Cinder 可启动卷缓存机制浅析技术文档 V1.0

Apr 14, 2016

Cinder 可启动卷缓存机制浅析技术文档 V1.0

术语和称谓

Volume: 虚拟的块设备，本文中称为“卷”
Image-Volume: 导入了虚拟机启动镜像的卷，称为“可启动卷”
Volume-backed Image: 以卷作为镜像后端

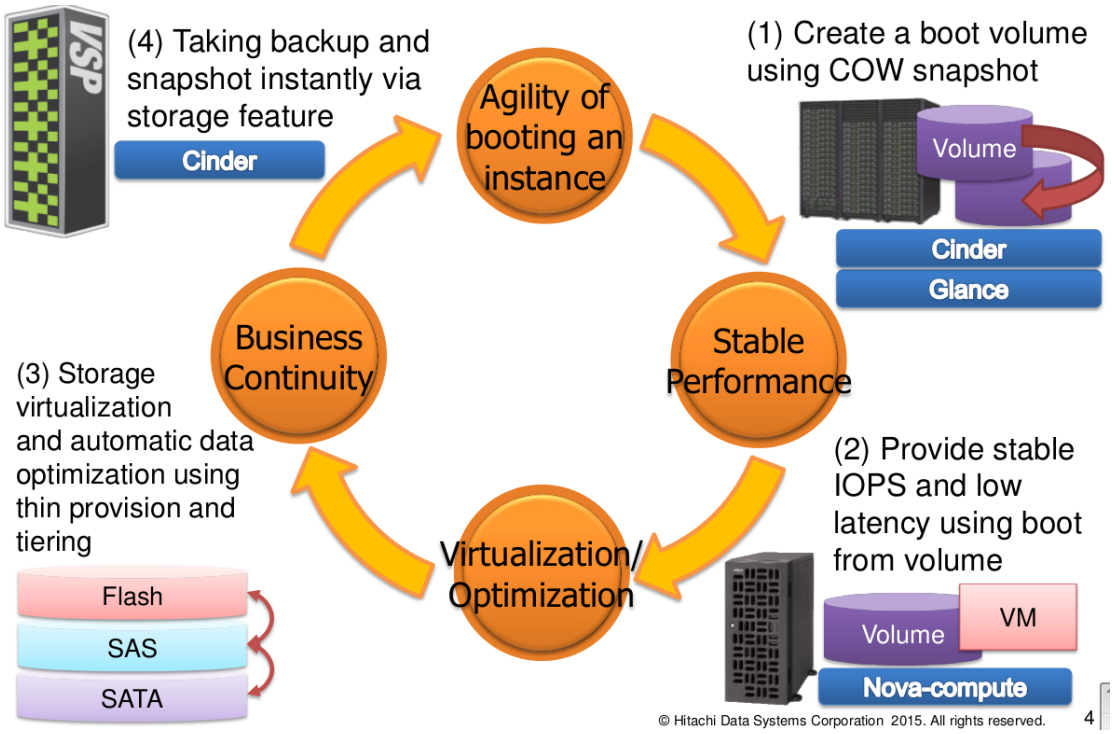
可启动卷的应用场景

Cinder 是 OpenStack 中提供块存储(Block Storage)管理服务的模块，同其他模块一样，Cinder 仅提供管理而不提供真正的存储能力，该能力由其后端来提供，Cinder 可以适配多种后端存储，从而达到灵活管理。

卷是 Cinder 管理的基本资源，一般的，卷被挂载到虚拟机上为存储数据所用，其生命周期与虚拟机相分离，因而被视为“永久的数据存储”。另外 OpenStack 还提供了从卷启动(Boot from volume)的功能，即由可启动卷来启动虚拟机。

从卷启动的优和劣

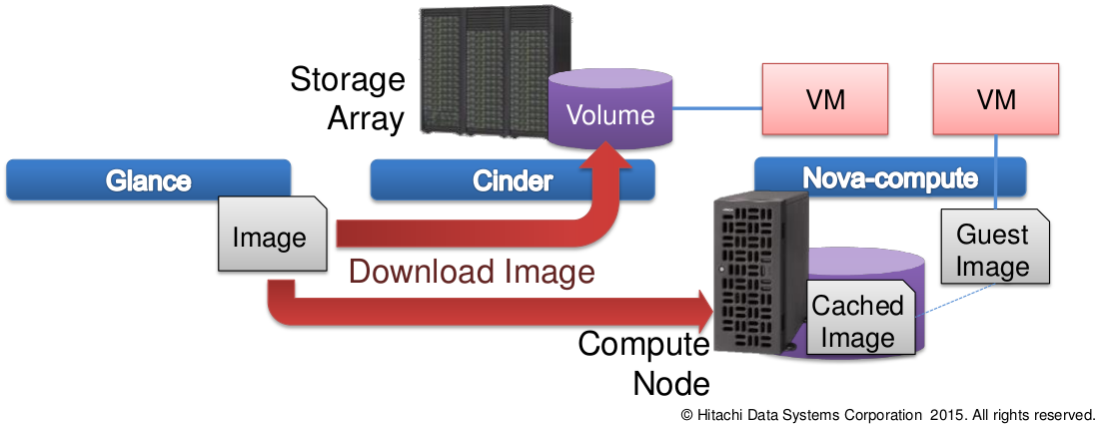
从卷启动为 OpenStack 对接块存储设备提供了一种解决方案，而且有很多优势：



由卷启动虚拟机的实验该过程如下：

- 创建可启动卷：cinder create --image-id --display-name a-image-volume
- 由卷启动虚拟机：nova boot --flavor --boot-volume name （可通过--block-device-mapping 添加更多参数，详见官方文档）

过程图解：



如图所示，对比由镜像启动和由卷启动虚拟机，两种方式都需要向 Glance 请求下载镜像文件，但由于镜像缓存的作用，每个计算节点每个镜像仅需要请求一次;而由卷启动则每次都需要下载镜像。那么问题来了，如果镜像文件比较大，则启动虚拟机的耗时也会比较长;如果要批量启动虚拟机，Glance API 服务的压力也会非常大。

为了解决上面的问题，社区提供了两种方案：

- 可启动卷缓存
- 以卷作为镜像后端

下面主要介绍第一种：可启动卷缓存

可启动卷缓存的机制

缓存什么？

对于 Cinder 来说，保存虚拟机的镜像文件显然并不是它该做的事情，但是保存了镜像文件的卷管理起来就游刃有余了。而且由于虚拟卷可以任意的 Extend，只缓存一份刚好满足镜像大小的卷，然后在需要的时候扩展成所需的大小就可以了。

缓存到哪里？

对于所有资源，OpenStack 都是基于项目(Project/Tenant)来管理。缓存的卷要归属于哪个项目呢？一般用户的项目显然是不合适的，好在 OpenStack 有一个部署下来就被创建但是一般不怎么用的项目： Service。

何时缓存？

通过上面图中的可启动卷的创建过程可以知道，由创建好的卷 Clone 出来一份做缓存就好。

如何匹配缓存？

在新创建可启动卷时，要想知道当前镜像是否被缓存过，那就需要借助数据库记录了：只需要在创建卷缓存时，将镜像与缓存的关联保存到表中，请求再次来到时，查询当前镜像是不是存在表中即可。

过程

以上几个问题大致概括了卷缓存的要点，下面顺序的描述该过程(忽略 API 接收和 Schedule 过程，直接进入卷创建逻辑):

Liberty 版本中默认不启用卷缓存特性，需要另外配置：

```
 vim  /etc/cinder/cinder.conf
image_volume_cache_enabled  =  True     #  是否启用镜像缓存功能
image_volume_cache_max_size_gb  =  100  #  镜像缓存的总容量
image_volume_cache_max_count  =  10  #  镜像缓存的总个数
cinder_internal_tenant_project_id  =  0f7d9f0f103642df9b1c2def87bca48
6  #  cinder 用户所在的项目(即 service 项目)id
cinder_internal_tenant_user_id  =  8bb3be9e49f049a69088c39aa0a81266
#  cinder 用户 id
```

- 数据库中新建 image_volume_cache_entries 表，用以保存卷和镜像的关联;
- 当由镜像创建卷时，根据数据库记录判断是否由该镜像创建过可启动卷：
  - 若未创建过(或之前的卷不在当前主机上)则向 glance 请求下载镜像，将镜像数据转入新卷中，然后由该新卷 clone 一个缓存卷(归属于所配置的 Service 项目);更新数据库表中的缓存记录;
  - 若创建过(并且卷在当前主机上)，则由缓存卷 clone 出新卷(该 clone 功能直接由底层服务提供),若新卷与请求的卷大小不一致，则扩展(Extend)该卷;

其他问题

- 关于缓存的总容量：
  - 若当前镜像的大小超过总容量大小，则不缓存镜像;
  - 若当前镜像大小未超过总容量大小，但是超过当前可用的缓存容量大小，则按照'最久未使用'的方式删除已经缓存的卷，直至可用容量满足当前镜像大小(当触发该条件时可能导致卷的创建变慢，需事前做好容量预估);
  - 若当前镜像大小未超过总容量大小，并且当前可用的缓存容量大小能够满足当前镜像，则直接创建镜像缓存;
- 除了配置缓存容量和个数，缓存卷的创建还会受到 service 项目 volume 配额的限制;
- 若缓存卷被删除，再由镜像创建卷会回到“下载镜像->创建卷->创建缓存卷”的过程

总结

上面对 Cinder 可启动卷缓存机制的描述，都来自对源码的阅读和实践，为方便理解忽略了很多细节，若感兴趣可参阅源码。

缓存机制并不能完全解决创建可启动卷的性能问题，因为本质上是将以前的镜像下载过程改为卷的 Clone 过程，其效率依赖于后端存储的 Clone 性能。如果在 Ceph 环境中实测是采用 COW 模式的 Clone，那么速度很快;但是若改为 flatten 模式，速度也就降下来了。