Nutanix 圣经

作者： Steven Poitras

翻译： Nutanix China SE team

目　录

1 第一部分：历史回顾

通过一个快速基础架构历史的回顾，让我们知晓今天我们在什么阶段。

1.1 数据中心的革命

数据中心在过去的几十年间已经发生翻天覆地的变化，下面将详细说明每一个阶段。

1.1.1 大型机的时代

大型机(MainFrame)在相当长的一段时间内作为数据中心核心业务系统的基础，它让很多公司获得关键的优势：

- 系统本身就融合的 CPU、内存和存储；
- 设计为内部冗余机制;

但是大型机也带来了如下的问题：

- 高昂的基础架构采购费用；
- 内部的复杂性；
- 缺乏弹性，高度竖井化的环境。

1.1.2 向独立的服务器迁移

对于大部分的公司或组织而言，其业务很难充分利用主机的能力，在一定程度上导致了独立服务器的出现，其关键的特性包括：

- CPU、内存和直连式存储 (DAS，即 Direct Attached Storage)；
- 比大型机更灵活的系统环境；
- 可通过网络访问；

但这些独立服务器模式仍然存在问题：

- 增加了竖井化的环境；
- 资源利用率低或不均；
- 对于计算和存储而言，服务器成为单点故障（SPOF）；

1.1.3 中心化存储

业务一直需要能够盈利，而数据是关键因素。借助直连存储（DAS），当前的组织或公司要么需要更多的本地化存储空间，或者是数据的高可用等特性，即当服务器失效不会引起数据的不可用。

中心化的存储替代了大型机和独立的服务器模式，实现了可共享的、更大的存储资源池，并且可以提供数据的保护能力，关键的特点在于：

- 池化的存储资源可以更好的被利用；
- 通过 RAID 方式数据得到保护，从而避免了当服务器宕机引起的数据丢失；
- 存储通过网络更广泛的被使用；
- 共享存储依然会存在如下的问题：
- 隐性的硬件成本高昂，然而数据比硬件更有价值；
- 增加了系统的复杂度（SAN 架构，WWPNs，RAID 组，卷，存储控制器等）；
- 需要额外的管理工具和技术团队。

1.1.4 虚拟化的出现

我们可以观察到，数据中心计算资源利用率比较低，资源的效率也接近最低线。虚拟化的引入使多种工作负载和操作系统以虚拟机(VMs)的形式运行在单一物理服务器之上。虚拟化可以增加业务对于独立物理服务器的利用率，但也确实增加了竖井式系统的数量和故障等因素。其特征有：

- 操作系统和物理硬件解耦合（VM）
- 高效的计算使用率可以整合工作负载；

早期的虚拟化存在一定的问题：

- 竖井式业务系统变多，管理复杂度增加；
- 缺乏虚拟机的高可用，如果计算节点失效将造成很大影响；
- 池化资源的缺乏；
- 需要单独的管理工具和团队。

1.1.5 虚拟化的成熟

今天的 x86 虚拟化成为一个非常有效且功能丰富的方案，借助高级特性，包括在线迁移(vMotion/Live Migration)、高可用(High Availability)和动态资源负载均衡(Distributed Resource Scheduler)等，用户真正获得了虚拟机的高可用等能力，可以真正地动态迁移计算工作负载。但需要注意的是，中心化存储虽然可以合并路径，但是由于虚拟机快速增长和存储阵列的负载提升会导致存储 IO 瓶颈。

关键的特性有：

- 集群化计算资源池；
- 在计算节点间具备动态迁移的能力（DRS/vMotion）；
- 计算节点失效，VM 具备高可用性；
- 中心化存储的需求。

问题依然存在：

- 虚拟机的快速增长导致更高的存储需求；
- 更多存储阵列的使用导致更多竖井化管理和复杂度；
- 由于需要增加阵列，导致更高的单位容量成本；
- 可能会导致阵列资源的冲突；
- 存储配置更加复杂：
  - ✓ VM 与 Datastore 和 LUN 的配比；
  - ✓ 存储控制器的数量应对 I/O 需求。

1.1.6 固态硬盘

固态硬盘(Solid State Disks)可以无需借助更多的盘阵就就可以实现很高的 I/O 性能，从而解决 I/O 瓶颈问题。然而，在这种高性能下，存储的控制器和网络仍然不能处理海量 I/O 的吞吐能力。固态硬盘的关键特性有：

- 比传统 HDD 具备更好的性能；
- 消除查询的时间；

但固态硬盘也有问题：

- 来自于存储控制器和网络的瓶颈；
- 竖井依然存在；
- 阵列配置仍旧复杂。

1.2 延迟的重要性

下面给出特定 I/O 类型的不同延迟特性：

Item	Latency	Comments
L1 cache reference	0.5 ns	
Branch Mispredict	5 ns	
L2 cache reference	7 ns	14x L1 cache
Mutex lock/unlock	25 ns	
Main memory reference	100 ns	20x L2 cache, 200x L1 cache
Compress 1KB with Zippy	3,000 ns	
Sent 1KB over 1Gbps network	10,000 ns	0.01 ms
Read 4K randomly from SSD	150,000 ns	0.15 ms
Read 1MB sequentially from memory	250,000 ns	0.25 ms
Round trip within datacenter	500,000 ns	0.5 ms
Read 1MB sequentially from SSD	1,000,000 ns	1 ms, 4x memory
Disk seek	10,000,000 ns	10 ms, 20x datacenter round trip
Read 1MB sequentially from disk	20,000,000 ns	20 ms, 80x memory, 20x SSD
Send packet CA -> Netherlands -> CA	150,000,000 ns	150 ms

(来源: Jeff Dean, https://gist.github.com/jboner/2841832)

上面的表格给出 CPU 访问它的缓存需要 0.5 纳秒到 7 纳秒不等（L1 vs L2），对于内存，访问的时间则为 100 纳秒，而一个固态硬盘的 4K 读则需要 150,000 纳秒即 0.15 毫秒。如果我们研究一个企业级固态硬盘（例如 Intel S3700 系列），这个设备能力如下：

- 随机 I/O 性能：
  - ✓ 随机 4K 读可达 75000 IOPS
  - ✓ 随机 4K 写可达 36000 IOPS
- 序列化带宽：
  - ✓ 持续读可达 500 MB/s
  - ✓ 持续写可达 460 MB/s
- 延迟：
  - ✓ 读延迟 50 微秒
  - ✓ 写延迟 65 微秒

1.2.1 带宽的考量

对于传统存储，有几种结构类型：

- 光纤通道 (Fiber Channel)
  - ✓ 4/8/10Gb
- 以太网(包括 FCoE)
  - ✓ 1/10Gb (40Gb Infiniband)等

我们使用 Intel S3700 系列固态硬盘的 500 MB/s 读和 460 MB/s 写带宽作为计算基础，参考如下公式：

numSSD = ROUNDUP((numConnections * connBW (in GB/s))/ ssdBW (R or W))

注释：固态硬盘数量按照整块计算，也不考虑 CPU 去处理所有 I/O 的损耗，假设 CPU 的处理能力是足够的。

Network BW		SSDs required to saturate network BW	
Controller Connectivity	Available Network BW	Read I/O	Write I/O
Dual 4Gb FC	8Gb == 1GB	2	3
Dual 8Gb FC	16Gb == 2GB	4	5
Dual 16Gb FC	32Gb == 4GB	8	9
Dual 1Gb ETH	2Gb == 0.25GB	1	1
Dual 10Gb ETH	20Gb == 2.5GB	5	6

上面的表格显示，如果想达到一块固态硬盘提供的 I/O 能力理论最大值，网络可能会成为瓶颈，不同的网络带宽对应到 1 块到 9 块固态硬盘的处理能力。

1.2.2 内存延迟的影响

典型内存的延迟在 100 纳秒左右(不同型号及品牌的内存延迟会不同)，我们可以依据下面公式计算：

- 本地内存读延迟= 100ns + [OS / hypervisor overhead]
- 网络内存读延迟= 100ns + NW RTT latency + [2 x OS / hypervisor overhead]

如果我们假设一个典型的网络响应时间为 0.5 毫秒(不同交换机延迟会不同)，转换成 500，00 纳秒计算公式有如下：

- 网络内存度延迟= 100ns + 500,000ns + [2 x OS / hypervisor overhead]

如果我们假设一个非常快速的网络，RTT 只有 10,000 纳秒：

- 网络内存度延迟= 100ns + 10,000ns + [2 x OS / hypervisor overhead]

这就意味着一个理论上非常快速的网络与一个本地内存访问的模式比，其延迟开销超过 10,000%，而对于一个比较慢的网络而言，这个延迟开销相比可能超过 500,000%。为了缓解延迟的开销，服务器端的缓存技术出现了。

1.3 Web-Scale

名词解释：Web Scale，一种全新的计算架构方式，面向基础架构和计算资源等，可以实现互联网规模的弹性扩展能力。

这部分将介绍一些 Web Scale 基础架构的核心概念，以及为什么我们要采用它。在我们开始介绍以前，必须清楚的了解 Web Scale 不意味着你真的要扩展到像 Google、Facebook 或 Microsoft 等这种互联网规模。这种架构适用于任何扩展情况(从 3 节点到上千节点)，并且会受益匪浅。

我们还是来看一下面对的挑战：

- 复杂，复杂，还是复杂；
- 业务快速增长的需要；
- 对敏捷性的需求等。

有一些关于 Web Scale 的关键架构和概念：

- 超融合
- 软件定义的智能化
- 分布式自治系统
- 线性增加地扩展

其他相关概念：

- 基于 API 的自动化和丰富的分析
- 自恢复能力

下面通过技术视角来解释这些概念的意义。

1.3.1 超融合

针对于超融合的概念有着不同的理解，因为组件不同（虚拟化、网络等）而理解不同。然而，核心的概念是这样阐述：天然地(Natively)将两个或多个组件组合到一个独立的单元中。在这里，天然(Natively)是一个关键词。为了更加有效率，组件一定是天然地整合在一起，而不是简单地捆绑在一起。对于 Nutanix，我们天然地将计算和存储融合到我们设备的单一节点中 。这就真正意味着天然地将两个或多个组件整合在一个独立的、可容易扩展的单元中。

优势在于：

- 独立单元的扩展
- 本地 I/O 处理
- 消除传统计算/存储的竖井式结构，融合它们在一起

1.3.2 软件定义的智能化

软件定义的智能化是在通用的、商品化的硬件之上通过运行软件来实现核心的逻辑，而这些逻辑之前用专有的硬件编程方式实现(例如 ASIC/FPGA 等)。对于 Nutanix 而言，我们将传统的存储逻辑(例如 RAID，去重，压缩等)采用软件方式去实现，这些软件运行在标准的 x86 硬件上的 Nutanix 控制虚拟机(Controller Virtual Machine，即 CVM)内。那就真正意味着把关键处理逻辑从专有硬件中剥离放入到运行在商用的硬件设备的软件之中。

好处在于：

- 快速地版本迭代周期
- 消除了专有硬件的依赖
- 更好利用通用的、标准的硬件

1.3.3 分布式自治系统

分布式自治系统涉及从传统的单一集中模式处理业务转向跨集群内的所有节点分布式处理业务，可以考虑创建一个完全分布式的系统。传统角度考虑问题是假设硬件是可靠的，在某种程度上是对的。然而分布式系统的核心思想是硬件终究会出问题，在一个简单的、业务不间断的方式中处理故障是关键点。这些分布式系统的

设计是为了调整和修复故障，达到自恢复和自治的目的。在组件发生故障时，系统将透明地处理和修复故障，并持续按照预期运行。将会提醒用户知晓故障的存在，但不会作为一个紧急事件被提出来，任何一种修复(如：替代一个失效的节点)都可以按照管理员事先设定好的计划表去自动化的处理。另外一种方式是重建而不需要替换，一个主数据节点被随机选举出来，当其故障后新的主数据节点会被选举出来，利用 MapReduce 的概念来分配任务的处理。这就真正说明：

- 分配的角色和任务到系统内的所有节点
- 利用 MapReduce 等机制执行分布式任务处理
- 当需要一个新的主数据节点时，采用选举机制

优势有：

- 解决了单点故障 (SPOF)
- 分布式业务负载，消除任何瓶颈

1.3.4 可累积线性扩展

可累积线性扩展是指从一组资源开始，当需要横向扩展的时候可以随扩展线性增加系统的性能。上面提到的所有结构都是重要的推动因素，使得线性扩展成为现实。例如，一个运行虚拟化负载的传统三层架构：服务器、存储和网络，每层的扩展都是独立的。当横向扩展服务器数量和性能的时候，却不能实现存储性能的横向扩展。但是像 Nutanix 的超融合平台，却可以同时随着节点数的增加而同时扩展：

- 计算节点数量；
- 存储控制器的数量；
- 计算和存储的性能及容量；
- 参与集群运行的节点数量。

这就真正意味着：

- 存储和计算节点的线性增加可以实现性能和容量的线性增长；

优势在于：

- 从小规模开始很想扩展；
- 在任何规模级别的均匀和一致性的性能增长。

1.3.5 总结

针对于前面篇幅，总结如下：

- 计算资源使用率低促使虚拟化产生
- 高级特性，包括 vMotion、HA 和 DRS 等需要集中化存储
- VM 的快速增长不仅增加存储负载，而且增加存储资源争抢，带来瓶颈
- 固态硬盘改善了问题，但依然不能解决网络和控制器带来的瓶颈
- 跨网络的缓存或内存访问将面临巨大的开销，优势不再明显
- 阵列配置复杂性依然存在
- 服务器端的缓存被用来降低阵列的负载和网络影响，但是引入了新的组件
- 为了克服传统的网络问题，本地化帮助降低瓶颈和负载
- 将焦点从复杂的基础架构转到管理的易用性和系统堆栈的简化
- Web Scale 的世界诞生了

2 第二部分：Prism

名词解释：Prism，控制面板，为数据中心运营提供一键式管理和接口。

2.1 设计方法

建设一个美观的、易于使用且直观的产品是 Nutanix 平台设计的核心思想，这一个章节将介绍我们的设计方法和如何重复使用它们。在空暇之余可以先了解我们产品设计负责人 Jeremy Sallee 的设计思路和迭代过程：

http://salleedesign.com/stuff/sdwip/blog/nutanix-case-study/

您也可以去下载 Nutanix Visio 的 模板：

http://www.visiocafe.com/nutanix.htm

2.2 架构

Prism 是一个分布式的资源管理平台，允许用户跨 Nutanix 集群环境管理和监控对象及服务。这些能力被分为两种类别：

- 接口
  - ✓ HTML5 UI, REST API, CLI, PowerShell CMDlets 等；
- 管理
  - ✓ 策略定义与合规，服务设计与状态，分析和监控。
  - 下图重点解释了作为 Nutanix 平台一部分的 Prism 概念特征：

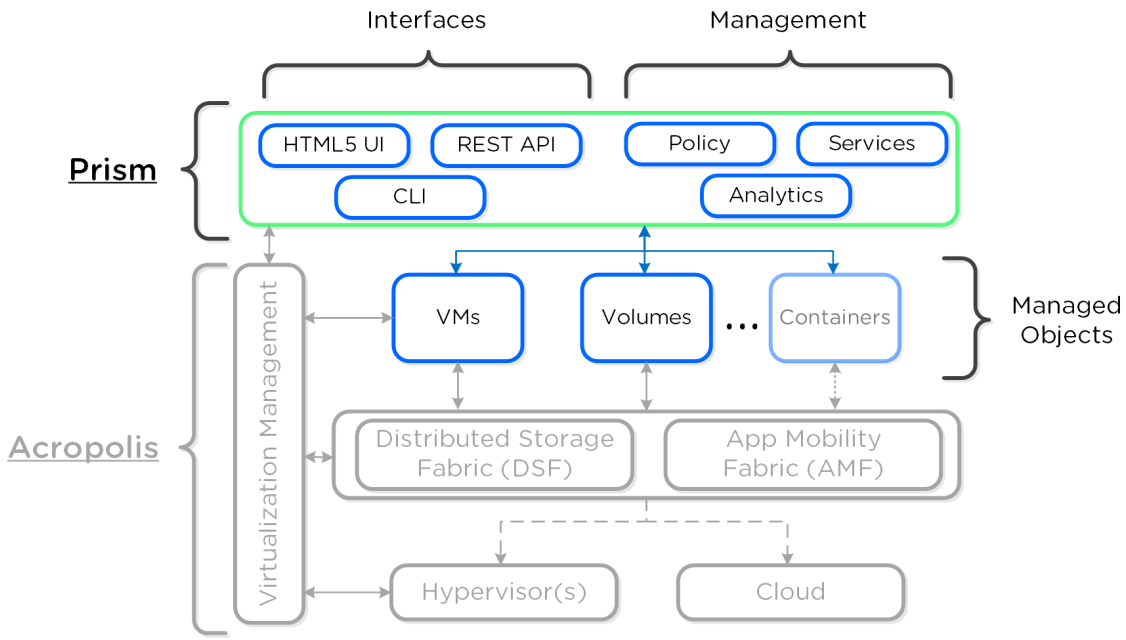


图 5-1 Prism 的概要架构

Prism 主要由两个组件构成：

- Prism Central (PC)
  - ✓ 多个集群的统一管理者，通过一个单一的、中心化的管理接口负责管理多个 Acropolis 集群。Prism Central 是一个软件设备（虚拟机），可以选择部署在 Acropolis 集群之上或集群外部；
  - ✓ 一对多的方式；
- Prism Element (PE)
  - ✓ 本地集群管理者，负责本地集群的管理和运行，每个 Acropolis 集群都有自己内置的 Prism Element；
  - ✓ 一对一的方式。

下图指出在 Prism Central 和 Prism Element 之间的逻辑关系：

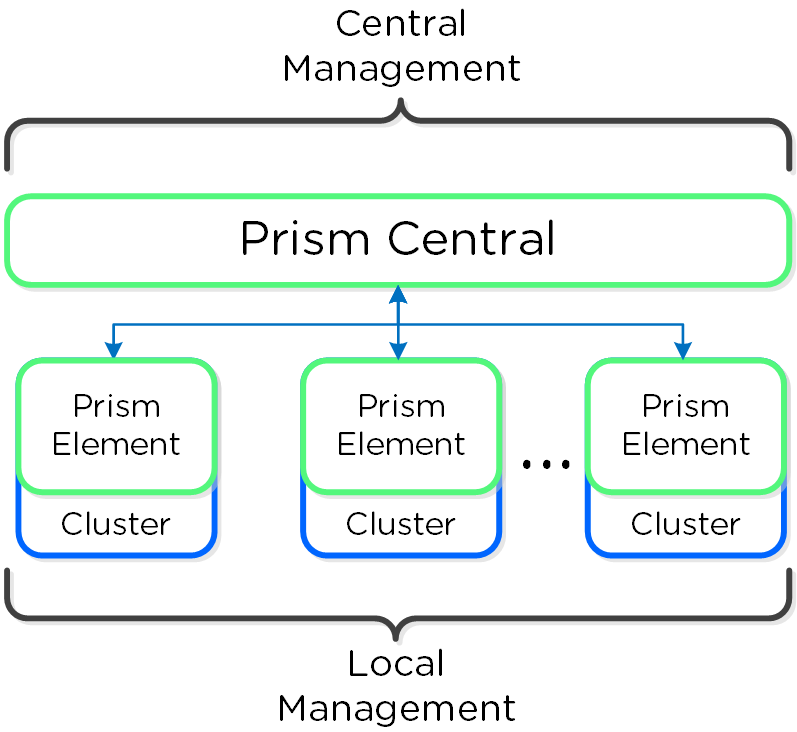


图 5-2：Prism Central 和 Prism Element 关系架构

小提示：对于规模大或者分布式部署（例如超过一个集群或多个站点的情况），推荐采用 Prism Central 简化管理操作，使用一个管理界面面向所有集群和站点。

2.2.1 Prism 的服务

Prism 服务运行在每一个 CVM 之上，其中一个 CVM 上的 Prism 服务会被选为 Prism Leader 组件，负责处理集群的 HTTP 请求。与其他的组件有 Master 类似，如果一个 Prism Leader 出问题，一个新的 Leader 会被选出。当不是 Prism Leader 的 CVM 得到 HTTP 的请求，它将永远重定向请求到当前的 Prism Leader，并且返回 HTTP 应答码 301。下图是解释 Prism 服务是如何处理 HTTP 请求：

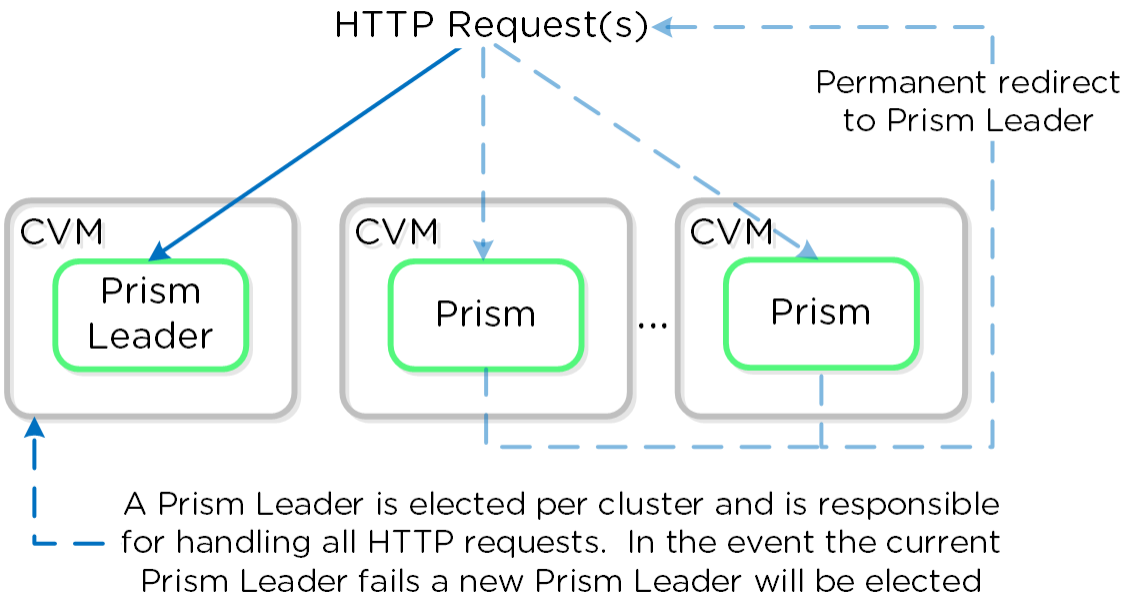


图 5-3：Prism 服务处理 HTTP 请求

Prism 端口：Prism 监听 80 和 9440 端口，如果 HTTP 流量来自于 80 端口，它将会被重定向到 9440 端口。

如果我们使用集群外部 IP（推荐），它将一直被当前的 Prism Leader 所有。即使 Prism Leader 出问题，集群 IP 也会转移到新选举出的 Prism Leader 上。一个免费 ARP（gratuitous ARP，即 gARP）将被用来清理旧的 ARP 缓存条目。在上面的场景中，任意时间集群 IP 都可以被用来访问 Prism，而无需重定向需要，因为 Prism Leader 已经被选出。

小提示：可以在 CVM 里面使用命令改变当前的 Prism Leader，请运行：

“curl localhost:2019/prism/leader”

2.3 管理导航

Prism 提供相当直观和简单的使用方式，然而我们在此只覆盖一些主要的页面和使用方式。Prism Central(如果部署)可以使用指定的 IP 地址或相应的 DNS 条目访问，而 Prism Element 可以通过 Prism Central 导航到任意 Nutanix CVM 或集群 IP。一旦页面被加载将登陆到 Prism 欢迎界面，可以使用 Prism 账户或 AD 认证条目登陆。

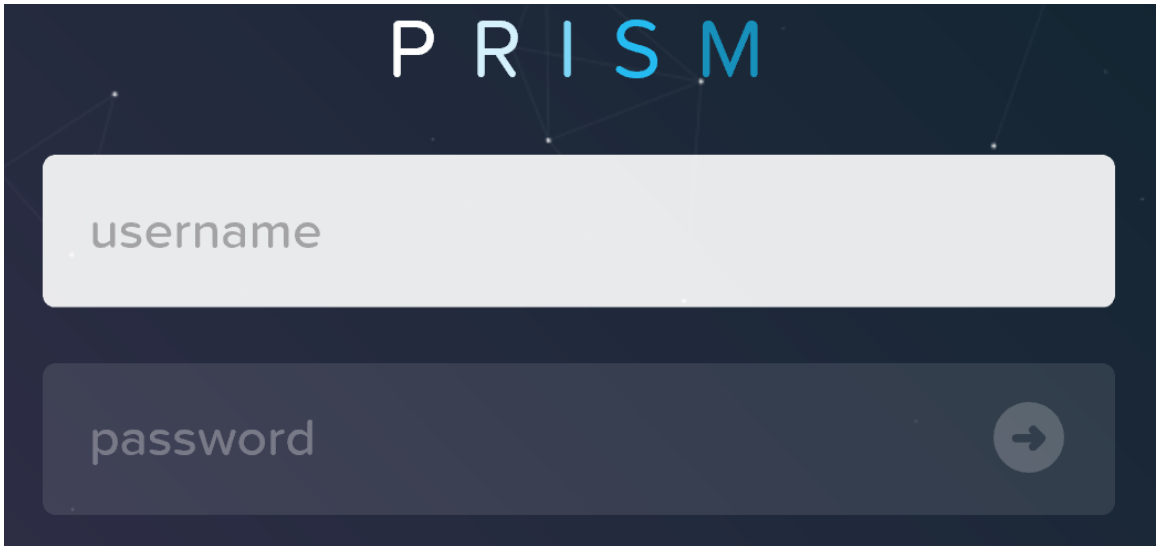


图 6-1：Prism 登陆页面

成功登陆之后将看到 Prism Central 或本地 Prism Element 集群内的，提供整个集群管理信息的面板页面

2.3.1 Prism Central

Prism Central 包括如下主要页面：

- 主页面
  - ✓ 整体环境的监控仪表盘，包括服务状态、容量计划、性能、任务等详细信息，为了获得更多信息，可以点击每一个条目；ß
- 导航页面
  - ✓ 运维和监控服务、集群、虚拟机和主机等；
- 分析页面
  - ✓ 具有事件关联性的集群及管理对象的详细性能分析；
- 告警
  - ✓ 整体环境的告警信息。

下图显示出 Prism Central 的仪表盘，可以监控和管理多个集群：

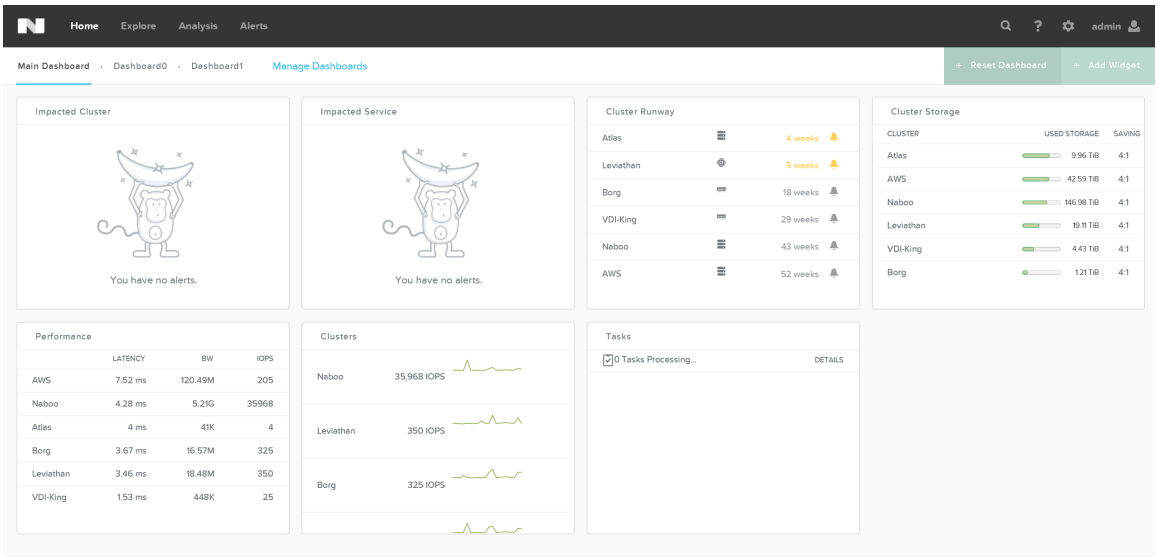


图 6-2：Prism Central 仪表盘

在仪表盘上可以监控到所有的环境状态，如果有任何告警或选项，可以深入查看。

小提示：如果监控项是绿色，代表所有环境是正常的。

2.3.2 Prism Element

Prism Element 包括如下主要页面：

- 主页面
  - ✓ 本地集群监控仪表盘，包括了告警、容量、性能、健康度、任务等详细信息，可以点击具体条目获得更多的信息；
- 健康
  - ✓ 环境、硬件及被管理对象的健康及状态信息，也包括 NCC 健康检查信息；
- 虚拟机
  - ✓ 在 Acropolis Hypervisor 上实现全面的虚拟机管理、监控及创建、运行、更新和删除等功能；
  - ✓ 虚拟机监控（非 Acroplis Hypervisor）
- 存储
  - ✓ 容器管理、监控及创建、运行、更新和删除等功能；
- 硬件
  - ✓ 服务器、磁盘和网络管理、监控和健康检查，也涵盖了集群的扩展；

- 数据保护
  - ✓ 容灾、公有云对接和同城 Metro Availability 的配置，管理保护域的对象、快照、复制和恢复；
- 分析
  - ✓ 具有事件关联性的集群及管理对象的详细性能分析；
- 告警
  - ✓ 本地集群和环境的告警。

主页面将提供详细的告警、服务状态、容量、性能及任务等信息，可以点击任意条目获取更多的信息。下图展现了一个 Prism Element 管理下的本地集群仪表盘：

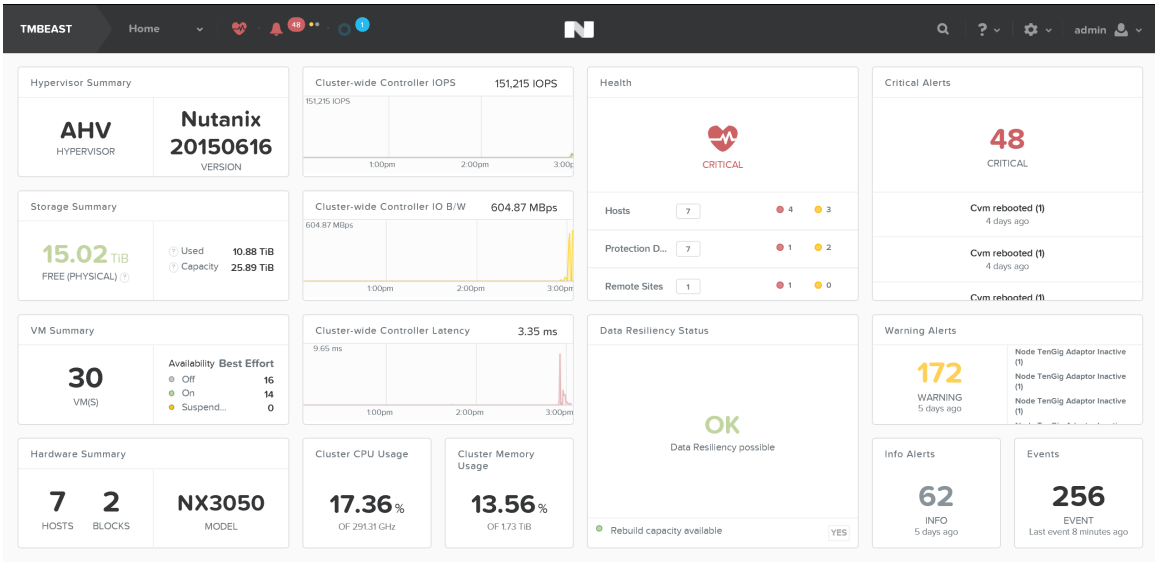


图 6-3：Prism Element 仪表盘

2.3.3 快捷键

快速接入在 Prism 里面至关重要，为了简化用户的使用允许用户使用键盘的快捷键来访问不同的视图和内容：

切换视图：

- O – Overview View
- D – Diagram View

- T – Table View

活动和事件

- A – Alerts
- P – Tasks

下拉导航和菜单

- M – 菜单选项
- S – 设置选项
- F – 搜索条目
- U – 用户条目
- H – 帮助

2.4 使用和问题解决

在下面内容中我们将介绍典型的 Prism 使用和问题查找等内容。

2.4.1 Nutanix 软件升级

执行 Nutanix 软件升级是一个非常简单且无需中断业务的过程，在登录到 Prism 以后点击界面右上角的齿轮图标，或者快捷键 S 选择更新软件：

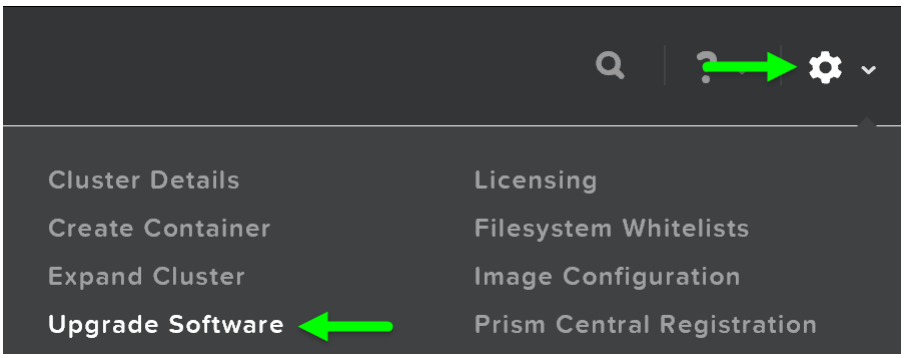


图 7-1：设置更新软件

启动“更新软件”对话框，然后显示出当前的软件版本，是否有新的版本更新，也可以手动上载 NOS 文件更新软件版本。如下图所示，可以从外部云环境下载更新版本，也可以手动上传新版本软件：

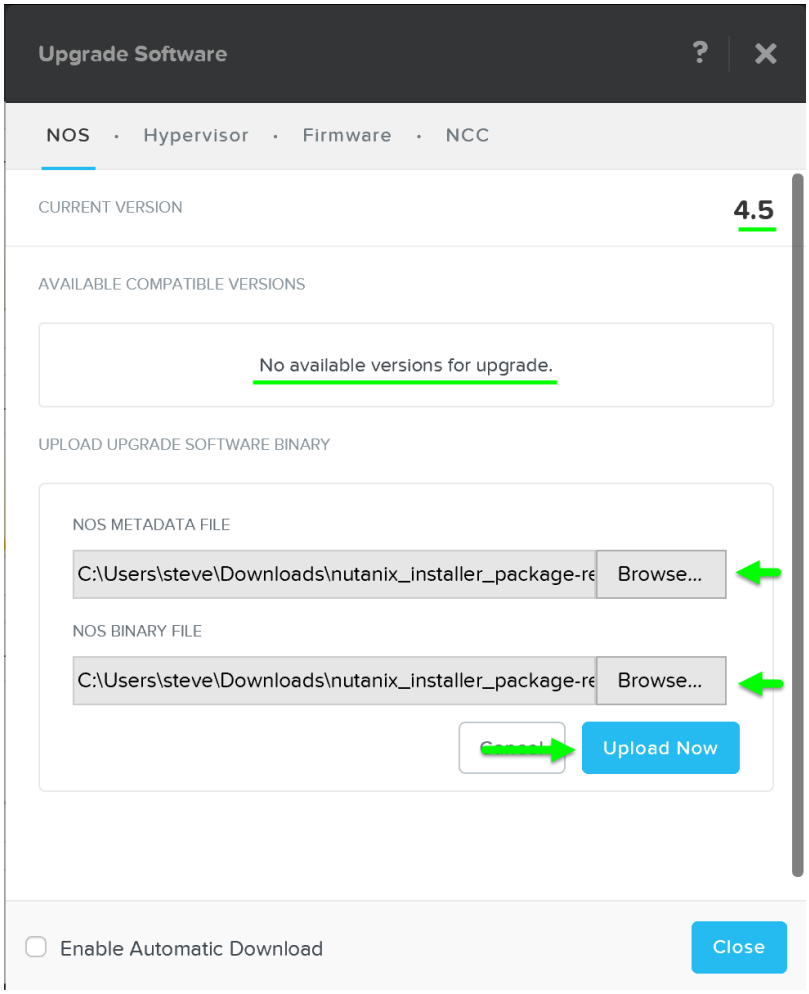


图 7-2：更新软件主界面

如果点击上传，则将更新软件上传到 Nutanix CVM 中：

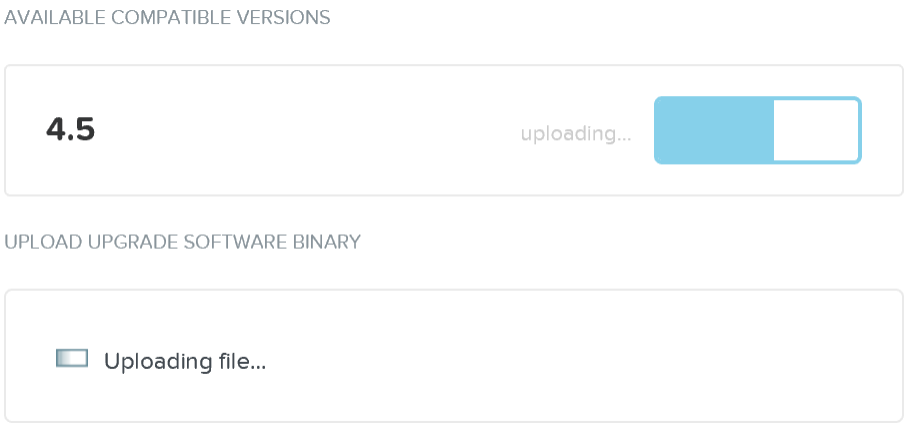


图 7-3：更新软件上传

在软件上传完成后点击”Upgrade”按钮，启动更新流程：

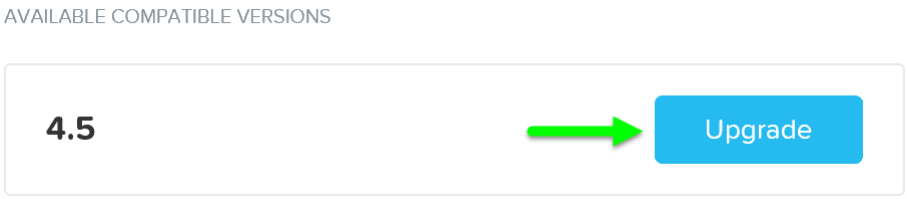


图 7-4：启动更新流程

再次确认是否要更新软件：

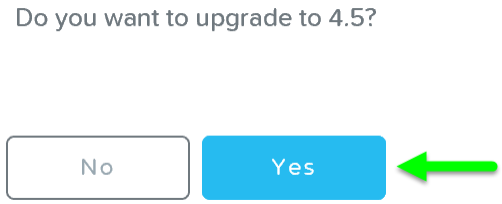


图 7-5：更新确认

更新过程启动后进行更新检查，然后自动更新软件：

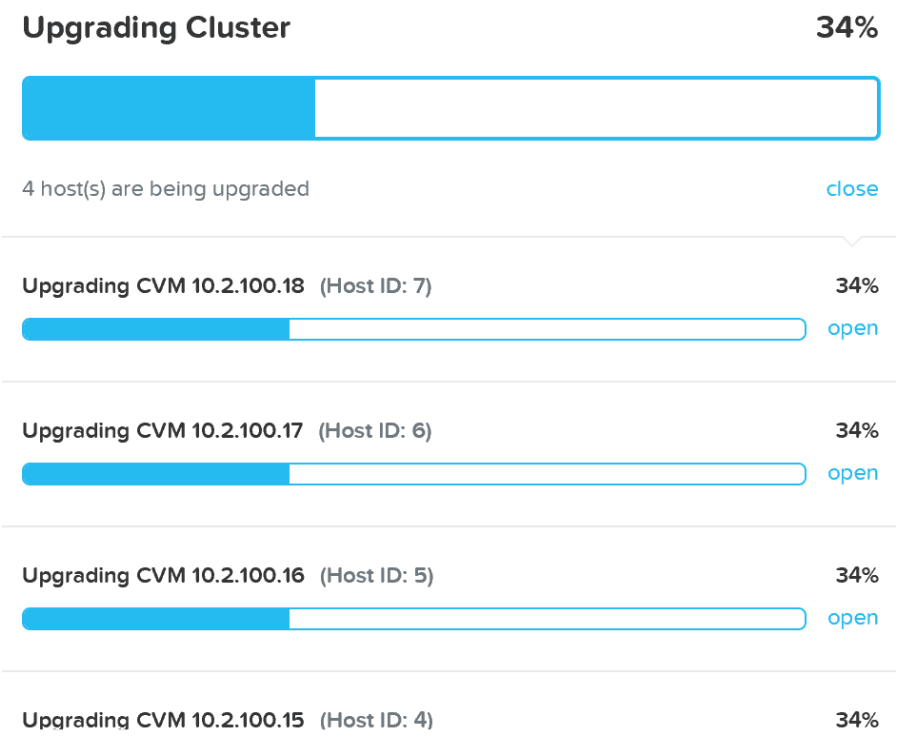


图 7-6：更新过程执行

一旦更新过程完成，将看到更新状态，随后即刻可以访问新的功能：

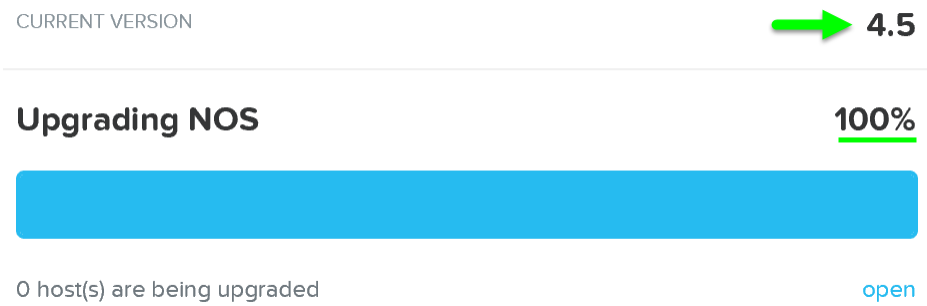


图 7-7：更新完成

小提示：当 Prism Leader 升级时，Prism 会话会出现暂时中断，但所有的虚拟机和服务运行不受影响。

2.4.2 Hypervisor 升级

类似 Nutanix 软件升级，Hypervisor 的升级也能通过 Prism 采用完全自动化方式，其升级步骤与之前类似，运行弹出“Upgrade Software”对话框，然后选择”Hypervisor”。Hypervisor 升级可以选择从云端自动下载软件，也可以手动上传新的软件版本。

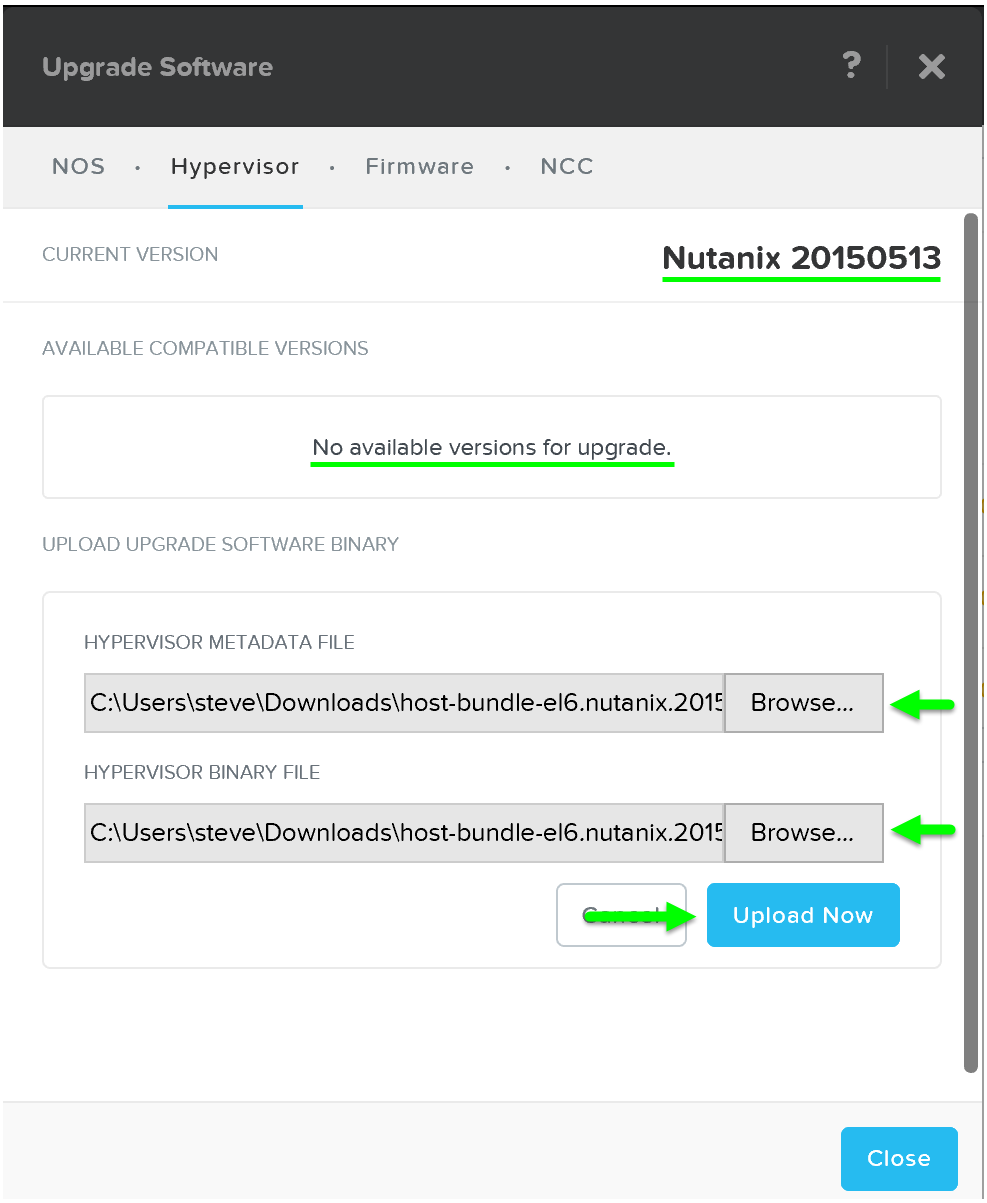


图 7-8：升级 Hypervisor — 主菜单

它将加载升级软件到虚拟化层，一旦新版本软件被成功加载，点击“Upgrade”按钮开始升级过程。

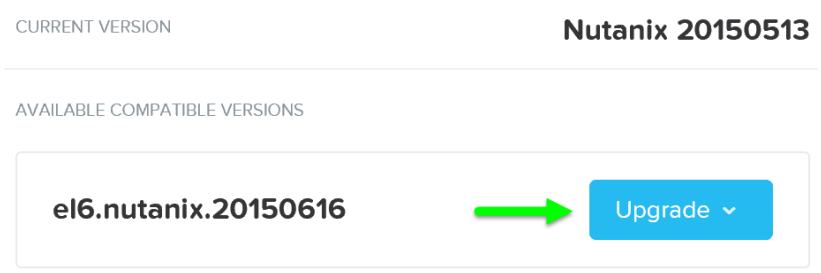


图 7-9：升级 Hypervisor — 开始升级过程

选择再次确认 Hypervisor 升级，点击“Upgrade”。

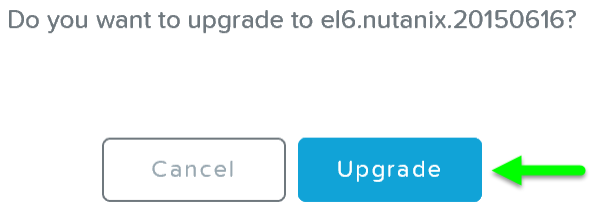


图 7-10：升级 Hypervisor — 确认升级

系统将自动进行升级前的检查，然后上传 Hypervisor 进行整个集群的升级工作。

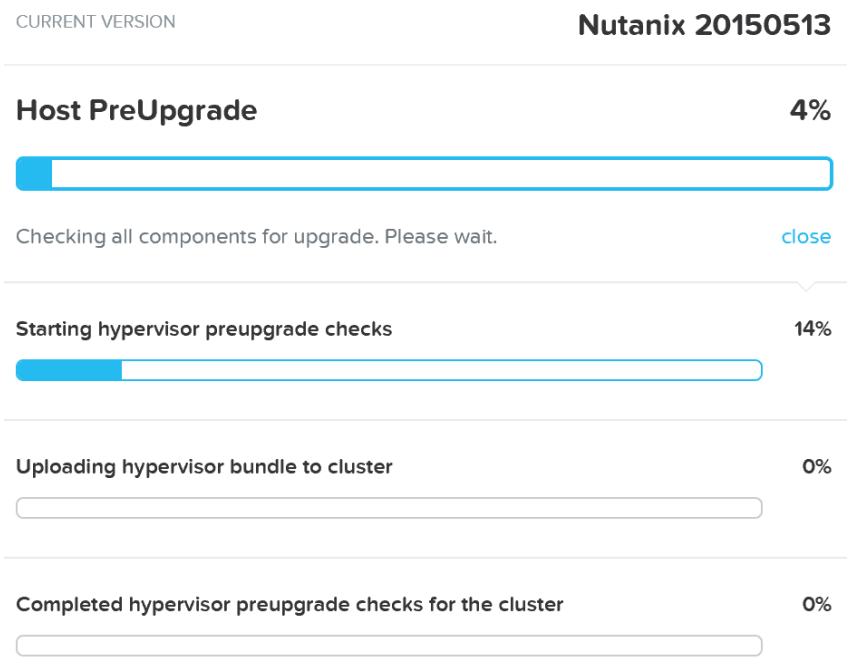


图 7-11：升级 Hypervisor — 预检查过程

一旦升级前检查完成，虚拟化层升级将以进度条形式开始进行：

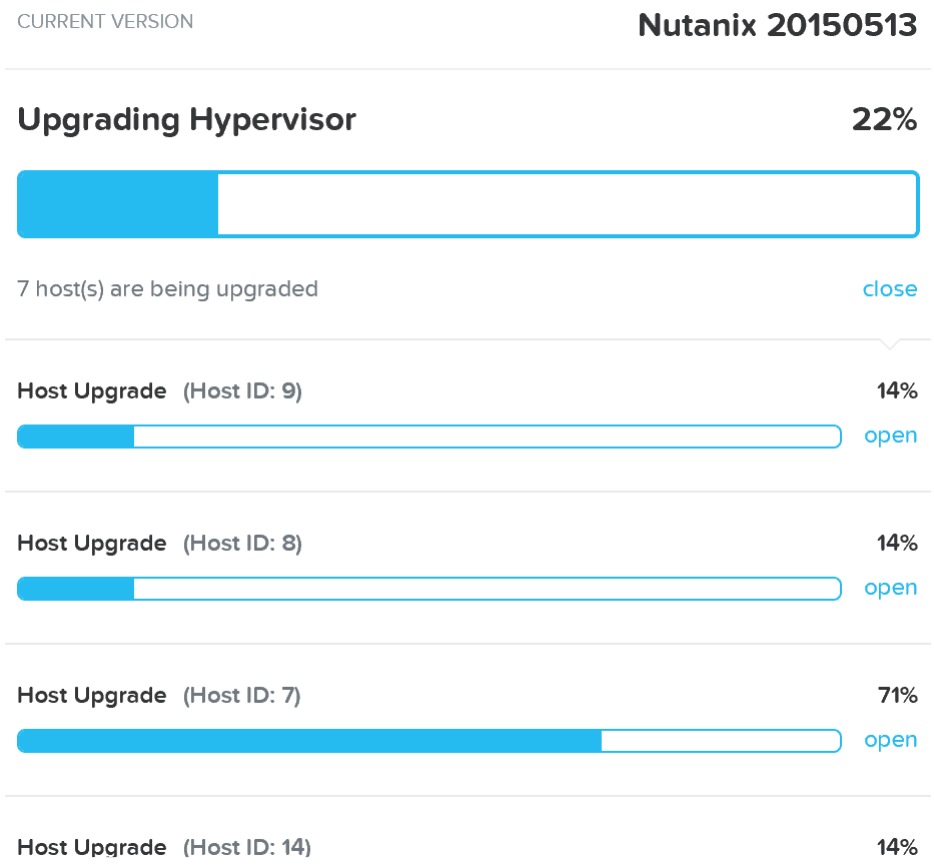


图 7-12：升级 Hypervisor — 升级执行

类似于 Nutanix 软件升级，集群的每个主机都会依次序在运行虚拟机不受影响的情况下完成升级工作。当主机 Hypervisor 升级的时候，上面虚拟机会被自动在线迁移到其他节点上；在 Hypervisor 升级过程中，主机会被自动重启。这个过程会依次序进行，直到集群内所有节点都被升级完成。

小提示：可以通过 Nutanix CVM 运行命令“host_upgrade --status”查看集群升级的状态，也可以通过查看 CVM 的日志“/data/logs/host_upgrade.out ”获取集群状态信息。

一旦升级完成，可以看见更新后的状态，立刻可以访问新的功能。

图 7-13：升级 Hypervisor — 升级完成

2.4.3 集群延伸(增加节点)

Acropolis 集群动态扩展能力是核心的功能。为了扩展一个 Acroplis 集群，将节点上架、摆放和插线，然后启动节点。一旦节点被启动后，采用 mNDS 方式自动可以被当前集群发现。下图展现了一个 7 节点的集群，同时动态发现 1 个新节点的出现。

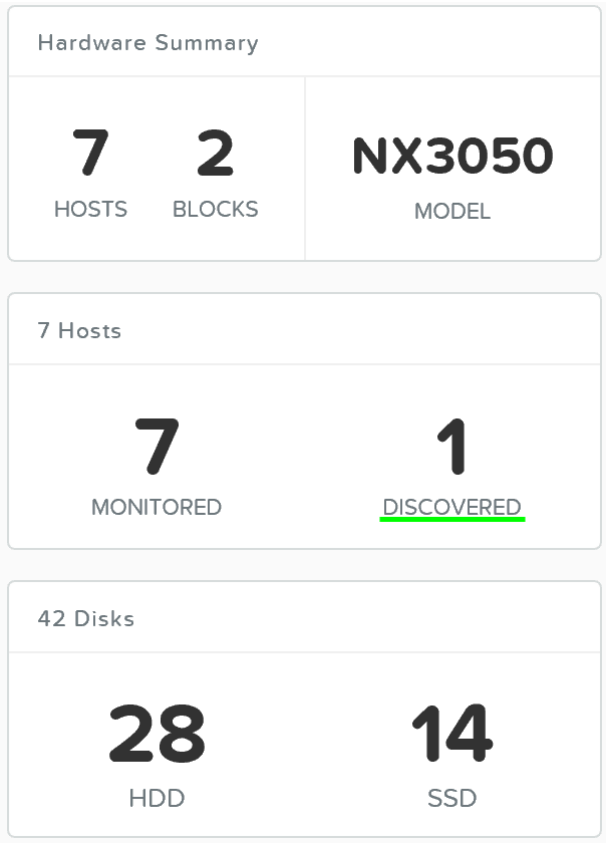


图 7-14：动态发现节点

多个节点可以被发现并被同时加入到集群中。一旦节点被发现，可以点击 Hardware 主页面右上角的“Expand Cluster”开始延伸这个集群。

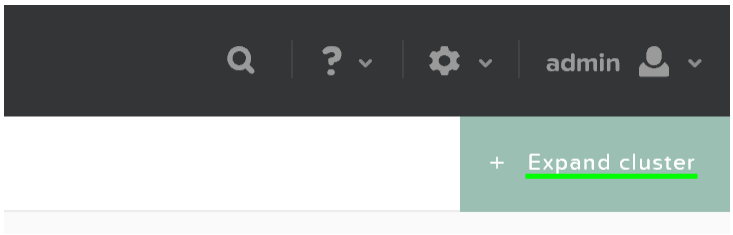


图 7-15：Hardware 页面中的 Expand Cluster

也可以在任意页面通过点击页面顶端右上角的齿轮图标，并找到 Expand Cluster。

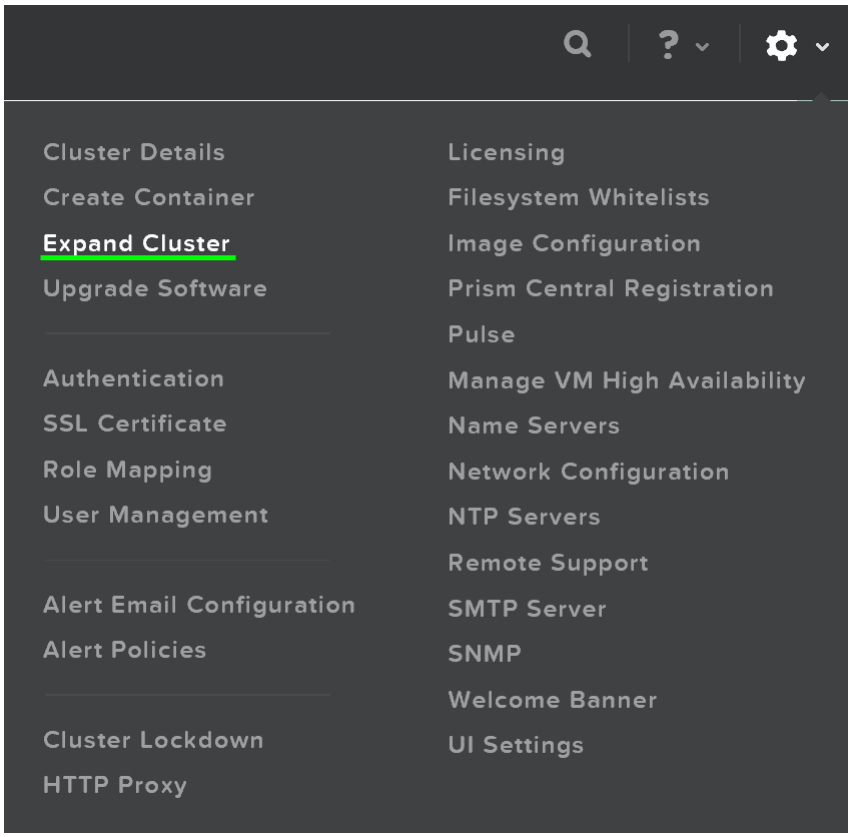


图 7-16：齿轮图标下的 Expand Cluster

点开 Expand Cluster 菜单，可以选择要增到集群内的节点和指定相应的 IP 地址。

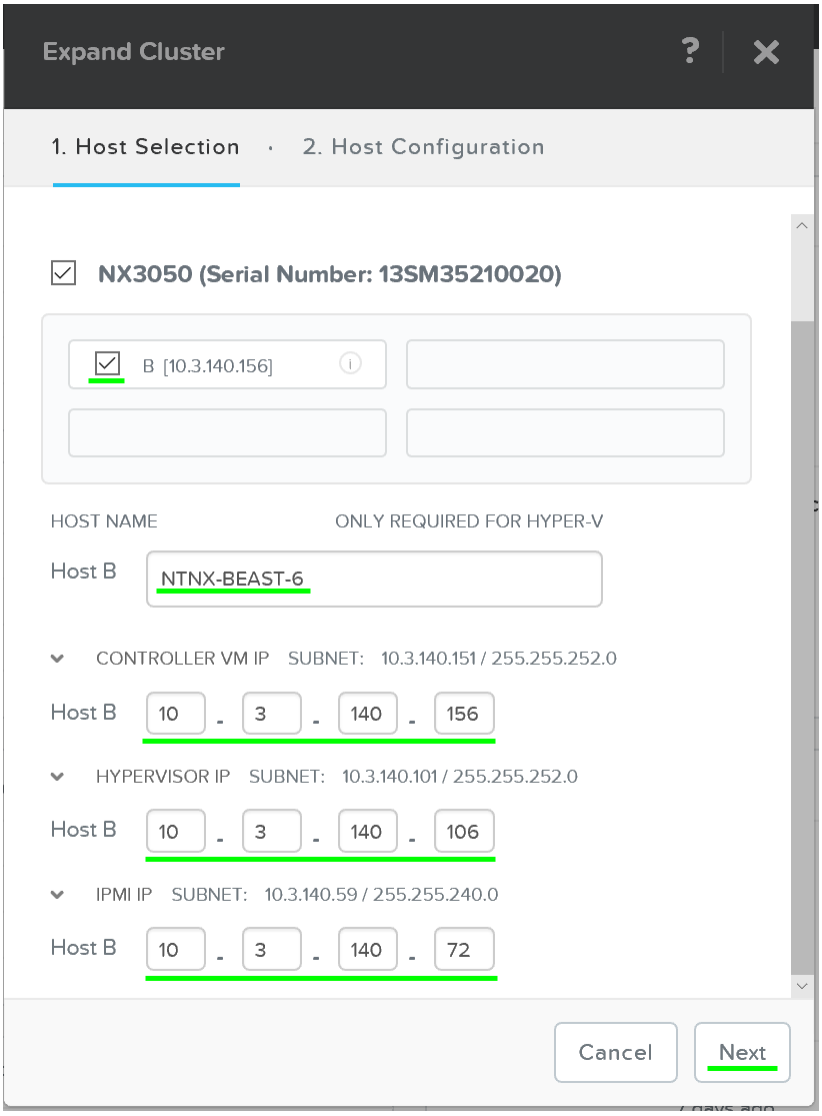


图 7-17：Expand Cluster—主机选择

在主机被选中后，将会被提示选择升级 Hypervisor 镜像。

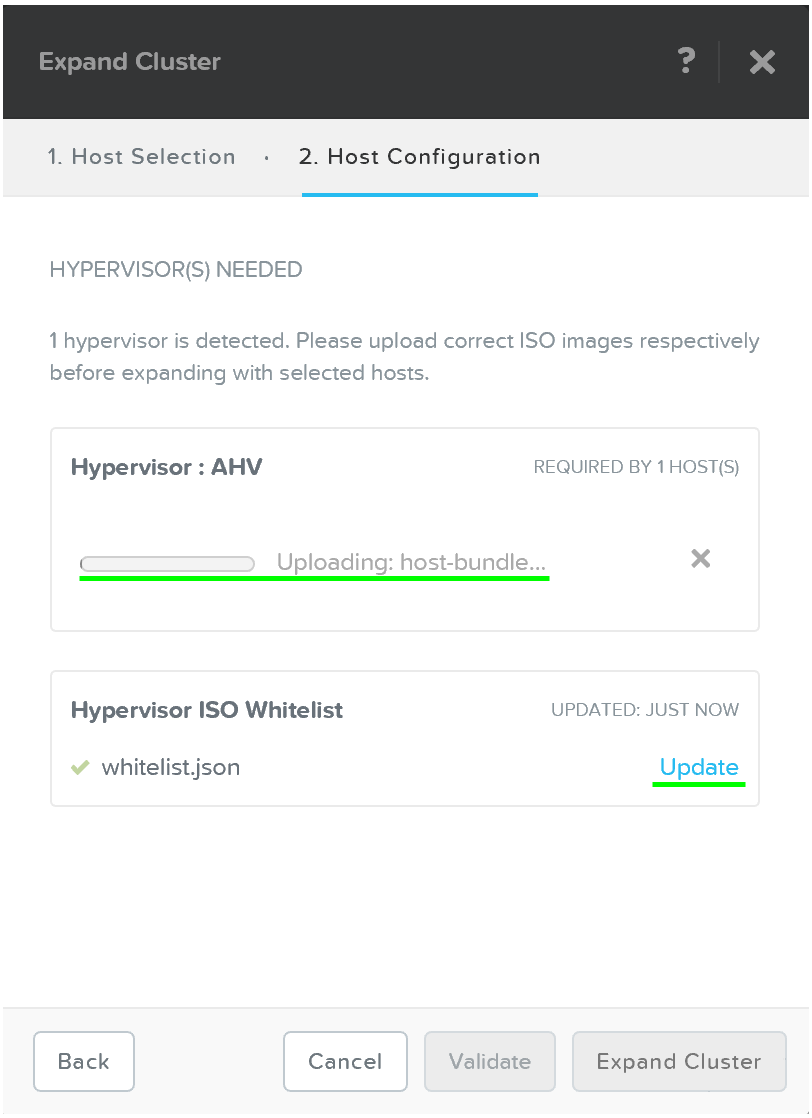


图 7-18：Expand Cluster—主机配置

在上载完成后，可以点击 Expand Cluster 启动镜像安装和集群延伸任务。

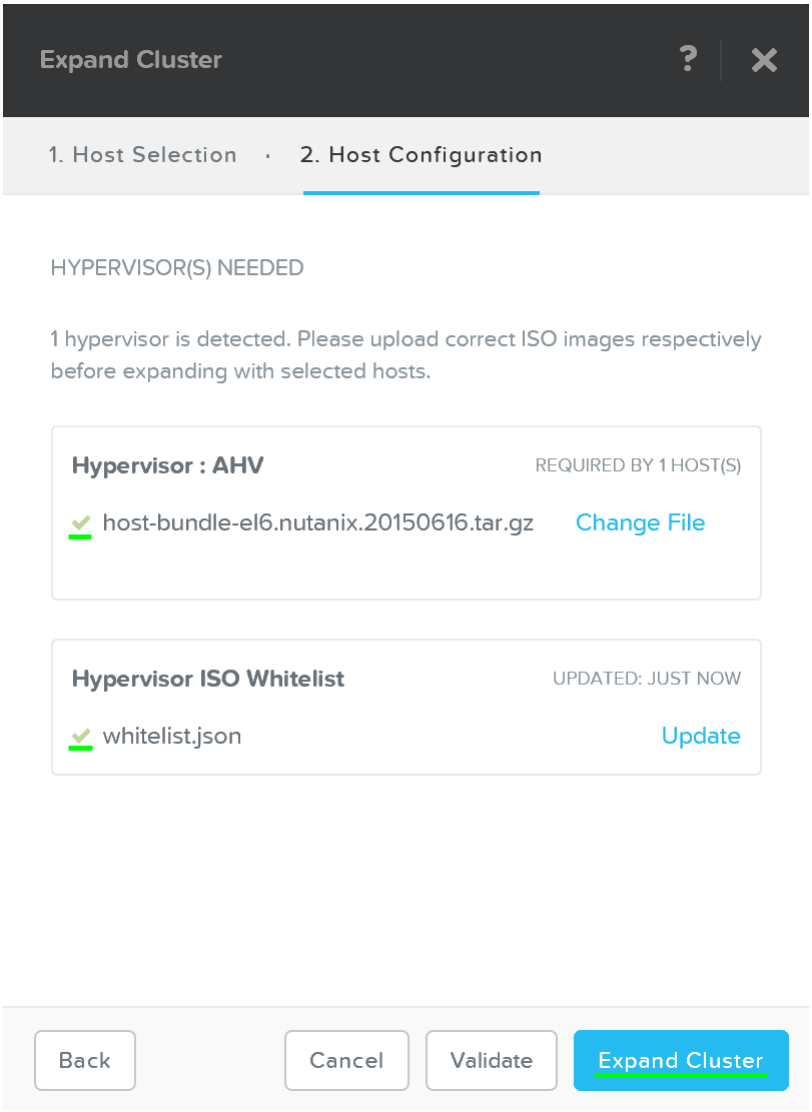


图 7-19：Expand Cluster—执行

执行任务将被提交，在任务栏提示任务执行的应答信息。

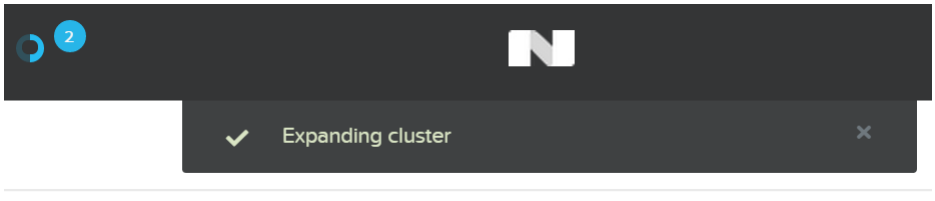


图 7-20：Expand Cluster—执行

任务执行状态可以在状态栏里查看（左上角蓝色圆圈）。

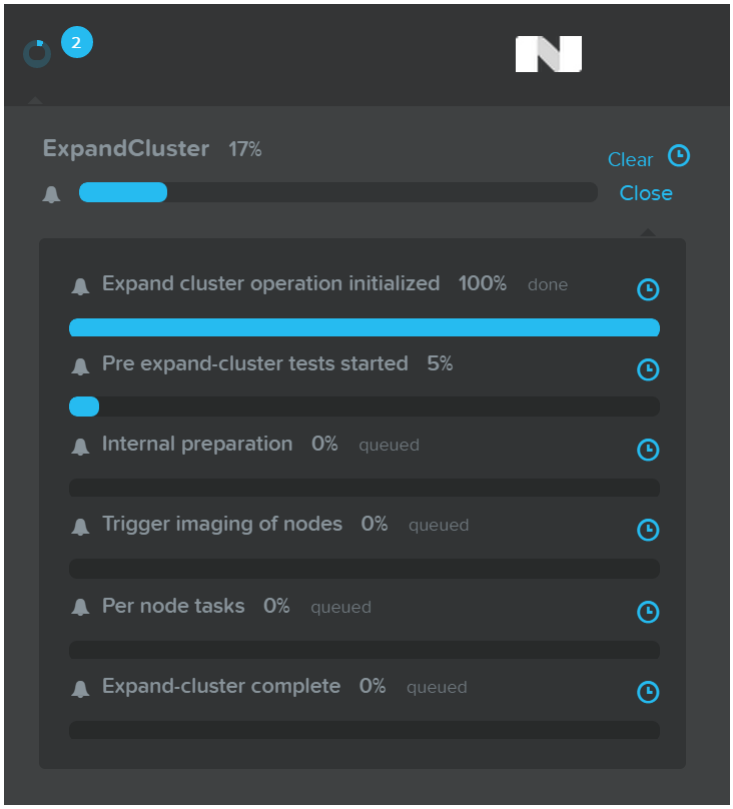


图 7-21：Expand Cluster—执行

在镜像安装和延伸节点的过程完成后，可以看到更新后的集群内节点数量和资源情况。

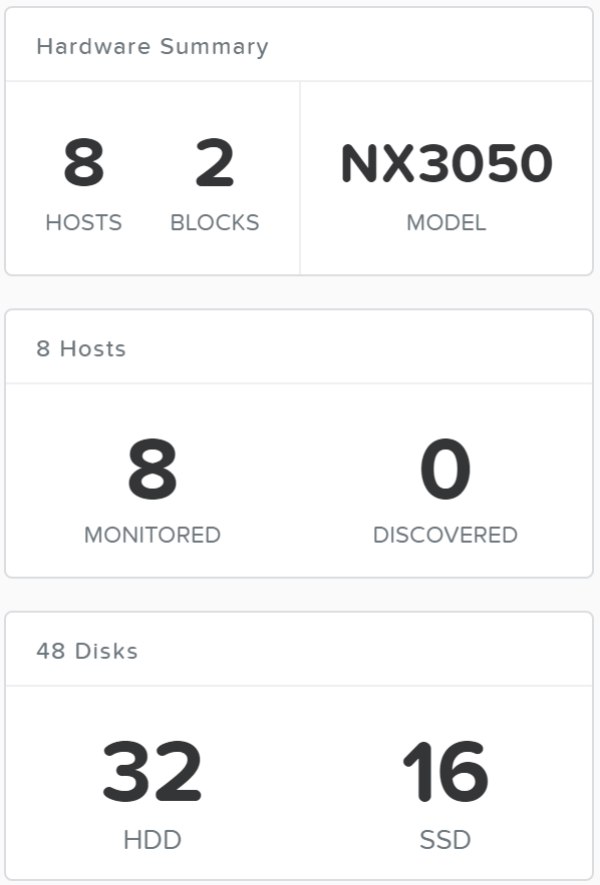


图 7-22：Expand Cluster—执行

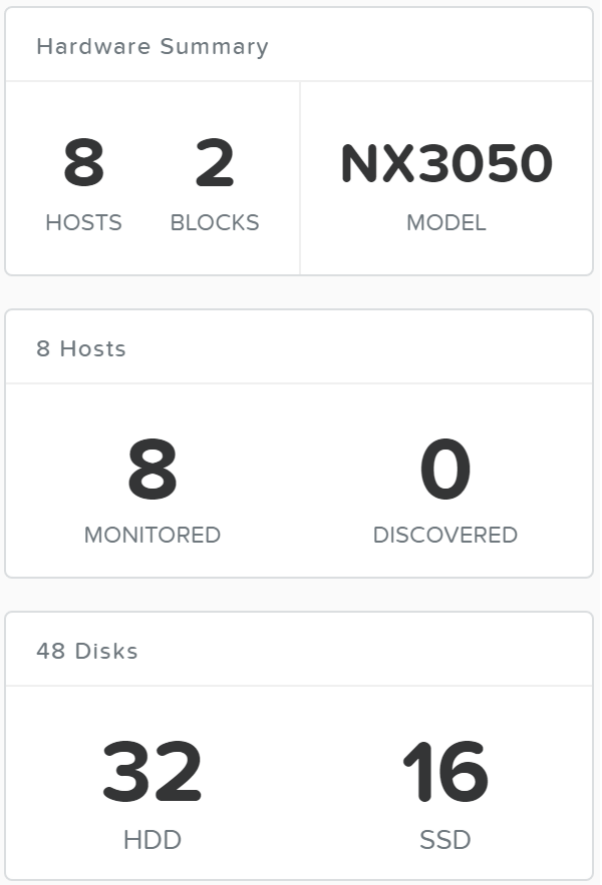


图 7-22. Expand Cluster – 执行

2.4.4 容量规划

可以通过在 Prism Central 的“cluster runway”中点击特定的集群来查看容量规划的具体信息。

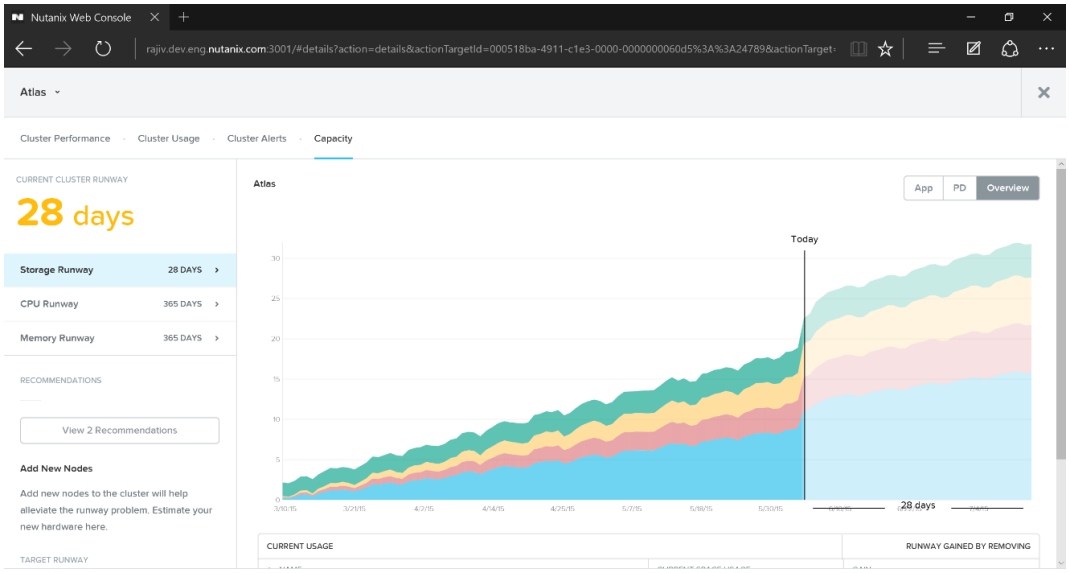


图 7-23. Prism Central - Capacity Planning

这张视图提供了关于集群使用趋势的具体信息并帮助鉴别当前最吃紧的资源情况。你也可以从中了解到占用资源最多的那些虚机，并得到一些释放资源的建议和扩展集群的节点类型推荐。

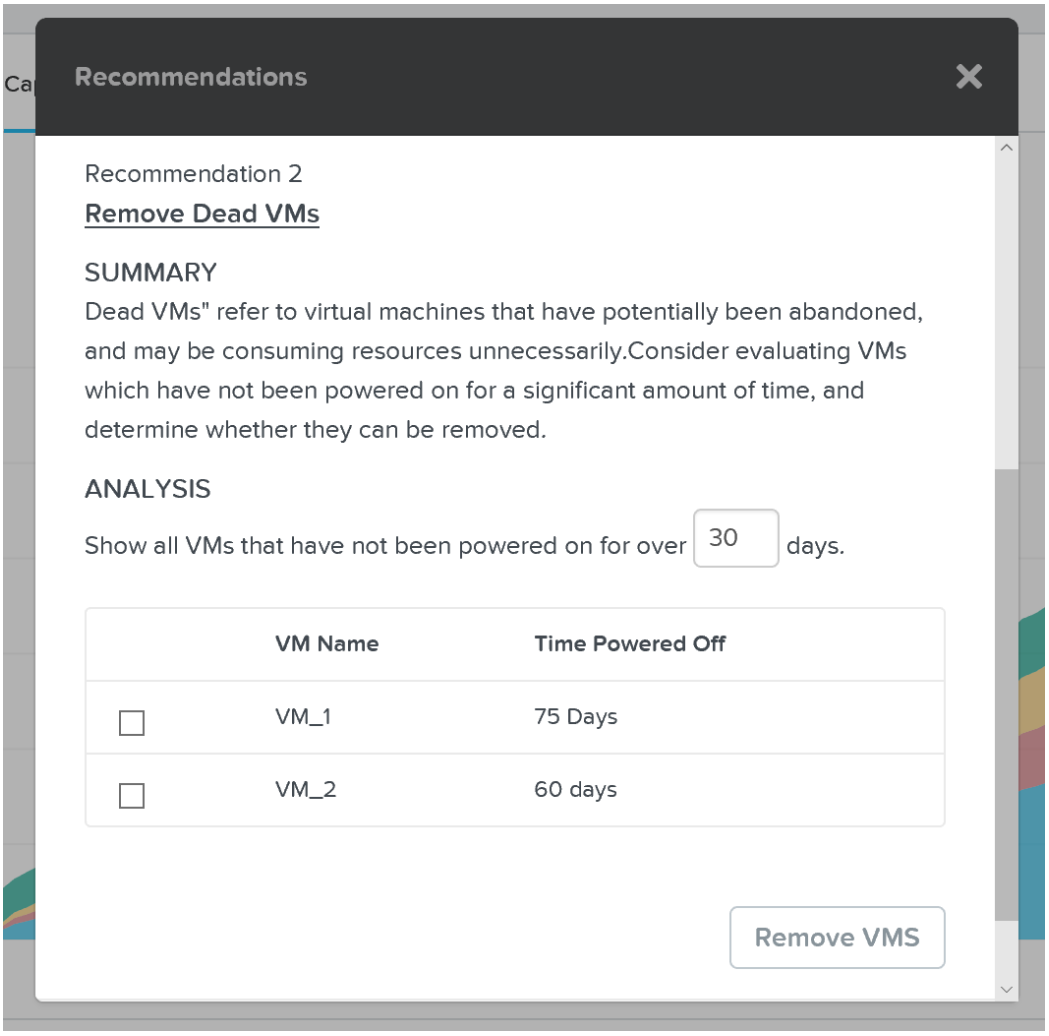


图 7-24. Prism Central – 容量规划 – 推荐建议

基于 HTML5 的用户界面让 Prism 成为易于使用的管理工具。同时，另一项核心功能是对于程序自动化接口的实现。所有 Prism 用户界面提供的功能都可以通过 REST API 编程接口供外部程序调用。这使得用户或是合作伙伴可以实现自动化部署、第三方工具集成甚至是创建他们自己的用户界面。

下面章节涵盖了关于这些接口的内容和一些实例。

2.5 APIs 接口

在动态的、软件定义的环境中，Nutanix 提供了一组/系列接口，让程序调用变得非常简单。以下是主要的接口：

- REST API
- CLI - ACLI & NCLI

- Scripting interfaces

作为这其中的关键部分，REST API 可调用 Prism 的每一个功能和数据点，让工作流/自动化工具可以轻易的驱动 Nutanix 做出反应。可与 Nutanix 集成的自动化工具包括 Saltstack，Puppet，vRealize Operations，System Center Orchestrator 和 Ansible 等。这当然也意味着任何第三方开发人员都能创建他们自己的用户界面或从 Nutanix 平台抽取数据。

下图展示了 Nutanix REST API 浏览器的一小部分，以表明开发者是如何调用这些 API 接口并得到其所希望的数据格式：

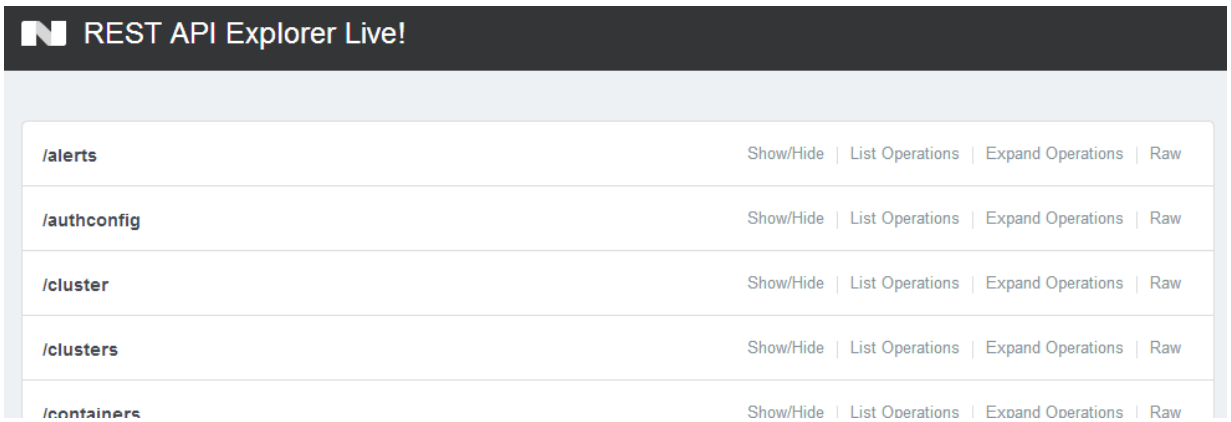


图 8-1. Prism REST API Explorer

可以展开操作来显示 REST 调用的具体信息和例子：

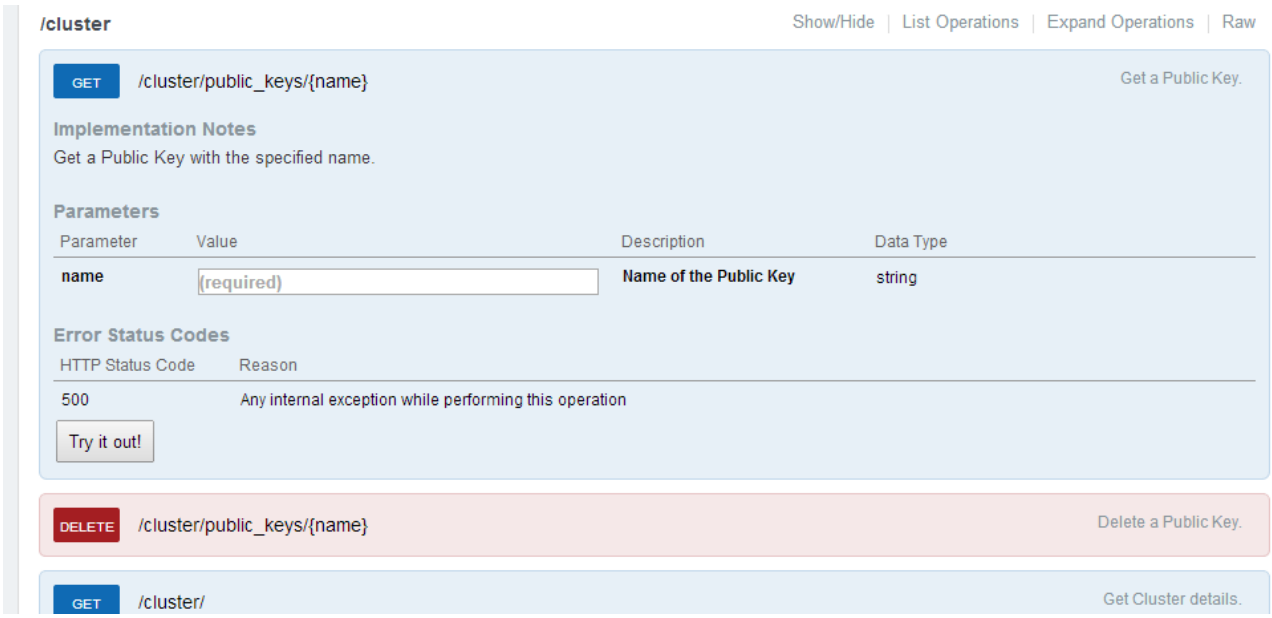


图 8-2. Prism REST API Sample Call

API 验证方案

对于版本 4.5.x 客户端和 HTTP 调用的验证都通过使用 HTTPS 的基本验证方式来实现。

2.5.1 ACLI

Acropolis CLI (ACLI) 是用来管理 Nutanix 产品中 Acropolis 那部分功能的命令行接口。这些功能从 4.1.2 之后被开放。

说明：所有这些操作都可以通过 HTML5 图形用户界面和 REST API 完成。我这里用的命令只是作为我用来实现自动化脚本的一部分。

进入 ACLI Shell

描述: 进入 ACLI shell (可从任何 CVM 运行)。

```
Acli
```

或者

描述: 通过 Linux shell 执行 ACLI 命令

```
ACLI <Command>
```

ACLI 的响应以 json 格式输出

描述: 以 json 格式输出 acli 命令的结果

```
Acli –o json
```

列出主机

描述: 列出集群中的 Acropolis 节点

```
host.list
```

创建网络

描述：根据 VLAN 创建网络

```
net.create <TYPE>.<ID>[.<VSWITCH>] ip_config=<A.B.C.D>/<NN>
Example: net.create vlan.133 ip_config=10.1.1.1/24
```

列出网络

描述：列出网络

```
net.list
```

创建 DHCP 范围

描述：创建 DHCP 范围

```
net.add_dhcp_pool <NET NAME> start=<START IP A.B.C.D> end=<END IP W.X.Y.Z>
```

说明： 如果在创建网络时没有指定 Acropolis DHCP 服务器的地址，那么 .254 被保留并用作 Acropolis DHCP 服务器地址。

```
Example: net.add_dhcp_pool vlan.100 start=10.1.1.100 end=10.1.1.200
```

获得已有网络的详细信息

描述：获得网络属性等信息

```
net.get <NET NAME>
Example: net.get vlan.133
```

获得已有网络中虚机的详细信息

描述：获得一个网络中的虚机及其名字、UUID、MAC 地址及 IP 地址等信息

```
net.list_vms <NET NAME>
Example: net.list_vms vlan.133
```

为网络配置 DHCP 和 DNS 服务器

描述：设置 DHCP 和 DNS

```
net.update_dhcp_dns <NET NAME> servers=<COMMA SEPARATED DNS IPs> domains=<COMMA SEPARATED DOMAINS>
Example: net.set_dhcp_dns vlan.100 servers=10.1.1.1,10.1.1.2 domains=splab.com
```

创建虚机

描述：创建虚机

```
vm.create <COMMA SEPARATED VM NAMES> memory=<NUM MEM MB> num_vcpus=<NUM VCPU> num_cores_per_vcpu=<NUM CORES> ha_priority=<PRIORITY INT>
Example: vm.create testVM memory=2G num_vcpus=2
```

批量创建虚机

描述：批量创建虚机

```
vm.create  <CLONE PREFIX>[<STARTING INT>..<END INT>] memory=<NUM MEM MB> num_vcpus=<NUM VCPU> num_cores_per_vcpu=<NUM CORES> ha_priority=<PRIORITY INT>
Example: vm.create testVM[000..999] memory=2G num_vcpus=2
```

基于已有虚机克隆

描述：创建已有虚机的克隆

```
vm.clone <CLONE NAME(S)> clone_from_vm=<SOURCE VM NAME>
Example: vm.clone testClone clone_from_vm=MYBASEVM
```

基于已有虚机批量克隆

描述：批量创建已有虚机的克隆

```
vm.clone <CLONE PREFIX>[<STARTING INT>..<END INT>] clone_from_vm=<SOURCE VM NAME>
Example: vm.clone testClone[001..999] clone_from_vm=MYBASEVM
```

创建磁盘并添加到虚机

```
# Description: Create disk for OS
vm.disk_create <VM NAME> create_size=<Size and qualifier, e.g. 500G> container=<CONTAINER NAME>
class="codetext"Example: vm.disk_create testVM create_size=500G container=default
```

给虚机添加网卡

描述：创建并添加网卡

```
vm.nic_create <VM NAME> network=<NETWORK NAME> model=<MODEL>
Example: vm.nic_create testVM network=vlan.100
```

设置虚机的启动磁盘

描述：设置启动设备

通过磁盘 ID 设置从指定的磁盘启动

```
vm.update_boot_device <VM NAME> disk_addr=<DISK BUS>
Example: vm.update_boot_device testVM disk_addr=scsi.0
```

设置虚机的启动光驱

设置从光驱启动

```
vm.update_boot_device <VM NAME> disk_addr=<CDROM BUS>
Example: vm.update_boot_device testVM disk_addr=ide.0
```

挂载 ISO 到光驱

描述：挂载 ISO 到虚机的光驱

步骤:

1. 上载 ISOs 到 container
2. 为欲设置的客户端 IP 地址开启白名单
3. 上载 ISOs 以共享

基于 ISO 创建光驱设备

```
vm.disk_create <VM NAME> clone_nfs_file=<PATH TO ISO> cdrom=true
Example: vm.disk_create testVM clone_nfs_file=/default/ISOs/myfile.iso cdrom=true
```

如果光驱已经创建则直接挂载

```
vm.disk_update <VM NAME> <CDROM BUS> clone_nfs_file<PATH TO ISO>
Example: vm.disk_update atestVM1 ide.0 clone_nfs_file=/default/ISOs/myfile.iso
```

从光驱卸除 ISO

描述：从光驱移除 ISO

```
vm.disk_update <VM NAME> <CDROM BUS> empty=true
```

开启虚机

描述：开启虚机

```
vm.on <VM NAME(S)>
```

```
Example: vm.on testVM
```

开启所有虚机

```
Example: vm.on *
```

开启一段编号范围的虚机

```
Example: vm.on testVM[01..99]
```

2.5.2 NCLI

说明：所有这些操作都可以通过 HTML5 图形用户界面和 REST API 完成。我这里用的命令只是作为我用来实现自动化脚本的一部分。.

添加子网到 NFS 白名单

描述：添加特定的子网到 NFS 白名单

```
ncli cluster add-to-nfs-whitelist ip-subnet-masks=10.2.0.0/255.255.0.0
```

显示 Nutanix 版本

描述：显示当前 Nutanix 软件的版本

```
ncli cluster version
```

显示 NCLI 的隐藏选项

描述：显示 ncli 的隐藏命令/选项

```
ncli helpsys listall hidden=true [detailed=false|true]
```

列出存储资源池

描述：显示已存在的存储资源池

```
ncli sp ls
```

列出容器

描述：列出已有的容器

```
ncli ctr ls
```

创建容器

描述：创建一个新的容器

```
ncli ctr create name=<NAME> sp-name=<SP NAME>
```

列出虚机

描述：显示已存在的虚机

```
ncli vm ls
```

列出共有密钥

描述：列出已存在公有密钥

```
ncli cluster list-public-keys
```

添加公有密钥

描述：为集群访问添加公有密钥

通过 SCP 传输公有密钥到 CVM

添加公有密钥到集群

```
ncli cluster add-public-key name=myPK file-path=~/mykey.pub
```

移除公有密钥

描述：移除集群访问的公有密钥

```
ncli cluster remove-public-keys name=myPK
```

创建保护域

描述：创建一个保护域

```
ncli pd create name=<NAME>
```

创建远程站点

描述：为数据复制创建远程站点

```
ncli remote-site create name=<NAME> address-list=<Remote Cluster IP>
```

为容器内的所有虚机创建保护域

描述：保护特定容器中的所有虚机

```
ncli pd protect name=<PD NAME> ctr-id=<Container ID> cg-name=<NAME>
```

为特定虚机创建保护域

描述：保护特定虚机

```
ncli pd protect name=<PD NAME> vm-names=<VM Name(s)> cg-name=<NAME>
```

为 DSF 文件（亦称作 vDisk）创建保护域

描述：保护特定的 DSF 文件

```
ncli pd protect name=<PD NAME> files=<File Name(s)> cg-name=<NAME>
```

创建保护域的快照

描述：为保护域创建一次性的快照

```
ncli pd add-one-time-snapshot name=<PD NAME> retention-time=<seconds>
```

创建快照和同步到远程站点的复制计划

描述：创建一个持续的快照计划，同步数据到远程站点

```
ncli pd set-schedule name=<PD NAME> interval=<seconds> retention-policy=<POLICY> remote-sites=<REMOTE SITE NAME>
```

列出同步状态

描述：监控数据复制状态

```
ncli pd list-replication-status
```

迁移保护域到远程站点

描述：故障切换一个保护域到远程站点

```
ncli pd migrate name=<PD NAME> remote-site=<REMOTE SITE NAME>
```

激活保护域

描述：激活远程站点的保护域

```
ncli pd activate name=<PD NAME>
```

开启 DSF 的 Shadow Clones

描述：开启 DSF 的 Shadow Clone 功能

```
ncli cluster edit-params enable-shadow-clones=true
```

打开 vDisk 的去重

描述：为特定 vDisk 开启指纹识别并启用去重

```
ncli vdisk edit name=<VDISK NAME> fingerprint-on-write=<true/false> on-disk-dedup=<true/false>
```

2.5.3 PowerShell CMDlets

下面将讲述 Nutanix 对 PowerShell CMDlets 命令行支持，包括如何使用和一些关于 Windows PowerShell 的背景知识。

基础

Windows PowerShell 是一套强大的集基于.NET 框架的 shell 脚本支持工具。它是非常易用的编写语言，直观且交互性好。PowerShell 中有一些关键结构和组件：

CMDlets

CMDlets 是一组执行特定操作的命令或.NET 类。他们符合 Getter/Setter 方法论，并且通常使用<动词>-<名词>的操作结构。例如：Get-Process, Set-Partition 等。

管道（Piping）和流水线（Pipelining）

管道是 PowerShell 中的重要概念（类似于 Linux 中的用法），用好了的话能使很多事变得简单。通过管道，你其实是把一个流水线操作的输出导向到作为下一个流水线操作的输入。流水线可以根据需要变长。一个非常简单的例子：获得当前进程，找到那些符合一定特性或过滤条件的然后再排序。

```
Get-Service | where {$_.Status -eq "Running"} | Sort-Object Name
```

管道也可以被用在 for-each 循环中，例如：

```
# For each item in my array
$myArray | %{
 # Do something
}
```

关键对象类型

下面是 PowerShell 中一些关键对象类型。你可以通过.getType()方法非常容易的得到一个对象的类型，比如：$someVariable.getType()将会返回特定对象的类型。

变量

```
$myVariable = "foo"
```

说明：你可以为变量赋值为一系列命令或管道的输出:

```
$myVar2 = (Get-Process | where {$_.Status -eq "Running”})
```

在这个例子中，将先运算括号中的命令，然后把其赋值给变量。

数组

```
$myArray = @("Value","Value")
```

说明：你可以得到阵列、哈希表或自定义对象的数组。

哈希表

```
$myHash = @{"Key" = "Value","Key" = "Value"}
```

有用的命令

获得特定 CMDlet 命令的帮助信息（类似于 Linux 中的 man）

```
Get-Help <CMDlet Name>
Example: Get-Help Get-Process
```

列出一个命令或对象的属性和方法

```
<Some expression or object> | Get-Member
Example: $someObject | Get-Member
```

核心 Nutanix CMDlets 和使用方法

下载 Nutanix CMDlets 安装器。Nutanix CMDlets 可以通过 Prism（4.0.1 以后版本）用户界面直接下载，就在右上角的下拉列表中。

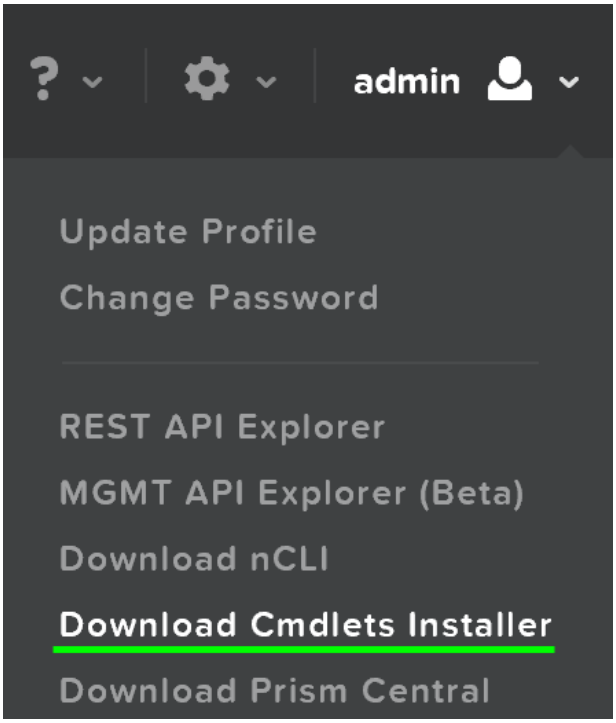


图 8-3. Prism CMDlets Installer Link

装载 Nutanix Snap-in

检查 snap-in 是否被装载，如没有就装载

```
    if ( (Get-PSSnapin -Name NutanixCmdletsPSSnapin -ErrorAction SilentlyContinue) -eq $null )
{
   Add-PsSnapin NutanixCmdletsPSSnapin
}
```

列出 Nutanix CMDlets

```
    Get-Command | Where-Object{$_.PSSnapin.Name -eq "NutanixCmdletsPSSnapin"}
```

连接到一个 Acropolis 集群

```
    Connect-NutanixCluster -Server $server -UserName "myuser" -Password "myuser" -
AcceptInvalidSSLCerts
```

或者采用更加安全的交互式输入口令

```
    Connect-NutanixCluster -Server $server -UserName "myuser" -Password (Read-Host "Password: ") -
AcceptInvalidSSLCerts
```

获得满足一定搜索条件的虚机

变量方式

```
$searchString = "myVM"
$vms = Get-NTNXVM | where {$_.vmName -match $searchString}
```

交互式

```
Get-NTNXVM | where {$_.vmName -match "myString"}
```

交互式的格式化输出

```
Get-NTNXVM | where {$_.vmName -match "myString"} | ft
```

获得 Nutanix vDisks

变量方式

```
$vdisks = Get-NTNXVDisk
```

交互式

```
Get-NTNXVDisk
```

交互式的格式化输出

```
Get-NTNXVDisk | ft
```

获得 Nutanix 容器

变量方式

```
$containers = Get-NTNXContainer
```

交互式

```
Get-NTNXContainer
```

交互式的格式化输出

```
Get-NTNXContainer | ft
```

获得 Nutanix 保护域

变量方式

```
$pds = Get-NTNXProtectionDomain
```

交互式

```
Get-NTNXProtectionDomain
```

交互式的格式化输出

```
Get-NTNXProtectionDomain | ft
```

获得 Nutanix 一致性组

变量方式

```
$cgs = Get-NTNXProtectionDomainConsistencyGroup
```

交互式

```
Get-NTNXProtectionDomainConsistencyGroup
```

交互式的格式化输出

```
Get-NTNXProtectionDomainConsistencyGroup | ft
```

资源和脚本：

- Nutanix Github - https://github.com/nutanix/Automation
- Manually Fingerprint vDisks - http://bit.ly/1syOqch
- vDisk Report - http://bit.ly/1r34MIT
- Protection Domain Report - http://bit.ly/1r34MIT
- Ordered PD Restore - http://bit.ly/1pyolrb

你可以在位于以下网址的 Nutanix Github 上找到更多脚本

https://github.com/nutanix

2.6 集成

2.6.1 OpenStack

OpenStack 是一个用来管理和构建云的开源平台。它主要被分为两个部分，前端(仪表盘和 API)和基础架构服务(计算，存储资源等)。

OpenStack 和 Nutanix 解决方案有以下主要组件构成：

- OpenStack 控制器(OSC)
  - 一个现存的或新创建的虚机或主机，用来提供 OpenStack 用户界面、API 和服务。处理所有 OpenStack 的 API 调用。它能跟 Acropolis OpenStack 驱动一起共存在同一个 Acropolis OVM 中。
- Acropolis OpenStack Driver
  - 负责处理来自 OpenStack 控制器的 OpenStack RPC，并翻译成 native Acropolis API 调用。这可以部署在 OpenStack 控制器，预装的 OVM 或是一个新的虚机上。
- Acropolis OpenStack 服务虚机 (OVM)

- 装有 Acropolis 驱动的虚机，负责处理来自 OpenStack 控制器的 OpenStack RPC，并翻译成 native Acropolis API 调用。

OpenStack 控制器可以是一个已经存在的虚机/主机，或是部署为 OpenStack on Nutanix 解决方案的一部分。Acropolis OVM 通常作为一个帮助类型的虚机部署在 Nutanix 的 OpenStack 解决方案当中。

客户端用他们期望的方式（Web UI / HTTP, SDK, CLI or API）来与 OpenStack 控制器沟通。而 OpenStack 控制器则与 Acropolis OVM 通讯，后者利用 OpenStack 驱动把指令转换成 Acropolis 自身的 REST API 调用。

下图描述这种通讯的概要过程：

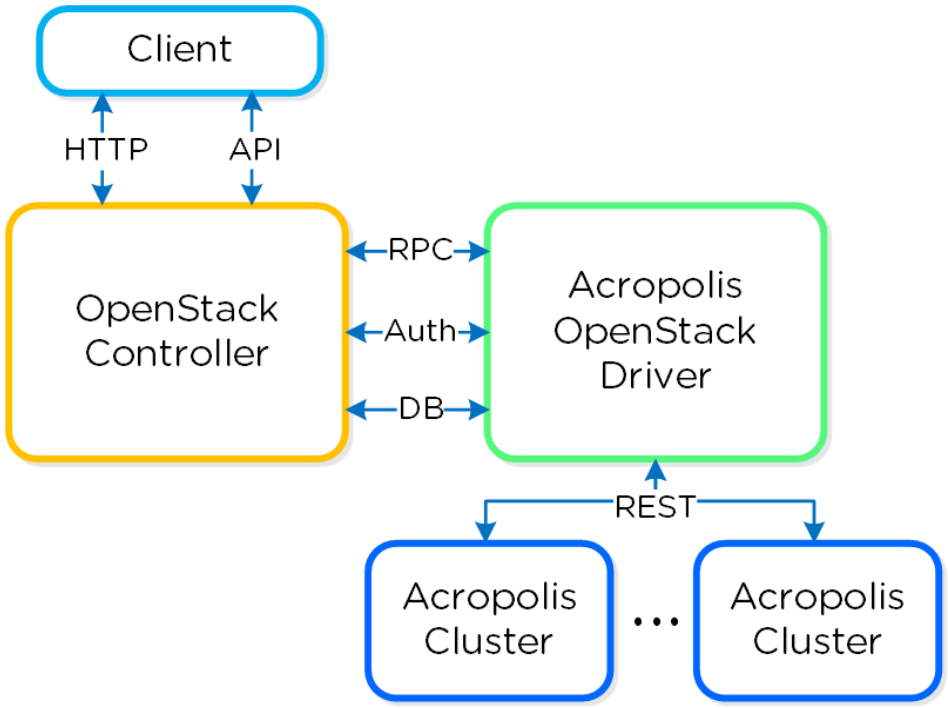


图 9-1. OpenStack + Acropolis OpenStack Driver

支持的 OpenStack 控制器

当前版本 (对应于 4.5.1) 需要一个 Kilo 或更新版本的 OpenStack 控制器。

下表列出了各组件角色映射的概要信息：

条目	角色	OpenStack 控制器	Acropolis	Acropolis 集群

			OVM	
Tenant Dashboard	User interface and API	X		
Orchestration	Object CRUD and lifecycle management	X		
Quotas	Resource controls and limits	X		
Users, Groups and Roles	Role based access control (RBAC)	X		
SSO	Single-sign on	X		
Platform Integration	OpenStack to Nutanix integration		X	
Infrastructure Services	Target infrastructure (compute, storage, network)			X

2.6.1.1 **OpenStack 组件**

OpenStack 由一系列提供基础架构功能服务的组件构成。这中间的有些功能由 OpenStack 控制器提供，有些则由 Acropolis OVM 提供。

下表列出了核心的 OpenStack 组件及其角色的映射：

组件	角色	OpenStack 控制	Acropolis OVM
Keystone	Identity service	X	
Horizon	Dashboard and UI	X	
Nova	Compute		X
Swift	Object storage	X	X
Cinder	Block storage		X
Glance	Image service	X	X
Neutron	Networking		X
Heat	Orchestration	X	
Others	All other components	X	

下面提供了一个 OpenStack 组件及其通讯过程更为详细的视图：

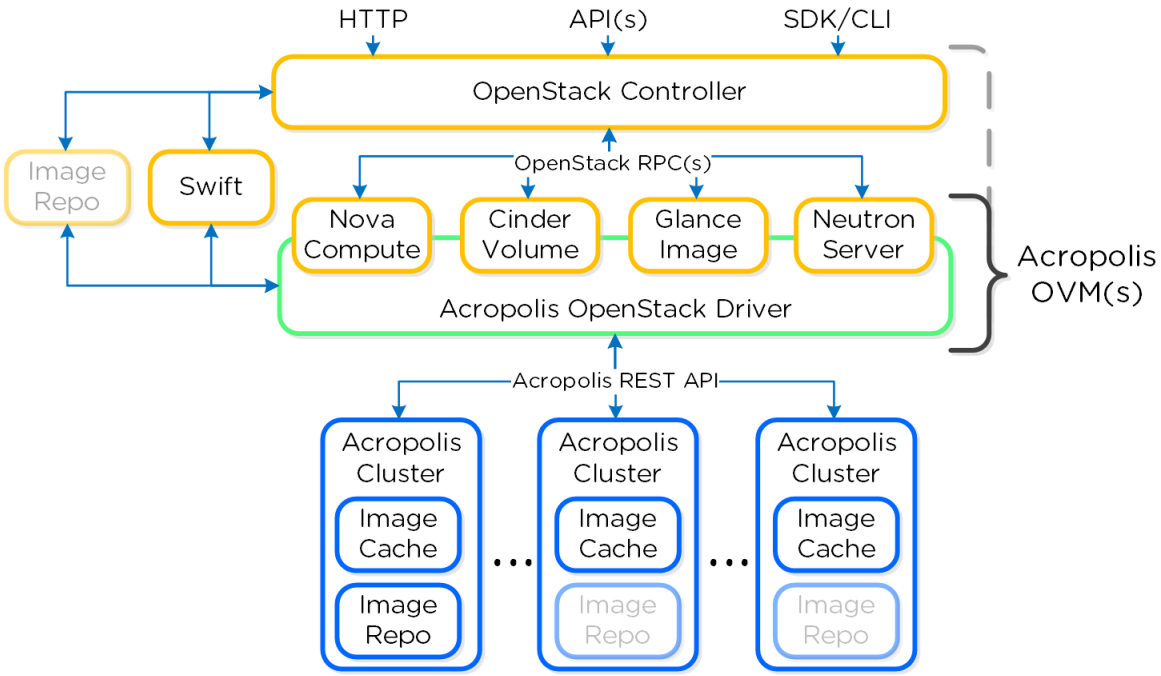


图 9-2. OpenStack + Nutanix API Communication

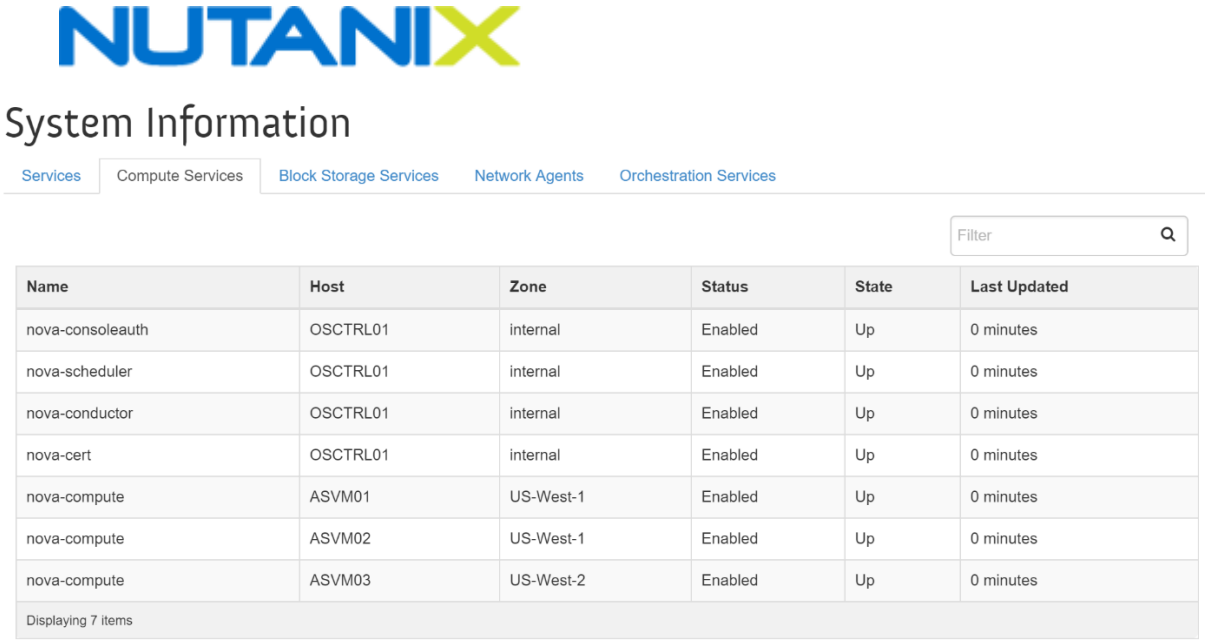
在下面的章节中我们将遍历主要的 OpenStack 组件并了解他们是如何被集成到 Nutanix 平台中的。

Nova

Nova 是 OpenStack 平台中计算资源的引擎和调度者。在 Nutanix 的 OpenStack 解决方案中每个 Acropolis OVM 类似于一个计算机主机，而每个 Acropolis 集群将充当一个单一的 hypervisor 主机，为 OpenStack 实例提供资源调度。这样的 Acropolis OVM 运行 Nova 计算资源服务。

你可以通过 OpenStack 门户从下面路径中查看 Nova 服务：'Admin'->'System'->'System Information'->'Compute Services'.

下图展示了 Nova 服务，主机及其状态：



Name	Host	Zone	Status	State	Last Updated
nova-consoleauth	OSCTRL01	internal	Enabled	Up	0 minutes
nova-scheduler	OSCTRL01	internal	Enabled	Up	0 minutes
nova-conductor	OSCTRL01	internal	Enabled	Up	0 minutes
nova-cert	OSCTRL01	internal	Enabled	Up	0 minutes
nova-compute	ASVM01	US-West-1	Enabled	Up	0 minutes
nova-compute	ASVM02	US-West-1	Enabled	Up	0 minutes
nova-compute	ASVM03	US-West-2	Enabled	Up	0 minutes

Displaying 7 items

图 9-3. OpenStack Nova Services

Nova 调度器基于选定的可用区，决定在哪个计算主机（如 Acroplis OVM）上存放实例。这些请求将会被发送到选定的 Acropolis OVM 上，然后由 Acropolis OVM 转发请求到目标主机的 Acropolis 调度器。Acropolis 调度器将在集群中选出最优的节点存放方式。集群中各节点不会直接暴露给 OpenStack。

你可以通过 OpenStack 门户，在路径'Admin'->'System'->'Hypervisors'中查看计算机和 hypervisor 主机。

下图以计算机主机的形式列出了 Acropolis OVM：

Host	Zone	Status	State
ASVM01	US-West-1	enabled	up
ASVM02	US-West-1	enabled	up
ASVM03	US-West-2	enabled	up

图 9-4. OpenStack Compute Host

下图以 hypervisor 主机的形式列出了 Acropolis 集群：

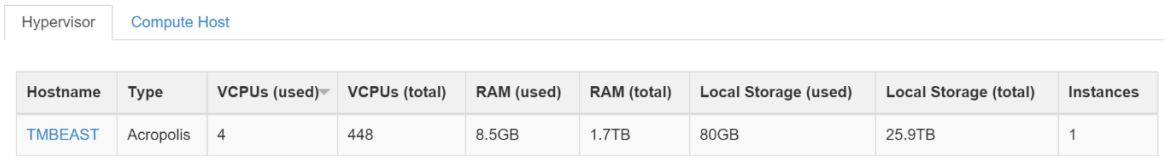


图 9-5. OpenStack Hypervisor Host

如你从上面的图示所见，所有集群的资源以单个 hypervisor 主机的方式展现。

Swift

在对象存储中 Swift 被用来存放和获得文件。它当前主要被用来做快照和镜像的备份/恢复。

Cinder

Cinder 是 OpenStack 的卷组件，对外提供 iSCSI 对象。在 Nutanix 解决方案中 Cinder 的实现借助于 Acropolis 卷管理 API。这些卷将以块设备的形式直接附加到实例上去（相较于 in-guest）。

你可以通过 OpenStack 门户，在路径'Admin'->'System'->'System Information'->'Block Storage Services'下查看 Cinder 服务。

下图显示了 Cinder 服务，主机和状态：

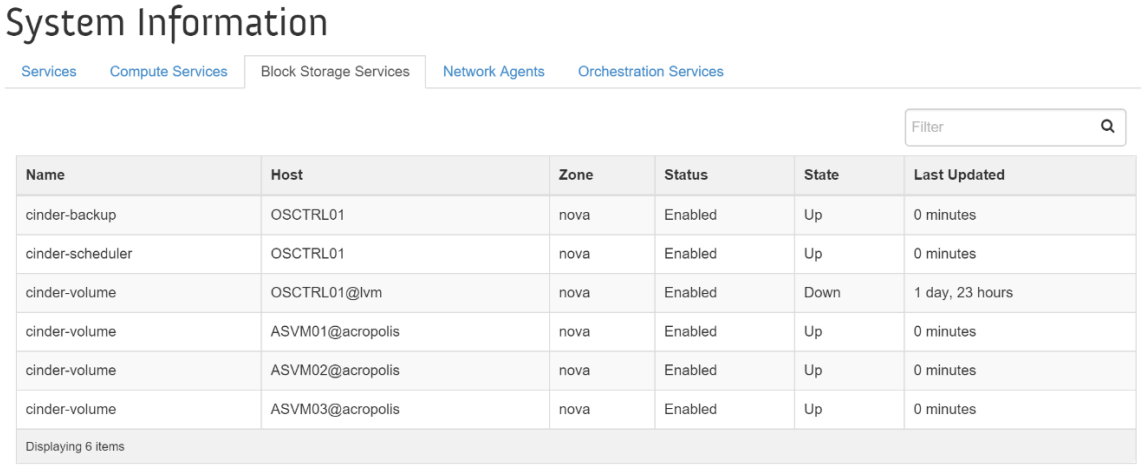


图 9-6. OpenStack Cinder Services

Glance / Image Repo

Glance 是 OpenStack 中的镜像库，展现部署实例的可用镜像。镜像包括 ISO，磁盘和快照。

Image Repo 是用来保存由 Glance 发布的可用镜像的资源池。这些镜像将在 Nutanix 环境中被落实或是被外部源所访问。当镜像被保存在 Nutanix 平台上时，

他们通过 OVM 以 Glance 的形式发布到 OpenStack 控制器。当 Image Repo 只存在于外部源时，Glance 将被 OpenStack 控制器接管，而 Acropolis 集群则会借助于 Image Cache。

Glance 以跟集群一对一的方式被启用，并且总是和 Image Repo 共存。当 Glance 同时在多个集群中启动时，Image Repo 会横跨那些集群，并且通过 OpenStack 们创建的镜像将会被推送到所有运行 Glance 的集群。没有运行 Glance 的集群将利用 Image Cache 在本地缓存镜像。

专家提示

对于大规模部署，每个站点至少保证有两个 Acropolis 集群在运行 Glance。这样就能对 Image Repo 实现高可用，以便在一个集群不可用时，镜像还是能够被访问到，即便它不在 Image Cache 中。

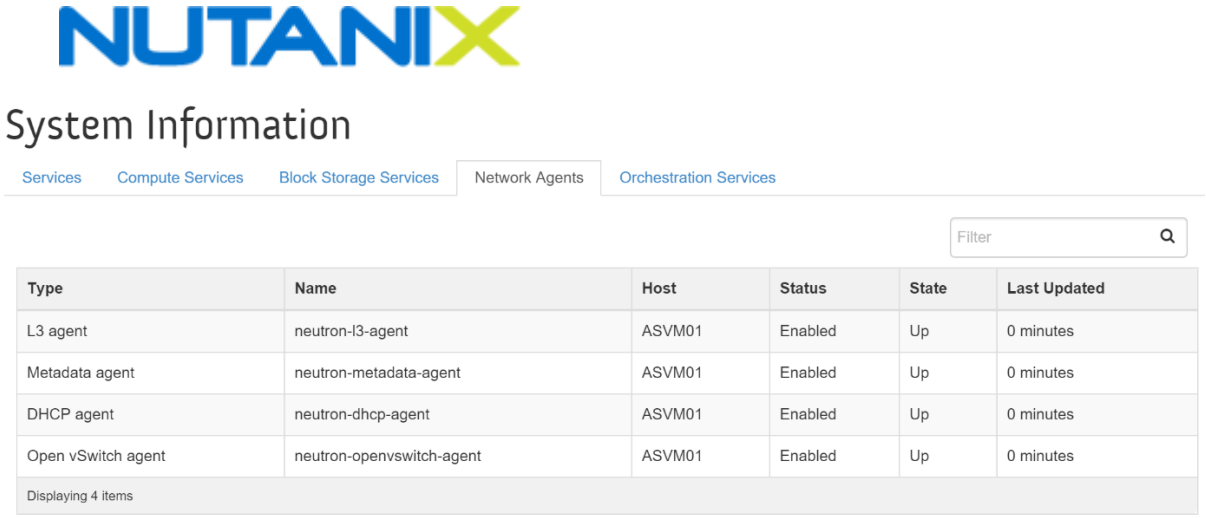
当以外部源运行 Image Repo/Glance 时，Nova 会负责把数据从外部源转移到目标端的 Acropolis 集群。在这种情况下目标端的 Acropolis 集群会利用 Image Cache 来缓存镜像，以便后续访问时能从本地读到。

Neutron

Neutron 是 OpenStack 中的网络组件，用来实现网络配置。Acropolis OVM 允许用户通过 OpenStack 门户进行网络 CRUD 的操作，随后把相应的更改更新到 Acropolis 中去。

你可以通过 OpenStack 门户，从'Admin'->'System'->'System Information'->'Network Agents'查看 Neutron 服务。

下图展示了 Neutron 服务，主机和状态：



+ 当前只支持 LAN 和 VLAN 类型

图 9-7. OpenStack Neutron Services

Neutron 在实例启动时分配 IP 地址。Acropolis 也正是以这种方式得到相应虚机的 IP 地址。当虚机发出 DHCP 请求时，Acropolis Master 将像对待 Acropolis Hypervisor 一样在私有的 VXLAN 上对此请求做出响应。

被支持的网络类型

当前只支持 LAN 和 VLAN 网络类型

运行在 OpenStack 控制器中的 Keystone 和 Horizon 组件也都和 Acropolis OVM 打交道。OVM 中的 OpenStack 驱动将把 OpenStack API 调用翻译成原生的 Acropolis API 调用。

2.6.1.2 设计和部署

对于大规模的云部署，一个好的交付拓扑是至关重要的。它将被分发出去，在提供灵活性和本地性的同时又能满足最终用户的需求。

下面是在 OpenStack 中用到的高层级的概念：

- 区域-Region
  - 一个地理意义上区域，其上可构建多个可用区（AZ）。区域可以设置成像 US-Northwest 或是 US-West 这样。
- 可用区-Availability Zone (AZ)
  - 一个运行着云服务的特定站点或数据中心的位置。它可以包含像是 US-Northwest-1 or US-West-1 这样的站点。

- 主机聚合-Host Aggregate
  - 一组计算主机，可以是一排、一整片或是如站点或 AZ 那样的范围。
- 计算主机-Compute Host
  - 一个运行着 nova 计算服务的 Acropolis OVM。
- 虚拟管理程序主机-Hypervisor Host
  - 一个 Acropolis 集群（以一个单一主机的形式可见）。

下图展示了以上各概念的高层级关系：

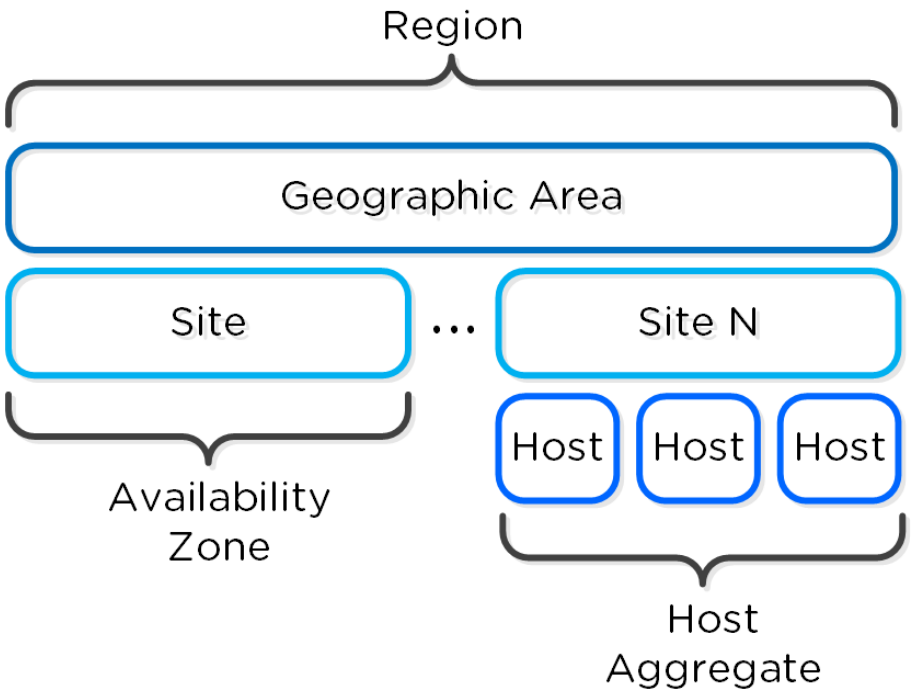


图 9-8. OpenStack - Deployment Layout

下图展示了各概念应用的例子：

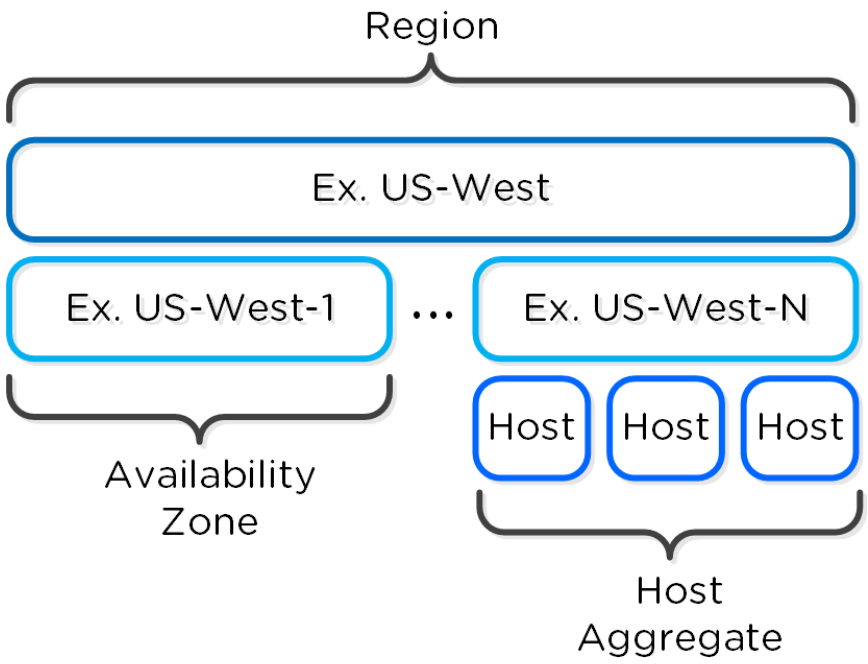


图 9-9. OpenStack - Deployment Layout - Example

你可以通过 OpenStack 门户，从'Admin'->'System'->'Host Aggregates'查看和管理主机，主机聚合和可用区。

下图展示了主机聚合，可用区和主机：

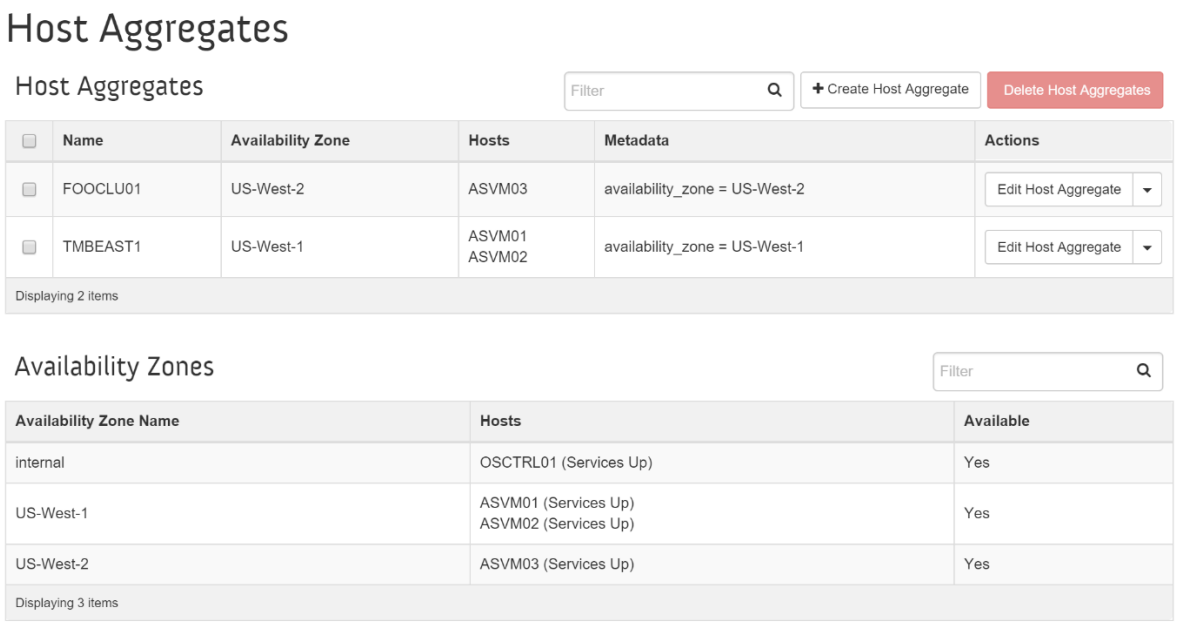


图 9-10. OpenStack Host Aggregates and Availability Zones

2.6.1.3 服务设计和延展

对于大规模部署来说，推荐以负载均衡的方式把多个 Acropolis OVM 连接到 OpenStack 控制器。这样既保证了可用性又均分了工作负载。OVM 本身不包含任何处于延展性要求的状态信息。

下图展示了如何在一个单一站点扩展 OVM：

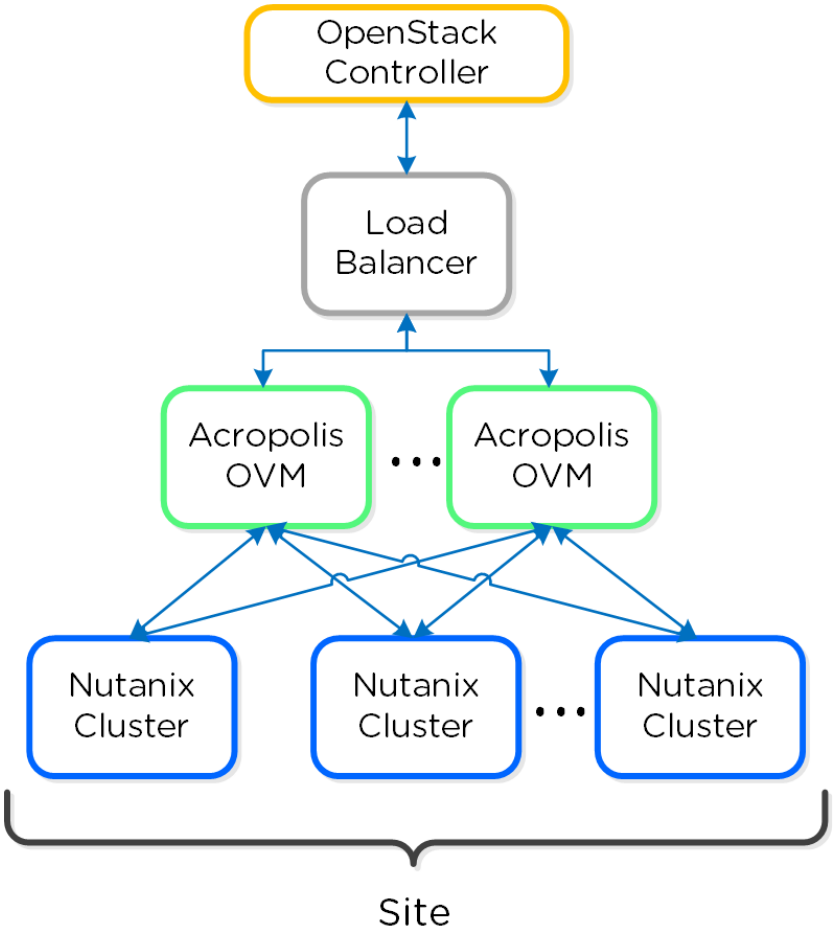


图 9-11. OpenStack - OVM Load Balancing

一种实现此种 OVM 架构的方式是使用 Keepalived 和 HAproxy。

对于横跨多个站点的环境，OpenStack 控制器会跟这些跨站点的 Acropolis OVM 都进行对话。

下图展示了一个跨站点部署的例子：

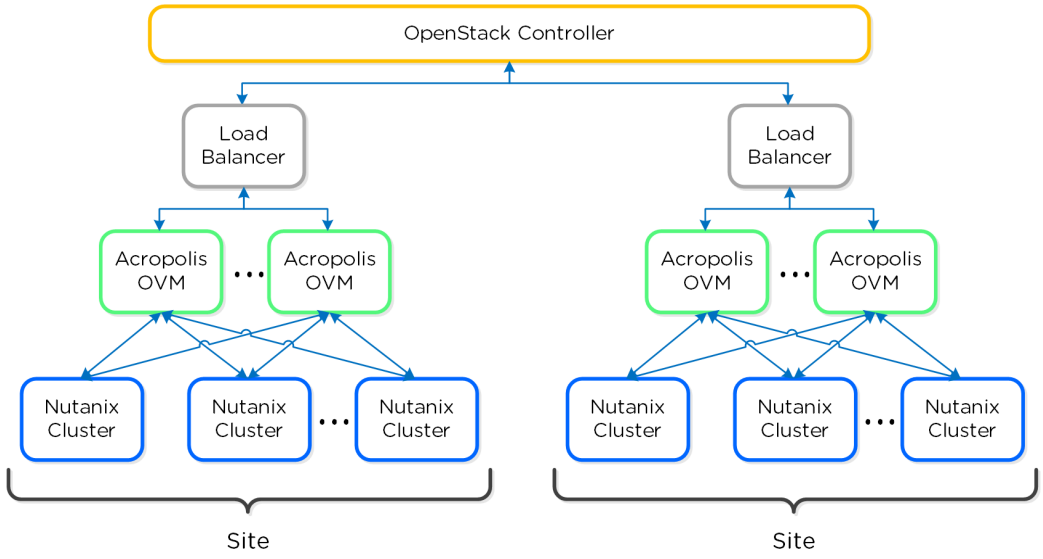


图 9-12. OpenStack - Multi-Site

2.6.1.4部署

OVM 可以以独立的 RPM 包形式部署在 CentOS/Redhat 发行版本当中，或是作为一个虚机直接运行。Acropolis OVM 可以部署在任何 Nutanix 或非 Nutanix 的平台，只要其能连接到 OpenStack 控制器和 Nutanix 集群。

Acropolis OVM 的虚机可以通过以下方式被部署到 Nutanix AHV 集群。你可以用完整的 OVM 镜像或从一个 CentOS/Redhat 的虚机镜像做起。

首先我们将把 Acropolis OVM 的磁盘镜像导入到 Acropolis 集群。可以用 SCP 来传输镜像或是指定一个 URL 来导入。我们将用镜像服务的 API 来完成此步。注意：OVM 的虚机能部署到任何地方，不一定是 Acropolis 集群。

运行以下命令，通过镜像 API 导入磁盘镜像：

```
image.create <IMAGE_NAME> source_url=<SOURCE_URL> container=<CONTAINER_NAME>
```

从任何 CVM 运行下列 ACLI 命令来创建 OVM 虚机：

```
vm.create <VM_NAME> num_vcpus=2 memory=16G
vm.disk_create <VM_NAME> clone_from_image=<IMAGE_NAME>
vm.nic_create <VM_NAME> network=<NETWORK_NAME>
vm.on <VM_NAME>
```

虚机创建完毕并开启后，用给定的用户名和口令 SSH 到 OVM。

OVMCTL Help

在 OVM 上运行以下命令来获得帮助文本:

```
ovmctl --help
```

OVM 支持两种部署方式：

- OVM-allinone
  - OVM 包含所有 Acropolis 驱动和 OpenStack 控制器
- OVM-services
  - OVM 包含所有 Acropolis 驱动并且和外部/远端的 OpenStack 控制通讯

在下面章节两种部署方式都会讲到。你可以用其中的任何一种并且自由切换。

OVM-allinone

下列步骤涵盖了 OVM-allinone 的部署，从 SSH 到 OVM 开始：

```
    # Register OpenStack Driver service
ovmctl --add ovm --name <OVM_NAME> --ip <OVM_IP>
    # Register OpenStack Controller
ovmctl --add controller --name <OVM_NAME> --ip <OVM_IP>
    # Register Acropolis Cluster(s) (run for each cluster to add)
ovmctl --add cluster --name <CLUSTER_NAME> --ip <CLUSTER_IP> --username <PRISM_USER> --
password <PRISM_PASSWORD>

The following values are used as defaults:
Number of VCPUs per core = 4
Container name = default
Image cache = disabled, Image cache URL = None
```

接下来我们用下面命令来确认配置：

```
    ovmctl --show
```

至此为止，所有的服务都应该起来并运行着。

OVM-services

下列步骤介绍了 OVM-services 的部署方式，从 SSH 到 OVM 开始：

```
    # Register OpenStack Driver service
ovmctl --add ovm --name <OVM_NAME> --ip <OVM_IP>
    # Register OpenStack Controller
ovmctl --add controller --name <OS_CONTROLLER_NAME> --ip <OS_CONTROLLER_IP> --username
<OS_CONTROLLER_USERNAME> --password <OS_CONTROLLER_PASSWORD>

The following values are used as defaults:
Authentication: auth_strategy = keystone, auth_region = RegionOne
auth_tenant = services, auth_password = admin
Database: db_{nova,cinder,glance,neutron} = mysql, db_{nova,cinder,glance,neutron}_password = admin
RPC: rpc_backend = rabbit, rpc_username = guest, rpc_password = guest
    # Register Acropolis Cluster(s) (run for each cluster to add)
ovmctl --add cluster --name <CLUSTER_NAME> --ip <CLUSTER_IP> --username <PRISM_USER> --
password <PRISM_PASSWORD>

The following values are used as defaults:
Number of VCPUs per core = 4
Container name = default
Image cache = disabled, Image cache URL = None
```

如果 OpenStack 控制器用的是非缺省的口令，我们需要更新以下信息：

```
    # Update controller passwords (if non-default are used)
ovmctl --update controller --name <OS_CONTROLLER_NAME> --auth_nova_password <> --
auth_glance_password <> --auth_neutron_password <> --auth_cinder_password <> --db_nova_password <>
--db_glance_password <> --db_neutron_password <> --db_cinder_password <>
```

接下来我们用下面命令来确认配置：

```
    ovmctl --show
```

既然 OVM 已经被设置完毕，我们就要配置 OpenStack 控制器使其能够获知 Glance 和 Neutron 的 endpoint 信息。

登录 OpenStack 控制器并进入 keystonerc_admin 源环境：

```
    # enter keystonerc_admin source ./keystonerc_admin
```

首先我们要删除原来指向控制器的 Glance endpoint：

```
# Find old Glance endpoint id (port 9292) keystone endpoint-list # Remove old keystone endpoint for Glance
keystone endpoint-delete <GLANCE_ENDPOINT_ID>
```

下一步我们将创建新的指向 OVM 的 Glance endpoint：

```
# Find Glance service id
keystone service-list | grep glance
# Will look similar to the following:
| 9e539e8dee264dd9a086677427434982 | glance | image |

# Add Keystone endpoint for Glance
keystone endpoint-create \
--service-id <GLANCE_SERVICE_ID> \
--publicurl http://<OVM_IP>:9292 \
--internalurl http://<OVM_IP>:9292 \
--region <REGION_NAME> \
--adminurl http://<OVM_IP>:9292
```

接下来删除原有指向控制器的 Neutron endpoint：

```
# Find old Neutron endpoint id (port 9696) keystone endpoint-list # Remove old keystone endpoint for Neutron
keystone endpoint-delete <NEUTRON_ENDPOINT_ID>
```

下一步创建新的指向 OVM 的 Neutron endpoint：

```
# Find Glance service id
keystone service-list | grep glance
# Will look similar to the following:
| f4c4266142c742a78b330f8bafe5e49e | neutron | network |

# Add Keystone endpoint for Neutron
keystone endpoint-create \
--service-id <NEUTRON_SERVICE_ID> \
--publicurl http://<OVM_IP>:9696 \
--internalurl http://<OVM_IP>:9696 \
--region <REGION_NAME> \
--adminurl http://<OVM_IP>:9696
```

Endpoint 创建完毕后，我们将用新的 Glance 主机的 Acropolis OVM 的 IP 来更新 Nova 和 Cinder 的配置。

依照下列内容编辑/etc/nova/nova.conf

```
[glance]
...
# Default glance hostname or IP address (string value)
host=<OVM_IP>

# Default glance port (integer value)
port=9292
...
# A list of the glance api servers available to nova. Prefix
# with https:// for ssl-based glance api servers.
# ([hostname|ip]:port) (list value)
api_servers=<OVM_IP>:9292
```

现在停掉 OpenStack 控制器上的 Nova 计算服务：

```
systemctl disable openstack-nova-compute.service
systemctl stop openstack-nova-compute.service
service openstack-nova-compute stop
```

依照下列内容编辑/etc/cinder/cinder.conf

```
# Default glance host name or IP (string value)
glance_host=<OVM_IP>
# Default glance port (integer value)
glance_port=9292
# A list of the glance API servers available to cinder
# ([hostname|ip]:port) (list value)
glance_api_servers=$glance_host:$glance_port
```

注释掉 lvm 作为 cinder 后台的选项，因为它根本不会被用到：

```
# Comment out the following lines in cinder.conf
#enabled_backends=lvm
#[lvm]
#iscsi_helper=lioadm
#volume_group=cinder-volumes
```

```
#iscsi_ip_address=
#volume_driver=cinder.volume.drivers.lvm.LVMVolumeDriver
#volumes_dir=/var/lib/cinder/volumes
#iscsi_protocol=iscsi
#volume_backend_name=lvm
```

现在停掉 OpenStack 控制器上的 cinder 服务：

```
    systemctl disable openstack-cinder-volume.service
systemctl stop openstack-cinder-volume.service
service openstack-cinder-volume stop
```

也停掉 OpenStack 上的 Glance 镜像服务：

```
    systemctl disable openstack-glance-api.service
systemctl disable openstack-glance-registry.service
systemctl stop openstack-glance-api.service
systemctl stop openstack-glance-registry.service
service openstack-glance-api stop
service openstack-glance-registry stop
```

当所有配置文件都被更新后，我们将重启 Nova 和 Cinder 服务使更改生效。用下列命令来重启服务，当然也可以用脚本来完成（可供下载）。

```
    # Restart Nova services
service openstack-nova-api restart
service openstack-nova-consoleauth restart
service openstack-nova-scheduler restart
service openstack-nova-conductor restart
service openstack-nova-cert restart
service openstack-nova-novncproxy restart

# OR you can also use the script which can be downloaded as part of the helper tools:
~/openstack/commands/nova-restart

# Restart Cinder
service openstack-cinder-api restart
service openstack-cinder-scheduler restart
service openstack-cinder-backup restart
```

```
# OR you can also use the script which can be downloaded as part of the helper tools:
~/openstack/commands/cinder-restart
```

2.6.1.5 排错与高级管理

关键日志目录

组件	关键日志目录
Keystone	/var/log/keystone/keystone.log
Horizon	/var/log/horizon/horizon.log
Nova	/var/log/nova/nova-api.log /var/log/nova/nova-scheduler.log /var/log/nova/nove-compute.log*
Swift	/var/log/swift/swift.log
Cinder	/var/log/cinder/api.log /var/log/cinder/scheduler.log /var/log/cinder/volume.log
Glance	/var/log/glance/api.log /var/log/glance/registry.log
Neutron	/var/log/neutron/server.log /var/log/neutron/dhcp-agent.log* /var/log/neutron/l3-agent.log* /var/log/neutron/metadata-agent.log* /var/log/neutron/openvswitch-agent.log*

标星号的日志只存在于 Acropolis OVM 中。

专家提示

如果一个服务在 OpenStack 管理器（图形界面或是命令行）被发现是 down 的状态，请检查 NTP 设置，即使它在 OVM 中是运行着的。许多服务对 OpenStack 控制器和 Acropolis OVM 间的时间同步有要求。

命令参考

装载 Keystone 源（先于其他命令执行）

```
source keystonerc_admin
```

列出 Keystone 服务

```
keystone service-list
```

列出 Keystone endpoints

```
keystone endpoint-list
```

创建 Keystone endpoint

```
keystone endpoint-create \
--service-id=<SERVICE_ID> \
--publicurl=http://<IP:PORT> \
--internalurl=http://<IP:PORT> \
--region=<REGION_NAME> \
--adminurl=http://<IP:PORT>
```

列出 Nova 实例

```
nova list
```

显示实例细节

```
nova show <INSTANCE_NAME>
```

列出 Nova hypervisor 主机

```
nova hypervisor-list
```

显示 Nova hypervisor 主机细节

```
nova hypervisor-show <HOST_ID>
```

列出 Glance 镜像

```
glance image-list
```

显示 Glance 镜像细节

```
glance image-show <IMAGE_ID>
```

3 第三部分： Acropolis

a·crop·o·lis - /ə ' kräpəlis/ - 名词 – 数据平面存储，计算和虚拟化平台

3.1 架构

Acropolis 是一个分布式的多资源管理器，集协同管理和数据平台功能于一身。它可以被细分为如下三个主要组件：

- 分布式存储架构 (DSF)
  - 这是 Nutanix 核心的赖以生存的组件，其基于 Nutanix 分布式文件系统（HDFS）扩展而来。NDFS 现如今已从一个分布式系统存储资源资源池进化成一个更大功能更强的存储平台。
- 应用移动性架构 (AMF)
  - 类似于 Hypervisor 把操作系统从硬件剥离而来，AMF 把工作负载（虚机、存储和容器等）从 Hypervisor 抽象剥离开。这使我们能在不同的 Hypervisor 之间切换和移动工作负载。
- 虚拟化管理器（AHV）
  - 一个基于 CentOS KVM hypervisor 的多用途虚拟化管理器组件。

Nutanix 努力的方向是一切皆是分布式的，我们正把它延伸到虚拟化和资源管理领域。Acropolis 是一个面向负载和资源管理、资源划分和系统操作的后台服务，其目标是把易耗的资源(如 hypervisor, 预置平台和云等)从运行的负载中剥离，即平台与应用的解耦，但同时又能提供一致性可操作的平台。

这赋予了工作负载一种可在 hypervisor、云提供商和平台之间无缝迁移的能力。

下图以概要的方式展示了 Acropolis 不同层次的结构和关系：

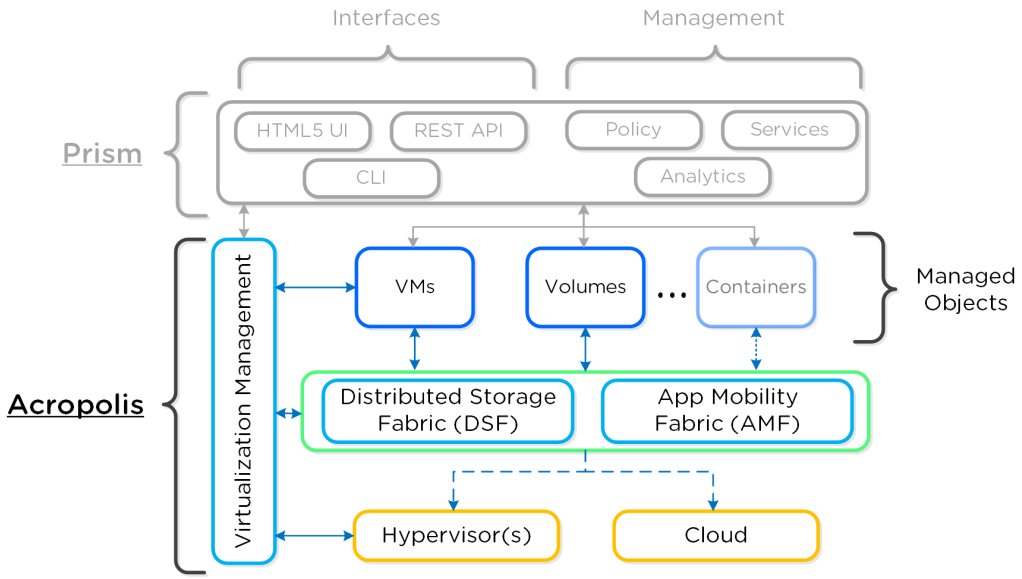


图 10-1. High-level Acropolis Architecture

用于虚机管理的 Hypervisor

当前，只有 AHV 才能对虚机进行全面的管理，将来会拓展到其他 Hypervisor。不过，即使是其他的 Hypervisor，也可以进行卷 API 和状态读取的操作。

3.1.1 融合平台

你可以观看以下视频帮助理解 https://youtu.be/OPYA5-V0yRo

Nutanix 解决方案是一个融合了存储和计算资源于一体的解决方案。它利用本地资源/组件来为虚拟化构建一个分布式的平台，亦称作虚拟计算平台。该方案是一个软硬件一体化平台，在 2U 空间中提供 2 或 4 个节点。

每个节点运行业界标准的 hypervisor（ESXi, KVM, Hyper-V）和 Nutanix 控制器虚机（CVM）。Nutanix CVM 中运行着 Nutanix 核心软件，服务于所有虚机和虚机对应的 I/O 操作。得益于 Intel VT-d（VM 直接通路）技术，对于运行着 VMware vSphere 的 Nutanix 单元，SCSI 控制（管理 SSD 和 HDD 设备）被直接传递到 CVM。

下图提供了一个例子，解释了典型的节点逻辑架构：

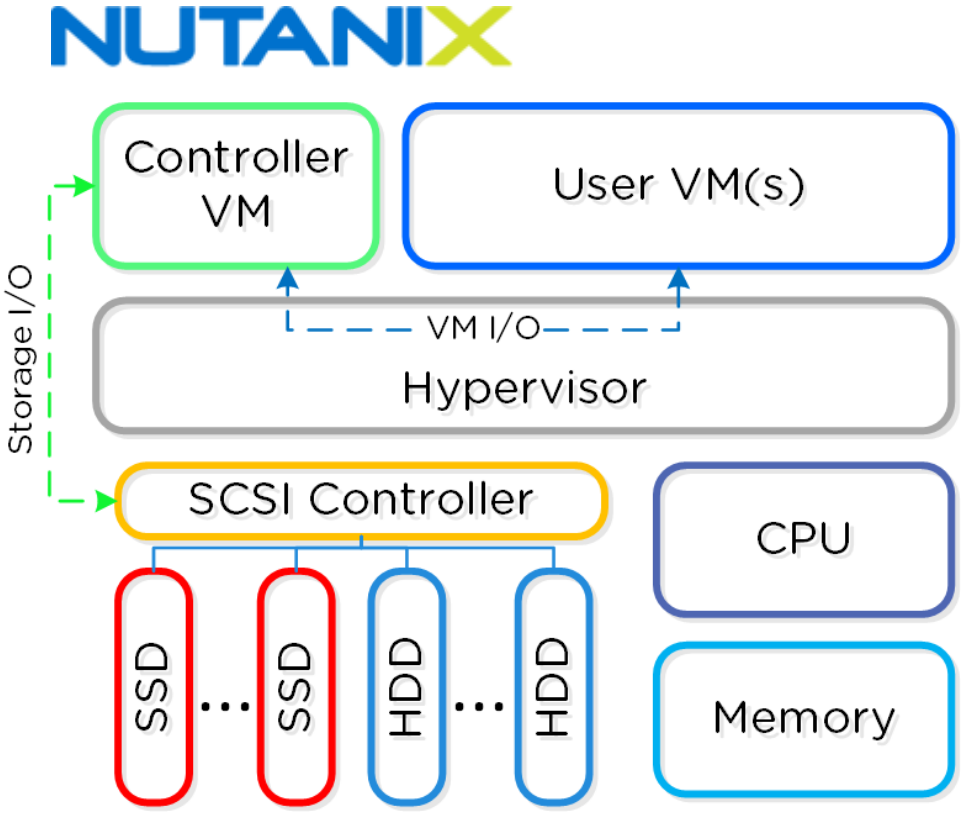


图 10-3. Converged Platform

3.1.2 软件定义

如之前所提到的，Nutanix 平台是一个基于软件的而以软硬件一体方式交付的解决方案。控制器虚机（CVM）中实现了 Nutanix 软件绝大多数功能与逻辑，从一开始就被设计成可扩展并支持插件的架构。软件定义而非依赖于硬件加速的一个关键着眼点在于扩展性。在任何一个产品生命周期中，都可以做一些改进个新功能。

由于不依赖于定制化的 ASIC/FPGA 或硬件能力，Nutanix 以软件更新的方式即可开发和部署新功能。这意味着可以通过软件升级来获得新功能（如去重）。这也让那些老型号的设备可以支持新型号上推出的功能。比如说，你正在一款老的 2400 机型上运行着工作负载，而它没有提供数据去重的功能。你发觉你能从这项功能（去重）中大大受益，因此就在不中断业务的情况下进行在线升级。马上你就有了这项数据去重的能力，如此简单而已。

就像添加新功能，你也可以为 DSF 创建新的适配器或接口。当你刚拿到产品时，只能在 hypervisor 层创建 iSCSI 设备，而之后却能创建 NFS 和 SMB 设备了。将来我们支持从不同负载和 hypervisor（HDFS 等）创建新的适配器。再次强调，所有这些都能通过软件升级就能实现。这跟传统的基于硬件的更新或是要另外购买

软件形成极大的反差。Nutanix 把它变得不一样，由于所有功能都通过软件部署，所以它能运行在任何硬件平台和任何 hypervisor。

下图是一个对于软件定义的控制框架的逻辑展现：

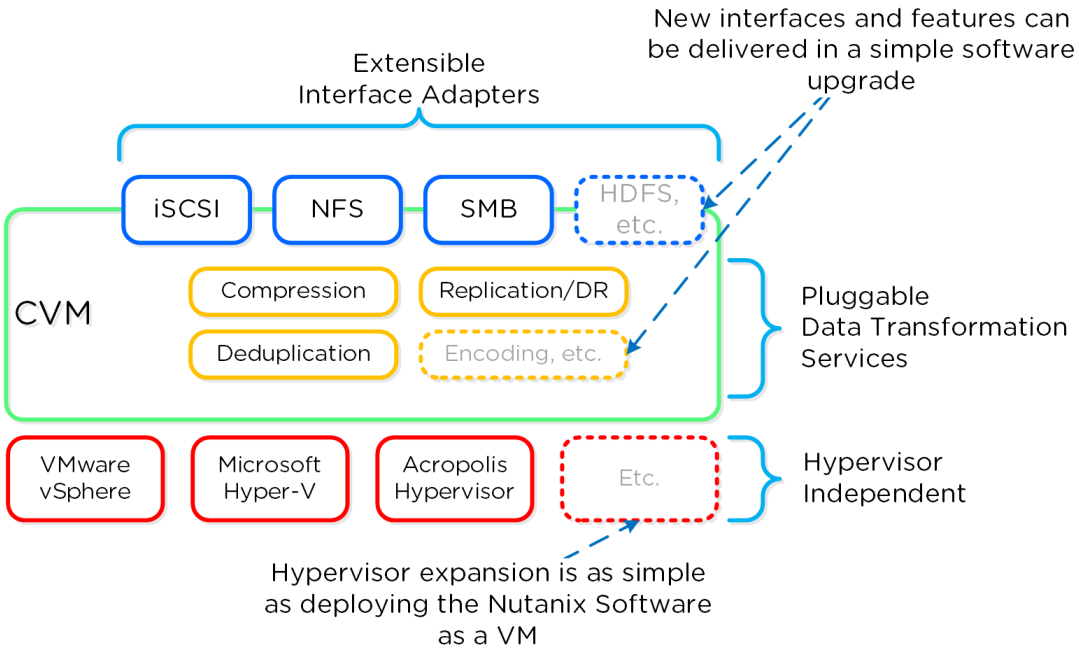


图 10-4. Software-Defined Controller Framework

3.1.3 集群组件

你可以通过观看下面视频来帮助理解： https://youtu.be/3v5RI_IbfV4

Nutanix 平台由下列宏观组件构成：

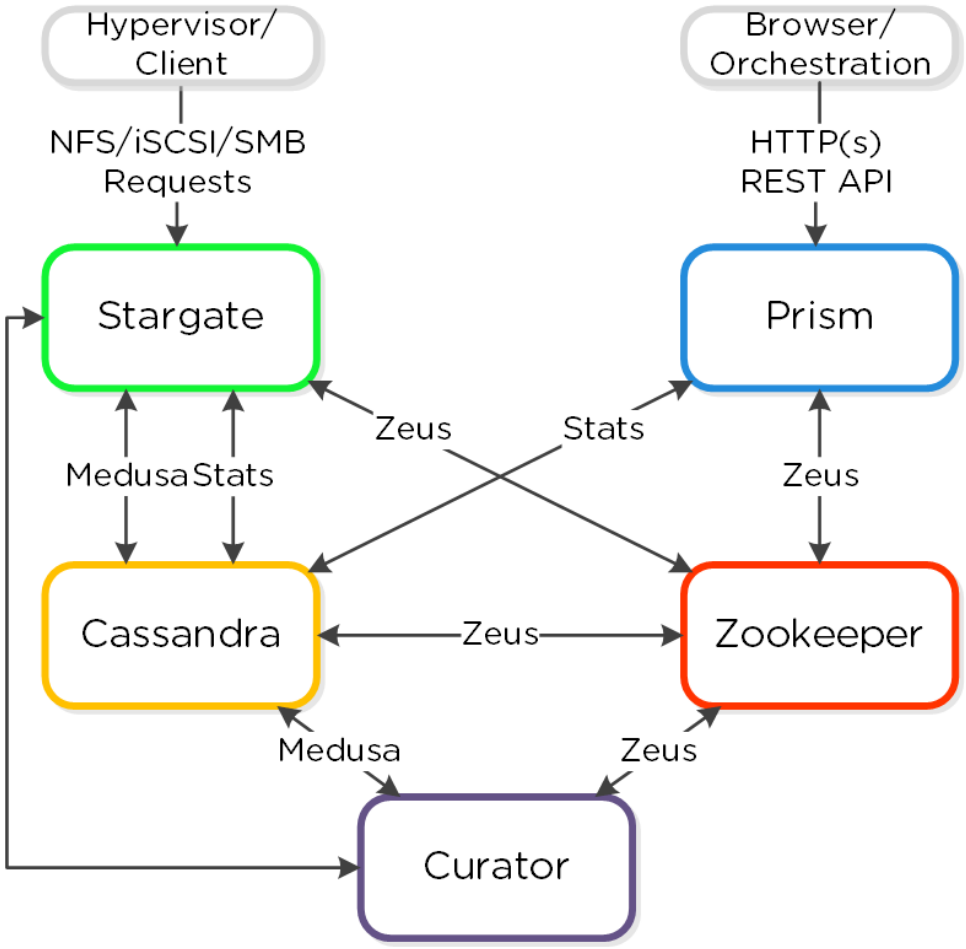


图 10-5. Nutanix Cluster Components

Cassandra

- 关键角色: 分布式元数据存储
- 描述：Cassandra 基于重度修改过的 Apache Cassandra，以分布式环的方式存放和管理所有的集群元数据。Paxos 算法被用来保证严密的一致性。在集群中所有节点上都运行着这个服务。Cassandra 通过一个叫做 Medusa 的协议来访问。

Zookeeper

- 关键角色: 集群配置管理
- 描述：基于 Apache Zookeeper 实现，Zookeeper 存放了所有的集群配置信息，包括主机、IP 地址和状态等。集群中有三个节点会运行此服务，其中的一个被选举成 leader。Leader 接收所有请求并转发到它的组员。一旦 leader 失去了反应，新的 leader 会被自动选举出来。Zookeeper 通过称作 Zeus 的接口来访问。

Stargate

- 关键角色: 数据 I/O 管理
- 描述：Stargate 负责所有的数据管理和 I/O 操作，是 hypervisor 主要的接口（通过 NFS、iSCSI 或 SMB）。为了提供本地 I/O 操作的能力，集群中所有节点都运行此服务。

Curator

- 关键角色：以 Mapreduce 方式管理和清理集群
- 描述：Curator 负责在整个集群间分配和调度任务，诸如磁盘容量平衡、预清理等。Curator 运行在所有节点上，受控于主 Curator（其负责任务委托）。Curator 有两种扫描类型，每 6 小时一次全扫描和每 1 小时一次的部分扫描。

Prism

- 关键角色：用户界面和 API
- 描述：Prism 是一个组件管理网关，它让管理员能配置和监控 Nutanix 集群。它提供多种管理手段，如 Ncli、HTML5 UI 和 REST API。Prism 运行在集群中的每个节点，如同集群中其他组件一样也采用 leader 选举制。

Genesis

- 关键角色：集群组件和服务管理
- 描述：Genesis 是一个负责配置初始化和服务交互的进程，运行在每个节点上。Genesis 不依赖于集群，即不管集群是否配置或运行与否，它都运行着。它唯一的前提是 Zookeeper 必须起来并运行着。Cluster_init 和 cluster_status 这两个页面所显示的信息由 Genesis 进程提供。

Chronos

- 关键角色：任务调度
- 描述：Chronos 负责把由 Curator 扫描产生的任务在节点间调度执行并合理分配。Chronos 运行在每个节点上，受控于主 Chronos（负责任务委托且和主 Curator 运行在同一节点）。

Cerebro

- 关键角色：数据复制和容灾管理
- 描述：Cerebro 负责 DSF 中的数据复制和容灾管理部分，包含快照的调度、远程站点的数据同步及站点的迁移和故障切换。Cerebro 运行在 Nutanix 集群的每个节点上，并且每个节点都参与远程站点/集群的数据同步。

Pithos

- 关键角色：vDisk 配置管理
- 描述：Pithos 负责 vDisk（DSF 文件）的配置数据。Pithos 构建于 Cassandra 之上，并运行在每个节点。

3.1.4 Acropolis 服务

集群内的每个 CVM 上都会运行一个 Acropolis 服务，其中一个会被选举为首选 Acropolis，负责任务调度，任务执行，IPAM（IP 地址管理）等，其余的均为从属 Acropolis，类似于其他具有首选节点的组件，当集群中首选 Acropolis 发生故障时，就会在余下的 Acropolis 服务中选举出一个新的首选 Acropolis。

Acropolis 服务的角色分类如下：

首选 Acropolis

任务调度&执行

统计信息收集/发布

网络控制器（支持 Hypervisor）

VNC 代理（支持 Hypervisor）

HA（支持 Hypervisor）

从属 Acropolis

统计信息收集/发布

VNC 代理（支持 Hypervisor）

下图展示了首选 Acropolis 与从属 Acropolis 之间关系的概念视图：

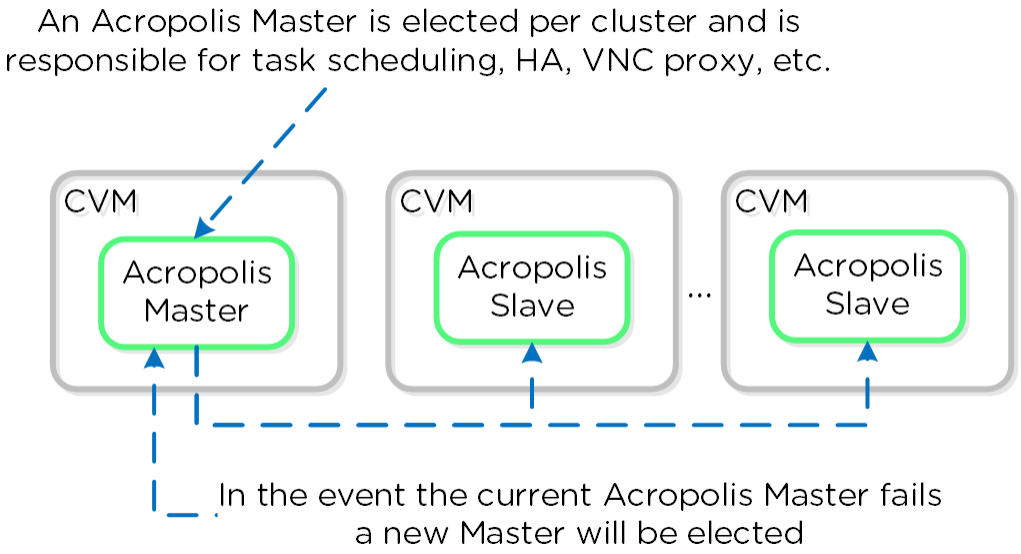


图 10-2. Acropolis 服务

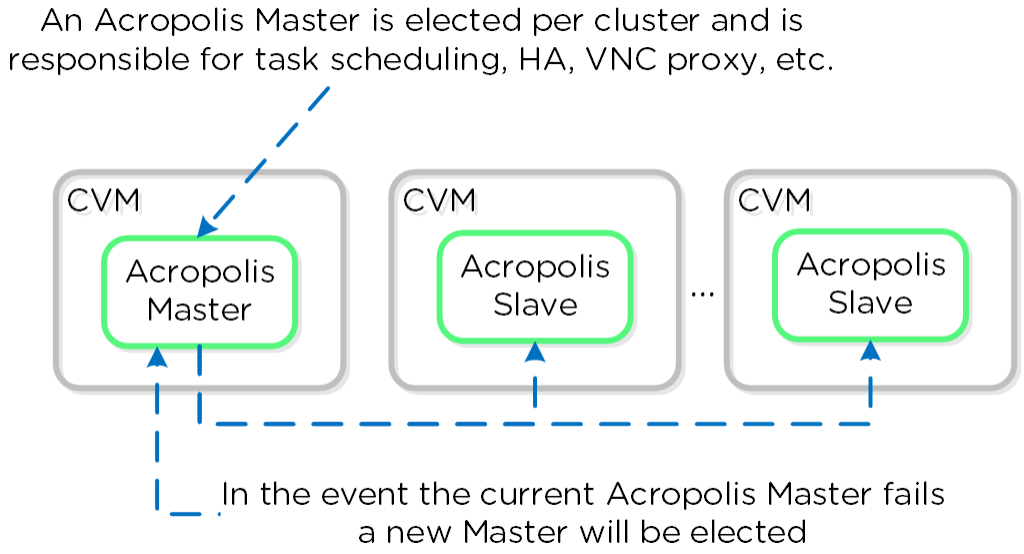


图 10-2. Acropolis Services

3.1.5 驱动器分解

这一章节，我们将涉及到各种存储设备（SSD/HDD）如何在 Nutanix 平台被分解，分区和使用。注：所有容量单位是二进制吉字节(GiB)而不是十进制吉字节(GB)，同时把驱动器格式化的文件系统和相关的消耗也一并考虑在内。

3.1.5.1 SSD 设备

SSD 设备存储了一些关键的组件，详细介绍请参照之前章节：

- Nutanix 主目录（CVM 核心）
- Cassandra （元数据 存储）
- OpLog （持久写缓冲）
- 内容缓存 （SSD 缓存）
- 扩展存储 （持久存储）

下图展示了一个 Nutanix 节点 SSD 的存储分解例子：

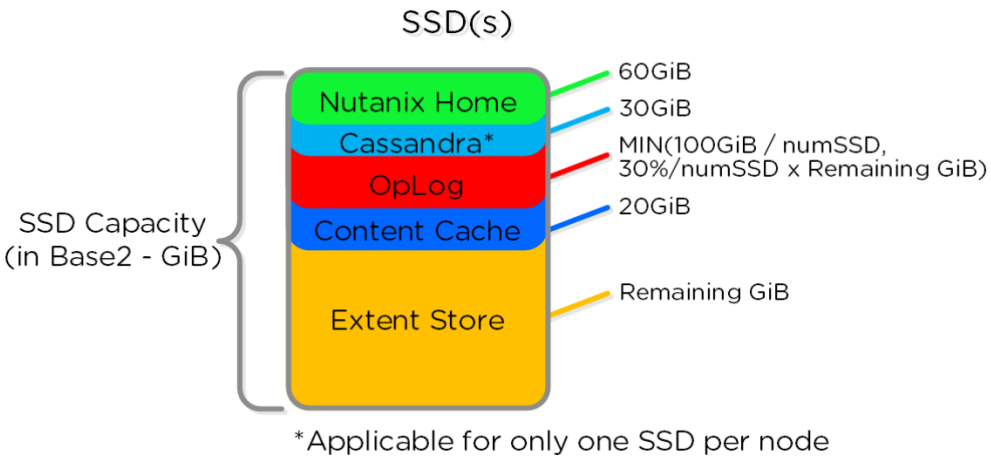


图 10-6 SSD Drive Breakdown

注：OpLog 的大小是动态的（如 4.0.1 版本），允许扩展存储部分动态增长。数值是基于假设 OpLog 完全被使用。上图中，图形和份额不是按比例绘制的。当评估剩余 GiB 容量时，采用自顶而下的方法。例如，用于 OpLog 计算的剩余容量 (GiB) 应该是 SSD 被格式化后的容量减去 Nutanix 主目录和 Cassandra 占用的容量。

大部分型号配备 1 或者 2 个 SSD，然而同样适用于配备更多 SSD 的型号。例如，3060 或者 6060 节点有 2*400GB 的 SSD，每个节点将会有 100GiB 的 OpLog，40GiB 的内容缓存，和大约 440GiB 的 SSD 扩展存储。

3.1.5.2 HDD 设备

由于 HDD 设备主要作用于大容量存储，分解起来非常简单：

- Curator 保留（Curator 存储）
- 扩展存储（持久存储）

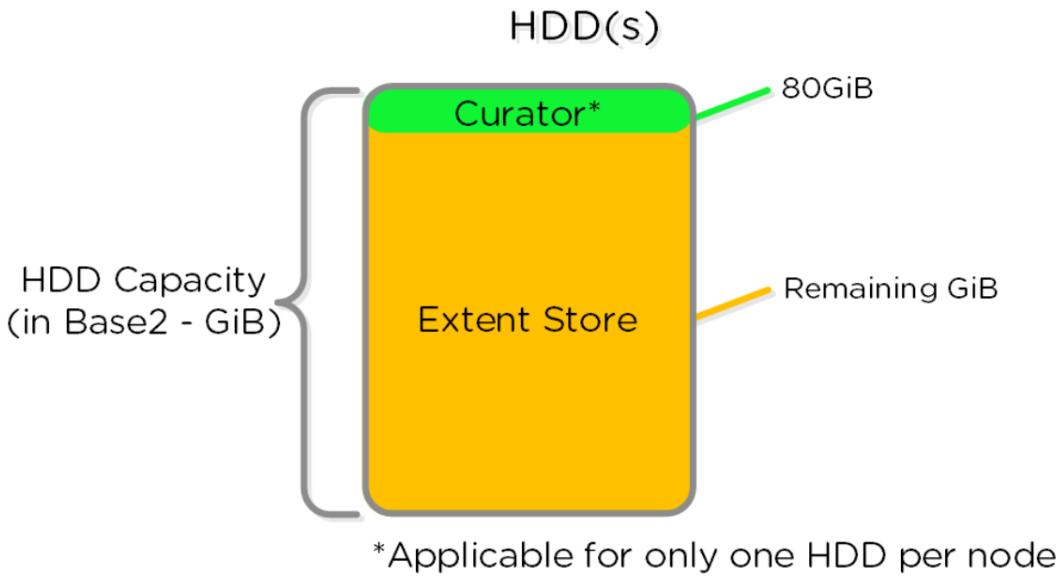


图 10-7 HDD Drive Breakdown

例如，3060 节点有 4*1TB HDD，每个节点会有 80GiB 作为 Curator 保留和大约 3.4 TiB 的扩展存储硬盘容量。

注：上述数值是基于 4.0.1 版本的，不同版本可能有变化。

3.2 分布式存储结构

一组 Nutanix 节点一起构成一个称为 Acropolis 分布式存储结构（DSF）的分布式平台。DSF 呈现给 Hypervisor 就像任何集中存储阵列，然而所有的 I/O 是在本地处理从而提供更高的性能。这些节点如何构成分布式系统的更详细内容会在下面章节里介绍。

下图展示了 Nutanix 节点如何构成 DSF 的一个例子：

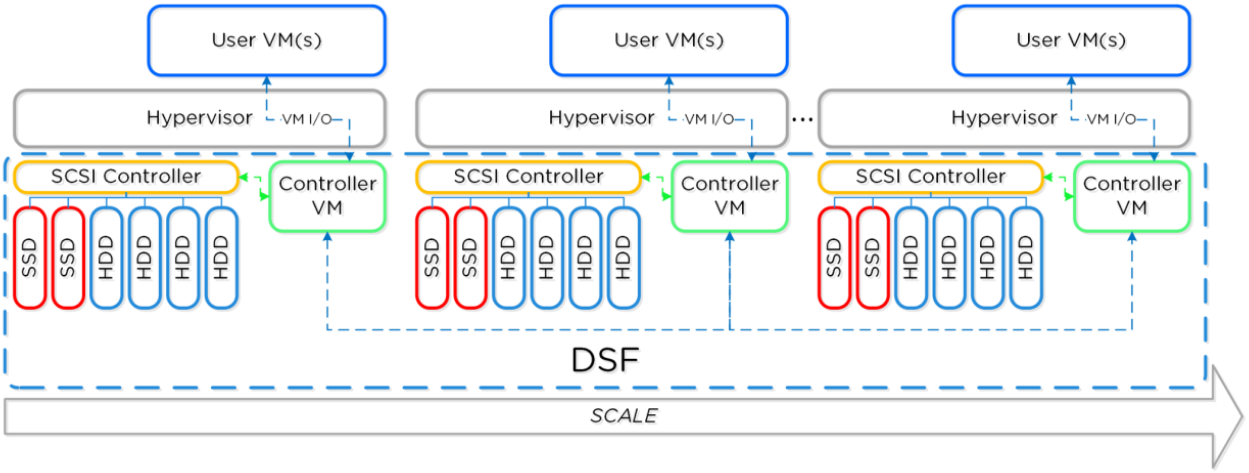


图 11-1 Distributed Storage Fabric Overview

3.2.1 数据结构

Acropolis 分布式存储结构的高层组件如下：

3.2.1.1 存储池

角色：一组物理存储设备

描述：一个存储池是一组物理存储设备，包括集群内所有节点的 PCIe SSD、SSD 和 HDD。存储池可以跨多个 Nutanix 节点并且随着集群的扩展而扩展。在大部分情况下，建议单个集群配置一个存储池。

3.2.1.2 容器

角色：一组虚拟机或者文件的逻辑分组

描述：容器（container）从逻辑上划分存储池，并包含一组虚拟机或者文件（即虚拟磁盘）。一些配置项（例如 RF）是在容器层实现的，然后应用在单个虚拟机文件层面。Container 和 Datastore 是一一对应的（在 NFS、SMB 场景中）。

3.2.1.3 虚拟磁盘

角色：虚拟磁盘文件

描述：一个虚拟磁盘文件(vDisk)是 DSF 上任意一个大于 512KB 的文件（包括 VMDK 和虚拟机硬盘）。vDisk 由 extent 组成，它被分组并保存在磁盘上作为 extend group。

下图显示了各组件在 DSF 与 Hypervisor 之间的对应关系：

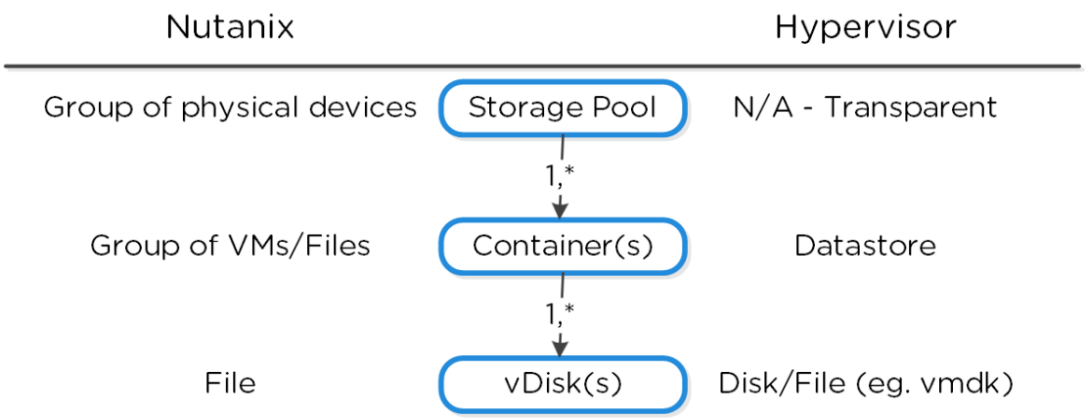


图 11-2 High-level Filesystem Breakdown

3.2.1.4 Extend

角色：逻辑上连续的数据块

描述：一个 Extent 是逻辑上连续的 1MB 大小的数据块，它由 n 个连续的 block 组成（block 是可变的，取决于不同操作系统）。Extent 的读写修改是基于更小的子块（也称为 slice）以保证可粒度和有效性。当一个 extent 的 slice 被读入 cache 时，可以被修剪（trimmed），这依赖于被读取的数据量大小。

3.2.1.5 Extend 组

角色：物理上连续的数据块

描述：一个 Extent Group 是物理上连续存储的 1MB 或者 4MB 大小的数据块。它以文件的形式由 CVM 管理并保存在磁盘上。Extent 被动态分布在多个 Extent

组中，提供数据跨节点和磁盘的条带功能用来提高 IO 性能。注：在 NOS4.0 中，Extent 组可以是 1MB 或者 4MB，这取决于消重功能。

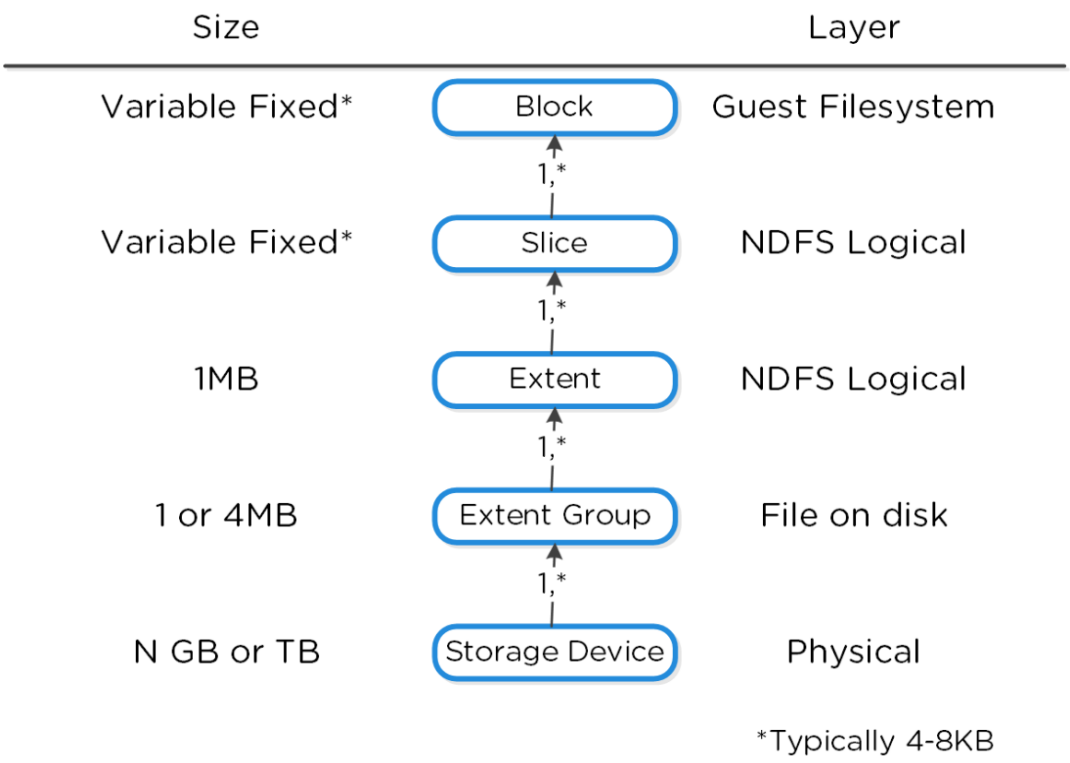


图 11-3 Low-level Filesystem Breakdown

这是另外一个表明文件系统各部分关系的示意图：

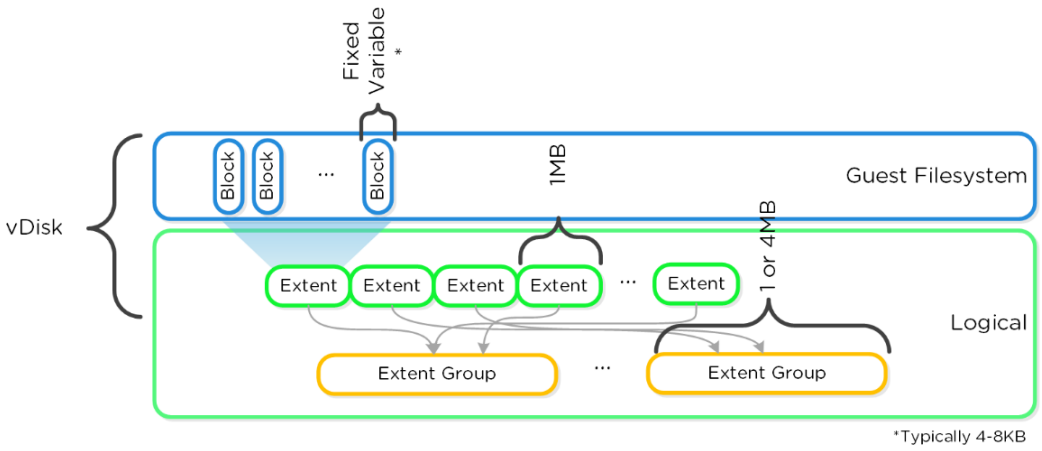


图 11-4 Graphical Filesystem Breakdown

3.2.2 I/O 路径和缓存

你可以通过观看下面视频来帮助理解： https://youtu.be/SULqVPVXefY

Nutanix 的 IO 路径由以下一些组件组成：

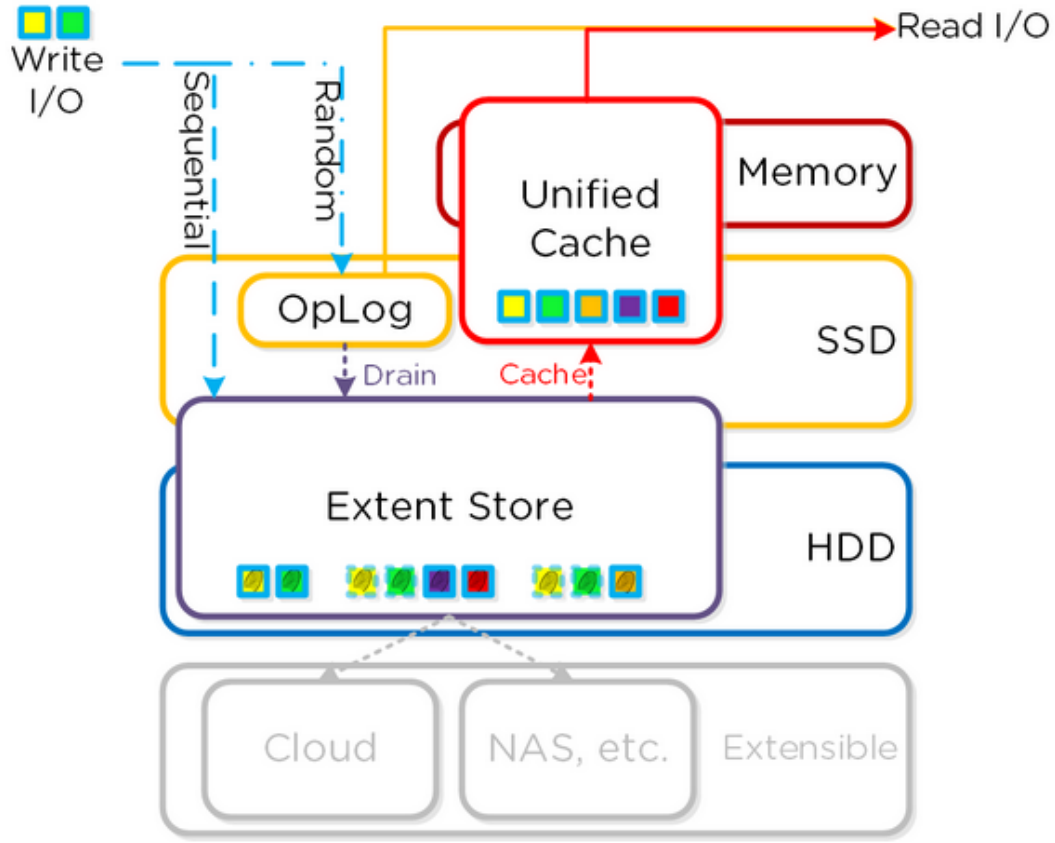


图 11-5 DSF I/O 路径

3.2.2.1 **Oplog**

角色：持久性写缓存

描述：Oplog 类似于文件系统的日志（journal），用来处理突发的写操作，并聚合这些写操作顺序地写到 Extent Store 中。为了保证数据高可用性的要求，在写操作完成（Ack）之前，数据写入 Oplog 的同时被同步复制到另一个 CVM 的

Oplog 中。所有 CVM 的 Oplog 都参与复制操作，并依据 CVM 负载情况自动选择。Oplog 保存在 CVM 的 SSD 层以提供极高的 IO 写性能，特别是随机 IO 写操作。对于顺序的 IO 写操作会直接写到 Extent Store 上而绕过 Oplog。如果数据当前在 Oplog 中，所有的读请求会直接从 Oplog 中反馈，直到数据被合并推送到 Extent Store 中后，只有当读的数据不在 oplog 中,读请求才会从 Extent Store/Unified Cache 中获得。如果在 Container 中的消重功能被启用时，则所有写 IO 操作时，数据块会被标记指纹，以便在数据进入 Unified Cache 时进行消重操作。

3.2.2.2 **Extent Store**

角色：持久性数据存储

描述：Extent Store 是 DSF 中持久性大容量存储组件，它横跨 SSD 和 HDD，并且能扩展到其他节点的存储设备上。有两种数据流进入 Extent Store， 一种是从 Oplog 中被合并的数据顺序写入到 Extent Store，另外一中是绕过 Oplog 的顺序写操作直接写入 Extent Store 中。Nutanix 的 ILM（information lifecycle management）基于 IO 类型（IO Pattern）动态决定数据保存的热分层位置，并且在不同热分层之间移动数据。

3.2.2.3 **Unified Cache**

角色：动态读缓存

描述:Unified Cache 是可消重的读缓存，它横跨 CVM 的内存和 SSD。当读取的数据不在缓存中（或基于一个特定的指纹），数据会放到 Unified Cache 的 Single-touch 池，该池完全保留在内存中，并且使用 LRU(最近最少使用算法)直到数据被弹出。如果后续有对该数据的读取操作，数据会被“移动”（其实没有实际的数据移动，移动的只是数据的元数据（metadata）到 Multi-touch 池（Multi-toiuch 池跨内存和 SSD）中的内存段。这时开始，数据块具备 2 个 LRU，一个是其位于内存段的 LRU，如果 LRU 耗尽，则数据被移动到 Multi-touch 池的 SSD 段，并被赋予一个新的 LRU。如果数据再次被读取，则数据被移动到 Multi-touch 池的顶端，

并重置其位于内存段的 LRU。指纹标记可以在 Containter 层面通过管理 UI 进行配置。默认指纹标记被禁用。

下图是 Unified Cache 中逻辑描述图：

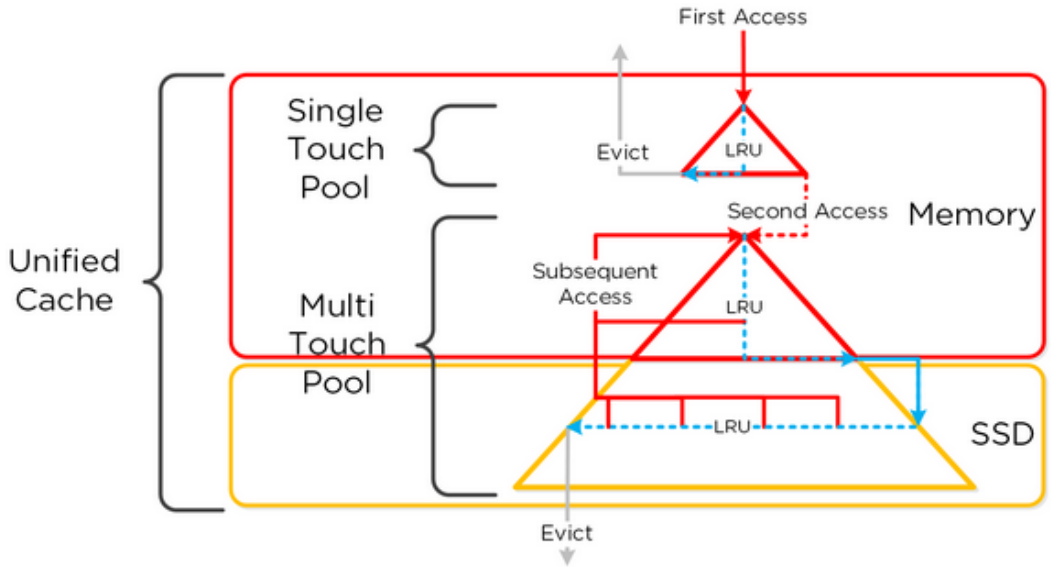


图 11-6 DSF Unified Cache

缓存的粒度和逻辑：

数据是以 4K 粒度读进缓存的，所有缓存是实时完成的，没有延迟，也没有采用批处理方式把数据读进缓存中。

3.2.2.4 **Extent Cache**

角色： 内存中的读缓存

描述：Extent Cache 是完全位于 CVM 内存中的读缓存。当指纹标记和消重被禁用时，它用来存储没有被标记指纹的 Extents。在 3.5 版本中 Extent Cache 和 content Cache 是相互独立的，但在 4.5 版本这两个 Cache 被合并成统一的 Cache。

3.2.3 可扩展的元数据

你可以通过观看下面视频来帮助理解：https://youtu.be/MlQczJhQI3U

元数据（Metadata）是任何智能系统的核心，对于文件系统和存储阵列来说是至关重要的。在 DSF 中，我们使用了一些关键技术来确保数据是 100%正确（即“强一致性”），并且保证 DSF 扩展到超大规模数据量时依然可靠。在上述 Architecture 章节中提到，DSF 使用一种“环状”的 Key-Value 结构的分布式数据库来保存重要的元数据和其它平台数据（例如 stats 等）。为了确保元数据的可用性和冗余度，也同样引入了复制因子（RF）。一旦一条元数据(Metadata)被写或者更新后，这条记录将同时写到“环”中的另一个节点，然后被复制到 n 个其他节点（n 决定与集群的大小）。集群中大多数（majority）节点同意之前确保所有信息一致，这就是强一致性的 Paxos 算法。这确保了 Nutanix 平台数据的“强一致性”以及元数据作为平台的一部分进行存储。

下图显示了在一个 4 节点集群中，元数据的插入和更新操作：

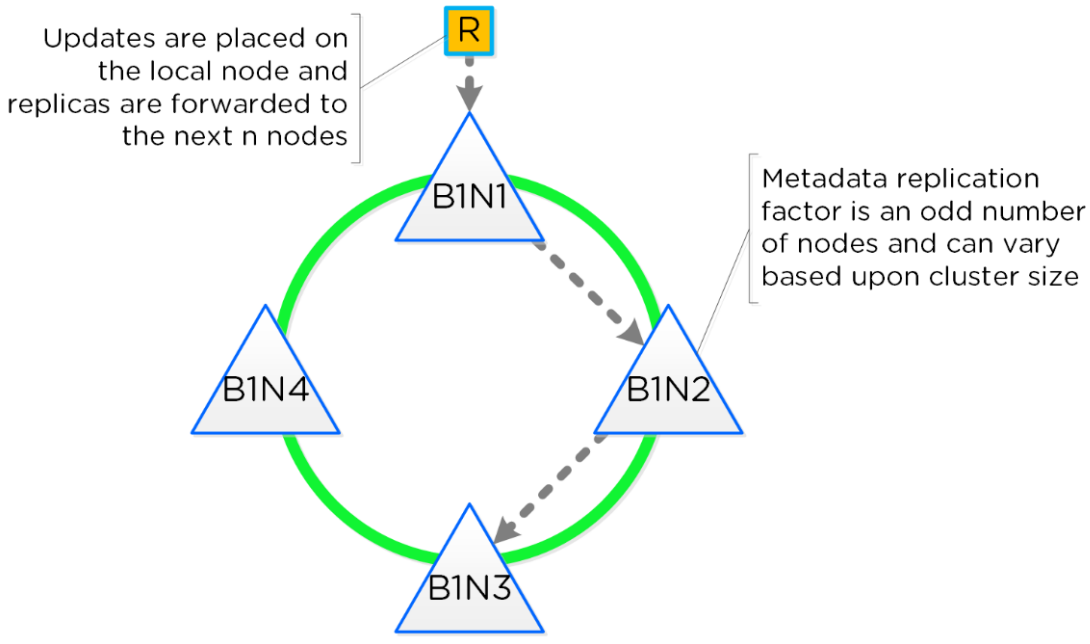


图 11-8 Cassandra Ring Structure

对于 DSF 的元数据数据库，集群扩展时的性能也是至关重要的。与传统的“双控”和主备模式不同，每个 Nutanix 节点只负责整个集群元数据中的一部分。这种

方式消除了传统的性能瓶颈问题，并且允许元数据被集群中所有节点共同维护。并且使用“一致性散列算法（Consistent Hashing）”来保证当节点数量变化时，需要被 remapping 的元数据量最少。当节点数量从 4 增加到 8 个时，新节点将被插入到环中各个老节点之间，使用类似的 block 感知能力提供可靠性。

下图显示了环被扩展时的情况：

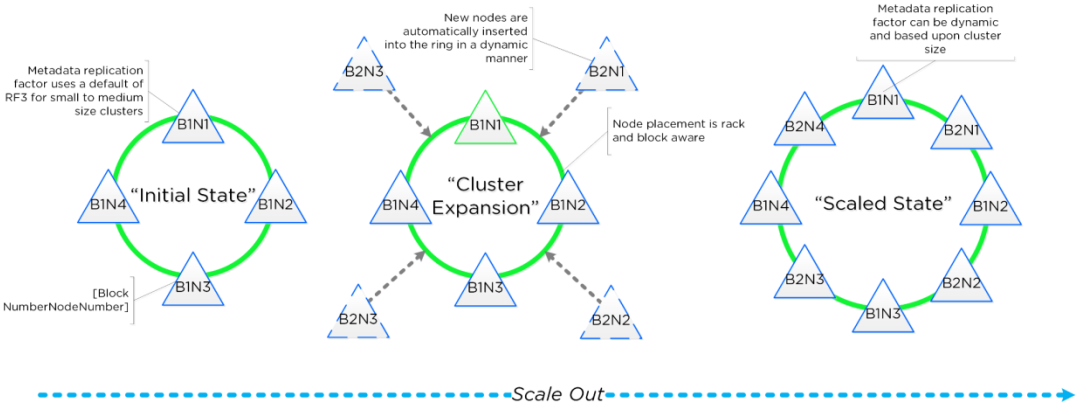


图 11-9 Cassandra Scale Out

3.2.4 数据保护

你可以通过观看下面视频来帮助理解： https://youtu.be/OWhdo81yTpk

Nutanix 平台使用复制因子（RF - Replication Factor/Resillience Factor）和校验和（Checksum）来保证当节点或者磁盘失效时，数据的冗余度和可用性。按照上面的解释，Oplog 作为暂存区来接受所有写操作，并保存到低延迟的 SSD 层上。当数据写入本地 Oplog 时，数据被“同步”复制到另 1 个或者 2 个 Nutanix CVM 的 Oplog 之中（取决于 RF 设置），当这个操作完成之后，此次写操作才被确认（Ack）。这样能确保数据至少存在于 2 个或者 3 个独立的节点上，保证数据的冗余度。注：当 RF 为 3 时，至少需要 5 个节点，并且元数据（metadata）数据会保留 5 份（RF5）。

用户数据的 RF 是通过 Prism 在 container 层面进行配置。所有节点都参与 Oplog 的复制操作，这样能消除“热点节点”，并保证线性的性能扩展。当数据被写入时，同时计算该数据块的校验和，并且作为数据块元数据中的一部分进行存储。

随后数据块在保证满足 RF 的前提下，被“异步”推送到 Extent Store 中。当发生节点或者磁盘失效，数据块会重新在所有节点间进行复制以满足复制因子的设置。任何时候，读取数据块并同时计算其校验和以确保数据块有效。当数据块检查结果不匹配校验和时，副本数据将会覆盖该无效数据块。

下图显示了这一过程的逻辑图：

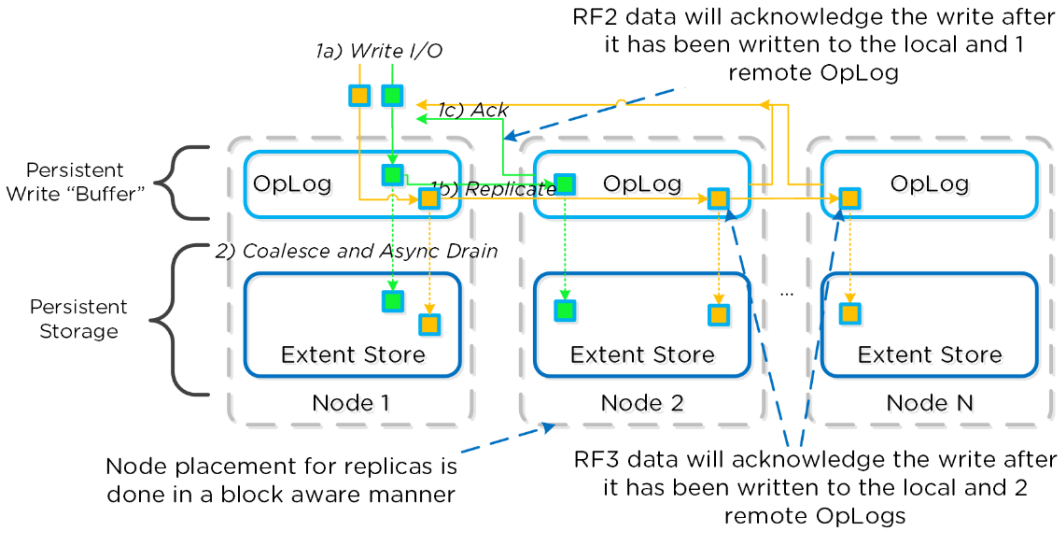


图 11-7 DSF Data Protection

3.2.5 可用域

你可以通过观看下面视频来帮助理解：https://youtu.be/LDaNY9AJDn8

可用域（也称为节点/模块/机架感知）是分布式系统的关键结构，以遵守组件和数据放置的决定。DSF 现在是节点和模块感知的，然而，随着集群规模的增长，需要提升到机架感知。Nutanix 说的模块(block)是指一个机箱，包含一个，两个或者四个服务器节点。注：如果需要实现模块感知，最少需要 3 个模块，否则缺省是节点感知。

建议使用统一的较新的模块来保证模块感知是允许的。通常的场景和使用的感知级别能够在本章末找到。需要 3 个模块是用于确保符合规定要求(quorum)。例如，3450 是一个拥有 4 个节点的模块。把角色和数据分散到不同模块的原因是确

保当一个模块故障或需要维护时，系统可以不中断地持续运行。注：在一个模块里，冗余的电源和风扇是仅有的共享组件。

感知可以应用到下面这些关键的方面：

- 数据（虚拟机数据）
- 元数据（Cassandra）
- 配置数据（Zookeeper）

3.2.5.1 数据

DSF 的数据副本将会被写到集群中的其它模块以保证当一个模块故障或者计划停机时，数据仍然可用。这适用于 RF2 和 RF3，以及在模块故障的场景。一个简单的比较是节点感知，当一个节点故障时，副本将被复制到另外的节点以提供保护。模块感知进一步增强了，当模块故障时提供数据的可用性保证。

下图展示了在一个 3 模块的部署中副本如何放置的例子：

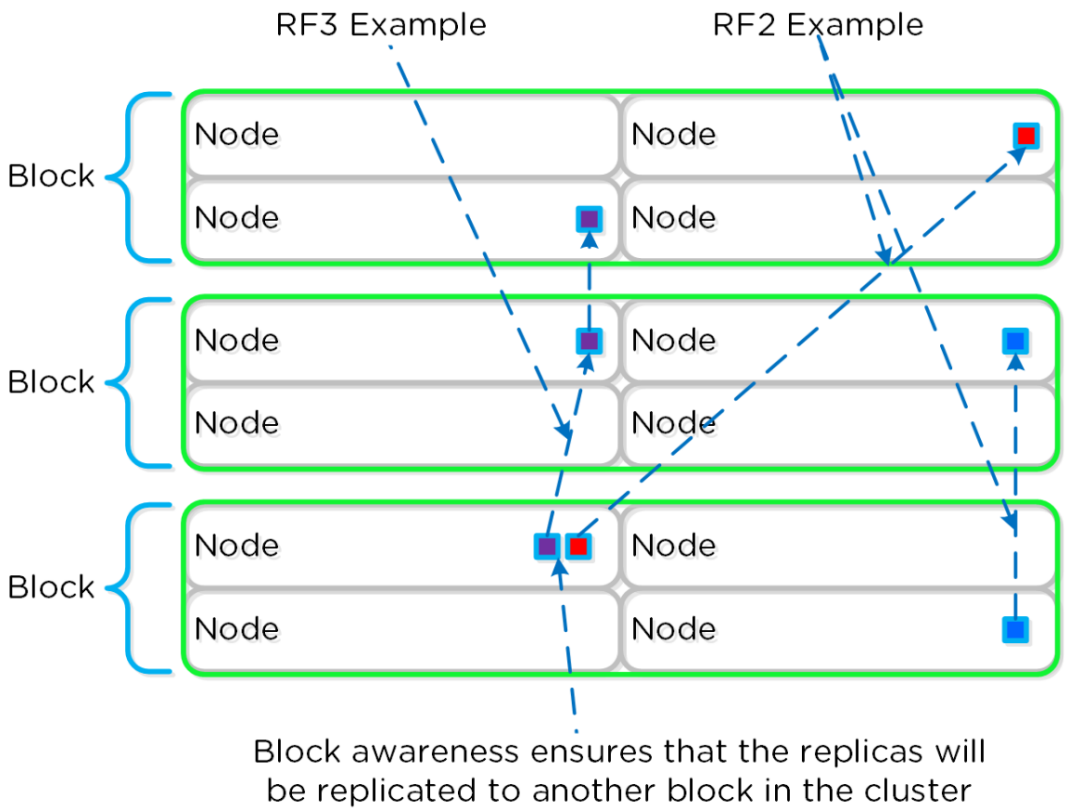


图 11-25 Block Aware Replica Placement

当模块故障时，模块感知将被维持，副本将重新复制到其他模块中：

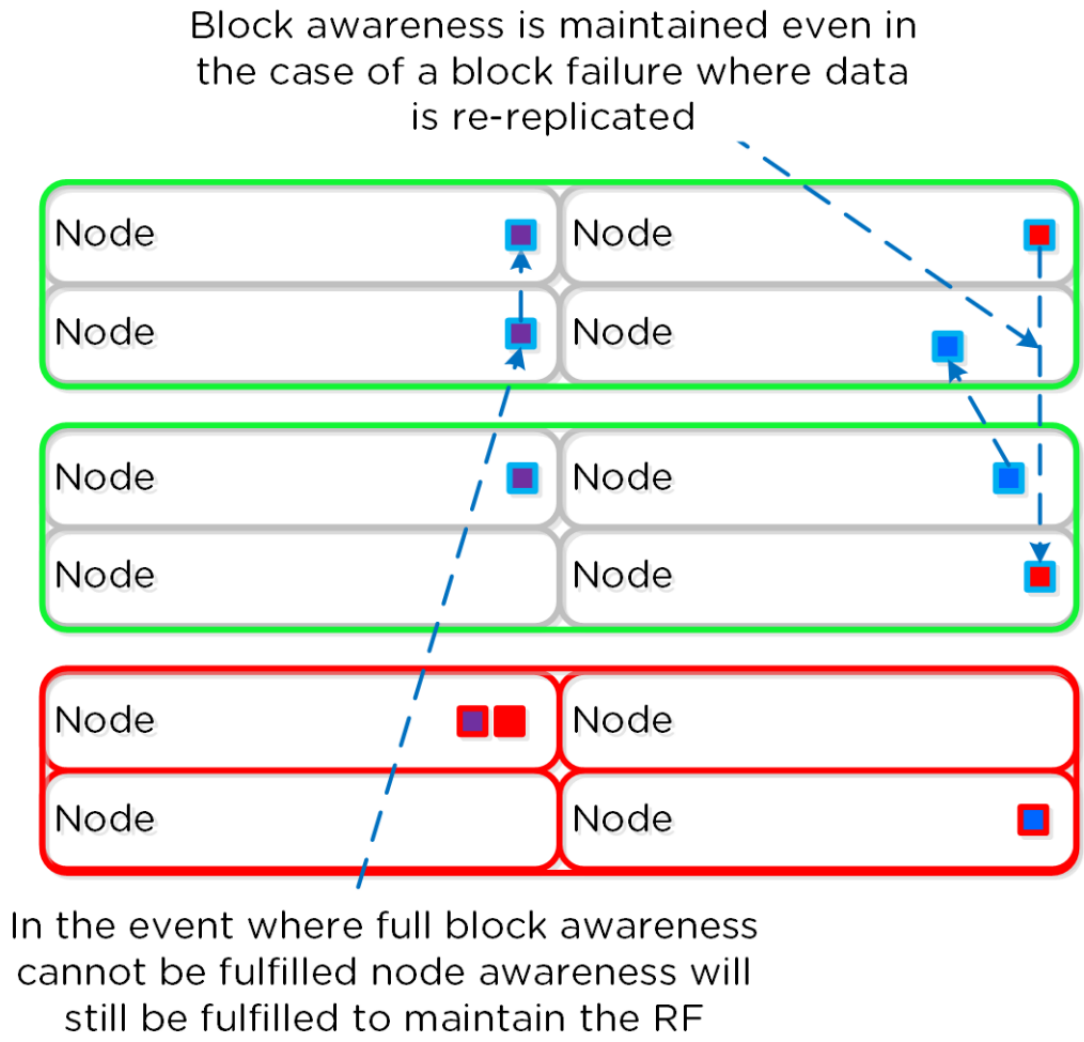


Figure 11-26. Block Failure Replica Placement

3.2.5.2 感知条件和容错

下面我们划分为一些通用的场景以及哪个级别的容错将被使用：

- <3 模块：节点感知
- 3+模块统一的：模块+节点感知
- 3+模块不统一的：
  - 如果 SSD 或者 HDD 层在不同模块间的差别>最大方差：节点（4.5 版本前）或者尽力模块（4.5 版本后）感知
  - 如果 SSD 或者 HDD 层在不同模块间的差别<最大方差：模块+节点感知

3.2.5.3 元数据

正如在可扩展的元数据一章提到的，Nutanix 利用一个深度修改过的 Cassandra 平台来存储元数据和其它重要的信息。Cassandra 利用一个环形架构，复制到环里 n 个对等（peer）来保证数据的一致性和可用性。

下图显示了一个 12 节点集群的 Cassandra 环的例子：

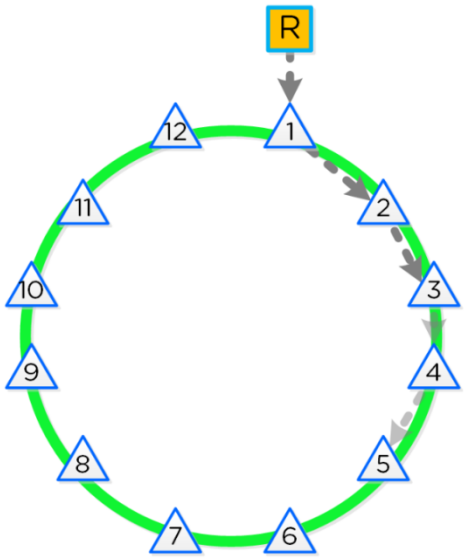


Figure 11-27. 12 Node Cassandra Ring

Cassandra 对等复制在整个环里用顺时钟的方式迭代到各节点。在模块感知下，对等（peer）会分布到模块里，确保没有两个对等会在同一个模块里。

下图显示了一个节点布局按照上述环形转换到模块布局的例子：

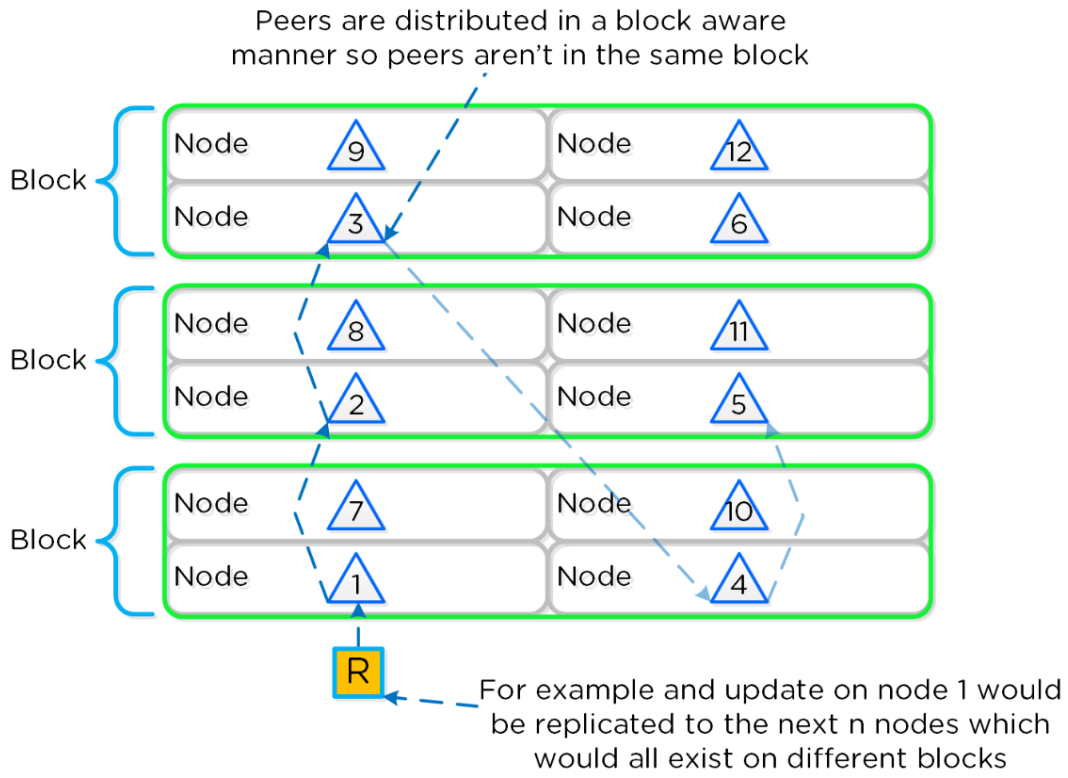


Figure 11-28. Cassandra Node Block Aware Placement

在模块感知特性下，即使发生模块故障的情况下，仍然有至少两份数据副本（元数据 RF3，在大型集群可以利用 RF5）。

下图显示了所有节点复制拓扑形成环的例子

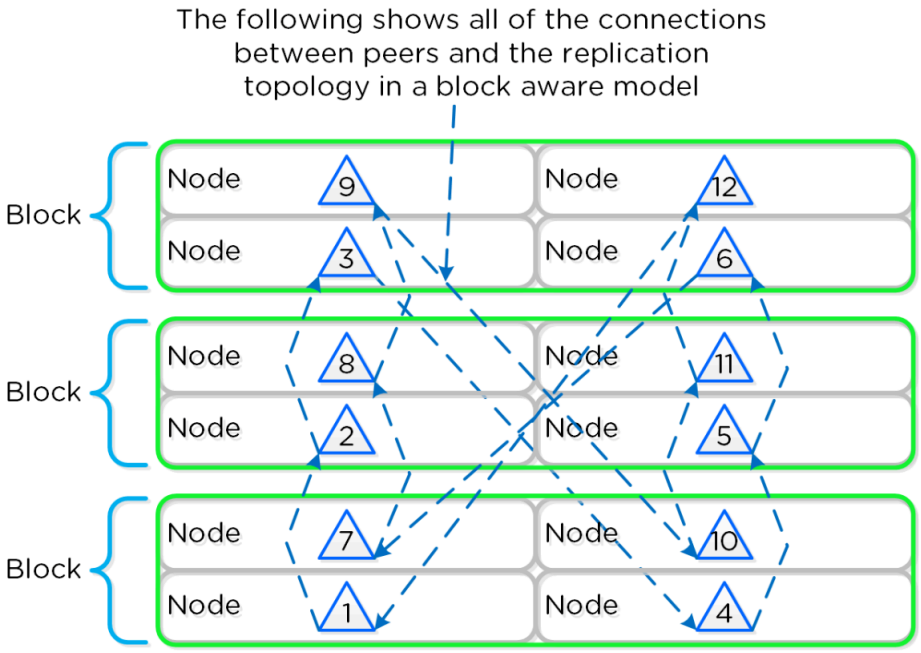


Figure 11-29. Full Cassandra Node Block Aware Placement

元数据感知条件

下面我们看一下一些常见的场景和那个级别的感知会被使用：

- **FT1**（数据 **RF2/**元数据 **RF3**）将会是模块感知，如果：
  - **>3** 模块
  - **X** 代表模块最大节点数，剩余模块的节点数最少有 **2X** 个

  例 1：4 个模块分别有 2，3，4，2 个节点，不会模块感知，因为 2+3+2<2*4

  例 2：4 个模块分别有 3，3，4，3 个节点，模块感知，因为 3+3+3>2*4
- **FT2**（数据 **RF3/**元数据 **RF5**）将会模块感知
  - **>5** 模块
  - **X** 代表模块最大节点数，剩余模块的节点数最少有 **4X** 个

  例 1:6 个模块分别有 2,3,4,2,3,3 个节点，不会模块感知，因为 2+3+2+3+3<4*4

  例 2:6 个模块分别有 2,4,4,4,4,4 个基点，模块感知，因为 2+4+4+4+4>4*4

3.2.5.4 配置数据

Nutanix 利用 Zookeeper 来存储集群里必要的配置数据。这些角色也使用模块感知的方式进行分布以确保模块故障时仍然可用。

下图显示了 3 个 Zookeeper 节点分布在一个模块感知的方式的例子：

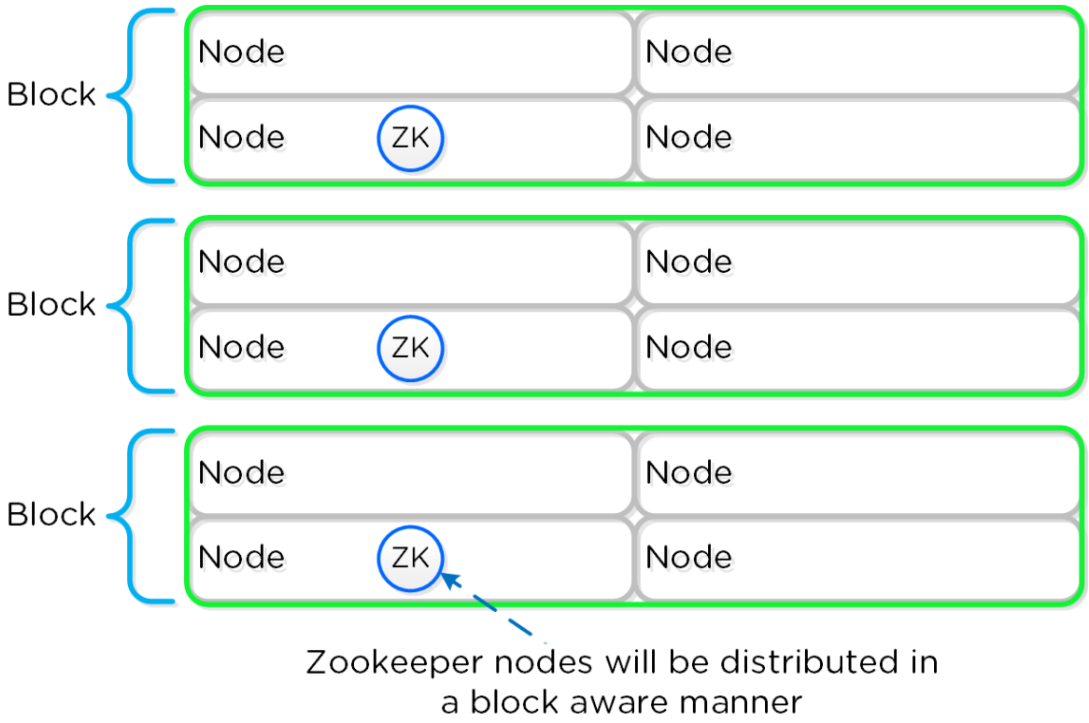


Figure 11-30. Zookeeper Block Aware Placement

在一个模块失效的事件里，意味着其中一个 Zookeeper 节点没有了，ZooKeeper 的角色将会被转移到集群里的另外一个节点上，如下图示：

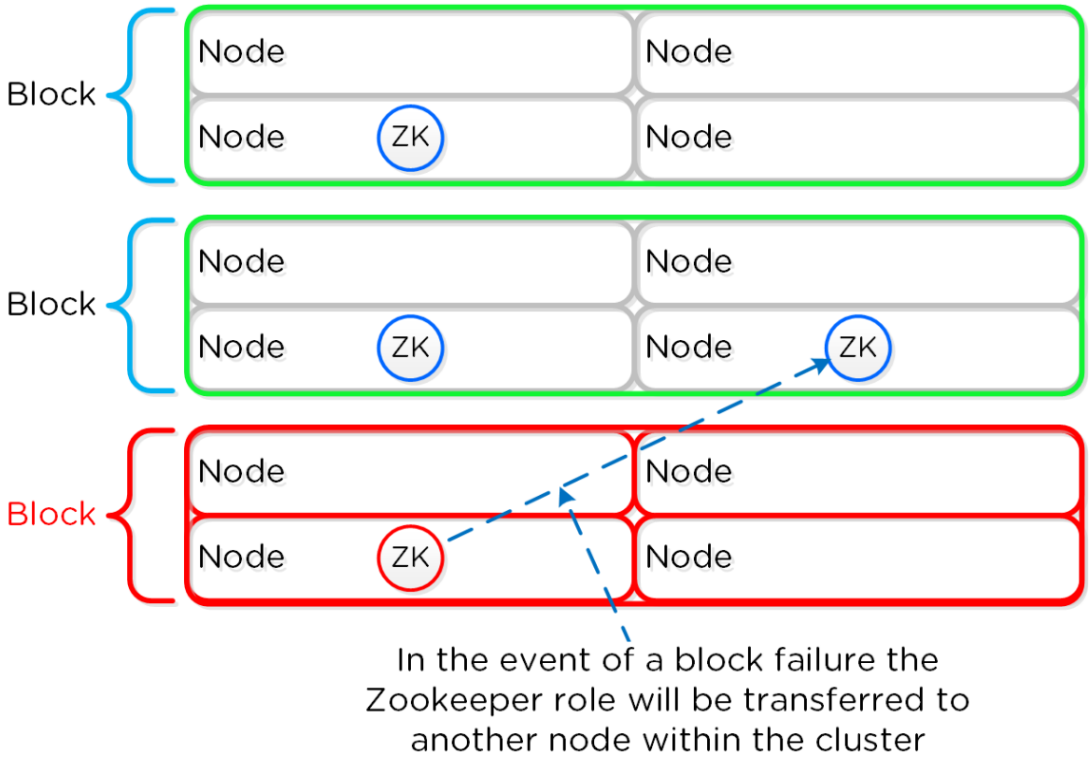


Figure 11-31. Zookeeper Placement Block Failure

当这个模块恢复正常上线后，Zookeeper 角色将会被转移回来，以维持模块感知。

注：4.5 版本前，迁移需要手工方式，不能自动进行。

3.2.6 数据通道弹性

你可以通过观看下面视频来帮助理解：https://youtu.be/SJIb_mTdMPg

可靠性和弹性是 DSF 或者任何主要存储平台设计的关键。传统的架构是基于硬件应该是可靠的思路建设的，相反，Nutanix 使用一个不同的方法，它认为硬件始终会出故障的。基于此，系统设计使用简洁和不中断的方式来处理这些故障。

注：但这并不意味着 Nutanix 的硬件质量不过关，只是概念的转变。Nutanix 硬件和品质部门一直致力于高质量和审核过程。

作为分布式系统，DSF 是基于处理组件、服务和 CVM 故障而构建的，可以分为如下几个级别：

- 磁盘故障
- CVM 故障
- 节点故障

3.2.6.1 磁盘故障

磁盘故障的特点是磁盘被拔掉、磁盘彻底坏了或者由于 I/O 出错被主动地删除。

虚拟机影响：

- HA 事件：没有
- I/O 失败：没有
- 延迟：没有影响

当出现磁盘故障时，会立即触发一个 Curator 扫描（MapReduce 框架）。将会扫描元数据以发现故障磁盘上的数据和哪些节点/磁盘拥有副本。

一旦发现数据需要重新复制，它会把复制任务分派到整个集群的节点。

一个重要的事情需要在这里强调的是鉴于 Nutanix 把数据和副本分散到所有节点/CVMs/磁盘，所有节点/CVMs/磁盘将参与到重新复制中。

这样实质上减少了重新回到保护状态的时间，因为可以利用整个集群的能力。集群越大，越快回到保护状态。

3.2.6.2 CVM 故障

CVM 故障的特点是由于 CVM 的权利行为引起 CVM 临时不可用。系统设计本身可以很好的透明地处理这些故障。当 CVM 故障时，I/O 将被重定向到集群的其它 CVM。具体机制因 Hypervisor 不同而各异。CVM 的滚动升级过程实际上是利用了这个能力，每次升级一个 CVM，迭代到整个集群。

虚拟机影响：

- HA 事件：没有
- I/O 失败：没有
- 延迟：由于 I/O 经过网络，潜在的高延迟

在出现 CVM 故障的时候，由原来故障 CVM 处理的 I/O 将会被转发到集群里的其它 CVM 上。ESXi 和 Hyper-V 是通过一个叫 CVM 自动路径的进程来处理的，利用 HA.py(比如"happy")，修改路由，把发送到内部地址（192.168.5.2）的数据包转发到集群里其它 CVM 的外部地址。这样使得数据存储保持原封不动，只是 I/O 由远程 CVM 来处理。

一旦本地 CVM 恢复正常并且稳定了，路由将被删除，本地 CVM 也将接管新的 I/O。

对于 KVM，是利用 iSCSI 多路径技术，主路径是本地 CVM，另外两个路径是远程。当主路径故障时，其中一个远程路径将被激活。

类似于 ESXi 和 Hyper-V 的自动路径技术，当本地 CVM 恢复正常后，本地 CVM 将接管成主要路径。

3.2.6.3 节点故障

虚拟机影响：

- HA 事件：发生
- I/O 失败：没有
- 延迟：没有影响

如果一个节点失效了，虚拟机 HA 机制将启动，将会在另外的节点上重新启动虚拟机。一旦被重新启动，虚拟机将在本地 CVM 接管下继续提供服务。类似于磁盘失效，Curator 扫描将发现失效的数据节点和冗余的数据节点。与磁盘失效类似，节点失效也会在所有正常的集群节点内实现被保护数据的再平衡。

当节点故障持续相当长的时间，失效的 CVM 将被从元数据环中删除，当节点恢复和稳定一段时间后，CVM 将被重新加回环中。

3.2.7 容量优化

Nutanix 平台集成了多种存储优化技术并协同工作，使任何工作负载的可用容量被有效利用，这些技术具备智能和自适应的工作特性，消除了需要手动配置和微调。

下面是一些集成的优化技术：

- 纠删码
- 压缩
- 消重

数据转换	应用	说明
纠删码	所有	在不影响正常写入或读 IO 性能的同时提供比传统的 RF 更高的可用性，且降低开销。在磁盘/节点/块故障的情况下，同时数据必须被解码的情况下有一些读的开销
在线压缩	所有	不影响随机 IO，有助于提高存储层利用率。通过减少数据从磁盘复制和读取，有利于大的 IO 或顺序的 IO 性能
事后压缩	无	在线压缩将仅仅压缩大的或连续的写入，而随机或小的 IO/ 应该采用事后压缩处理
Perf Tier 消重	P2V/V2V,Hyper-V (ODX),Cross-container clones	为没有克隆或通过采用 Acropolis 克隆的数据提供更大的缓存效率
容量层消重	P2V/V2V,Hyper-V (ODX),Cross-container clones	拥有上面的好处同时，减少在磁盘上的开销

3.2.7.1 纠删码

Nutanix 平台依赖于 RF 来保护数据和可用性，这个方法提供了最高程度的可用性，因为当数据失效时，不需要从其他存储位置或者重新计算来读取。然而，由于需要完全复制，存储资源的成本比较高。

为了提供一个可用性和降低存储容量的平衡，DSF 提供了使用纠删码（EC）来对数据进行编码。

和 RAID（级别 4，5，6 等）的概念相似，校验位是计算出来的。EC 对不同节点上的数据块条带进行编码和计算校验位。当主机或者磁盘故障时，校验位可以被利用为计算任何缺失的数据块（解码）。在 DSF 里，数据块是一个 extend 组，每个数据块必须在不同的节点上，属于不同的 vDisk。

一个条带里的数据块的数量和校验块是可以基于要求能容忍的失效而配置的。配置通常采用数据块数量/校验块数量来表示。

例如，RF2 级别的可用性（N+1）在一个条带里能够包括 3 个或者 4 个数据块和一个校验数据块（例如 3/1 或者 4/1）。RF3 级别的可用性（N+2）在一个条带里能够包括 3 个或者 4 个数据块和两个校验数据块（例如 3/2 或者 4/2）。

专家提示：你可以通过 NCLI 命令 'ctrl [create/edit] … erasure-code=<N>/<K>'来修改缺省的条带大小，其中 N 代表数据块的数量，K 代表校验块的数量。

预期的消耗可以用校验块数量/数据块数量来计算。例如，一个 4/1 的条带有 25%的消耗，或者是 1.25*（RF2 是 2*）。一个 4/2 条带有 50%的消耗或者是 1.5*（RF3 是 3*）。

下表是列出了常见条带大小和消耗的例子：

	FT1 (RF2 equiv.)		FT2 (RF3 equiv.)	
Cluster Size (nodes)	EC Strip Size (data/parity blocks)	EC Overhead (vs. 2X of RF2)	EC Strip Size (data/parity)	EC Overhead (vs. 3X of RF3)
4	2/1	1.5X	N/A	N/A
5	3/1	1.33X	N/A	N/A
6	4/1	1.25X	N/A	N/A
7+	4/1	1.25X	4/2	1.5X

专家提示：强烈建议集群里节点的数量要比条带大小组合（数据块数量+校验块数量）至少加一，以便当节点故障时可以重建条带。这样避免了条带重建时（由 Curator 自动进行）产生的计算过载。例如，一个 4/1 条带应该在集群里至少有 6 个节点。上面的表格是按照最佳实践得出来的。

编码是事后进行的，利用 Curator MapReduce 框架来分发任务。由于是事后处理，不会影响传统的写 I/O 通道。

一个正常使用 RF 的环境看起来像这样的：

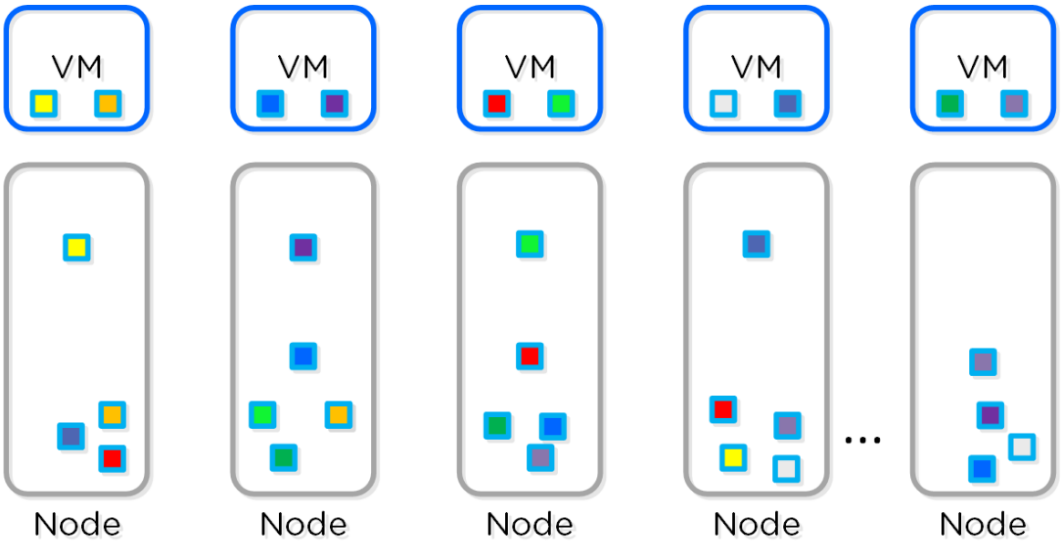


Figure 11-13. Typical DSF RF Data Layout

在这个场景里，我们有混合 RF2 和 RF3 数据的集群，主数据块在本地，副本数据块在集群的其它节点上。

当 Curator 运行完全扫描时，它会发现合适的 extend 组可以被编码的。合适的 extend 组必须是“写冷的”，也就是它们写入超过一个小时以上。在合适的 extend 组被发现后，编码的任务将通过 Chronos 分发和控制。

下图展示了一个 4/1 和 3/2 条带的例子：

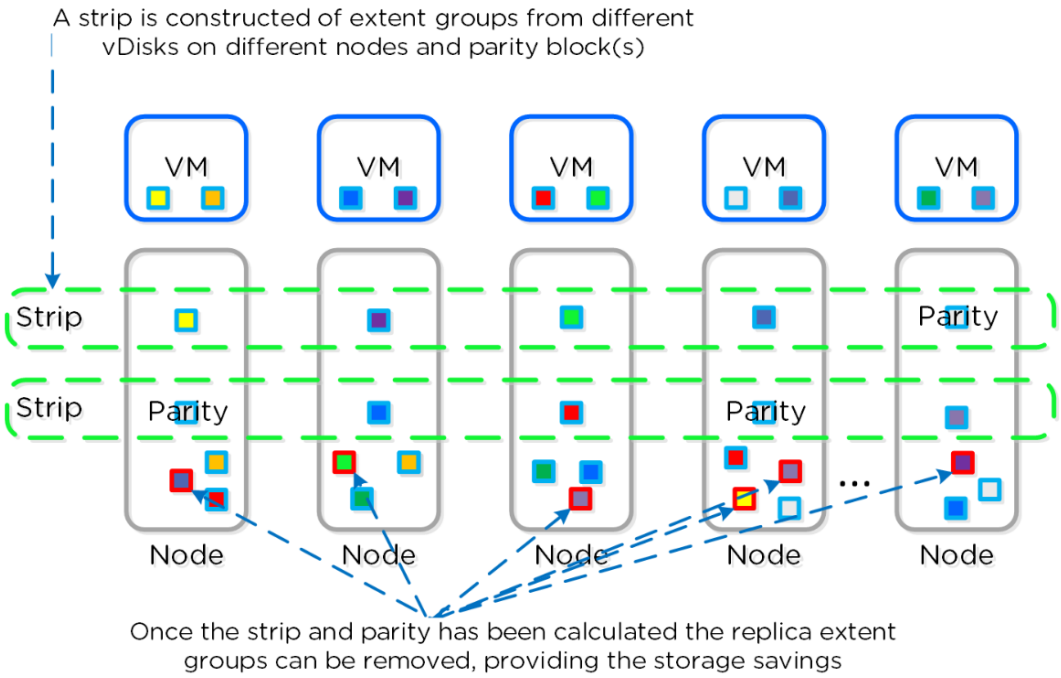


Figure 11-14. DSF Encoded Strip - Pre-savings

一旦数据被成功地编码（条带和校验计算后），extend 组的副本将被删除。

下图展示了运行 EC 节省存储空间后的环境：

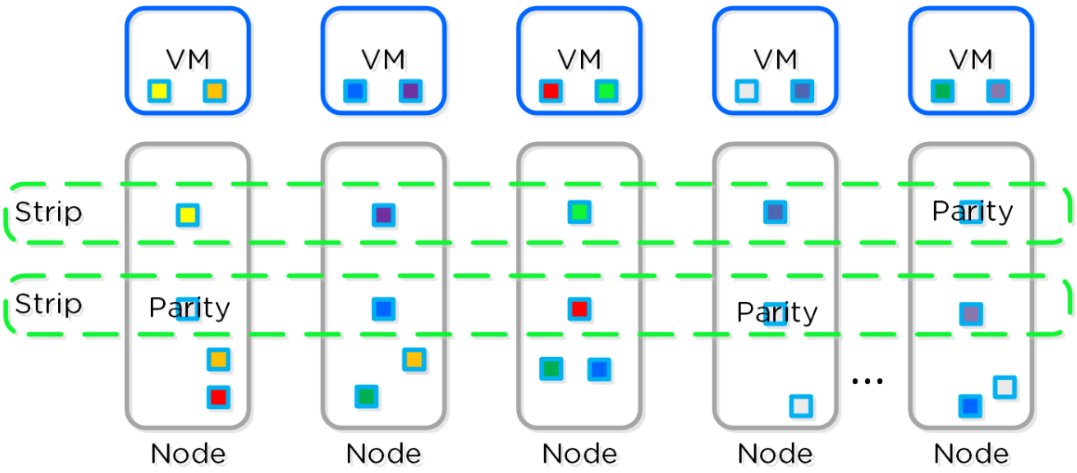


Figure 11-15. DSF Encoded Strip - Post-savings

专家提示：纠删码和在线压缩可以完美地结合使用，大大节省存储空

3.2.7.2 压缩

你可以通过观看下面视频来帮助理解：https://youtu.be/ERDqOCzDcQY

Nutanix 容量优化引擎（COE）负责数据的转换来增加磁盘中数据的效率。目前压缩是 COE 实现数据优化的一项关键特性。DSF 提供在线和事后两种压缩方式以满足客户的需求和不同数据类型。

在线压缩是在数据写入到磁盘之前，对顺序的数据流或大 I/O 数据块在内存里进行压缩。事后压缩是正常写数据，然后利用 Curator 框架在集群范围内压缩数据。当启用在线压缩时，随机 I/O 数据没有压缩地写入 OpLog，合并后在内存里压缩，然后才写入扩展存储。压缩时使用 Google Snappy 压缩库，提供较高的压缩率，减少计算负担，特别快的压缩和解压缩速度。

下面图示展现了一个在线压缩和 DSF 写 I/O 通道如何交互的例子：

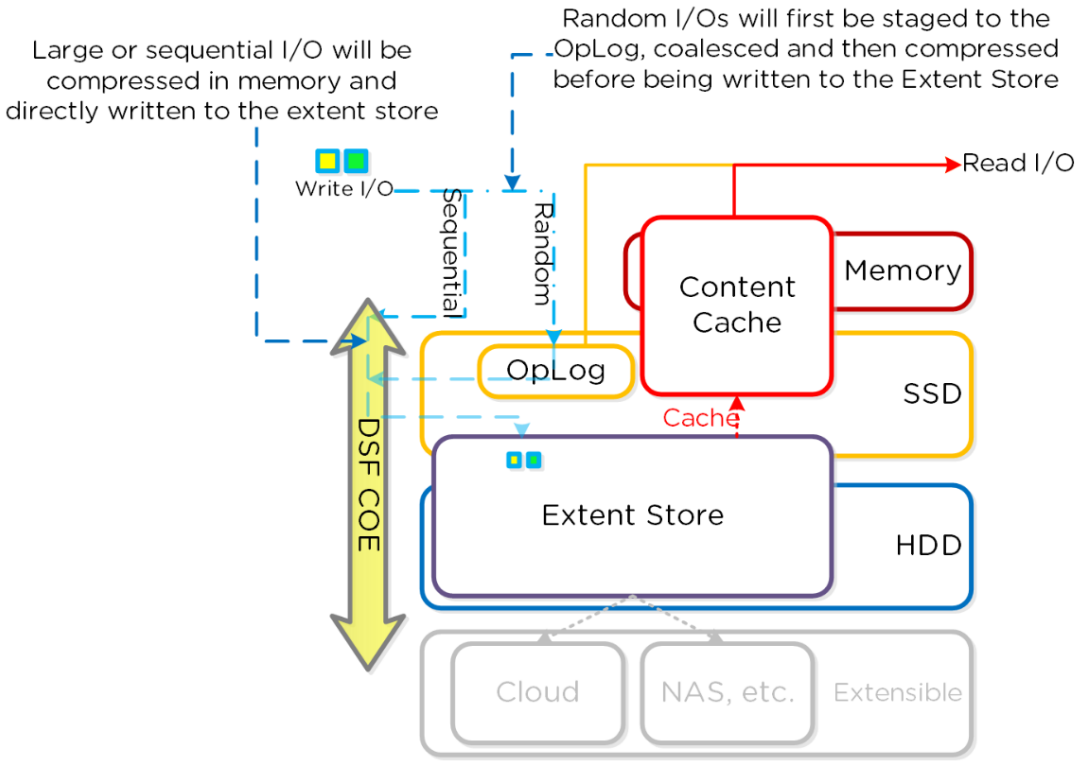


Figure 11-10. Inline Compression I/O Path

专家提示：建议总是使用在线压缩（压缩延迟=0），它只压缩大的/顺序写，不影响随机写的性能。在线压缩也完美地和纠删码配合使用。

对于事后压缩，所有的写 I/O 按照正常 DSF 的 I/O 通道没有压缩地写入。当压缩延迟（可配置）到达后，数据变冷（通过 ILM 把数据下移到 HDD 层），数据这时可以被压缩。事后压缩使用 Curator 的 MapReduce 框架，所有节点将参与压缩任务。压缩任务将被 Chronos 控制。

下图展示了一个事后压缩和 DSF 写 I/O 通道如何交互的例子：

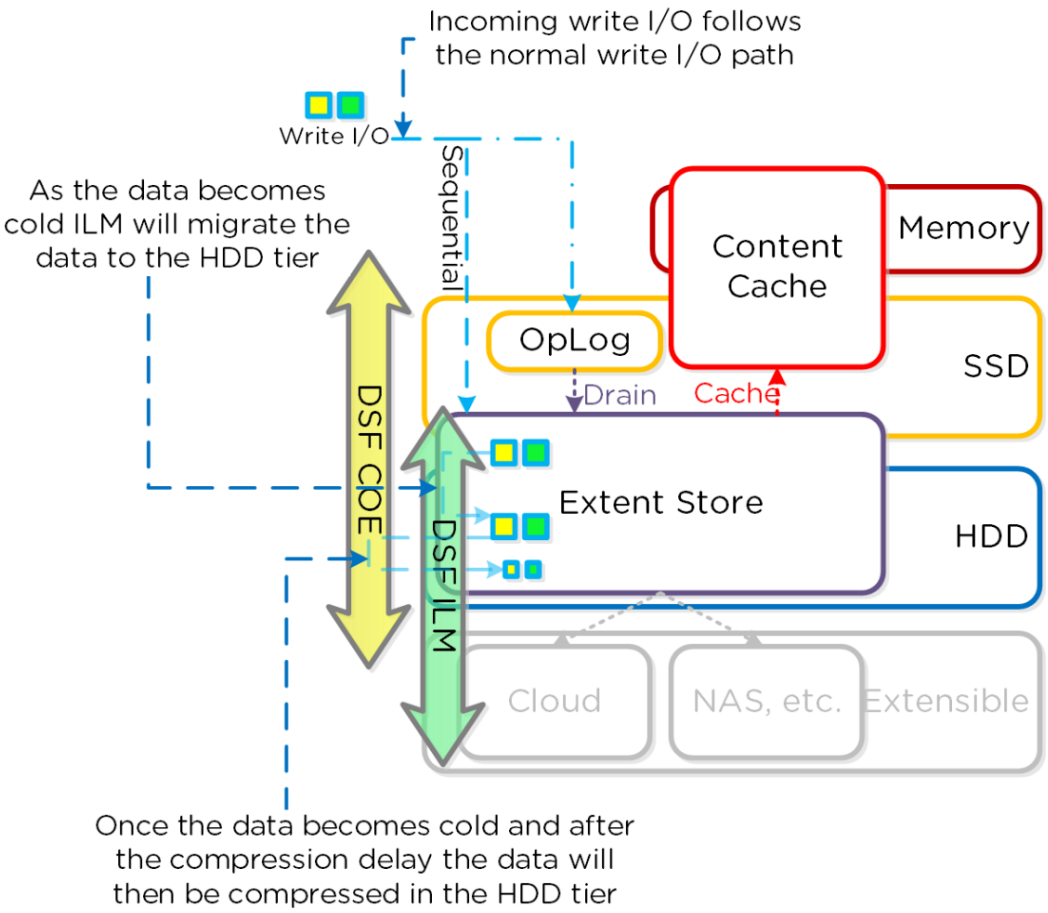


Figure 11-11. Post-process Compression I/O Path

对于读 I/O，数据首先在内存里解压，然后提供 I/O 服务。如果数据被频繁访问，数据将在 HDD 层变成解压的，然后能够利用 ILM 移上 SSD 层，并且在 Cache 上存放。

下图展现了一个在读的过程中解压缩和 DSF I/O 通道如何交互的例子

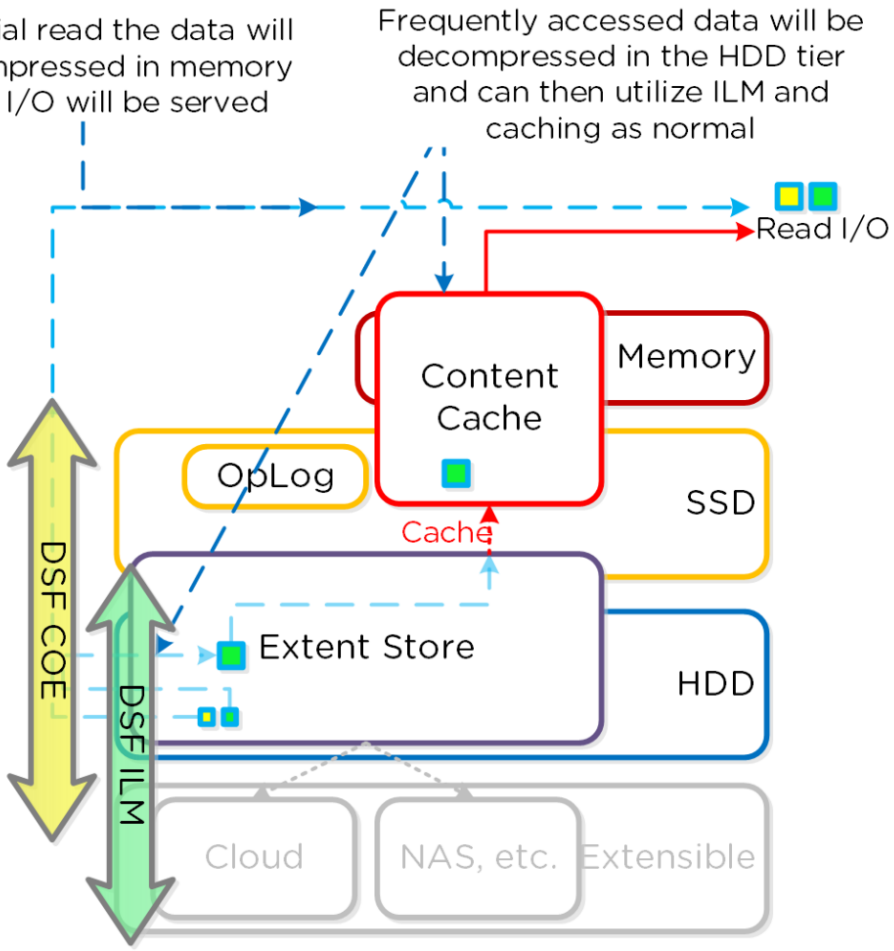


Figure 11-12. Decompression I/O Path

你可以从 Prism 上，在存储>仪表盘页面上看到当前压缩率。

3.2.7.3 弹性消重引擎

你可以通过观看下面视频来帮助理解：https://youtu.be/C-rp13cDpNw

弹性消重引擎是 DSF 中一个基于软件的特性，它允许在容量层面（HDD）和性能层面（SSD/内存）对数据进行消重操作。顺序的数据流在写入（ingest）时，按照 16KB 粒度进行 SHA-1 的散列（hash）运算标记指纹。指纹标记操作仅在数据块被写入（ingest）的时候进行，并作为该数据块的元数据信息的一部分被持久保存。注意：最初采用 4KB 粒度进行指纹标记，然而经过测试， 16KB 粒度的指

纹标记操作能在消重能力和减少元数据(metadata)开销之间取得最好的平衡。当已消重数据进入 unified cache 之后，将以 4KB 粒度进行消重。

传统消重操作需要在后台进行数据块扫描，数据被重新读取并开销大量系统资源，Nutanix 在写入时执行在线的指纹标记操作。消重时，外部存储上重复数据块可以被直接删除，而无需重新读取和计算指纹。

为了使元数据更加高效率，指纹标记参考计数被监控来跟踪可消重能力。参考计数低的指纹计数将被丢弃以减少元数据的过载。为最小化碎片，完全 extents 是用于在容量层面的消重首选.

专家提示：

对基本影像（Base images）使用性能层面的消重会利用内容缓存的优势。（你可以手工对它们进行指纹标记，使用 vdisk_manipulator）。

当使用 Hyper-V 时（由于 ODX 实行完全数据拷贝），或者进行跨容器的克隆时（通常不建议，因为最好是一个容器），使用容量层面对 P2V/V2V 消重。

在大部分其它案例，压缩是节省容量最大的，应该建议使用。

下图显示了弹性消重引擎如何扩展和处理本地虚机 I/O 请求的：

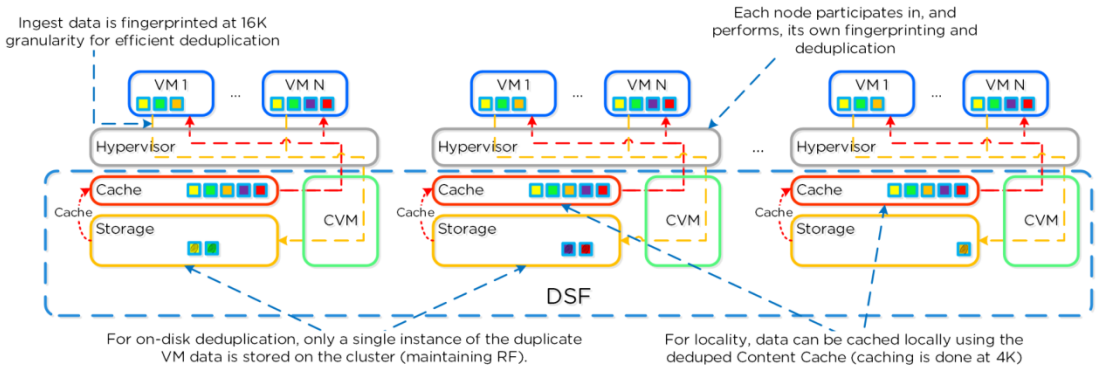


Figure 11-16. Elastic Dedupe Engine - Scale

指纹标记操作是在数据写入大小为 64KB 或更大时发生，Intel 的加速器能使得 SHA-1 的计算增加的额外 CPU 开销最小。如果数据不是持续写入（eg. 小 IO 操作），将不进行指纹标记操作，此时指纹标记操作将在后台发生。弹性消重操作不仅发生在磁盘层面（HDD），并且也发生在高性能的存储层面（SSD/Memory）。当确定了哪些重复数据后，即基于相同指纹的多份数据，一个后台进程会使用 DSF 的 MapReduce 框架（curator）来移除重复数据。

由于标记指纹的数据已经被读取，会被保存在 DSF 的 Unified Cache 中（这是 multi-tier/pool cache）。任何后续对于同样指纹的数据请求将直接从 Cache 层面响应。要了解统一缓存和池架构，请参考 I/O 路径概览章节相关内容。

专家提示：指纹标识 vDisk Offsets

在 4.5 之前，只有 vDisk 的前 12GB 可以被指纹标识，这是用来维护一个较小的元数据指纹标识并且由于 OS 通常是最普通的数据。4.5 之后，增加到

下图显示了弹性消重操作如何优化 IO 操作：

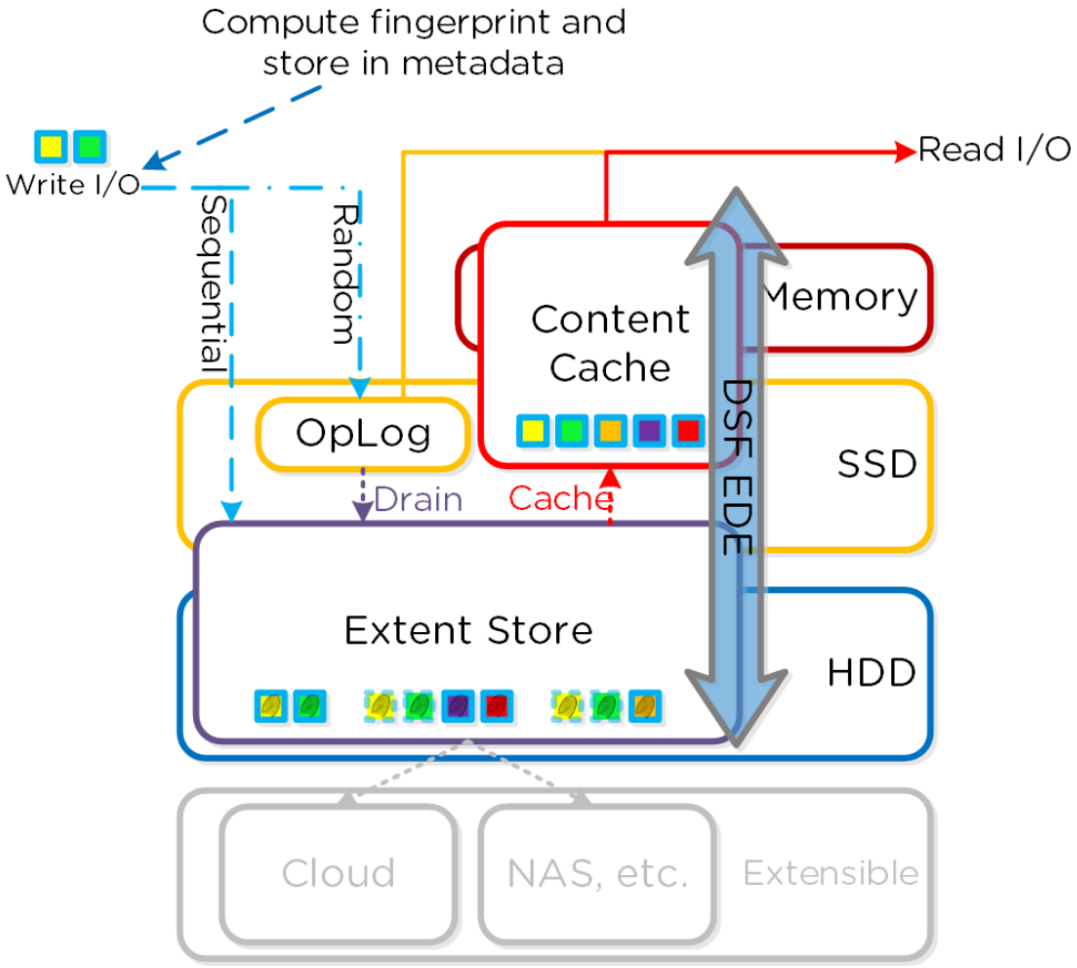


Figure 11-17. EDE I/O Path

你可以从 Prism 上的存储>面板页查看当前的消重率。

专家提示：消重+压缩

在 4.5 版本，消重和压缩都可以在一个容器里启用，但是，除非数据是可以被消重的（条件在之前的章节里解析过），坚持使用压缩。

3.2.8 存储分层和优先级

在磁盘平衡章节我们谈到了如何将集群内所有节点的磁盘容量作为存储池统一管理，并且通过 ILM 实现热数据本地化。简单而言，磁盘的分层是实现在集群内所有节点的 SSD 和 HDD 上，并且由 ILM 负责触发数据迁移。

本地节点的 SSD 层总是最高优先级的，负责所有本地虚拟机 I/O 的读写操作。并且还可以使用集群内所有其他节点的 SSD，因为 SSD 层总是能提供最好的读写性能，并且在混合存储环境中尤为重要。

下图显示了热分层优先级顺序：

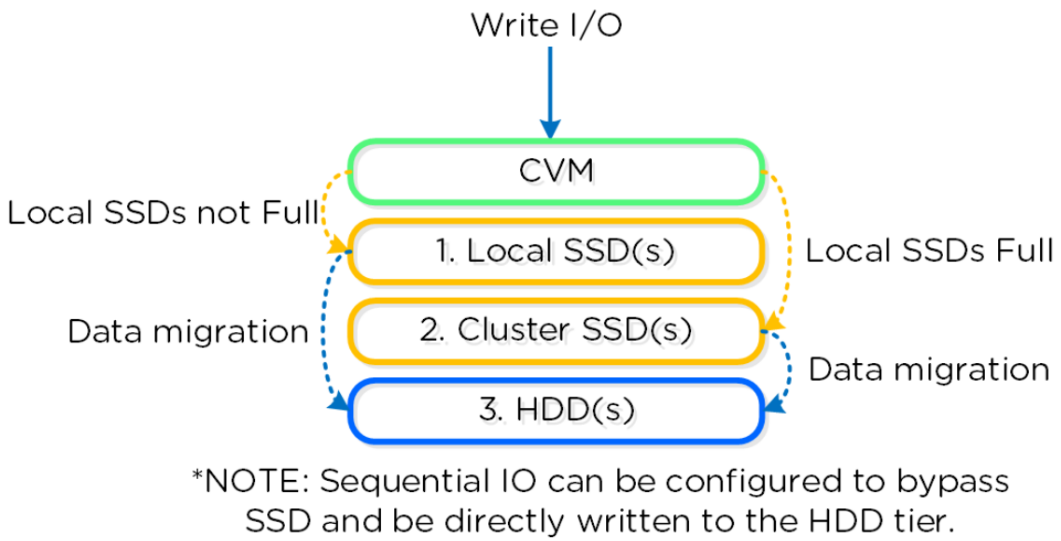


Figure 11-18. DSF Tier Prioritization

特定类型的存储资源在集群范围内被池化统一管理，形成集群范围的存储分层。这意味着集群内任何节点都可以使用到整个存储分层，无论这一存储资源在本地还是在其他节点上。

下图显示的时池化后的集群分层示意图：

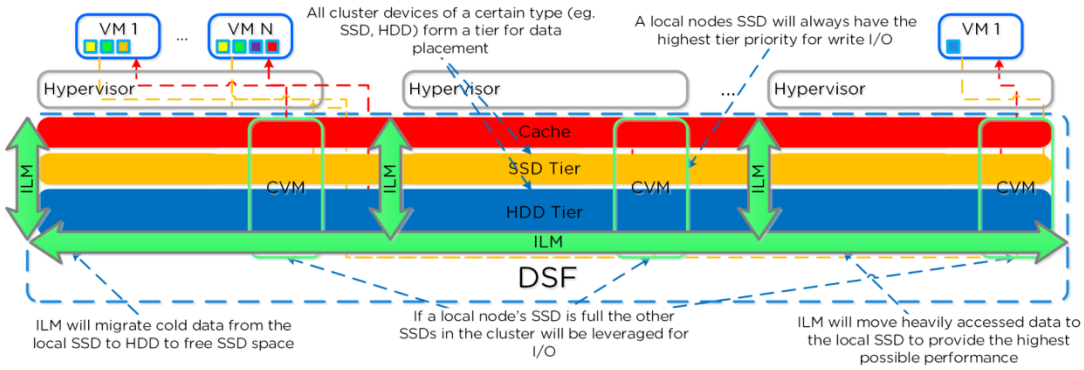


Figure 11-19. DSF Cluster-wide Tiering

一个经常被问到的问题就是：如果节点本地的 SSD 被用满之后，性能是否会下降？之前我们提到使用磁盘平衡功能会保证集群范围内所有磁盘的利用率基本一致。那么当本地 SSD 利用率非常高时，磁盘平衡功能会自动迁移 SSD 中最冷的数据到集群内另一个节点的 SSD 上。这将释放本地 SSD 的空间，数据会继续写到本地 SSD 上，而不会因为通过网络写到另一个节点的 SSD 上而导致性能下降。一个关键点需要提及的是所有 CVM 和 SSD 都参与这种远程 IO 操作，可以消除潜在的瓶颈，并且通过执行远程 IO 操作可以自愈一些命中率的问题。

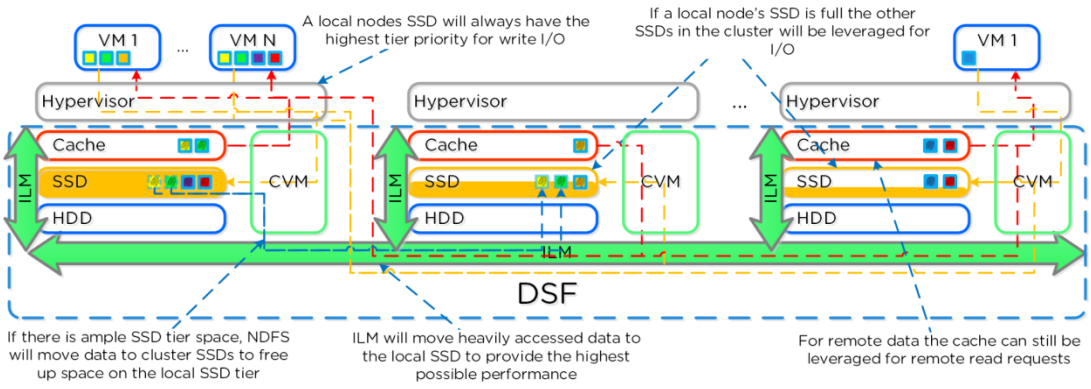


Figure 11-20. DSF Cluster-wide Tier Balancing

另一种情况是当存储利用率到达一定的阈值（curator_tier_usage_ilm_threshold_percent，默认是 75），Curator 框架可以自动触发，并将数据从 SSD 上迁到 HDD 上，并且通过参数

（curator_tier_free_up_percent_by_ilm，默认是 15）确定迁移的数据量，可能更多。

迁移数据是使用最后访问时间来判断。例如当 SSD 利用率到 95%，那么将有 20% 的数据会被移动到 HDD（95% -- 75%），如果 SSD 利用率时 80%，则只有 15%的数据会被移动到 HDD。

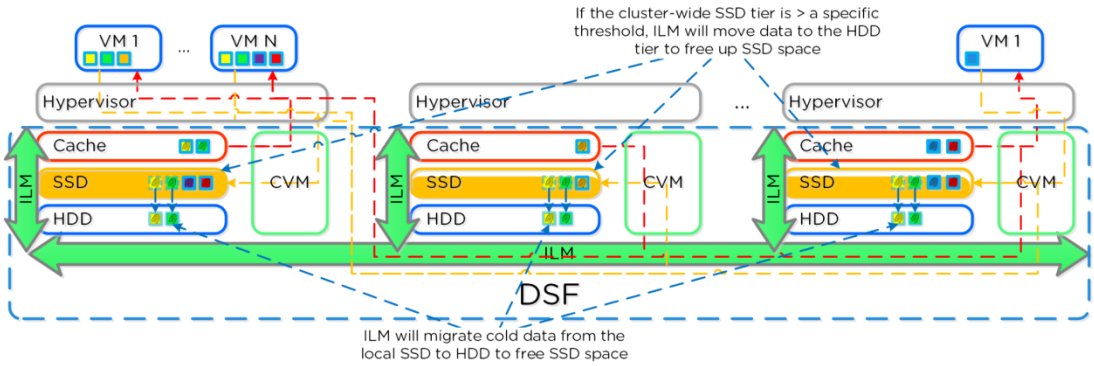


Figure 11-21. DSF Tier ILM

DSF 的 ILM 也会持续监控数据分布，必要时迁移数据，使最热的数据本地化。

3.2.9 磁盘平衡

你可以通过观看下面视频来帮助理解：https://youtu.be/atbkgrmpVNo

DSF 被设计成为非常动态的平台，可以适用于不同工作负载的应用，并且允许混合节点类型：例如将计算密集型（3050 系列）和存储密集型（60X0 系列）混合在一个集群中。对于集群内部磁盘容量大小不同的，确保数据一致的分布非常重要。

DSF 有自带的称为磁盘平衡的技术，用来确保数据一致的分布在集群内部各节点上。磁盘平衡功能与各节点的本地磁盘利用率和内置的 DSF ILM（数据生命周期管理）一同工作。它的目标是一旦利用率超出设定阈值时，使得所有节点的磁盘利用率大致相等。

下图显示了一个混合集群（3050 系列+6000 系列）的不平衡状态：

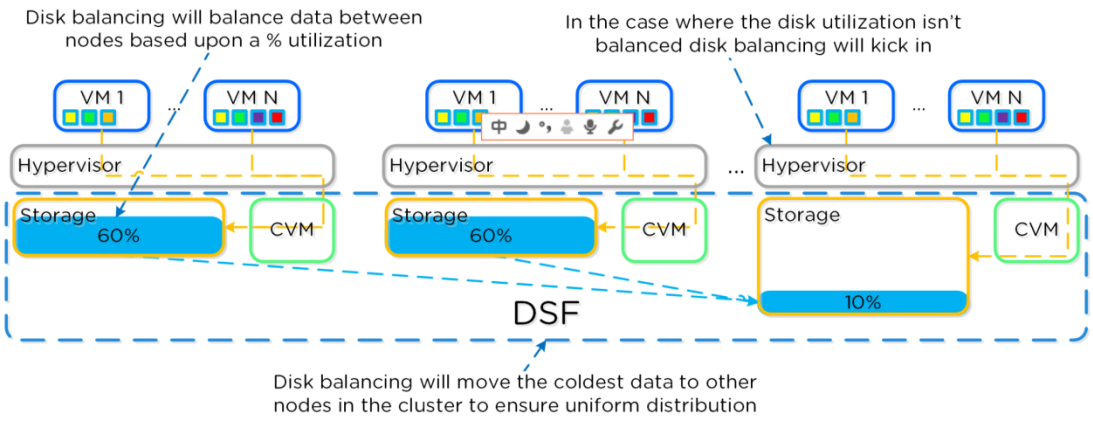


Figure 11-22. Disk Balancing - Unbalanced State

磁盘平衡功能使用 DSF 的 Curator 框架作为调度进程，当磁盘阈值被触发时，在后台自动运行（例如，当本地节点利用率超过 n%后自动运行）。当数据不平衡时，Curator 会决定哪些数据移动到哪里，并自动生成 MapReduce 任务分发到各节点执行。例如，集群各节点同样都是相同型号（例如，3050 系列），每个节点的利用率应该基本一致。

然而，当一些 VM 写了过多的数据，导致该 VM 数据快速增长，使得该节点磁盘利用率高于其他节点。此时磁盘平衡功能可以将该节点上最冷的数据移动到其他节点上，保证所有节点磁盘利用率达到基本平衡的状态。例如，集群各节点时不同型号（例如，3050 系列加 6020/50/70 系列），或者一些节点仅作为存储节点使用时，可能会触发磁盘平衡功能。

下图显示了一个混合集群（3050 系列+6000 系列）的“平衡”状态：

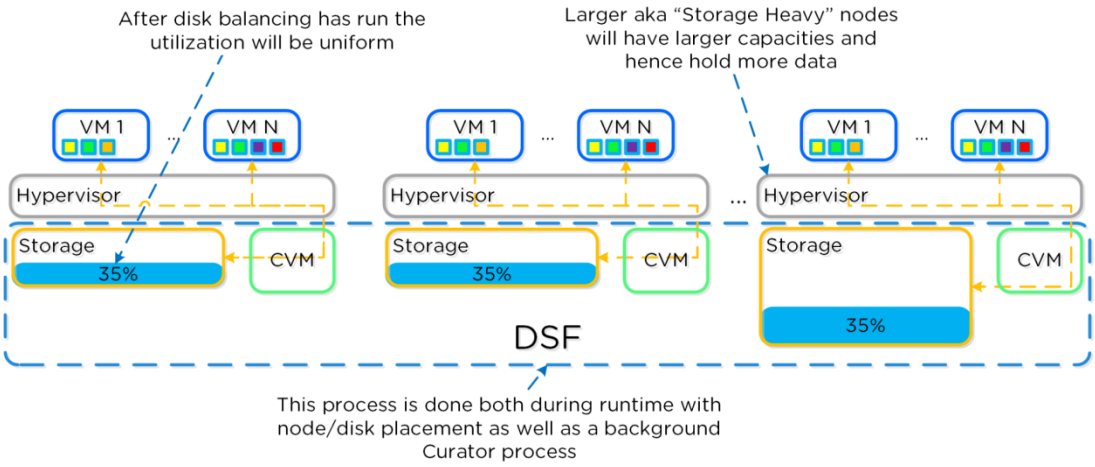


Figure 11-23. Disk Balancing - Balanced State

在一些情形下，客户可能希望将一个节点只作为存储使用，则这个节点上将没有虚拟机运行，它的主要功能作为大容量存储设备使用。在这种情况下，节点上所有内存都可以给到 CVM 使用，来提供更大的读缓存。

下图显示了混合集群中包含仅存储节点，并使用“磁盘平衡”功能后数据从活动虚拟机节点移动到它上的情况：

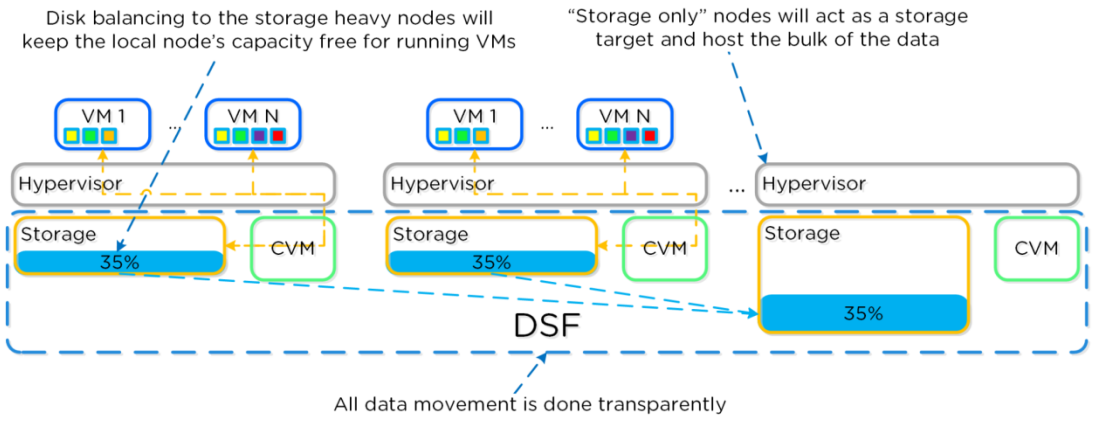


Figure 11-24. Disk Balancing - Storage Only Node

3.2.10 快照和克隆

你可以通过观看下面视频来帮助理解：https://youtu.be/uK5wWR44UYE

DSF 内置支持快照和克隆的卸载(offload)，能够利用 VAAI,ODX,ncli,REST,Prism 等进行快照和克隆。快照和克隆均利用最有效和高效的写时重定向（redirect-on-write）算法。正如在上述数据结构章节解析的，一个虚拟机构成的文件（vmdk/vhdx）在 Nutanix 平台是 vDisks。

vDisk 是由 extend（逻辑上连续的数据块）构成，extend 存放在 extend 组里，extend 组是物理上连续的数据，在存储设备里以文件形式存放。当快照或者克隆发生时，基础 vDisk 被标识为不可改变的，创建另外一个 vDisk 为可读写。这时候，两个 vDisk 都有相同的数据块映射，它是 vDisk 到相应 extend 的元数据映射。传统的方法需要遍历快照链(增加读延迟)，而现在每个 vDisk 有它自己的数据块映射。这样就消除了常见的很深的快照链的过载，允许你持续地进行快照而不影响性能。

下图显示了快照是如何进行工作的例子：

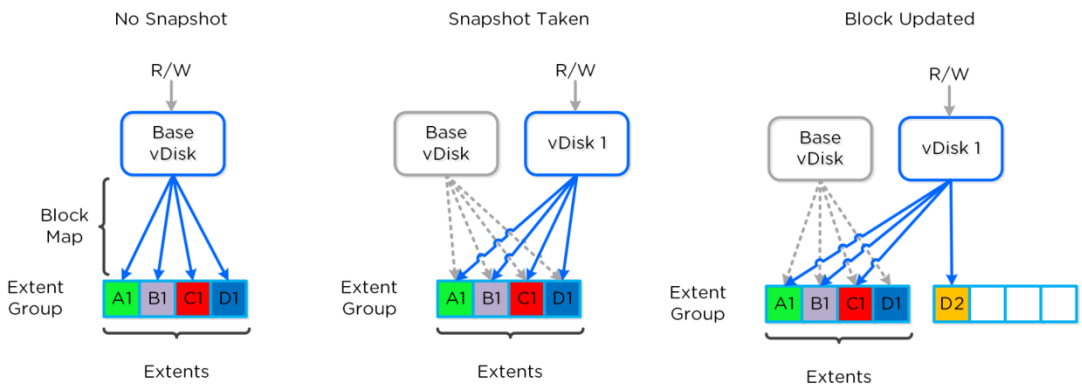


Figure 11-32. Example Snapshot Block Map

当一个 vDisk 的快照和克隆继续进行快照和克隆时，采用相同的方法：

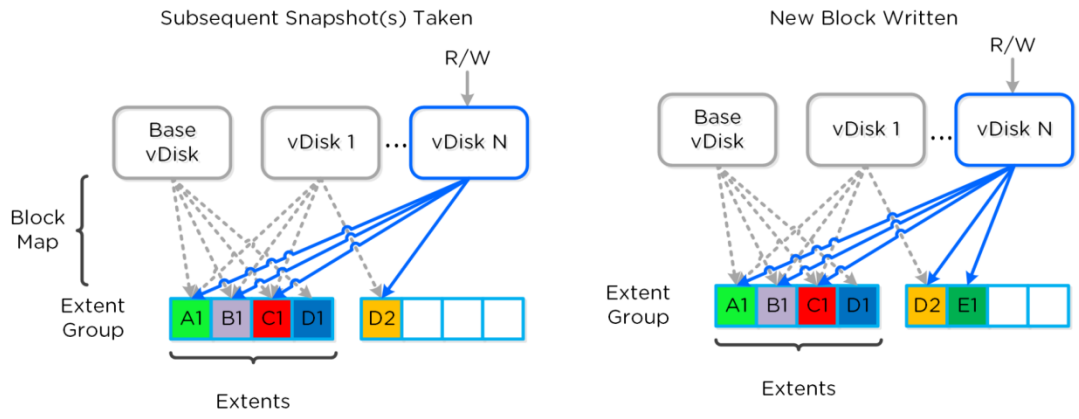


Figure 11-33. Multi-snap Block Map & New Write

采用相同的方法进行虚机和 vDisk 的快照和克隆。当一个虚机或 vDisk 进行克隆时，当前的块映射被锁定，克隆被创建出来。这些更新仅仅是元数据，没有 I/O 实际发生。相同的方法应用到克隆的克隆。之前克隆的虚机充当基础 vDisk，数据块映射被锁定，2 个克隆被创建：一个给已经克隆的虚机，另外一个给新的克隆。

它们均继承了之前的数据块映射，任何新的写入和更新将会发生在它们单独的数据块映射里。

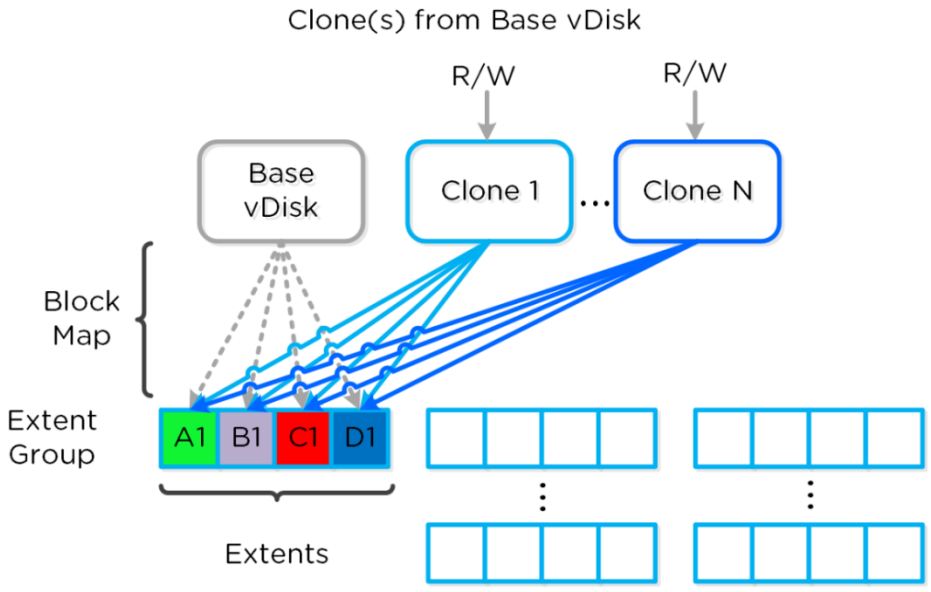


Figure 11-34. Multi-Clone Block Maps

如之前说过，每个虚机/vDisk 有它自己的单独的数据块映射。所以在上述例子里，所有从基础虚机来的克隆将会有它们自己的数据块映射，所有写和更新将会发生在那里。

下图显示了这些看起来是什么样的例子：

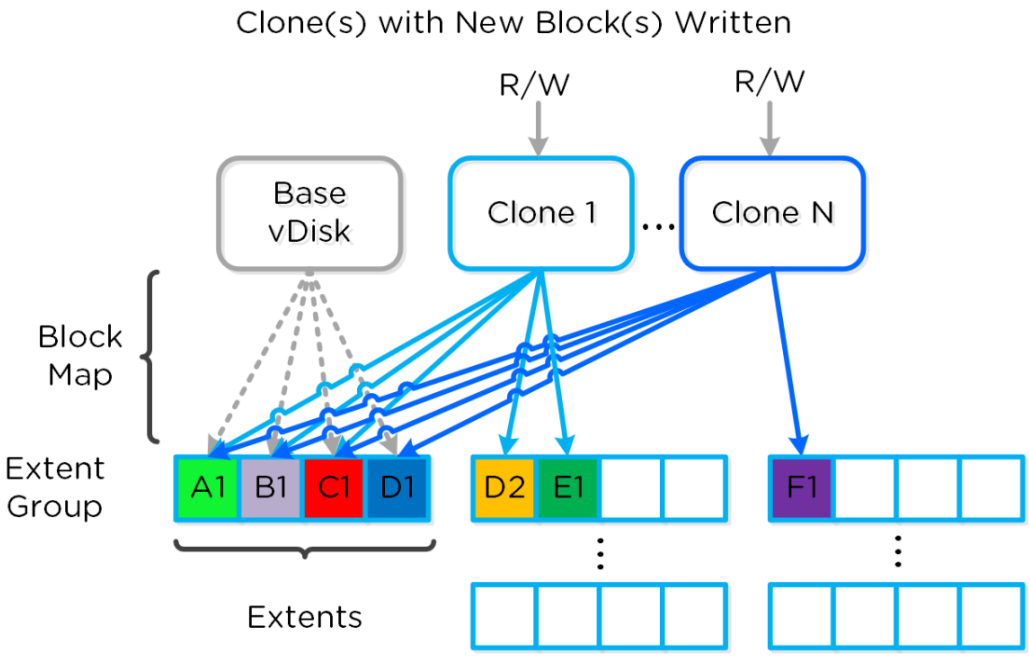


Figure 11-35. Clone Block Maps - New Write

任何后续虚机/vDisk 克隆或快照将会引起原始数据块映射被锁定，并将产生一个新的读写访问。

3.2.11 Replication and DR

关于视频解说，你可以查看以下链接：https://youtu.be/AoKwKI7CXIM

Nutanix 提供原生的容灾（DR）和复制功能，它们构建于“快照”&“克隆”等功能之上。Cerebro 是在分布式存储（DSF）中负责管理 DR 和复制的组件。Cerebro 运行于每个节点之中，通过内部选举产生 Master（类似于 NFS Master），并由此节点管理复制任务。如果当 Cerebro Master 所在的 CVM 发生故障，剩余的节点将选举出新的 Master。通过<CVM IP>:2020，即可打开 Cerebro 的相关界面。DR 功能可以分解为以下关键要点：

- .Replication Topologies
- Implementation Constructs
- Replication Lifecycle
- Global Deduplication

Replication Topologies

一直以来，有几种主要的复制网络拓扑：点对点（Site to site），菊花链（hub and spoke），全网状 / 部分网状（full and / or partial mesh）。相对于传统的方案，它们仅提供“点到点”或“菊花链”的方式，Nutanix 提供“全网状”或更加灵活的“多到多”的拓扑方式。

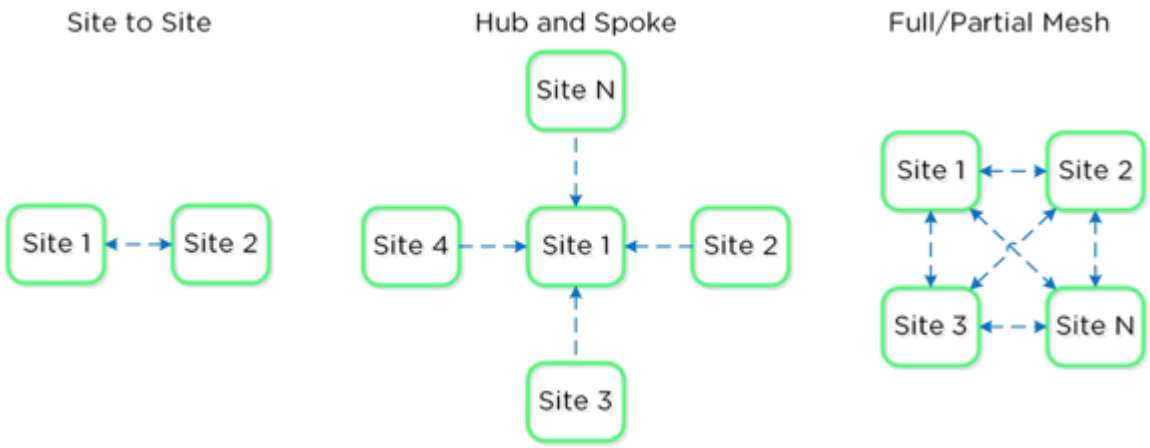


图 11-36. Example Replication Topologies

这将让管理员能够灵活的配置复制功能，从而更好的满足公司需求。

Implementation Constructs

在 Nutanix 的 DR，包含以下主要的功能组件：

远程站点（Remote Site）

- 主要角色： 远程站点的 Nutanix 群集
- 描述： 远程站点的 Nutanix 群集，用于充当备份或 DR 的目标端。

专家提示：

请确保容灾端有足够的资源（计算 / 存储）用于接管生产端的业务虚机。

保护域（PD / Protection Domain）

- 主要角色：同时保护的多个“虚拟机 / 文件”的逻辑组
- 描述：一组多个虚拟机或文件基于某个相同的保护策进行复制保护。一个 PD 可以保护一整个容器（Container）或你所选中的多个虚拟机或文件。

专家提示：

你可以针对不同的 RPO / RTO 需求，创建都个不同的 PD。例如你可以针

一致性组（CG / Consistency Group）

- 主要角色：PD 中多个相关联的 VM 或文件构成的一个子集，以实现故障时一致性。
- 描述：PD 中多台相关联的 VM 或文件需要在“同一时刻”发起快照。从而确保在虚拟机或文件回滚时的数据一致性。一个 PD 中可包含多个 CG。

专家提示：

相互依赖的多个应用或服务类虚拟机（e.g. APP and DB）放置在一个 CG 中，可确保在发生回滚时的数据一致性。

复制时间策略（Replication Schedule）

- 主要角色：快照、复制的时间策略

专家提示：

快照的时间策略应该符合预期 RPO 的要求

- 描述：为 PD 或 CG 中的 VM 提供定制的快照、复制的时间策略

保留策略（Retention Policy）

- 主要角色：本地或远程站点中保留的快照数量
- 描述：保留策略定义了本地或远程站点中保留的快照数量。注意：在远程保留/复制

> 专家提示：
>
> 保留策略即为 VM 或文件的还原点数量

策略配置前，必须先配置远程站点。

以下的图片展示了一个站点内，PD、CG 和 VM/File 间的逻辑关系

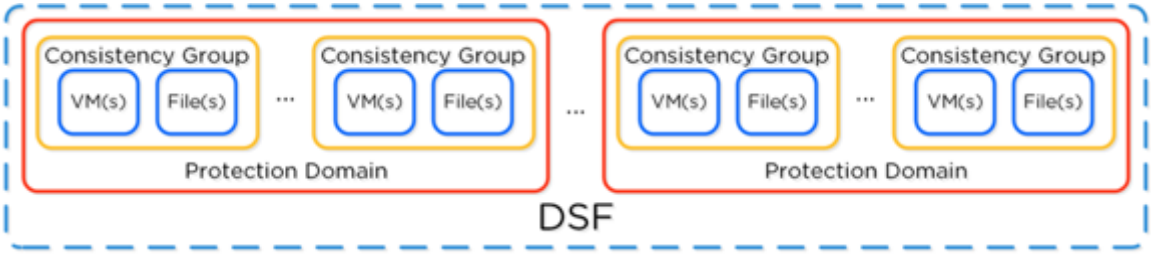


图 11-37. DR Construct Mapping

需要重点指出的是，我们不仅可以简单的就可以将一整个容器（Container）进行保护，还可以提供针对单个虚拟机或文件级的更加细致的保护。

Replication Lifecycle

Nutanix 通过 Cerebro 实现数据的复制。Cerebro 服务分为 Cerebro Master 和 Cerebro Slave。Cerebro Master 由动态选举产生，除 Cerebro Master 节点之外的 CVM 中，均运行 Cerebro 从属服务（Cerebro Slaves）。一旦“Cerebro Master”所对应的 CVM 宕机，新的 Master 将被自动选举产生。

Cerebro Master 负责委派任务给本地的 Cerebro 从属节点，以及协调远端的 Cerebro Master，实现容灾数据复制。

在复制过程中，Cerebro Master 将负责确认哪些数据需要被复制，同时将任务委派给 Cerebro 从属节点，随后将告知 Stargate 哪些数据需要被复制，需要被复制到哪里。

在复制过程中，复制数据在多个层面上提供保护。Extend 在源读取时是校验的来保证源数据的一致性（类似于 DFS 读），在目标端，新的 Extend 也被校验（类似于 DFS 写）。TCP 提供网络层面的一致性。

以下是相关架构的示意图：

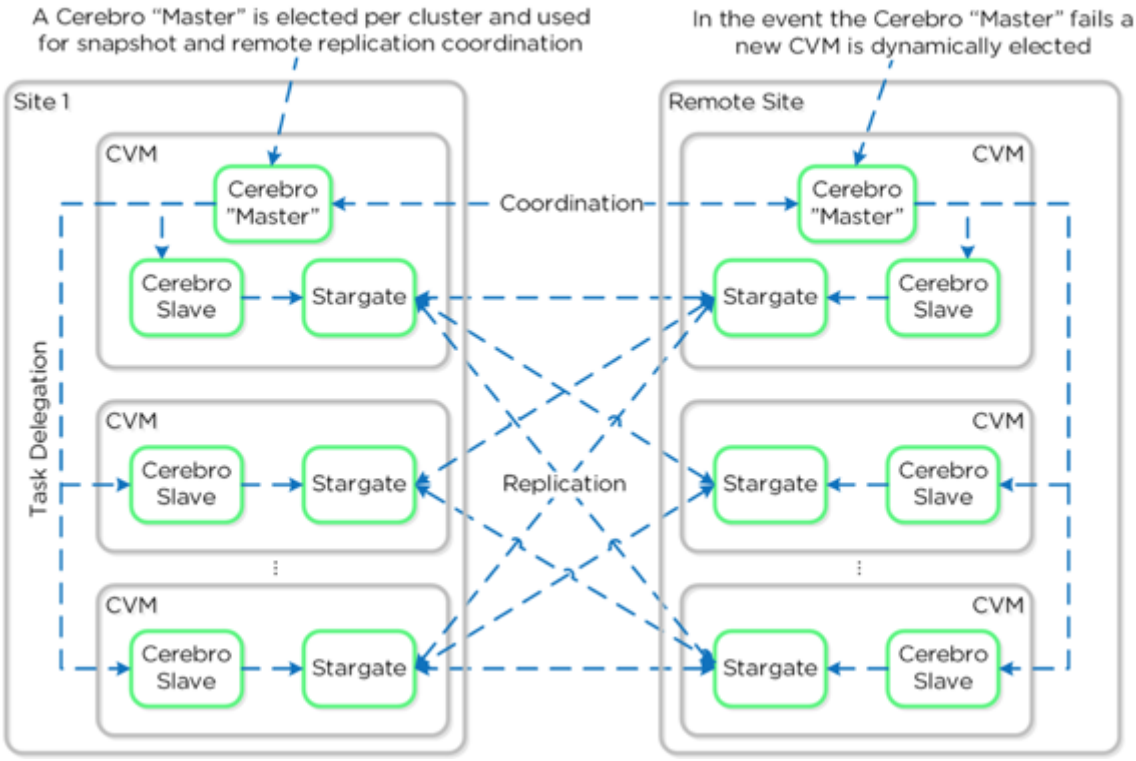


图 11-38. Replication Architecture

我们也可以通过配置代理，使用桥接的方式承载整个集群的协调和复制的数据流。

以下是使用代理的架构示意图：

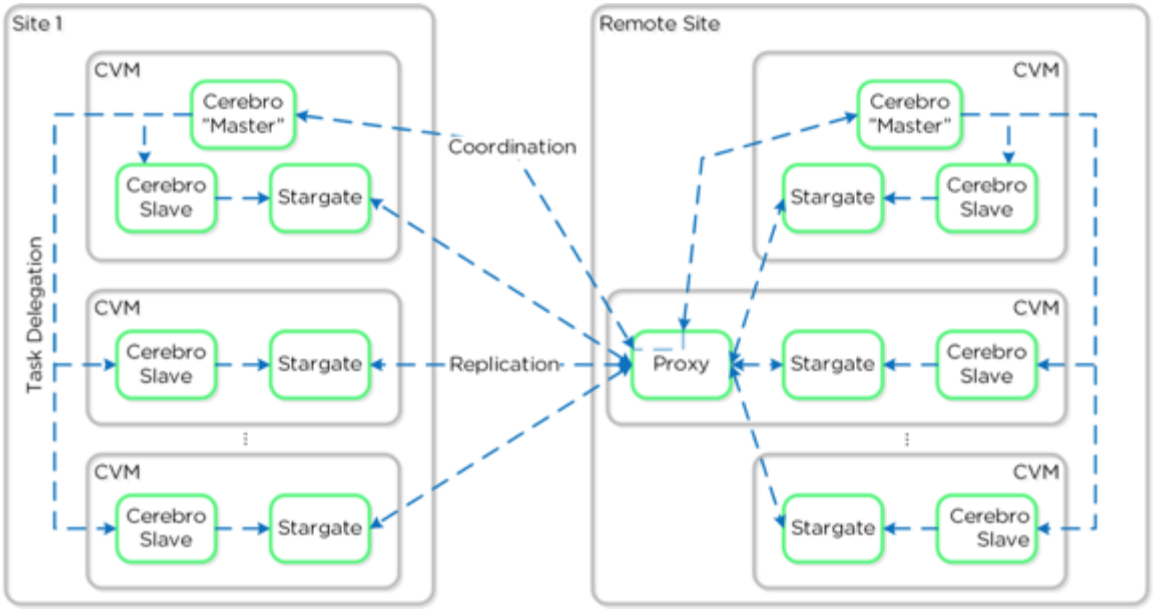


图 11-39. Replication Architecture - Proxy

> 注意：
>
> 这只适用于非生产环境下，并请使用集群 IP 以确保高可用。

在某些情况下，我们也可以配置 SSH 隧道链接两个 CVM。

以下是使用 SSH 隧道的架构示意图：

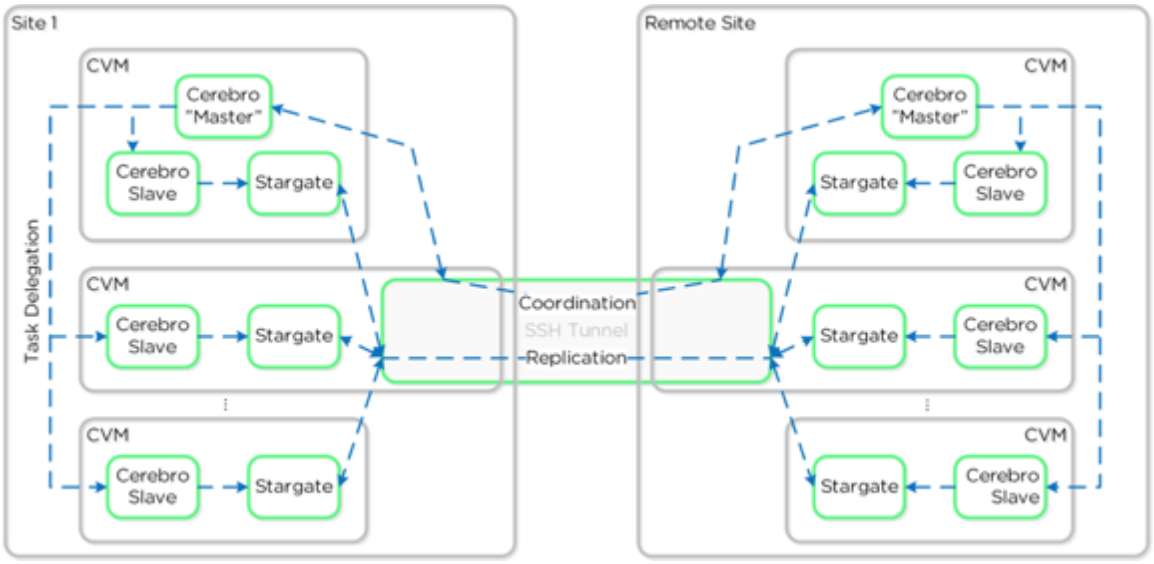
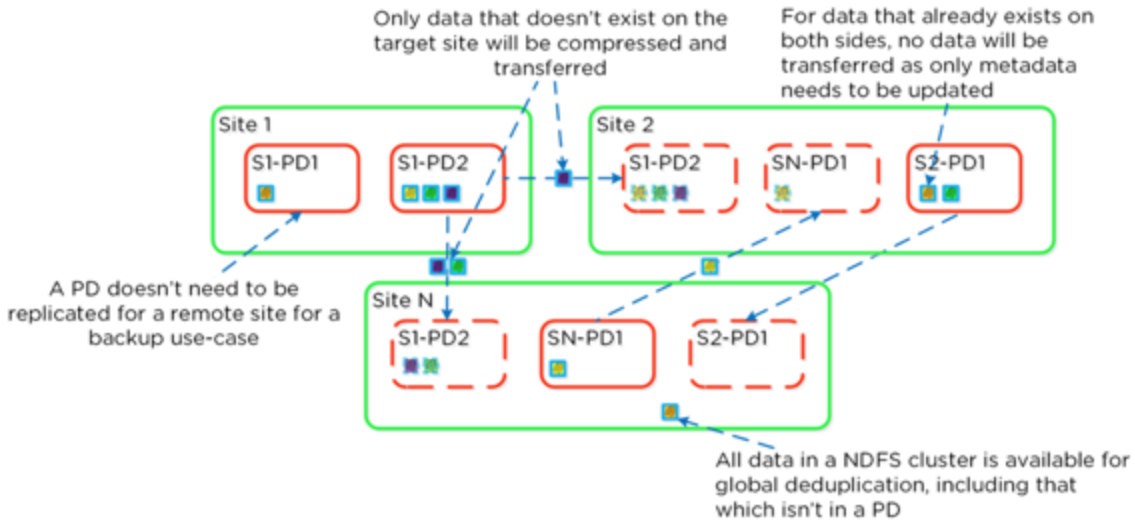


图 11-40. Replication Architecture - SSH Tunnel

Global Deduplication

正如之前“Elastic Deduplication Engine”章节中所提到的，分布式存储（DSF）可以仅通过更新元数据（Metadata）指针实现重复数据删除。同样的实现方式也可被用于 DR 和复制中。通过网络发送数据之前，DFS 将检查远程站点中是否已经存在相关数据的记录(fingerprint)。如果有，则不发送数据，仅更新元数据。如果远程站点中没有此数据记录，数据将被压缩并发送至目标站点中。此时，该数据被用于两个站点中的重复数据删除了。

以下是 3 个站点的部署示例，每个站点包含多个 PD：



注意：

Fingerprinting 功能必须在源端和目标端的 Container / vstore 中都开启，实现复制间的重复数据删除。

图 11-41. Replication Deduplication

3.2.12 Cloud Connect

云链接（Cloud Connect）功能可以很好的扩展分布式存储原生的 DR / 复制功能（当前支持 Amazon Web Services, 或 AWS）。注意：该功能当前仅适用于备份 / 复制。

该过程非常类似于基于原生的 DR / 复制功能去创建一个远程站点，实际上是一个“云端站点”被创建出来。当此云端站点被创建时，Nutanix 将自动在 EC2（当前为 m1.xlarge）或 Azure 虚拟机（当前为 D3）创建一个单节点的集群作为目标端。

运行在 AWS 上的 Amazon Machine Image (AMI)也是基于跟本地群集相同的 NOS 源代码构建。这就意味着所有原生的复制功能（例如：全局重复数据删除、两地三中心等）可以直接使用。

在这个示例中，有多个 Nutanix 群集使用了云链接，它们可以共享同一个的 AMI 实例 ，或配置一个全新的实例。

云实例的存储是使用由 S3(AWS)或者 BlobStore(Azure)提供的逻辑磁盘，称为“云磁盘”。数据存储为通常的 Egroup，它在对象存储中作为文件存放。

以下是基于 AWS 的云端站点云链接的示意图：

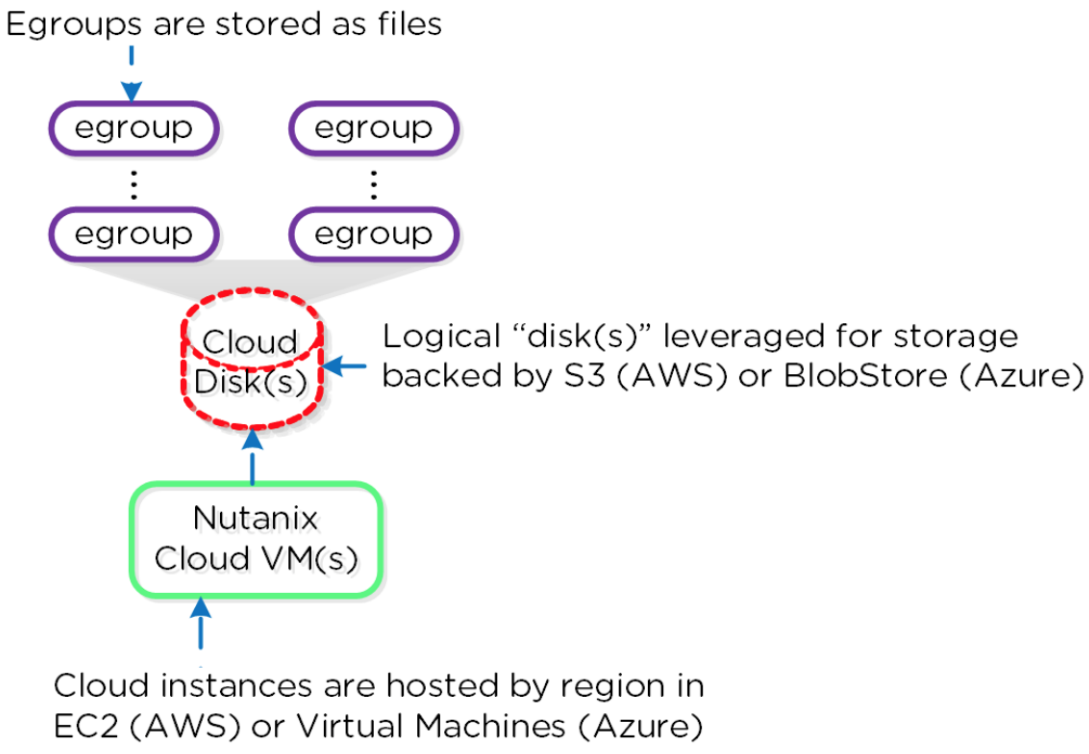


Figure 11-42. Cloud Connect Region

图 11-42. Cloud Connect Region

由于基于 AWS 创建的云端站点类似于 Nutanix 的普通远程站点，群集可将数据复制到多个不同地域，从而获得更高的业务连续性（例如，地域性的大规模停电，仍可保证数据可用）：

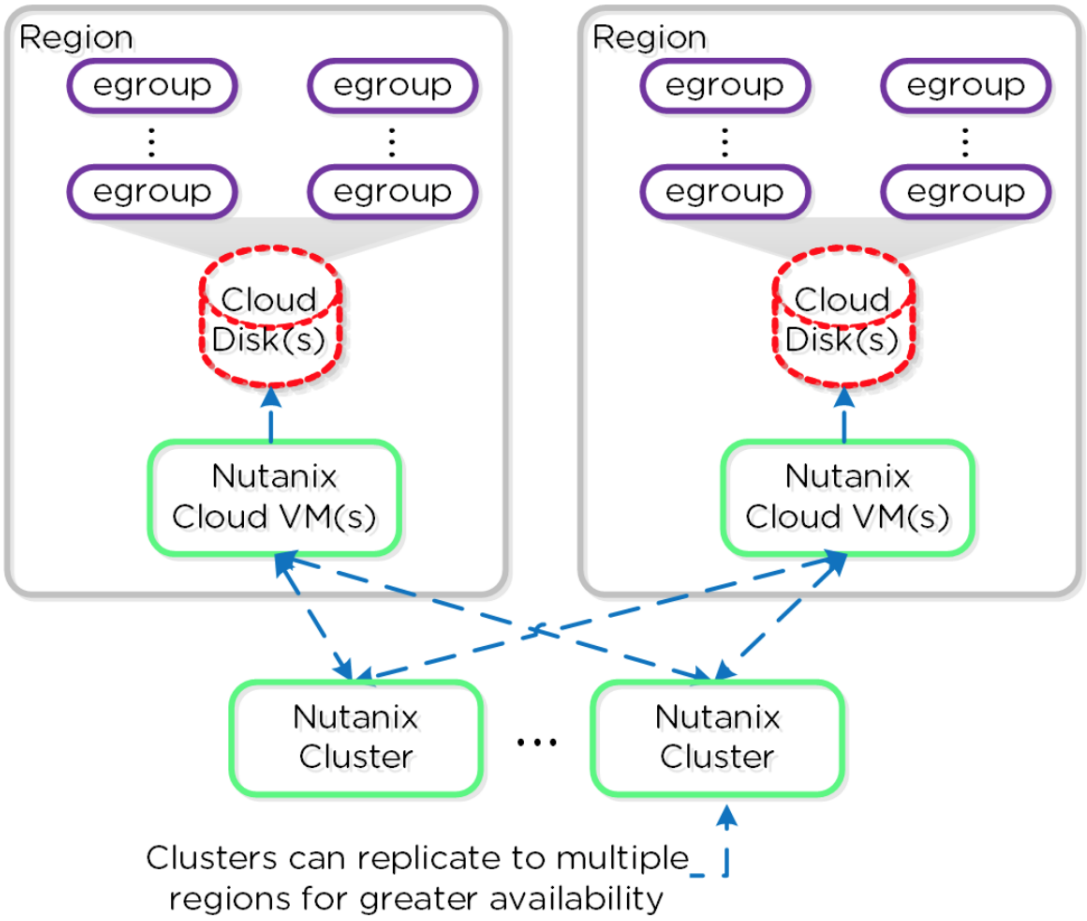


图 11-43. Cloud Connect Multi-region

同样的复制 / 保留策略也可被用于云链接的数据复制之中。当数据或快照失效或过期时，云集群将执行必要的清理动作。

如果复制的频率不高（每天一次或每周一次），甚至乎你可以配置为在复制开始前才启动云实例，并在复制完成后，将其关机。

复制到云区域中的数据也可随时下传、恢复到现有的或新建的 Nutanix 群集中，当然该群集需要配置、链接到了云端站点。

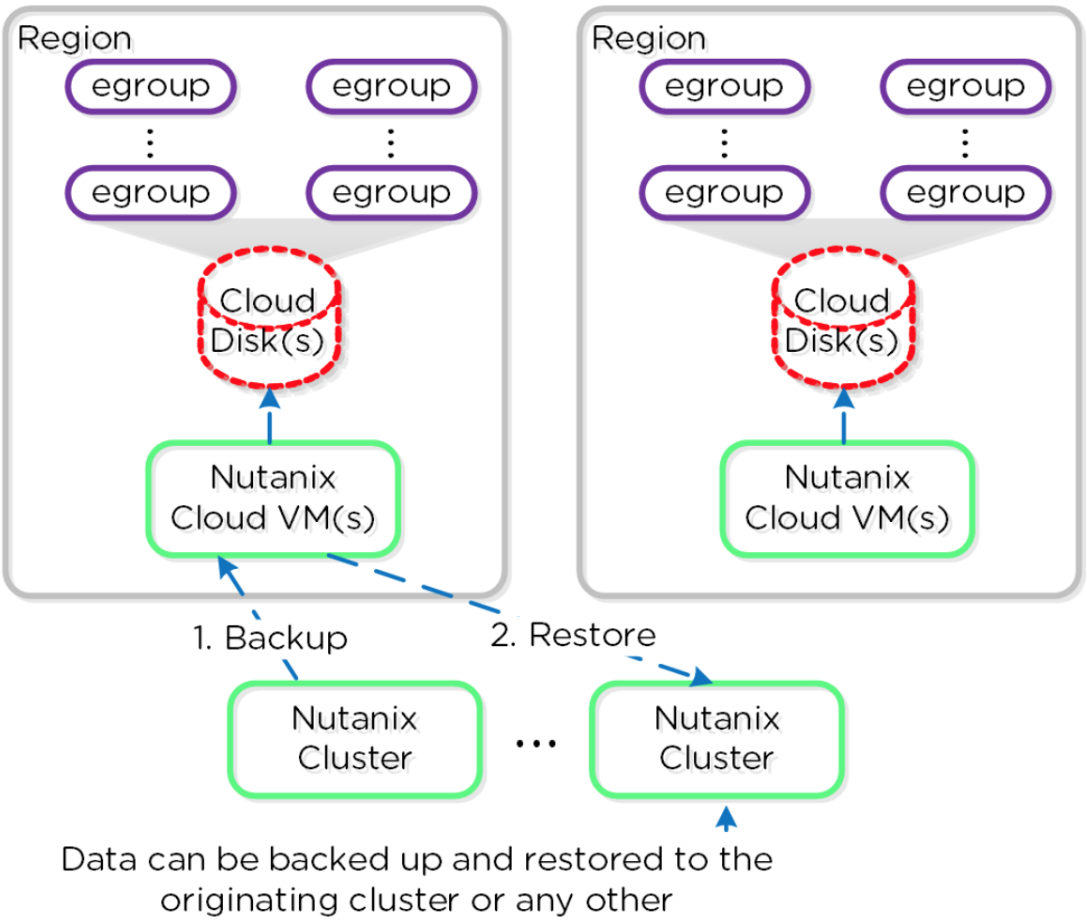


图 11-44. Cloud Connect - Restore

3.2.13 Metro Availability

Nutanix 具有“Stretch Clustering”的能力，允许它的计算和存储群集可以跨越多个物理站点。这种部署架构下，计算集群可以跨越两个站点并共享一个存储池。

这将 VM HA 集群从一个站点扩展到两个站点之间，提供近乎等于 0 的 RTO 和 RPO。

这种部署架构下，每个站点都有自己的 Nutanix 集群，但其中的容器（Container）的写操作被同步复制到远端的站点之中。

以下是此架构的详细示意图：

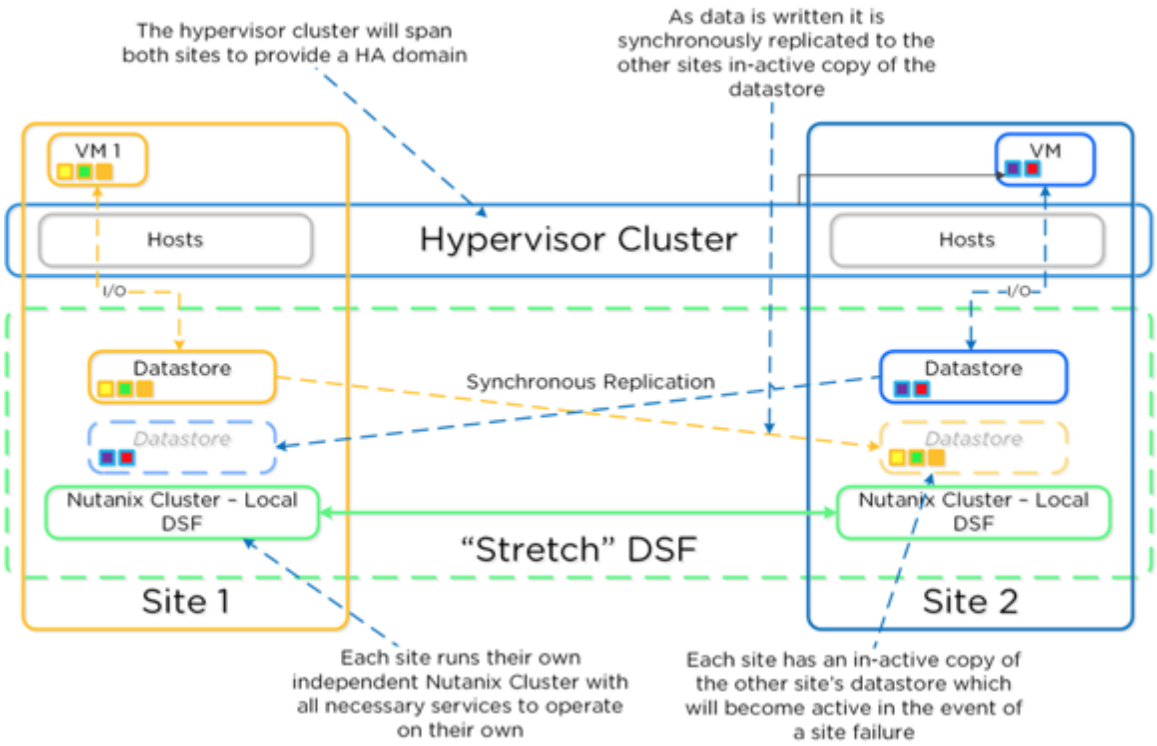


图 11-45. Metro Availability - Normal State

当站点发生故障，HA 将发生切换，VM 将在另一个站点启动。

以下是一个站点发生故障的示意图：

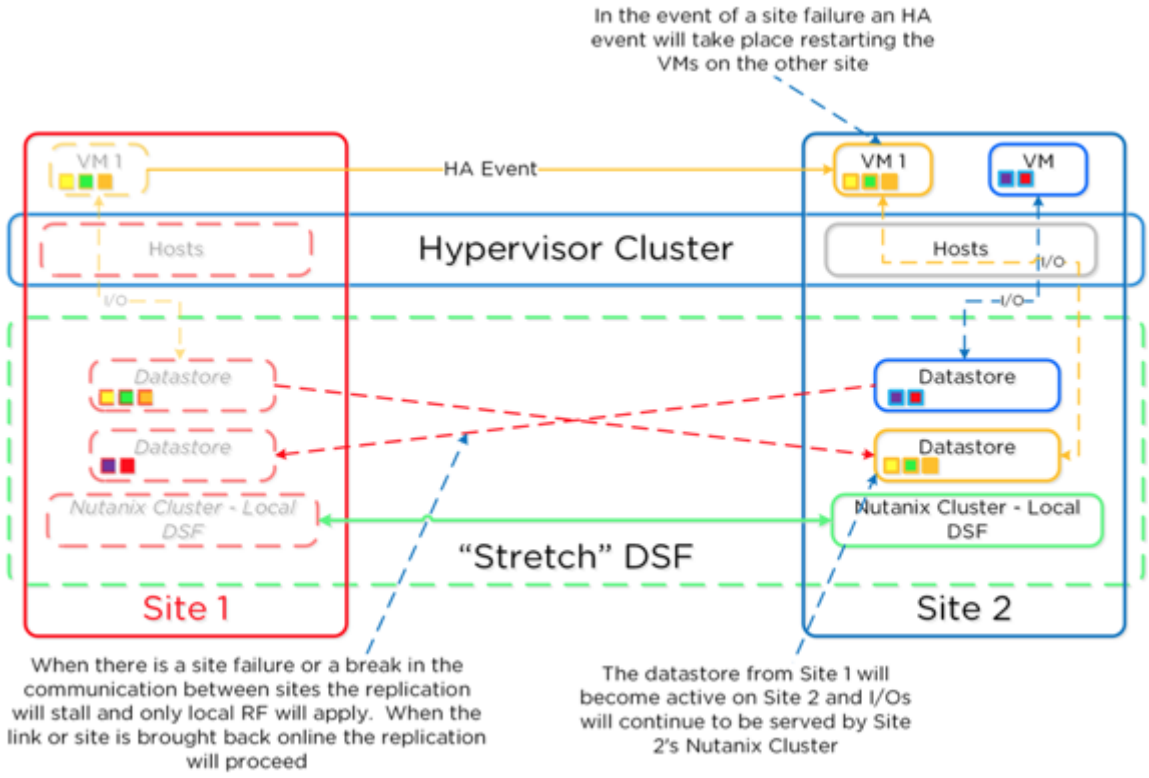


图 11-46. Metro Availability - Site Failure

如果站点间的网络链路故障，集群将分别独立运行。一旦网络链路修复，站点将进行增量同步，并重新恢复数据同步。

以下是网络链路故障的示意图：

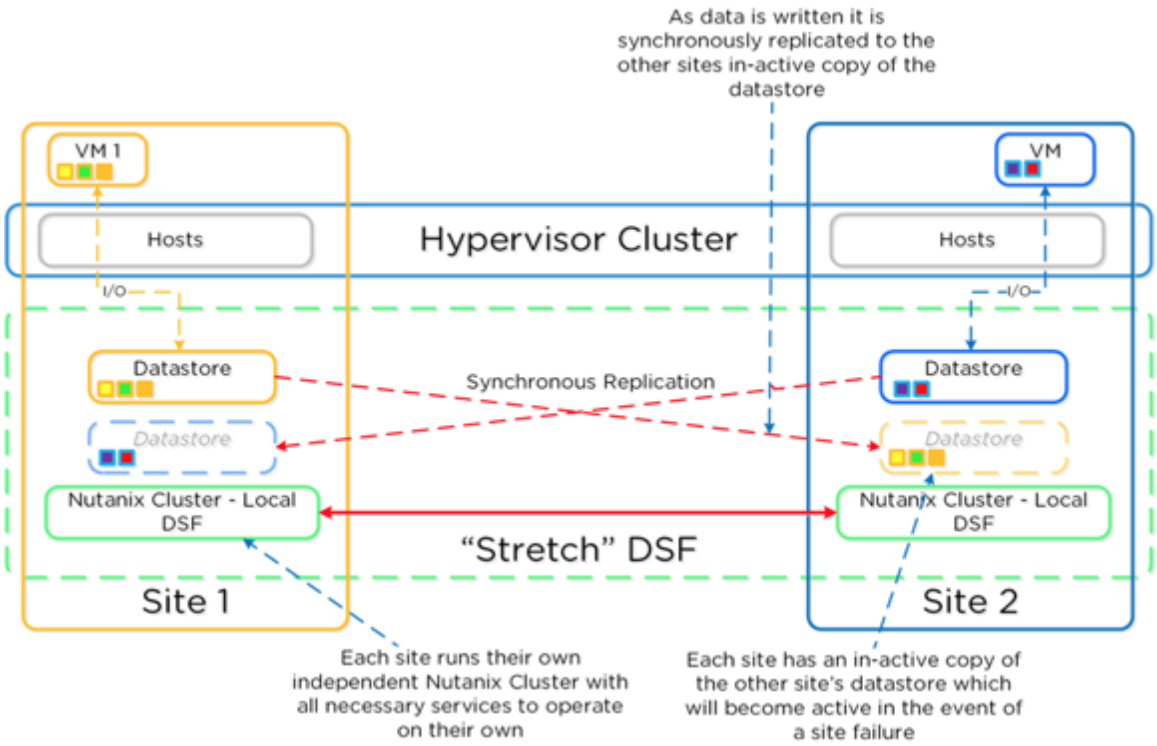


图 11-47. Metro Availability - Link Failure

3.2.14 Volumes API

Acropolis 的 Volumes API 接口可以很好的将内部分布式存储通过 iSCSI 呈现给外部的实例（Guest OS、Physical Hosts、Containers 等）。

这允许任何操作系统访问分布式存储和其存储空间。在此部署方案中，客户机操作系统将绕过虚拟化层（HyperVisor）直接访问 Nutanix 后端存储。

Volumes API 使用的主要场景：

- 共享磁盘
  - ✧ Oracle RAC，Microsoft Failover Clustering 等；
- 特殊应用
  - ✧ 核心数据，要求极低的时延；
  - ✧ Containers，OpenStack 等

- Guest-initiated iSCSI
  - ✧ 裸金属架构需求（Bare-metal consumers）
  - ✧ Vsphere 中部署 Exchange

Volumes API 包含以下部分：

- **Volume Group**：iSCSI 的目标，为适用集中管理、快照和其他应用策略而构建的一组磁盘设备的集合
- **Disks**： Volume Group 中的存储设备（例如 iSCSI 中的 Lun）
- **Attachment**：允许特定的 iSCSI 发起端（initiator IQN）访问此 Volume Group

注意：在后端存储中，一个 VG 的磁盘仅仅是分布式存储中的一个虚拟磁盘（vDisk）。

若要使用 Volumes API,请参照以下步骤：

1、创建新的 Volume Group

2、添加磁盘至 Volume Group

3、添加 iSCSI 发起 IQN 至 Volume Group

-

Example 11-1. Create Volume Group

```
# Create VG
vg.create  <VG  Name>
# Add disk(s) to VG
Vg.disk_create <VG  Name> container=<CTR Name>  create_size=<Disk size, e.g.500G>
 # Attach initiator IQN to VG
Vg.attach_external  <VG  Name><Initiator IQN>
```

以下是在 Nutanix 中的一个 VM 直接挂载后端存储的示意图：

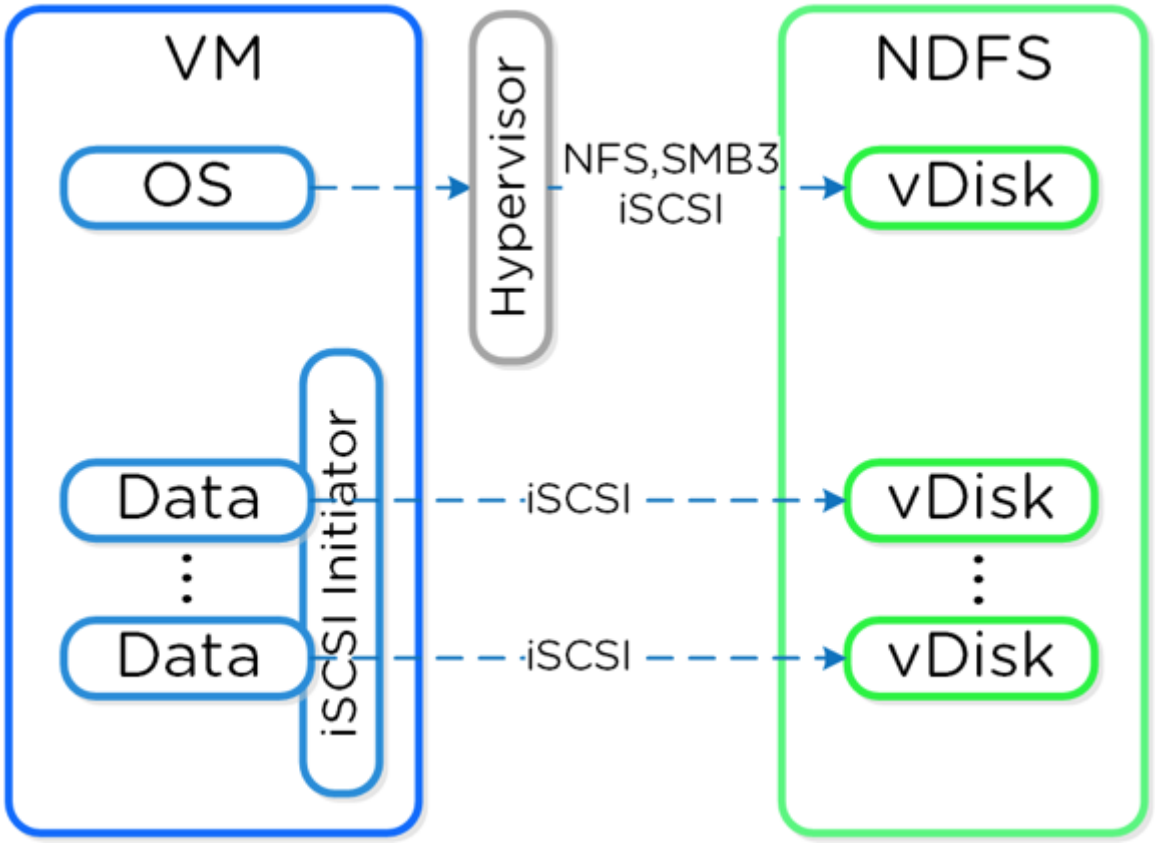


图 11-48. Volume API - Example

在 Windows 部署中，可以利用 Windows MPIO 功能设置 iSCSI 的多路径。这里建议使用默认的“Failover only”，以确认 vDisk 的所有权不会随意切换。

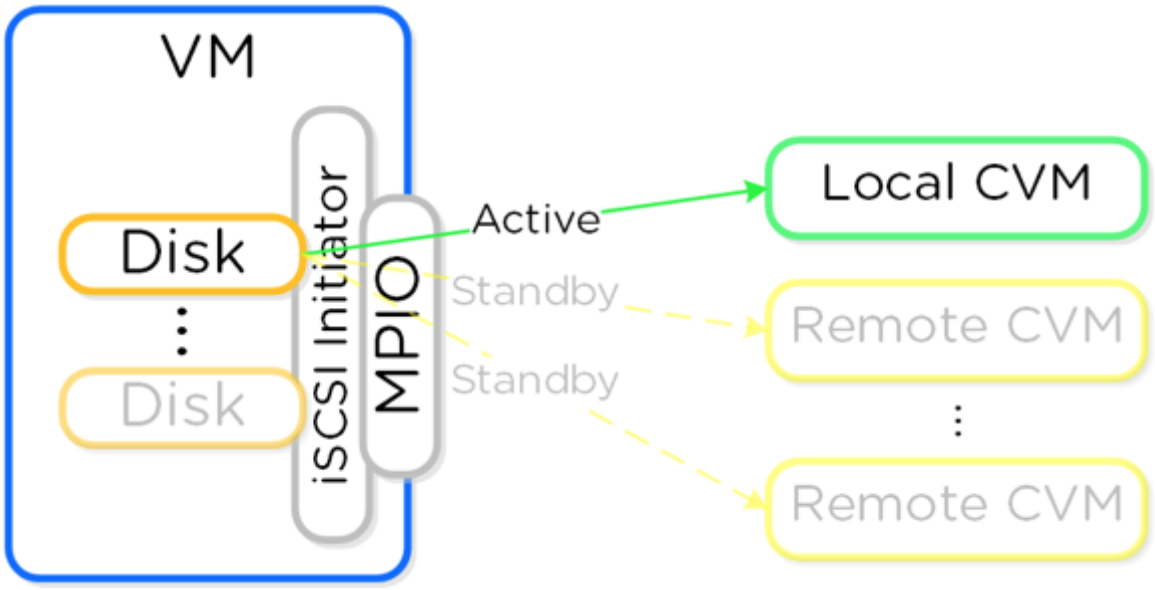


图 11-49. MPIO Example - Normal State

下图中包含多个磁盘设备，每个磁盘都有自己的活动路径链接至本地 CVM：

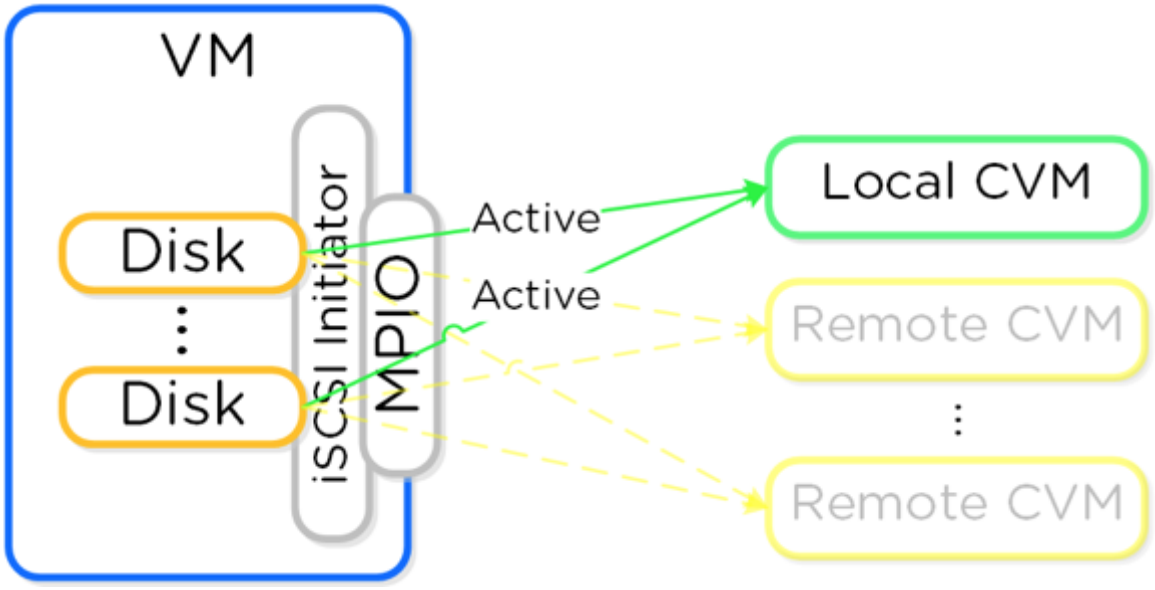


图 11-50. MPIO Example - Multi-disk

如果活动的 CVM 宕机，磁盘的活动路径将发生切换，I/O 即可恢复：

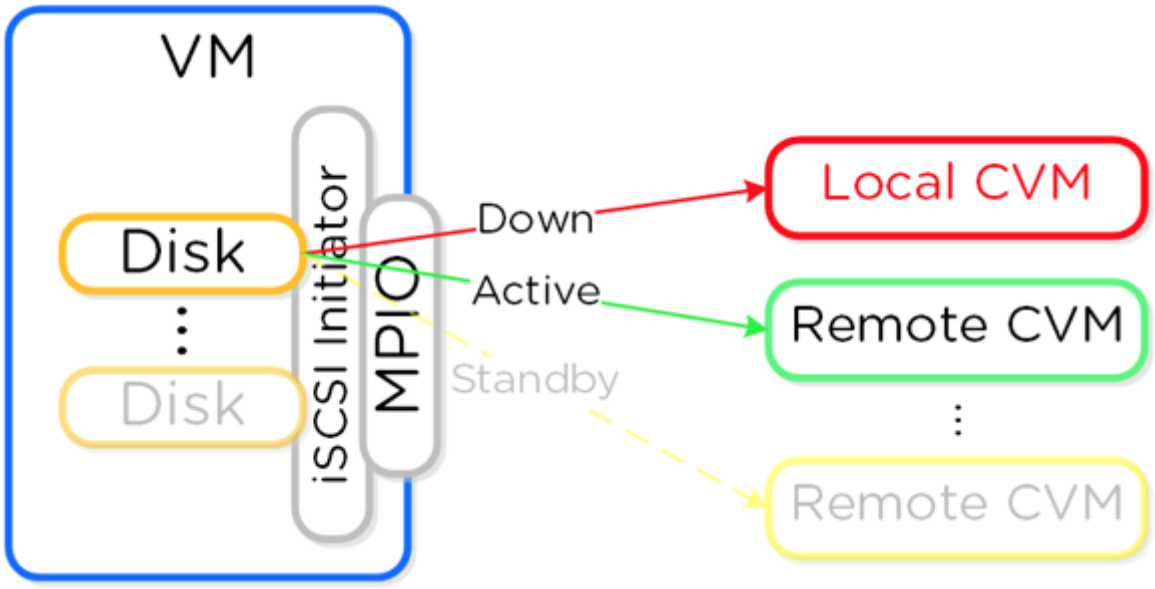


图 11-51. MPIO Example - Path Failure

在测试中，MPIO 花费了大约 15-16 秒以完成路径切换，低于 Windows 磁盘的超时限制（默认为 60 秒）。

如果实际环境中迫切需要使用 Raid 或 LVM，亦可将挂载的磁盘设备加入到动态磁盘组（Dynamic disk）或逻辑磁盘（Logical disk）中：

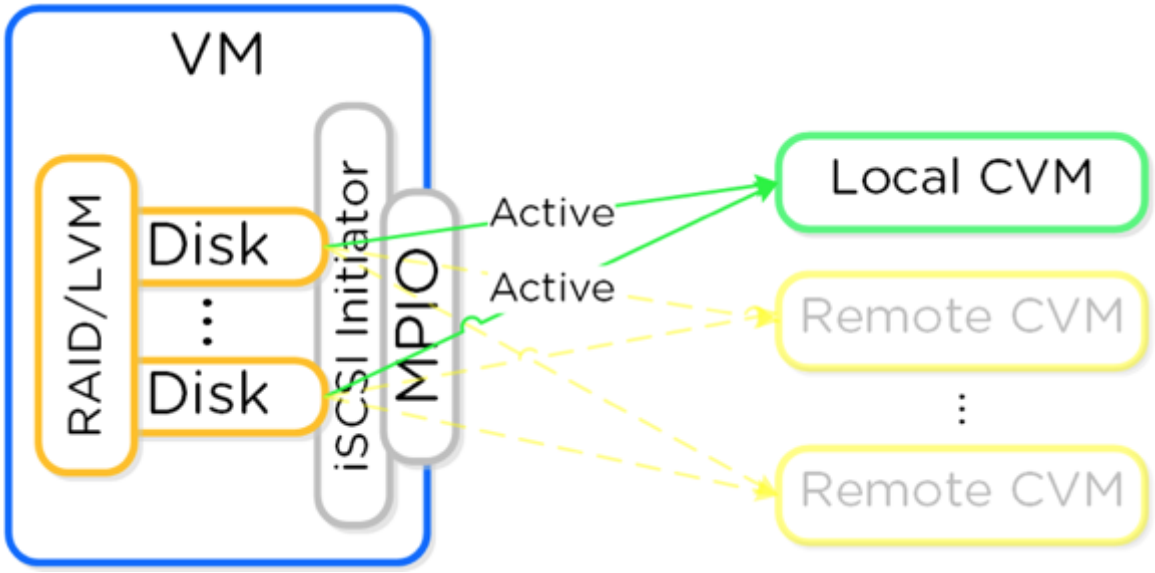


图 11-52. RAID / LVM Example - Single-path

下图中，本地的 CVM 负载过高，部分磁盘的活动链路将切换至其他 CVM 中。这样就可以在多个 CVM 中平衡 I/O 的负载，但是会大幅增加网络的负载：

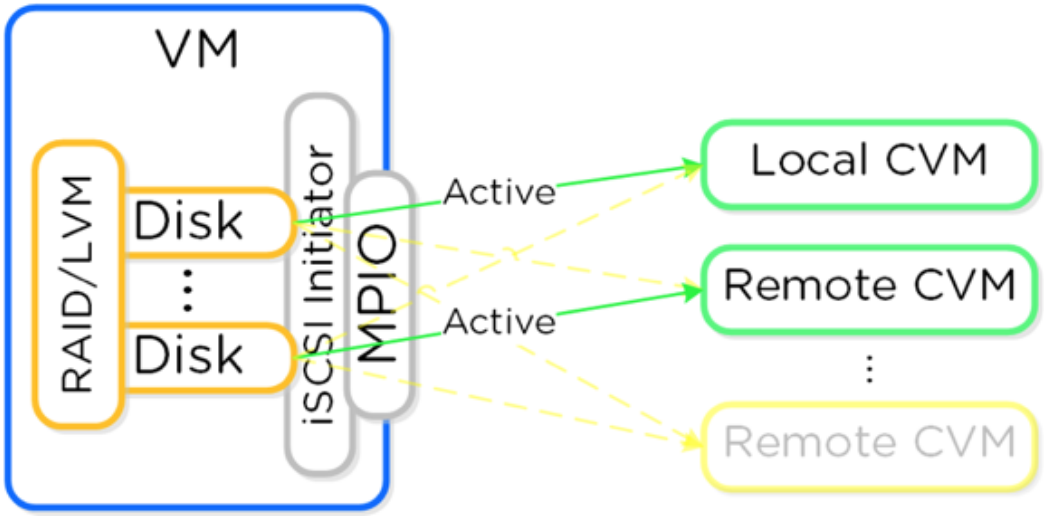


图 11-53. RAID / LVM Example - Multi-path

3.2.15 Networking and I/O

关于视频解说，你可以查看以下链接：（https://youtu.be/Bz37Eu_TgxY)

Nutanix 平台的内部节点间无背板链接，而是依赖于标准的 10GbE 网络进行通讯。运行于 Nutanix 平台上的虚机的所有存储 I/O 都在专用私有网络里由虚拟化监控程序（HyperVisor）处理。虚拟化层接收 I/O 请求，随后将其转发给本地 CVM 的私网 IP。此 CVM 随后将通过外部 IP 及外部的 10GbE 网络把数据远程复制至其他 CVM 中。对于读请求，绝大部分都将由本地 CVM 提供服务，而无需通过外部的 10GbE 网络。这就意味着，只有 DFS 的远程复制和 VM 的网络流量需要用到 10GbE 网络，除非本地 CVM 宕机或跨节点访问远程数据（如 vMotion 后）。当然，集群的其他一些情况也会临时使用到 10GbE 网络，例如：磁盘的负载均衡等。

以下是 VM 使用到的 I/O 路径的示意图，包含内部、外部 10GbE 网络：

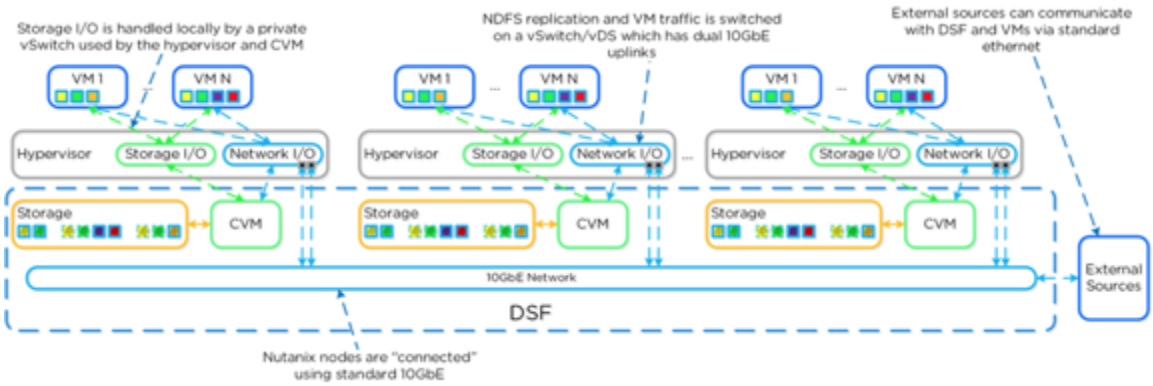


图 11-54. DSF Networking

3.2.16 Data Locality

关于视频解说，你可以查看以下链接：(https://youtu.be/ocLD5nBbUTU)

I/O 和数据的本地化（data locality），是 Nutanix 超融合平台强劲性能的关键所在。正如之前 I/O 路径（I/O Path）章节所述，所有的读、写 I / O 请求都藉由 VM 的所在节点的本地 CVM 所响应处理。VM 的数据都将由本地的 CVM 及其所管理的本地磁盘提供服务。当 VM 由一个节点迁移至另一个节点时（或者发生 HA 切换），此 VM 的数据又将由现在所在节点中的本地 CVM 提供服务。当读取旧的数据（存储在之前节点的 CVM 中）时，I / O 请求将通过本地 CVM 转发至远端 CVM。所有的写 I / O 都将在本地 CVM 中完成。DFS 检测到 I/O 请求落在其他节点时，将在后台自动将数据移动到本地节点中，从而让所有的读 I/O 由本地提供服务。数据仅在被读取到才进行搬迁，进而避免过大的网络压力。

以下展示了数据是如何随着 VM 而迁移的：

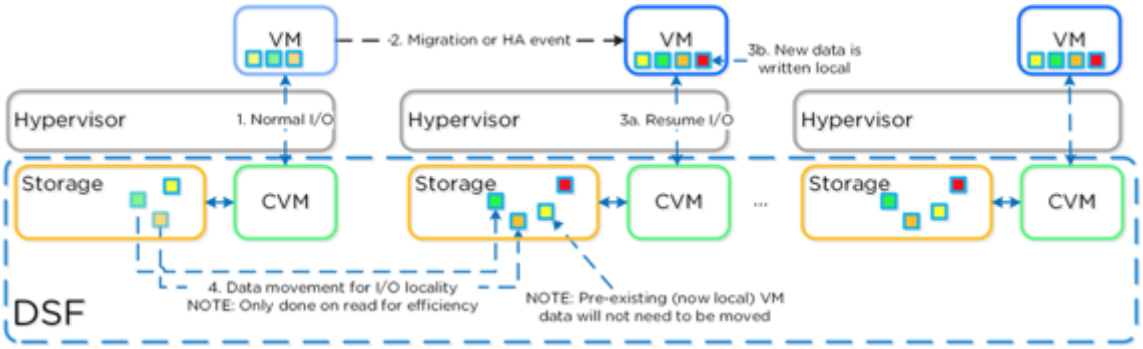


图 11-55. Data Locality

数据迁移条件：

Data Locality 是一个实时的操作过程，当满足以下条件时将搬迁此数据

3.2.17 Shadow Clones

关于视频解说，你可以查看以下链接：(https://youtu.be/oqfFDMYQFJg)

Acropolis DSF 有一项功能称之为“Shadow Clones”，允许在大量并发读取时分布式的缓存特定的 vDisk 或 VM 数据。最好的例子莫过于在 VDI 部署中，大量的链接克隆（Linked Clones）需要读取模版虚机（A central master or ‘Base VM’）。在 Vmware View 中，这被称之为磁盘副本（replica disk），并被所有链接克隆所读取。在 XenDesktop 中，这被称之为 MCS 主虚机（MCS Master VM）。这也可以被用于其他的大量并发读取的环境中（例如：部署虚拟服务器等）。数据和 I/O 本地化是将 VM 性能最大化，以及 DSF 的架构特性的关键之所在。

关于 Shadow Clones，DSF 将监控 vDisk 的“本地化”访问趋势。如果有超过 2 个以上 CVM（包括本地的 CVM）同时访问，且全部都是读请求，该 vDisk 将被标记为只读。一旦 vDisk 被标记为只读，则该 vDisk 可被缓存至所有 CVM 的本地，实现本地的数据读取（或可称之为基于 vDisk 的 Shadow Clones）。这样就可以让所有的 VM 在本地就可以读取模版虚拟的 vDisk 了。在 VDI 中，这就意味着所有的磁盘副本可以被缓存到所有节点的本地，所有的读请求将由本地节点给予响应。

注意：数据将在读取后执行搬迁，因此不会大量增加网络负载，并且可以大幅提升缓存的利用率。

如果模版虚机被更改，则 Shadow Clones 将停止，并丢弃。Shadow Clones 默认被开启（如 NOS4.0.2），你可以通过以下命令行开启或关闭此功能：

```
ncli cluster edit-params enable-shadow-clones=<true/false>
```

下图展示的 Shadow Clones 是如何工作及允许分布式缓存的：

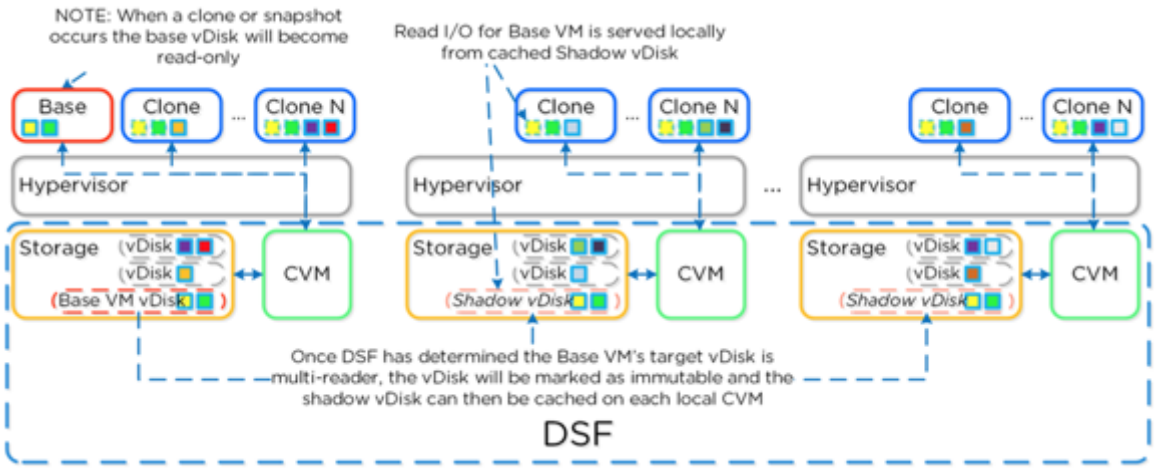


图 11-56. Shadow Clones

3.2.18 Storage Layers and Monitoring

Nutanix 平台监控着存储中的多个层面，从 VM 或 Guest OS 到物理磁盘。了解各个层面以及它们之间的关系对于后续的监控、运维、操作等是很重要的。

下图显示各个层面相关的监控事项及监控的颗粒度等：

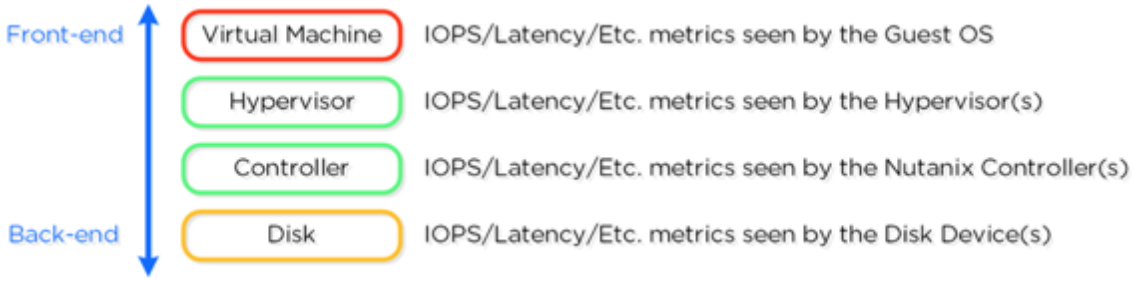


图 11-57. Storage Layers

Virtual Machine Layer

- 主要角色： 通过虚拟化层获取 VM 的状态参数
- 描述：通过虚拟化层直接获取 VM 或 Guest 的状态参数，标示出 VM 当前的性能，以及应用系统当前的 I/O 性能。
- 使用场景：故障定位或 VM 性能分析

HyperVisor Layer

- 主要角色： 虚拟化层的状态参数
- 描述：通过虚拟化层获取当前的最为精准的状态参数。这些数据包含整个群集或其中任一节点的状态。这一层面上可以提供各个平台中的性能参数，可被用于更多的的分析场景。当然，你也可以通过监控不同的颗粒度，获取更加精准的指标。这其中就包含 Nutanix CVM 的缓存命中率。
- 使用场景：提供更有价值的详细的性能指标

Controller Layer

- 主要角色： Nutanix 控制器的状态参数
- 描述：此状态参数由 Nutanix 控制器虚机（例如：通过 Stargate 2009 端口）直接提供，它将标示出前端采用了 NFS / SMB / iSCSI 中的哪种链接的方式，以及后端发生了哪些操作（例如：ILM、磁盘负载均衡等）。你可以查看单个控制器虚机或整个控制器集群的相关参数。这些状态参数中部分数据将与虚拟化层的提供的数据相吻合，另外，数据中还将包括后端的操作（例如：ILM、磁盘负载均衡等），当然这其中就包含 Nutanix CVM 的缓存命中率。在某些情况下，IOPS 的指标可能会有差异，因为 NFS / SMB/iSCSI 客户端会将大的 IO 进行拆分。但它们表现出来的带宽应该是一致的。
- 使用场景：类似于 HyperVisor，却能够更好监控后端操作

Disk Layer

- 主要角色： 磁盘设备的状态参数
- 描述：此状态参数由物理磁设备直接提供（通过 CVM），标示出了后端的操作。其中包括 I/O 在磁盘操作时命中 Oplog 或者 Extend Store。可以监控到单块磁盘、某个节点的磁盘，或集群中的全部磁盘。一般情况下，对磁盘的操作包括 IO 的写入及未在缓存中命中的读操作。任何在缓存中命中的读操作，都将不计入磁盘的监控数据中。
- 使用场景：可以查看有多少 IO 操作落到了缓存或磁盘中

保存期限

以上状态信息，在本地的 Prism 中仅保存 90 天。如使用 Prism Central 和

3.3 Application Mobility Fabric

更多内容，随后更新！

3.4 Acropolis Hypervisor

3.4.1 节点架构（Node Architecture）

在 Acropolis 虚拟化的部署中，控制器虚机（CVM）以 VM 形式运行，并通过 PCI 直通方式访问磁盘。这样就可以让 CVM 绕过虚拟化层直接控制 PCI 设备。Acropolis 虚拟化层是基于 CentOS KVM 改写的。

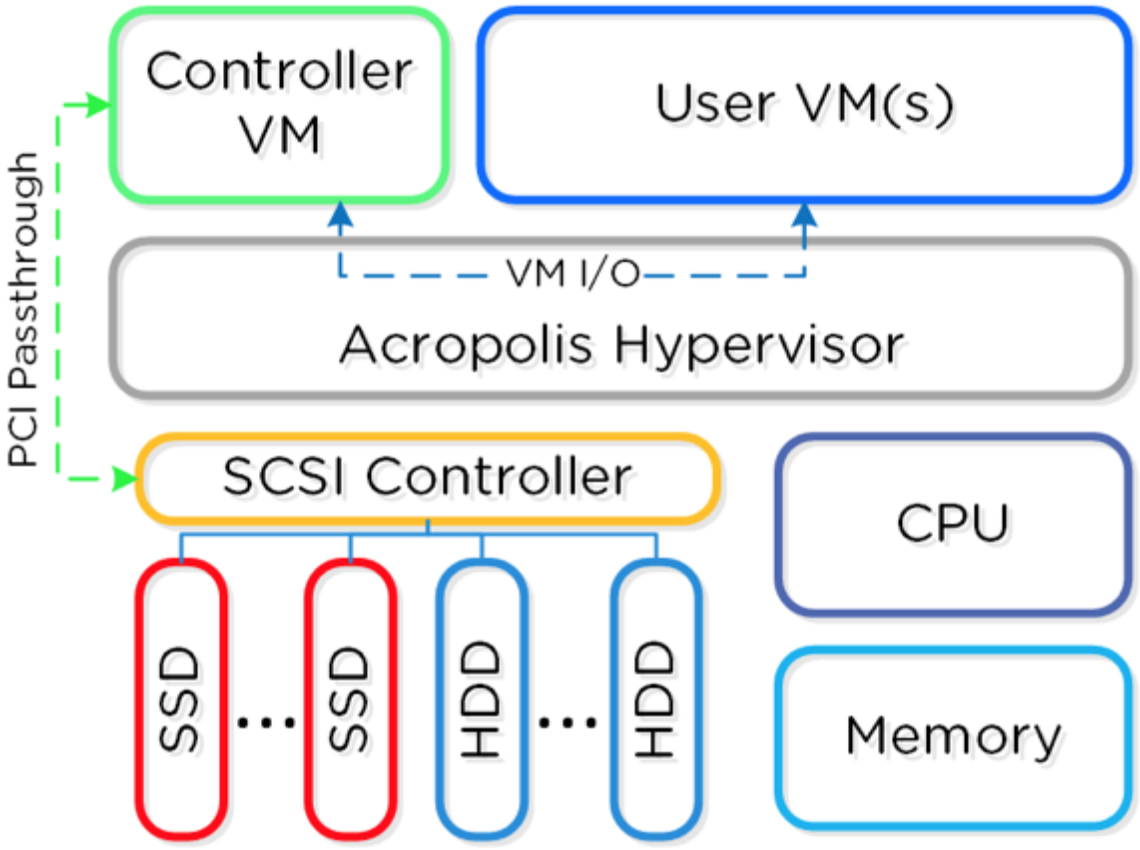


图 13-1. Acropolis Hypervisor Node

Acropolis 构建在 CentOS KVM 基础之上，加入了很多新的功能，例如：HA，在线迁移（Live Migration）等。

Acropolis 是经过“微软服务器虚拟化认证计划”（Microsoft Server Virtualization Validation Program）认证的，可以用于运行微软的操作系统和应用。

3.4.2 KVM 架构（KVM Architecture）

KVM 中包含以下主要组成：

- KVM-kmod
  - KVM 内核

- Libvirtd
  - API 接口，针对 KVM 和 QEMU 的监控、管理工具。
    - Acropolis 通过 libvirtd 与 KVM / QEMU 进行通讯。
- Qemu-kvm
  - 一个“模拟器”（machine emulator），使得各个虚拟机能够独立运行。

    Acropolis 通过它来实现硬件的虚拟化，并使得 VM 以 HVM 的形式运行。

以下是各个组成的逻辑示意图：

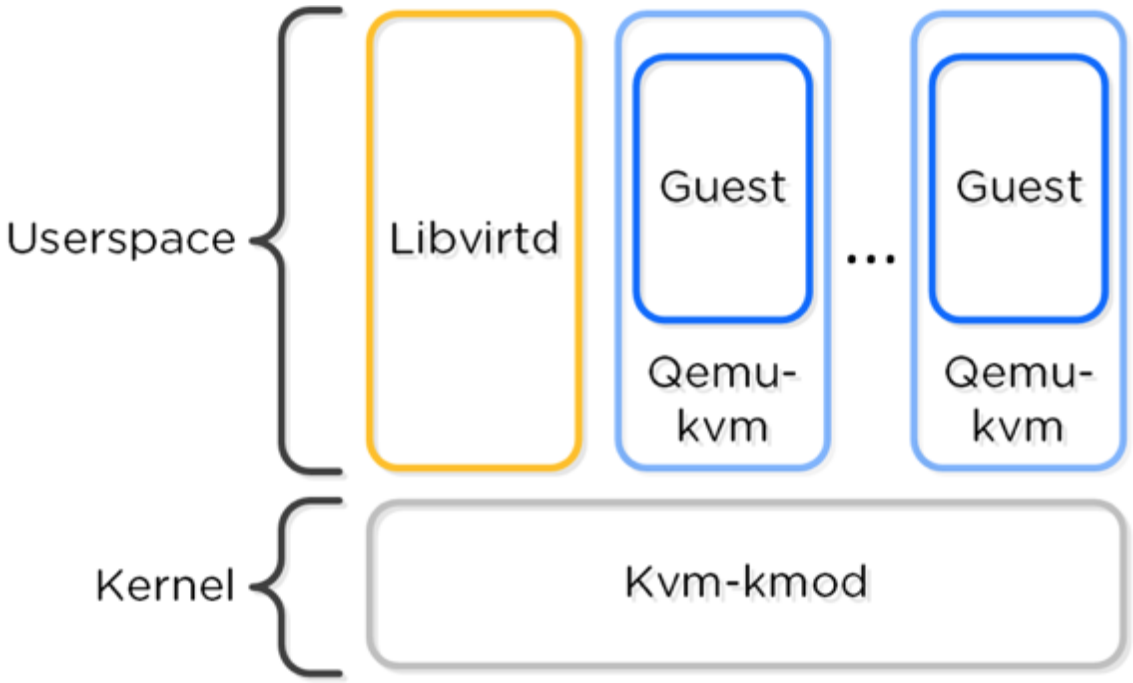


图 13-2. KVM Component Relationship

Acropolis 通过 libvirtd 与 KVM 进行通讯

处理器兼容性

类似于 Vmware 的 Enhanced vMotion Capability（EVC），它允许 VM 在不同代次的处理器间进行迁移。Acropolis 将检测群集中代次最老的处理器，并把所有的 QEMU 限定在此级别上。这样就可以允许不同代次的处理器进行混用，并确保主机之间可以实现 VM 的在线迁移。

3.4.3 配置的最大范围（Configuration Maximums and Scalability）

以下是配置的最大限制及范围

- 群集最大数量（Maximum cluster size）: N/A – 跟 Nutanix 群集一样

- 每个 VM 的最大虚拟 CPU（Maximum vCPUs per VM）: 每台主机的最大物理核数
- 每个 VM 的最大内存（Maximum memory per VM）: 2TB
- 每个节点的最大 VM 数量（Maximum VMs per host）: N/A – 取决于内存的分配限制
- 每个集群的最大 VM 数量（Maximum VMs per cluster）: N/A –取决于内存的分配限制

3.4.4 网络（Networking）

Acropolis 虚拟化使用 Open vSwitch（OVS）为虚机提供网络服务。可以通过 Prism 或 ACLI（Acropolis 命令行）配置 VM 的网路，每个虚拟网卡链接一个虚拟网口（tap interface）

OVS 架构示意图如下：

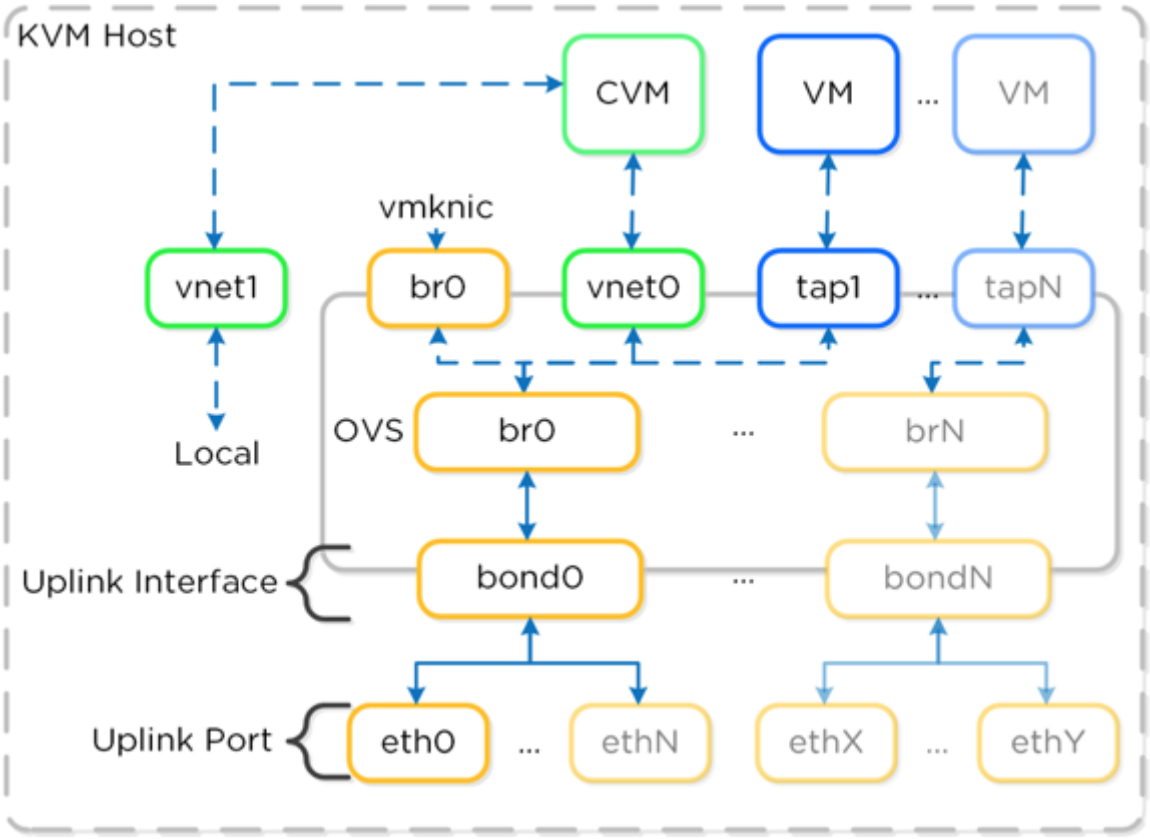


图 13-3. Open vSwitch Network Overview

3.4.5 工作原理（How It Works）

3.4.5.1 **iSCSI Multi-pathing**

每台 KVM 主机都运行一个 iSCSI redirector 守护进程，通过 NOP OUT 命令检测集群的 Stargate 的健康状态。

QEMU 配置 iSCSI redirector 作为一个 iSCSI 目标端。一旦发起登陆请求，redirector 将把 iSCSI 定向至一个健康的 Stargate（本地的 Stargate 优先）。

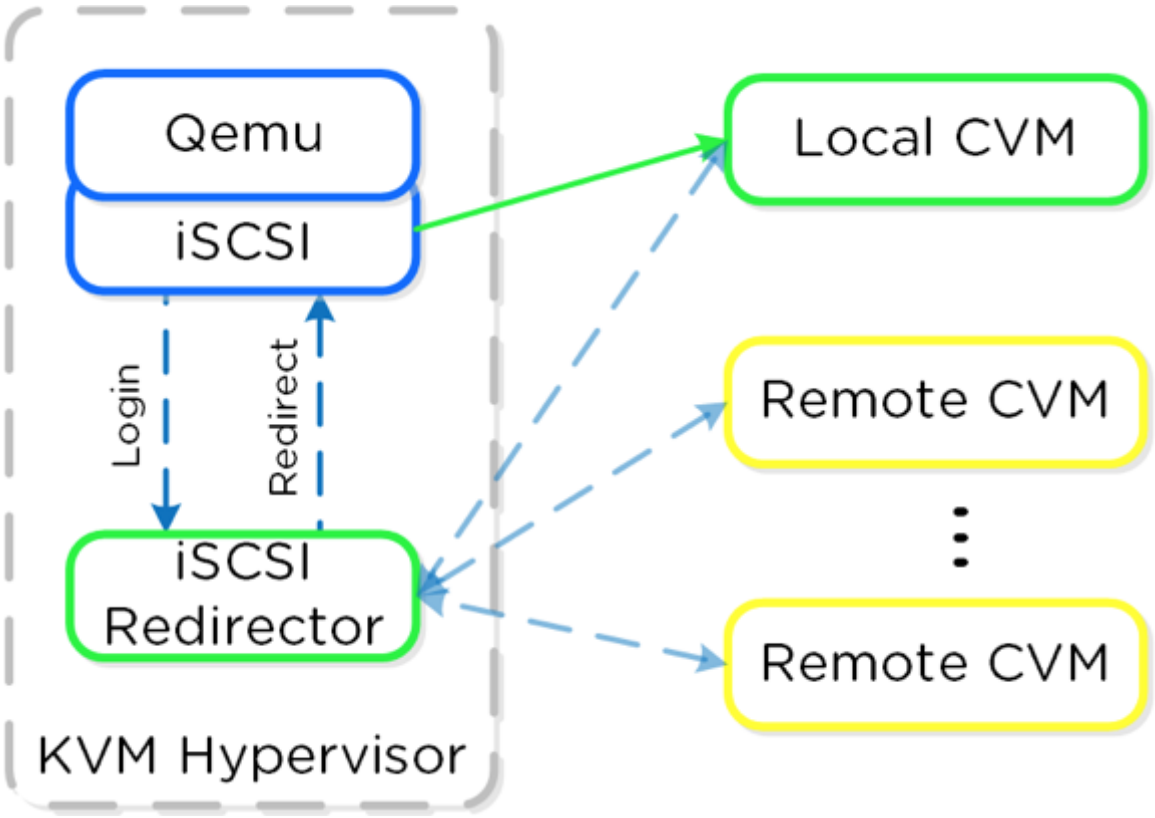


图 13-4. iSCSI Multi-pathing - Normal State

如果当前正在使用的 Stargate 宕机（对 NOP OUT 命令无响应），iSCSI redirector 将把本地的 Stargate 的标示为不健康。当 QEMU 重新发起 iSCSI 登陆，redirector 将重定向至其他的健康的 Stargate。

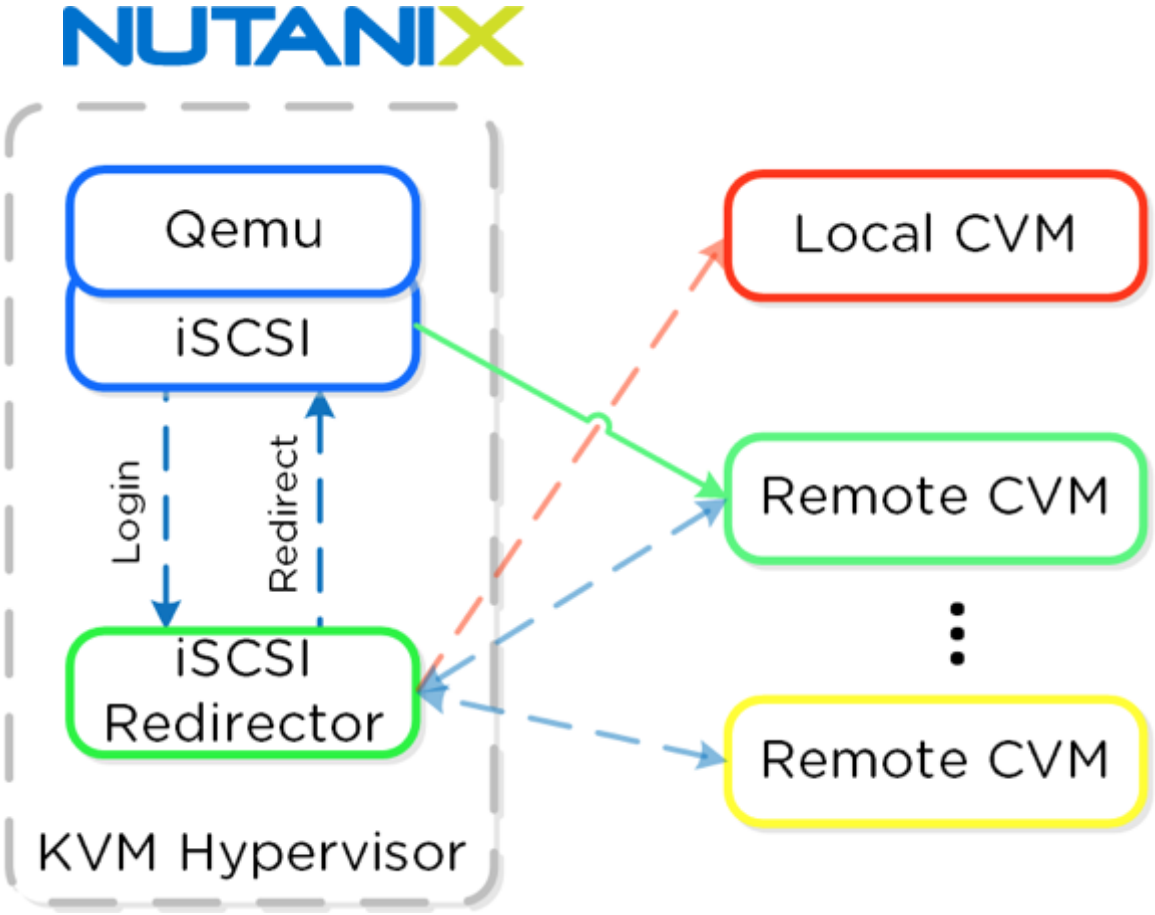


图 13-5. iSCSI Multi-pathing - Local CVM Down

一旦本地的 Stargate 恢复（重新响应 NOP OUT 的命令），iSCSI redirector 将杀死所有连接至远程 Stargate 的 TCP 进程。QEMU 会重新发起 iSCSI 登陆，并重定向至本地的 Stargate。

3.4.5.2 IP 地址管理（IP Address Management）

Acropolis IP 地址管理（IPMI）可以提供 DHCP 服务，并分配 IP 地址给 VM。底层是由 VXLAN 和 OpenFlow 来响应 DHCP 服务。

以下是 Nutanix IPMI 的 DHCP 服务示意图，其中 Acropolis Master 运行在本地：

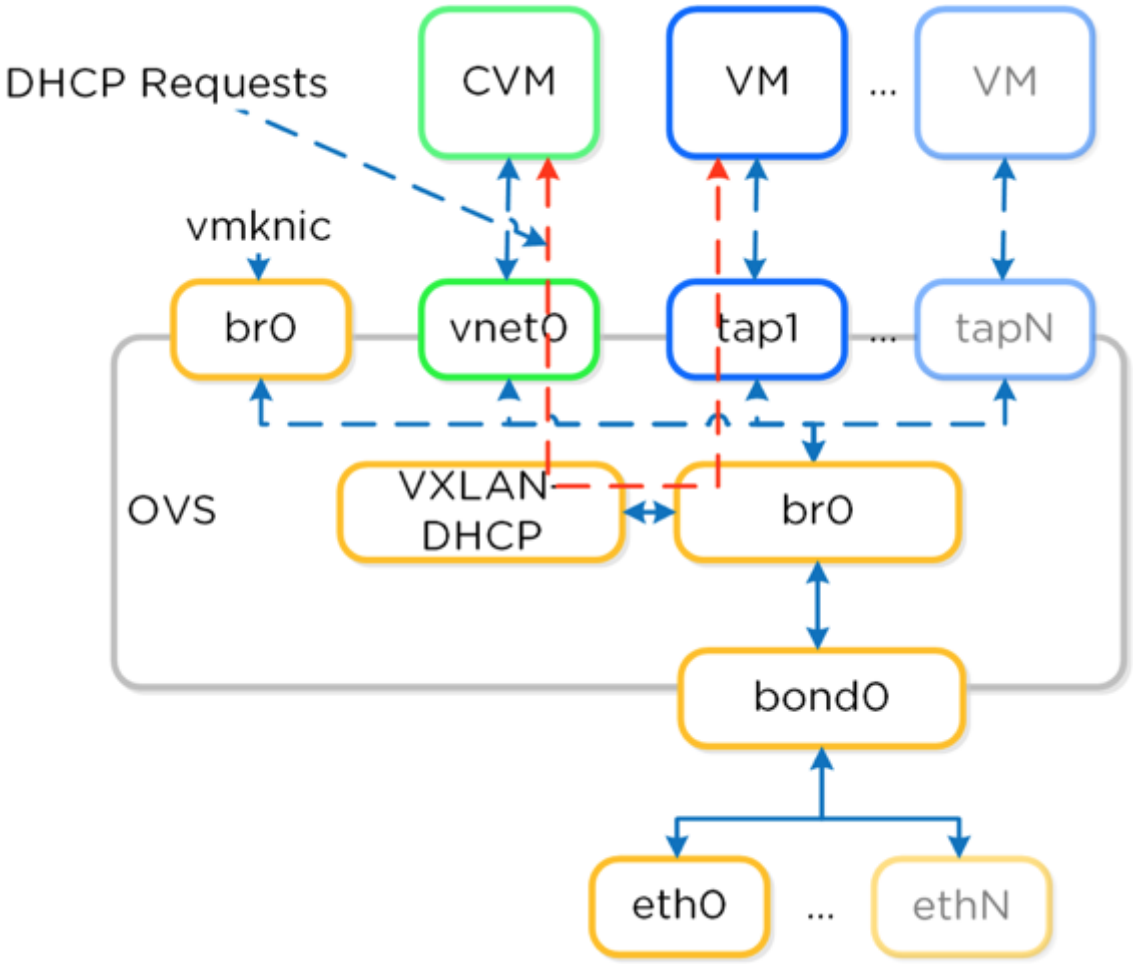


图 13-7. IPAM - Local Acropolis Master

如果 Acropolis Master 运行在远端，每个节点都会有 VXLAN 隧道跨网络提供服务。

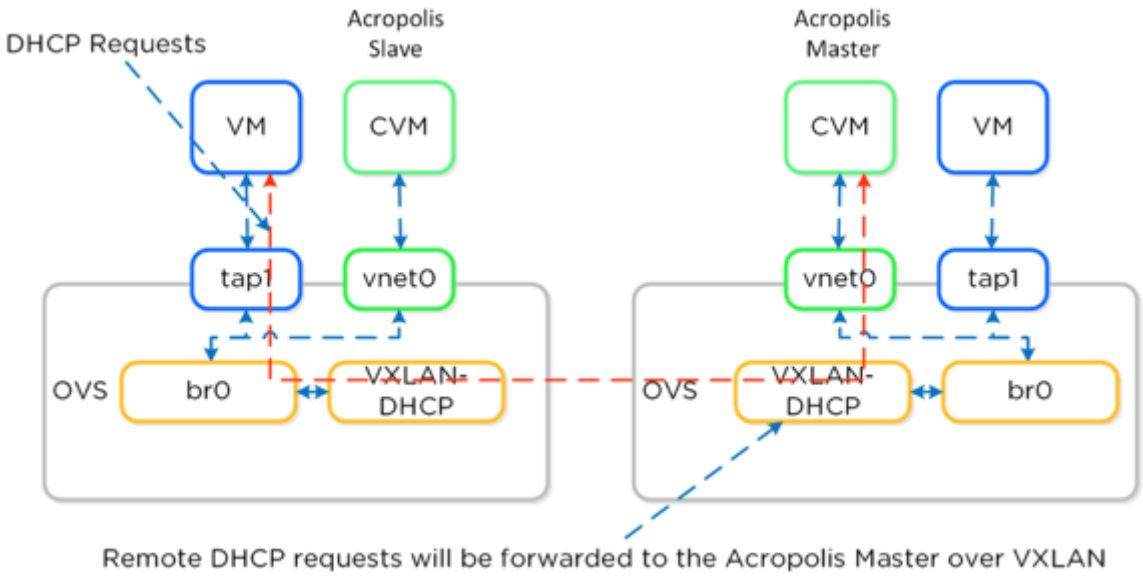


图 13-8. IPAM - Remote Acropolis Master

在一个不受管理的网络场景下，传统的 DHCP / IPAM 也可以继续被使用。

3.4.5.3 VM High Availability (HA)

Acropolis 的 VM HA 可以确保当主机或 Block 掉电时，VM 的持续运行。当某个主机宕机时，VM 将在集群中某个健康节点中重新启动。其中，Acropolis Master 负责该 VM 的重启操作。

Acropolis Master 通过 Libvirt 监测节点的健康状况：

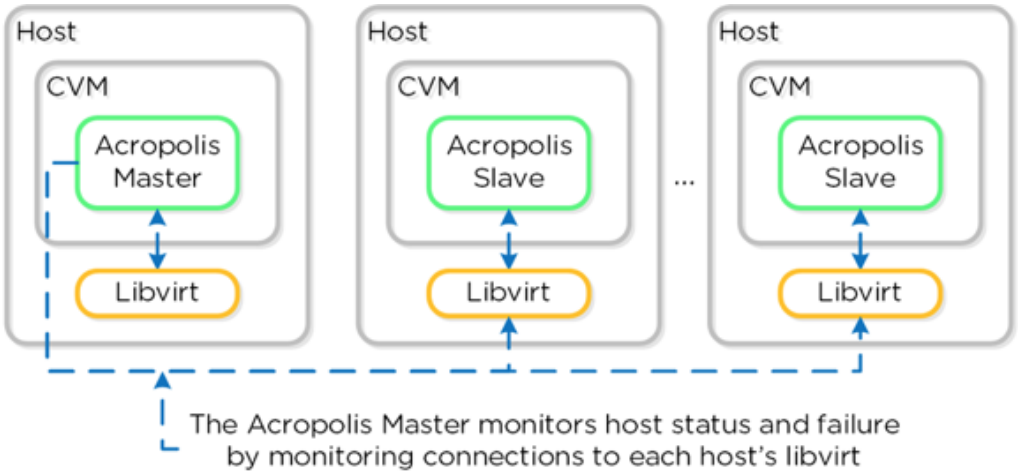


图 13-9. HA - Host Monitoring

如果 Acropolis Master 被隔离、宕机，新的 Acropolis Mater 将在当前健康的节点中被选出。如果 Cluster 网络被隔离（节点之间均无法互通），仲裁机制将不发生切换，VM 将在原有节点上重启。

默认的 VM 重启策略

当一个主机宕机时，默认 AHV 集群将尽可能的去重新启动 VM。在此模式下，当某一主机不可用时，剩余的健康主机将尽其所能的去启动之前运行的

这里有两种资源保留形式供 HA 选择：

- Reserve Hosts
  - 预留 X 个主机，X 为允许故障的主机数量（例如：1、2）
  - 默认用于所有主机配置相同数量的内存时

- Reserve Segments
  - 跨整个集群的 N 个主机预留部分资源。这取决于集群的容错级别、集群中的主机数量、虚拟机的大小等
  - 默认用于所有主机配置不同数量的内存时

专家提示：

以下情况下使用 Reserve Hosts：

- 群集中的节点为同构的（所有的主机配置相同数量的内存）
- 对稳定性的要求高于性能

以下情况下使用 Reserve Segments：

- 群集中的节点为异构的（所有的主机配置的内存数量不同）
- 对性能的要求高于稳定性

将针对这两个选项，诠释如下。

Reserve Hosts

默认的，允许故障的主机数与群集 FT 的级别相同（例如：FT1/RF2 时为 1，FT2/RF3 时为 2）。通过 ACLI 可以对其进行修改。

> 专家提示：
>
> 你可通过以下命令行，手工修改预留的主机数：

以下是 Reserve Hosts 的示意图：

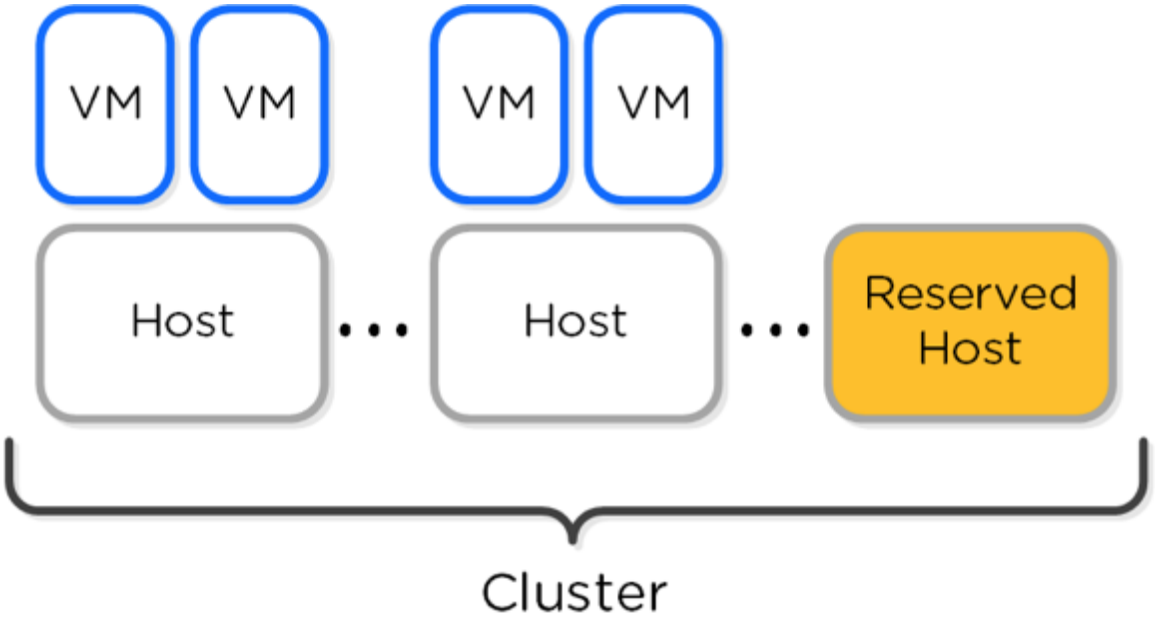


图 13-10. HA - Reserved Host

如果主机发生故障，VM 将在预留的主机中重启：

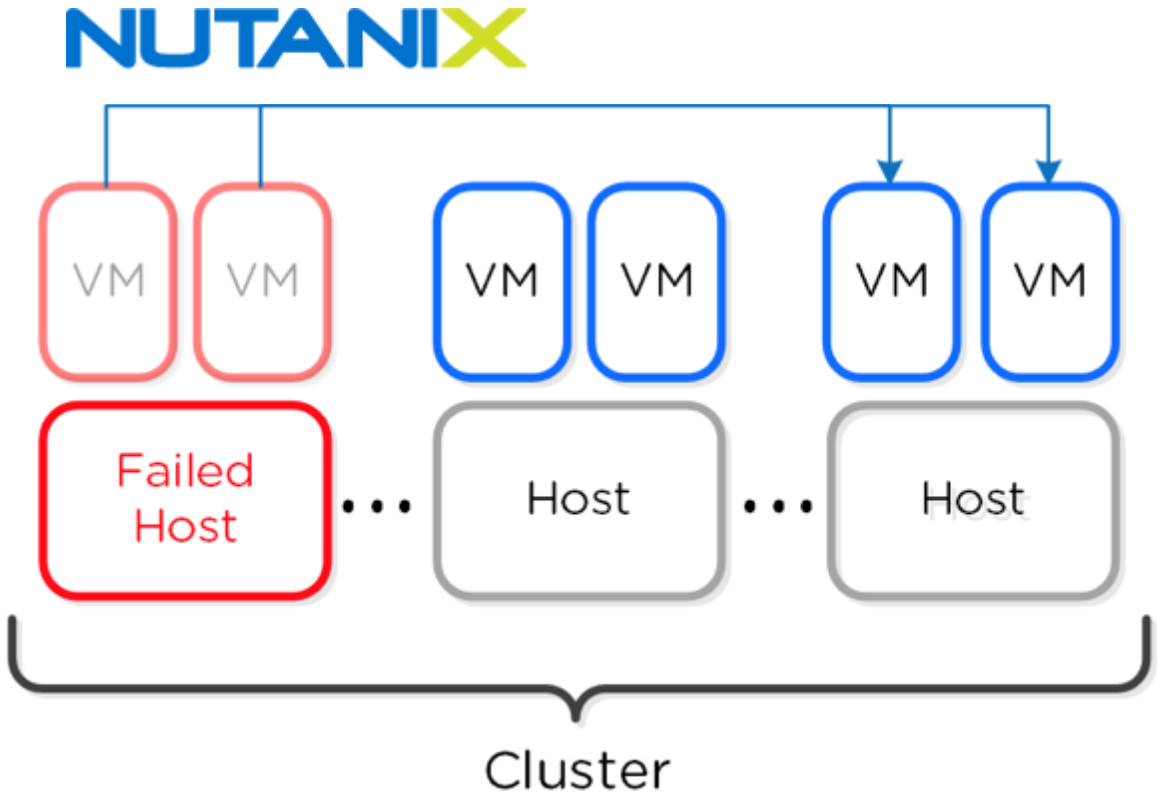


图 13-11. HA - Reserved Host - Fail Over

如果故障的主机被修复，则 VM 将回迁至原有的主机，以减少因为 Data Locality 导致的数据搬迁：

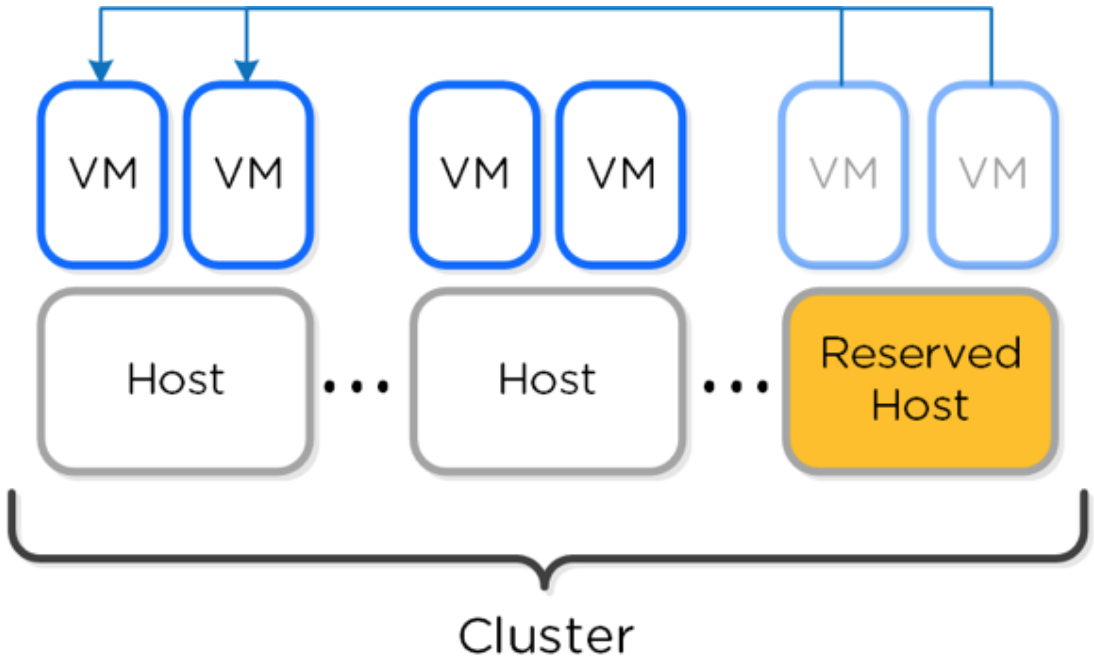


图 13-12. HA - Reserved Host - Fail Back

Reserve Segments

Reserve Segments 将预留资源分布在集群的所有主机中。这种情况下，每个主机都会贡献一部分资源用于 HA。这将确保当一台主机宕机时，集群有足够的资源用于故障切换并重启 VM。

专家提示：

当使用 Reserve Segments 时，请确保各个主机的负载相对均衡。这样可以

以下是 Reserve Segments 的示意图：

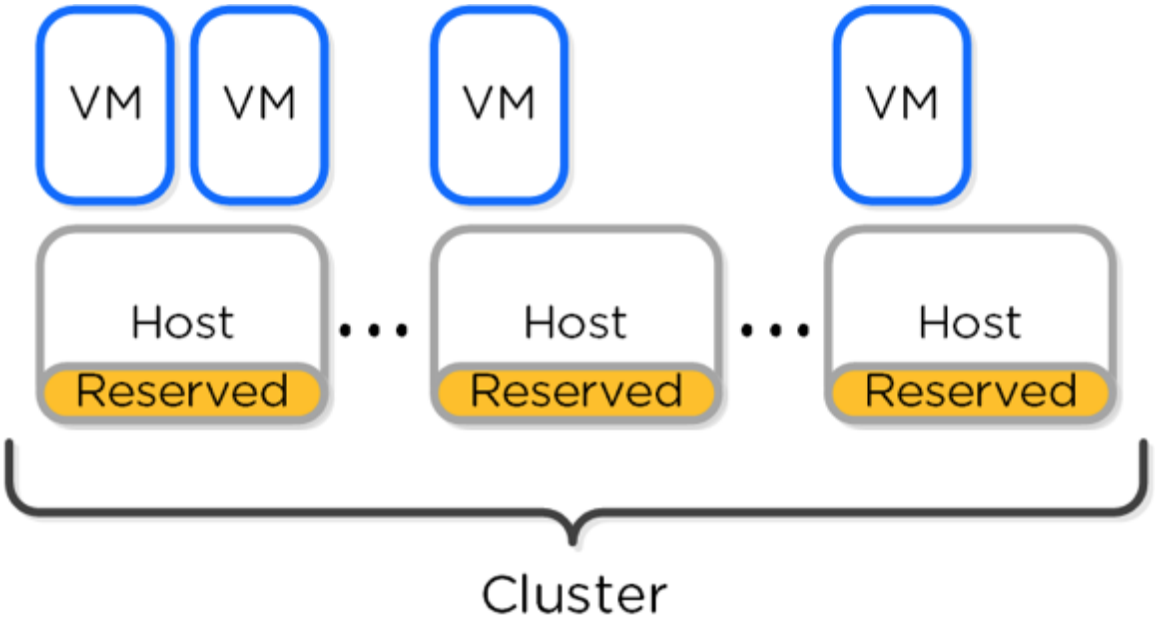


图 13-13. HA - Reserved Segment

如果主机发生故障，集群将在剩余健康的主机中重启 VM：

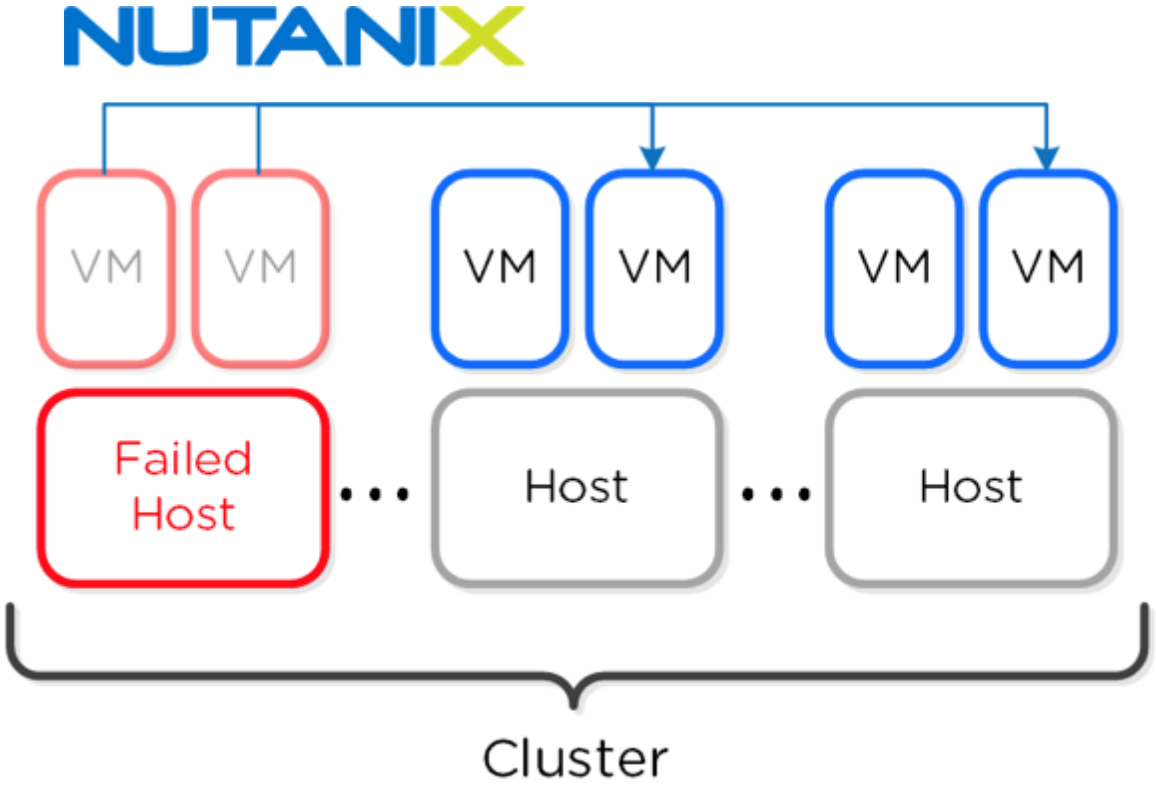


图 13-14. HA - Reserved Segment - Fail Over

Reserve Segments 资源计算：

系统将自动计算总的资源预留，及每个主机的资源预留量。要想更加深入的了解其计算方式，请参见以下文本。

Acropolis HA 使用固定大小的 segment，预留足够的资源，用于保证主机故障时，VM 能够重新启动。segment 的大小对应于群集中最大的 VM。Acropolis HA 可以把多个小的 VM 放置到一个固定的 segment 中。群集中包含大小不一的 VM，单个 segment 可以容纳多个 VM，以此减少因 segment 固定带来的资源浪费。

如何最为有效的安置 VM（最小数量的 segment 预留）就跟 Bin-Packing（一个计算机领域内总所周知的问题）问题一样。最理想的解决方法是 NP-hard，但 heuristic solutions 可以最为接近常见的情况。Nutanix 将继续改进此安置算法。未来版本中，相对于普通的情况，我们预计仅有 0.25 的额外开支。当前，资源碎片的比例在 0.5～1 之间，相对于每个主机故障需要预留的资源在 1.5～2 之间。

当使用 Segment 预留资源时，请留意以下主要构成：

- **Segment size** ＝ 运行中，最大的 **VM** 内存（**GB**）

- **Most loaded host** = 大部分的，运行 **VM** 的主机内存（**GB**）
- **Fragmentation overhead** = 0.5 -1

基于以上的输入，你可以计算预期的 segment 预留数量：

- **Reserve Segments** =（**Most loaded host** / **Segment size**） **x**（**1** + **Fragmentation overhead**）

3.4.6 管理

即将推出!

3.4.7 重要页

即将推出！

3.4.8 命令引用

在 OVS 上启用万兆 uplinks

说明：在本地主机的 Bound0 上启用万兆 uplinks

```
manage_ovs --interfaces 10g update_uplinks
```

说明：启用群集 OVS 的万兆 uplinks

allssh "manage_ovs --interfaces 10g update_uplinks"

显示 ovs 的 uplinks

说明：显示本地主机 OVS 的 uplinks

```
manage_ovs show_uplinks
```

说明：显示群集 OVS 的 uplinks

```
allssh "manage_ovs show_uplinks"
```

显示 OVS 的 interfaces

说明：显示本地主机 OVS 的 interfaces

```
manage_ovs show_interfaces
```

显示群集的 interfaces

```
allssh "manage_ovs show_interfaces"
```

显示 OVS 交换机信息

说明：显示交换机信息

```
ovs-vsctl show
```

列出 OVS 的网桥

说明：列出桥接

```
ovs-vsctl list br
```

显示 OVS 网桥信息

说明：显示 OVS 端口信息

```
ovs-vsctl list port br0
ovs-vsctl list port <bond>
```

显示 OVS 接口信息

说明：显示接口信息

```
ovs-vsctl list interface br0
```

显示网桥的端口/接口

说明：显示网桥端口

```
ovs-vsctl list-ports br0
```

说明：显示网桥接口

```
ovs-vsctl list-ifaces br0
```

创建 OVS 网桥

说明：创建网桥

```
ovs-vsctl add-br <bridge>
```

添加网桥端口

说明：添加网桥端口

```
ovs-vsctl add-port <bridge> <port>
```

Description: Add bond port to bridge 说明：添加网桥的 bond 端口

```
ovs-vsctl add-bond <bridge> <port> <iface>
```

显示 OVS bond 详细信息

说明：显示 Bond 详细信息

```
ovs-appctl bond/show <bond>
```

示例：

```
ovs-appctl bond/show bond0
```

设定 bond 模式并配置 LACP

说明：启用端口上 LACP

```
ovs-vsctl set port <bond> lacp=<active/passive>
```

说明：启用所有主机 bond0 上的 LACP 功能

```
for i in `hostips`;do echo $i; ssh $i source /etc/profile > /dev/null 2>&1; ovs-vsctl set port bond0
lacp=active;done
```

显示 bond 上 LACP 详细信息

说明：显示 LACP 详细信息

```
ovs-appctl lacp/show <bond>
```

设定 bond 模式

说明：设定端口上的 bond 模式

```
ovs-vsctl set port <bond> bond_mode=<active-backup, balance-slb, balance-tcp>
```

显示 OpenFlow 信息

说明：显示 OVS openflow 详细信息

```
ovs-ofctl show br0
```

说明：显示 Openflow 规则

```
ovs-ofctl dump-flows br0
```

获得 QEMU PIDS 和指定进程的运行信息

说明：获得 QEMU PIDs

```
ps aux | grep qemu | awk '{print $2}'
```

说明：获得指定 PID 进程的运行信息

```
top -p <PID>
```

Get active Stargate for QEMU processes 获得 QEMU 进程的活动信息

说明：获得每个 QEMU 进程的 Stargate（数据管理）存储 I/O

```
netstat –np | egrep tcp.*qemu
```

3.4.9 指标和阀值

即将推出！

3.4.10 故障排除和高级管理

检查 iSCSI 重定向日志

说明：检查所有主机的 iSCSI 重定向日志

```
for i in `hostips`; do echo $i; ssh root@$i cat /var/log/iscsi_redirector;done
```

单个主机示例

```
Ssh root@<HOST IP>
Cat /var/log/iscsi_redirector
```

Monitor CPU steal (stolen CPU)

说明：监视 CPU steal

执行 top 查看 Steal Time 百分比

```
Cpu(s):  0.0%us, 0.0%sy,  0.0%ni, 96.4%id,  0.0%wa,  0.0%hi,  0.1%si,  0.0%st
```

监视虚拟机的网络资源状态

说明：监视虚拟机资源状态

Launch virt-top

```
Virt-top
```

去网络页

2 – Networking

3.5 管理

3.5.1 重要页

除了标准管理页面之外还提供了高级页用来监视详细数据和指标，URL 访问方式如下：http://<Nutanix CVM IP/DNS>:<Port/path (下文中提到的)> 示例:

http://MyCVM-A:2009，注：如果你在与 CVM 不同的子网访问页面，需要禁用 CVM 防火墙。

2009 Page

这是 Stargate 页用于监控后端存储，只能由高级用户使用。

2009/latency Page

这是 Stargate 页用于监控后端延迟。

2009/vdisk_stats Page

这是 Stargate 页用于显示各种 vDisk 统计信息，包含 I/O、延迟、写命中(e.g., OpLog, eStore)、读命中(cache, SSD, HDD, etc.)等柱状图。

2009/h/traces Page

这是 Staegate 页用于监控操作的活跃跟踪

2009/h/vars Page

这是 Stargate 页用于监控各种计数器

2010 Page

这是 Curator 页用于监控 Curator 运行

2010/master/control Page

这是 Curator 控制页用于手工启动 Curator 工作

2011 Page

这是 Chronos 页监控通过 Curator 监控工作和任务计划

2020 Page

这是 Cerebro 页用于监控保护域、复制状态和 DR

2020/h/traces Page

This is the Cerebro page used to monitor activity traces for PD operations and replication.

这是 Cerebro 页用于监控 PD 操作和复制的实施追踪

2030 Page

这是 Acropolis 主要页，显示主机环境详细信息、正在运行的任务和网络详细信息。

2030/sched Page

这是一个 Acropolis 页面，通过显示虚拟机资源调度信息帮助决定虚拟机运行位置，这页显示虚拟机运行的主机的可用资源。

2030/tasks Page

这是一个 Acropolis 页面用于显示 Acropolis 任务和状态，你可以点击任务 UUID 来获得任务的详细 JSON 信息。

2030/vms Page

这是一个 Acropolis 页用于显示 Acropolis 虚拟机及虚拟机的详细信息，你可以点击虚拟机名称连接到虚拟机控制台。

3.5.2 群集命令

检查群集状态

说明：通过命令行检查群集状态

```
cluster status
```

检查本地 CVM 服务状态

说明：通过命令行检查本地 CVM 的服务状态

```
genesis status
```

Nutanix 群集升级

说明：通过命令行执行滚动(aka "live")群集升级

上传升级包到一台 CVM 的~/tmp/

解压缩升级包

```
tar xzvf ~/tmp/nutanix*
```

运行升级

```
~/tmp/install/bin/cluster -i ~/tmp/install upgrade
```

检查状态

```
upgrade_status
```

节点升级

说明：运行升级将指定节点升级到当前群集版本

任意 CVM 上运行以下命令

```
cluster -u <NODE_IP(s)> upgrade_node
```

Hypervisor 升级状态

说明：任意 CVM 运行命令检查 Hypervisor 升级状态

```
host_upgrade --status
```

详细日志(on every CVM)

```
~/data/logs/host_upgrade.out
```

命令行重启群及服务

说明：通过命令行重启单一群集服务

停止服务

```
cluster stop <Service Name>
```

启动服务

```
cluster start  #NOTE: This will start all stopped services
```

通过命令行启动群集服务

说明：命令行启动已经停止的群集服务

启动服务

```
cluster start  #NOTE: This will start all stopped services
```

或者
启动单一服务

```
Start single service: cluster start  <Service Name>
```

命令行重启本地服务

说明：命令行重启单一群集服务

停止服务

```
genesis stop <Service Name>
```

启动服务

```
cluster start
```

命令行启动本地服务

说明：命令行启动已经停止的群集服务

```
cluster start #NOTE: This will start all stopped services
```

通过命令行添加群集节点

说明：通过命令行添加群集节点

```
ncli cluster discover-nodes | egrep "Uuid" | awk '{print $4}' | xargs -I UUID ncli cluster add-node node-uuid=UUID
```

显示 vDisks 数量

说明：显示 vDisks 数量

```
vdisk_config_printer | grep vdisk_id | wc -l
```

找到群集 ID

说明：找到当前群集 ID

```
zeus_config_printer | grep cluster_id
```

打开端口

说明：通过 IPtables 启用端口

```
sudo vi /etc/sysconfig/iptables
-A INPUT -m state --state NEW -m tcp -p tcp --dport <PORT> -j ACCEPT
sudo service iptables restart
```

检查 Shadow Clones

说明：通过以下命令格式显示 shadow clones：name#id@svm_id

```
vdisk_config_printer | grep '#'
```

重置延迟页统计

说明：重置延迟页(<CVM IP>:2009/latency)计数器

```
allssh "wget $i:2009/latency/reset"
```

查找 vDisks 数量

说明：查找 Distributed Storage Fabric （DFS）上的 vDisk 数量

```
vdisk_config_printer | grep vdisk_id | wc -l
```

命令行开始 Curator 扫描

Description: Starts a Curator full scan from the CLI 说明：命令行开始 Curator 全扫描

```
allssh "wget -O - "http://$i:2010/master/api/client/StartCuratorTasks?task_type=2";"
```

压缩环

说明：压缩元数据环

```
allssh "nodetool -h localhost compact"
```

查找 NOS 版本

说明：查找 NOS 版本（注：也可使用 NCLI）

```
allssh "cat /etc/nutanix/release_version"
```

查找 CVM 版本

说明：查找 CVM 镜像版本

```
allssh "cat /etc/nutanix/svm-version"
```

手工添加 vDisk 指纹

说明：创造一个特别的 vdisk 指纹（为重复数据删除） 注：重复数据删除，必须在容器上启用

```
vdisk_manipulator –vdisk_id=<vDisk ID> --operation=add_fingerprints
```

Echo Factory_Config.json for all cluster nodes

说明：Echos the factory_config.jscon for all nodes in the cluster

```
allssh "cat /etc/nutanix/factory_config.json"
```

升级一个单一节点的 NOS 版本

说明：升级单一节点 NOS 版本匹配群集

```
~/cluster/bin/cluster -u <NEW_NODE_IP> upgrade_node
```

列出 DSF 文件（vDisk）

说明：为了存储 vDisks 列出 DSF 上的文件和相关信息

```
Nfs_ls
```

获取帮助

```
Nfs_ls --help
```

安装 Nutanix Cluster Check(NCC)

说明：安装 Nutanix Cluster Check（NCC）健康检查脚本来测试潜在的问题和集群健康

从 Nutanix Support Portal (portal.nutanix.com) 下载 NCC

SCP .tar.gz to the /home/nutanix directory

Untar NCC .tar.gz

```
tar xzmf <ncc .tar.gz file name> --recursive-unlink
```

运行安装脚本

```
./ncc/bin/install.sh -f <ncc .tar.gz file name>
```

创建链接

```
source ~/ncc/ncc_completion.bash
echo "source ~/ncc/ncc_completion.bash" >> ~/.bashrc
```

运行 Nutanix Cluster Check (NCC)

说明：运行 Nutanix Cluster Check (NCC) 健康检查脚本来测试潜在的问题和集群健康，但群集故障时这是排查的第一步

确保 NCC 已经安装（以上步骤）

运行 NCC health checks

```
ncc health_checks run_all
```

3.5.3 指标和阈值

以下部分将涉及 Nutanix 后台具体的指标和阈值。更多更新即将推出！

3.5.4 Gflags

即将推出!

3.5.5 故障排除和高级管理

查找 Acropolis 日志

说明：查找 Acropolis 的集群日志

```
allssh "cat ~/data/logs/Acropolis.log"
```

查找群集错误日志

说明：为集群查找错误日志

```
allssh "cat ~/data/logs/<COMPONENT NAME or *>.ERROR"
```

Example for Stargate

```
allssh "cat ~/data/logs/Stargate.ERROR"
```

查找群集致命日志

说明：为群集查找致命日志

```
allssh "cat ~/data/logs/<COMPONENT NAME or *>.FATAL"
```

Example for Stargate

```
allssh "cat ~/data/logs/Stargate.FATAL"
```

3.5.5.1 使用 2009 Page (Stargate)

在大多数情况下，Prism 该能够给你所有你所需要的信息和数据。然而，在某些情况下，如果你想要一些更详细的数据，你可以利用 Stargate 又名 2009 页。2009 页可以通过导航到<CVM IP>：2009 查看。

访问后端页

如果你在不同的网段（L2 子网），你需要添加一个访问规则来访问任何后端页。

在页面的顶部是概述细节显示有关集群的各种详细信息：

Start time	20150618-11:12:52-GMT-0700
Build version	el6-release-master-038d4c7d75cbc6ed21e64d357c94303350159807
Build last commit date	2015-05-30 14:14:03 -0700
Stargate handle	10.3.140.151:2009
iSCSI handle	10.3.140.151:3261
SVM id	7
Incarnation id	30986558
Highest allocated opid	38138394
Highest contiguous completed opid	36132415
Content cache total hits(NonDedup)	86.17%
Content cache flash pagin pct(NonDedup)	0
Content cache total hits(Dedup)	0%
Content cache flash pagin pct(Dedup)	0%
Content cache memory usage	3220 MB
Content cache physical memory usage	3224 MB
Content cache flash usage	0 MB
QoS Queue (size/admitted)	0/72
Oplog QoS queue (size/admitted)	0/0
NFS Flush Queue (size/admitted)	0/0
NFS cache usage	0 MB

图 14-1.2009 页 - Stargate 概述

在本段有两个区域要留意，第一个是 I/O 队列，显示 admitted / outstanding 操作的数量。

该图显示了概述的队列部分：

QoS Queue (size/admitted)	0/26
Oplog QoS queue (size/admitted)	0/0

图 14-2.2009 页 - Stargate 概述 – 队列

第二部分是内容缓存的细节,显示了高速缓存大小的信息和命中率

该图显示了概述的内容缓存部分

Content cache total hits(NonDedup)	86.17%
Content cache flash pagin pct(NonDedup)	0
Content cache total hits(Dedup)	0%
Content cache flash pagin pct(Dedup)	0%
Content cache memory usage	3220 MB
Content cache physical memory usage	3224 MB
Content cache flash usage	0 MB

图 14-3. 2009 页 - Stargate 概览 – 内容缓存

> 提示
>
> 在理想的情况下，如果工作负荷使用了最好的读取性能命中率应高于 80%-90%+

注意：这些值是每个 Stargate / CVM

接下来的部分是“群集状态”，显示群集中不同的 Stargate 详细信息和它们的磁盘使用率。

下图显示了 Stargate 和磁盘利用率（可用/全部）：

SVM Id	IP:port	Incarnation	SSD-PCIe	SSD-SATA		DAS-SATA			
7	10.3.140.151:2009	30986558		154 (188/209)	153 (188/209)	152 (477/862)	151 (477/862)	150 (477/862)	149 (438/782)
8	10.3.140.152:2009	30174288		146 (190/209)	145 (190/209)	144 (474/862)	143 (476/862)	142 (474/862)	141 (434/782)
9	10.3.140.153:2009	30972235		50 (188/209)	49 (188/208)	48 (487/862)	47 (488/862)	44 (486/862)	43 (440/782)
10	10.3.140.154:2009	30989925		139 (189/209)	138 (189/209)	137 (483/862)	136 (482/862)	135 (484/862)	134 (432/782)
11	10.3.140.155:2009	30332545		90 (186/209)	89 (190/209)	88 (594/862)	87 (591/862)	86 (591/862)	85 (533/782)
13	10.3.140.157:2009	30813522		123 (165/209)	122 (165/209)	121 (481/862)	120 (480/862)	119 (481/862)	117 (429/782)
14	10.3.140.158:2009	30460780		78 (189/209)	77 (189/208)	76 (482/862)	75 (477/862)	74 (477/862)	73 (436/782)

图 14-4.2009 页-群集状态-磁盘使用情况

接下来的部分是“NFS Slave”部分，显示每个虚拟磁盘的各种详细信息和统计信息。

下图显示了虚拟磁盘和各种 I/O 的详细信息：

VDisk Name	Unstable data			Outstanding ops		Ops/s			KB/s		Avg latency (usec)		Avg op size	Avg outstanding	% busy
	KB	Ops/s	KB/s	Read	Write	Read	Write	Error	Read	Write	Read	Write			
NFS:31181822 (55b04a56-8e98-4f04-8c0f-617a57cb0450)	0	0	0	0	6	2248	907	0	8992	3628	178	2740	4096	5	85
NFS:31181826 (ede19589-df09-40a8-9640-6cad4e2d48bd)	0	0	0	1	5	2228	922	0	8912	3688	172	2756	4096	6	85
NFS:31182359 (0f1ae5e0-be91-40b8-9acf-5a3227f5c480)	0	0	0	0	0	0	0	0	0	0	0	0		0	0
NFS:31181823 (1a8ef41c-ae3f-4293-a46e-49d7e6c1c907)	0	0	0	0	6	2192	986	0	8768	3944	104	2198	4096	1	82
NFS:31181821 (813b97ae-99c1-4198-a3aa-791bd915b959)	0	0	0	0	5	2254	936	0	9016	3744	173	2761	4096	3	86
NFS:15678286 (bdc1e686-fb82-4157-9657-b8b038ab3e00)	0	0	0	0	0	0	0	0	0	0	0	0		0	0
NFS:31181824 (3d9bcd64-93de-4891-9351-500e589db2db)	0	0	0	0	3	2258	921	0	9032	3684	185	3243	4096	3	86
NFS:31181825 (4a3fc350-73a3-4ead-9104-4d5823143f48)	0	0	0	1	1	2289	956	0	9156	3824	133	2575	4096	3	84

图 14-5.2009 页- NFS Slave-虚拟磁盘统计

提示

When looking at any potential performance issues I always look at the following: 当看到任何潜在的性能问题请关注下几点：

Avg. latency

Avg. op size

Avg. outstanding

欲了解更多具体细节请浏览 vdisk_stats 页面。

3.5.5.2 使用 2009 / vdisk_stats 页

2009 vdisk_stats 页提供了每个虚拟磁盘进一步的数据点。此页面包括细节和随机条带图，延迟条带图，I/O 大小和工作组的细节。

可以通过在左栏点击“vDisk Id”导航到 vdisk_stats 页面。

下图显示了部分和超链接的虚拟磁盘 ID：

Hosted VDisks

The dedup usage is only periodically computed by the curator and may be stale.
Oplog flush statistics.
Stats for all VDisks.

VDisk Id	VDisk Name	Usage (GB)	Dedup (GB)	Oplog				Outstanding ops			Ops/s				KB/s		Avg latency (usec)	Avg op size	Avg qlen	% busy
				KB	Fragments	Ops/s	KB/s	Read	Write	Estore	Read	Write	Error	Random	Read	Write				
15432793	NFS:20581239 (a849adb6-3809-423e-a89c-dd2f02d665d6)	0	0	0	0	0	0	0, 0KB	0, 0KB	0	0	0	0		0	0	0		0	0
31181823	NFS:31181823 (1a8ef41c-ae3f-4293-a46e-49d7e6c1c907)	2	0	198372	49593	990	3960	0, 0KB	0, 0KB	0	2192	1	0	2193	8768	320	46	4243	0	10
31181821	NFS:31181821 (813b97ae-99c1-4198-a3aa-791bd915b959)	2	0	196920	49230	939	3756	0, 0KB	0, 0KB	0	2253	0	0	2253	9012	0	38	4096	0	11

图 14-6.2009 页-托管虚拟磁盘

这将给你详细的虚拟磁盘统计。注：这些值是实时的，并且可以通过刷新页面更新。

第一个关键区域是“Ops and Randomness”部分，这将显示的 I/O 模式是否在本质上随机或顺序崩溃。

该图显示了“行动和随机性”部分：

VDisk 31181841 Ops and Randomness

	Read	Write	Total
IOPS (kIO/s)	2	1	3
IO Rate (MB/s)	9	4	13
Random %	100	100	
Sequential %	0	0	

图 14-7.2009 页-虚拟磁盘统计-行动和随机性

下一个区域显示前端读写的条带图和 I/O 延迟

该图显示了“前端读取延迟”条带图：

VDisk 31181841 Frontend Read Latency

Latency Range	Average Bucket Latency	Number of Read IOs	Percent of Read IOs	Bar Graph
0 <= x < 1 ms	286	6211	92%	
1 ms <= x < 2 ms	1362	371	6%	
2 ms <= x < 5 ms	2855	135	2%	
5 ms <= x < 10 ms	5433	6	0%	
10 ms <= x < 20 ms				
20 ms <= x < 50 ms				
50 ms <= x < 100 ms				
100 ms <= x < inf				
Total	401	6723	100%	

图 14-8.2009 页-虚拟磁盘统计-前端读取延迟

该图显示了“前端写入延迟”条带图：

VDisk 31181841 Frontend Write Latency

Latency Range	Average Bucket Latency	Number of Write IOs	Percent of Write IOs	Bar Graph
0 <= x < 1 ms				
1 ms <= x < 2 ms	1724	167	6%	
2 ms <= x < 5 ms	3422	2098	72%	
5 ms <= x < 10 ms	6369	562	19%	
10 ms <= x < 20 ms	12297	88	3%	
20 ms <= x < 50 ms	23386	6	0%	
50 ms <= x < 100 ms				
100 ms <= x < inf				
Total	4200	2921	100%	

图 14-9.2009 页-虚拟磁盘统计-前端写延迟

接下来的关键区域是 I/O 大小分布，显示读取的条带图和写入 I/O 的大小。

该图显示了“Read Size Distribution”条带图：

VDisk 31181841 Read Size Distribution

Latency Range	Average Bucket Size	Number of Read IOs	Percent of Read IOs	Bar Graph
0 <= x < 4 kB				
4 kB <= x < 8 kB	4096	6723	100%	
8 kB <= x < 16 kB				
16 kB <= x < 32 kB				
32 kB <= x < 64 kB				
64 kB <= x < 512 kB				
512 kB <= x < 1 MB				
1 MB <= x < inf				
Total	4096	6723	100%	

图 14-10.2009 页-虚拟磁盘统计-读 I/O 大小

下图显示了“Write Size Distribution”条带图：

VDisk 31181841 Write Size Distribution

Latency Range	Average Bucket Size	Number of Write IOs	Percent of Write IOs	Bar Graph
0 <= x < 4 kB				
4 kB <= x < 8 kB	4096	2921	100%	
8 kB <= x < 16 kB				
16 kB <= x < 32 kB				
32 kB <= x < 64 kB				
64 kB <= x < 512 kB				
512 kB <= x < 1 MB				
1 MB <= x < inf				
Total	4096	2921	100%	

图 14-11. 2009 页-虚拟磁盘统计-写 I/O 大小

接下来的重点区域是“Working Set Size”部分，提供有关 Working Set Size 的最后 2 分钟和 1 小时的洞察力。细分为读写 I/O。

下图显示了“Working Set Sizes”表：

VDisk 31181841 Working Set Sizes

	2 min	1 hr
Read WSS (MB)	2030	0
Write WSS (MB)	2030	0
Union WSS (MB)	2030	0

图 14-12.2009 页-虚拟磁盘统计-Working Se

“Read Source”提供正在从服务于该层或位置的读取 I/O 信息。

下图显示了“Read Source”的详细信息：

NUTANIX

VDisk 31181841 Read Source

Source	Data (MB/s)	Percent
Oplog		
Extent cache		
Cache DRAM	7	75%
Cache SSD	2	25%
Estore SSD		
Estore HDD		
Block Store		
Zeroes		
Total	9	100%

图 14-13.2009 页-虚拟磁盘统计-Read Source

提示

如果您看到高的读取延迟，看一下虚拟磁盘的读源，并看一下 I/O 的服务源。在大多数情况下，高延迟可以通过读取来自 HDD（Estore HDD）引起的。

“Write Destination”部分将显示其中新写入的 I/O。

该图显示了“Write Destination”表：

VDisk 31181841 Write Destination

Destination	Data (MB/s)	Percent
Oplog	4	100%
Estore		
Block Store		
Total	4	100%

图 14-14.2009 页-虚拟磁盘统计-Write Destination

提示

随机或更小的 I/O（<64K）将被写入到 OPLOG。较大的或顺序的 I/O 将绕过 OPLOG 和直接写到盘区存储（ESTORE）。

另一个有趣的数据是什么数据通过 ILM 被从 HDD 硬盘迁移到 SSD 固态硬盘。在'Extent Group Up-Migration “表显示，在过去 300、3600 和 86400 秒已经迁移的数据。

下图显示了“Extent Group Up-Migration ”表：

NUTANIX

VDisk 31181841 Extent Group Up-Migration

Last 300 seconds	0
Last 3600 seconds	18
Last 86400 seconds	18

图 14-15. 2009 页-虚拟磁盘统计-Extent Group Up-Migration

3.5.5.3 使用 2010 页（Curator）

2010 页是用于监控 Curator MapReduce 框架的详细页面。该页面提供了作业、扫描和相关任务的详细信息。

您可以通过导航到 http://<CVM IP>:2010。注意：如果你不是 Curator Master 请点击“Curator Master:”后的 IP。

页面的顶部将显示 Curator Master 细节包括正常运行时间、版本等等。

下一节是“Curator Nodes”表，它显示有关集群中的节点、角色和健康状况的各种细节。这些节点将被用于分布式进程和选举任务。

该图显示了“Curator Nodes”表：

Curator Nodes

Sl. No.	IP:port	Type	Node	Incarnation Id	MapReduce Version	Build Version	Curator Disks	Health Status
1	10.3.140.151:2010	Master	357	30058470	42000160	159807	1	Healthy
2	10.3.140.152:2010	Slave	319	30174391	42000160	159807	1	Healthy
3	10.3.140.153:2010	Slave	360	30972348	42000160	159807	1	Healthy
4	10.3.140.154:2010	Slave	356	30661501	42000160	159807	1	Healthy
5	10.3.140.155:2010	Slave	318	30332648	42000160	159807	1	Healthy
6	10.3.140.157:2010	Slave	321	30813635	42000160	159807	1	Healthy
7	10.3.140.158:2010	Slave	322	30491731	42000160	159807	1	Healthy

图 14-16 2010 页-Curator Nodes

接下来的部分是“Curator jobs”表，该表显示已完成或当前正在运行的作业。

有作业两种主要类型，其中包括部分扫描是每隔 60 分钟，全面扫描是每 6 小时。注意：基于利用和其他活动的时间是可变的。

这些扫描然将定期调度运行也可以由某些群集事件触发。

这里有一些原因的工作执行：

- 定期（正常状态下）
- 磁盘/节点/模块失效
- ILM 失衡
- 磁盘/层失衡

下图显示“Curator Jobs”表:

Curator Jobs

Job id	Execution id	Job name	Status	Reasons	Background tasks			Start time	End time	Total time (secs)	Scan timeout (secs)
					Submitted	Canceled	Generated				
1	21063	Partial Scan	Succeeded	Periodic	0	0	0	Jul 13 13:59:14	Jul 13 14:05:12	358	43200
1	21061	Partial Scan	Succeeded	Periodic	0	0	0	Jul 13 12:53:07	Jul 13 12:59:13	366	43200
1	21059	Partial Scan	Succeeded	Periodic	0	0	0	Jul 13 11:46:33	Jul 13 11:53:06	393	43200
0	21054	Full Scan	Succeeded	Periodic	34	0	34	Jul 13 10:29:30	Jul 13 10:46:32	1022	43200
1	21052	Partial Scan	Succeeded	Periodic	0	0	0	Jul 13 09:55:01	Jul 13 10:01:12	371	43200
1	21050	Partial Scan	Succeeded	Periodic	0	0	0	Jul 13 08:48:44	Jul 13 08:55:00	376	43200
1	21048	Partial Scan	Succeeded	Periodic	0	0	0	Jul 13 07:42:04	Jul 13 07:48:43	399	43200
1	21046	Partial Scan	Succeeded	Periodic	0	0	0	Jul 13 06:36:08	Jul 13 06:42:03	355	43200
1	21044	Partial Scan	Succeeded	Periodic	6	0	6	Jul 13 05:29:30	Jul 13 05:36:07	397	43200
0	21039	Full Scan	Succeeded	Periodic	37	0	37	Jul 13 04:12:12	Jul 13 04:29:29	1037	43200

图 14-17. 2010 页-Curator Jobs

该表显示了一些由每个作业进行的高级别活动

Table 0. Curator 扫描任务

活动	全扫描	局部扫描
ILM	X	X
磁盘平衡	X	X
压缩	X	X
重复数据删除	X	
擦除编码	X	
垃圾清理	X	

点击“Execution id”将带您到作业详细信息页面，该页面显示各种统计工作，以及所产生的任务。

在页面的顶部表将显示在工作中的各种信息的类型、原因、任务和持续时间。

接下来的部分是“后台任务统计”表，该表上的任务类型显示各种信息，产生量和优先级。

下图中显示作业详细信息表：

Job id	0
Execution id	21054
Scan Type	Full
Reasons	Periodic
Bg tasks generated	34
Bg tasks submitted	34
Bg tasks cancelled	0
Start time	Jul 13 10:29:30
End time	Jul 13 10:46:32
Total time (secs)	1022
Scan timeout (secs)	43200

图 14-18.2010 页 -Curator Job – 详细信息

该图显示了“后台任务统计”表：

Background Task Stats

Job Name	Generated				Submitted				Cancelled			
	High	Medium	Low	Total	High	Medium	Low	Total	High	Medium	Low	Total
MigrateExtents	0	0	1	1	0	0	1	1	0	0	0	0
UpdateRefcounts	0	33	0	33	0	33	0	33	0	0	0	0
Totals	0	33	1	34	0	33	1	34	0	0	0	0

图 14-19. 2010 页- Curator Job – 任务

接下来的部分是“MapReduce 任务”表，该表显示了每个 Curator job 开始实际的 MapReduce 工作。局部扫描将有一个单一的 MapReduce 工作，全面扫描，将有四个 MapReduce 工作。

下图显示“MapReduce Jobs”表

MapReduce Jobs

Job id	Job name	Status	Map tasks done	Reduce tasks done	Fg tasks	Bg tasks	Errors	Start time	End time	Total time (secs)
21064	PartialScan MapReduce	Succeeded	35/35	35/35	2493	0	0	Jul 13 14:00:14	Jul 13 14:05:12	298
21062	PartialScan MapReduce	Succeeded	35/35	35/35	2492	0	0	Jul 13 12:54:07	Jul 13 12:59:12	305
21060	PartialScan MapReduce	Succeeded	35/35	35/35	2493	0	0	Jul 13 11:47:34	Jul 13 11:53:05	331
21058	FullScan MapReduce #4	Succeeded	14/14	28/28	2621	34	0	Jul 13 10:41:13	Jul 13 10:46:30	317
21057	FullScan MapReduce #3	Succeeded	7/7	7/7	0	0	0	Jul 13 10:38:26	Jul 13 10:41:13	167
21056	FullScan MapReduce #2	Succeeded	7/7	7/7	0	0	0	Jul 13 10:33:47	Jul 13 10:38:26	279
21055	FullScan MapReduce #1	Succeeded	15/15	14/14	33	0	0	Jul 13 10:33:30	Jul 13 10:33:47	17
21053	PartialScan MapReduce	Succeeded	35/35	35/35	2493	0	0	Jul 13 09:56:01	Jul 13 10:01:11	310
21051	PartialScan MapReduce	Succeeded	35/35	35/35	2492	0	0	Jul 13 08:49:45	Jul 13 08:54:59	314
21049	PartialScan MapReduce	Succeeded	35/35	35/35	2492	0	0	Jul 13 07:43:04	Jul 13 07:48:42	338

图 14-20.2010 页- MapReduce 作业

点击“Job ID”将带您到 MapReduce 工作的详细信息页面，该页面会显示任务的状态，各种计数器以及关于 MapReduce 工作细节。

图中显示了一些工作的计数器的示例：

NUTANIX

Job Counters

Name	Value
MapExtentGroupIdMap	2155990
MapExtentGroupAccessDataMap	2155985
NumExtentGroupsToMigrateForILM	0
NumExtentGroupsToMigrateForDiskBalancing	0
NumTasksToMigrateErasureCodedExtents	0
ReduceNonDedupExtentIdTrueRefCount	13023596
ReduceDedupExtentIdTrueRefCount	3295838
ReduceNonDedupExtentIdRefCount	13037299
ReduceDedupExtentIdRefCount	3295838
ReduceDiskIdExtentGroupId	3753217

图 14-21. 2010 页- MapReduce 作业-计数器

主页上的下一个部分是“Queued Curator Jobs”和“Last Successful Curator Scans”部分。这些表显示当定期扫描正常运行时，最后一次成功扫描的详细信息。

该图显示了“Curator 作业队列”和“Curator 最后一次成功扫描”部分：

Queued Curator Jobs

Ring change in progress? No

Job id	Job name	Eligible time (secs)
1	Partial Scan	2516
0	Full Scan	8596

Last Successful Curator Scans

Job name	Start time	End time	Total time (secs)	Master handle	Incarnation id	Job id	Execution id	Status	Reasons	Num MR jobs
Partial Scan	Jul 13 13:59:14	Jul 13 14:05:12	358	10.3.140.151:2010	30058470	1	21063	Succeeded	Periodic	1
Full Scan	Jul 13 10:29:30	Jul 13 10:46:31	1021	10.3.140.151:2010	30058470	0	21054	Succeeded	Periodic	4

图 14-22.2010 页-队列和成功扫描

4 第四部分：vSphere

4.1 架构

4.1.1 节点架构

在 ESXi 的部署中，控制器虚拟机（CVM）硬盘使用的 VMDirectPath I/O 方式。这使得完整的 PCI 控制器（和附加设备）通过直通方式连接 CVM 并绕过虚拟化层。

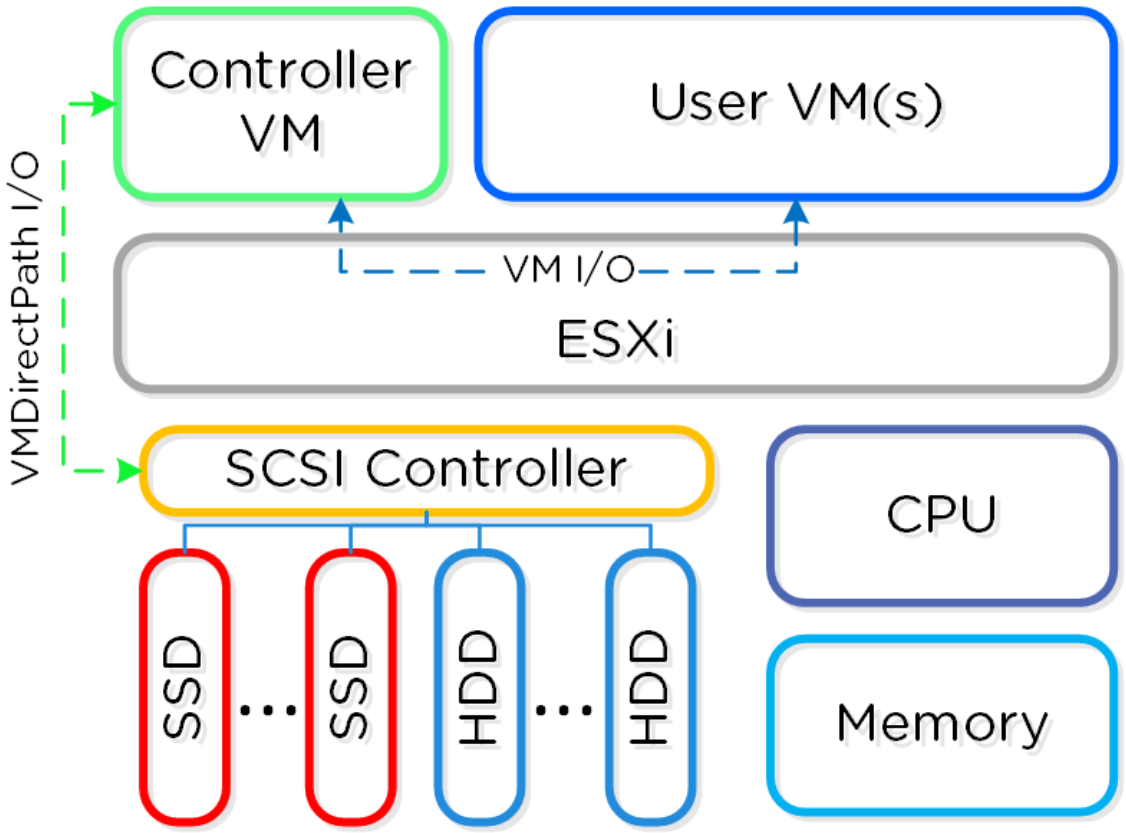


图 15-1.ESXi 的节点架构

4.1.2 最高配置和可扩展性

下面的最高配置和可扩展性的限制适用：

- 最大群集大小：**64**
- 每个虚拟机的最大虚拟 **CPU**：**128**
- 每个虚拟机最大内存：**4TB**

- 每台主机最大的虚拟机：**1024**
- 每个集群最大的虚拟机：**8000（如果启用 HA 2048 每个数据存储）**

注：适用于 vSphere 6.0

4.1.3 网络

每个 ESXi 主机有一个本地的 vSwitch 用来做 Nutanix CVM 和主机之间的内部通信。对于 VM 和外部的通信则利用标准的 vSwitch（默认）和 dvSwitch 来实现。

本地 vSwitch (vSwitchNutanix)用来做 Nutanix CVM 和 ESXi 主机的本地通信。主机上有 vmkernel 端口与 vSwitch (vmk1 - 192.168.5.1)相连， CVM 有一个接口绑定在内部 switch 的端口组上 (svm-iscsi-pg - 192.168.5.2)。这是一个私有的存储通信路径。

外部 vSwitch 可以是标准的 vSwitch 或者 dvSwitch，主要负责 ESXi 主机和 CVM 的外部接口以及 VM 在主机上所需要的端口组。外部 vmkernel 接口用来做主机管理、vMotion 等。 外部 CVM 接口用来做和其他 Nutanix CVM 的通信。如果 Trunk 上开启 VLAN，那么端口组就可以根据需要创建。

下图显示了 vSwitch 的结构:

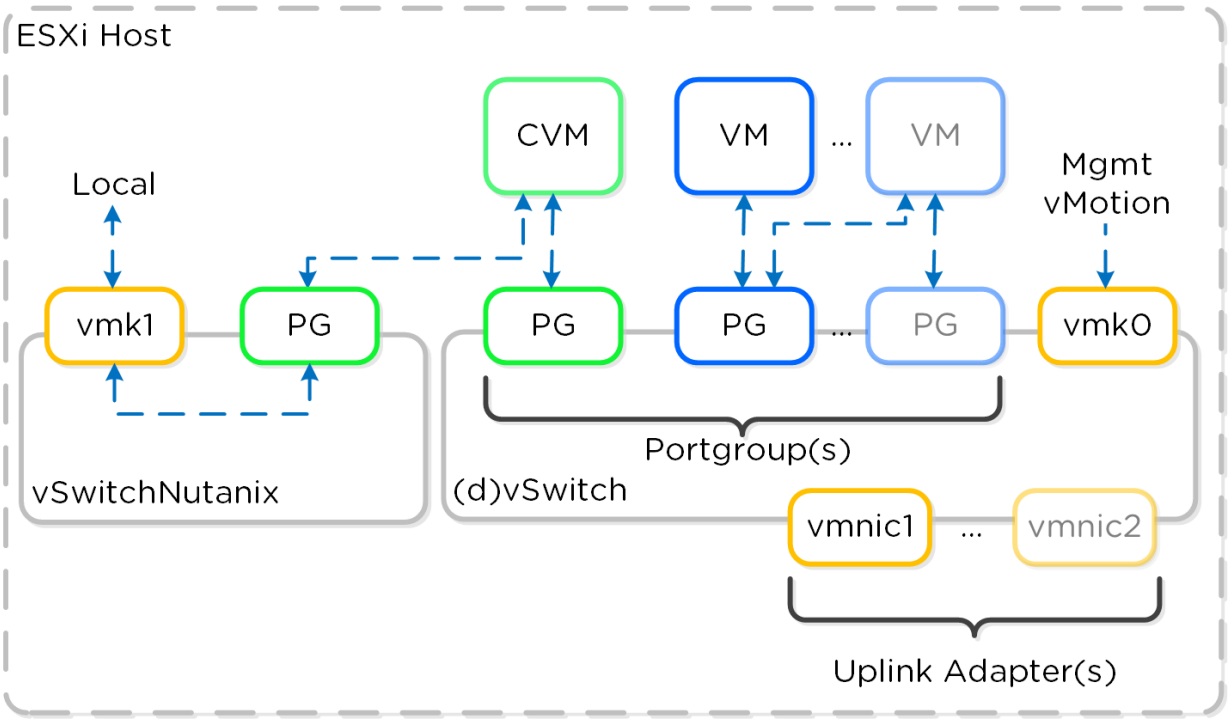


图. ESXi vSwitch Network Overview

上行和链路绑定策略

推荐采用两个 ToR 交换机和上行链路 来保证交换机的 HA。系统默认的上行接口是 active/passive 模式。具有 active/active 上行接口(e.g. vPC, MLAG, etc.)的上行交换机可以用作其他的网络流量。

4.2 如何工作

4.2.1 磁盘阵列减轻负载—VAAI

Nutanix 平台支持 VMware API 进行阵列集成（VAAI），它允许 hypervisor 卸载某些任务负载到阵列。这将使 Hypervisor 效率更高并且不需要“中间人”。Nutanix 目前支持 NAS 的 VAAI 原语，包括“全文件克隆”，“快速文件克隆”和“预留空间”原语。这里有一个很好的文章，解释的各种原语：http://cormachogan.com/2012/11/08/vaai-comparison-block-versus-nas/。

- 克隆的 VM 快照 - > VAAI 将不能使用
- 克隆虚拟机没有快照且处于关机状态 - > VAAI 将被使用

- 克隆虚拟机到不同的数据存储/容器 - > VAAI 将不能使用
- 克隆开机状态的虚拟机 - > VAAI 将不能使用

这些方案适用于 VMware View：

- View 完整克隆（模板与快照） - > VAAI 将不能使用
- View 完整克隆（模板 W/O 快照） - > VAAI 将被使用
- View 链接克隆（VCAI） - > VAAI 将被使用

您可以验证 VAAI 的操作，通过使用“NFS 适配器”活动痕迹页面。

4.2.2 CVM Autopathing aka Ha.py

在本节中，我将介绍如何 CVM 的失败'的处理方式（我将讨论我们如何处理未来的更新组件故障）。一个 CVM“失败”可能包括用户将 CVM 关机， CVM 升级，或任何可能会搞垮 CVM 的事件。DSF 有一个称为 autopathing 特征，其中当一个本地 CVM 变得不可用时，I/O 的透明传递给其他 CVMS 集群中的处理。Hypervisor 和 CVM 通信使用一个专用的 vSwitch 的私网地址 192.168.5.0（详见上图）。这意味着，对于所有存储的 I/O，这些都发生在上 CVM（192.168.5.2）的内部 IP 地址。在 CVM 的外部 IP 地址用于远程复制和 CVM 通信。

下图显示了这样的一个例子：

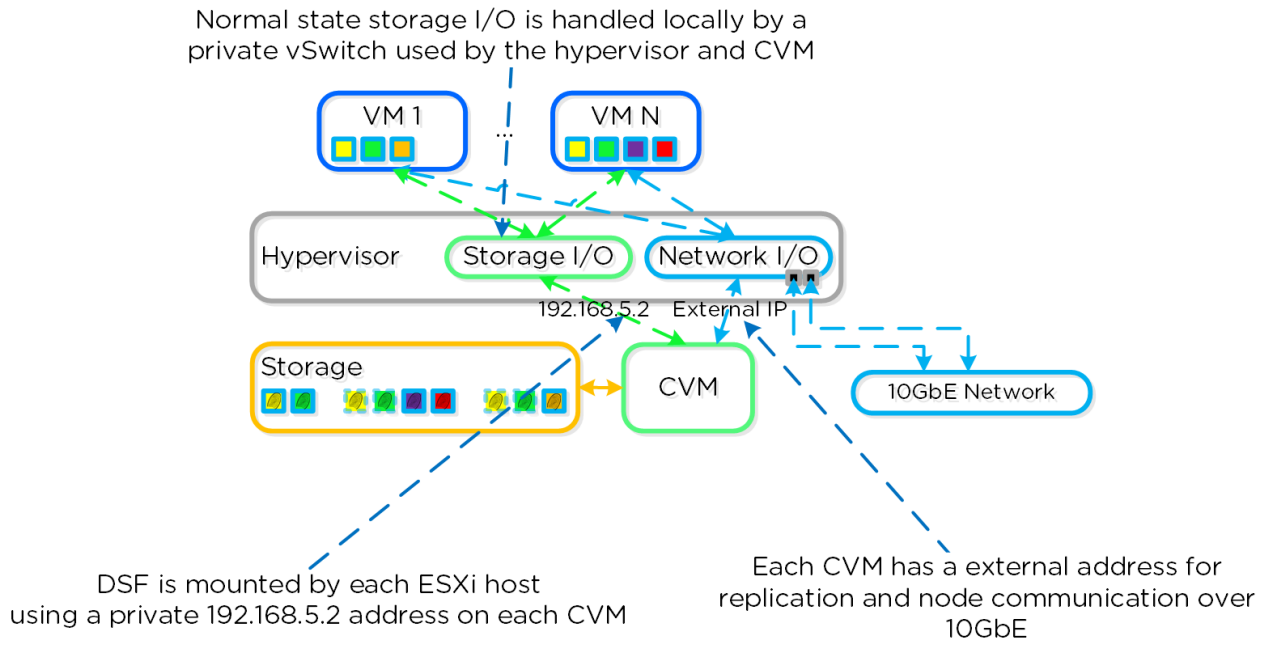


图 16-1 ESXi 主机网络

在一个本地的 CVM 故障的情况下，先前由本地 CVM 托管在本地 192.168.5.2 地址是不可用的。DFS 将自动检测到该故障并通过万兆网络重定向这些 I/O 到群集中另一个 CVM 上。Hypervisor 和主机上运行的虚拟机将以透明的方式重新路由。这意味着，即使一个 CVM 关机，虚拟机仍然会继续能够执行 I/O。一旦本地 CVM 的还原并可用，流量随后将无缝地转交给本地 CVM 继续服务。

下图显示了如何寻找一个失败的 CVM 的图示：

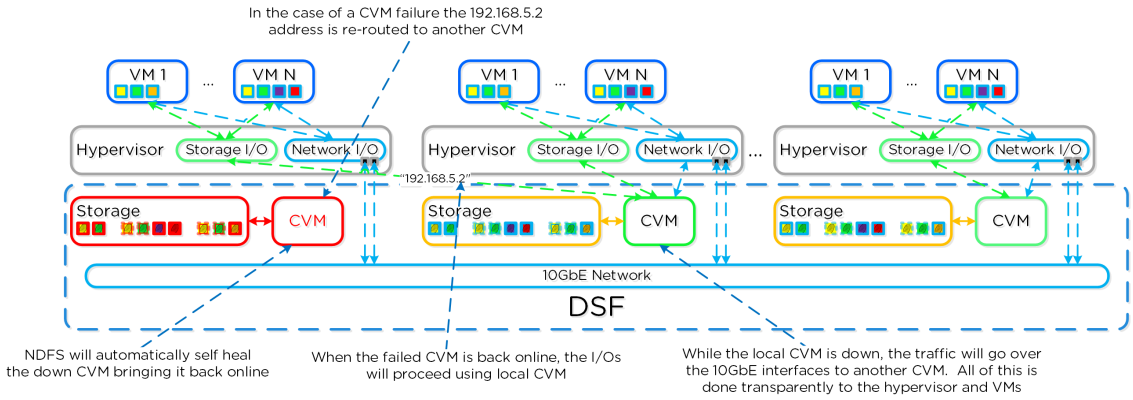


图 16-2. ESXi Host Networking - Local CVM Down

4.3 管理

4.3.1 重要页

即将推出更多内容！

4.3.2 命令参考

ESXi 的集群升级

说明：使用 CLI 执行 ESXi 主机的自动升级

上传离线升级包到 Nutanix NFS 容器

登录到 Nutanix CVM

进行升级

```
for i in `hostips`;do echo $i && ssh root@$i "esxcli software vib install -d /vmfs/volumes/<Datastore Name>/<Offline bundle name>";done
```

示例

```
for i in `hostips`;do echo $i && ssh root@$i "esxcli software vib install -d /vmfs/volumes/NTNX-upgrade/update-from-esxi5.1-5.1_update01.zip";done
```

在自动化主机重启对于 PowerCLI 的：执行 ESXi 主机的滚动重启

重新启动 ESXi 主机服务

说明：以一个增量的方式重新启动每个 ESXi 主机服务

```
for i in `hostips`;do ssh root@$i "services.sh restart";done
```

显示 ESXi 主机的网卡“up”状态

说明：显示 ESXi 主机的网卡这是在“UP”状态

```
for i in `hostips`;do echo $i && ssh root@$i esxcfg-nics -l | grep Up;done
```

显示 ESXi 主机的 10GbE 网卡和状态

说明：显示 ESXi 主机的 10GbE 网卡和状态

```
for i in `hostips`;do echo $i && ssh root@$i esxcfg-nics -l | grep ixgbe;done
```

显示 ESXi 主机的活动适配器

说明：显示 ESXi 主机的工作，待机和未使用的适配器

```
for i in `hostips`;do echo $i &&  ssh root@$i "esxcli network vswitch standard policy failover get --vswitch-name vSwitch0";done
```

显示 ESXi 主机路由表

说明：显示 ESXi 主机的路由表

```
for i in `hostips`;do ssh root@$i 'esxcfg-route -l';done
```

检查 VAAI 在 Datastore 上是否启用

说明：检查 VAAI 在 Datastore 上是否启用/支持

```
vmkfstools -Ph /vmfs/volumes/<Datastore Name>
```

设定 VIB 校验级别为 community supported

说明：设置 VIB 校验程度为 CommunitySupported 允许第三方安装 VIB

```
esxcli software acceptance set --level CommunitySupported
```

安装 VIB

说明：不检查签名安装 VIB

```
esxcli software vib install --viburl=/<VIB directory>/<VIB name> --no-sig-check
```

或者

```
esxcli software vib install --depoturl=/<VIB directory>/<VIB name> --no-sig-check
```

检查的 ESXi 虚拟盘空间

说明: 检查 ESXi ramdisk 可用空间

```
for i in `hostips`;do echo $i; ssh root@$i 'vdf -h';done
```

清除 pynfs 日志

说明：清除每个 ESXi 主机上的 pynfs 记录

```
for i in `hostips`;do echo $i; ssh root@$i '> /pynfs/pynfs.log';done
```

4.3.3 指标和阈值

即将推出！

4.3.4 故障排除和高级管理

即将推出!

5 第五部分：Hyper-V

5.1 架构

5.1.1 节点架构

在 Hyper-V 部署中，控制器 VM（CVM）运行的虚拟机和磁盘使用磁盘直通方式（passthrough）。

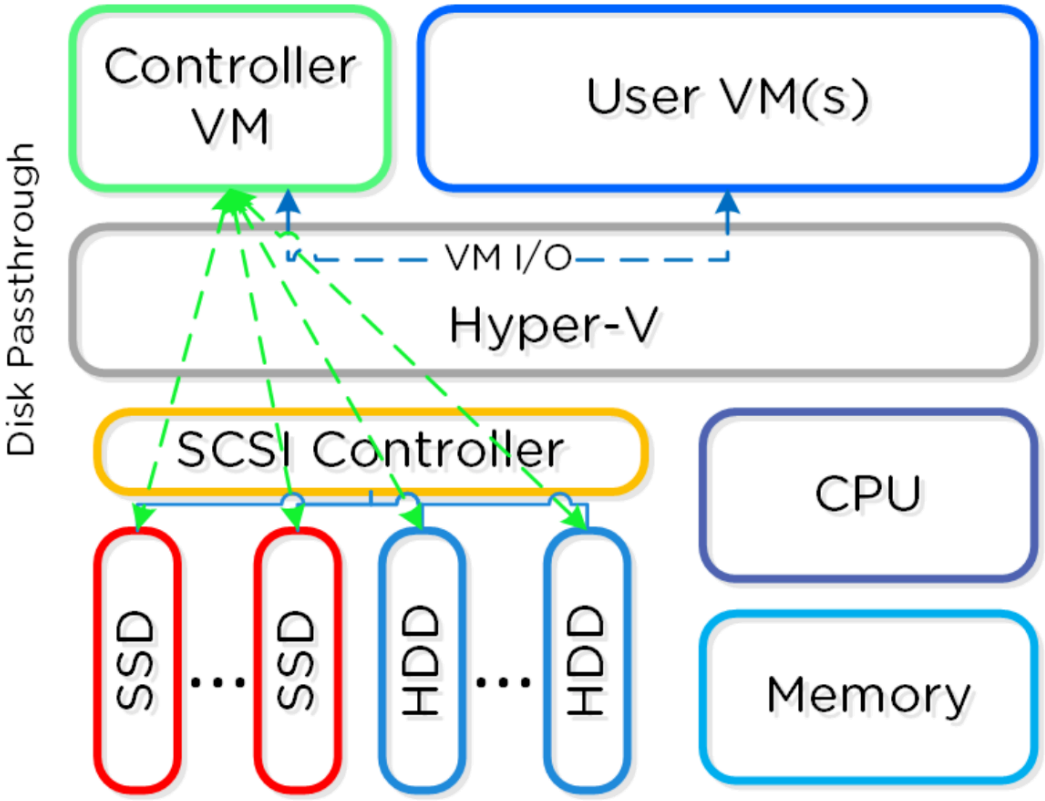


图 18-1.Hyper-V 的节点架构

5.1.2 最高配置和可扩展性

下面是最高配置和可扩展性的限制适用：

- 最大群集大小：**64**
- 每个 VM 最大的 **vCPU**：**64**
- 每个虚拟机最大内存：**1TB**
- 每台主机最大的虚拟机：**1024**
- 每个集群最大的虚拟机：**8000**

注：适用 Hyper-V 2012 R2 版本

5.1.3 网络

每个 Hyper-V 主机有一个内部的虚拟交换机用来做 Nutanix CVM 和主机之间的内部通信。对于 VM 和外部的通信则利用外部的虚拟交换机（默认）或逻辑交换机来实现。

内部交换机 (InternalSwitch)用来做 Nutanix CVM 和 Hyper-V 主机的本地通信。主机上有 vmkernel 端口与 vSwitch (vmk1 - 192.168.5.1)相连， CVM 有一个接口绑定在内部 switch 的端口组上 (svm-iscsi-pg - 192.168.5.2)。 在这个内部交换机上，主机有一个虚拟以太网接口 (vEth) (192.168.5.1)，CVM 在内部交换机上有一个虚拟以太网接口 vEth (192.168.5.2)。这是一个私有的存储通信路径。

外部 vSwitch 可以是标准的 vSwitch 或者逻辑交换机，主要负责 Hyper-V 主机和 CVM 的外部接口以及 VM 在主机上所需要逻辑网络。外部 vEth 接口用来做主机管理、在线迁移等。 外部 CVM 接口用来做和其他 Nutanix CVM 的通信。如果 Trunk 上开启 VLAN，那么端口组就可以根据需要创建。

下图显示了 虚拟交换机 的结构:

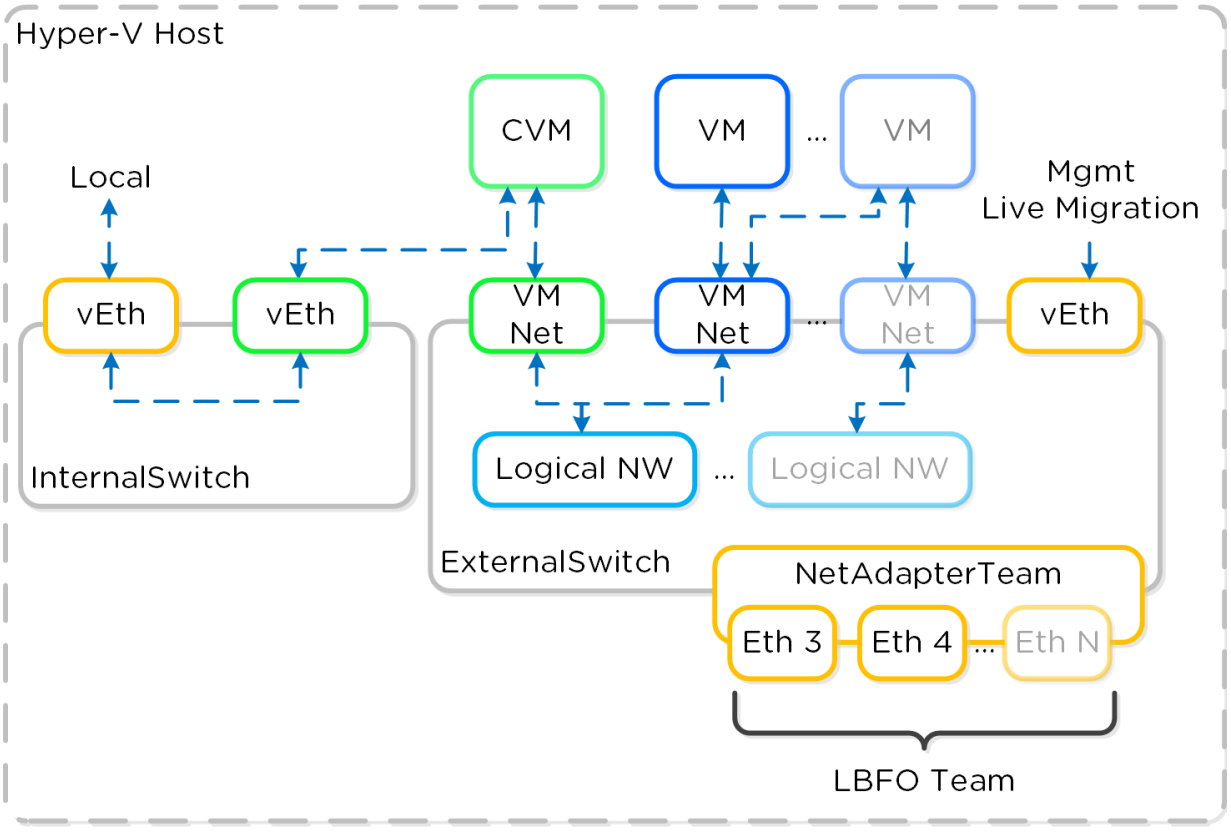


图. Hyper-V Virtual Switch Network Overview

上行和链路绑定策略

推荐采用两个 ToR 交换机和上行链路来保证交换机的 HA。系统在交换机独立模式默认采用 LBFO 绑定，这无需其他特殊的配置。

5.2 如何工作

5.2.1 磁盘阵列减轻负载－ODX

Nutanix 平台支持微软 Offloaded Data Transfer（ODX），允许 Hypervisor 卸载某些任务负载到阵列。这将使 Hypervisor 效率更高并且不需要“中间人”。Nutanix 目前支持 ODX 为中小型企业，其中包括完整的复制和归零操作。然而，与 VAAI 其中有一个“快速的文件”克隆操作（使用可写快照）相反，不具有同等功效并执行完全拷贝。鉴于此，更有效的方法是依靠本地 DSF 克隆，目前可以通过调用通过 nCLI、REST、或 Powershell CMDlets 调用。

目前 ODX 是以下操作调用：

- 在 DSF SMB 共享上的虚拟机或虚拟机 VM 文件副本
- SMB 共享文件副本

从 SCVMM 库（DSF SMB 共享）部署的模板 - 注：共享必须被添加到使用短名称的 SCVMM 群集（例如，不是 FQDN）。一个简单的方法来迫使这是一个条目添加到主机文件的群集（例如：10.10.10.10 nutanix-130）。

ODX 不能通过以下操作来调用：

- 通过 SCVMM 克隆虚拟机
- 从 SCVMM 库（非 DSF SMB 共享）部署模板
- XenDesktop 的克隆部署

您可以验证 ODX 操作正在发生，通过使用“NFS 适配器”的活动痕迹页面（是的 NFS，尽管正在通过 SMB 执行）。当复制 vDISK 时该操作活动显示“NfsSlaveVaaiCopyDataOp”当一块磁盘置 0 时显示'NfsSlaveVaaiWriteZerosOp“时。

5.3 管理

5.3.1 重要页

即将推出更多内容！

5.3.2 命令参考

在多台远程主机上执行命令

说明：在一个或多个远程主机执行一个 PowerShell

```
$targetServers = "Host1","Host2","Etc"
Invoke-Command -ComputerName  $targetServers {
    <COMMAND or SCRIPT BLOCK>
}
```

检查可用 VMQ Offloads

Description: Display the available number of VMQ offloads for a particular host 说明：显示特定主机 VMQ Offloads 的可用数量

```
gwmi –Namespace "root\virtualization\v2" –Class Msvm_VirtualEthernetSwitch | select elementname, MaxVMQOffloads
```

禁用匹配特定的前缀虚拟机的 VMQ

说明：禁用特定虚拟机的 VMQ

```
$vmPrefix = "myVMs"
Get-VM | Where {$_.Name -match $vmPrefix} | Get-VMNetworkAdapter | Set-VMNetworkAdapter -VmqWeight 0
```

启用匹配特定的前缀虚拟机的 VMQ

说明：启用特定虚拟机的 VMQ

```
$vmPrefix = "myVMs"
Get-VM | Where {$_.Name -match $vmPrefix} | Get-VMNetworkAdapter | Set-VMNetworkAdapter -VmqWeight 1
```

开机特定前缀的虚拟机

说明：**将特定前缀的虚拟机开机**

```
$vmPrefix = "myVMs"
Get-VM | Where {$_.Name -match $vmPrefix -and $_.StatusString -eq "Stopped"} | Start-VM
```

关机特定前缀的虚拟机

说明：**将特定前缀的虚拟机关机**

```
$vmPrefix = "myVMs"
Get-VM | Where {$_.Name -match $vmPrefix -and $_.StatusString -eq "Running"}} | Shutdown-VM -
RunAsynchronously
```

停止特定前缀的虚拟机

说明：**将特定前缀的虚拟机停止**

```
$vmPrefix = "myVMs"
Get-VM | Where {$_.Name -match $vmPrefix} | Stop-VM
```

获取 Hyper-V 主机的 RSS 设置

说明：获取 Hyper-V 主机 RSS(recieve side scaling 设置

```
Get-NetAdapterRss
```

检查 Winsh 和 WinRM 连接

说明：通过执行一个简单的查询检查 Winsh 和 WinRM 连接/状态，应返回“the computer system object not an error”

```
allssh "winsh "get-wmiobject win32_computersystem"
```

5.3.3 指标和阈值

即将推出！

5.3.4 故障排除和高级管理

即将推出!

6 后记

感谢您阅读的 Nutanix 圣经！敬请期待更多的即将到来的更新并享受 Nutanix 平台！