# Kubernetes 常见开源分布式存储选型对比

## 1. 性能对比测试

测试结果参考medium社区[文章](https://medium.com/volterra-io/kubernetes-storage-performance-comparison-v2-2020-updated-1c0b69f0dcf4)

参照列表:

- GlusterFS（开源）

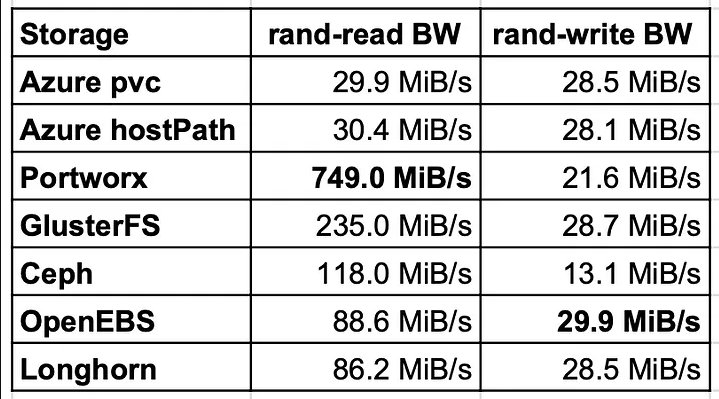
- Ceph（开源）

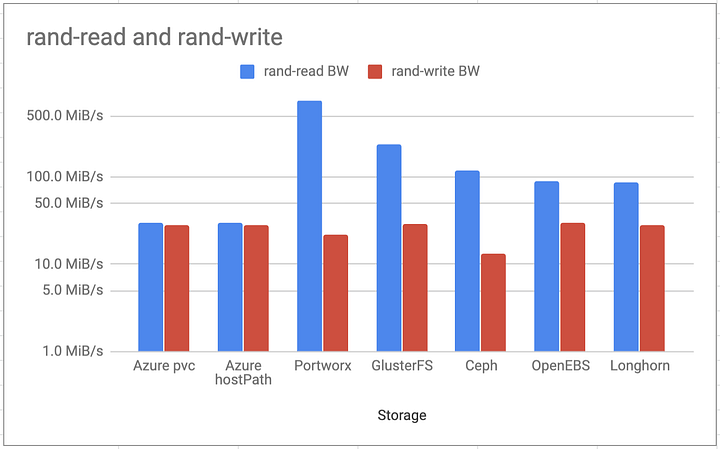
- OpenEBS （开源）

- Longhorn（开源）

备注：其他三款为商业版本，在此仅供对比参考

### **1.1 随机读写带宽**





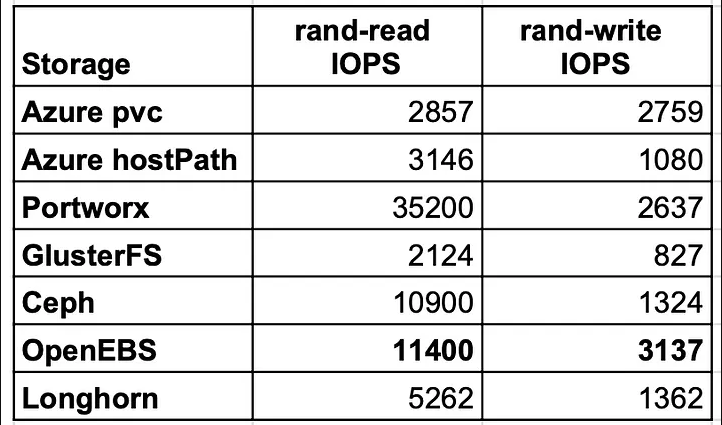
随机读写带宽结果：

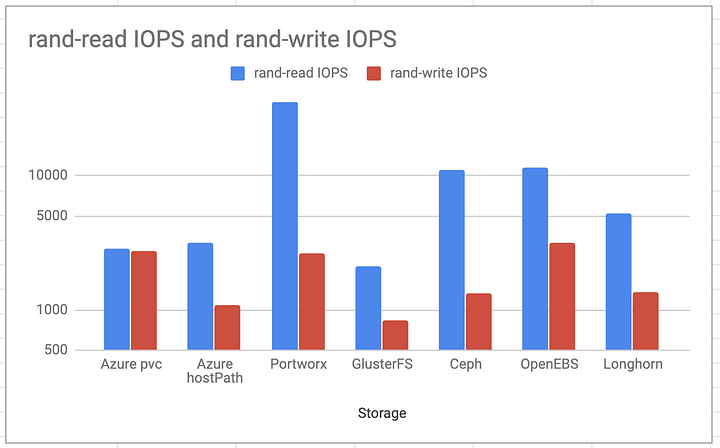
- 随机读BW： GlusterFS > Ceph 1.3x> OpenEBS > Longhorn > Azure hostPath > Azure pvc

- 随机写BW: Ceph 2x> OpenEBS

- OpenEBS和Longhorn的性能几乎是本地磁盘的两倍。原因是读取了缓存

### **1.2 随机读写IOPS**



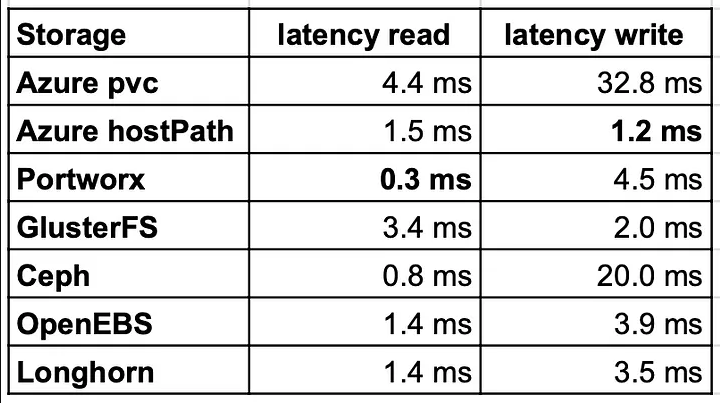


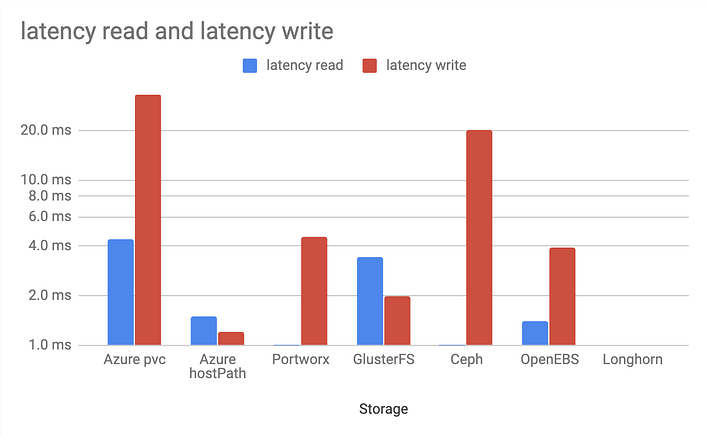
随机读写结果：

- 随机读IOPS：Portworx 3x> OpenEBS = Ceph 2x> Longhorn

- 随机写IOPS: OpenEBS 2.3x> Portworx > Ceph > Longhorn

### **1.3 读写延迟**

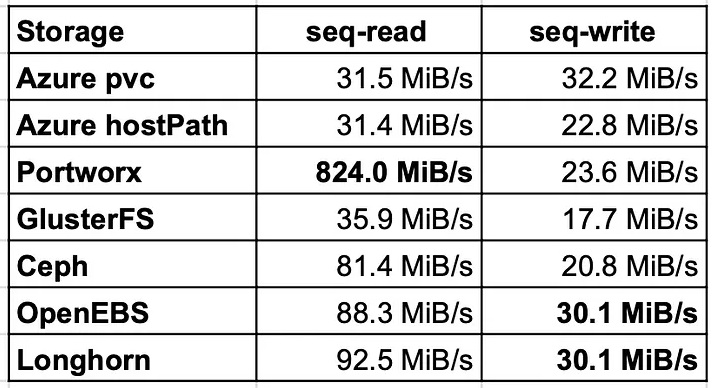


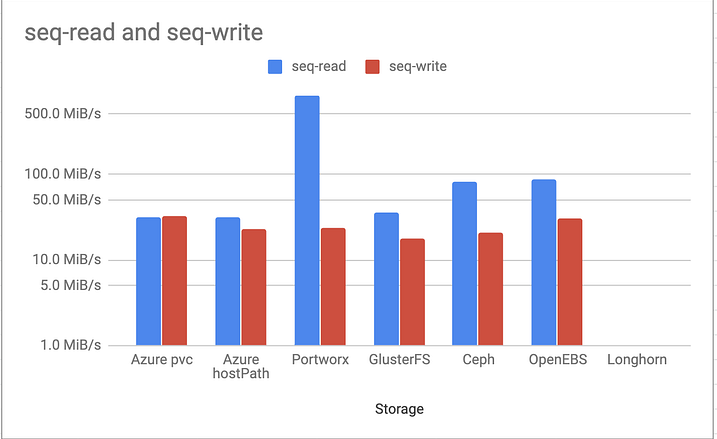


读写延迟结果

- OpenEBS和Longhorn上写入的延迟更好。 GlusterFS比其他存储更好。

### **1.4 顺序读/写**





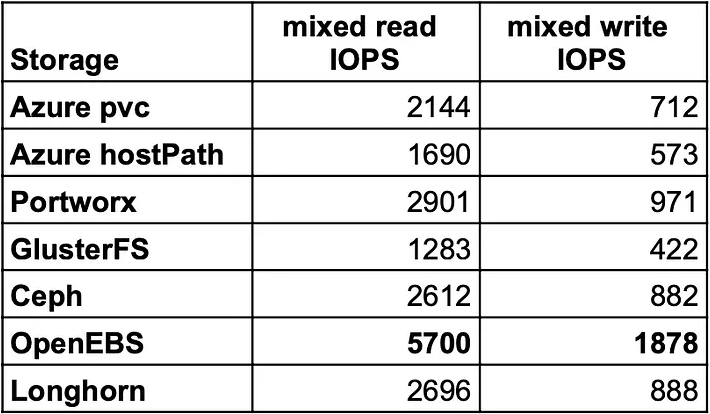
结果：

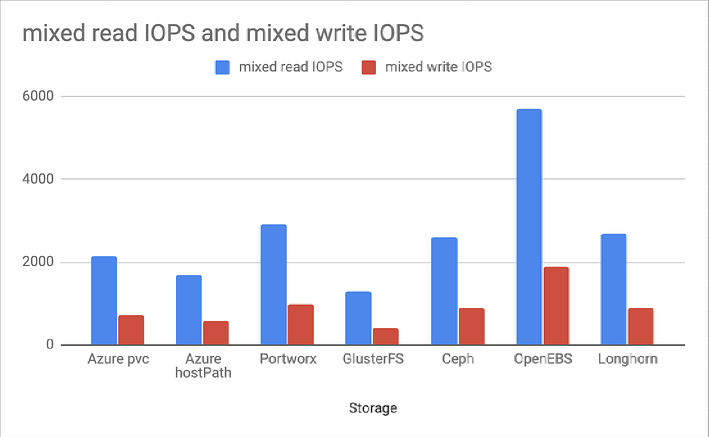
- 顺序读/写测试显示的结果与随机测试相似

- 顺序读：Longhorn ≈ OpenEBS ≈ Ceph

- 顺序写：Longhorn ≈ OpenEBS 1.5x> Ceph

### **1.5 混合读/写IOPS**





结果：

- 在读写方面，OpenEBS交付的速度几乎是PortWorx或Longhorn的两倍。

- 混合读写 ：OpenEBS 2x> Ceph = Longhorn

### 1.6 性能对比总结

* OpenEBS：围绕NVMe的稳健设计，OpenEBS似乎已成为最快的开源容器存储选项之一。OpenEBS的Jiva存储引擎是基于块设备的存储解决方案，它直接利用Kubernetes节点上的本地块设备。由于直接访问本地块设备，Jiva可以提供较低的延迟和较高的IOPS性能。
* Ceph：设计目标是提供高可用性和数据冗余，而不是追求最佳的单个节点性能。由于Ceph涉及复杂的数据分布和网络通信，它的性能可能会受到一些额外的开销。如何仅考虑性能维度，在Ceph与OpenEBS之间做出选择，那么Ceph更适合读多写少的场景。
* Longhorn：在小块IO和随机IO的场景下表现良好，它适用于测试和开发环境，简单易用
* GlusterFS：现阶段不建议k8s的后端存储使用GlusterFS. 在某些场景中可能遇到一些挑战，例如对于大规模、高吞吐量和低延迟的工作负载来说，可能需要进行一些优化和调整。

## 2. 功能特性对比

参考云解决方案提供商SmartX的[文章](https://zhuanlan.zhihu.com/p/644480430)

在存储方案特性方面，Gartner 在《如何在容器与 Kubernetes 环境进行存储选型和实践（How Do I Approach Storage Selection and Implementation for Containers and Kubernetes Deployments）?》报告中，列举了 5 项云原生数据服务对传统存储方案的改进要求：

* 软件定义与“硬件无关（hardware agnostic）”。
* 可编程，可作为“基础设施即代码（IaC）”进行管理，由 API 驱动并支持高级和细粒度的数据服务（如高可用与数据保护）。
* 基于分布式架构，可以任意规模部署。
* 与多种 Kubernetes 发行版通过认证，并可与之进行互操作、实现完全集成。
* 具有简单且可预测的跨环境许可模型。

参考以上要求，对Longhorn，Ceph，OpenEBS，以及商业版的Portworx四款产品，从技术开闭源、本土化支持、存储架构、高级数据服务、与k8s的集成程度等方面进行了全面对比：

### 2.1 对比图表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能特性 | | Longhorn | Ceph | OpenEBS | Portworx(商业版) |
| **是否开源** | | **是** | **是** | **部分** | **否** |
| **商业支持** | | **Suse/Rancher** | **本土Ceph商业化厂商** | **DataCore**  **（中国本土无支持）** | **Portworx**  **（中国本土无支持）** |
| **融合部署资源占用** | | **Volume越多**  **消耗越大** | **磁盘越多**  **OSD消耗越大** | **Volume越多**  **消耗越大** | **暂无公开资料** |
| **存储访问模式支持** | | **块** | **块、文件、对象** | **块、文件** | **块** |
| **内核依赖** | | **否** | **是**  **（需要rbd内核模块、使用FS storage quota功能需要内核>4.17）** | **否** | **是**  **（部署时需要确保内核兼容性）** |
| **数据分布策略** | **局部化** | **不支持** | **不支持** | **不支持** | **暂无资料** |
| **容量均衡** | **支持** | **支持** | **支持** | **支持** |
| **条带化** | **不支持** | **支持** | **支持** | **不支持** |
| **高可用与数据保护主要特性** | **节点内部** | **不支持** | **支持** | **不支持** | **暂无资料** |
| **跨节点** | **副本** | **副本** | **副本** | **副本** |
| **备份与容灾** | **支持备份到外部NFS和S3存储** | **支持** | **支持，通过velero实现** | **备份和同步、异步的完整支持** |
| **容器编排平台支持** | | **任何CNCF认证的k8s发行版** | **任何CNCF认证的k8s发行版，版本需要v1.21以上** | **任何CNCF认证的k8s发行版，版本需要v1.18以上** | **任何CNCF认证的k8s发行版，主流的公有云平台** |
| **PV认证和鉴权** | | **支持** | **不支持** | **不支持** | **支持** |

基于以上对比，在功能特性层面的选型上需要注意以下几点：

* 关注数据安全与合规性的用户（如金融与政府机构），应尽量选择闭源 k8s存储方案。同时，闭源厂商往往具备更强的存储核心代码支持能力，基于开源存储技术的方案，若遇到存储部分的故障，可能无法得到及时解决。
* OpenEBS 与 Portworx 当前没有中国本土支持，更考验运维人员在 K8s 环境方面的知识与经验储备。
* 若采用 K8s 支持对 I/O 性能要求较高、数据一致性要求较强的应用场景（数据库、消息中间件、缓存等），应尽量选择支持块存储的解决方案。
* 运行对连续性要求较高的应用，应尽量选择具备高可用与数据保护的产品。

### 2.2 功能对比总结

* **Longhorn** 在小规模集群使用时，性价比较高，配置简单，但是兼容性会存在一定问题，比如没用正式支持xfs，默认支持ext4并在此条件下运行可靠
* **Ceph**是一个功能强大的分布式存储系统，功能支持、性能以及可靠性综合方面最好，适用于需求更加综合和广泛的业务场景。
* **OpenEBS**更关注容器化环境下的数据持久化和存储管理，提供了用于容器存储的多种特性，如数据隔离、快照、副本等，如果只需要基本的块存储，OpenEBS在部署和维护方面相比Ceph来说会更简单，性价比更高
* **GlusterFS，**功能扩展和兼容性较差，适用于大量数据的场景，提供了并行读写和数据分布的能力，对于需要存储大型多媒体文件（音视频）的应用场景，其可提供高性能和可扩展的存储，以满足对大容量和高带宽的存储需求。而对于小文件处理能力有限。

## 3. 社区支持和生态系统对比

### 3.1 对比图表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Ceph | OpenEBS | Longhorn | GlusterFS |
| 发行时间 | 2004年 | 2016年 | 2019年 | 2005年 |
| Github Stars | 12k ☆ | 8.2k ☆ | 4.8k ☆ | 4.2k☆ |
| 社区支持度 | 最好 | 好 | 良好 | 一般 |
| 版本活跃度 | v18.1.3 | v3.8.0 | v1.5.1 | v10.4 |

### 3.2 社区支持对比总结

1. GlusterFS的生态远远没有Ceph的好，如监控工具GlusterFS-exporter和增删改卷的工具heketi，目前官方均停止维护。在使用周边工具时可能会遇到各种问题，如在使用GlusterFS-exporter时会查看所有节点的uuid,将最大的uuid作为全面收集信息的对象，而其它节点会缺少部分信息。对于集群规模的volume status命令是需要所有节点来通信的，如果存在两三百个volume,对每个volume进行exporter操作，那么通信负担将会非常大。

社区资料比较匮乏。GlusterFS目前暂无官方权威指南，意味着只能靠网上的资料或者源码来深入了解GlusterFS，此外社区讨论的版本还停留在老版本如3.x居多，社区活跃度较差。尤其对于国内社区。大部分情况下，如何要在Ceph和GlusterFS中选择的话，Ceph都是目前k8s存储选型中比较好的选择。

## 4. 部署和维护成本对比

1. Longhorn配置最简单，可靠性高，支持快照、备份，对于简单的块存储用例，是非常有效的选择，其与OpenEBS Jiva相似。

2. GlusterFS因为其较少的功能扩展，维护相对简单，但是为了避免后期维护成本过高，建议加装支持k8s的存储设备，如支持k8s csi快速启动的块存储和支持s3对象存储的接口的相关设备。

3．OpenEBS相对简单，可通过Helm Chart和Operator进行快速部署和管理。

4. Ceph的硬件要求和维护成本较高，在部署和后期管理阶段较为困难，需要有专业人员做支撑。对于去中心化的分布式解决方案，需要提前做好规划设计，但是Ceph的一个很大的优点是只需要维护一个系统，可以把所有的东西放在一个盒子里:块存储、对象存储、文件存储，进行统一维护。

# 附录

## Ceph使用指南（硬件方向）

### 1.1 CPU

- Ceph 元数据处理器应该有相当大的处理能力（四核心或更高的CPU）

- Ceph OSDs 运行RADOS服务，用CRUSH计算数据放置、复制数据，并维护自己的集群地图副本。因此，OSD应该有合理的处理能力（例如双核处理器）

- Ceph monitor 守护进程维护了 clustermap，它不给客户端提供任何数据，因此它是轻量级的，并没有严格的处理器要求，普通的单核服务器处理器即可。但必须考虑机器以后是否还会运行 Ceph 监视器以外的 CPU 密集型任务。例如，如果服务器以后要运行用于计算的虚拟机（如 OpenStack Nova ），你就要确保给 Ceph 进程保留了足够的处理能力，所以我们推荐在其他机器上运行 CPU 密集型任务。

**一个 Ceph OSD 守护进程需要相当数量的处理性能，因为它提供数据给客户端。要评估 Ceph OSD 的 CPU 需求，知道服务器上运行了多少 OSD 上非常重要的。通常建议每个 OSD 进程至少有一个核心。可以通过以下公式计算 OSD 的 CPU 需求：**

（（CPU sockets \* CPU cores per socket \* CPU clock speed in GHz ）/ No.of OSD ） >= 1

case 1: Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz 6 core --> 1 \* 6 \* 2.40 = 14.4 适合多达 14 个 OSD 的 Ceph 节点

case 2: Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz 12 core --> 1 \* 12 \* 2.50 = 30 适合多达 30 个 OSD 的 Ceph 节点

**如果打算使用 Ceph 纠删码特性，最好能有一个更高性能的CPU ，因为运行纠删码需要更强的处理能力。**

### 1.2 RAM

**| Monitors and managers (ceph-mon and ceph-mgr)**

- 监视器和管理器守护进程的内存使用量一般会随着集群的大小而变化。

- 元数据服务器和监视器必须可以尽快地提供它们的数据，对于小型集群，一般来说，1-2GB就足够了。对于大型集群，应该提供更多（5-10GB）。

- 你可能还需要考虑调整设置，如mon\_osd\_cache\_size或 rocksdb\_cache\_size

**| OSDs (ceph-osd)**

OSD 的日常运行不需要那么多内存（如每进程 500MB ）差不多了；然而在恢复期间它们占用内存比较大（如每进程每 TB 数据需要约 1GB 内存）。通常内存越多越好。

### 1.3 Memory

- 通常不建议将osd\_memory\_target设置为2GB以下，可能会将内存保持在2GB以下，同时也可能导致性能极慢。

- 将内存目标设置在2Gb和4Gb之间通常有效，但可能会导致性能下降，因为元数据可能在IO期间从磁盘读取，除非活动数据集相对较小。

- 4GB是目前默认的osd\_memory\_target大小，这样设置的目的是为了平衡内存需求和OSD的性能，以满足典型的使用情况

- 设置osd\_memory\_target高于4GB时，当有许多（小的）或大的（256GB/OSD)数据集被处理时，可能会提高性能。

**重要：**

**OSD的内存自动尽力而为。虽然OSD可能会解除内存映射，让内核回收内存，但不能保证内核会在任何特定的时间框架内实际回收释放的内存。**

**这在旧版本的Ceph中尤其如此，因为透明的巨页会阻止内核从碎片化的巨页中回收内存。**

**现代版本的Ceph在应用级禁用透明巨页以避免这种情况，但这仍然不能保证内核会立即回收未映射的内存。OSD有时仍然可能会超过它的内存目标。**

tips: 我们建议在系统中保留20%左右的额外内存，以防止OSD在临时高峰期或由于内核延迟回收空闲页而导致的OSD出现OOM。这个值可能会比需要的多或少取决于系统的具体配置。

在使用传统的FileStore后端时，页面缓存是用来缓存数据的，所以一般不需要调优，OSD的内存消耗一般与系统中每个守护进程的PG数量有关

### **1.4 Data Storage**

要谨慎地规划数据存储配置，在规划数据存储时，需要考虑较高的成本和性能权衡。来自操作系统的并行操作和到单个硬盘的多个守护进程并发读、写请求操作会极大地降低性能。

重要提示：因为 Ceph 发送 ACK 前必须把所有数据写入日志（至少对 xfs 和 ext4 来说是），因此均衡日志和 OSD的性能相当重要。

**| HDD**

- Ceph 最佳实践指示，你应该分别在单独的硬盘运行操作系统、 OSD 数据和 OSD 日志。

- 推荐独立的驱动器用于安装操作系统和软件，另外每个 OSD 守护进程占用一个驱动器

**| SDD**

- 使用固态硬盘（ SSD ）可以降低\*\*随机访问时间和读延时，同时增加吞吐量。

- 在大量投入 SSD 前，强烈建议核实 SSD 的性能指标，并在测试环境下衡量性能。

- SSD 很适合 Ceph 里占存储空间较少的的数据部分

- 用于日志和 SSD 时还有几个重要考量：

- 写密集语义：日志记录涉及写密集语义，所以要确保选用的 SSD 写入性能和好于或等于HDD。廉价 SSD 可能在加速访问的同时引入写延时，有时候高性能HDD的写入速度可以和便宜 SSD 相媲美。

- 顺序写入：在一个 SSD 上为多个 OSD 存储多个日志时也必须考虑 SSD 的顺序写入极限，因为它们要同时处理多个 OSD 日志的写入请求。

- 分区对齐：采用了 SSD 的一个常见问题是人们喜欢分区，却常常忽略了分区对齐，这会导致 SSD 的数据传输速率慢很多，所以请确保分区对齐了。

- 成本：通过将OSD的日志存储在固态硬盘上，并将OSD的对象存储存储在独立的机械硬盘上，可能会看到性能的显著提升，并降低成本

**| 其他考虑因素**

- 确保你的OSD硬盘的总吞吐量之和不超过服务于客户端读取或写入所需的网络带宽

- 应该考虑集群在每台主机上存储的数据占整体数据的百分比。如果某个特定主机上的百分比很大，而该主机出现故障，可能会导致超过 full ratio等问题，从而导致Ceph停止工作，作为防止数据丢失的安全规范措施。

- 每个主机上运行多个OSD时，你还需要确保内核是最新的。以确保你的硬件在每个主机上运行多个OSD时，能按照预期的方式执行。

- OSD 数量较多（如 20 个以上）的主机会派生出大量线程，尤其是在恢复和重均衡期间。很多 Linux 内核默认的最大线程数较小（如 32k 个），如果您遇到了这类问题，可以把 `kernel.pid\_max` 值调高些。理论最大值是 4194303 。例如把下列这行加入 `/etc/sysctl.conf` 文件：

**kernel.pid\_max = 4194303**

- 通常，大量的小容量OSD节点比少量的大容量OSD节点要好，但这并不是定论，应该选择适当的 Ceph 节点密度，使得三个节点容量小于总容量的 10%。例如：在一个 1PB的ceph集群中，应该避免使用 4 个 250TB 的 OSD 节点，因为每个节点占用了 25% 的集群容量。相反，可以使用 13 个 80TB 的 OSD节点，每个节点容量小于集群容量的 10%。

### 1.5 故障域

故障域是指任何阻止一个或多个OSD的故障。这可能是主机上的守护进程停止；硬盘故障、操作系统崩溃、网卡故障、电源故障、网络中断、断电等等。在规划硬件需求的时候，你必须平衡一下，把大部分的职能集中于低故障域中来降低成本，以及隔离每个潜在故障域所带来的额外成本

## 2. OpenEBS使用指南

### 2.1 OpenEBS 是什么？

OpenEBS 是一种开源云原生存储解决方案，托管于 CNCF 基金会，目前该项目处于沙箱阶段。

OpenEBS 是一组存储引擎，允许您为有状态工作负载 (StatefulSet) 和 Kubernetes 平台类型选择正确的存储解决方案。在高层次上，OpenEBS 支持两大类卷——本地卷和复制卷。

OpenEBS 是 Kubernetes 本地超融合存储解决方案，它管理节点可用的本地存储，并为有状态工作负载提供本地或高可用的**分布式持久卷**。作为一个完全的 Kubernetes 原生解决方案的另一个优势是，管理员和开发人员可以使用 **kubectl、Helm、 Prometheus、Grafana、Weave Scope** 等 Kubernetes 可用的所有优秀工具来交互和管理 OpenEBS。

### 2.2 OpenEBS 能做什么？

OpenEBS 管理 k8s 节点上存储，并为 k8s 有状态负载（StatefulSet）提供本地存储卷或分布式存储卷。

* 本地卷（Local Storage）

1. OpenEBS 可以使用宿主机裸块设备或分区，或者使用 Hostpaths 上的子目录，或者使用 LVM、ZFS 来创建持久化卷
2. 本地卷直接挂载到 Stateful Pod 中，而不需要 OpenEBS 在数据路径中增加任何开销
3. OpenEBS 为本地卷提供了额外的工具，用于监控、备份 / 恢复、灾难恢复、由 ZFS 或 LVM 支持的快照等

* 对于分布式卷 (即复制卷)

1. OpenEBS 使用其中一个引擎 (Mayastor、cStor 或 Jiva) 为每个分布式持久卷创建微服务
2. 有状态 Pod 将数据写入 OpenEBS 引擎，OpenEBS 引擎将数据同步复制到集群中的多个节点。OpenEBS 引擎本身作为 Pod 部署，并由 Kubernetes 进行协调。当运行 Stateful Pod 的节点失败时，Pod 将被重新调度到集群中的另一个节点，OpenEBS 将使用其他节点上的可用数据副本提供对数据的访问
3. 有状态的 Pods 使用 iSCSI (cStor 和 Jiva) 或 NVMeoF (Mayastor) 连接 OpenEBS 分布式持久卷
4. OpenEBS cStor 和 Jiva 专注于存储的易用性和持久性。它们分别使用自定义版本的 ZFS 和 Longhorn 技术将数据写入存储。OpenEBS Mayastor 是最新开发的以耐久性和性能为设计目标的引擎，高效地管理计算 (大页面、核心) 和存储 (NVMe Drives)，以提供快速分布式块存储

“注意：OpenEBS 分布式块卷被称为复制卷，以避免与传统的分布式块存储混淆，传统的分布式块存储倾向于将数据分布到集群中的许多节点上。复制卷是为云原生有状态工作负载设计的，这些工作负载需要大量的卷，这些卷的容量通常可以从单个节点提供，而不是使用跨集群中的多个节点分片的单个大卷

### 2.3 对比传统分布式存储

OpenEBS 与其他传统存储解决方案不同的几个关键方面 :

* 使用微服务体系结构构建，就像它所服务的应用程序一样。OpenEBS 本身作为一组容器部署在 Kubernetes 工作节点上。使用 Kubernetes 本身来编排和管理 OpenEBS 组件
* 完全建立在用户空间，使其高度可移植性，以运行在任何操作系统 / 平台。
* 完全意图驱动，继承了 Kubernetes 易用性的相同原则
* OpenEBS 支持一系列存储引擎，因此开发人员可以部署适合于其应用程序设计目标的存储技术。像 Cassandra 这样的分布式应用程序可以使用 LocalPV 引擎进行最低延迟的写操作。像 MySQL 和 PostgreSQL 这样的单片应用程序可以使用使用 NVMe 和 SPDK 构建的 Mayastor 或基于 ZFS 的 cStor 来实现弹性。像 Kafka 这样的流媒体应用程序可以在边缘环境中使用 NVMe 引擎 Mayastor 以获得最佳性能。

#### 2.3.1 本地卷类型

本地卷只能从集群中的单个节点访问。必须在提供卷的节点上调度使用 Local Volume 的 Pods。本地卷通常是分布式工作负载的首选，比如 Cassandra、MongoDB、Elastic 等，这些工作负载本质上是分布式的，并且内置了高可用性（分片）

根据附加到 Kubernetes 工作节点上的存储类型，您可以从不同的动态本地 PV 进行选择——Hostpath、Device、LVM、ZFS 或 Rawfile

#### 2.3.2 可复制卷类型

复制卷顾名思义，是指将数据同步复制到多个节点的卷。卷可以支持节点故障。还可以跨可用性区域设置复制，以帮助应用程序跨可用性区域移动。

复制卷还能够提供像快照、克隆、卷扩展等企业存储特性。复制卷是有状态工作负载 (如 Percona/MySQL、Jira、GitLab 等) 的首选。

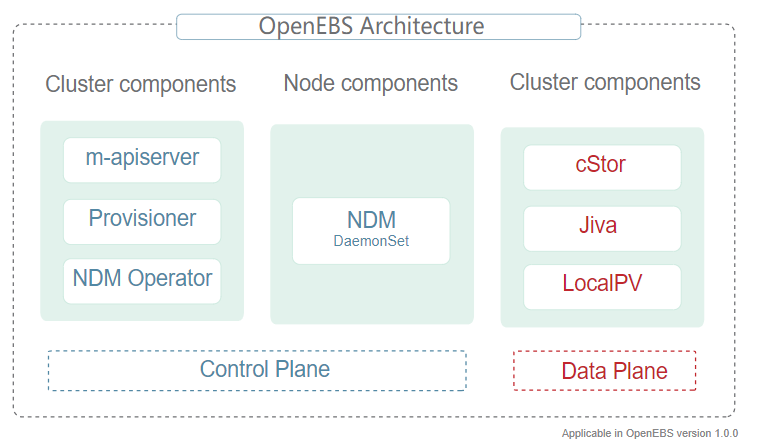
根据附加到 Kubernetes 工作节点的存储类型和应用程序性能需求，您可以从 Jiva、cStor 或 Mayastor 中进行选择

### 2.4 OpenEBS存储引擎建议

|  |  |  |
| --- | --- | --- |
| 应用需求 | 存储类型 | OpenEBS 卷类型 |
| 低时延、高可用性、同步复制、快照、克隆、精简配置 | SSD/ 云存储卷 | OpenEBS Mayastor |
| 高可用性、同步复制、快照、克隆、精简配置 | 机械 /SSD/ 云存储卷 | OpenEBS cStor |
| 高可用性、同步复制、精简配置 | 主机路径或外部挂载存储 | OpenEBS Jiva |
| 低时延、本地 PV | 主机路径或外部挂载存储 | Dynamic Local PV - Hostpath, Dynamic Local PV - Rawfile |
| 低时延、本地 PV | 本地机械 /SSD/ 云存储卷等块设备 | Dynamic Local PV - Device |
| 低延迟，本地 PV，快照，克隆 | 本地机械 /SSD/ 云存储卷等块设备 | OpenEBS Dynamic Local PV - ZFS , OpenEBS Dynamic Local PV - LVM |

### 2.5 OpenEBS架构介绍

OpenESB遵循容器附加存储（CAS）模型，每个卷都有一个专用的控制器POD和一组副本POD。 OpenEBS操作和使用都很简单，因为它看起来和感觉上就像其他云原生和Kubernetes友好的项目。



OpenEBS有许多组件，可以分为以下类别:

* 控制面组件 - Provisioner, API Server, volume exports,volume sidecars
* 数据面组件 - Jiva、cStor
* 节点磁盘管理器 - Discover, monitor, 管理连接k8s的媒介
* 与云原生工具的集成 - 已经与Prometheus,Grafana, Fluentd、Jaeger集成

### 2.6 OpenEBS特性介绍

* 容器附加存储

OpenEBS是一个容器附加存储CAS的例子，通过 OpenEBS 提供的卷总是被容器化。每个卷都有一个专用的存储控制器，用于提高有状态应用程序的持久性存储操作的敏捷性和粒度。

* 同步复制

同步复制是 OpenEBS 的一个可选的流行特性。当与 Jiva、cStor 和 Mayastor 存储引擎一起使用时，OpenEBS 可以同步复制数据卷以实现高可用性。跨 Kubernetes 区域进行复制，从而为跨 AZ 设置提供高可用性。这个特性对于使用 GKE、EKS 和 AKS 等云提供商服务上的本地磁盘构建高可用状态应用程序特别有用

* 快照和克隆

写时拷贝快照是 OpenEBS 另一个可选的流行特性。使用 cStor 引擎时，快照是瞬时创建的，并且不受快照个数的限制。增量快照功能增强了跨 Kubernetes 集群和跨不同云提供商或数据中心的数据迁移和可移植性。对快照和克隆的操作完全以 Kubernetes 原生方法执行，使用标准 kubectl 命令。常见的用例包括用于备份的高效复制和用于故障排除或针对数据的只读副本进行开发的克隆

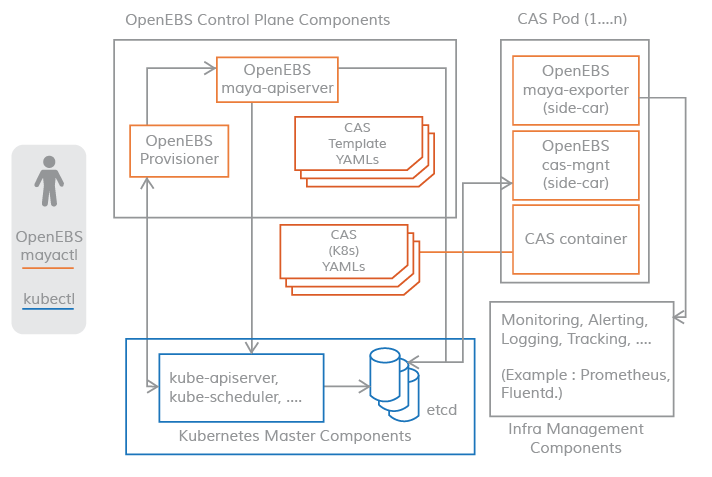
* 备份和恢复

OpenEBS 卷的备份和恢复可以通过开源的 OpenEBS Velero 插件与 Kubernetes 备份和恢复解决方案 (如 Velero(前身为 Heptio Ark)) 协同工作。经常使用 OpenEBS 增量快照功能，将数据备份到 AWS S3、GCP object storage、MinIO 等对象存储目标。这种存储级别的快照和备份只使用增量数据进行备份，节省了大量的带宽和存储空间。

#### 2.6.1 控制面

OpenEBS集群的控制平面通常被称为Maya

OpenEBS控制平面负责提供卷、相关的卷操作，如快照、克隆、创建存储策略、执行存储策略、导出Prometheus/grafana使用的卷指标，等等。



OpenEBS提供了一个动态提供程序，这是Kubernetes的标准外部存储插件。 OpenEBS PV提供者的主要任务是向应用程序PODS启动卷供应，并实现PV的Kubernetes规范。

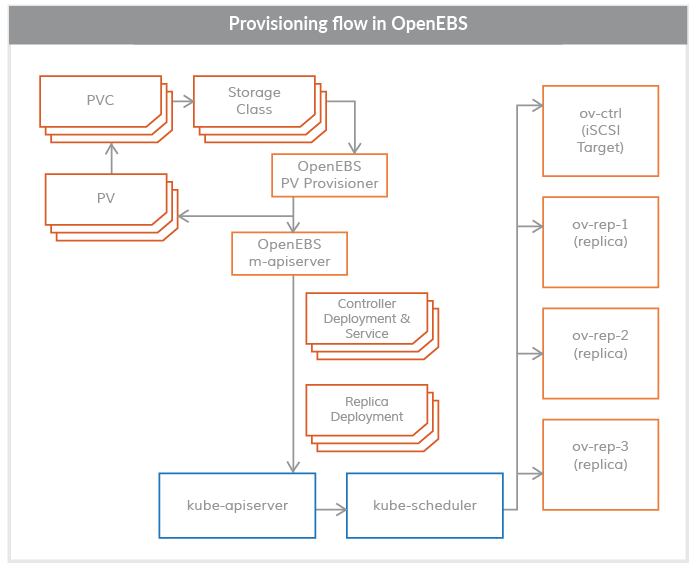
m-apiserver开放存储的REST API，并承担大量卷策略处理和管理工作。

控制平面和数据平面之间的连通性使用Kubernetes sidecar模式。控制平面需要与数据平面通信的场景如下所示。

对于卷的统计，如IOPS，吞吐量，延迟等--通过卷暴漏的sidecar实现

使用卷控制器pod执行卷策略，使用卷副本pod进行磁盘/池管理-通过卷管理sidecar实现

##### OpenEBS PV Provisioner

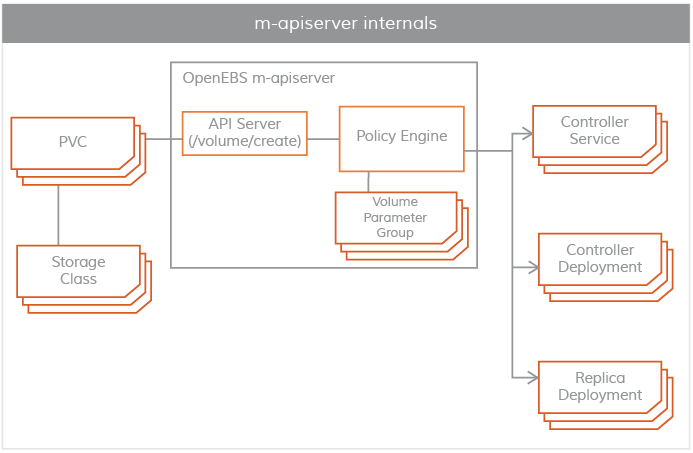


此组件作为POD运行，并做出配置决策,它的使用方式是:

开发人员用所需的卷参数构造一个声明，选择适当的存储类，并在YAML规范上调用kubelet。 OpenEBS PV动态提供程序与maya-apiserver交互，在适当的节点上为卷控制器pod和卷副本pod创建部署规范。 可以使用PVC规范中的注释来控制卷Pod(控制器/副本)的调度。

目前，OpenEBS Provisioner只支持一种绑定类型，即iSCSI。

##### Maya-ApiServer



m-apiserver作为POD运行。顾名思义，m-apiserver公开OpenEBS REST api

m-apiserver还负责创建创建卷pod所需的部署规范文件。 在生成这些规范文件之后，它将调用kube-apiserver来相应地调度这些pods。 OpenEBS PV提供者在卷发放结束时，将创建一个PV对象并将其挂载到应用程序pod上。 PV由控制器pod承载，控制器pod由不同节点中的一组副本pod支持。 控制器pod和复制pod是数据平面的一部分，在存储引擎部分有更详细的描述。

m-apiserver的另一个重要任务是卷策略管理。OpenEBS为表示策略提供了非常细粒度的规范。 m-apiserver解释这些YAML规范，将它们转换为可执行的组件，并通过容量管理sidecar来执行它们

##### Maya Volume Exporter

Maya卷导出器是每个存储控制器pod的sidecar

这些sidecar将控制平面连接到数据平面以获取统计信息。统计信息的粒度在卷级别。一些统计数据示例如下：

卷读取延迟

卷写入延迟

卷每秒读取速度

卷每秒写入速度

读取块大小

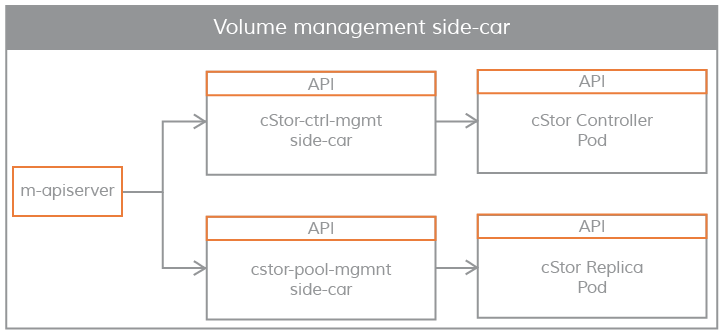
写入块大小

容量统计

这些统计信息通常由Prometheus客户端来拉取，该客户端在OpenBS安装期间安装和配置

##### 卷管理sidecar

Sidecars还用于将控制器配置参数和卷策略传递给卷控制器pod(卷控制器pod是一个数据平面)， 并将副本配置参数和副本数据保护参数传递给卷副本pod。



#### 2.6.2 数据面

OpenEBS数据平面负责实际的卷IO路径。存储引擎在数据平面实现实际的IO路径。 目前，OpenEBS提供了两个可以轻松插入的存储引擎。它们被称为Jiva和cStor。 这两个存储引擎都完全运行在Linux用户空间中，并基于微服务架构。

##### Jiva

Jiva存储引擎基于Rancher's LongHorn与gotgt开发实现， 使用go语言开发，并运行于用户命名空间下。 LongHorn控制器将输入的IO同步复制到LongHorn副本。该副本将Linux稀疏文件视为构建存储特性(如精简配置、快照、重建等)的基础。

##### cStor

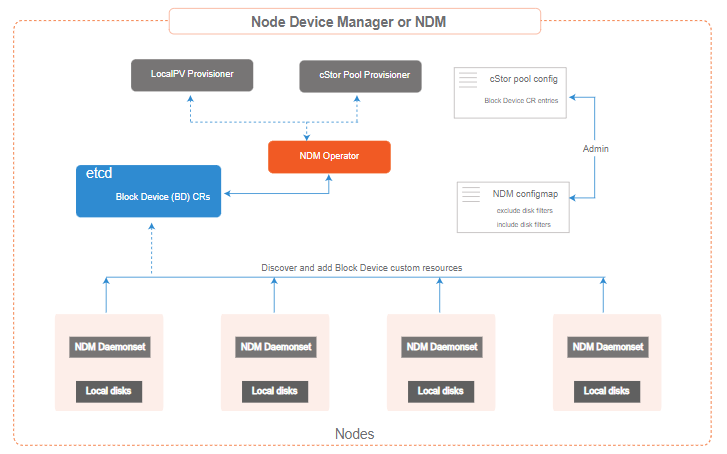
cStor数据引擎使用C语言编写，具有高性能的iSCSI target和Copy-On-Write块系统，提供数据完整性、数据弹性和时间点的快照和克隆。 cStor有一个池特性，它以条带、镜像或RAIDZ模式聚合一个节点上的磁盘，以提供更大的容量和性能单位。 cStor还可以跨区域将数据同步复制到多个节点，从而避免节点丢失或节点重启导致数据不可用。

##### LocalPV

对于那些不需要存储级复制的应用程序，LocalPV可能是很好的选择，因为它能提供更高的性能。 OpenEBS LocalPV与Kubernetes LocalPV类似，不同之处在于它是由OpenEBS控制平面动态提供的， 就像任何其他常规PV一样。OpenEBS LocalPV有两种类型:hostpath LocalPV和device LocalPV。 hostpath LocalPV指的是主机上的子目录，LocalPV指的是在节点上发现的磁盘(可以是直接连接的，也可以是网络连接的)。 OpenEBS引入了一个LocalPV提供者，用于根据PVC和存储类规范中的一些标准选择匹配的磁盘或主机路径。

##### 节点磁盘管理器

节点磁盘管理器(NDM)填补了使用Kubernetes管理有状态应用程序的持久存储所需的工具链的空白。 容器时代的DevOps架构师必须以一种自动化的方式满足应用程序和应用程序开发人员的基础设施需求， 这种方式可以跨环境提供弹性和一致性。这些要求意味着存储堆栈本身必须非常灵活， 以便Kubernetes和云原生生态系统中的其他软件可以轻松地使用这个堆栈。 NDM在Kubernetes的存储堆栈中起着基础性的作用，它统一了不同的磁盘， 并通过将它们标识为Kubernetes对象，提供了将它们汇聚的能力。 此外，NDM发现、提供、监视和管理底层磁盘的方式，可以让Kubernetes PV提供者(如OpenEBS和其他存储系统)和Prometheus管理磁盘子系统



### 2.7 OpenEBS存储引擎对比

OpenEBS提供了三种存储引擎：

#### Jiva

Jiva是OpenEBS 0.1版中发布的第一个存储引擎，使用起来最简单。它基于GoLang开发，内部使用LongHorn和gotgt堆栈。 Jiva完全在用户空间中运行，并提供同步复制等标准块存储功能。 Jiva通常适用于容量较小的工作负载，不适用于大量快照和克隆特性是主要需求的情况

#### cStor

cStor是OpenEBS 0.7版本中最新发布的存储引擎。cStor非常健壮，提供数据一致性，很好地支持快照和克隆等企业存储特性。 它还提供了一个健壮的存储池特性，用于在容量和性能方面进行全面的存储管理。 cStor与NDM (Node Disk Manager)一起，为Kubernetes上的有状态应用程序提供完整的持久化存储特性

#### OpenEBS Local PV

OpenEBS Local PV是一个新的存储引擎，它可以从本地磁盘或工作节点上的主机路径创建持久卷或PV。 CAS引擎可以从OpenEBS的1.0.0版本中获得。使用OpenEBS Local PV，性能将等同于创建卷的本地磁盘或文件系统(主机路径)。 许多云原生应用程序可能不需要复制、快照或克隆等高级存储特性，因为它们本身就提供了这些特性。这类应用程序需要以持久卷的形式访问管理的磁盘



大多数场景推荐cStor，因其提供了强大的功能，包括快照/克隆、存储池功能（如精简资源调配、按需扩容等）。

Jiva适用于低容量需求的工作负载场景，例如5到50G。 尽管使用Jiva没有空间限制，但建议将其用于低容量工作负载。 Jiva非常易于使用，并提供企业级容器本地存储，而不需要专用硬盘。 有快照和克隆功能的需求的场景，优先考虑使用cStor而不是Jiva。

### 2.7 总结

* 多机环境，如果有额外的块设备（非系统盘块设备）作为数据盘，选用 OpenEBS Mayastor、OpenEBS cStor
* 多机环境，如果没有额外的块设备（非系统盘块设备）作为数据盘，仅单块系统盘块设备，选用 OpenEBS Jiva
* 单机环境，建议本地路径 Dynamic Local PV - Hostpath, Dynamic Local PV - Rawfile，由于单机多用于测试环境，数据可靠性要求较低。

由此看来，OpenEBS 常用场景为以上三个场景

## 3. Longhorn使用指南

### 3.1 Longhorn 架构介绍

## 4. GlusterFS使用指南

### 4.1 GlusterFS架构介绍

GlusterFS采用无中心架构，各节点之间采用全互联模式，也就意味着通信带宽消耗要求会比master-slave模式高很多。

另外GlusterFS更加适合做冷数据存储集群，也就是一些重要数据压缩打包以后放在那里。在调研过程中发现，目前使用Glusterfs集群的场景，很多上层应用的支持非常不好，例如nexus官方已经明确说明不支持该存储，GlusterFS这种架构的支持度存在诸多问题。

### 4.2 Gluste**r**FS