# 第3章

# 云存储

## 技能目标

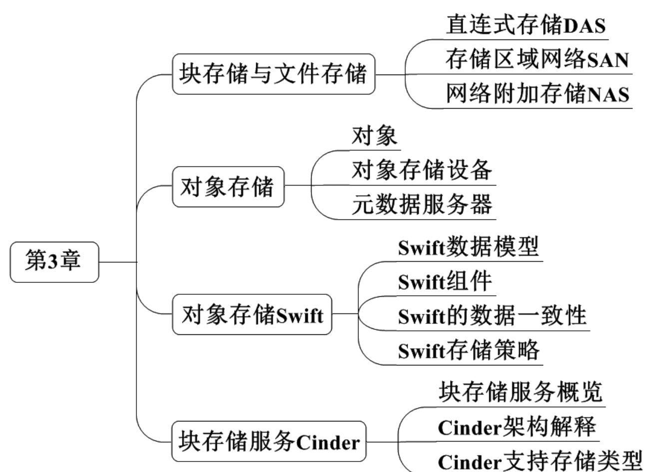
- 理解对象存储与块存储的概念
- 理解 swift 的核心概念
- 理解 cinder 的核心概念

## 本章导读

市面上常见的主流的存储类型包括块存储与文件存储，DAS 和 SAN 都是典型的块存储类型，NAS 是文件级存储。

对象存储（Object-based Storage）是一种新的网络存储架构，兼具 SAN 高速直接访问磁盘特点及 NAS 的分布式共享特点。块存储服务（Cinder）提供块存储，块存储适合性能敏感性业务场景，例如数据库存储。

知识服务






## 3.1 块存储与文件存储

市面上常见的主流的存储类型包括块存储与文件存储。而目前常见的三种存储方式 DAS、SAN、NAS 中 DAS 和 SAN 都是典型的块存储类型，NAS 是文件级存储，均被广泛应用于企业存储设备中。

### 1．直连式存储 DAS

DAS（Direct Attach Storage）是直接连接于主机服务器的一种储存方式，每一台主机服务器有独立的存储设备，每台主机服务器的存储设备无法互通，通常用在单一网络且数据交换量不大，性能要求不高的环境下。

### 2．存储区域网络 SAN

SAN（Storage Area Network）是一种用高速（光纤）网络连接专业主机服务器的一种存储方式，此系统位于主机群的后端，它使用高速 I/O 连接方式，如 SCSI、Fibre Channel。采用 SCSI 块 I/O 的命令集，通过在磁盘或 FC（Fiber Channel）级的数据访问提供高性能的随机 I/O 和数据吞吐率，具有高带宽、低延迟的优势。一般而言，SAN 应用在对网络速度要求高、对数据的可靠性和安全性要求高、对数据共享的性能要求高的应用环境中，在高性能计算中占有一席之地，但是由于 SAN 系统的价格较高，且可扩展性较差，已不能满足成千上万个 CPU 规模的系统。

### 3．网络附加存储 NAS

NAS（Network Attached Storage）：是一套网络存储设备，通常是直接连在网络上并提供资料存取服务，一套 NAS 存储设备就如同一个提供数据文件服务的系统。它采用 NFS 或 CIFS 命令集访问数据，以文件为传输协议，通过 TCP/IP 实现网络化存储，可扩展性好、用户易管理，性价比高，常用在诸如教育、政府、企业等数据存储应用。

目前在集群计算中应用较多的也是NFS文件系统，但由于NAS的协议开销高、带宽低、延迟大，不利于在高性能集群中应用。

针对 Linux 集群对存储系统高性能和数据共享的需求，国际上已开始研究全新的存储架构和新型文件系统，希望能有效结合 SAN 和 NAS 系统的优点，支持直接访问磁盘以提高性能，通过共享的文件和元数据以简化管理。目前对象存储系统已成为 Linux 集群系统中高性能存储系统的研究热点。

## 3.2 对象存储

对象存储（Object-based Storage）是一种新的网络存储架构。兼具 SAN 高速直接访问磁盘的特点及 NAS 分布式共享的特点。其核心是将数据通路（数据读或写）和控制通路（元数据）分离，并且基于对象存储设备（Object-based Storage Device，OSD）构建存储系统，每个对象存储设备具有一定的智能，能够自动管理其上的数据分布。目前国际上通常采用刀片式结构实现对象存储设备。

### 3.2.1 对象存储结构组成

对象存储结构组成包括：对象、对象存储设备、元数据服务器、对象存储系统的客户端几个组成部分。

#### 1. 对象

对象是系统中数据存储的基本单位，一个对象实际上就是文件的数据和一组属性信息（Meta Data）的组合。

#### 2. 对象存储设备

对象存储设备也叫做 OSD（Object-based Storage Device），是一个智能设备，具有自己的 CPU、内存、网络和磁盘系统。它同块设备的区别在于两者提供的访问接口不同。OSD 的主要功能包括数据存储和安全访问。尤其是以下这三个主要功能：

（1）数据存储

OSD 管理对象数据，并将它们放置在标准的磁盘系统上，OSD 并不提供块接口访问方式，而是由 Client 请求数据时用对象 ID、偏移进行数据读写。

（2）智能分布

OSD 用其自身的 CPU 和内存优化数据分布，支持数据的预取。由于 OSD 可以智能地支持对象的预取从而可以优化磁盘的性能。

（3）每个对象元数据的管理

OSD 管理存储在其上对象的元数据，该元数据与传统的 inode 元数据相似，通常包括对象的数据块和对象的长度。而在传统的 NAS 系统中，这些元数据是由文件服务器进行维护。对象存储架构将系统中主要的元数据管理工作由 OSD 来完成，同时也降





云计算部署实战zw.indd 87　　2017-6-20 10:59:42

低了 Client 的开销。

3．元数据服务器

元数据服务器（Metadata Server，MDS）：控制 Client 与 OSD 对象的交互，主要提供以下几个功能：

（1）对象存储访问

MDS 构造、管理描述每个文件分布的视图，为 Client 提供访问该文件所含对象的能力，并且允许 Client 直接访问对象。

OSD 在接收到每个请求时将先验证该能力，然后才可以进行访问。

（2）文件和目录访问管理

MDS 在存储系统上构建一个文件结构，包括限额控制、目录和文件的创建和删除、访问控制等。

（3）Client Cache 一致性

为了提高 Client 性能，在对象存储系统设计时通常支持 Client 方的 Cache。由于引入 Client 方的 Cache，带来了 Cache 一致性问题，当 Cache 的文件发生改变时，将通知 Client 刷新 Cache，从而防止 Cache 不一致引发的问题。

（4）对象存储系统的客户端 Client

为了有效支持 Client 访问 OSD 上的对象，需要在计算节点实现对象存储系统的 Client，通常提供 POSIX 文件系统接口，允许应用程序像执行标准的文件系统操作一样。

## 3.3 对象存储 Swift

Swift 是用来创建可扩展的、冗余的、开源的对象存储（引擎），可以使用标准化的服务器存储 PB 级可用数据。Swift 不是文件系统也不是数据库，而是使用 account-container-object 概念存储 object，适合存储非结构化数据存储，如虚拟机镜像、图片存储、邮件存储、文档备份等。

### 3.3.1 Swift 数据模型

Swift 采用层次数据模型，共设三层逻辑结构：Account/Container/Object（即账户 / 容器 / 对象），每层节点数均没有限制，可以任意扩展，如图 3.1 所示。

具体来说 Account 是出于访问安全性考虑，使用 Swift 系统的每个用户必须有一个账号（Account）。只有通过 Swift 验证的账号才能访问 Swift 系统中的数据。提供账号验证的节点被称为 Account Server。可由 Swauth 提供账号权限认证服务。用户通过账号验证后将获得一个验证字符串（authentication token），后续的每次数据访问操作都需要传递这个字符串。Container 是 Swift 中类似于 Windows 操作系统中的文件夹或者 UNIX 类操作系统中的目录，用于组织管理数据，所不同的是 Container 不能嵌套。数据都以 Object 的形式存放在 Container 中。





云计算部署实战zw.indd 88 2017-6-20 10:59:42

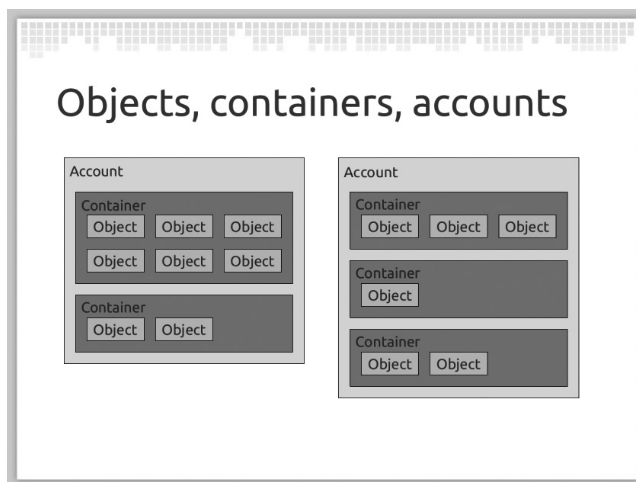



图 3.1　Swift 逻辑结构

简单来说就是 Account 对应租户，用于隔离；Container 对应某个租户数据的存储区域；Object 对应存储区域中具体的 Block。

Swift 存储数据的过程类似文件存储过程，如图 3.2 所示。

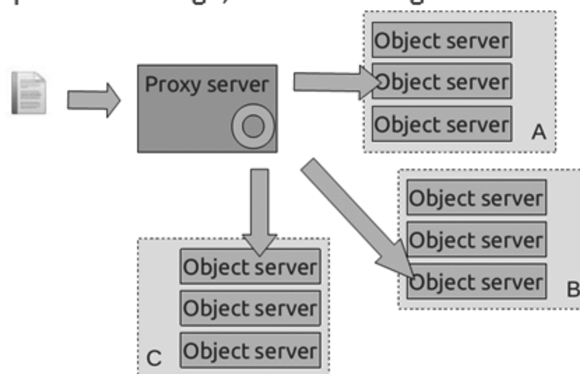



图 3.2　Swift 存储数据过程

比如，我们存储一个文件，会在对应的磁盘块组中对应的元数据区域产生对应的 inode，基于这个 inode 将数据存储到数据区中对应的块组中的 Block。

### 3.3.2　Swift 组件

Swift 并不是单一的存在，它包含很多组件，下面就来具体了解下 Swift 所包含的这些组件。

1. 代理服务（Proxy Service）

代理服务对外提供对象服务 API，会根据环的信息来查找服务地址并转发用户请求至相应的账户、容器或者对象服务；由于采用无状态的 REST 请求协议，可以进行横向扩展来进行均衡负载。

2. 认证服务（Authentication Service）

认证服务用来验证访问用户的身份信息，并获得一个在一定的时间内会一直有效的对象访问令牌（Token），认证服务会验证访问令牌的有效性并缓存下来直至过期时间。

3. 缓存服务（Cache Service）

缓存服务用于内容的缓存，缓存包括对象服务令牌，账户和容器的存在信息，但不会对象本身的数据进行缓存。缓存服务可采用 Memcached 集群来实现，Swift 会使用一致性散列算法来分配 Memcached 缓存地址。

4. 账户服务（Account Service）

账户服务用于提供账户元数据和统计信息，并维护所含容器列表的服务，将每个账户的信息存储在一个 SQLite 数据库中。

5. 容器服务（Container Service）

容器服务用来提供容器元数据和统计信息，同时对所含对象列表的服务进行维护，每个容器的信息也存储在一个 SQLite 数据库中。

6. 对象服务（Object Service）

对象服务用来提供对象元数据和内容服务，每个对象的内容会以文件的形式存储在文件系统中，元数据会作为文件属性来存储，建议采用支持扩展属性的 XFS 文件系统。

7. 复制服务（Replicator）

复制服务会检测本地分区副本和远程副本是否一致，具体是通过对比散列文件和高级水印来完成，发现不一致时会采用推式（Push）更新远程副本，例如对象复制服务会使用远程文件复制工具 rsync 来同步。另外一个任务是确保被标记删除的对象从文件系统中移除。

8. 更新服务（Updater）

更新服务可以当对象由于高负载的原因而无法立即更新时，将任务序列化到本地文件系统中进行排队，以便服务恢复后进行异步更新。例如成功创建对象后容器服务器没有及时更新对象列表，这个时候容器的更新操作就会进入排队中，更新服务会在系统恢复正常后扫描队列并进行相应的更新处理。

9. 审计服务（Auditor）

审计服务用来检查对象、容器和账户的完整性，如果发现比特级的错误，文件将被隔离，并复制其他的副本以覆盖本地损坏的副本；其他类型的错误会被记录到日志中。

10．账户清理服务（Account Reaper）

账户清理服务用来移除被标记为删除的账户，删除其所包含的所有容器和对象。

### 3.3.3　Swift 的数据一致性

存储在 Swift 里面的数据有好几个备份，而且各个节点之间是平等的关系，没有“主节点”这个概念，因此任意一个节点出现故障时，数据并不会丢失。Swift 必须面对的一个问题就是如何保持数据的一致性。因为一个文件并不是只保存一份的，在 Swift 中默认要保存 3 个副本，当更新的时候这 3 个文件要同时更新，当其中一个文件损坏时必须能迅速地复制一份完整的文件来替换。

为了保证 Swift 数据的一致性，Swift 有 3 个服务来解决这个问题：Auditor、Updater 和 Replicator。Auditor 运行在每个 Swift 服务器的后台持续地扫描磁盘来检测数据的完整性。如果发现数据损坏，Auditor 就会将该文件移动到隔离区域，然后由 Replicator 负责用一个完好的副本文件来替代该数据。如果更新失败，该次更新在本地文件系统上会被加入队列，然后 Updaters 会继续处理这些失败了的更新工作。



### 3.3.4　Swift 存储策略

在对象存储中存储的不仅是数据还有丰富的与数据相关的属性信息。系统会给每一个对象分配一个唯一的 ID。对象本身是平等的，所有的 ID 都属于一个平坦的地址空间，而并非文件系统那样的树状逻辑结构，如图 3.3 所示。这种存储结构带来的好处是可以实现数据的智能化管理，因为对象本身包含了元数据信息，甚至更多的属性，我们可以根据这些信息对对象进行高效的管理。

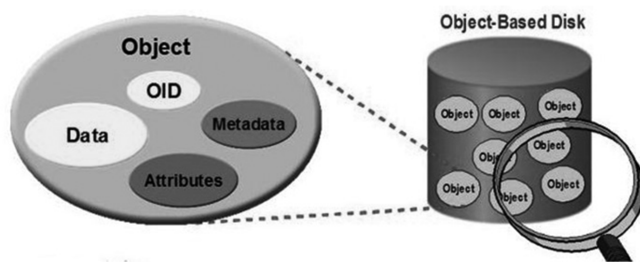



图 3.3　Swift 存储策略

那么 Client 要存储一个文件，用什么策略决定存在哪台 Storage Node 上呢？例如我们可以制定这样的存储策略，如果对象中包含 priority:high 这样的属性，我们就对文件做比平常文件多的备份次数。平坦地址空间的设计使得访问对象只通过一个唯一的 ID 标识即可，不需要复杂的路径结构。因此可以知道 Swift 中没有“路径”这个概念，所以也没有所谓的“文件夹”这样的概念。

那么如何根据对象 ID 把对象存在合适的 Storage Node 呢？



云计算部署实战zw.indd　91　　2017-6-20　10:59:43

解决的方法是使用“一致性 Hash 法”。一致性哈希算法的基本实现原理是将机器节点和 Key（在本文里就是对象的 ID）值都按照一样的 hash 算法映射到一个 0 ～ $2^{32}$ 的圆环上。当有一个写入缓存的请求到来时，计算 Key 值 K 对应的哈希值 Hash(K)，如果该值正好对应之前某个机器节点的 Hash 值，则直接写入该机器节点，如果没有对应的机器节点，则顺时针查找下一个节点，进行写入，如果超过 $2^{32}$ 还没找到对应节点，则从 0 开始查找（因为是环状结构），如图 3.4 所示。

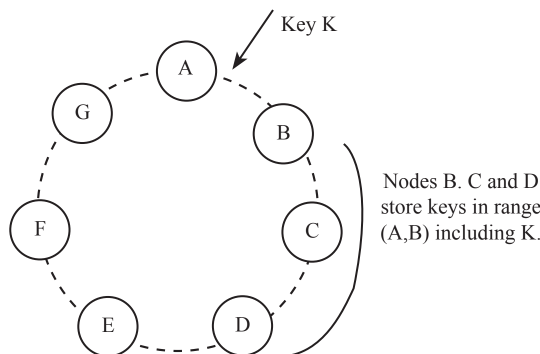



图 3.4　一致性 Hash 法

经过一致性哈希算法散列之后，当有新的机器加入时，将只影响一台机器的存储情况，例如新加入的节点 H 的散列在 B 与 C 之间，则原先由 C 处理的一些数据可能将移至 H 处理，而其他所有节点的处理情况都将保持不变。而如果删除一台机器，例如删除 C 节点，此时原来由 C 处理的数据将移至 D 节点，而其他节点的处理情况仍然不变。而由于在机器节点散列和缓冲内容散列时都采用了同一种散列算法，因此也很好地降低了分散性和负载。

现将所有 Swift 支持的操作总结如表 3-1 所示。

表 3-1　Swift RESTful API 总结

| 资源类型 | URL | GET | PUT | POST | DELETE | HEAD |
|---|---|---|---|---|---|---|
| 账户 | /account/ | 获取容器列表 | | | | 获取账户元数据 |
| 容器 | /account/container | 获取对象列表 | 创建容器 | 更新容器元数据 | 删除容器 | 获取容器元数据 |
| 对象 | /account/container/object | 获取对象内容和元数据 | 创建、更新或复制对象 | 更新对象元数据 | 删除对象 | 获取对象元数据 |

### 3.3.5　对象存储 Swift 补充

#### 1. 副本

如果集群中的数据在本地节点上只有一份，一旦发生故障就可能会造成数据的永





云计算部署实战zw.indd　92　　2017-6-20　10:59:44

久性丢失。因此，需要有冗余的副本来保证数据安全。

Swift 中引入了副本的概念，其默认值为 3，理论依据主要来源于 NWR 策略（也叫 Quorum 协议）。NWR 是一种在分布式存储系统中用于控制一致性级别的策略。

在 Amazon 的 Dynamo 云存储系统中，使用了 NWR 来控制一致性。其中，N 代表同一份数据的副本的份数，W 是更新一个数据对象时需要确保成功更新的份数；R 代表读取一个数据需要读取的副本的份数。公式 W+R>N，保证某个数据不被两个不同的事务同时读和写；公式 W>N/2，保证两个事务不能并发写某一个数据。

在分布式系统中，数据的单点是不允许存在的。即线上正常存在的副本数量为 1 的情况是非常危险的，因为一旦这个副本再次出错，就可能发生数据的永久性错误。假如我们把 N 设置成为 2，那么只要有一个存储节点发生损坏，就会有单点的存在，所以 N 必须大于 2。N 越高，系统的维护成本和整体成本就越高。工业界通常把 N 设置为 3。例如，对于 MySQL 主从结构，其 NWR 数值分别是 N= 2, W = 1, R = 1，没有满足 NWR 策略。而 Swift 的 N=3, W=2, R=2，完全符合 NWR 策略，因此 Swift 系统是可靠的，没有单点故障。



### 2．Zone

如果所有的 Node 都在一个机架或一个机房中，那么一旦发生断电、网络故障等，都将造成用户无法访问。因此需要一种机制对机器的物理位置进行隔离，以满足分区容忍性（CAP 理论中的 P）。

因此，引入了 Zone 的概念，把集群的 Node 分配到每个 Zone 中。其中同一个 Partition 的副本不能同时放在同一个 Node 上或同一个 Zone 内，如图 3.5 所示。注意，Zone 的大小可以根据业务需求和硬件条件自定义，可以是一块磁盘、一台存储服务器，也可以是一个机架甚至一个 IDC。

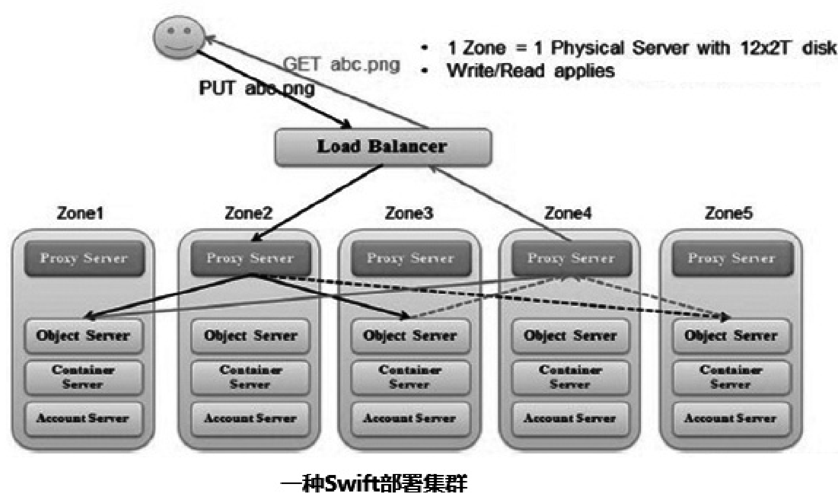



图 3.5　Zone 概念的引入

## 3.4 块存储服务 Cinder

块存储服务（cinder）提供块存储。存储的分配和消耗是由块存储驱动器，或者多后端配置的驱动器决定的。还有很多驱动程序可用：NAS/SAN，NFS，ISCSI，Ceph 等。块存储适合性能敏感性业务场景，例如数据库存储大规模可扩展的文件系统或服务器需要访问到块级的裸设备存储。

典型情况下，块服务 API 和调度器服务运行在控制节点上。取决于使用的驱动，卷服务器可以运行在控制节点、计算节点或单独的存储节点。

### 3.4.1 块存储服务概览

块存储服务为 OpenStack 中的实例提供持久的存储，块存储提供一个基础设施用于管理卷，以及和 OpenStack 计算服务交互，为实例提供卷。此服务也会激活管理卷的快照和卷类型的功能。



### 3.4.2 块存储服务组件

块存储服务也包含很多组件，各组件之间的关系如图 3.6 所示。

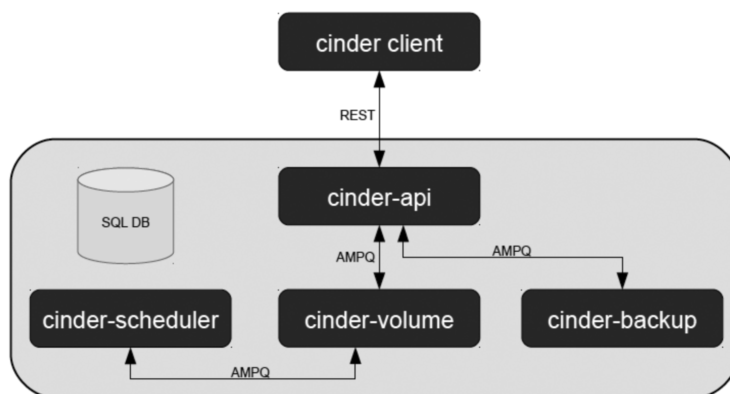



图 3.6　Cinder 组件

#### 1. cinder-api

cinder-api 用来接受 API 请求，并将其路由到 cinder-volume 执行。

#### 2. cinder-volume

cinder-volume 用来与块存储服务和例如 cinder-scheduler 的进程进行直接交互。它也可以与这些进程通过一个消息队列进行交互。cinder-volume 服务响应送到块存储服务的读写请求来维持状态。它也可以和多种存储提供者在驱动架构下进行交互。



云计算部署实战zw.indd 94　　2017-6-20 10:59:44

### 3．cinder-scheduler 守护进程

cinder-scheduler 守护进程会选择最优存储提供节点来创建卷。其与 nova-scheduler 组件类似。

### 4．cinder-backup 守护进程

cinder-backup 服务提供任何种类备份卷到一个备份存储提供者。就像 cinder-volume 服务，它与多种存储提供者在驱动架构下进行交互。

### 5．消息队列

消息队列作用是在块存储的进程之间路由信息。

## 3.4.3　Cinder 架构解释

Cinder 主要核心是对卷的管理，允许对卷、卷的类型、卷的快照进行处理。它并没有实现对块设备的管理和实际服务，而是为后端不同的存储结构提供了统一的接口，不同的块设备服务厂商在 Cinder 中实现其驱动支持以与 OpenStack 进行整合。在 CinderSupportMatrix 中可以看到众多存储厂商如 HP、NetAPP、IBM、SolidFire、EMC 和众多开源块存储系统对 Cinder 的支持。其核心架构由 API Service、Scheduler Service、Volume Service 三部分组成，它们之间的关系如图 3.7 所示。

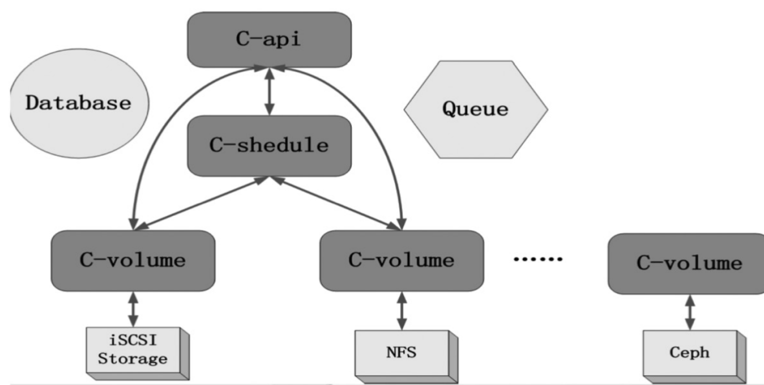



图 3.7　Cinder 核心架构

### 1．API Service

API Service 负责接受和处理请求，并将请求放入 RabbitMQ 队列。

### 2．Scheduler Service

Scheduler Service 处理任务队列的任务，并根据预定策略选择合适的 Volume Service 节点来执行任务。目前版本的 Cinder 仅仅提供了一个 Simple Scheduler，该调度器选择卷数量最少的一个活跃节点来创建卷。

3．Volume Service

Volume Service 该服务运行在存储节点上，管理存储空间。每个存储节点都有一个 Volume Service，若干个这样的存储节点联合起来可以构成一个存储资源池。为了支持不同类型和型号的存储，均通过 Drivers 的形式为 Cinder 的 Volume Service 提供相应的 Cinder-Volume。

### 3.4.4 Cinder 支持存储类型

Cinder 目前支持的存储类型有：

本地存储如 LVM，Sheepdog；

网络存储如 NFS，Ceph；

HP 的 3PAR(iSCSI/FC)，LeftHand(iSCSI) 存储设备；

IBM 的 Storwize family/SVC (iSCSI/FC)，XIV (iSCSI)，GPFS，zVM 设备；

Netapp 的 NetApp(iSCSI/NFS) 设备；

EMC 的 VMAX/VNX (iSCSI)，Isilon(iSCSI) 设备；

Solidfire 的 Solidfire cluster(iSCSI) 设备。

关于 Cinder 的配置请上课工场 APP 或官网 kgc.cn 观看视频。



## 本章总结

- 所有磁盘阵列都是基于 Block 块的模式，所以 DAS 和 SAN 都是典型的块存储类型，而所有的 NAS 产品都是文件级存储。
- Swift 是开源的对象存储（引擎），提供给 OpenStack 对象存储服务。
- Cinder 是块存储，提供给 OpenStack 永久的块存储服务。
- Swift 和 Cinder 都属于开源项目，由各自的核心组件组成。



云计算部署实战zw.indd 96 2017-6-20 10:59:45