请教一个问题，当使用Local Persistent Volume的时候，pv中声明的local path如果所在节点没有这个目录会不会自动创建？

2018-11-03



虎虎 ❤️

👍 1

思考题，我的理解：

因为当一个pvc创建之后，kubernetes因为dynamic provisioning机制会调用pvc指定的storageclass里的provisioner自动使用local disk的信息去创建pv。而且pv一旦创建，nodeaffinity参数就指定了固定的node。而此时，provisioner并没有pod调度的相关信息。

延迟绑定发生的时机是pod已经进入调度器。此时对应的pv已经创建，绑定了node。并可能与pod的调度信息发生冲突。

如果dynamic provisioning机制能够推迟到pod调度的阶段，同时考虑pod调度条件和node硬件信息，这样才能实现dynamic provisioning。实现上可以参考延迟绑定，来一个延迟provision。另外实现一个controller在pod调度阶段创建pv。

2018-10-29



A-

👍 0

如果是容器化的kubelet要如何解决本地PV的问题，因为发现kubelet写的的数据，都写在了容器里了，而没有在宿主机上。

2018-11-08

作者回复

这是你volume没弄好吧.....

2018-11-09



看不穿

👍 0

老师，问两个问题，关于开发人员用镜像发版的问题：

- 1, 之后即使改一个文件，也要重新制作镜像文件？
- 2, 镜像文件相对来说size比较大，上传会很耗时吧？
有没有解决办法？

2018-10-30



silver

👍 0

'一个 PV 一块盘'能再解释下么，如果直接写宿主机上的本地磁盘目录只要把每个container所消耗的硬盘空间都加个cap就能防止磁盘被写满吧？

2018-10-30



shaoboai

👍 0

k8S部署kfk，es可以用local persistent volume吗

2018-10-29

| 作者回复

可以，但是没有持久战性

2018-10-30