OpenStack 架构设计指南

当前最新 (2015-08-09)

摘要

欲充分利用OpenStack的优点，用户需要精心准备规划、设计及架构。认真纳入用户的需求、揣摩透一些用例。

目录

前言

约定

OpenStack文档使用了多种排版约定。

声明

以下形式的标志为注意项

注意

便签或提醒

重要

提醒用户进一步操作之前必须谨慎

警告

关于有丢失数据的风险或安全漏洞的危险信息提示

命令行提示符

$ 提示符	$ 提示符表示任意用户，包括 root用户，都可以运行的命令。
# 提示符	# 提示符表明命令仅为 root可以执行。当然，用户如果可以使用sudo命令的话，也可以运行此提示符后面的命令。

文档变更历史

此版本的向导完全取代旧版本的，且所有旧版本的在此版本中不再生效。

下面表格描述的是近期的改动：

Revision Date	Summary of Changes
October 15, 2014	· 参与编辑请遵照OpenStack风格。
July 21, 2014	· 初始版

第 1 章 介绍

目录

在云计算技术的“淘金热”中，OpenStack无疑是作为领导者出现的，在云自助服务和基础设施即服务(IaaS)的市场，为形形色色的>组织发现更大的灵活性，并加速这个市场。当然，要想知道它的全部好处，云必须要有正确的架构设计。

一个完美的云架构可以提供一个稳定的IT环境，可轻松访问需要的资源、基于使用的付费、按需扩充容量、灾难恢复、安全等。但是一个完美的云架构不会拥有魔法般的自己构建。它需要对多种因素进行精心思考，无论是技术的还是非技术的。

对于OpenStack云部署来说，还没有所谓的那个架构是“正确”的。OpenStack可用于多种目的和场景，而每一种都有着自己独特的需求和架构特点。

本书的初衷是着眼于使用OpenStack云的通用用例(即使不那么通用，至少是作为好的案例)以及解释什么问题是应该去考虑的，以及为什么这么考虑，提供丰富的知识和建议，以帮助一些组织设计和构建一个完美架构的、满足独特需求的OpenStack云。

目标读者

本书是写给OpenStack云的架构师和设计师们。本书不是为那些打算部署OpenStack的人们写的。关于部署和实战的指南请参考 OpenStack 实战指南(http://docs.openstack.org/openstack-ops)。

读者需要有云计算架构和原理的知识背景，企业级系统设计经验，linux和虚拟化经验，以及可理解基本的网络原理和协议。

本书是如何组织的

本书采用多个章节来组织，通过使用案例来帮助用户作出架构选择，以OpenStack云安装来具体体现。每一章关注一个单一的架构以鼓励读者

可随意的阅读，但是每一章包含的一些适用场景的信息也是其他章节需要的。云架构师也许通过阅读所有的案例将此书作为一个全面的参考，但是很可能仅重复阅读其中适合于某一案例的章节。当选择阅读挑选的使用案例时，请记住需要阅读更多的章节，以为云制定完整的设计。本指南中覆盖到的使用案例包括：

- 通用型: 使用通用的组件构建一个云，是多数的使用案例，占80%。
- 计算型: 专注于计算密集性、高负载的云，例如高性能计算(HPC)
- 存储型: 用于存储密集负载的云，例如基于并行文件系统的数据分析。
- 网络型: 依赖于网络的高性能和高可靠的云，例如 内容传送网络 (CDN)。
- 多站点: 构建一个基于地理分布、可靠性、数据位置等原因的应用程序部署且需多个站点可供服务的云。
- 混合云: 一种多个不同的云相杂的架构，目的是提供可用性，失效切换，防止单一云的突然崩溃。
- 大规模扩展: 专注于云计算服务提供者或其他大规模扩展安装的架构。

章节 特殊用例阐释了上述所定义的使用案例中没有覆盖到的架构信息。

本指南的每一章都会分出以下小节：

- 介绍：此架构使用案例的额要介绍
- 用户需求：用户所考虑的内容，那些典型的在组织中起关键作用的。
- 技术考虑：在实际案例中所处理的技术问题。
- 实践考虑：在实际案例中和架构考虑中的那些不断的操作任务。
- 架构：总体架构
- 实例：架构设计被实际部署的一个或几个场景。

书中所用到的术语词汇表

我们为什么及如何写作此书

OpenStack环境从验证测试迁移到实际生产部署的速度决定着和架构设计相关的问题的多寡。目前已经存在的文档中，都是定位在特定的部署、配置以及实际操作的，还没有整体框架式的相关描述。

我们写作此书是希望能够给读者带来抛砖引玉的作用，让读者可以根据自身所在的组织具体需求来设计OpenStack架构。此指南针对通用情况的使用

案例，抽取了我们认为重要的设计考虑，且根据我们的设计提供了具体的例子。此指南的目标不是为安装和配置云提供相关介绍，是专注于那些和用户需求密切相关的设计原理，相比技术和操作上的考虑也少了许多。在OpenStack文档相关的板块中关于安装和配置的相关向导/指南已经有很多资料了，寻找想要的即可。

本书是基于sprint格式书写的，sprint格式是一种针对写书的便利、快速开发的一种方法。想得知更多信息，请访问Book Sprints官方站点(www.booksprints.net)。

本书的写作共花了5天的时间，在2014年的7月份，中间当然历经辛勤的汗水、激情以及合理的饮食。在帕洛阿尔托的VMware公司总部午饭时我们庆祝了本书的完成。整个过程我们在Twitter中使用#OpenStackDesign 直播。Faith Bosworth 和 Adam Hyde帮助我们轻松驾驭Book Sprint。

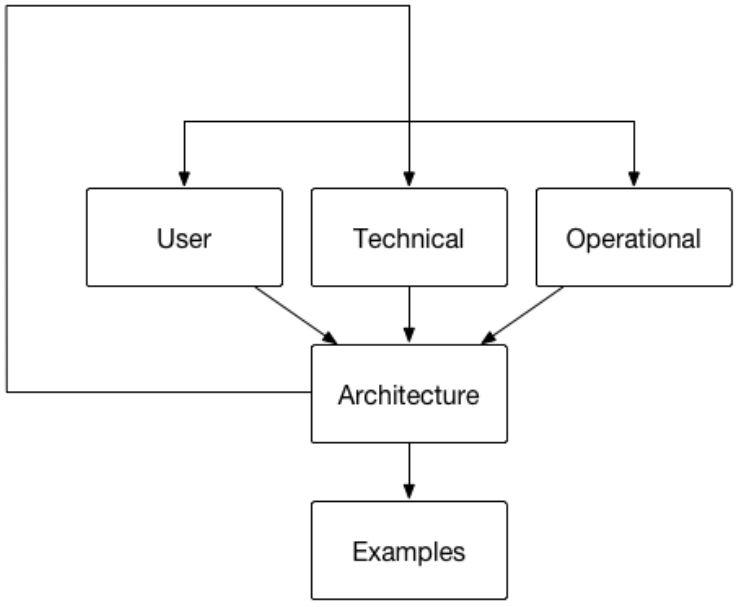
我们非常感谢VMware的盛情款待，以及我们的雇主，Cisco, Cloudscaling,Comcast,EMC,Mirantis,Rackspace,Red Hat, Verizon和VMware,能够让我们花时间做点有意义的事情。尤其感谢Anne Gentle 和 Kenneth Hui，由于二位的领导和组织，才有此书诞生的机会。

作者团队成员有：

- Kenneth Hui (EMC) @hui_kenneth
- Alexandra Settle (Rackspace) @dewsday
- Anthony Veiga (Comcast) @daaelar
- Beth Cohen (Verizon) @bfcohen
- Kevin Jackson (Rackspace) @itarchitectkev
- Maish Saidel-Keesing (Cisco) @maishsk
- Nick Chase (Mirantis) @NickChase
- Scott Lowe (VMware) @scott_lowe
- Sean Collins (Comcast) @sc68cal
- Sean Winn (Cloudscaling) @seanmwinn
- Sebastian Gutierrez (Red Hat) @gutseb
- Stephen Gordon (Red Hat) @xsgordon
- Vinny Valdez (Red Hat) @VinnyValdez

方法论

云的魅力在于它可以干任何事。无论从健壮性还是灵活性，都是这个世界上目前来说最好的。是的，云具有很高的灵活性，且可以做几乎任何事，但是想要充分发挥云的威力，定义好云将如何使用是非常重要的，否则创建和测试的使用案例意义就不复存在。此章讲述的是那些设计最适合、用户最关注的云架构背后的思考过程。



上图展示了在获取用户需求和构建使用案例过程中的抽象流程。一旦用例被定义，它就可以被用于设计云的架构。

用例选择规划看起来是反直觉的，比如，使用Amazon的服务，从注册到使用也就是花5分钟的时间而已。难道Amazon真的不关心用户的计划？错了，Amazon的产品管理部门花了大量的时间去研究吸引他们典型用户的究竟是什么，也在琢磨着交付更好的服务。对于企业来说，计划的流程没有什么不同，只是不会去规划外部的付费用户罢了，举例来说，用户仅仅是内部的应用程序开发者或者是web门户。下面所列的目标在考虑创建用例时用的到。

总体业务目标

- 开发务必清晰定义，符合业务目标和需求
- 增加项目的支持，积极的和业务相关人员、客户以及最终用户保持沟通。

技术

- 协调OpenStack架构中各个项目，努力的通过社区获得更多的支持。
- 为自动化设计好的架构可大大的加速开发和部署。
- 使用恰当的工具可增加开发的效率。
- 创建更好的、更多的测试指标和测试工具，以支持开发的持续集成、测试流程和自动化。

组织

- 选用好多消息工具，良好的管理支持团队。
- 增强文化氛围，诸如对于开源的理解，云架构，敏捷开发，持续开发/测试/集成等，总而言之，涉及到开发的所有环节都需要。

举例来说明上述的一切，想想一个业务目标是将其公司的电子商务站点使用云来构建。此目标意味着应用程序将要支持每秒成千上万的会话，各种负载以及大量复杂和瞬息完变的数据。通过识别诸如每秒并发事务，数据库的大小等关键指标，相信构建出一个基于假设的测试方法是可行的。

基于用户场景开发功能. 基于用户场景开发功能，可轻松开发测试用例，即可在项目的整个过程中有个掂量。假如组织没有准备好一个应用，或者没有准备好一个用于开发用户需求的应用，那么就需要创建需求，以构建合适的测试工具和开发可用的指标。一旦指标确认，即使遇到需求变更，也可轻松面对快速变化，而且无须过度担心提前具体需求。将此类比于使用创新的方法配置系统，不能因为需求的变化而每次都重新设计。

云的局限特性集. 建立需求是发掘用户痛点，而不是重新创建OpenStack已经有的工具集。认为建立OpenStack的只有一种好办法，那只能弄巧成拙。在开发一个平台中通过限制集中带来的慢速是非常重要的，因为它会导致需求转变为处理工具本身，所以请不要重新创建一整套的工具。欲成功的部署云，与技术产品负责人一起建立关键功能显得不可或缺。

为云准备好应用程序

尽管云被设计出来是让事情变得更简单，但是要意识到“使用云”不仅仅是建立一个实例然后将应用丢进去就这么简单是非常重要的。这种一夜之间平地起高楼的事情是需要确定的情况的，要明白在云和过去的基于服务器硬件环境以及过去虚拟化环境是有着本质上的区别的。

在过去的环境中，还有那些过去的企业级应用，服务器和应用的运行都是被当作“宠物”看待的，他们被精心的照看和爱护，甚至每台服务器都有

自己的名字如“甘道夫“或”哆啦唉梦“,一旦他们生病了，则需要人去护理直到恢复健康。所有的这些设计表明应用是不可以停止。

在云的环境中，打个比方，服务器更像是牛群中的某一头牛。它们太多了，成千上万，命名的方式可能是类似的编号 NY-1138-Q，如果它们中的一个出问题了，管理员会直接抛弃它，再重新安装和启动一个即可。遗留的应用还没有准备好运行在此云环境中，所以很自然的会遭遇停运，丢失数据，甚至更加糟糕的。

在为云设计应用还有另外一些原因需要注意。一些是防御性的，例如一些应用并不能准确的确定在什么地方或什么样的硬件去启动，所以他们需要灵活性，即使做不到至少应该保持适应性更强。另外一些则是主动性的，例如，使用云的一个优点是其有高扩展性，所以应用需要被设计的能够使用这些优点，当然云还有更多的优点，也得一并考虑。

决定那些应用程序是可在云中运行

寻找适合于在云中运行的应用，还是有几种方法可以考虑的。

结构 那些大型的、单层次的、铁板一块的旧的应用是非常典型的不适合云环境的。将负载分割为多个实例会获得高效，所以部分系统失效不会特别影响其他部分的系统，或者说随应用的需要而横向扩展。

依赖 如果应用程序依赖特别的硬件，比如特别的芯片或者是类似指纹识别等外设，是不适合于云计算的。除非使用特别的方式来实现。同样的，如果应用依赖于操作系统或一组程序库不能用于云环境，或者无法在虚拟化环境中运行，这就真的成了问题了。

连通性 自包含的应用或它们所依赖的资源在云的环境无法实现，将无法运行。也许在一些情况下，通过自定义网络解决了这些问题，但是运行是否良好还得看选择云的环境。

耐久性和弹性 尽管有服务水平协议(SLA)的存在,但是还是会有一些坏的事情发生：服务器宕机，网络连接发生紊乱，多个租户无法访问服务。应用程序必须足够的稳固，以应对上述事情的发生。

设计云

这里有一些原则忠告，在为一个应用设计云的时候请时刻铭记：

- 做一个悲观主义者：承担一切失败，基于目标选择手段。善待那些将系统搞坏的程序。
- 将鸡蛋放在多个篮子里：考虑多个供应商，基于地理分区不同的数据中心，多可用的zones以容纳本地存在的隐患。可移植性的设计。

- 考虑效率：低效的设计将不可扩展。高效的设计可以无须花费多少钱即可轻松扩展。去掉那些不需要的组件或容量。
- 保持偏执：深度设计防御，通过构建在每一层和每个组件之间的安全确保零差错。不信任任何人。
- 但是也不要太偏执：不是所有的应用都需要钻石级的解决方案。为不同的服务水平协议、不同的服务层和安全级别等分别设计不同的架构。
- 管理数据：数据通常是最为灵活的也最复杂的一块，尤其是在云中或云集成的架构中。要在分析和传送数据付出更多的努力。
- 放手：自动化可以增加一致性、质量以及减少响应的时间。
- 分离和征服：尽可能分区和并行的分层。尽可能的确保组件足够小且可移植。在层之间使用负载均衡。
- 保持弹性：随着增加的资源，确保结果是增加的性能和扩展。减少资源要没有负面影响。
- 保持动态：激活动态配置变化诸如自动扩展，失效恢复，资源发现等，以适应变化的环境，错误的发生以及负载的变化。
- 贴近原则：通过移动高度密切的组件和相似数据靠近以减少延迟。
- 保持松散：松耦合，服务接口，分离关注度高的点，抽象并定义好的应用程序接口，交付灵活。
- 注意开销：自动扩展，数据传输，虚拟软件许可证，保留的实例等等均会快速增加每月的使用支出。密切监测使用情况。

第 2 章 通用型

目录

OpenStack通用型云通常被认为是构建一个云的起点。它们被设计为平衡的组件以及在整体的计算环境中不强调某个特定的领域。云的设计必须对计算、网络、存储等组件作必要的公平权衡。通用型云可以是私有云，也可以是公有云，当然也可以混合的环境，自身可以适用于各种用例。

注意

通用型云基于很普通的部署，不适用于特殊的环境或边缘案例的情况。

通常使用通用性的云包括：

- 提供一简单的数据库
- 一个web应用程序运行时环境
- 共享的应用开发平台
- 测试实验平台

在通用型云架构中，用例能够明确感受到横向扩展带来好处远远大于纵向扩展。

通用型云被设计为有一系列潜在用途或功能，不是为特殊的用例特殊设计。通用型架构是为满足80%用例而设计的。基础设施其本身就是一非常特别的用例，在设计过程将之用于基本模型未尝不可。通用型云被设计为适合通用应用的平台。

通用型云被限制在了大部分的基本组件，但是可以包含下面增加的资源，例如：

- 虚拟机磁盘镜像库

- Raw 块存储
- 文件或对象存储
- 防火墙
- 负载均衡器
- IP 地址
- 网络覆盖或者(VLANs)
- 软件集合

用户需求

当构建通用型云时，用户需要遵循 Infrastructure-as-a-Service (IaaS)模式，基于简单的需求为用户寻求最合适的平台。通用型云的用户需求并不复杂。尽管如此，也要谨慎对待，即使项目是较小的业务，或者诸如概念验证、小型实验平台等技术需求。

注意

以下用户考量的内容来自于云构建者的记录，并非最终用户。

成本 财务问题对任何组织来说都是头等大事。开销是一个重要的标准，通用型云以此作为基本，其它云架构环境以此作为参考。通用型云一般不会为特殊的应用或情况提供较划算的环境，除非是不赚钱或者成本已经定了，当选择或设计一个通用架构时开销才不是唯一考虑的。

上线时间 当构建一个通用型云时，有能力在灵活的时间内交付服务或产品，是业务普遍的一个情形。在今天高速发展的商务世界里，拥有在6个月内交付一款产品去替代2年完成的能力，这驱动着背后的决策者们决定构建通用型云。通用型云实现用户自服务，按需求获得访问计算、网络、存储资源，这将大大节省上线时间。

赢利空间 赢利空间对于云来说是个最基本的义务。一些通用型云为商业客户构建产品，但也有替代品，可能使通用云的正确选择。例如，一个小型的云服务提供商(CSP)会构建一个通用云而放弃大规模的扩展云，也许是因为没有足够的财力支撑，有或许是因为无法得知客户使用云的目的是什么。对于一些用户来说，高级云本身就意味着赢利。但是另外一些用户，通用型云所提供的基本功能会阻碍使用，导致潜在的收入机会的停滞。

法律需求

很多辖区对于云环境中的数据的保管及管理都有相关的法律上或者监管上的要求。这些规章的常见领域包括：

- 确保持久化数据的保管和记录管理以符合数据档案化需求的数据保留政策。
- 管理数据的所有权和责任的数据所有权政策。
- 管理位于外国或者其它辖区的数据存储问题的数据独立性政策。
- 数据合规性-基于常规某些类型的数据需要放在某些位置，同样的原因，不能放在其他位置更加的重要。

Examples of such legal frameworks include the data protection framework of the European Union and the requirements of the Financial Industry Regulatory Authority in the United States. Consult a local regulatory body for more information.

技术需求

技术云架构需求相比业务需求，比重占得更多一些。

性能	作为基础产品，通用型云并不对任何特定的功能提供优化性能。虽然通用型云希望能够提供足够的性能以满足所以用户的考虑，但是性能本身并不是通用型云所关注的。
非预先定义的使用模型	由于缺少预先定义的使用模型，导致用户在根本不知道应用需求的情况运行各式各样的应用。这里（OpenStack 通用型云，译者注）提供的独立性和灵活性的深度，也就只能是这么多了，不能提供更多的云场景。
按需或自服务应用	根据定义，云提供给最终用户通过简单灵活的方式自适应的计算能力、存储、网络和软件的能力。用户必须能够在不破坏底层主机操作的情况下扩展资源到满足自身。使用通用型云架构的一个优点就是，在有限的资源下启动，随着时间的增加和用户的需求增长而轻松扩张的能力。
公有云	对于一家公司对基于OpenStack构建一个商业公有云有兴趣的话，通用型架构模式也许是最好的选择。设计者毋须知道最终用户使用云的目的和具体负载。
内部消费(私有)云	组织需要决定在内部建立自己的云的逻辑。私有云的使用，组织需要维护整个架构中的控制点以及组件。

注意

用户打算既使用内部云又可访问外部云的组合。假如遇到类似的用例，也许值得一试的就是基于多个云的途径，考虑至少多个架构要点。

设计使用多个云，例如提供一个私有云和公有云的混合，有关“多云”的描述场景，请参考第 6 章 多区域 [111]。

安全性

安全应根据资产，威胁和脆弱性风险评估矩阵来实现。对于云领域来说，更加增加了计算安全、网络安全和信息安全等的需求。通用型云不被认为是恰当的选择。

技术因素

通用型云通常实现所包括的基本服务：

- 计算
- 网络
- 存储

这些服务中的每一个都有不同的资源需求。所以，提供所有服务意味着提供平衡的基础设施，直接关系到最终的服务，依据这些作出设计决定是非常重要的。

考虑到每个服务的不一致方面，需要设计为分离的因素，服务的复杂性，都会影响到硬件的选择流程。硬件的设计需要根据下列不同的资源池生成不同的配置，

- 计算
- 网络
- 存储

许多额外的硬件决策会影响到网络的架构和设施的规划。这些问题在OpenStack云的整个架构中扮演非常重要的角色。

规划计算资源

当设计计算资源池时，有很多情形会影响到用户的设计决策。例如，和每种hypervisor相关的处理器、内存和存储仅仅是设计计算资源时考虑的一

个因素。另外，将计算资源用户单一的池还是用户多个池也是有必要考虑的，我们建议用户设计将计算资源设计为资源池，实现按需使用。

一个计算设计在多个池中分配资源，会使云中运行的应用资源利用的最为充分。每个独立的资源池应该被设计为特定的类型的实例或一组实例提供服务，设计多个资源池可帮助确保这样，当实例被调度到hypervisor节点，每个独立的节点资源将会被分配到最合适的可用的硬件。这就是常见的集装箱模式。

使用一致的硬件来设计资源池中的节点对装箱起到积极作用。选择硬件节点当作计算资源池的一部分最好是拥有一样的处理器、内存以及存储类型。选择一致的硬件设计，将会在云的整个生命周期展现出更加容易部署、支持和维护的好处。

OpenStack支持配置资源超分配比例，即虚拟资源比实际的物理资源，目前支持CPU和内存。默认的CPU超分配比例是16：1，默认的内存超分配比例是1.5：1。在设计阶段就要想要是否决定调整此超分配比例，因为这会直接影响到用户计算节点的硬件布局。

举例说明，想像一下，现在有一个m1.small类型的实例，使用的是1 vCPU,20GB的临时存储，2048MB的内存。当设计为此类实例提供计算资源池的硬件节点时，考虑节点需要多少个处理器、多大的磁盘、内存大小才可以满足容量需求。对于一个有2颗10个核的CPU，并开启超线程的服务器，基于默认的CPU超分配比例16：1来计算的话，它总共能运行640(2x10x2x16)个m1.small类型的实例。同样，基于默认的内存超分配比例1.5：1来计算的话，用户则需要至少853GB(640x2048/1.5x1048)内存。当基于内存来丈量服务器节点时，考虑操作系统本身和其服务所需要使用的内存也是非常重要的。

处理器的选择对于硬件设计来讲实在是太过重要了，尤其是对比不同处理器之间的特性和性能。一些近来发布的处理器，拥有针对虚拟化的特性，如硬件增强虚拟化，或者是内存分页技术(著名的EPT影子页表)。这些特性对于在云中运行的虚拟机来说，有着非常关键的影响。

考虑到云中计算需求的资源节点也是很重要的，资源节点即非hypervisor节点，在云中提供下列服务：

- 控制器
- 对象存储
- 块存储
- 联网服务

处理器核数和线程数和在资源节点可以运行多少任务线程是有直接关联的。结果就是，用户必须作出设计决定，直接关联的服务，以及为所有服务提供平衡的基础设施。

在通用型云中负载是不可预测的，所以能将每个特别的用例都做到胸有成竹是非常困难的。在未来给云增加计算资源池，这种不可预测是不会带来任何问题的。在某些情况下，在确定了实例类型的需求可能不足以需要单独的硬件设计。综合来看，硬件设计方案是先满足大多数通用的实例来启动的，至于以后，寻求增加额外的硬件设计将贯穿整个多样的硬件节点设计和资源池的架构。

规划网络资源

传统的OpenStack云是有多个网段的，每个网段都提供在云中访问不仅是操作人员还可以是租户的资源。此外，网络服务本身也需要网络通讯路径以和其他网络隔离。当为通用型云设计网络服务时，强烈建议规划不论是物理的还是逻辑的都要将操作人员和租户隔离为不同的网段。进一步的建议，划分出另外一个网段，专门用于访问内部资源，诸如消息队列、数据库等各种云服务。利用不同网段隔离这些服务对于保护敏感数据以及对付没有认证的访问很有帮助。

基于云中实例提供的服务的需求，网络服务的选择将会是影响用户设计架构的下一个决定。

在作为OpenStack计算组件的部分遗留网络(nova-network)和OpenStack网络(neutron)之间作出选择，会极大的影响到云网络基础设施的架构和设计。

传统联网方式(nova-network)

遗留的网络服务(nova-network)主要是一个二层的网络服务，有两种模式的功能实现。在遗留网络中，两种模式的区别是它们使用VLAN的方式不同。当使用遗留网络用于浮动网络模式时，所有的网络硬件节点和设备通过一个二层的网段来连接，提供应用数据的访问。

当云中的网络设备可通过VLANs支持分段时，遗留网络的第二种模式就可操作了。此模式中，云中的每个组户都被分配一个网络子网，其是映射到物理网络的VLAN中的。尤其重要的一点，要记得在一个生成树域里VLANs的最大数量是4096.在数据中心此限制成为了可能成长的一个硬性限制。当设计一个支持多租户的通用型云时，特别要记住使用和VLANs一起使用遗留网络，而且不使用浮动网络模式。

另外的考虑是关于基于遗留网络的网络的管理是由云运维人员负责的。租户对网络资源没有控制权。如果租户希望有管理和创建网络资源的能力，如创建、管理一个网段或子网，那么就有必要安装OpenStack网络服务，以提供租户访问网络。

OpenStack 网络 (neutron)	OpenStack网络 (neutron) 是第一次实现为租户提供全部的控制权来建立虚拟网络资源的网络服务。为了实现给租户流量分段，通常是基于已有的网络基础设施来封装通信路径，即以隧道协议的方式。这些方法严重依赖与特殊的实现方式，大多数通用的方式包括GRE隧道，VXLAN封 装以及VLAN标签。

首先建议的设计是至少划分三个网段，第一个是给租户和运维人员用于访问云的REST 应用程序接口。这也是通常说的公有网络。在多数的用例中，控制节点和swift代理是唯一的需要连接到此网段的设备。也有一些用例中，此网段也许服务于硬件的负载均衡或其他网络设备。

第二个网段用于云管理员管理硬件资源，也用于在新的硬件上部署软件和服务的配置管理工具。在某些情况下，此网段也许用于内部服务间的通信，包括消息总线和数据库服务。介于此网段拥有高度安全的本质，需要将此网络设置为未经授权不得访问。此网段在云中的所有硬件节点需要互联互通。

第三个网段用于为应用程序和消费者访问物理网络，也为最终用户访问云中运行的应用程序提供网络。此网络是隔离能够访问云API的网络，并不能够直接和云中的硬件资源通信。计算资源节点需要基于此网段通信，以及任何的网络网关服务将允许应用程序的数据可通过物理网络到云的外部。

规划存储资源

OpenStack有两个独立的存储服务需要考虑，且每个都拥有自己特别的设计需求和目标。除了提供存储服务是他们的主要功能之外，在整个云架构中，需要额外考虑的是它们对计算/控制节点的影响。

规划OpenStack对象存储

当为OpenStack对象存储设计硬件资源时，首要的目标就尽可能的为每个资源节点加上最多的存储，当然也要在每TB的花费上保持最低。这往往涉及到了利用服务器的容纳大量的磁盘。无论是选择使用2U服务器直接挂载磁盘，还是选择外挂大量的磁盘驱动，主要的目标还是为每个节点得到最多的存储。

我们并不建议为OpenStack对象存储投资企业级的设备。OpenStack对象存储的一致性和分区容错保证的数据的更新，即使是在硬件损坏的情况下仍能保证数据可用，而这都无须特别的数据复制设备。

OpenStack对象存储一个亮点就是有混搭设备的能力，基于swift环的加权机制。当用户设计一个swift存储集群时，建议将大部分>的成本花在存储上，保证永久可用。市场上多数的服务器使用4U，可容纳60块甚至更多的磁盘驱动器，因此建议在1U空间内置放最多的驱动器，就是最好的每TB花费。进一步的建议，在对象存储节点上没必要使用RAID控制器。

为了实现数据存储和对象一样的持久和可用，设计对象存储资源池显得非常的重要，在硬件节点这层之外的设计，考虑机柜层或区域层是非常重要的，在对象存储服务中配置数据复制多少份保存(默认情况是3份)。每份复制的数据最好独立的可用区域中，有独立的电源、冷却设备以及网络资源。

对象存储节点应该被设计为在集群中不至于区区几个请求就拖性能后腿的样子。对象存储服务是一种频繁交互的协议，因此确定使用多个处理器，而且要多核的，这样才可确保不至于因为IO请求将服务器搞垮。

规划OpenStack块存储

当设计OpenStack块存储资源节点时，有助于理解负载和需求，即在云中使用块存储。由于通用型云的使用模式经常是未知的。在>此建议设计块存储池做到租户可以根据他们的应用来选择不同的存储。创建多个不同类型的存储池，得与块存储服务配置高级的存储调度相结合，才可能为租户提供基于多种不同性能级别和冗余属性的大型目录存储服务。

块存储还可以利用一些企业级存储解决方案的优势。由硬件厂商开发的插件驱动得以实现。基于OpenStack块存储有大量的企业存储写了它们带外的插件驱动(也有很大一部分是通过第三方渠道来实现的)。作为通用型云使用的是直接挂载存储到块存储节点，如果未来需要为租户提供额外级别的块存储，只须增加企业级的存储解决方案即可。

决定在块存储节点中使用RAID控制卡主要取决于应用程序对冗余和可用性的需求。应用如果对每秒输入输出(IOPS)有很高的要求，不仅得使用RAID控制器，还得配置RAID的级别值，当性能是重要因素时，建议使用高级别的RAID值，相对比的情况是，如果冗余的因素考虑更多谢，那么就使用冗余RAID配置，比如RAID5或RAID6。一些特殊的特性，例如自动复制块存储卷，需要使用第三方插件和企业级的块存储解决方案，以满足此高级需求。进一步讲，如果有对性能有极致的要求，可以考虑使用告诉的SSD磁盘，即高性能的flash存储解决方案。

软件选择

软件筛选的过程在通用型云架构中扮演了重要角色。下列的选择都会在设计云时产生重大的影响。

- 选择操作系统
- 选择OpenStack软件组件
- 选择Hypervisor
- 选择支撑软件

操作系统(OS)的选择在云的设计和架构中扮演着重要的角色。有许多操作系统是原生就支持OpenStack的，它们是：

- Ubuntu
- 红帽企业Linux(RHEL)
- CentOS
- SUSE Linux Enterprise Server (SLES)

注意

原生并非是限制到某些操作系统，用户仍然可以自己选择Linux的任何发行版(甚至是微软的Windows),然后从源码安装OpenStack(或者编译为某个发行版的包)。尽管如此，事实上多数组织还是会从发行版支持的包或仓库去安装OpenStack(使用分发商的OpenStack软件包也许需要他们的支持)。

操作系统的选择会直接影响到hypervisor的选择。一个云架构会选择Ubuntu,RHEL,或SLES等，它们都拥有灵活的hypervisor可供选择，同时也被OpenStack 计算(nova)所支持，如KVM,Xen,LXC等虚拟化。一个云架构若选择了Hyper-V,那么只能使用Windows服务器版本。同样的，一个云架构若选择了XenServer,也限制到基于CenOS dom0操作系统。

影响操作系统-hypervisor选择的主要因素包括：

用户需求	选择操作系统-虚拟机管理程序组合，首先且最重要的是支持用户的需求。
支持	选择操作系统-虚拟机管理程序组合需要OpenStack支持。
互操作性	在OpenStack的设计中，为满足用户需求，需要操作系统的hypervisor在彼此的特性和服务中要有互操作性。

虚拟机管理程序

OpenStack支持多种Hypervisor,在单一的云中可以有一种或多种。这些Hypervisor包括：

- KVM (and QEMU)
- XCP/XenServer
- vSphere (vCenter and ESXi)
- Hyper-V
- LXC
- Docker
- 裸金属

支持hypervisor和它们的兼容性的完整列表可以从

通用型云须确保使用的hypervisor可以支持多数通用目的的用例，例如KVM或Xen。更多特定的hypervisor需要根据特定的功能和支持特性需求来做出选择。在一些情况下，也许是授权所需，需要运行的软件必须是在认证的hpervisor中，比如来自VMware，微软和思杰的产品。

通过OpenStack云平台提供的特性决定了选择最佳的hyperviosr。举个例子，要使通用型云主要支持基于微软的迁移，或工作人员>所管理的是需要特定技能的去管理hypervisor或操作系统，Hyper-V也许是最好的选择。即使是决定了使用Hyper-V，这也并不意味着不可以去管理相关的操作系统，要记得OpenStack是支持多种hypervisor的。在设计通用型用时需要考虑到不同的hypervisor有他们特定的硬件需>求。例如欲整合VMware的特性：活迁移，那么就需要安装安装vMotion,给ESXi hypervisor所使用的VCenter/vSphere，这即是增加的基础设施需求。

在混合的hypervisor环境中，等于聚合了计算资源，但是各个hypervisor定义了各自的能力，以及它们分别对软、硬件特殊的需求等。这些功能需要明确暴露给最终用户，或者通过为实例类型定义的元数据来访问。

OpenStack 组件

通用型OpenStack云的设计，使核心的OpenStack服务的互相紧密配合为最终用户提供广泛的服务。在通用型云中建议的OpenStack核心服务有：

- OpenStack 计算 (nova)
- OpenStack 网络 (neutron)

- OpenStack 镜像服务 (glance)
- OpenStack 认证 (keystone)
- OpenStack 仪表盘 (horizon)
- Telemetry module (ceilometer)

通用型云还包括OpenStack 对象存储 (swift).。选择OpenStack 块存储 (cinder) 是为应用和实例提供持久性的存储，

注意

尽管如此，依赖于用例，这些是可选。

增强软件

通用型OpenStack的软件部署不仅仅是OpenStack特定的组件。一个典型的部署，如提供支撑功能的服务，须包含数据库，消息队列，以及为OpenStack环境提供高可用的软件。设计的决定围绕着底层的消息队列会影响到控制器服务的多寡，正如技术上为数据库提供高弹性功能，例如在MariaDB之上使用Galera。在这些场景中，服务的复制依赖于预订的机器数，因此，考虑数据库的 节点，例如，至少需要3个节点才可满足Galera节点的失效恢复。随着节点数量的增加，以支持软件的特性，考虑机柜的空间和交换机的端口显得非常的重要。

多数的通用型部署使用硬件的负载均衡来提供API访问高可用和SSL终端，但是软件的解决方案也要考虑到，比如HAProxy。至关重要的是软件实现的高可用也很靠谱。这些高可用的软件Keepalived或基于Corosync的Pacemaker。Pacemaker和Corosync配合起来可以提供双活或者单活的高可用配置，至于是否双活取决于OpenStack环境中特别的服务。使用Pacemaker会影响到设计，假定有至少2台控制器基础设施，其中一个节点可在待机模式下运行的某些服务。

Memcached是一个分布式的内存对象缓存系统，Redia是一个key-value存储系统。在通用型云中使用这两个系统来减轻认证服务的负载。memcached服务缓存令牌，基于它天生的分布式特性可以缓减授权系统的瓶颈。使用memcached或Redis不会影响到用户的架构设计，虽然它们会部署到基础设施节点中为OpenStack提供服务。

性能

OpenStack的性能取决于和基础设施有关的因素的多少以及控制器服务的多少。按照用户需求性能可归类为通用网络性能，计算资源性能，以及存储系统性能。

控制器基础设施

控制器基础设施节点为最终用户提供的管理服务，以及在云内部为运维提供服务。控制器较典型的现象，就是运行消息队列服务，在每个服务之间携带传递系统消息。性能问题常和消息总线有关，它会延迟发送的消息到应该去的地方。这种情况的结果就是延迟了实际操作的功能，如启动或删除实例、分配一个新的存储卷、管理网络资源。类似的延迟会严重影响到应用的反应能力，尤其是使用了自动扩展这样的特性。所以运行控制器基础设施的硬件设计是头等重要的，具体参考上述硬件选择一节。

控制器服务的性能不仅仅限于处理器的强大能力，也受限于所服务的并发用户。确认应用程序接口和Horizon服务的负载测试可以>承受来自用户客户的压力。要特别关注OpenStack认证服务(Keystone)的负载，Keystone为所有服务提供认证和授权，无论是最终用户还是OpenStack的内部服务。如果此控制器服务没有正确的被设计，会导致整个环境的性能低下。

网络性能

通用型OpenStack云，网络的需求其实帮助决定了其本身的性能能力。例如，小的部署环境使用千兆以太网(GbE)，大型的部署环境或者许多用户就要求使用10 GbE。运行着的实例性能收到了它们的速度限制。设计OpenStack环境，让它拥有可以运行混合网络的能力是可能的。通过利用不同速度的网卡，OpenStack环境的用户可以选择不同的网络去满足不同的目的。

例如，web应用的实例在公网上运行的是OpenStack网络中的1 GbE，但是其后端的数据库使用的是OpenStack网络的10 GbE以复制数据，在某些情况下，为了更大的吞吐量使用聚合链接这 样的设计。

网络的性能可以考虑有硬件的负载均衡来帮助实现，为云抛出的API提供前端的服务。硬件负载均衡通常也提供SSL终端，如果用户的环境有需求的话。当实现SSL减负时，理解SSL减负的能力来选择硬件，这点很重要。

计算主机

为计算节点选择硬件规格包括CPU,内存和磁盘类型，会直接影响到实例的性能。另外直接影响性能的情形是为OpenStack服务优化参数，例如资源的超分配比例。默认情况下OpenStack计算设置16:1为CPU的超分配比例，内存为1.5:1。运行跟高比例的超分配即会导致有些服务无法启动。调整您的计算环境时，为了避免这种情况下必须小心。运行一个通用型的OpenStack环境保持默认配置就可以，但是也要检测用户的环境中使用量的增加。

存储性能

当考虑OpenStack块设备的性能时，硬件和架构的选择就显得非常重要。块存储可以使用企业级后端系统如NetApp或EMC的产品，也可以使用横向扩展存储如GlusterFS和Ceph，更可以是简单的在节点上直接附加的存储。块存储或许是云所关注的贯穿主机网络的部署，又或许是前端应用程序接口所关注的流量性能。无论怎么，都得在控制器和计算主机上考虑使用专用的数据存储网络和专用的接口。

当考虑OpenStack对象存储的性能时，有几样设计选择会影响到性能。用户访问对象存储是通过代理服务，它通常是在硬件负载均衡之后。由于高弹性是此存储系统的天生特性，所以复制数据将会影响到整个系统的性能。在此例中，存储网络使用10 GbE(或更高)网络是我们所建议的。

可用性

在OpenStack架构下，基础设施是作为一个整体提供服务的，必须保证可用性，尤其是建立了服务水平协议。确保网络的可用性，在完成设计网络架构后不可以有单点故障存在。考虑使用多个交换机、路由器以及冗余的电源是必须考虑的，这是核心基础设施，使用bonding网卡、提供多个路由、交换机高可用等。

OpenStack自身的那些服务需要在跨多个服务器上部署，不能出现单点故障。确保应用程序接口的可用性，作为多个OpenStack服务的成员放在高可用负载均衡的后面。

OpenStack其本身的部署是期望高可用的，实现此需要至少两台服务器来完成。他们可以运行所有的服务，由消息队列服务连接起来，消息队列服务如RabbitMQ或QPID，还需要部署一个合适的数据库服务如MySQL或MariaDB。在云中所有的服务都是可横向扩展的，其实后端的服务同样需要扩展。检测和报告在服务器上的使用和采集，以及负载测试，都可以帮助作出横向扩展的决定。

当决定网络功能的时候务必小心谨慎。当前的OpenStack不仅支持遗留网络(nova-network)系统还支持新的，可扩展的OpenStack网络(neutron)。二者在提供高可用访问时有各自的优缺点。遗留网络提供的网络服务的代码是由OpenStack计算来维护的，它可提供移除来自路由的单点故障特性，但是此特性在当前的Openstack网络中不被支持。遗留网络的多主机功能受限于仅在运行实例的主机，一旦失效，此实例将无法被访问。

另一方面，当使用OpenStack网络时，OpenStack控制器服务器或者是分离的网络主机掌控路由。对于部署来说，需要在网络中满足此特性，有可能使用第三方软件来协助维护高可用的3层路由。这么做允许通常的应用程序接口来控制网络硬件，或者基于安全的行为提供复杂多层的web应用。从

OpenStack网络中完全移除路由是可以的，取而代之的是硬件的路由能力。在此情况下，交换基础设施必须支持3层路由。

OpenStack网络和遗留网络各有各的优点和缺点。它们有
效的和支持的属性适合不同的网络部署模式，详细描述
见 "http://docs.openstack.org/openstack-ops/content/
network_design.html#network_deployment_options">OpenStack

确保用户的部署有足够的后备能力。举例来说，部署是拥有两台基础设施的控制器节点，此设计须包括控制器可用。一旦丢失单个控制器这种事件发生，云服务将会停止对外服务。当有高可用需求的设计时，解决此需求的办法就是准备冗余的、高可用的控制器节点。

应用程序的设计也必须记入到云基础设施的能力之中。如果计算主机不能提供无缝的活迁移功能，就必须预料到一旦计算节点失效，运行在其上的实例，以及其中的数据，都不可访问。反过来讲，提供给用户的是具有高级别的运行时间保证的实例，基础设施必须被部署为没有任何的单点故障，即使是某计算主机消失，也得马上有其他主机顶上。这需要构建在企业级存储的共享文件系统之上，或者是OpenStack块存储，以满足保证级别和对应的服务匹配。

关于OpenStack高可用更多信息，请阅读 OpenStack 高可用指南.

安全性

一个安全域在一个系统下包括让用户、应用、服务器和网络共享通用的信任需求和预期。典型情况是他们有相同的认证和授权的需求和用户。

这些安全域有：

- 公有
- 客户机
- 管理
- 数据

这些安全域可以映射给分开的OpenStack部署，也可以是合并的。例如，一些部署的拓扑合并了客户机和数据域在同一物理网络，其他一些情况的网络是物理分离的。无论那种情况，云运维人员必须意识到适当的安全问题。安全域须制定出针对特定的OpenStack部署拓扑。所谓的域和其信任的需求取决于云的实例是公有的，私有的，还是混合的。

公有安全域对于云基础设施是完全不可信任的区域。正如将内部暴露给整个互联网，或者是没有认证的简单网络。此域一定不可以被信任。

典型的用户计算服务中的实例对实际的互联互通，客户安全域，在云中掌握着由实例生成的计算机数据，但是不可以通过云来访问，包括应用程序接口调用。公有云或私有云提供商不会严格的控制实例的使用，以及谁受限访问互联网，所以应指定此域是不可信任的。或许私有云提供商可以稍宽松点，因为是内部网络，还有就是它可以控制所有的实例和租户。

管理安全域即是服务交互的集中地，想一下“控制面板”，在此域中网络传输的都是机密的数据，比如配置参数、用户名称、密码等。多数部署此域为信任的。

数据安全域主要关心的是OpenStack存储服务相关的信息。多数通过此网络的数据具有高度机密的要求，甚至在某些类型的部署中，还有高可用的需求。此网络的信任级别高度依赖于其他部署的决定。

在一个企业部署OpenStack为私有云，通常是安置在防火墙之后，和原来的系统一并认为是可信任网络。云的用户即是原来的员工，由公司本来的安全需求所约束。这导致推动大部分的安全域趋向信任模式。但是，当部署OpenStack为面向公用的角色时，不要心存侥幸，它被攻击的几率大大增加。例如，应用程序接口端点，以及其背后的软件，很容易成为一些坏人下手的地方，获得未经授权的访问服务或者阻止其他人访问正常的服务，这可能会导致丢失数据、程序失效，乃至名声。这些服务必须被保护，通过审计以及适当的过滤来实现。

无论是公有云还是私有云，管理用户的系统必须认真的考虑。身份认证服务允许LDAP作为认证流程的一部分。Including such systems in an OpenStack deployment may ease user management if integrating into existing systems.

理解用户的认证需要一些敏感信息如用户名、密码和认证令牌是非常重要的。基于这个原因，强烈建议将认证的API服务隐藏在基于硬件的SSL终端之后。

更多关于OpenStack安全的信息，请访问OpenStack 安全指南。

运营因素

在构建过程中的规划和设计阶段，能够考虑到运维的内容显得异常的重要。对通用型云来说，运维因素影响着设计的选择，而且运维人员担任着维护云环境的任务，以及最初大规模的安装、部署。

服务水平协议(SLA)的设定，就是知道何时以及何地实现冗余和高可用会直接影响到预期。SLA是提供保证服务可用性合同义务。他们定义的可用性级别驱动着技术的设计，若不能实现合同义务，则会得到相应的惩罚。

影响设计的SLA术语包括：

- 由多基础设施服务和高可用负载均衡实现API可用性保证。
- 网络运行时间保证影响这交换机的设计，需要交换机的冗余，以及其电源的冗余。
- 网络安全规则的需求需要在部署时被处理。

支持和维护

为了能够在安装之后能够支持和维护，OpenStack云管理需要运维人员理解设计架构内容。运维人员和工程师人员的技能级别，以及他们之间的界限，依赖于安装的规模大小和目的。大型的云服务提供商或者电信运营商，会有针对性的培训，专门的运维部门，小型的实现的话，支持人员通常是工程师、设计师、和运维人员的合体。

维护OpenStack安装需要大量的技术技能。例如，如果用户既做架构有设计试图减轻运维的负担，那一定建议运维自动化。或许用第三方管理公司的专长去管理OpenStack部署也是不错的选择。

监控

OpenStack云需要合适的检测平台来确保错误可以及时捕获，能够更好的管理。一些特别的计量值需要重点监测的有：

- 镜像磁盘使用
- 计算API的响应时间

借助已有的监测系统是一种有效的检查，确保OpenStack环境可以被监测。

宕机时间

为有效的运行云，开始计划宕机包括建立流程和支持的架构有下列内容：

- 计划内(维护)
- 计划外(系统出错)

保持弹性的系统，松耦合的组件，都是SLA的需求，这也意味着设计高可用(HA)需要花费更多。

举例来说，如果计算主机失效，这就是运维要考虑的；需要将之前运行在其上的实例从快照中恢复或者重新建一个一模一样的实例。整个应用程序

的设计都会被影响，通用型云不强求有提供将一个实例从一台主机迁移到另外一台主机的能力。假如用户期望其应用是被设计为容错的，那么就要额外考虑支持实例的迁移，也需要额外的支持服务，包括共享存储挂载到计算主机，就需要被部署。

容量计划

在通用型云环境中的容量限制包括：

- 计算限制
- 存储限制

计算环境的规模和支撑它的OpenStack基础设施控制器节点之间的关系是确定的。

增加支持计算环境的规模，会增加网络流量、消息，也会增加控制器和网络节点的负载。有效的监测整个环境，对决定扩展容量很有帮助。

计算节点自动挂接到OpenStack云，结果就是为OpenStack云添加更多的计算容量，亦即是横向扩展。此流程需要节点是安置在合适的可用区域并且是支持主机聚合。当添加额外的计算节点到环境中时，要确保CPU类型的兼容性，否则可能会使活迁移的功能失效。扩展计算节点直接的结果会影响到网络及数据中心的其他资源，因为需要增加机柜容量以及交换机。

通过评估平均负载，在计算环境中调整超分配比例来增加运行实例的数量是另外一个办法。重要的是记住，改变CPU超分配比例有负面影响以及引起其它实例故障。加大超分配的比例另外的风险是当计算节点失效后会引发更多的实例失效。

计算主机可以按需求来进行相应的组件升级，这就是传说中的纵向扩展。升级更多核的CPU,增加整台服务器的内存，要视运行的应用是需要CPU更紧张，还是需要内存更急切，以及需要多少。

磁盘容量不足会给整个性能带来负面影响，会波及到CPU和内存的使用。这取决于后端架构的OpenStack块存储层，可以是为企业级存储系统增加磁盘，也可以是安装新的块存储节点，也可以是为计算主机直接挂接存储，也可以为实例从共享存储中添加临时空间。都有可能。

有关这些题目的更加深入的讨论，请参考 OpenStack 运维实战

架构

硬件选择分为三大块：

- 计算
- 网络
- 存储

为通用型OpenStack云选择硬件，印证了云是没有预先定义的使用模型的。通用型云为运行广泛的应用而设计，而每种应用对资源利用有着各自不同的需求。这些应用不会是下面分类以外的类型：

- 内存密集型
- CPU密集型
- 存储密集型

为通用型OpenStack云选择硬件必须提供所有主要资源的平衡性。

确定硬件的形式，也许更适合通用型OpenStack云，因为通用型有对资源的对等(接近对等)平衡的需求。服务器硬件须提供如下资源平衡细节描述：

- 计算容量(内存和CPU)对等(或接近对等)
- 网络容量(连接数量和速度)
- 存储能力(每秒输入/输入操作(IOPS)，GB或TB)

服务器硬件是围绕4个冲突的维度来进行评估的。

服务器密度	关于多少台服务器能够放下到一个给定尺寸的物理空间的量度，比如一个机柜单位[U]。
资源容量	一个给定的服务器能够提供的 CPU 数目、内存大小以及存储大小。
延伸性	服务器上还能够继续添加直到到达其限制的资源数目。
成本	相对硬件的购买价格，与构建系统所需要的设计功力的级别成反比。

增加服务器密度意味这损失资源容量和扩展性，同理，增加资源容量和扩展性会增加开销和减少服务器密度。结果就是，为通用型OpenStack架构决定最合适的服务器硬件意味着怎么选择都会影响到其他设计。从以下列表元素中作出选择：

- 刀片服务通常都支持双插槽多核的CPU，对于通用型云部署此配置通常被考虑为“甜点“。刀片拥有卓越的密度。举例，无论是 HP BladeSystem

还是 Dell PowerEdge M1000e，均支持在占用10U的机柜空间下达到16台服务器的密度。尽管如此，刀片服务器本身具有有限的存储和网络容量，另外扩展到多台刀片服务器也是有局限的。

- 1U机架式服务器仅占用1U的机柜空间，他们的有点包括高密度，支持双插槽多核的CPU，支持内存扩展。局限性就是，有限的存储容量、有限的网络容量，以及有限的可扩展性。

- 2U的机架式服务器相比1U服务器，提供可扩展的存储和网络容量，但是相应的降低了服务器密度(1U机架式服务器密度的一半)

- 高U的机架式服务器，比如4U的服务器，可提供更多的CPU容量，通常可提供四个甚至8个CPU插槽。这些服务器拥有非常强的可扩展性，还拥有升级的最好条件。尽管如此，它们也意味着低密度和更高的开销。

- “雪撬服务器”亦是机架式服务器的一种，支持在单个2U或3U的空间放置多个独立的服务器。此种类型增加的密度超过典型的1U-2U机架服务器，但是单个服务器支持的存储和网络容量往往受到限制。

支持通用型OpenStack云的服务器硬件最佳类型，是由外部的业务和成本因素所驱动的。没有那个单一的参考架构可以满足所有的实现，决定一定要依据用户需求，技术因素，以及运维因素。这里有一些能够影响到服务器硬件的选择的关键的因素：

实例密度
: 对于通用型OpenStack云来说规模大小是一个很重要的考虑因素。预料或预期在每个hypervisor上可以运行多少实例，是部署中衡量大小的一个普遍元素。选择服务器硬件需要支持预期的实例密度。

主机密度
: 物理数据中心【相对应的有虚拟数据中心，译者注】受到物理空间、电源以及制冷的限制。主机(hypervisor)的数量需要适应所给定的条件(机架，机架单元，地板)，这是另外一个重要衡量规模大小的办法。地板受重通常是一个被忽视的因素。数据中心的地板必须能够支撑一定数量的主机，当然是放在一个机架或一组机架中。这些因素都需要作为主机密度计算的一部分和服务器硬件的选择来考虑，并需要通过。

电源密度
: 数据中心拥有一定的电源以满足指定的机架或几组机架。老的数据中心拥有的电源密度一个低于每机架20AMP。近年来数据中心通常支持电源密度为高于每机架120 AMP。选择过的服务器硬件必须将电源密度纳入考虑范围。

网络连通性
: 已选择的服务器硬件必须有合适数量的网络连接，以及恰当的类型，目的是支持建议的架构。为了确保这点，最小的情况也需要至少为每个机架连接两条网络。如果架构要求更多的冗余，也许

就需要和电信供应商确认有多少线路可以连接。大多数的数据中心具备这个能力。

形式因素或体系结构的选择会影响服务器硬件的选择，例如，假如设计一个横向扩展的存储架构，那么服务器硬件的选择就得小心谨慎的考虑需求，以匹配商业解决方案的需求。

确保选择的服务器硬件的配置支持足够的存储容量(或存储扩展性)，以满足所选择的横向扩展存储解决方案的需求。举例，如果需求是基于集中存储解决方案，比如存储厂商的集中存储阵列需要InfiniBand或FDDI连接，那么服务器硬件就需要安装对应的网卡来兼容此特定厂商的存储阵列。

同样的，网络架构会影响到服务器硬件的选择，反之亦然。举例，确保服务器配置了足够的网络端口和扩展卡，以支持所有的网络需求。因为有网络扩展卡的兼容性问题，所以要意识到潜在影响和兼容性问题，以及架构中的其他组件。

选择存储硬件

存储硬件结构主要是由所选择的存储体系结构来确定。存储架构的选择，以及相应的存储硬件， 通过评估可能的解决方案，即针对关键的因素，用户需求，技术因素和运维因素来做决定。需要被并入存储体系架构的因素包括：

成本 存储在整个系统的开销中占有很大一部分。对于一个组织来说关心的是提供商的支持，以及更加倾向于商业的存储解决方案，尽管它们的价格是很高。假如最初的投入希望是最少的，基于普通的硬件来设计系统也是可接受的，这就是权衡问题，一个是潜在的高支持成本，还有兼容性和互操作性的高风险问题。

可扩展性 可扩展性是通用型OpenStack云主要考虑的因素。正因为通用型云没有固定的使用模式，也许导致预测最终的使用大小是很困难的。也许就有必要增加初始部署规模以应对数据增长和用户需求。

延伸性 对于通用型OpenStack云存储解决方案来说，可扩展性也是主要的架构因素。一个存储解决方案能够扩展到50PB,而另外一个只能扩到10PB,前者明显比后者的扩展性强好多倍。此扩展与scalability有关，但是不一样，scalability更加倾向于解决方案的性能衡量。举例，一个存储架构为专注于开发者平台的云和商业产品云在expandability和scalability上就不可能一样。

使用在服务器直接挂接存储(DAS)来作为横向扩展存储解决方案，是比较适合通用型OpenStack云的。举例，无论是采用计算主机类似的网格计算解决方案，还是给主机提供专用的块存储，这样的存储方式是可能的。当在

计算主机合适的硬件上部署存储时，需要支持存储和计算服务在同一台硬件。

正确理解云服务的需求对于决定该使用什么横向扩展解决方案有非常大的帮助。假如现在需要一个单一的，高度可扩展，高度垂直化，可扩展的性能等需求，那么在设计中采用中心化的存储阵列就应该被考虑。一旦一种方法已经被确定，存储硬件需要在此基础上的标准来进行选择。

这个列表扩展了在设计通用型OpenStack云可能产生的影响，包括对特定存储架构(以及相应的存储硬件)

连通性 确保连通性，如果存储协议作为存储解决方案的一部分使用的是非以太网，那么选择相应的硬件。如果选择了中心化的存储阵列，hypervisor若访问镜像存储就得能够连接到阵列。

用量 特定的存储架构如何使用是决定架构的关键。一些配置将直接影响到架构，包括用于hypervisor的临时实例存储，OpenStack对象存储使用它来作为对象存储服务。

实例和镜像存放地 实例和镜像存放在哪里会影响到架构。

服务器硬件 如果是一个囊括了DAS的横向存储架构的解决方案，它将影响到服务器硬件的选择。这会波及到决策，因为影响了主机密度，实例密度，电源密度，操作系统-Hypervisor，管理工具及更多。

通用型OpenStack云有很多属性，影响存储硬件的选择的关键因素有以下：

容量 对于资源节点的硬件选择来说，须为云服务提供足够的存储。定义初始的需求和确保设计是支持增加的能力是很重要的。为对象存储选择硬件节点须支持大量的廉价磁盘，无须冗余，无须RAID控制卡。为块存储选择硬件节点须具备支持高速存储解决方案，拥有RAID控制卡保证性能，以及在硬件层的冗余能力。选择硬件RAID控制器，是其可以自动修复损坏的阵列，以帮助人工的介入，替代及修复已经降级或损坏的存储设备。

性能 对于对象存储的磁盘选择来说，不需要速度太快的磁盘。我们建议对象存储节点利用最佳的每TB开销比即可。与此相反，为块存储服务选择磁盘，则须利用提高性能的特性来选择，比如使用SSD或flash磁盘来提供高性能的块存储池。即使是为实例运行的临时的磁盘也得考虑性能。如果计算池希望大量采用临时存储，或者是要求非常高的性能，就需要部署上述块存储硬件的解决方案。

容错 对象存储资源节点对于硬件容错或RAID控制器没有任何的要求。没必要为对象存储硬件规划容错是因为对象存储服务本身就提供了在zone

之间的复制。块存储节点、计算节点以及云控制器节点都须在硬件层有容错机制，使用硬件RAID控制器的话请确认RAID配置的级别。RAID级别的选择对于云的需求来说须性能和可用性兼顾。

选择网络硬件

选择网络架构决定了那些网络硬件将会被使用。而网络的软件是由选择好的网络硬件所决定的。举例，选择了网络硬件仅支持千兆以太网 (GbE)会影响到整个设计。同样地，决定使用 万兆以太网(10GbE)仅会影响到整个设计的局部一些区域。

有许多设计上的细节需要被考虑。选择好了网络硬件(或网络软件)会影响到管理工具的使用。除非出现下面情况：越来越多的用户倾向于使用"开放"的网络软件来支持广泛的网络硬件，意味着某些情况下网络硬件和软件之间的关系是不是被严格定义。关于此类型的软件Cumulus Linux即是个例子，Cumulus Linux具有运行多数交换机供应商的硬件的能力。

选择联网硬件时需要考虑的一些关键因素包括：

端口数目 设计要求网络硬件有充足的端口数目。

端口密度 由于端口数量的需求，就需要更大的物理空间，以至于会影响到网络的设计。一旦首选敲定了高端口密度，在设计中就得考虑为计算和存储留下机架空间。进一步还得考虑容错设备和电力密度。高密度的交换机会非常的昂贵，亦需考虑在内，当然如若不是刚性需求，这个是没有设计网络本身重要。

端口速度 网络硬件必须支持常见的网络速度，例如：1GbE、10GbE 或者 40GbE(甚至是 100GbE)。

冗余 网络硬件的冗余级别需求是受用户对于高可用和开销考虑的影响的。网络冗余可以由增加冗余的电力供应和结对的交换机来实现，加入遇到这样的需求，硬件需要支持冗余的配置。

电力要求 确保物理数据中心为选择的网络硬件提供了必要的电力。

注意

这可能会给脊柱交换机的叶和面带来问题，同样排尾(EoR)交换机也会有问题。

网络硬件没有单一的最佳实践架构以支持一个通用型OpenStack云，让它满足所有的实现。一些在选择网络硬件时有重大影响的关键元素包括：

连通性 一个OpenStack云中所有的节点都需要网络连接。在一些情况下，节点需要访问多个网段。云的设计必须围绕充足的网络容量和带宽去确保所有的通信，无论南北流量还是东西流量都需要有充足的资源可用。

可扩展性 网络设计须围绕物理网路和逻辑网络能够轻松扩展的设计来开展。网络硬件须提供给服务器节点所需要的合适的接口类型和速度。

可用性 为确保云内部访问节点不会被中断，我们建议网络架构不要存在单点故障，应该提供一定级别的冗余和容错。网络基础设施本身就能提供一部分，比如使用网络协议诸如LACP,VRRP或其他的保证网络连接的高可用。另外，考虑网络实现的API可用亦非常重要。为了确保云中的API以及其它服务高可用，我们建议用户设计网络架构的负载均衡解决方案，以满足这些需求。

软件选择

软件选择对于通用型 OpenStack 架构设计来说需要包括以下三个方面：

- 操作系统(OS)和虚拟机管理软件
- OpenStack 组件
- 增强软件

操作系统和虚拟机管理软件

操作系统(OS)和hypervisor对于整个设计有着深刻的影响。选择一个特定的操作系统和hypervisor能够直接影响到服务器硬件的选择。确保存储硬件和拓扑支持选择好的操作系统和Hypervisor组合。也得确保网络硬件的选择和拓扑可在选定的操作系统和Hypervisor组合下正常工作。例如，如果设计中使用了链路聚合控制协议(LACP)的话，操作系统和hypervisor都需要支持它才可正常工作。

可能受到 OS 和虚拟管理程序的选择所影响的一些领域包括：

成本 选择商业支持的hypervisor，诸如微软 Hyper-V，和使用社区支持的开源hypervisor相比，在开销方面有很大的差别。开源的hypervisor有KVM, Kinstance or Xen。当比较开源操作系统解决方案时，选择了Ubuntu而不是红帽(反之亦然)，由于支持的合同不同，开销也会不一样。

受支持程度 依赖于所选定的hypervisor，相关工作人员需要受过对应的培训以及接受相关的知识，才可支持所选定的操作系统和hpervisor组合。如果没有的话，那么在设计中就得考虑培训的提供是需要另外的开销的。

管理工具 Ubuntu和Kinstance的管理工具和VMware vSphere的管理工具是不一样的。尽管OpenStack支持它们所有的操作系统和hypervisor组合。这也会对其他的设计有着非常不同的影响，结果就是选择了一种组合再作出选择。

规模和性能 确保所选择的操作系统和Hypervisor组合能满足相应的扩展和性能需求。所选择的架构需要满足依据所选择的操作系统-hypervisor组合目标实例-主机比例。

安全性 确保设计能够在维护负载需求时能够容纳正常的所安装的应用的安全补丁。为操作系统-hypervisor组合打安全补丁的频率会影响到性能，而且补丁的安装流程也会影响到维护工作。

支持的特性 决定那些特性是需要OpenStack提供的。这通常也决定了操作系统-hypervisor组合的选择。一些特性仅在特定的操作系统和hypervisor中是可用的。举例，如果确认某些特性无法实现，设计也许需要修改代码去满足用户的需求。

互操作性 用户需要考虑此操作系统和Hypervisor组合和另外的操作系统和hypervisor怎么互动，甚至包括和其它的软件。操作某一操作系统-hypervisor组合的故障排除工具，和操作其他的操作系统-hypervisor组合也许根本就不一样，那结果就是，设计时就需要交付能够使这两者工具集都能工作的工具。

OpenStack 组件

选择包含那些OpenStack组件对整个设计有着显著的影响。一些OpenStack组件，如计算和镜像服务，在每个架构中都是硬性需求，但另外一些组件，如编排，就不经常被用户所需要。

去除某些OpenStack组件会导致其他组件的功能受限。举例，如果架构中包含了编排但是去除了Telmetry，那么这个设计就无法使用编排的自动扩展功能。在决定最终架构之前，研究组件间的内部依赖是很重要的技术需求。

支撑组件

OpenStack为了构建一个平台提供云服务，是完全公平的收集软件的项目。在任何给定的OpenStack设计中都需要考虑那些附加的软件。

联网软件

OpenStack联网为实例提供各种各样的联网服务。有许多联网软件包可用于管理OpenStack组件，举几个例子：

- 提供负载均衡的软件
- 网络冗余协议
- 路由守护进程

这些软件包更加详细的描述可参考 OpenStack高可用指南 中的(Network 控制器集群 章节)。

对于通用型OpenStack云来说，基础设施组件需要高可用。如果设计时没有包括硬件的负载均衡器，那么就得包含网络软件包如HAProxy。

管理软件

选择支撑软件解决方案影响着整个OpenStack云设计过程。这些软件可提供诸如集群、日志、监控以及告警。

在集群软件中如 Corosync和Pacemaker在可用性需求中占主流。包含(不包含)这些软件包是主要决定的，要使云基础设施具有高可用的话，当然在部署之后会带来复杂的配置。OpenStack 高可用指南 提供了更加详细的安装和配置Corosync和Pacemaker，所以在设计中这些软件包需要被包含。

对日志、监测以及报警的需求是由运维考虑决定的。它们的每个子类别都包含了大量的属性。举例，在日志子类别中，某些情况考虑使用Logstash,Splunk，instanceware Log Insight等，或者其他的日志聚合-合并工具。日志需要存放在中心地带，使分析数据变得更为简单。日志数据分析引擎亦得提供自动化和问题通知，通过提供一致的预警和自动修复某些常见的已知问题。

如果这些软件包都需要的话，设计必须计算额外的资源使用率(CPU，内存，存储以及网络带宽)。其他一些潜在影响设计的有：

- 操作系统-hypervisor组合：确保选择的日志、监测、告警等工具支持打算组合的操作系统-Hypervisor。
- 网络硬件：网络硬件的选择，要看其支持日志系统、监测系统以及预警系统的情况。

数据库软件

OpenStack组件通常需要访问后端的数据库服务以存放状态和配置信息。选择合适的后端数据库以满足可用性和容错的需求，这是OpenStack服务所要求的。OpenStack服务支持的连接数据库的方式是由SQLAlchemy python所驱动，尽管如此，绝大多数部署还是使用MySQL或其变种。我们建议为通用型云提供后端服务的数据库使用高可用技术确保其高可用，方可达到架构设计的目标。

针对性能敏感的负载

虽然性能对于通用型OpenStack云是一个至为关键的因素，但不是决定性的因素，但是仍然有一些性能敏感的负载部署。关于设计性能敏感负载的想到，我们建议用户参考本书后面针对专用场景的描述。某些针对不同资源的向导可以帮助解决性能敏感负载的问题。

计算型的负载

在计算型OpenStack云中的一些设计选择是可以帮助满足他们的负载的。计算型云的负载要求有更多的CPU和内存资源，相对存储和网络的性能优先级要低的多。关于设计此类型云的指南，请参考 第 3 章 计算型 [37]。

网络型负载

在网络型OpenStack云中，一些设计选择能够改进这些类型负载的性能。网络型OpenStack的负载对于网络带宽和服务有着苛刻的需求，所以需要特别的考量和规划。关于此类型云的设计指南，请参考 第 5 章 网络型 [87]。

存储型负载

存储型OpenStack云需要被设计成可容纳对对象存储和块存储服务有着苛刻需求的负载。关于设计此类型云的指南，请参考第 4 章 存储型 [65]。

示例

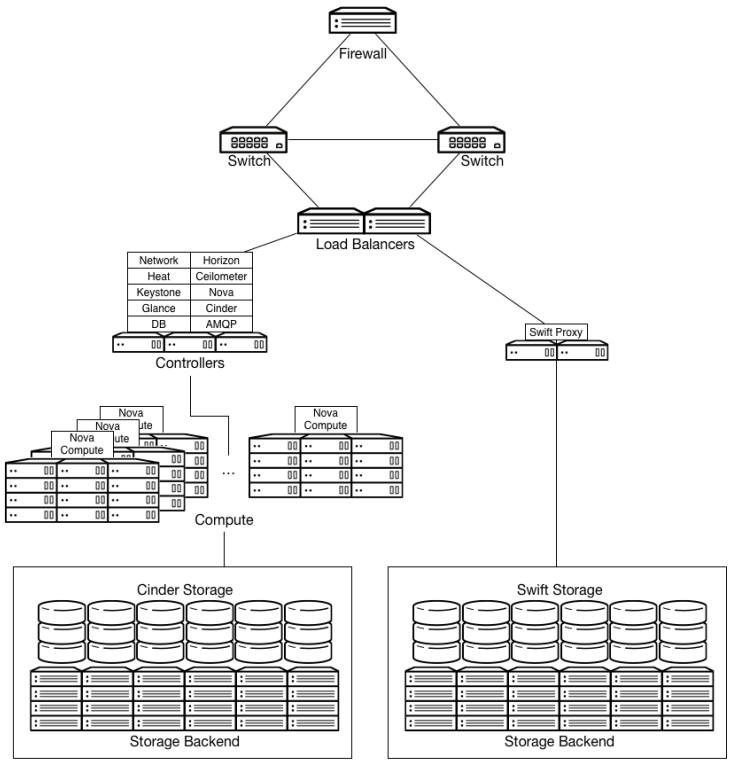
一家在线的广告公司，名称暂时保密，打算基于私有云方式运行他们的web应用，属于网站典型的架构：Tomcat + Nginx + MariaDB。 为了迎合他们的合规性需求，云基础设施运行在他们自己的数据中心。公司对负载需求有过预测，但是仍然提出了预防突发性的需求而能够灵活扩展。他们目前的环境不具有灵活的调整目标到运行开源的应用程序接口环境。目前的环境是如下面这样：

- Nginx和Tomcat的安装量在120和140之间，每个应用的实例是2 虚拟CPU和4 GB内存
- MariaDB安装在3个节点并组成Galera集群，每节点拥有4 vCPU和8GB内存

公司的网站运行着基于硬件的负载均衡服务器和多个web应用服务，且他们的编排环境是混合使用Puppet和脚本。网站每天都会产生大量的日志文件需要归档。

解决方案将由下列OpenStack组件组成：

- 防火墙、交换机、以及负载均衡设备在公网中直接面向全网的连接。
- OpenStack控制器运行着诸如镜像服务、认证服务、以及网络服务等，支撑它们的服务有诸如：MariaDB、RabbitMQ，至少有三台控制器节点配置为高可用。
- OpenStack计算节点运行着KVM的hypervisor。
- OpenStack块存储为计算实例所使用，需要持久性存储(正如数据库对于动态站点)。
- OpenStack对象存储为静态对象(例如镜像)服务。



运行140个web实例以及少量的MariaDB实例需要292颗vCPU,以及584GB内存。在典型的1U的服务器，使用双socket，16核，开启超线程的IntelCPU，计算为2:1的CPU超分配比例，可以得出需要8台这样的OpenStack计算节点。

web应用实例均运行在每个OpenStack计算节点的本地存储之上。所有的web应用实例都是无状态的，也就是意味着任何的实例宕掉，都不会影响到整体功能的继续服务。

MariaDB服务器实例将数据存储在共享存储中，使用企业级存储，有NetApp和Solidfire的设备。如果一个MariaDB的实例宕掉，另外一个实例会重新挂接原来的存储，且重新加入到Galera集群中。

将web应用的日志放在OpenStack对象存储中，用来集中处理和归档。

附加功能可实现将静态的web内容迁移到OpenStack对象存储中，且使用OpenStack对象存储作为OpenStack镜像服务的后端。

注意

增加OpenStack对象存储同时也意味着需要考虑更大的带宽。运行OpenStack对象存储，请使用10 GbE或更高的网络连接。

借助于Orchestration和Telemetry模块可以为web应用环境提供自动扩展的能力。使用Puppet脚本定义web应用的 Heat Orchestration Templates (HOT)即可实现。

OpenStack网络可以通过使用插件和网络的应用程序接口控制硬件负载均衡器。允许用户控制硬件负载均衡池以及作为池的成员的实例，但是使用它们在生产环境的话要小心权衡它的稳定程度。

第 3 章 计算型

目录

计算型是通用型openstack架构的一个特定子集。通用型架构需要应对各种各样的工作负载和应用而不特别着意于任一计算方向，而计算型则需要被设计和构建以专门应对计算密集型的工作负载。基于此，该架构必须专门设计以支撑计算密集型的工作负载。计算密集型特指ＣＰＵ密集型，ＲＡＭ密集型亦或两者亦有。但通常不是存储密集型或网络密集型。计算密集型的工作负载可能包含下面这些用例：

- 高性能计算(HPC)
- 使用Hadoop或其他分布式数据处理程序来分析大数据
- 持续集成/持续部署(CI/CD)
- 平台即服务(PaaS)
- 专门处理网络功能虚拟化(NFV)

Based on the use case requirements, such clouds might need to provide additional services such as a virtual machine disk library, file or object storage, firewalls, load balancers, IP addresses, or network connectivity in the form of overlays or virtual local area networks (VLANs). A compute-focused OpenStack cloud does not typically use raw block storage services as it does not generally host applications that require persistent block storage.

用户需求

High utilization of CPU, RAM, or both defines compute intensive workloads. User requirements determine the performance demands for the cloud.

成本 Cost is not generally a primary concern for a compute-focused cloud, however some organizations might be concerned with cost avoidance. Repurposing existing resources to tackle compute-

intensive tasks instead of acquiring additional resources may offer cost reduction opportunities.

上线时间 Compute-focused clouds can deliver products more quickly, for example by speeding up a company's software development life cycle (SDLC) for building products and applications.

赢利空间 Companies that want to build services or products that rely on the power of compute resources benefit from a compute-focused cloud. Examples include the analysis of large data sets (via Hadoop or Cassandra) or completing computational intensive tasks such as rendering, scientific computation, or simulations.

法律需求

很多辖区对于云环境中的数据的保管及管理都有相关的法律上或者监管上的要求。这些规章的常见领域包括：

- 确保持久化数据的保管和记录管理以符合数据档案化需求的数据保留政策。
- 管理数据的所有权和责任的数据所有权政策。
- 管理位于外国或者其它辖区的数据存储问题的数据独立性政策。
- Data compliance: certain types of information need to reside in certain locations due to regular issues and, more importantly, cannot reside in other locations for the same reason.

Examples of such legal frameworks include the data protection framework of the European Union and the requirements of the Financial Industry Regulatory Authority in the United States. Consult a local regulatory body for more information.

技术因素

The following are some technical requirements you must consider in the architecture design:

性能 If a primary technical concern is to deliver high performance capability, then a compute-focused design is an obvious choice because it is specifically designed to host compute-intensive workloads.

持久负载 Workloads can be either short-lived or long-running. Short-lived workloads can include continuous integration and continuous

deployment (CI-CD) jobs, which create large numbers of compute instances simultaneously to perform a set of compute-intensive tasks. The environment then copies the results or artifacts from each instance into long-term storage before destroying the instance. Long-running workloads, like a Hadoop or high-performance computing (HPC) cluster, typically ingest large data sets, perform the computational work on those data sets, then push the results into long-term storage. When the computational work finishes, the instances remain idle until they receive another job. Environments for long-running workloads are often larger and more complex, but you can offset the cost of building them by keeping them active between jobs. Another example of long-running workloads is legacy applications that are persistent over time.

存储 Workloads targeted for a compute-focused OpenStack cloud generally do not require any persistent block storage, although some uses of Hadoop with HDFS may require persistent block storage. A shared filesystem or object store maintains the initial data sets and serves as the destination for saving the computational results. By avoiding the input-output (IO) overhead, you can significantly enhance workload performance. Depending on the size of the data sets, it may be necessary to scale the object store or shared file system to match the storage demand.

用户界面 Like any other cloud architecture, a compute-focused OpenStack cloud requires an on-demand and self-service user interface. End users must be able to provision computing power, storage, networks, and software simply and flexibly. This includes scaling the infrastructure up to a substantial level without disrupting host operations.

安全性 Security is highly dependent on business requirements. For example, a computationally intense drug discovery application has much higher security requirements than a cloud for processing market data for a retailer. As a general rule, the security recommendations and guidelines provided in the OpenStack Security Guide are applicable.

运营因素

From an operational perspective, a compute intensive cloud is similar to a general-purpose cloud. See the general-purpose design section for more details on operational requirements.

技术因素

In a compute-focused OpenStack cloud, the type of instance workloads you provision heavily influences technical decision making. For example, specific use cases that demand multiple, short-running jobs present different requirements than those that specify long-running jobs, even though both situations are compute focused.

Public and private clouds require deterministic capacity planning to support elastic growth in order to meet user SLA expectations. Deterministic capacity planning is the path to predicting the effort and expense of making a given process consistently performant. This process is important because, when a service becomes a critical part of a user's infrastructure, the user's experience links directly to the SLAs of the cloud itself. In cloud computing, it is not average speed but speed consistency that determines a service's performance. There are two aspects of capacity planning to consider:

- planning the initial deployment footprint
- planning expansion of it to stay ahead of the demands of cloud users

Plan the initial footprint for an OpenStack deployment based on existing infrastructure workloads and estimates based on expected uptake.

其出发点是云计算的核心数量。通过相关的比例，用户可以收集有关信息：

- The number of expected concurrent instances: (overcommit fraction × cores) / virtual cores per instance
- Required storage: flavor disk size × number of instances

Use these ratios to determine the amount of additional infrastructure needed to support the cloud. For example, consider a situation in which you require 1600 instances, each with 2 vCPU and 50 GB of storage. Assuming the default overcommit rate of 16:1, working out the math provides an equation of:

- 1600 = (16 x (物理核数)) / 2
- 存储需要 = 50GB x 1600

表面上，公式的计算结果是需要200个物理核和80TB的存储，这些可在路径/var/lib/nova/instances/中找到。尽管如此，另外还得重点看着其他负

载的使用量，诸如API服务，数据库服务，队列服务等，它们同样需要被计入总体。

Consider, for example, the differences between a cloud that supports a managed web-hosting platform and one running integration tests for a development project that creates one instance per code commit. In the former, the heavy work of creating an instance happens only every few months, whereas the latter puts constant heavy load on the cloud controller. The average instance lifetime is significant, as a larger number generally means less load on the cloud controller.

Aside from the creation and termination of instances, consider the impact of users accessing the service, particularly on nova-api and its associated database. Listing instances gathers a great deal of information and, given the frequency with which users run this operation, a cloud with a large number of users can increase the load significantly. This can even occur unintentionally. For example, the OpenStack Dashboard instances tab refreshes the list of instances every 30 seconds, so leaving it open in a browser window can cause unexpected load.

Consideration of these factors can help determine how many cloud controller cores you require. A server with 8 CPU cores and 8 GB of RAM server would be sufficient for a rack of compute nodes, given the above caveats.

关键的硬件规格也是确保用户实例的性能的指标。请确保考虑好预算和性能需求，包括存储性能 (spindles/core), 内存可用性 (RAM/core), 网络带宽 (Gbps/core), 以及整个的CPU性能 (CPU/core).

The cloud resource calculator is a useful tool in examining the impacts of different hardware and instance load outs. See: https://github.com/noslzzp/cloud-resource-calculator/blob/master/cloud-resource-calculator.ods

扩展计划

A key challenge for planning the expansion of cloud compute services is the elastic nature of cloud infrastructure demands. Previously, new users or customers had to plan for and request the infrastructure they required ahead of time, allowing time for reactive procurement processes. Cloud computing users have come to expect the agility of having instant access to new resources as required. Consequently, plan for typical usage and for sudden bursts in usage.

Planning for expansion is a balancing act. Planning too conservatively can lead to unexpected oversubscription of the cloud and dissatisfied users. Planning for cloud expansion too aggressively can lead to unexpected underutilization of the cloud and funds spent unnecessarily on operating infrastructure.

The key is to carefully monitor the trends in cloud usage over time. The intent is to measure the consistency with which you deliver services, not the average speed or capacity of the cloud. Using this information to model capacity performance enables users to more accurately determine the current and future capacity of the cloud.

CPU 和内存

Adapted from: http://docs.openstack.org/openstack-ops/content/compute_nodes.html#cpu_choice

In current generations, CPUs have up to 12 cores. If an Intel CPU supports Hyper-Threading, those 12 cores double to 24 cores. A server that supports multiple CPUs multiplies the number of available cores. Hyper-Threading is Intel's proprietary simultaneous multi-threading implementation, used to improve parallelization on their CPUs. Consider enabling Hyper-Threading to improve the performance of multithreaded applications.

Whether the user should enable Hyper-Threading on a CPU depends on the use case. For example, disabling Hyper-Threading can be beneficial in intense computing environments. Running performance tests using local workloads with and without Hyper-Threading can help determine which option is more appropriate in any particular case.

If they must run the Libvirt or KVM hypervisor drivers, then the compute node CPUs must support virtualization by way of the VT-x extensions for Intel chips and AMD-v extensions for AMD chips to provide full performance.

OpenStack enables users to overcommit CPU and RAM on compute nodes. This allows an increase in the number of instances running on the cloud at the cost of reducing the performance of the instances. OpenStack Compute uses the following ratios by default:

- CPU 超分配比例: 16:1
- RAM 超分配比例: 1.5:1

默认的CPU超分配比例是16:1,这意味着调度器可以为每个物理核分配16个虚拟核。举例来说，如果物理节点有12个核，调度器就拥有192个虚拟核。在典型的flavor定义中，每实例4个虚拟核，那么此超分配比例可以在此物理节点上提供48个实例。

同样的，默认的内存超分配比例是1.5:1，这意味着调度器为实例分配的内存总量要少于物理节点内存的1.5倍。

举例来说，如果一台物理节点有48GB的内存，调度器为实例分配的内存总量要少于72GB(如果每个实例分配了8GB内存的话，此节点可建立9个实例)。

You must select the appropriate CPU and RAM allocation ratio based on particular use cases.

额外的硬件

在计算节点中使用一些额外的设备对于某些用例有着显著的益处。举几个常见的例子：

- 使用图形处理单元(GPU)，可大大有益于高性能计算任务。
- Cryptographic routines 受益于硬件随机数生成器，以避免entropy starvation。
- Database management systems that benefit from the availability of SSDs for ephemeral storage to maximize read/write time.

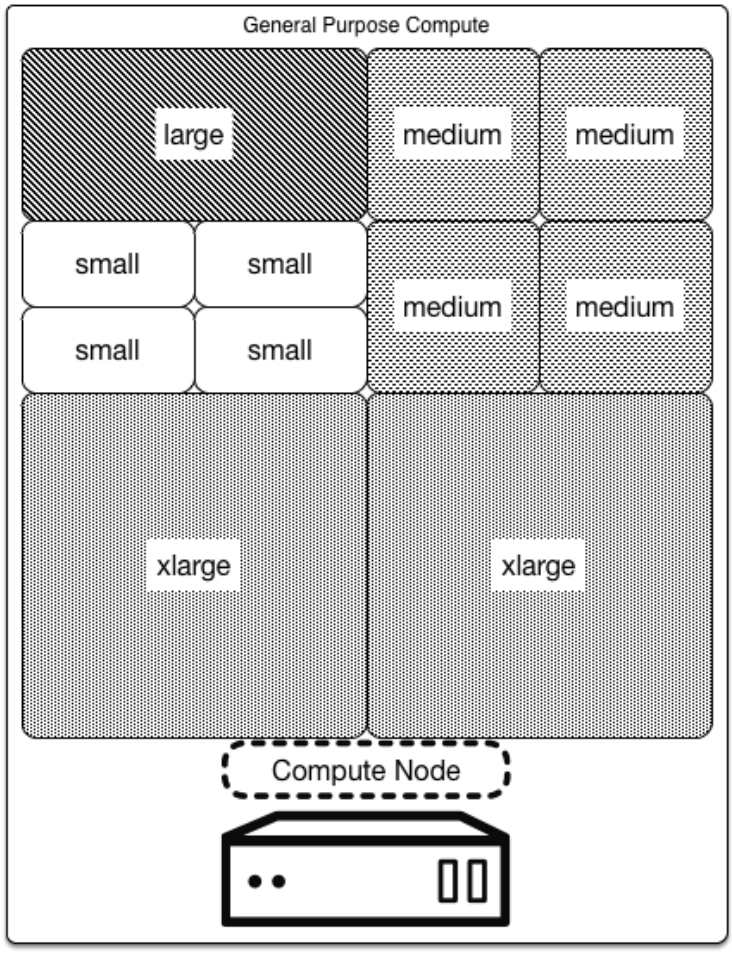
Host aggregates group hosts that share similar characteristics, which can include hardware similarities. The addition of specialized hardware to a cloud deployment is likely to add to the cost of each node, so consider carefully consideration whether all compute nodes, or just a subset targeted by flavors, need the additional customization to support the desired workloads.

量力而行

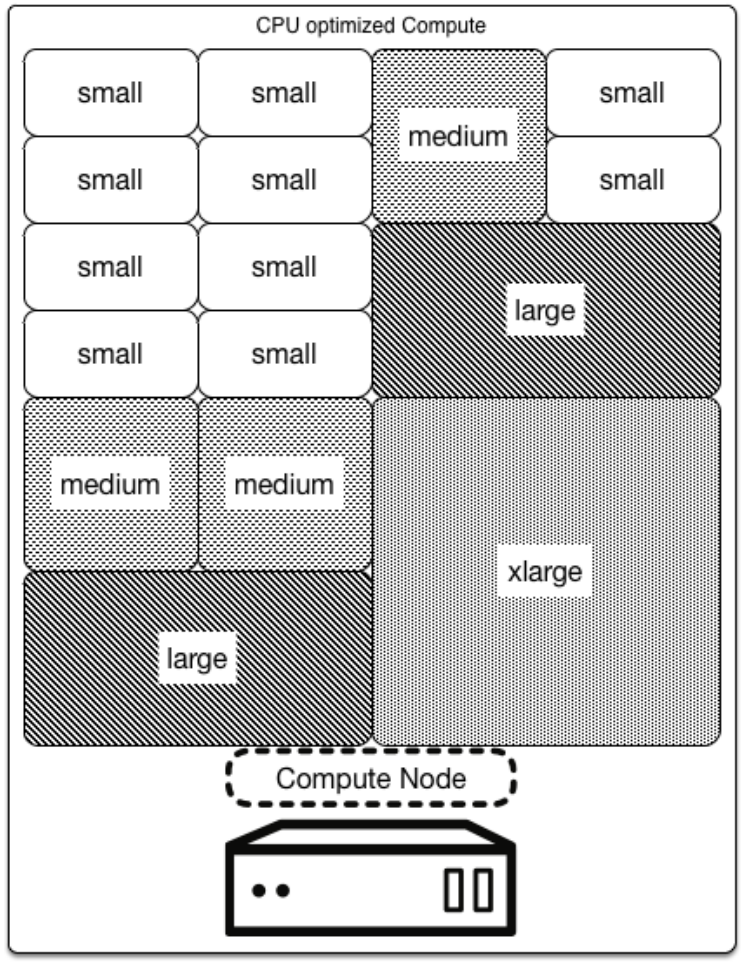
基础设施即服务所提供的，包括OpenStack，使用类型来提供标准的虚拟机资源需求视图，为分配实例时能够充分利用可用的物理资源提供了简单的解决办法。

In order to facilitate packing of virtual machines onto physical hosts, the default selection of flavors provides a second largest flavor that

is half the size of the largest flavor in every dimension. It has half the vCPUs, half the vRAM, and half the ephemeral disk space. The next largest flavor is half that size again. The following figure provides a visual representation of this concept for a general purpose computing design:



The following figure displays a CPU-optimized, packed server:



These default flavors are well suited to typical configurations of commodity server hardware. To maximize utilization, however, it may be necessary to customize the flavors or create new ones in order to better align instance sizes to the available hardware.

负载的特点常会影响到硬件的选择和实例类型的配置，尤其是他们有不同的CPU、内存、硬盘的比例需求。

对于Flavor的更多信息请参考： http://docs.openstack.org/openstack-ops/content/flavors.html

性能

So that workloads can consume as many resources as are available, do not share cloud infrastructure. Ensure you accommodate large scale workloads.

The duration of batch processing differs depending on individual workloads. Time limits range from seconds to hours, and as a result it is difficult to predict resource use.

安全性

The security considerations for this scenario are similar to those of the other scenarios in this guide.

A security domain comprises users, applications, servers, and networks that share common trust requirements and expectations within a system. Typically they have the same authentication and authorization requirements and users.

这些安全域有：

1. 公有

2. 客户机

3. 管理

4. 数据

You can map these security domains individually to the installation, or combine them. For example, some deployment topologies combine both guest and data domains onto one physical network, whereas in other cases these networks are physically separate. In each case, the cloud operator should be aware of the appropriate security concerns. Map out security domains against specific OpenStack deployment topology. The domains and their trust requirements depend on whether the cloud instance is public, private, or hybrid.

The public security domain is an untrusted area of the cloud infrastructure. It can refer to the Internet as a whole or simply to networks over which the user has no authority. Always consider this domain untrusted.

Typically used for compute instance-to-instance traffic, the guest security domain handles compute data generated by instances on the cloud. It does not handle services that support the operation of the cloud, for example API calls. Public cloud providers and private cloud providers who do not have stringent controls on instance use or who allow unrestricted Internet access to instances should consider this an untrusted domain. Private cloud providers may want to consider this an internal network and therefore trusted only if they have controls in place to assert that they trust instances and all their tenants.

The management security domain is where services interact. Sometimes referred to as the "control plane", the networks in this domain transport confidential data such as configuration parameters, user names, and passwords. In most deployments this is a trusted domain.

The data security domain deals with information pertaining to the storage services within OpenStack. Much of the data that crosses this network has high integrity and confidentiality requirements and depending on the type of deployment there may also be strong availability requirements. The trust level of this network is heavily dependent on deployment decisions and as such we do not assign this a default level of trust.

When deploying OpenStack in an enterprise as a private cloud, you can generally assume it is behind a firewall and within the trusted network alongside existing systems. Users of the cloud are typically employees or trusted individuals that are bound by the security requirements set forth by the company. This tends to push most of the security domains towards a more trusted model. However, when deploying OpenStack in a public-facing role, you cannot make these assumptions and the number of attack vectors significantly increases. For example, the API endpoints and the software behind it become vulnerable to hostile entities attempting to gain unauthorized access or prevent access to services. This can result in loss of reputation and you must protect against it through auditing and appropriate filtering.

Take care when managing the users of the system, whether in public or private clouds. The identity service enables LDAP to be part of the authentication process, and includes such systems as an OpenStack deployment that may ease user management if integrated into existing systems.

We recommend placing API services behind hardware that performs SSL termination. API services transmit user names, passwords, and generated tokens between client machines and API endpoints and therefore must be secure.

For more information on OpenStack Security, see http://docs.openstack.org/security-guide/

OpenStack 组件

Due to the nature of the workloads in this scenario, a number of components are highly beneficial for a Compute-focused cloud. This includes the typical OpenStack components:

- OpenStack 计算(nova)
- OpenStack Image service (glance)
- OpenStack 认证(keystone)

也会考虑一些特别的组件：

- Orchestration 模块 (heat)

It is safe to assume that, given the nature of the applications involved in this scenario, these are heavily automated deployments. Making use of Orchestration is highly beneficial in this case. You can script the deployment of a batch of instances and the running of tests, but it makes sense to use the Orchestration module to handle all these actions.

- Telemetry 模块 (ceilometer)

Telemetry and the alarms it generates support autoscaling of instances using Orchestration. Users that are not using the Orchestration module do not need to deploy the Telemetry module and may choose to use external solutions to fulfill their metering and monitoring requirements.

See also: http://docs.openstack.org/openstack-ops/content/logging_monitoring.html

- OpenStack 块存储(cinder)

Due to the burst-able nature of the workloads and the applications and instances that perform batch processing, this cloud mainly uses memory or CPU, so the need for add-on storage to each instance is not a likely requirement. This does not mean that you do not use OpenStack Block Storage (cinder) in the infrastructure, but typically it is not a central component.

- 网络

When choosing a networking platform, ensure that it either works with all desired hypervisor and container technologies and their OpenStack drivers, or that it includes an implementation of an ML2 mechanism driver. You can mix networking platforms that provide ML2 mechanisms drivers.

运营因素

Operationally, there are a number of considerations that affect the design of compute-focused OpenStack clouds. Some examples include:

- Enforcing strict API availability requirements
- Understanding and dealing with failure scenarios
- Managing host maintenance schedules

Service-level agreements (SLAs) are contractual obligations that ensure the availability of a service. When designing an OpenStack cloud, factoring in promises of availability implies a certain level of redundancy and resiliency.

- Guarantees for API availability imply multiple infrastructure services combined with appropriate, highly available load balancers.
- Network uptime guarantees affect the switch design and might require redundant switching and power.
- Factoring of network security policy requirements in to deployments.

支持和维护

OpenStack cloud management requires a certain level of understanding and comprehension of design architecture. Specially trained, dedicated operations organizations are more likely to manage larger cloud service providers or telecom providers. Smaller implementations are more inclined to rely on smaller support teams that need to combine the engineering, design, and operation roles.

The maintenance of OpenStack installations requires a variety of technical skills. To ease the operational burden, consider incorporating features into the architecture and design. Some examples include:

- Automating the operations functions
- Utilizing a third party management company

监控

OpenStack clouds require appropriate monitoring platforms that help to catch and manage errors adequately. Consider leveraging any existing monitoring systems to see if they are able to effectively monitor an OpenStack environment. Specific meters that are critically important to capture include:

- 镜像磁盘使用
- 计算API的响应时间

计划内和计划外服务器宕机时间

Unexpected server downtime is inevitable, and SLAs can address how long it takes to recover from failure. Recovery of a failed host means restoring instances from a snapshot, or respawning that instance on another available host.

It is acceptable to design a compute-focused cloud without the ability to migrate instances from one host to another. The expectation is that the application developer must handle failure within the application itself. However, provisioning a compute-focused cloud provides extra resilience. In this scenario, the developer deploys extra support services.

容量计划

Adding extra capacity to an OpenStack cloud is a horizontally scaling process.

注意

Be mindful, however, of additional work to place the nodes into appropriate Availability Zones and Host Aggregates.

We recommend the same or very similar CPUs when adding extra nodes to the environment because they reduce the chance of breaking live-migration features if they are present. Scaling out hypervisor hosts also has a direct effect on network and other data center resources. We recommend you factor in this increase when reaching rack capacity or when requiring extra network switches.

Changing the internal components of a Compute host to account for increases in demand is a process known as vertical scaling. Swapping a CPU

for one with more cores, or increasing the memory in a server, can help add extra capacity for running applications.

Another option is to assess the average workloads and increase the number of instances that can run within the compute environment by adjusting the overcommit ratio. While only appropriate in some environments, it's important to remember that changing the CPU overcommit ratio can have a detrimental effect and cause a potential increase in a noisy neighbor. The added risk of increasing the overcommit ratio is that more instances fail when a compute host fails. We do not recommend that you increase the CPU overcommit ratio in compute-focused OpenStack design architecture, as it can increase the potential for noisy neighbor issues.

架构

硬件选择涵盖三个方面：

- 计算
- 网络
- 存储

An OpenStack cloud with extreme demands on processor and memory resources is compute-focused, and requires hardware that can handle these demands. This can mean choosing hardware which might not perform as well on storage or network capabilities. In a compute-focused architecture, storage and networking load a data set into the computational cluster, but are not otherwise in heavy demand.

Consider the following factors when selecting compute (server) hardware:

服务器密度 一个衡量就是在给定的物理空间内可以放多少台服务器，例如机架式单元(U)。

资源容量 CPU核数，多少内存，或者多少存储可以交付。

延伸性 The number of additional resources you can add to a server before it reaches its limit.

成本 相对硬件的购买价格，与构建系统所需要的设计功力的级别成反比。

Weigh these considerations against each other to determine the best design for the desired purpose. For example, increasing server density means sacrificing resource capacity or expandability. Increasing resource capacity and expandability can increase cost but decreases server density. Decreasing cost can mean decreasing supportability, server density, resource capacity, and expandability.

A compute-focused cloud should have an emphasis on server hardware that can offer more CPU sockets, more CPU cores, and more RAM. Network connectivity and storage capacity are less critical. The hardware must provide enough network connectivity and storage capacity to meet minimum user requirements, but they are not the primary consideration.

Some server hardware form factors suit a compute-focused architecture better than others. CPU and RAM capacity have the highest priority. Some considerations for selecting hardware:

- Most blade servers can support dual-socket multi-core CPUs. To avoid this CPU limit, select "full width" or "full height" blades. Be aware, however, that this also decreases server density. For example, high density blade servers such as HP BladeSystem or Dell PowerEdge M1000e support up to 16 servers in only ten rack units. Using half-height blades is twice as dense as using full-height blades, which results in only eight servers per ten rack units.

- 1U rack-mounted servers that occupy only a single rack unit may offer greater server density than a blade server solution. It is possible to place forty 1U servers in a rack, providing space for the top of rack (ToR) switches, compared to 32 full width blade servers. However, as of the Icehouse release, 1U servers from the major vendors have only dual-socket, multi-core CPU configurations. To obtain greater than dual-socket support in a 1U rack-mount form factor, purchase systems from original design (ODMs) or second-tier manufacturers.

- 2U rack-mounted servers provide quad-socket, multi-core CPU support, but with a corresponding decrease in server density (half the density that 1U rack-mounted servers offer).

- 大型机架式服务器，比如4U服务器，可提供更为强大的CPU容量。通常支持4个甚至8个CPU插槽。拥有很强的扩展性，但是这些服务器会带来低密度，以及更加昂贵的开销。

- "Sled servers" are rack-mounted servers that support multiple independent servers in a single 2U or 3U enclosure. These deliver higher density as compared to typical 1U or 2U rack-mounted servers. For

example, many sled servers offer four independent dual-socket nodes in 2U for a total of eight CPU sockets in 2U. However, the dual-socket limitation on individual nodes may not be sufficient to offset their additional cost and configuration complexity.

下列因素会严重影响到计算型OpenStack架构设计的服务器硬件选择：

实例密度 In a compute-focused architecture, instance density is lower, which means CPU and RAM over-subscription ratios are also lower. You require more hosts to support the anticipated scale due to instance density being lower, especially if the design uses dual-socket hardware designs.

主机密度 Another option to address the higher host count of dual socket designs is to use a quad socket platform. Taking this approach decreases host density, which increases rack count. This configuration may affect the network requirements, the number of power connections, and possibly impact the cooling requirements.

电源和制冷密度 The power and cooling density requirements for 2U, 3U or even 4U server designs might be lower than for blade, sled, or 1U server designs because of lower host density. For data centers with older infrastructure, this may be a desirable feature.

When designing a compute-focused OpenStack architecture, you must consider whether you intend to scale up or scale out. Selecting a smaller number of larger hosts, or a larger number of smaller hosts, depends on a combination of factors: cost, power, cooling, physical rack and floor space, support-warranty, and manageability.

存储硬件选择

For a compute-focused OpenStack architecture, the selection of storage hardware is not critical as it is not a primary consideration. Nonetheless, there are several factors to consider:

成本 The overall cost of the solution plays a major role in what storage architecture and storage hardware you select.

性能 The performance of the storage solution is important; you can measure it by observing the latency of storage I-O requests. In a compute-focused OpenStack cloud, storage latency can be a major

consideration. In some compute-intensive workloads, minimizing the delays that the CPU experiences while fetching data from storage can impact significantly on the overall performance of the application.

可扩展性 Scalability refers to the performance of a storage solution as it expands to its maximum size. A solution that performs well in small configurations but has degrading performance as it expands is not scalable. On the other hand, a solution that continues to perform well at maximum expansion is scalable.

延伸性 Expandability refers to the overall ability of a storage solution to grow. A solution that expands to 50 PB is more expandable than a solution that only scales to 10PB. Note that this meter is related to, but different from, scalability, which is a measure of the solution's performance as it expands.

For a compute-focused OpenStack cloud, latency of storage is a major consideration. Using solid-state disks (SSDs) to minimize latency for instance storage reduces CPU delays related to storage and improves performance. Consider using RAID controller cards in compute hosts to improve the performance of the underlying disk subsystem.

Evaluate solutions against the key factors above when considering your storage architecture. This determines if a scale-out solution such as Ceph or GlusterFS is suitable, or if a single, highly expandable, scalable, centralized storage array is better. If a centralized storage array suits the requirements, the array vendor determines the hardware. You can build a storage array using commodity hardware with Open Source software, but you require people with expertise to build such a system. Conversely, a scale-out storage solution that uses direct-attached storage (DAS) in the servers may be an appropriate choice. If so, then the server hardware must support the storage solution.

下面列出在一个计算型OpenStack云中可能影响到特定的存储架构及其对应的存储硬件的因素：

连通性 Ensure connectivity matches the storage solution requirements. If you select a centralized storage array, determine how the hypervisors should connect to the storage array. Connectivity can affect latency and thus performance, so ensure that the network characteristics minimize latency to boost the overall performance of the design.

延迟 决定的是如果用例中拥有并行或高度变化的延迟。

吞吐量 To improve overall performance, ensure that you optimize the storage solution. While a compute-focused cloud does not usually have major data I-O to and from storage, this is an important factor to consider.

服务器硬件 If the solution uses DAS, this impacts the server hardware choice, host density, instance density, power density, OS-hypervisor, and management tools.

When instances must be highly available or capable of migration between hosts, use a shared storage file-system to store instance ephemeral data to ensure that compute services can run uninterrupted in the event of a node failure.

选择网络硬件

Some of the key considerations for networking hardware selection include:

端口数目 The design requires networking hardware that has the requisite port count.

端口密度 The required port count affects the physical space that a network design requires. A switch that can provide 48 10 GbE ports in 1U has a much higher port density than a switch that provides 24 10 GbE ports in 2U. A higher port density is better, as it leaves more rack space for compute or storage components. You must also consider fault domains and power density. Although more expensive, you can also consider higher density switches as it is important not to design the network beyond requirements.

端口速度 The networking hardware must support the proposed network speed, for example: 1 GbE, 10 GbE, 40 GbE, or 100 GbE.

冗余 User requirements for high availability and cost considerations influence the level of network hardware redundancy you require. You can achieve network redundancy by adding redundant power supplies or paired switches. If this is a requirement, the hardware must support this configuration. User requirements determine if you require a completely redundant network infrastructure.

电力要求 确保物理数据中心为所选择的网络硬件提供了必要的电力。对于机柜顶端型(ToR)交换机没有什么影响，但是spine交换机的叶和fabric以及排尾(EoR)交换机就可能发生问题，假如电力不足的话。

We recommend designing the network architecture using a scalable network model that makes it easy to add capacity and bandwidth. A good example

of such a model is the leaf-spline model. In this type of network design, it is possible to easily add additional bandwidth as well as scale out to additional racks of gear. It is important to select network hardware that supports the required port count, port speed, and port density while also allowing for future growth as workload demands increase. It is also important to evaluate where in the network architecture it is valuable to provide redundancy. Increased network availability and redundancy comes at a cost, therefore we recommend weighing the cost versus the benefit gained from utilizing and deploying redundant network switches and using bonded interfaces at the host level.

软件选择

Consider your selection of software for a compute-focused OpenStack:

- 操作系统(OS)和虚拟机管理软件
- OpenStack 组件
- 增强软件

决定了上述其中的任何一项，都要影响到其余两个的架构设计。

操作系统和虚拟机管理软件

The selection of operating system (OS) and hypervisor has a significant impact on the end point design. Selecting a particular operating system and hypervisor could affect server hardware selection. The node, networking, and storage hardware must support the selected combination. For example, if the design uses Link Aggregation Control Protocol (LACP), the hypervisor must support it.

OS and hypervisor selection impact the following areas:

成本 Selecting a commercially supported hypervisor such as Microsoft Hyper-V results in a different cost model from choosing a community-supported, open source hypervisor like Kinstance or Xen. Even within the ranks of open source solutions, choosing one solution over another can impact cost due to support contracts. On the other hand, business or application requirements might dictate a specific or commercially supported hypervisor.

受支持程度 Staff require appropriate training and knowledge to support the selected OS and hypervisor combination. Consideration of training costs may impact the design.

管理工具 The management tools used for Ubuntu and Kinstance differ from the management tools for VMware vSphere. Although OpenStack supports both OS and hypervisor combinations, the choice of tool impacts the rest of the design.

规模和性能 Ensure that selected OS and hypervisor combinations meet the appropriate scale and performance requirements. The chosen architecture must meet the targeted instance-host ratios with the selected OS-hypervisor combination.

安全性 Ensure that the design can accommodate the regular installation of application security patches while maintaining the required workloads. The frequency of security patches for the proposed OS-hypervisor combination has an impact on performance and the patch installation process can affect maintenance windows.

支持的特性 Determine what features of OpenStack you require. The choice of features often determines the selection of the OS-hypervisor combination. Certain features are only available with specific OSs or hypervisors. For example, if certain features are not available, modify the design to meet user requirements.

互操作性 Consider the ability of the selected OS-hypervisor combination to interoperate or co-exist with other OS-hypervisors, or with other software solutions in the overall design. Operational and troubleshooting tools for one OS-hypervisor combination may differ from the tools for another OS-hypervisor combination. The design must address if the two sets of tools need to interoperate.

OpenStack 组件

The selection of OpenStack components has a significant impact. There are certain components that are omnipresent, for example the compute and image services, but others, such as the orchestration module may not be present. Omitting heat does not typically have a significant impact on the overall design. However, if the architecture uses a replacement for OpenStack Object Storage for its storage component, this could have significant impacts on the rest of the design.

For a compute-focused OpenStack design architecture, the following components may be present:

- 认证 (keystone)

- 仪表盘 (horizon)
- 计算 (nova)
- 对象存储 (swift, ceph or a commercial solution)
- 镜像 (glance)
- 网络 (neutron)
- 编排 (heat)

A compute-focused design is less likely to include OpenStack Block Storage due to persistent block storage not being a significant requirement for the expected workloads. However, there may be some situations where the need for performance employs a block storage component to improve data I-O.

The exclusion of certain OpenStack components might also limit the functionality of other components. If a design opts to include the Orchestration module but excludes the Telemetry module, then the design cannot take advantage of Orchestration's auto scaling functionality as this relies on information from Telemetry.

增强软件

While OpenStack is a fairly complete collection of software projects for building a platform for cloud services, there are invariably additional pieces of software that you might add to an OpenStack design.

联网软件

OpenStack Networking provides a wide variety of networking services for instances. There are many additional networking software packages that might be useful to manage the OpenStack components themselves. Some examples include software to provide load balancing, network redundancy protocols, and routing daemons. The OpenStack High Availability Guide (http://docs.openstack.org/high-availability-guide/content) describes some of these software packages in more detail.

For a compute-focused OpenStack cloud, the OpenStack infrastructure components must be highly available. If the design does not include hardware load balancing, you must add networking software packages like HAProxy.

管理软件

所选择的支撑软件解决方案会影响到整个OpenStack云的设计，它们包括能够提供集群、日志、监测以及预警的软件。

The availability of design requirements is the main determination for the inclusion of clustering Software, such as Corosync or Pacemaker. Therefore, the availability of the cloud infrastructure and the complexity of supporting the configuration after deployment impacts the inclusion of these software packages. The OpenStack High Availability Guide provides more details on the installation and configuration of Corosync and Pacemaker.

Operational considerations determine the requirements for logging, monitoring, and alerting. Each of these sub-categories includes various options. For example, in the logging sub-category consider Logstash, Splunk, Log Insight, or some other log aggregation-consolidation tool. Store logs in a centralized location to ease analysis of the data. Log data analytics engines can also provide automation and issue notification by alerting and attempting to remediate some of the more commonly known issues.

If you require any of these software packages, then the design must account for the additional resource consumption. Some other potential design impacts include:

- OS-hypervisor combination: ensure that the selected logging, monitoring, or alerting tools support the proposed OS-hypervisor combination.
- Network hardware: the logging, monitoring, and alerting software must support the network hardware selection.

数据库软件

A large majority of OpenStack components require access to back-end database services to store state and configuration information. Select an appropriate back-end database that satisfies the availability and fault tolerance requirements of the OpenStack services. OpenStack services support connecting to any database that the SQLAlchemy Python drivers support, however most common database deployments make use of MySQL or some variation of it. We recommend that you make the database that provides back-end services within a general-purpose cloud highly available. Some of the more common software solutions include Galera, MariaDB, and MySQL with multi-master replication.

示例

The Conseil Europ é en pour la Recherche Nucl é aire (CERN), also known as the European Organization for Nuclear Research, provides particle accelerators and other infrastructure for high-energy physics research.

CERN在2011年准备在欧洲建立第三个数据中心。

数据中心	大体的容量
瑞士日内瓦	• 3.5 万瓦特 • 91000 个核 • 120 PB 硬盘 • 100 PB 磁带 • 310 TB 内存
匈牙利布达佩斯	• 2.5 晚瓦特 • 20000 个核 • 6 PB 硬盘

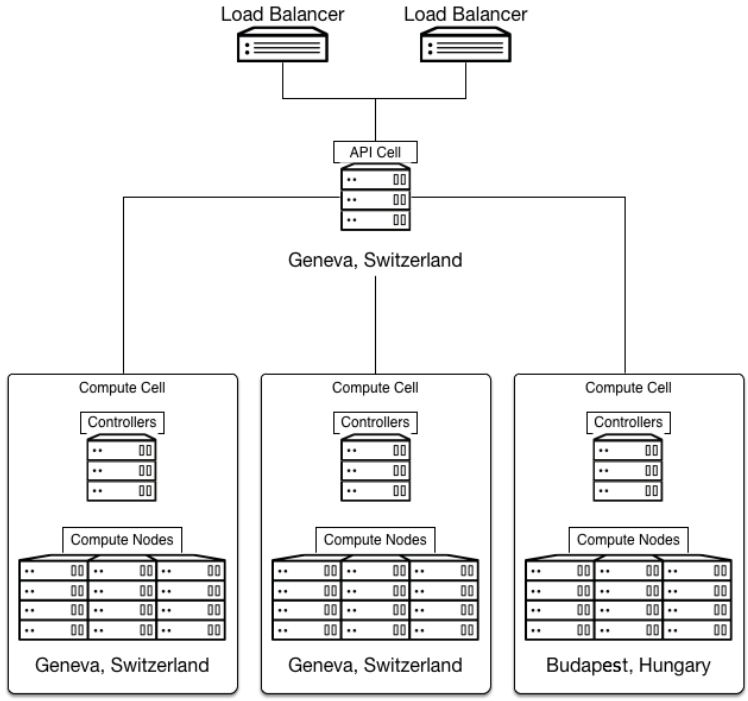
To support a growing number of compute-heavy users of experiments related to the Large Hadron Collider (LHC), CERN ultimately elected to deploy an OpenStack cloud using Scientific Linux and RDO. This effort aimed to simplify the management of the center's compute resources with a view to doubling compute capacity through the addition of a data center in 2013 while maintaining the same levels of compute staff.

The CERN solution uses cells for segregation of compute resources and for transparently scaling between different data centers. This decision meant trading off support for security groups and live migration. In addition, they must manually replicate some details, like flavors, across cells. In spite of these drawbacks cells provide the required scale while exposing a single public API endpoint to users.

CERN created a compute cell for each of the two original data centers and created a third when it added a new data center in 2013. Each cell contains three availability zones to further segregate compute resources and at least three RabbitMQ message brokers configured for clustering with mirrored queues for high availability.

The API cell, which resides behind a HAProxy load balancer, is in the data center in Switzerland and directs API calls to compute cells using a customized variation of the cell scheduler. The customizations allow

certain workloads to route to a specific data center or all data centers, with cell RAM availability determining cell selection in the latter case.



在cell里的可以定制的过滤器：

- ImagePropertiestFilter - 为提供特殊处理，取决于客户机操作系统的使用(基于Linux或基于Windows)
- ProjectsToAggregateFilter - 为了提供特殊处理，取决于该实例相关联的项目。
- default_schedule_zones -允许选择多个默认可用区域，而不是单一的。

A central database team manages the MySQL database server in each cell in an active/passive configuration with a NetApp storage back end. Backups run every 6 hours.

网络架构

To integrate with existing networking infrastructure, CERN made customizations to legacy networking (nova-network). This was in the form of a driver to integrate with CERN's existing database for tracking MAC and IP address assignments.

The driver facilitates selection of a MAC address and IP for new instances based on the compute node where the scheduler places the instance.

The driver considers the compute node where the scheduler placed an instance and selects a MAC address and IP from the pre-registered list associated with that node in the database. The database updates to reflect the address assignment to that instance.

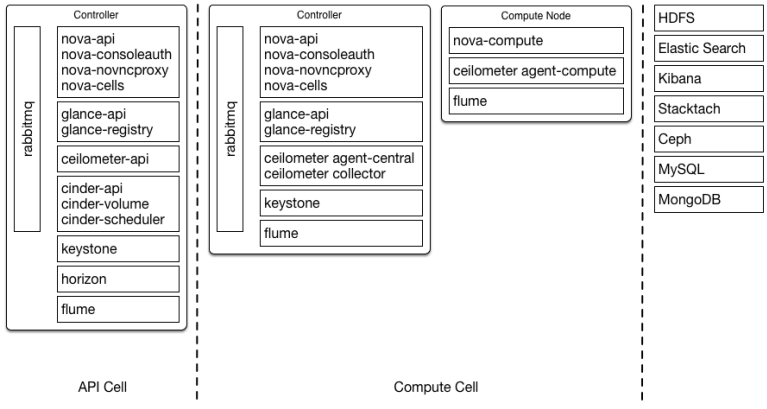
存储架构

CERN deploys the OpenStack Image service in the API cell and configures it to expose version 1 (V1) of the API. This also requires the image registry. The storage back end in use is a 3 PB Ceph cluster.

CERN maintains a small set of Scientific Linux 5 and 6 images onto which orchestration tools can place applications. Puppet manages instance configuration and customization.

监控

CERN does not require direct billing, but uses the Telemetry module to perform metering for the purposes of adjusting project quotas. CERN uses a sharded, replicated, MongoDB back-end. To spread API load, CERN deploys instances of the nova-api service within the child cells for Telemetry to query against. This also requires the configuration of supporting services such as keystone, glance-api, and glance-registry in the child cells.



另外使用的监测工具有Flume, Elastic Search, Kibana, 以及CERN开发的项目Lemon。

参考

架构设计指南的作者们非常感谢CERN,他们公开了他们在他们的环境中部署OpenStack的文档，是这一节的内容的组织的基础。

- http://openstack-in-production.blogspot.fr
- 深入学习CERN云基础设施

第 4 章 存储型

目录

云存储是一种数据存储的模型。这种模型下电子数据存放在逻辑的存储池之中，物理的存储设备则分布在多个服务器或者地点之中。云存储一般来说指的是所支持的对象存储服务，然而，随着发展，这个概念包括了其它能够作为服务提供的数据存储，比如块存储。

云存储是运行在虚拟化基础设施之上的，并且在可访问接口、弹性、可扩展性、多租户以及可测量资源方面都类似于更广泛意义上的云计算。云存储服务可以是场外的服务，也可以在内部进行部署。

云存储由很多分散的但是同质化的资源所组成，并且通常被称为集成存储云。云存储具有非常高的容错能力，这是通过冗余以及数据的分布存储实现的。通过创建版本化的副本，云存储是非常耐用的，而且对于数据的副本来说，其一致性也是非常高的。

在规模到达一定的程度时，数据操作的管理对于整个组织来说就会是一个资源密集型的过程。分级存储管理(HSM)系统以及数据网格能够帮助对数据评估的基准值作出注解以及报告，从而做出正确的决定以及自动化该数据决策。HSM 支持自动化的排列和移动，以及数据操作的协调编排。数据网格是一个架构，或者是一项不断发展的服务集合技术，此项技术能够将多个服务协调到一起，让用户能够管理大规模的数据集合。

以下是云存储类型的应用部署的例子：

- 活跃归档、备份和分级存储管理
- 通用内容存储和同步。比如私有的 dropbox。
- 基于并行文件系统的数据分析。
- 某些服务的非结构化数据存储。比如社交媒体的后端存储。

- 持久化的块存储。
- 操作系统和应用镜像存储。
- 媒体流。
- 数据库。
- 内容分发。
- 云存储配对。

用户需求

据数据的需求来定义存储型云，他们包括：

- 性能
- 访问模式
- 数据结构

成本以及用户需求之间的平衡，决定了在云架构中需要使用的方法和技术。

成本　用户只为他实际使用的存储买单。这个值通常由用户在一个月之中的平均使用量表示。这并不意味着云存储更便宜，云存储只让运营费用变得便宜，而非资产支出。从商业角度来说，方案能够适当的扩展，避免了提前购买一个不能被完全利用的大容量存储，是有益的。

法律需求　很多辖区对于云环境中的数据的存储及管理都有相关的法律上或者监管上的要求。这些规章的常见领域包括数据保管政策和数据所有权政策。

法律需求

很多辖区对于云环境中的数据的保管及管理都有相关的法律上或者监管上的要求。这些规章的常见领域包括：

注意

Examples of such legal frameworks include the data protection framework of the European Union and the requirements of the

Financial Industry Regulatory Authority in the United States. Consult a local regulatory body for more information.

数据保管 确保持久化数据的保管和记录管理以符合数据档案化需求的政策。

数据所有权 管理数据的所有权和责任的政策。

数据独立性 管理位于外国或者其它辖区的数据存储问题的政策。

数据合规性 管理由于法律原因因而特定类型的信息必须存放在特定的地点或者由于相同的原因数据不能够存放在其它地点的政策。

技术需求

以下是一些需要被合并到架构设计中的技术性需求：

数据亲近度 设计必须考虑每个宿主机上的存储，或者由集中存储设备所提供的存储，以提供高性能或者大容量的存储空间。

性能 可能需要使用不同的技术来缓存磁盘操作以提供性能。

可用性 关于可用性的特殊需求将影响用以存储及保护数据的技术。这些需求也将会影响成本以及要实施的方案。

安全性 数据无论是在传输时或其他方式都将会被保护。

技术因素

对存储型的 OpenStack 设计架构来说比较关键的一些技术上的考虑因素包括：

输入-输出需求 在对最终的存储框架做出决定之前，需要对输入-输出的性能需求进行研究并将其模型化。为输入-输出性能运行一些检测能够提供一个大致的预期中性能水平的基准。假如测试能够包含详细信息，比如说，在对象存储系统中的对象大小，或者对象存储以及块存储的不同 容量级别，那么测试得出来的数据就能够帮助对不同的负载之下的行为和结果进行模型化。在架构的生命周期中运行小规模脚本化的测试能够帮助在不同时间点及时记录系统的健康状态。这些脚本化测试得出的数据，能够在以后对整个组织的需要进行研究和深入了解的时候提供帮助。

规模 在一个存储型的 OpenStack 架构设计当中，存储解决方案的规模，是由初始的需求，包括 IOPS、容量以及带宽等，以及未来的需要所决定的。对于一个设计来说，在整个预算周期之中，基于项目需要而计划容量，是很重要的。理想情况下，所选择的架构必须在成本以及容量之间做出平衡，同时又必须提供足够的灵活性，以便在新技术和方法可用时实现它们。

安全性 关于数据的安全性设计，基于 SLA、法律要求、行业规章以及系统或者用户所需要的认证等方面的不同，有着不同的关注点。根据数据的类型，需要考虑遵从 HIPPA、ISO9000 以及 SOX 等。对有些组织来说，访问控制的等级也是很重要的。

OpenStack 兼容性 与 OpenStack 系统的互动性和集成情况，在决定存储硬件和存储管理平台的选择上，可能是最为重要的。这里所说的互动性和集成度，包括比如与 OpenStack 块存储的互操作性、与 OpenStack 对象存储的兼容性，以及与虚拟机管理程序的兼容性(影响为临时的实例使用存储的能力)等因素。

存储管理 在存储型 OpenStack 云的设计中，必须处理一系列和存储管理相关的考虑因素。这些因素包括但不限于，备份策略(以及恢复策略，因为不能恢复的备份没什么用处)、数据评估等级存储管理、保管策略、数据存放的位置以及工作流自动化。

数据网格 数据网格在准确解答关于数据评估的问题方面非常有帮助。当前的信息科学方面的一个根本的挑战就是确定哪些数据值得保存，数据应该在哪个级别的访问和性能上存在，以及数据保留在存储系统当中的时间应该多长。数据网格，通过研究访问模式、所有权、商业单位 收益以及其它的元数据的值等的相关性，帮助做出相关的决定，提供关于数据的可行信息。

当尝试构建一个基于行业标准核心的灵活设计时，实现这个的一个办法，可能是通过使用不同的后端服务于不同的使用场景。

运营因素

对于通用型云，运维的因素会影响到设计的选择，且运维人员在大型的安装维护云环境面临着巨大的任务。

维护任务： 存储的解决方案需要考虑存储的维护以及其对底层负载的影响。

可靠性和可用性： 可靠性和可用性依赖于广域网的可用性以及服务提供商采取的预防措施的级别。

灵活性： 组织需要有选择off-premise和on-premise云存储属性的灵活性。此概念依赖于相关的决策条件，它可以互补，会在初始>时有直接的成本节约潜力。举例，持续的维护、灾难恢复、安全、记录保留规定、法规以及规则。

在对存储资源有非常高要求的云环境中，确保在环境中安装了监控和警报服务并且将它们进行配置以提供存储系统实时的健康状态和性能状态的信息，是非常之重要的。使用集成的管理控制台，或者能够将 SNMP 数据可视化的面板程序，能够帮助发现和解决存储集群中 可能出现的问题。

存储型的云设计需要包括：

- 监测物理硬件资源。
- 监测诸如温度和湿度等的环境信息。
- 监测诸如可用存储空间、内存和 CPU 等存储资源信息。
- 监测高级的存储性能数据，以确保存储系统正常运转。
- 监测网络资源情况，关注可能影响存储访问的网络服务中断问题。
- 日志集中收集
- 日志分析能力。
- 票务系统(或者与其它票务系统的集成)以跟踪问题。
- 对负责团队的警报和通知，或者能够在存储出现问题时能够修复问题的自动化系统。
- 配备有网络运营中心(NOC)员工并保持工作状态以解决问题。

管理效率

运营人员会经常需要替换失效的驱动器或者节点，并且不断进行存储硬件的维护工作。

准备和配置新增或者升级的存储，是另外一个关于管理存储资源的重要考虑因素。方便地部署、配置以及管理存储硬件的能力，能够造就一个容易管理的解决方案。这同样也利用了能够将整个解决方案中的其它部分组件自动化的管理系统。例如，复制、保管、数据备份和恢复。

应用的可知性

当设计在云中利用存储解决方案的应用时，需要将应用设计成为能够觉察到底层的存储子系统的存在以及其可用的特性。

当创建一个需要副本数据的应用时，我们建议将应用设计成为能够检测副本复制是否是底层存储子系统的原生特性。在复制功能不是底层存储系统的特性的情况下，才将应用设计成为自身能够提供副本复制服务。一个被设计为能够觉察到底层存储系统的应用，同样可以在很大范围的基础设施中部署，并且依然拥有一些最基本的行为，而不管底层的基础设施有什么不同。

容错和可用性

在 OpenStack 云中为存储系统的容错和可用性进行设计，对于块存储和对象存储服务来说，是有很大的不同的。对象存储服务具>有一致性以及区块容错，能够作为应用的功能。因此，它并不依赖于硬件 RAID 控制器提供的物理磁盘冗余。与此相对的，块存储资源节点 通常都配备有高级的 RAID 控制器以及高性能的磁盘，被设计为能够在硬件层面上提供容错。

块存储的容错和可用性

块存储资源节点通常都配备有高级的 RAID 控制器以及高性能的磁盘，被设计为能够在硬件层面上提供容错。

在应用要求发挥块存储设备的极端性能的场景下，我们建议您部署高性能的存储方案，例如SSD 磁盘驱动器或者闪存存储系统。

在对块存储具有极端需求的环境下，我们建议利用多个存储池所带来的好处。在这种场景中，每个设备池，在池中的所有硬件节点上，必须都具有相似的硬件设计以及磁盘配置。这使得设计能够为应用提供到多个块存储池的访问，每个存储池都有其自身的冗余、可用性 和性能特点。部署多个存储池时，对负责在资源节点上准备存储的块存储调度器的影响，也是一个重要的考虑因素。确保应用能够将其存储卷分散到不同的区域中，每个区域具有其自身的网络、电源以及冷却基础设施，能够给予租户建立具有容错性应用的能力。该应用会被分布到多个可用区域中。

在块存储资源节点之外，为 API 以及相关的负责准备存储和提供到存储的访问的服务的高可用性和冗余进行设计也非常重要。我们建议设计中包含一个硬件或者软件的负载均衡器层，以实现相关 REST API 服务的高可用性，从而实现提供不中断的服务。某些情况下， 可能还需要部署一个额外的负载均衡层，以提供到后端数据库服务的访问，该后端数据库负责提供以及保存块存储卷的状态信息。我们同样建议设计一个高可用的数据库解决方案来存放块存储的数据库。当前有许多的高可用数据库的解决方案，

比如说 Galera 和 MariaDB，能够被用以帮助保持数据库服务在线，以提供不间断的访问服务，保证租户能够管理块存储卷。

在对块存储具有极端需求的云环境中，网络的架构需要考虑东西向的带宽流量，这些带宽是实例使用可用的存储资源所必需的。所选择的网络设备必须支持巨型帧以传输大块的数据。某些情况下，可能还需要创建额外的后端存储网络，专用于为实例和块存储资源之间提供网络连接，来保证没有对于网络资源的争抢。

对象存储容错和可用性

虽然一致性和区块容错性都是对象存储服务的内生特性，对整个存储架构进行设计，以确保要实施的系统能够满足这些目标依然还是很重要的。OpenStack 对象存储服务将特定数量的数据副本作为对象存放与资源节点上。这些副本分布在整个集群之中，基于存在于集群 中所有节点上的一致性哈希环。

对象存储系统必须设计得拥有充足的区域提供法定数量，为所定义的复制份数。举个例子，在一个swift集群中配置了三份复制，那么建议区域数量是5。当然，也可以部署时少于这个数，但是这有数据不可用的风险以及API请求无法应答。基于这个原因，在对象存储系 统中要确保正确的数量。

每个对象存储区域须包含到其自身的可用区域里。每个可用区域须拥有独立的访问网络、电力和制冷等基础设施，确保访问数据不会中断。另外，每个可用区域须服务于对象存储池代理服务，而代理服务提供访问存放在对象节点的数据。在每个region的对象代理应充分 利用本地的读写，所以尽可能的访问本地存储资源。我们建议在上游的负载均衡的部署要确保代理服务是分布在不同的区域的。在一些情况下，有必要使用第三方的解决方案来处理跨地域分布式服务。

对象存储集群内的区域是一个逻辑上的划分。一个区域可以是下列情形中的任意之一：

- 单个节点上的一块磁盘
- 每节点就是一个zone
- Zone是多个节点的集合
- 多个机柜
- 多数据中心

决定合适的区域设计的关键是对象存储集群的扩展，还能同时提供可靠和冗余的存储系统。进一步讲，也许还需要根据不同的需求配置存储的策

略，这些策略包括副本，保留以及其它在特定区域会严重影响到存储设计的因素。

扩展存储服务

将块存储和对象存储服务相比较，增加存储容量和带宽是完全不同的流程。增加块存储的容量是一个相对简单的过程，增加对象存储系统的容量和带宽是一个复杂的任务，需要经过精心的规划和周全的考虑。

扩展块存储

块存储池增加存储容量的升级较简单而且毋须中断整个块存储服务。节点添加到池中仅通过简单的安装和配置合适的软硬件，然后允许节点通过消息总线报告给对应的池即可。这是因为块存储节点报告给调度器服务表明他们可用。一旦节点处于在线状态，可用的租户立马就可以使用它的存储资源。

在一些情形下，来自实例对块存储的需求会耗尽可用的网络带宽。因此，设计网络基础设施时，考虑到如此的块存储资源使用一定得很容易的增加容量和带宽。这通常涉及到使用动态路由协议或者是高级网络解决方案允许轻松添加下游设备以增加容量。无论是前端存储网络设计还是后端存储网络设计都得围绕着快速和容易添加容量和带宽的能力来展开。

扩展对象存储

为对象存储集群增加后端容量需要谨慎的规划和考虑。在设计阶段，决定最大partition power非常重要，这由对象存储服务所需要，它直接决定可以提供最大数量的分区。对象存储分发数据到所有的可用存储上，但是一个分区不可以再分到一块磁盘之外的地方，所以 分区的最大数量只能是磁盘的数量。

例如，一个系统开始使用单个磁盘，partition power是3，那么可以有8(2^3) 个分区。增加第二块磁盘意味着每块将拥有4个分区。一盘一分区的限制意味着此系统不可能拥有超过8块磁盘，它的扩展性受限。因此，一个系统开始时使用单个磁盘，且partition power是10的话可以使用1024(2^10)个磁盘。

为系统的后端存储增加了容量后，分区映射会带来数据重新分发到存储节点，在一些情况下，这些复制会带来超大的数据集合，在此种情况下，建议使用后端复制链接，它也不会阻断租户访问数据。

当越来越多的租户开始访问对象存储集群的数据，他们的数据开始不断的增长时，就有必要为服务增加前端带宽以应对不断的访问需求。为对象存储集群增加前端带宽要谨慎规划，设计对象存储代理供租户访问数据使

用，为代理层设计高可用解决方案且可轻松扩展等等 ，建议设计一前端负载均衡层，为租户和最终消费者提供更加可靠的访问数据方式，这个负载均衡层可以是分发到跨区域，跨region，甚至是跨地域，这要视实际对地理位置解决方案的需求而定。

一些情况下，要求给代理服务和存储节点之间的网络资源服务增加带宽和容量。基于这个原因，用于访问存储节点和代理服务的网络架构要设计的具有扩展性。

架构

当选择存储硬件时有三大块需要考虑：

- 成本
- 性能
- 可靠性

存储型OpenStack云须能反映出是针对存储密集型的负载因素。这些负载既不是计算密集型也不是网络密集型，网络也许在传输数据时有很高的负载，但是这是另外一个网络密集的范畴。

对于存储型OpenStack架构设计来说，存储硬件的选择将决定整个架构的性能和可扩展性。在设计过程中，一些需要考虑的不同因素：

成本	使用什么存储架构和选择什么硬件将影响着开销。
性能	存储I/O请求的延迟影响着性能。解决方案的选择会影响到性能的需求。
可扩展性	此节参考术语”扩展性“，来解释存储解决方案的表现可扩展到最大规模是怎么个好法。一个存储解决方案在小型配置时表现良好，但是在规模扩展的过程中性能降低，这就不是好的扩展性，也不会被考虑。换句话说，一个解决方案只有在规模扩展最大性能没有任何的降低才是好的扩展性。
延伸性	规模扩展性是指在整个解决方案的扩张的整体能力。一个存储解决方案可以扩展到50PB，一定比仅能扩展到10PB的规模扩展性强，也会更加倾向于前者。

注意

This meter is related to scalability.

对于存储型OpenStack云来说，存储的延迟是主要应该考虑的。使用固态硬盘(SSD)以使实例存储延迟最小以及减少由于等待存储产生的CPU延迟，可大大提升性能。考虑让计算主机的磁盘子系统使用RAID控制卡对改善性能也大有帮助。

存储架构的选择，以及存储硬件的选择，上面列出的几个因素是相互有冲突的，需评估决定。如果是横向扩展解决方案的选择诸如Ceph，GlusterFS，或类似的,如果是单一的，高扩展性的，那么选择中心化的存储阵列就没错。如果中心存储阵列能够满 足需求，那么硬件的决定就是如何选择阵列供应商的事情了。使用商业硬件和开源软件来构建存储阵列也是完全可能的，只是需要构建这样的系统需要专业的人员罢了。

反过来说，一个横向扩展的存储解决方案使用直接挂接存储(DAS)给服务器也许是个恰当的选择。如果是这样的话，服务器硬件就需要配置以支持此种存储解决方案。

一些潜在的可能影响到存储型OpenStack云的特定的存储架构(及其相应的存储硬件):

连通性 根据所选择的存储解决方案，要确保它的连接和存储解决方案的需求是匹配的。如果是选择了中心化的存储阵列，决定有多少台hypervisor会连接到此阵列中是很重要的。连接会影响到延迟以及它们的性能，所以监测网络因素以达到最小延迟，提升整体的性能。

延迟 决定的是如果用例中拥有并行或高度变化的延迟。

吞吐量 要确保存储解决方案是基于应用的需求整个被优化。

服务器硬件 如果解决方案中使用了DAS，这会影响到但不限于，服务器硬件的选择会波及到主机密度、实例密度、电力密度、操作系统-hypervisor、以及管理工具。

计算(服务器)硬件选择

模拟计算(服务器)硬件需要从以下四个不同的方面进行评估：

服务器密度 关于多少台服务器能够放下到一个给定尺寸的物理空间的量度，比如一个机柜单位[U]。

资源容量 CPU核数，多少内存，或者多少存储可以交付。

延伸性 在达到容量之前用户可以增加服务器来增加资源的数量。

成本 相对硬件的权重是体现构建系统的设计水准所需要的。

为达到期望的目的而决定最佳设计需要对一些因素作出取舍和平衡。举例来说，增加服务器密度意味着牺牲资源的容量或扩展性。增加资源容量或扩展性又增加了开销但是降低了服务器密度。减少开销又意味着减低支持力度，服务器密度，资源容量和扩展性。

在选择服务器硬件时计算能力(CPU核和内存容量)是次要考虑的,服务器硬件必须能够提供更多的CPU插槽，更多的CPU核的数量，以及更多的内存，至于网络连接和存储容量就显得次要一些。硬件需要的配置以提供足够的网络连接和存储容量满足用户的最低需求即可，但是这不是主要需要考虑的。

一些服务器硬件的因素更加适合其他类型的，但是CPU和内存的能力拥有最高的优先级。

- 大多数的刀片服务器都支持双插槽、多核的CPU的。要避免去选择“全宽”或“全高”的刀片，它们会损失服务器密度。举个例子，使用高密度的刀片服务器，如HP BladeSystem和Dell ,它们都使用半高的刀片，可支持16台服务器，且仅占用10个机柜单元，它相比于全高的刀片有效的减低了50%的密度，因为全高的刀片在每10个机柜单元仅可以放置8台服务器。

警告

它相比于全高的刀片有效的减低了50%的密度，因为全高的刀片在(每10个机柜单元仅可以放置8台服务器)。

- 1U机架式服务器要比刀片服务解决方案提供更大的服务器密度。但是会有双CPU插槽、多核CPU配置的限制。

注意

就在IceHouse版本发布时，无论是HP，IBM还是Dell都无法提供多于2个CPU插槽的1U机架式服务器。

要想使1U的服务器支持超过2个插槽，用户需要通过原始设计制造商(ODM)或二线制造商来购买。

警告

这会给企业带来额外的问题：重新评估供应商的政策，支持的力度是否够，非1线供应商的硬件质量保证等。

- 2U机架式服务器提供四插槽、多核CPU的支持，但是它相应的降低了服务器密度(相当于1U机架式服务器的一半)。

- 大型机架式服务器，比如4U服务器，可提供更为强大的CPU容量。通常支持4个甚至8个CPU插槽。拥有很强的扩展性，但是这些服务器会带来低密度，以及更加昂贵的开销。

- “雪撬服务器”,支持在单个2U或3U的空间放置多个独立的服务器,增加的密度超过典型的1U-2U机架服务器，

  举例来说，很多雪撬服>务器提供在2U的空间置放4个独立的双插槽CPU,即2U服务器拥有8颗CPU，尽管如此，分离节点的双插槽限制不足以抵消它们额外带来的开销和配置的复杂性。

一些服务器硬件的因素更加适合其他类型的，但是CPU和内存的能力拥有最高的优先级。

实例密度　在此架构中实例密度要被考虑为低，因此CPU和内存的超额认购比例也要低。为了支持实例低密度的预期扩展需要更多的主机，尤其是设计中使用了双插槽的硬件。

主机密度　解决高主机计数的另外的办法是使用四路平台。这样的话降低了主机密度，也增加了机架数，这样的配置会影响到网络需求，电源连接数量，还有可能影响的制冷需求。

电源和制冷密度　电力和制冷的密度需求要低于刀片、雪撬或1U服务器，因为(使用2U,3U甚至4U服务器)拥有更低的主机密度。对于数据中心内有旧的基础设施，这是非常有用的特性。

存储型OpenStack架构设计的服务器硬件选择只有两种结果可决定：纵向扩展抑或横向扩展。在少而大的服务器主机和多而小的服务器主机选择更好的解决方案，取决于多种因素：预算、电力、制冷、物理机架和地板的空间、售后服务力度、以及可管理性。

网络硬件选择

选择网络硬件主要考虑应包括：

端口数目　用户将会对网络设备有充足的端口数有需求。

端口密度　网络的设计会受到物理空间的影响，需要提供足够的端口数。一个占用1U机柜空间的可提供48个 10GbE端口的交换机，显而易见的要比占用2U机柜空间的仅提供24个 10GbE端口的交换机有着更高的端口密度。高端口密度是首先选择的，因为其可以为计算和存储省下机柜空间。这也会引起人们的思考，容错的情况呢？电力密度？高密度的交换机更加的昂贵，也应该被考虑使用，但是没有必要覆盖设计中所有的网络，要视实际情况而定。

端口速度 网络硬件必须支持常见的网络速度，例如：1GbE、10GbE 或者 40GbE(甚至是 100GbE)。

冗余 网络硬件冗余级别需求会被用户对高可用和开销的考虑所影响。网络冗余可以是增加双电力供应也可以是结对的交换机。

注意

如果这是必须的，那么对应的服务器硬件就需要配置以支持冗余的情况。用户的需求也是决定是否采用全冗余网络基础设施的关键。

电力要求 确保物理数据中心为所选择的网络硬件提供了必要的电力。对于机柜顶端型(ToR)交换机没有什么影响，但是spine交换机的叶和fabric以及排尾交换机(EoR)就可能发生问题，假如电力不足的话。

协议支持 使用特定的网络技术如RDMA,SRP,iSER或SCST来提高单个存储系统的性能是可能，但是讨论这些技术已经超出了本书的范围。

软件选择

对于存储型OpenStack架构设计来说，选择包含的软件有以下三个方面的考虑：

- 操作系统(OS)和虚拟机管理软件
- OpenStack 组件
- 增强软件

上述选择项的任何一个设计决定都会影响到其余两个的OpenStack架构设计。

操作系统和虚拟机管理软件

操作系统和hypervisor对于整个设计有着深刻的影响。选择一个特定的操作系统和hypervisor能够直接影响到服务器硬件的选择。确保存储硬件和拓扑支持选择好的操作系统和Hypervisor组合。也得确保网络硬件的选择和拓扑可在选定的操作系统和Hypervisor组合下正常工作。例如，如果设计中使用了链路聚合控制协议(LACP)的话，操作系统和hypervisor都需要支持它才可正常工作。

操作系统和hypervisor的选择会影响到下列几个方面：

成本 选择商业支持的hypervisor如微软的Hyper-V，和选择社区支持的开源hypervisor如kinstance或Xen，在开销模式上是完全不同的。甚至在开源解决方案也表现不同，如选择Ubuntu而不是RedHat(或类似)在支持上有着不同的开销。换句话说，业务和应用的需求主宰着选用特殊的或商业支持的hypervisor。

受支持程度 无论选择那个hypervisor,相关的技术人员都要经过适当的培训和知识积累，才可以支持所选择的操作系统和hypervisor组合。如果这些维护人员没有培训过，那么就得提供，当然它会影响到设计中的之处。另外一个考虑的方面就是关于操作系统-hypervisor的支持问题，商业产品如Red Hat，SUSE或Windows等的支持是由操作系统供应商来支持的，如果选用了开源的平台，支持大部分得来自于内部资源。无论何种决定，都会影响到设计时的支出。

管理工具 Ubuntu和Kinstance的管理工具和VMware vSphere的管理工具是不一样的。尽管OpenStack对它们都支持。这也会对其他的设计有着非常不同的影响，结果就是选择了一种组合，然后再据此做出后面的选择。

规模和性能 确保所选择的操作系统和hypervisor组合能够满足扩展和性能的需求。所选择的架构需要满足目标实例-主机的比例。

安全性 确保设计能够在维护负载需求时能够容纳正常的所安装的应用的安全补丁。为操作系统-hypervisor组合打安全补丁的频率会影响>到性能，而且补丁的安装流程也会影响到维护工作。

支持的特性 决定那些特性是需要OpenStack提供的。这通常也决定了操作系统-hypervisor组合的选择。一些特性仅在特定的操作系统和hypervisor中是可用的。举例，如果确认某些特性无法实现，设计也许需要修改代码去满足用户的需求。

互操作性 用户需要考虑此操作系统和Hypervisor组合和另外的操作系统和hypervisor怎么互动，甚至包括和其它的软件。操作某一操作系统-hypervisor组合的故障排除工具，和操作其他的操作系统-hypervisor组合也许根本就不一样，那结果就是，设计时就需要交付此两种工具集的互操作性。

OpenStack 组件

Which OpenStack components you choose can have a significant impact on the overall design. While there are certain components that are always present, Compute and Image service, for example, there are other services that may not need to be present. As an example, a certain design may not require the Orchestration module. Omitting Orchestration would not

typically have a significant impact on the overall design, however, if the architecture uses a replacement for OpenStack Object Storage for its storage component, this could potentially have significant impacts on the rest of the design.

一个存储型设计也许需要使用Orchestration，能够启动带块设备卷的实例，以满足存储密集型任务处理。

在存储型OpenStack架构设计中，下列组件是典型使用的：

- OpenStack 认证(keystone)
- OpenStack GUI界面 (horizon)
- OpenStack 计算 (nova)(包括使用多hypervisor驱动)
- OpenStack 对象存储 (swift)(或者是另外的对象存储解决方案)
- OpenStack 块存储(cinder)
- OpenStack Image service (glance)
- OpenStack 网络 (neutron) 或遗留网路服务 (nova-network)

排除一些特定的OpenStack组件会让其他组件的功能受到限制。如果在一个设计中有Orchestration模块但是没有包括Telemetry模块，那么此设计就无法使用Orchestration带来自动伸缩功能的优点(Orchestration需要Telemetery提供监测数据)。用户使用Orchestration在计算密集型处理任务时可自动启动大量的实例，因此强烈建议在计算型架构设计中使用Orchestration。

增强软件

OpenStack为了构建一个平台提供云服务，是完全公平的收集软件的项目。在任何给定的OpenStack设计中都需要考虑那些附加的软件。

联网软件

OpenStack Networking (neutron) provides a wide variety of networking services for instances. There are many additional networking software packages that may be useful to manage the OpenStack components themselves. Some examples include HAProxy, keepalived, and various routing daemons (like Quagga). The OpenStack High Availability Guide describes some of these software packages, HAProxy in particular. See

the Network controller cluster stack chapter of the OpenStack High Availability Guide.

管理软件

管理软件所包含的能够提供的软件：

- 集群
- 日志记录
- 监控
- 警告

重要

这包括能够提供集群、日志、监测及预警等的软件。在此目录什么因素决定选择什么软件包已经超出了本书的范围。

在集群软件中如 Corosync和Pacemaker在可用性需求中占主流。包含这些软件包是主要决定的，要使云基础设施具有高可用的话，当然在部署之后会带来复杂的配置。OpenStack 高可用指南

对日志、监测以及报警的需求是由运维考虑决定的。它们的每个子类别都包含了大量的属性。举例，在日志子类别中，某些情况考虑使用Logstash,Splunk，instanceware Log Insight等，或者其他的日志聚合-合并工具。日志需要存放在中心地带，使分析数据变得更为简单。日志数据分析引擎亦得提供自动化和问题通知，通过提供一致的预警和自动修复某些常见的已知问题。

如果这些软件包都需要的话，设计必须计算额外的资源使用率(CPU，内存，存储以及网络带宽)。其他一些潜在影响设计的有：

- 操作系统-Hypervisor组合：确保所选择的日志系统，监测系统，或预警工具都是被此组合所支持的。
- 网络硬件：网络硬件的选择，要看其支持日志系统、监测系统以及预警系统的情况。

数据库软件

OpenStack组件通常需要访问后端的数据库服务以存放状态和配置信息。选择合适的后端数据库以满足可用性和容错的需求，这是OpenStack服务所要求的。

MySQL是OpenStack通常考虑的后端数据库，其它和MySQL兼容的数据也同样可以很好的工作。

注意

Telemetry 使用MongoDB。

为数据库提供高可用的解决方案选择将改变基于何种数据库。如果是选择了MySQL,有几种方案可供选择，如果是主-主模式集群，则使用 Galera复制技术；如果是主-备模式则必须使用共享存储。每个潜在的方案都会影响到架构的设计：

- 解决方案采用Galera/MariaDB，需要至少3个MySQL节点。
- MongoDB尤其自身的设计考虑，回馈就是可让数据库高可用。
- 通常在OpenStack的设计中是不包括共享存储的，但是在高可用的设计中，为了特定的实现一些组件会用得到共享存储。

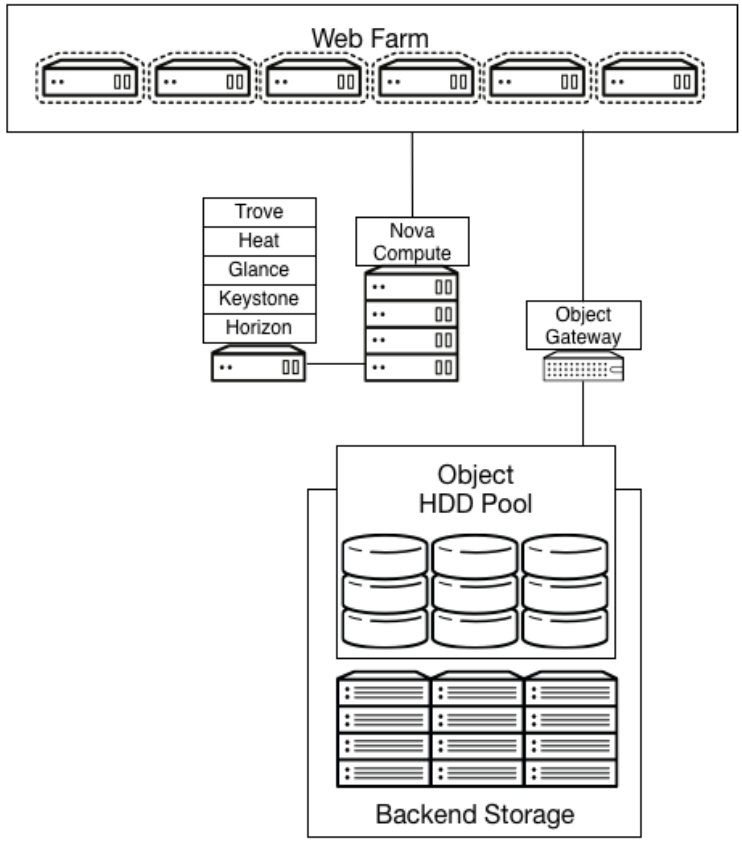
示例

存储型的架构十分依赖于明确的应用场景。本节讨论了三种不同使用场景的例子

- 一个带有 RESTful 接口的对象存储
- 使用并行文件系统的计算分析
- 高性能数据库

本例描绘了没有高性能需求的 REST 接口。

Swift 是一个高度可扩展的对象存储，同时它也是 OpenStack 项目中的一部分。以下是说明示例架构的一幅图示：



所展现的 REST 接口不需要一个高性能的缓存层，并且被作为一个运行在传统设备 上的传统的对象存储抛出。

此例使用了如下组件：

网络：

・10 GbE 可水平扩展的分布式核心后端存储及前端网络。

存储硬件：

・10 storage servers each with 12x4 TB disks equaling 480 TB total space with approximately 160 TB of usable space after replicas.

代理：

・3x 代理

・2x10 GbE 绑定的前端

- 2x10 GbE 后端绑定
- 到后端存储集群大约 60 Gb 的总带宽

注意

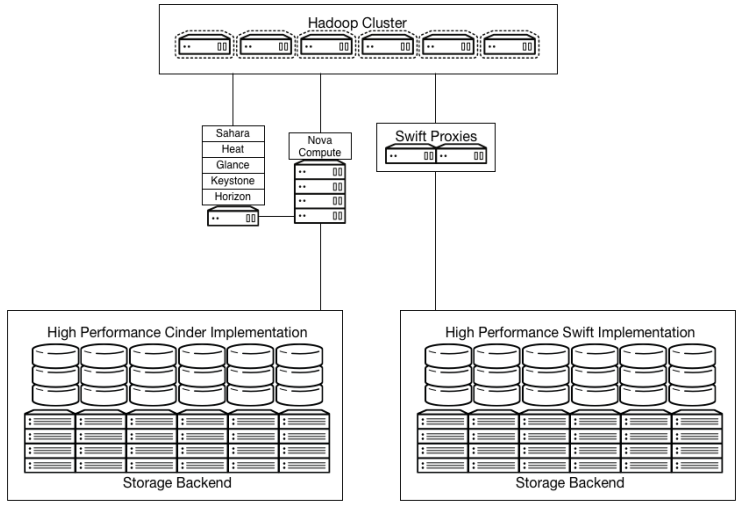
对于一些应用来说，可能需要引入第3方的缓存层以实现合意的性能。

带数据处理服务的计算分析

对大规模数据集的分析十分依赖于存储系统的性能。有些云使用类似Hadoop分布式文件系统(HDFS)的低效存储系统，可能会引起性能问题。

这个问题的一个解决方案是部署一个设计时便将性能考虑在内的存储系统。传统上来说，并行文件系统填补了 HPC 空间里的这个需要，在适用的时候，可以作为大规模面向性能的系统的一个备份的方案考虑。

OpenStack 通过数据处理项目Sahara与 Hadoop 集成。该项目被用以在云中管理 Hadoop 集群。此示意图展示了OpenStack作为高性能存储的需求：



实际的硬件需求以及配置与下面的高性能数据库例子相似。在这个场景中，采用的架构使用了 Ceph 的 Swift 兼容 REST 接口，以及允许连接至一个缓存池以加速展现出来的池的特性。

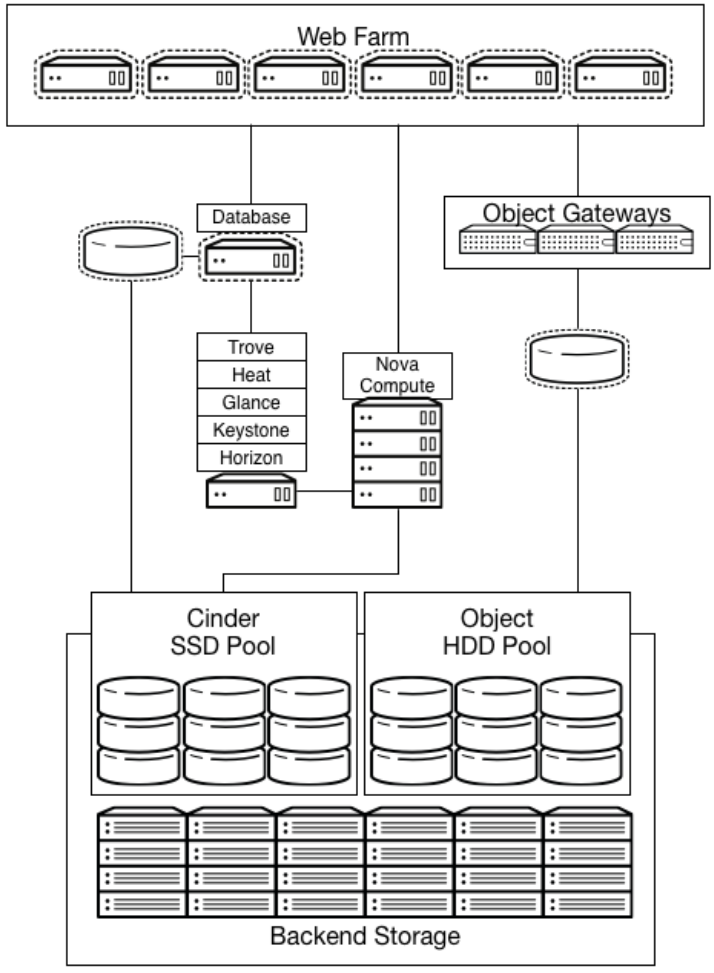
带数据库服务的高性能数据库

数据库是一种常见的能够从高性能数据后端中获益的负载。尽管企业级的存储并不在需求中，很多环境都已经拥有能够被用作 OpenStack 云后端的

存储。如下图中所示，可以划分出一个存储池出来，使用 OpenStack 块存储向实例提供块设备，同样也可以提供对象存储 接口。在这个例子中，数据库的 I-O 需求非常高，所需的存储是从一个高速的 SSD 池中抛出来的。

一个存储系统被用以抛出 LUN，其后端由通过使用传统的存储阵列与 OpenStack 块存储集成的一系列的 SSD 所支撑，或者是一个类似于 Ceph 或者 Gluster 之类的存储平台。

这种类型的系统也能够在其他场景下提供额外的性能。比如说，在下面的数据库例子中，SSD 池中的一部分可以作为数据库服务器的块设备。在高性能分析的例子中，REST 接口将会被内联的 SSD 缓存层所加速。



选用 Ceph 以抛出 Swift 兼容的 REST 接口，同样的也从一个分布式的存储集群中提供了块级别的存储。Ceph 非常的灵活，并且具有能够降低运营成

本的许多特性，比如说自动修复以及自动平衡。Ceph 中的 erasure coded 池被用以最大化可用空间的量。

注意

请注意关于 erasure coded 池的使用有特殊的考虑因素，比如说，更高的计算要求以及对象上所允许的操作限制。另外，部分写入在 erasure coded 的池中也不被支持。

跟上面的例子相关的 Ceph 的一个可能架构需要如下组件：

网络：

- 10 GbE 可水平扩展的分布式核心后端存储及前端网络。

存储硬件：

- 5 台作为缓存池的存储服务器，每台 24x1 TB SSD
- 10 storage servers each with 12x4 TB disks which equals 480 TB total space with about approximately 160 TB of usable space after 3 replicas

REST 代理：

- 3x 代理
- 2x10 GbE 绑定的前端
- 2x10 GbE 后端绑定
- 到后端存储集群大约 60 Gb 的总带宽

SSD 缓存层被用以直接向宿主机或者实例抛出块设备。SSD 缓存系统也能够被用作 REST 接口的内联缓存。

第 5 章 网络型

目录

All OpenStack deployments depend on network communication in order to function properly due to its service-based nature. In some cases, however, the network elevates beyond simple infrastructure. This chapter discusses architectures that are more reliant or focused on network services. These architectures depend on the network infrastructure and require network services that perform reliably in order to satisfy user and application requirements.

可能的用例方案包括：

内容分发网络
: This includes streaming video, viewing photographs, or accessing any other cloud-based data repository distributed to a large number of end users. Network configuration affects latency, bandwidth, and the distribution of instances. Therefore, it impacts video streaming. Not all video streaming is consumer-focused. For example, multicast videos (used for media, press conferences, corporate presentations, and web conferencing services) can also use a content delivery network. The location of the video repository and its relationship to end users affects content delivery. Network throughput of the back-end systems, as well as the WAN architecture and the cache methodology, also affect performance.

网络管理功能
: Use this cloud to provide network service functions built to support the delivery of back-end network services such as DNS, NTP, or SNMP. A company can use these services for internal network management.

网络服务提供	Use this cloud to run customer-facing network tools to support services. Examples include VPNs, MPLS private networks, and GRE tunnels.
web 门户或 web 服务	Web servers are a common application for cloud services, and we recommend an understanding of their network requirements. The network requires scaling out to meet user demand and deliver web pages with a minimum latency. Depending on the details of the portal architecture, consider the internal east-west and north-south network bandwidth.
高速及大量数据的事务性系统	These types of applications are sensitive to network configurations. Examples include financial systems, credit card transaction applications, and trading and other extremely high volume systems. These systems are sensitive to network jitter and latency. They must balance a high volume of East-West and North-South network traffic to maximize efficiency of the data delivery. Many of these systems must access large, high performance database back ends.
高可用	These types of use cases are dependent on the proper sizing of the network to maintain replication of data between sites for high availability. If one site becomes unavailable, the extra sites can serve the displaced load until the original site returns to service. It is important to size network capacity to handle the desired loads.
大数据	Clouds used for the management and collection of big data (data ingest) have a significant demand on network resources. Big data often uses partial replicas of the data to maintain integrity over large distributed clouds. Other big data applications that require a large amount of network resources are Hadoop, Cassandra, NuoDB, Riak, and other NoSQL and distributed databases.
虚拟桌面基础设施 (VDI)	This use case is sensitive to network congestion, latency, jitter, and other network characteristics. Like video streaming, the user experience is important. However, unlike video streaming, caching

	is not an option to offset the network issues. VDI requires both upstream and downstream traffic and cannot rely on caching for the delivery of the application to the end user.
IP 语音(VoIP)	This is sensitive to network congestion, latency, jitter, and other network characteristics. VoIP has a symmetrical traffic pattern and it requires network quality of service (QoS) for best performance. In addition, you can implement active queue management to deliver voice and multimedia content. Users are sensitive to latency and jitter fluctuations and can detect them at very low levels.
视频会议和 web 会议	This is sensitive to network congestion, latency, jitter, and other network characteristics. Video Conferencing has a symmetrical traffic pattern, but unless the network is on an MPLS private network, it cannot use network quality of service (QoS) to improve performance. Similar to VoIP, users are sensitive to network performance issues even at low levels.
高性能计算(HPC)	This is a complex use case that requires careful consideration of the traffic flows and usage patterns to address the needs of cloud clusters. It has high east-west traffic patterns for distributed computing, but there can be substantial north-south traffic depending on the specific application.

用户需求

Network-focused architectures vary from the general-purpose architecture designs. Certain network-intensive applications influence these architectures. Some of the business requirements that influence the design include:

- Network latency through slow page loads, degraded video streams, and low quality VoIP sessions impacts the user experience. Users are often not aware of how network design and architecture affects their experiences. Both enterprise customers and end-users rely on the network for delivery of an application. Network performance problems can result in a negative experience for the end-user, as well as productivity and economic loss.

- Regulatory requirements: Consider regulatory requirements about the physical location of data as it traverses the network. In addition, maintain network segregation of private data flows while ensuring an encrypted network between cloud locations where required. Regulatory requirements for encryption and protection of data in flight affect network architectures as the data moves through various networks.

很多辖区对于云环境中的数据的保管及管理都有相关的法律上或者监管上的要求。这些规章的常见领域包括：

- 确保持久化数据的保管和记录管理以符合数据档案化需求的数据保留政策。
- 管理数据的所有权和责任的数据所有权政策。
- 管理位于外国或者其它辖区的数据存储问题的数据独立性政策。
- Data compliance policies govern where information can and cannot reside in certain locations.

Examples of such legal frameworks include the data protection framework of the European Union (http://ec.europa.eu/justice/data-protection/) and the requirements of the Financial Industry Regulatory Authority (http://www.finra.org/Industry/Regulation/FINRARules) in the United States. Consult a local regulatory body for more information.

高可用问题

Depending on the application and use case, network-intensive OpenStack installations can have high availability requirements. Financial transaction systems have a much higher requirement for high availability than a development application. Use network availability technologies, for example quality of service (QoS), to improve the network performance of sensitive applications such as VoIP and video streaming.

High performance systems have SLA requirements for a minimum QoS with regard to guaranteed uptime, latency, and bandwidth. The level of the SLA can have a significant impact on the network architecture and requirements for redundancy in the systems.

风险

错误的网络配置	Configuring incorrect IP addresses, VLANs, and routers can cause outages to areas of the network or, in the worst-case scenario, the entire cloud infrastructure.

	Automate network configurations to minimize the opportunity for operator error as it can cause disruptive problems.
容量计划	Cloud networks require management for capacity and growth over time. Capacity planning includes the purchase of network circuits and hardware that can potentially have lead times measured in months or years.
网络调优	Configure cloud networks to minimize link loss, packet loss, packet storms, broadcast storms, and loops.
单点故障 (SPOF)	Consider high availability at the physical and environmental layers. If there is a single point of failure due to only one upstream link, or only one power supply, an outage can become unavoidable.
复杂性	An overly complex network design can be difficult to maintain and troubleshoot. While device-level configuration can ease maintenance concerns and automated tools can handle overlay networks, avoid or document non-traditional interconnects between functions and specialized hardware to prevent outages.
非标准特性	There are additional risks that arise from configuring the cloud network to take advantage of vendor specific features. One example is multi-link aggregation (MLAG) used to provide redundancy at the aggregator switch level of the network. MLAG is not a standard and, as a result, each vendor has their own proprietary implementation of the feature. MLAG architectures are not interoperable across switch vendors, which leads to vendor lock-in, and can cause delays or inability when upgrading components.

安全性

Users often overlook or add security after a design implementation. Consider security implications and requirements before designing the physical and logical network topologies. Make sure that the networks are properly segregated and traffic flows are going to the correct destinations without crossing through locations that are undesirable. Consider the following example factors:

- 防火墙

- 覆盖网络之间的相互连接以连接分离的租户网络
- 使路由通过或者避免通过某个特定网络

How networks attach to hypervisors can expose security vulnerabilities. To mitigate against exploiting hypervisor breakouts, separate networks from other systems and schedule instances for the network onto dedicated compute nodes. This prevents attackers from having access to the networks from a compromised instance.

技术因素

When you design an OpenStack network architecture, you must consider layer-2 and layer-3 issues. Layer-2 decisions involve those made at the data-link layer, such as the decision to use Ethernet versus Token Ring. Layer-3 decisions involve those made about the protocol layer and the point when IP comes into the picture. As an example, a completely internal OpenStack network can exist at layer 2 and ignore layer 3. In order for any traffic to go outside of that cloud, to another network, or to the Internet, however, you must use a layer-3 router or switch.

The past few years have seen two competing trends in networking. One trend leans towards building data center network architectures based on layer-2 networking. Another trend treats the cloud environment essentially as a miniature version of the Internet. This approach is radically different from the network architecture approach in the staging environment: the Internet only uses layer-3 routing rather than layer-2 switching.

基于二层协议设计的网络相比基于三层协议设计的网络有一定优势。尽管使用桥接来扮演路由的网络角色有困难，很多厂商、客户以及服务提供商都选择尽可能多地在他们的网络中使用以太网。选择二层网络设计的好处在于：

- 以太网帧包含了所有联网所需的要素。这些要素包括但不限于，全局唯一的源地址，全局唯一的目标地址，以及错误控制。
- 以太网帧能够承载任何类型的包。二层上的联网独立于三层协议之外。
- Adding more layers to the Ethernet frame only slows the networking process down. This is known as 'nodal processing delay'.
- You can add adjunct networking features, for example class of service (CoS) or multicasting, to Ethernet as readily as IP networks.
- VLAN 是一个简单的用于网络隔离的机制。

Most information starts and ends inside Ethernet frames. Today this applies to data, voice (for example, VoIP), and video (for example, web cameras). The concept is that, if you can perform more of the end-to-end transfer of information from a source to a destination in the form of Ethernet frames, the network benefits more from the advantages of Ethernet. Although it is not a substitute for IP networking, networking at layer 2 can be a powerful adjunct to IP networking.

使用二层以太网相对于使用三层 IP 网络有如下优势：

- 速度
- 减少了 IP 层的开销
- No need to keep track of address configuration as systems move around. Whereas the simplicity of layer-2 protocols might work well in a data center with hundreds of physical machines, cloud data centers have the additional burden of needing to keep track of all virtual machine addresses and networks. In these data centers, it is not uncommon for one physical node to support 30-40 instances.

重要

数据帧级别上的联网与数据包级别上的 IP 地址存在与否并没有关系。几乎所有端口、链路以及在 LAN 交换机构成的网络上的设备仍然有其 IP 地址，同样地，所有的源和目标主机也都有。有很多原因需要继续保留 IP 地址。其中最大的一个理由是网络管理上的需要。一个没有 IP 地址的设备或者连接通常对于大多数的管理程序来说是不可见的。包括远程接入的诊断工具、用于传输配置和软件的文件传输工具以及其它类似的应用程序都不能够在没有 IP 地址或者 MAC 地址的情况下运行。

二层架构的局限性

在传统数据中心之外，二层网络架构的局限性更加明显。

- VLAN 的数目被限制在 4096。
- 存储在交换表中的 MAC 地址数目是有限的。
- You must accommodate the need to maintain a set of layer-4 devices to handle traffic control.
- 通常被用于实现交换机冗余的 MLAG 是私有的解决方案，不能扩展至两个设备以上，并且迫使得厂商的选择受到限制。

- 解决一个没有 IP 地址和 ICMP 的网络上的问题可能会很困难。
- Configuring ARP can be complicated on large layer-2 networks.
- All network devices need to be aware of all MACs, even instance MACs, so there is constant churn in MAC tables and network state changes as instances start and stop.
- Migrating MACs (instance migration) to different physical locations are a potential problem if you do not set ARP table timeouts properly.

很重要的一点是，必须意识到二层网络上用于网络管理的工具非常有限。因此控制流量非常困难，由于二层网络没有管理网络或者对流量进行整形的机制，解决网络问题也非常困难。其中一个原因是网络设备没有 IP 地址。因此，在二层网络中没有合理的方式来检查网络延迟。

在大规模的二层网络上，配置 ARP 学习也可能很复杂。交换机上的 MAC 地址超时设置非常关键，并且，错误的设置可能引起严重的性能问题。例如 Cisco 交换机上的 MAC 地址的超时时间非常长。迁移 MAC 地址到另外的物理位置以实现实例的迁移就可能是一个显著的问题。这种情形下，交换机中维护的网络信息与实例的新位置信息就会不同步。

In a layer-2 network, all devices are aware of all MACs, even those that belong to instances. The network state information in the backbone changes whenever an instance starts or stops. As a result there is far too much churn in the MAC tables on the backbone switches.

三层架构的优势

In the layer 3 case, there is no churn in the routing tables due to instances starting and stopping. The only time there would be a routing state change is in the case of a Top of Rack (ToR) switch failure or a link failure in the backbone itself. Other advantages of using a layer-3 architecture include:

- 三层网络提供了与因特网相同的弹性及可扩展性。
- 使用路由度量进行流量控制非常直观。
- You can configure layer 3 to use BGP confederation for scalability so core routers have state proportional to the number of racks, not to the number of servers or instances.
- Routing takes instance MAC and IP addresses out of the network core, reducing state churn. Routing state changes only occur in the case of a ToR switch failure or backbone link failure.

- 有很多经过完善测试的工具，例如 ICMP，来监控和管理流量。
- Layer-3 architectures enable the use of Quality of Service (QoS) to manage network performance.

三层架构的局限性

The main limitation of layer 3 is that there is no built-in isolation mechanism comparable to the VLANs in layer-2 networks. Furthermore, the hierarchical nature of IP addresses means that an instance is on the same subnet as its physical host. This means that you cannot migrate it outside of the subnet easily. For these reasons, network virtualization needs to use IP encapsulation and software at the end hosts for isolation and the separation of the addressing in the virtual layer from the addressing in the physical layer. Other potential disadvantages of layer 3 include the need to design an IP addressing scheme rather than relying on the switches to keep track of the MAC addresses automatically and to configure the interior gateway routing protocol in the switches.

网络建议的总结

OpenStack has complex networking requirements for several reasons. Many components interact at different levels of the system stack that adds complexity. Data flows are complex. Data in an OpenStack cloud moves both between instances across the network (also known as East-West), as well as in and out of the system (also known as North-South). Physical server nodes have network requirements that are independent of instance network requirements, which you must isolate from the core network to account for scalability. We recommend functionally separating the networks for security purposes and tuning performance through traffic shaping.

You must consider a number of important general technical and business factors when planning and designing an OpenStack network. They include:

- 对厂商独立性的需要。要避免硬件或者软件的厂商选择受到限制，设计不应该依赖于某个厂商的路由或者交换机的独特特性。
- 对于大规模地扩展生态系统以满足百万级别终端用户的需要。
- 对于支持不确定的平台和应用的要求。
- 对于实现低成本运营以便获益于大规模扩展的设计的需要。
- 对于保证整个云生态系统中没有单点故障的需要。

- 对于实现高可用架构以满足客户服务等级协议(SLA)需求的要求。
- 对于容忍机柜级别的故障的要求。
- 对于最大化灵活性以便构架出未来的生产环境的要求。

Bearing in mind these considerations, we recommend the following:

- Layer-3 designs are preferable to layer-2 architectures.
- 设计一个密集的多路径网络核心以支持多个方向的扩展和确保灵活性。
- 使用具有层次结构的地址分配机制，因为这是扩展网络生态环境的唯一可行选项。
- 使用虚拟联网将实例服务网络流量从管理及内部网络流量中隔离出来。
- 使用封装技术隔离虚拟网络。
- 使用流量整形工具调整网络性能。
- 使用 eBGP 连接至因特网上行链路。
- 在三层网络上使用 iBGP 将内部网络流量扁平化。
- 确定块存储网络的最高效配置。

额外的考虑因素

There are several further considerations when designing a network-focused OpenStack cloud.

OpenStack 联网方式对传统联网(nova-network)的考虑

Selecting the type of networking technology to implement depends on many factors. OpenStack Networking (neutron) and legacy networking (nova-network) both have their advantages and disadvantages. They are both valid and supported options that fit different use cases:

传统联网方式(nova-network)	OpenStack 联网方式(neutron)
简单，单独一个代理程序	复杂，多个代理程序
更加成熟，稳定的	更新，正在发展不断成熟中
Flat 或者 VLAN	Flat、VLAN、覆盖网络、二层到三层、软件定义网络(SDN)
没有插件支持	有第三方的插件支持

传统联网方式(nova-network)	OpenStack 联网方式(neutron)
能够很好地扩展	需要第三方插件进行扩展
没有多层拓扑	具有多层拓扑

冗余网络：柜顶交换机高可用风险分析

A technical consideration of networking is the idea that you should install switching gear in a data center with backup switches in case of hardware failure.

Research indicates the mean time between failures (MTBF) on switches is between 100,000 and 200,000 hours. This number is dependent on the ambient temperature of the switch in the data center. When properly cooled and maintained, this translates to between 11 and 22 years before failure. Even in the worst case of poor ventilation and high ambient temperatures in the data center, the MTBF is still 2-3 years. See http://www.garrettcom.com/techsupport/papers/ethernet_switch_reliability.pdf for further information.

In most cases, it is much more economical to use a single switch with a small pool of spare switches to replace failed units than it is to outfit an entire data center with redundant switches. Applications should tolerate rack level outages without affecting normal operations, since network and compute resources are easily provisioned and plentiful.

为将来做准备：IPv6 支持

One of the most important networking topics today is the impending exhaustion of IPv4 addresses. In early 2014, ICANN announced that they started allocating the final IPv4 address blocks to the Regional Internet Registries (http://www.internetsociety.org/deploy360/blog/2014/05/goodbye-ipv4-iana-starts-allocating-final-address-blocks/). This means the IPv4 address space is close to being fully allocated. As a result, it will soon become difficult to allocate more IPv4 addresses to an application that has experienced growth, or that you expect to scale out, due to the lack of unallocated IPv4 address blocks.

对于关注网络方面的应用来说，未来将是 IPv6 协议的天下。IPv6 显著地扩大了地址空间，解决了 IPv4 协议长久以来存在的问题，并且将会成为未来面向网络应用的重要乃至本质部分。

OpenStack Networking supports IPv6 when configured to take advantage of it. To enable IPv6, create an IPv6 subnet in Networking and use IPv6 prefixes when creating security groups.

非对称连接

When designing a network architecture, the traffic patterns of an application heavily influence the allocation of total bandwidth and the number of links that you use to send and receive traffic. Applications that provide file storage for customers allocate bandwidth and links to favor incoming traffic, whereas video streaming applications allocate bandwidth and links to favor outgoing traffic.

性能

It is important to analyze the applications' tolerance for latency and jitter when designing an environment to support network focused applications. Certain applications, for example VoIP, are less tolerant of latency and jitter. Where latency and jitter are concerned, certain applications may require tuning of QoS parameters and network device queues to ensure that they queue for transmit immediately or guarantee minimum bandwidth. Since OpenStack currently does not support these functions, consider carefully your selected network plug-in.

The location of a service may also impact the application or consumer experience. If an application serves differing content to different users it must properly direct connections to those specific locations. Where appropriate, use a multi-site installation for these situations.

You can implement networking in two separate ways. Legacy networking (nova-network) provides a flat DHCP network with a single broadcast domain. This implementation does not support tenant isolation networks or advanced plug-ins, but it is currently the only way to implement a distributed layer-3 agent using the multi_host configuration. OpenStack Networking (neutron) is the official networking implementation and provides a pluggable architecture that supports a large variety of network methods. Some of these include a layer-2 only provider network model, external device plug-ins, or even OpenFlow controllers.

Networking at large scales becomes a set of boundary questions. The determination of how large a layer-2 domain must be is based on the amount of nodes within the domain and the amount of broadcast traffic that passes between instances. Breaking layer-2 boundaries may require the implementation of overlay networks and tunnels. This decision is a balancing act between the need for a smaller overhead or a need for a smaller domain.

选择网络设备时，必须意识到基于最大的端口密度所做出的决定通常也有一个弊端。聚合交换以及路由并不能完全满足柜顶交换机的需要，这可能

引起南北向流量上的瓶颈。其结果是，大量的下行网络使用可能影响上行网络设备，从而影响云中的服务。由于 OpenStack 目前并未提供流量整形或者速度限制的机制，有必要在网络硬件的级别上实现这些特性。

运营因素

Network-focused OpenStack clouds have a number of operational considerations that influence the selected design, including:

- Dynamic routing of static routes
- Service level agreements (SLAs)
- Ownership of user management

An initial network consideration is the selection of a telecom company or transit provider.

Make additional design decisions about monitoring and alarming. This can be an internal responsibility or the responsibility of the external provider. In the case of using an external provider, service level agreements (SLAs) likely apply. In addition, other operational considerations such as bandwidth, latency, and jitter can be part of an SLA.

Consider the ability to upgrade the infrastructure. As demand for network resources increase, operators add additional IP address blocks and add additional bandwidth capacity. In addition, consider managing hardware and software life cycle events, for example upgrades, decommissioning, and outages, while avoiding service interruptions for tenants.

Factor maintainability into the overall network design. This includes the ability to manage and maintain IP addresses as well as the use of overlay identifiers including VLAN tag IDs, GRE tunnel IDs, and MPLS tags. As an example, if you may need to change all of the IP addresses on a network, a process known as renumbering, then the design must support this function.

Address network-focused applications when considering certain operational realities. For example, consider the impending exhaustion of IPv4 addresses, the migration to IPv6, and the use of private networks to segregate different types of traffic that an application receives or generates. In the case of IPv4 to IPv6 migrations, applications should follow best practices for storing IP addresses. We recommend you avoid

relying on IPv4 features that did not carry over to the IPv6 protocol or have differences in implementation.

To segregate traffic, allow applications to create a private tenant network for database and storage network traffic. Use a public network for services that require direct client access from the internet. Upon segregating the traffic, consider quality of service (QoS) and security to ensure each network has the required level of service.

Finally, consider the routing of network traffic. For some applications, develop a complex policy framework for routing. To create a routing policy that satisfies business requirements, consider the economic cost of transmitting traffic over expensive links versus cheaper links, in addition to bandwidth, latency, and jitter requirements.

Additionally, consider how to respond to network events. As an example, how load transfers from one link to another during a failure scenario could be a factor in the design. If you do not plan network capacity correctly, failover traffic could overwhelm other ports or network links and create a cascading failure scenario. In this case, traffic that fails over to one link overwhelms that link and then moves to the subsequent links until all network traffic stops.

架构

Network-focused OpenStack architectures have many similarities to other OpenStack architecture use cases. There are several factors to consider when designing for a network-centric or network-heavy application environment.

Networks exist to serve as a medium of transporting data between systems. It is inevitable that an OpenStack design has inter-dependencies with non-network portions of OpenStack as well as on external systems. Depending on the specific workload, there may be major interactions with storage systems both within and external to the OpenStack environment. For example, in the case of content delivery network, there is twofold interaction with storage. Traffic flows to and from the storage array for ingesting and serving content in a north-south direction. In addition, there is replication traffic flowing in an east-west direction.

Compute-heavy workloads may also induce interactions with the network. Some high performance compute applications require network-based memory mapping and data sharing and, as a result, induce a higher network

load when they transfer results and data sets. Others may be highly transactional and issue transaction locks, perform their functions, and revoke transaction locks at high rates. This also has an impact on the network performance.

Some network dependencies are external to OpenStack. While OpenStack Networking is capable of providing network ports, IP addresses, some level of routing, and overlay networks, there are some other functions that it cannot provide. For many of these, you may require external systems or equipment to fill in the functional gaps. Hardware load balancers are an example of equipment that may be necessary to distribute workloads or offload certain functions. As of the Icehouse release, dynamic routing is currently in its infancy within OpenStack and you may require an external device or a specialized service instance within OpenStack to implement it. OpenStack Networking provides a tunneling feature, however it is constrained to a Networking-managed region. If the need arises to extend a tunnel beyond the OpenStack region to either another region or an external system, implement the tunnel itself outside OpenStack or use a tunnel management system to map the tunnel or overlay to an external tunnel. OpenStack does not currently provide quotas for network resources. Where network quotas are required, implement quality of service management outside of OpenStack. In many of these instances, similar solutions for traffic shaping or other network functions are needed.

Depending on the selected design, Networking itself might not support the required layer-3 network functionality. If you choose to use the provider networking mode without running the layer-3 agent, you must install an external router to provide layer-3 connectivity to outside systems.

Interaction with orchestration services is inevitable in larger-scale deployments. The Orchestration module is capable of allocating network resource defined in templates to map to tenant networks and for port creation, as well as allocating floating IPs. If there is a requirement to define and manage network resources when using orchestration, we recommend that the design include the Orchestration module to meet the demands of users.

对设计的影响

A wide variety of factors can affect a network-focused OpenStack architecture. While there are some considerations shared with a general use case, specific workloads related to network requirements influence network design decisions.

One decision includes whether or not to use Network Address Translation (NAT) and where to implement it. If there is a requirement for floating IPs instead of public fixed addresses then you must use NAT. An example of this is a DHCP relay that must know the IP of the DHCP server. In these cases it is easier to automate the infrastructure to apply the target IP to a new instance rather than to reconfigure legacy or external systems for each new instance.

NAT for floating IPs managed by Networking resides within the hypervisor but there are also versions of NAT that may be running elsewhere. If there is a shortage of IPv4 addresses there are two common methods to mitigate this externally to OpenStack. The first is to run a load balancer either within OpenStack as an instance, or use an external load balancing solution. In the internal scenario, Networking's Load-Balancer-as-a-Service (LBaaS) can manage load balancing software, for example HAproxy. This is specifically to manage the Virtual IP (VIP) while a dual-homed connection from the HAproxy instance connects the public network with the tenant private network that hosts all of the content servers. In the external scenario, a load balancer needs to serve the VIP and also connect to the tenant overlay network through external means or through private addresses.

另外一种可能有用的 NAT 是协议 NAT。某些情况下，可能需要在实例中只使用 IPv6 地址，然后让一个实例或者外部的服务来提供基于 NAT 的转换技术，比如说 NAT64 和 DNS64。这使得实例都有全局可达的 IPv6 地址，同时只在必要的情况下，或者以共享的方式使用 IPv4 地址。

Application workloads affect the design of the underlying network architecture. If a workload requires network-level redundancy, the routing and switching architecture have to accommodate this. There are differing methods for providing this that are dependent on the selected network hardware, the performance of the hardware, and which networking model you deploy. Examples include Link aggregation (LAG) and Hot Standby Router Protocol (HSRP). Also consider whether to deploy OpenStack Networking or legacy networking (nova-network), and which plug-in to select for OpenStack Networking. If using an external system, configure Networking to run layer 2 with a provider network configuration. For example, implement HSRP to terminate layer-3 connectivity.

Depending on the workload, overlay networks may not be the best solution. Where application network connections are small, short lived, or bursty, running a dynamic overlay can generate as much bandwidth as the packets it carries. It also can induce enough latency to cause issues with certain applications. There is an impact to the device generating the overlay

which, in most installations, is the hypervisor. This causes performance degradation on packet per second and connection per second rates.

Overlays also come with a secondary option that may not be appropriate to a specific workload. While all of them operate in full mesh by default, there might be good reasons to disable this function because it may cause excessive overhead for some workloads. Conversely, other workloads operate without issue. For example, most web services applications do not have major issues with a full mesh overlay network, while some network monitoring tools or storage replication workloads have performance issues with throughput or excessive broadcast traffic.

Many people overlook an important design decision: The choice of layer-3 protocols. While OpenStack was initially built with only IPv4 support, Networking now supports IPv6 and dual-stacked networks. As of the Icehouse release, this only includes stateless address auto configuration but work is in progress to support stateless and stateful DHCPv6 as well as IPv6 floating IPs without NAT. Some workloads are possible through the use of IPv6 and IPv6 to IPv4 reverse transition mechanisms such as NAT64 and DNS64 or 6to4. This alters the requirements for any address plan as single-stacked and transitional IPv6 deployments can alleviate the need for IPv4 addresses.

As of the Icehouse release, OpenStack has limited support for dynamic routing, however there are a number of options available by incorporating third party solutions to implement routing within the cloud including network equipment, hardware nodes, and instances. Some workloads perform well with nothing more than static routes and default gateways configured at the layer-3 termination point. In most cases this is sufficient, however some cases require the addition of at least one type of dynamic routing protocol if not multiple protocols. Having a form of interior gateway protocol (IGP) available to the instances inside an OpenStack installation opens up the possibility of use cases for anycast route injection for services that need to use it as a geographic location or failover mechanism. Other applications may wish to directly participate in a routing protocol, either as a passive observer, as in the case of a looking glass, or as an active participant in the form of a route reflector. Since an instance might have a large amount of compute and memory resources, it is trivial to hold an entire unpartitioned routing table and use it to provide services such as network path visibility to other applications or as a monitoring tool.

Path maximum transmission unit (MTU) failures are lesser known but harder to diagnose. The MTU must be large enough to handle normal traffic, overhead from an overlay network, and the desired layer-3 protocol.

Adding externally built tunnels reduces the MTU packet size. In this case, you must pay attention to the fully calculated MTU size because some systems ignore or drop path MTU discovery packets.

可调联网组件

Consider configurable networking components related to an OpenStack architecture design when designing for network intensive workloads that include MTU and QoS. Some workloads require a larger MTU than normal due to the transfer of large blocks of data. When providing network service for applications such as video streaming or storage replication, we recommend that you configure both OpenStack hardware nodes and the supporting network equipment for jumbo frames where possible. This allows for better use of available bandwidth. Configure jumbo frames across the complete path the packets traverse. If one network component is not capable of handling jumbo frames then the entire path reverts to the default MTU.

Quality of Service (QoS) also has a great impact on network intensive workloads as it provides instant service to packets which have a higher priority due to the impact of poor network performance. In applications such as Voice over IP (VoIP), differentiated services code points are a near requirement for proper operation. You can also use QoS in the opposite direction for mixed workloads to prevent low priority but high bandwidth applications, for example backup services, video conferencing, or file sharing, from blocking bandwidth that is needed for the proper operation of other workloads. It is possible to tag file storage traffic as a lower class, such as best effort or scavenger, to allow the higher priority traffic through. In cases where regions within a cloud might be geographically distributed it may also be necessary to plan accordingly to implement WAN optimization to combat latency or packet loss.

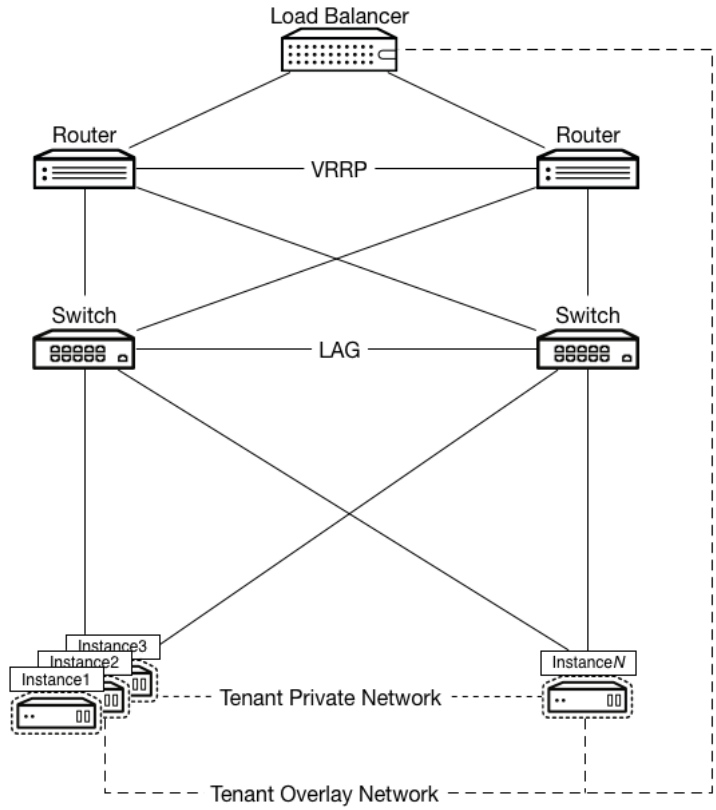
示例

An organization design a large-scale web application with cloud principles in mind. The application scales horizontally in a bursting fashion and generates a high instance count. The application requires an SSL connection to secure data and must not lose connection state to individual servers.

The figure below depicts an example design for this workload. In this example, a hardware load balancer provides SSL offload functionality and connects to tenant networks in order to reduce address consumption. This load balancer links to the routing architecture as it services the VIP for

the application. The router and load balancer use the GRE tunnel ID of the application's tenant network and an IP address within the tenant subnet but outside of the address pool. This is to ensure that the load balancer can communicate with the application's HTTP servers without requiring the consumption of a public IP address.

Because sessions persist until closed, the routing and switching architecture provides high availability. Switches mesh to each hypervisor and each other, and also provide an MLAG implementation to ensure that layer-2 connectivity does not fail. Routers use VRRP and fully mesh with switches to ensure layer-3 connectivity. Since GRE is provides an overlay network, Networking is present and uses the Open vSwitch agent in GRE tunnel mode. This ensures all devices can reach all other devices and that you can create tenant networks for private addressing links to the load balancer.



A web service architecture has many options and optional components. Due to this, it can fit into a large number of other OpenStack designs. A few

key components, however, need to be in place to handle the nature of most web-scale workloads. You require the following components:

- OpenStack 控制服务(镜像服务、认证服务、网络服务以及例如 MariaDB 和 RabbitMQ 之类的支撑服务)
- 运行着 KVM 宿主机的 OpenStack 计算服务
- OpenStack 对象存储
- 编排模块
- 计量模块

Beyond the normal Identity, Compute, Image service, and Object Storage components, we recommend the Orchestration module component to handle the proper scaling of workloads to adjust to demand. Due to the requirement for auto-scaling, the design includes the Telemetry module. Web services tend to be bursty in load, have very defined peak and valley usage patterns and, as a result, benefit from automatic scaling of instances based upon traffic. At a network level, a split network configuration works well with databases residing on private tenant networks since these do not emit a large quantity of broadcast traffic and may need to interconnect to some databases for content.

负载均衡

Load balancing spreads requests across multiple instances. This workload scales well horizontally across large numbers of instances. This enables instances to run without publicly routed IP addresses and instead to rely on the load balancer to provide a globally reachable service. Many of these services do not require direct server return. This aids in address planning and utilization at scale since only the virtual IP (VIP) must be public.

覆盖网络

The overlay functionality design includes OpenStack Networking in Open vSwitch GRE tunnel mode. In this case, the layer-3 external routers pair with VRRP, and switches pair with an implementation of MLAG to ensure that you do not lose connectivity with the upstream routing infrastructure.

性能调优

Network level tuning for this workload is minimal. Quality-of-Service (QoS) applies to these workloads for a middle ground Class Selector depending on existing policies. It is higher than a best effort queue but lower than an Expedited Forwarding or Assured Forwarding queue. Since this type of application generates larger packets with longer-lived connections, you can optimize bandwidth utilization for long duration TCP. Normal bandwidth planning applies here with regards to benchmarking a session's usage multiplied by the expected number of concurrent sessions with overhead.

网络功能

网络功能的话题比较宽泛，但通常是指围绕在为系统的其他部分的网络提供支持的目的的工作。这类网络负载通常都是由大量的存活期比较短的小网络包组成的，例如 DNS 请求或者 SNMP 陷阱等。这类消息需要很快地到达并且由于可能非常大量而不关注丢包的问题。这类型的负载还有一些额外的考虑因素需要顾及，并且可能改变直到宿主机级别的网络配置。假设一个应用为每个用户生成 10 个 TCP 会话，每个数据流的速度平均是 512 Kbps，而预期用户的数量是 1 万个用户同时使用，预期总计的带宽计划将达到大约 4.88 Gbps。

The supporting network for this type of configuration needs to have a low latency and evenly distributed availability. This workload benefits from having services local to the consumers of the service. Use a multi-site approach as well as deploying many copies of the application to handle load as close as possible to consumers. Since these applications function independently, they do not warrant running overlays to interconnect tenant networks. Overlays also have the drawback of performing poorly with rapid flow setup and may incur too much overhead with large quantities of small packets and therefore we do not recommend them.

QoS is desirable for some workloads to ensure delivery. DNS has a major impact on the load times of other services and needs to be reliable and provide rapid responses. Configure rules in upstream devices to apply a higher Class Selector to DNS to ensure faster delivery or a better spot in queuing algorithms.

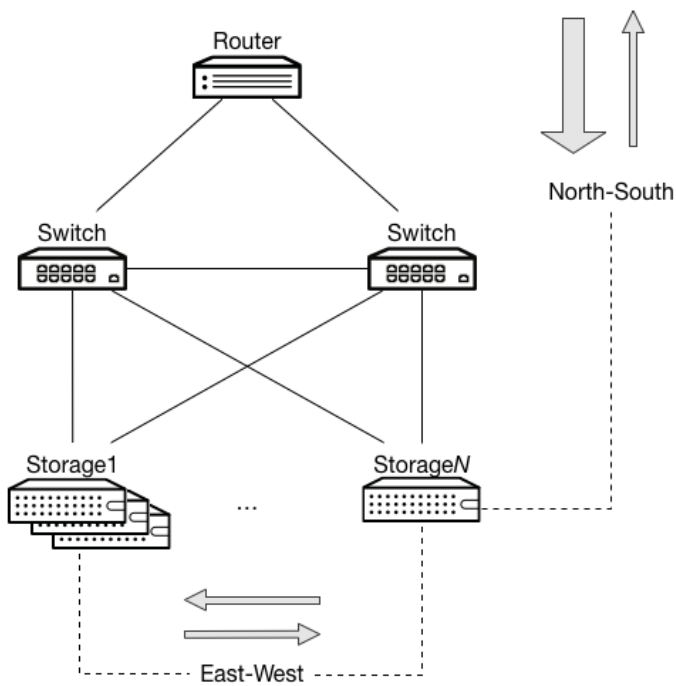
云存储

Another common use case for OpenStack environments is providing a cloud-based file storage and sharing service. You might consider this a storage-focused use case, but its network-side requirements make it a network-focused use case.

For example, consider a cloud backup application. This workload has two specific behaviors that impact the network. Because this workload is an externally-facing service and an internally-replicating application, it has both north-south and east-west traffic considerations:

南北向流量 When a user uploads and stores content, that content moves into the OpenStack installation. When users download this content, the content moves out from the OpenStack installation. Because this service operates primarily as a backup, most of the traffic moves southbound into the environment. In this situation, it benefits you to configure a network to be asymmetrically downstream because the traffic that enters the OpenStack installation is greater than the traffic that leaves the installation.

东西向流量 很可能是完全对称的网络。因为根据算法来说，复制行为可能从任何节点发起并指向多个其他节点，某个特定方向的流量更大的情况不大可能发生。然而，这些流量也可能干扰到南北向的流量。



此应用将南北向的流量的优先级设置得比东西向的流量的优先级更高：南北向的流量与面向客户的数据相关。

The network design in this case is less dependent on availability and more dependent on being able to handle high bandwidth. As a direct result, it is beneficial to forgo redundant links in favor of bonding those connections. This increases available bandwidth. It is also beneficial to configure all devices in the path, including OpenStack, to generate and pass jumbo frames.

第 6 章 多区域

目录

一个多区域openstack环境是指它的服务分布在多个数据中心来提供整体的服务.不同的多区域云可能在使用要求上区别会很大,然而它们还是有一些共同点的.openstack可以运行在多区域环境下,允许某部分服务有效地像管理一个单区域的云一样来管理着由多个区域组成的群组.通过在设计阶段仔细计划,面对不同的需求,openstack可以是一个多区域云的完美解决方案

一些应该使用多区域布署的案例可能会有以下特征:

- 一个有不同地域认证的组织
- 地理位置敏感数据
- 数据所在地应该接近用户,特别是一些特殊数据或者功能

用户需求

一个多区域架构是复杂的而且尤其自身的风险和考虑，因此确保考虑此架构的设计应满足用户和业务的需求显得异常重要。

很多辖区对于云环境中的数据的保管及管理都有相关的法律上或者监管上的要求。这些规章的常见领域包括：

- 确保持久化数据的保管和记录管理以符合数据档案化需求的数据保留政策。
- 管理数据的所有权和责任的数据所有权政策。
- 管理位于外国或者其它辖区的数据存储问题的数据独立性政策。
- 数据合规性,管理由于监管上的问题因而特定类型的数据必须存放在特定的地点或者更重要的，由于相同的原因，不能够存放在其它地点的数据管理政策。

Examples of such legal frameworks include the data protection framework of the European Union (http://ec.europa.eu/justice/data-protection) and the requirements of the Financial Industry Regulatory Authority (http://www.finra.org/Industry/Regulation/FINRARules) in the United States. Consult a local regulatory body for more information.

负载特性

预期的工作负荷是需要被捕获到指导决策的关键要求。理解负载在预想的多区域环境和用例上下文很重要。另外一个考虑负载的路径是想它怎么使用系统。一个负载可以是单个的应用或一组共同工作的应用。它也可以是在多个region中运行的多份应用的复制。通常在多区域部署的相同负载需要在多个不同物理地点运行相同的工作。

此多区域场景在本书中有一个或多个其他类似的场景，另外在两个或多个地点的额外负载需求。下面是可能一样的场景：

对于很多案例来说用户的负载会直接影响到其应用的性能，因此需要将此纳入考虑范围。欲确保应用的延迟为零或尽可能的最小化，多地点唯一的实现就是在部署云时。这些地点可以是不同的数据中心，不同的城市，不同的国家或地理位置，取决于用户需求和当地用户。

镜像和模板在跨不同站点时要保持一致性。

It is essential that the deployment of instances is consistent across the different sites. This needs to be built into the infrastructure. If the OpenStack Object Storage is used as a back end for the Image service, it is possible to create repositories of consistent images across multiple sites. Having central endpoints with multiple storage nodes allows consistent centralized storage for each and every site.

Not using a centralized object store increases operational overhead so that a consistent image library can be maintained. This could include development of a replication mechanism to handle the transport of images and the changes to the images across multiple sites.

高可用

如果高可用是提供持续基础设施运营的需求，那么高可用的基本需求就应该被定义。

OpenStack管理组件需要哪怕是最基本的或最小级别的冗余，举个最简单的例子，这样当任何一个站点失效时而不会受到显著的影响，因为有OpenStack服务依然可用。

The OpenStack High Availability Guide contains more information on how to provide redundancy for the OpenStack components.

多网络链路应该在站点之间部署，以提供所有组件的冗余。这其中包括存储的复制，存储应该使用专用网络隔离，或者使用VLAN的QoS功能来控制复制流量，或者为该流量提供高的优先级。记住，如果数据存储经常改动，网络的需求就会显著影响到维护站点的运维成本。

The ability to maintain object availability in both sites has significant implications on the object storage design and implementation. It also has a significant impact on the WAN network design between the sites.

连接多于两个站点会增加挑战，会给设计考虑增加更多的复杂性。多站点实现需要额外的规划，增加交付内外部连通性的复杂拓扑，一些选项包括完全mesh拓扑，hub spoke,spine leaf,或3d Torus。

不是所有运行在云中的应用都知道自己运行在云中。如果有这样的情况存在，应该清晰的去衡量和定义什么样的基础设施可以支持，但是更加重要的是什么样的基础设施不支持。举个例子，在站点之间支持共享存储。它是可能的，尽管OpenStack本身不支持这样的解决方案，需要第三方的硬件供应商来弥补这一需求。在举一个例子，应用可以直接消费对象存储，这些应用需要自己知道在使用OpenStack对象存储。

应用准备

一些应用可以容忍对象存储没有同步，但是也有另外的应用需要这些对象被复制且跨多个region可用。理解云如何实现会影响到新的和已有的应用，对于降低风险和整个云项目成功很重要。已经完成的应用期望基础设施并无冗余，已有的应用没有将运行在云中考虑的话就需要重写了。

成本

多个站点的需求带来更多的额外开销。更多的站点，意味着更多的开销和复杂性。开销可以细分为以下几类：

- 计算资源
- 网络资源
- 重复
- 存储
- 管理

- 运营成本

站点失效和恢复

中断会引起站点的部分或全部功能的尚失。理解并实现恢复策略，且要规划恢复方案。

- 已经部署的应用需要持续的服务，且更加重要的，需要考虑由于站点的不可用带来的性能冲击和应用的可靠性。
- It is important to understand what happens to the replication of objects and data between the sites when a site goes down. If this causes queues to start building up, consider how long these queues can safely exist until something explodes.
- 确保在灾难发生之后，当业务恢复正常之前的这段时间内，恢复站点的正确操作的几种办法要决定下来。我们建议用户恢复的架构要避免竞争状态。

合规性和地理位置

一个组织须有一定的法律义务和法规遵从来衡量什么是需要的负载或数据能否存放在指定的地区。

审计

为了能够快速的追查问题，一个经过好的经过深思熟虑的审计策略是非常重要的。对于安全组和租户的变动保持跟踪，在生产环境中可用于回滚，例如，如果一个租户的安全组规则消失了，能够快速追查问题的能力对于运维来说很重要。

职责分工

一个常见的需求是为不同的云管理功能定义不同的角色。此例即是站点需要合并职责和权限的需求。

站点之间的认证

Ideally it is best to have a single authentication domain and not need a separate implementation for each and every site. This, of course, requires an authentication mechanism that is highly available and distributed to ensure continuous operation. Authentication server locality is also something that might be needed as well and should be planned for.

技术因素

There are many technical considerations to take into account with regard to designing a multi-site OpenStack implementation. An OpenStack cloud can be designed in a variety of ways to handle individual application needs. A multi-site deployment has additional challenges compared to single site installations and therefore is a more complex solution.

当决定容量要算计的不仅仅是技术问题，还要考虑经济和运营问题，这些都可能会带来更多麻烦。

Inter-site link capacity describes the capabilities of the connectivity between the different OpenStack sites. This includes parameters such as bandwidth, latency, whether or not a link is dedicated, and any business policies applied to the connection. The capability and number of the links between sites determine what kind of options are available for deployment. For example, if two sites have a pair of high-bandwidth links available between them, it may be wise to configure a separate storage replication network between the two sites to support a single Swift endpoint and a shared object storage capability between them. (An example of this technique, as well as a configuration walk-through, is available at http://docs.openstack.org/developer/swift/replication_network.html#dedicated-replication-network). Another option in this scenario is to build a dedicated set of tenant private networks across the secondary link using overlay networks with a third party mapping the site overlays to each other.

The capacity requirements of the links between sites is driven by application behavior. If the latency of the links is too high, certain applications that use a large number of small packets, for example RPC calls, may encounter issues communicating with each other or operating properly. Additionally, OpenStack may encounter similar types of issues. To mitigate this, tuning of the Identity service call timeouts may be necessary to prevent issues authenticating against a central Identity service.

Another capacity consideration when it comes to networking for a multi-site deployment is the available amount and performance of overlay networks for tenant networks. If using shared tenant networks across zones, it is imperative that an external overlay manager or controller be used to map these overlays together. It is necessary to ensure the amount of possible IDs between the zones are identical. Note that, as of the Kilo release, OpenStack Networking was not capable of managing tunnel

IDs across installations. This means that if one site runs out of IDs, but other does not, that tenant's network is unable to reach the other site.

容量还有其它形式的表现，region增长的能力依赖于横向扩展更多可用的计算节点。此话题在章节计算型中会谈到更多的细节。总之，务必记得也许需要为各个region增加cell以超过一定的点，此点又依赖于集群的大小和每hypervisor对虚拟机的比例。

第三种形式的容量表现在multi-region-capable 的OpenStack组件，中心化的对象存储在跨多region通过单一的命名空间能够服务对象。此工作方式由swift代理来访问对象信息，因此是有可能代理过度负载。有两种方法可减低此问题，第一，部署大量的swift代理，此方法的缺陷在于代理没有负载均衡，大量的文件请求会访问同一个proxy。第二种降低负载的办法是在代理的前端使用HTTP代理缓存和负载均衡器。因此swift对象会通过HTTP来响应请求，此负载均衡器能缓和swift代理的负载请求。

量力而行

构建一个多区域OpenStack环境是本书的目的，但真正的考验来自于能否有一个应用能够利用好。

身份是多数OpenStack用户的第一道“门槛”，对于绝大多数OpenStack的主要操作都有和身份服务互动的需求，因此，确保你为用户提供身份认证服务单一的URL非常重要，同样重要的是身份服务的正确文档和配置region。在你安装时考虑好每个站点定义的region的身份命名空间，这对于系统管理很重要，当阅读身份文档时，它会要求当在API端点或GUI界面执行某些操作时所定义的region名称。

Load balancing is another common issue with multi-site installations. While it is still possible to run HAproxy instances with Load-Balancer-as-a-Service, these are local to a specific region. Some applications may be able to cope with this via internal mechanisms. Others, however, may require the implementation of an external system including global services load balancers or anycast-advertised DNS.

Depending on the storage model chosen during site design, storage replication and availability are also a concern for end-users. If an application is capable of understanding regions, then it is possible to keep the object storage system separated by region. In this case, users who want to have an object available to more than one region need to do the cross-site replication themselves. With a centralized swift proxy, however, the user may need to benchmark the replication timing of the Object Storage back end. Benchmarking allows the operational staff to provide users with an understanding of the amount of time required for a stored or modified object to become available to the entire environment.

性能

决定多区域安装的性能所包含的考虑因素和在单站点部署是不一样的，作为分布式部署，多区域部署在一些场景中会遭遇一些额外的性能瓶颈。

尽管多区域系统可以基于地理位置分离，它们会在跨region通信时遭遇糟糕的延迟和抖动。这种情况尤其影响系统的如OpenStack身份服务从region做认证尝试时没有实现中心化的身份服务。它也会影响到一些应用如远程过程调用(RPC)的正常操作，在高性能计算负载型中此类问题很常见。

Storage availability can also be impacted by the architecture of a multi-site deployment. A centralized Object Storage service requires more time for an object to be available to instances locally in regions where the object was not created. Some applications may need to be tuned to account for this effect. Block Storage does not currently have a method for replicating data across multiple regions, so applications that depend on available block storage need to manually cope with this limitation by creating duplicate block storage entries in each region.

安全性

Securing a multi-site OpenStack installation also brings extra challenges. Tenants may expect a tenant-created network to be secure. In a multi-site installation the use of a non-private connection between sites may be required. This may mean that traffic would be visible to third parties and, in cases where an application requires security, this issue requires mitigation. Installing a VPN or encrypted connection between sites is recommended in such instances.

另外的多区域部署安全需要考虑的是身份，在多区域云中认证服务需要中心化，所谓的中心化就是在部署中为用户提供单一的认证点，以及管理这个点的创建、读取、更新和删除的原有操作。中心化的认证也对审计目的有用，因为所有的认证令牌都来自同一个源。

就像在单站点部署的租户需要彼此隔离，在多区域安装中的租户也一样，在多区域设计的额外挑战是围绕着如何确保跨region的租户网络功能，不幸的是，OpenStack网络所提供的功能还不支持此种机制，因为需要外部的系统来管理这些映射，租户网络包含的敏感信息需要这种映射的精准和一致，以确保在一个站点的租户不会联系到另外一个站点的不同租户。

OpenStack 组件

Most OpenStack installations require a bare minimum set of pieces to function. These include the OpenStack Identity (keystone) for

authentication, OpenStack Compute (nova) for compute, OpenStack Image service (glance) for image storage, OpenStack Networking (neutron) for networking, and potentially an object store in the form of OpenStack Object Storage (swift). Bringing multi-site into play also demands extra components in order to coordinate between regions. Centralized Identity service is necessary to provide the single authentication point. Centralized dashboard is also recommended to provide a single login point and a mapped experience to the API and CLI options available. If necessary, a centralized Object Storage service may be used and will require the installation of the swift proxy service.

一些额外的选项安装会帮助改进一些场景，对于实例来说，安装Designate为每个region自动生成DNS域，为每个实例自动记录zone所有的资源信息，这种使用DNS机制会改进决策，为确定的应用选择那个region。

另外一个管理多区域安装的工具是Orchestration(heat)。Orchestration模块允许使用模板来定义一组实例，来一起启动或者是扩展已有的实例。它也可用于基于region的匹配设置或比较各组的不同。对于实例来说，如果一个应用要求在跨站点平均负载到多个节点，相同的heat模板可用于覆盖到每个站点，改动很小仅仅是一个region的名称。

运营因素

使用region来部署一个多区域OpenStack云，要求每个部署的服务的服务目录包含per-region元素，不仅仅是认证服务本身。当前现成的OpenStack部署工具中，对于定义多region的支持还很有限。

部署者务必意识到这点，为他们的站点提供合适的定制服务目录，无论是手动还是定制，都的使用这些部署工具。

注意

As of the Kilo release, documentation for implementing this feature is in progress. See this bug for more information: https://bugs.launchpad.net/openstack-manuals/+bug/1340509.

许可

多区域OpenStack部署需要考虑常规OpenStack云之外的许可，尤其是那些正在使用中的，这样具有成本效益。这些许可有主机操作系统的，客户机操作系统的，OpenStack发行版(如果购买过)，软件定义基础设施包括网络控制器和存储系统的，以及各种应用，都需要评估，基于多区域云的性质本身出发。

考虑的主题有：

- 在相关的许可证下一个站点由什么构成有着特别的定义，正如该术语不一定表示在传统意义上的一个地理或其他物理隔离的位置。
- “热”(活动)和”冷”(非活动)站点之间的区别在于，冷的预备站点只是用于灾难恢复的情况发生。
- 为应对不同站点的挑战，某些地点也许需要当地供应商的支持和服务，但是都得遵循当地的许可协议。

记录日志和监测

Logging and monitoring does not significantly differ for a multi-site OpenStack cloud. The same well known tools described in the Logging and monitoring chapter of the Operations Guide remain applicable. Logging and monitoring can be provided both on a per-site basis and in a common centralized location.

当尝试将日志和监测系统部署到一个中心位置时，要小心内部网络的负载。

升级

In multi-site OpenStack clouds deployed using regions each site is, effectively, an independent OpenStack installation which is linked to the others by using centralized services such as Identity which are shared between sites. At a high level the recommended order of operations to upgrade an individual OpenStack environment is (see the Upgrades chapter of the Operations Guide for details):

1. 升级OpenStack认证服务 (keystone).
2. Upgrade the OpenStack Image service (glance).
3. 升级 OpenStack 计算 (nova), 包括网络组件。
4. 升级 OpenStack 块存储 (cinder)。
5. 升级 OpenStack GUI程序 (horizon)。

升级多区域环境的流程没有什么特别的不一样：

1. 升级 OpenStack 认证共享服务 (keystone) 。
2. Upgrade the OpenStack Image service (glance) at each site.
3. 在每个区域升级OpenStack计算(nova),包括网络组件。
4. 在每个区域升级OpenStack块存储 (cinder)。

5. 在每个区域升级OpenStack GUI程序(horizon),或者假如它是共享的话，仅升级单个的数据中心即可。

请记住，在OpenStack Icehouse发布后，为每个站点升级计算组件可以是回滚的方式进行。计算控制器服务(API,调度器以及Conductor)可以为单独的某个计算节点升级而不会影响现有的服务。这最大化的为运维人员在进行升级时不会中断用户使用计算服务。

配额管理

为防止系统资源被耗尽而没有任何通知，OpenStack提供给运维人员定义配额的能力。配额用户设置操作限制，当前可在租户(或项目)中实现了强制，而用户级别则没有实现。

配置被定义为每个region的基本要素之一。运维人员也许希望定义相同的配额，在每个region下的租户，以提供给用户一致的体验，或者创建一个跨region同步分配配额的流程。但是请记住下面一点很重要，配额不会跨region进行限制。

举例来说，一个云有两个region，如果运维人员给某个用户的配额是在每个region中可以启动25个实例，那么此用户在两个region加起来就可以启动50个实例。但是不可以这么相加，因为配额只在一个region生效。

For more information on managing quotas refer to the Managing projects and users chapter of the OpenStack Operators Guide.

规则管理

OpenStack默认提供的是基于角色访问控制(RBAC)规则，在每个服务中，在文件policy.json定义。运维人员可以编辑此文件来定制规则。如果跨区域被认为有需要将RBAC规则一致处理的话，那么就有必要确保在所有的安装配置中将文件policy.json保持同步。

这个可以用普通的系统管理工具如rsync来完成，OpenStack目前没有提供跨区域的同步规则。

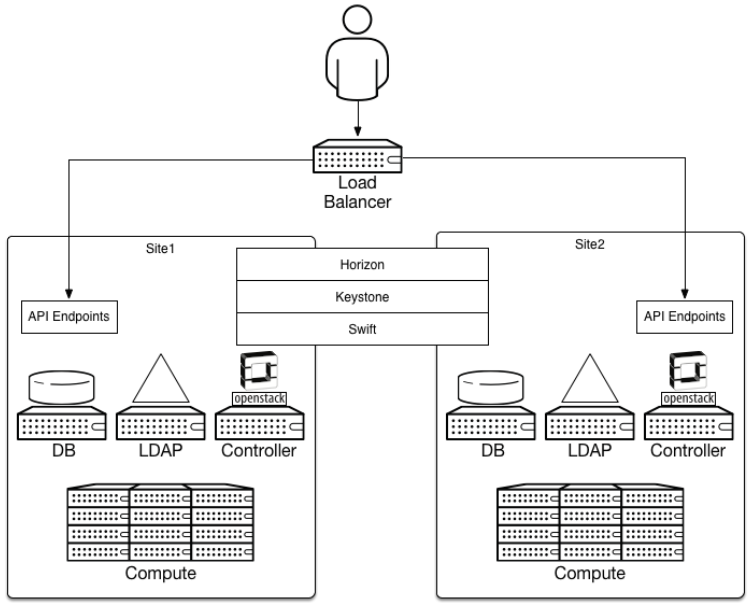
文档

Users must be able to leverage cloud infrastructure and provision new resources in the environment. It is important that user documentation is accessible by users of the cloud infrastructure to ensure they are given sufficient information to help them leverage the cloud. As an example, by default OpenStack schedules instances on a compute node automatically. However, when multiple regions are available, it is left to the end user to decide in which region to schedule the new instance. The dashboard

presents the user with the first region in your configuration. The API and CLI tools do not execute commands unless a valid region is specified. It is therefore important to provide documentation to your users describing the region layout as well as calling out that quotas are region-specific. If a user reaches his or her quota in one region, OpenStack does not automatically build new instances in another. Documenting specific examples helps users understand how to operate the cloud, thereby reducing calls and tickets filed with the help desk.

架构

This graphic is a high level diagram of a multi-site OpenStack architecture. Each site is an OpenStack cloud but it may be necessary to architect the sites on different versions. For example, if the second site is intended to be a replacement for the first site, they would be different. Another common design would be a private OpenStack cloud with replicated site that would be used for high availability or disaster recovery. The most important design decision is how to configure the storage. It can be configured as a single shared pool or separate pools, depending on the user and technical requirements.



OpenStack服务架构

OpenStack认证服务，用于所有其他OpenStack组件的验证，服务端点的目录，支持所有region。一个region是用于一组OpenStack彼此接近的

服务的逻辑构造。region的概念颇为灵活，它可以包含地理位置很远的OpenStack服务端点，也可以是很小的范围，region甚至可以是一个数据中心的机柜，或是一个刀片中心，甚至在同一个数据相邻的机柜都拥有多个region。

多数的OpenStack组件被设计为运行在单个region上下文中。OpenStack计算服务被设计为在一个region中管理计算资源，支持使用可用区域和单元来往下拆分。OpenStack网络服务可用于在同一广播域中或者说相互连接的交换机来管理网路资源。OpenStack块存储服务控制单一region中的存储资源，且这些资源须在统一存储网络中。和OpenStack计算服务一样，OpenStack块存储服务同样支持可用区域的构造，用于往下拆分存储资源。

OpenStack GUI，OpenStack认证，以及OpenStack对象存储等服务为了服务于多区域，这些组件需要部署在中心化位置。

存储

在多个OpenStack region，实现单一的OpenStack对象存储服务端点的话，就需要实现所有的region共享文件存储。对象存储服务内部复制文件到多个节点。这样有一个优点，那就是如果一个文件存放在对象存储服务对于所有region都可见，那么它就可以在任何的或全部的region里的应用和负载所使用。这是简单的高可用失效恢复和灾难恢复回滚。

为满足多region的负载理应扩展对象存储服务，那么多个代理运行，且拥有负载均衡，在每个region都安装存储节点，对象存储服务的入口前端配置HTTP缓存层都是必要的。这样的话，客户端请求对象时会访问缓存而不是直接到存储模块，可有效较少存储网络的负载。另外在HTTP缓存层，在代理和存储节点之间使用诸如Memcache可缓存对象。

如果在多个region的云被设计为不是单一的对象存储服务端点，而是在每个region中都有独立的对象存储服务端点的话，应用就需要掌控同步(如果必要)而且管理操作须确保跨节点的一致性。对于一些应用，在同样的region里访问多个对象存储服务端点意味着希望能够减少延迟，跨region带宽利用以及轻松的部署。

对于块存储服务来说，最重要的决定就是选择存储技术和是否使用专用网络用于从存储服务到计算节点的存储流量传输。

网络

当连接多个region到一起时需要考虑很多设计因素。覆盖网络技术的选择决定了网络包如何在region之间传输，以及如何给应用赋予逻辑网络和地址。如果有安全或监管需求，还得使用加密技术实现region之间的安全传

输。在region内部网络，租户网络的覆盖网络技术亦同等重要，网络覆盖技术和应用生成或接受的网络流量可以是互补的或者交叉目的。例如，使用覆盖技术为一个应用，而中间传输使用了大量的小网络包，如果配置不当会引起大量的延迟或增大每个包的开销。

依赖

The architecture for a multi-site installation of OpenStack is dependent on a number of factors. One major dependency to consider is storage. When designing the storage system, the storage mechanism needs to be determined. Once the storage type is determined, how it is accessed is critical. For example, we recommend that storage should use a dedicated network. Another concern is how the storage is configured to protect the data. For example, the recovery point objective (RPO) and the recovery time objective (RTO). How quickly can the recovery from a fault be completed, determines how often the replication of data is required. Ensure that enough storage is allocated to support the data protection strategy.

Networking decisions include the encapsulation mechanism that can be used for the tenant networks, how large the broadcast domains should be, and the contracted SLAs for the interconnects.

示例

There are multiple ways to build a multi-site OpenStack installation, based on the needs of the intended workloads. Below are example architectures based on different requirements. These examples are meant as a reference, and not a hard and fast rule for deployments. Use the previous sections of this chapter to assist in selecting specific components and implementations based on specific needs.

A large content provider needs to deliver content to customers that are geographically dispersed. The workload is very sensitive to latency and needs a rapid response to end-users. After reviewing the user, technical and operational considerations, it is determined beneficial to build a number of regions local to the customer's edge. In this case rather than build a few large, centralized data centers, the intent of the architecture is to provide a pair of small data centers in locations that are closer to the customer. In this use case, spreading applications out allows for different horizontal scaling than a traditional compute workload scale. The intent is to scale by creating more copies of the application in closer proximity to the users that need it most, in order to

ensure faster response time to user requests. This provider deploys two datacenters at each of the four chosen regions. The implications of this design are based around the method of placing copies of resources in each of the remote regions. Swift objects, Glance images, and block storage need to be manually replicated into each region. This may be beneficial for some systems, such as the case of content service, where only some of the content needs to exist in some but not all regions. A centralized Keystone is recommended to ensure authentication and that access to the API endpoints is easily manageable.

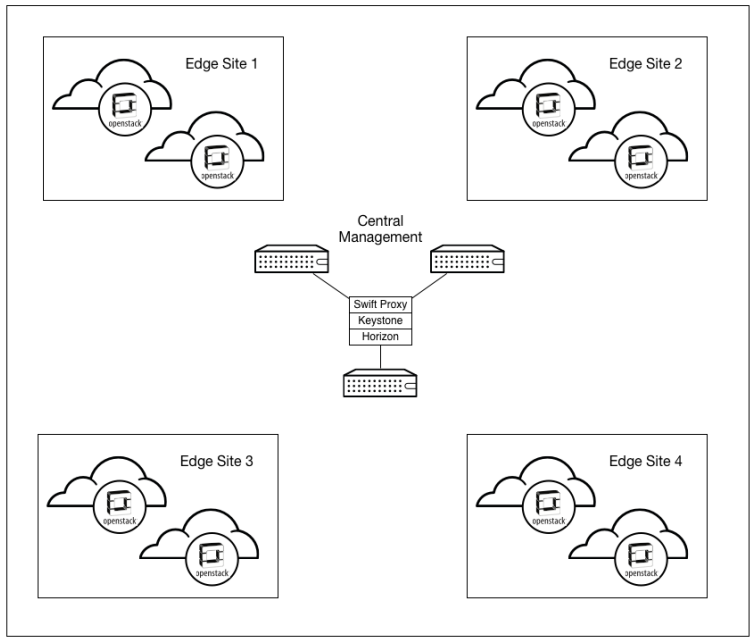
It is recommended that you install an automated DNS system such as Designate. Application administrators need a way to manage the mapping of which application copy exists in each region and how to reach it, unless an external Dynamic DNS system is available. Designate assists by making the process automatic and by populating the records in the each region's zone.

Telemetry for each region is also deployed, as each region may grow differently or be used at a different rate. Ceilometer collects each region's meters from each of the controllers and report them back to a central location. This is useful both to the end user and the administrator of the OpenStack environment. The end user will find this method useful, as it makes possible to determine if certain locations are experiencing higher load than others, and take appropriate action. Administrators also benefit by possibly being able to forecast growth per region, rather than expanding the capacity of all regions simultaneously, therefore maximizing the cost-effectiveness of the multi-site design.

One of the key decisions of running this sort of infrastructure is whether or not to provide a redundancy model. Two types of redundancy and high availability models in this configuration can be implemented. The first type revolves around the availability of the central OpenStack components. Keystone can be made highly available in three central data centers that host the centralized OpenStack components. This prevents a loss of any one of the regions causing an outage in service. It also has the added benefit of being able to run a central storage repository as a primary cache for distributing content to each of the regions.

The second redundancy topic is that of the edge data center itself. A second data center in each of the edge regional locations house a second region near the first. This ensures that the application does not suffer degraded performance in terms of latency and availability.

此示意图描述所设计的解决方案，同时拥有一个为OpenStack服务的中心化的核心数据中心和两个边缘化的数据中心：



地理冗余负载均衡

在脑子里想着一个大型扩展的web应用被设计为遵循云的原则。应用被设计为在应用商店中7*24的提供服务，公司拥有典型的2层架构，前端是用户需求的web服务，后端是用NoSQL数据库存放信息。

最后，经常由于一些因素他们的应用运行在单个地理位置,主要的公有云提供者因此会停止一些服务。因此设计中要考虑如何降低由于单个站点引起的中断业务的几率。

解决方案将由下列OpenStack组件组成：

- 防火墙、交换机、以及负载均衡设备在公网中直接面向全网的连接。
- OpenStack Controller services running, Networking, dashboard, Block Storage and Compute running locally in each of the three regions. The other services, Identity, Orchestration, Telemetry, Image service and Object Storage can be installed centrallywith nodes in each of the region providing a redundant OpenStack Controller plane throughout the globe.
- OpenStack计算节点运行着KVM的hypervisor。

- OpenStack Object Storage for serving static objects such as images can be used to ensure that all images are standardized across all the regions, and replicated on a regular basis.
- 在所有的region中都有一个分布式DNS服务可用，允许动态的为已经部署的实例更新DNS记录。
- A geo-redundant load balancing service can be used to service the requests from the customers based on their origin.

An autoscaling heat template can be used to deploy the application in the three regions. This template includes:

- Web服务，运行 Apache。
- 当实例启动时，中心DNS服务器会填写合适的user_data。
- 正确的Telemetry警告，维护着应用的状态且允许掌控region或实例失效。

Another autoscaling Heat template can be used to deploy a distributed MongoDB shard over the three locationswith the option of storing required data on a globally available swift container. According to the usage and load on the database serveradditional shards can be provisioned according to the thresholds defined in Telemetry.

Two data centers would have been sufficient had the requirements been met. But three regions are selected here to avoid abnormal load on a single region in the event of a failure.

这里用到Orchestration是由于自动伸缩的内部功能和偶然增长的负载后的自动复原功能。另外一些配置管理工具，如Puppet或Chef也会在此场景下用到，但是不会被选择用于Orchestration，在OpenStack云中它们做不到内部的钩子，也不是OpenStack本身的工具。此外，其实相对简单的场景根本用不到这些额外的工具。

OpenStack Object Storage is used here to serve as a back end for the Image service since it is the most suitable solution for a globally distributed storage solutionwith its own replication mechanism. Home grown solutions could also have been used including the handling of replicationbut were not chosen, because Object Storage is already an intricate part of the infrastructureand proven solution.

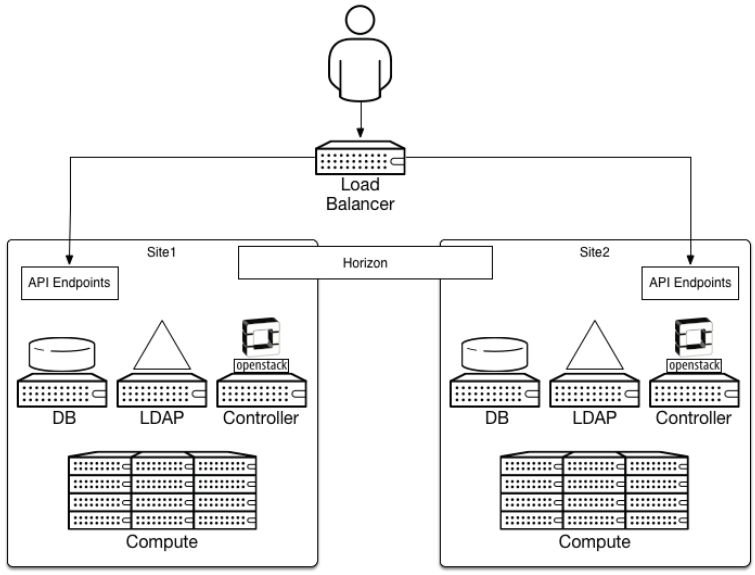
一个额外的负载均衡服务将被使用，而不是OpenStack的LBaaS,因为OpenStack的LBaaS在OpenStack中不是冗余的解决方案而且不具有地理位置的任何特征。

本地服务

A common use for a multi-site deployment of OpenStack, is for creating a Content Delivery Network. An application that uses a location-local architecture requires low network latency and proximity to the user, in order to provide an optimal user experience, in addition to reducing the cost of bandwidth and transit, since the content resides on sites closer to the customer, instead of a centralized content store that requires utilizing higher cost cross-country links.

此架构通常包括一个放置在最接近节点的用户需求的地理位置组件。在此场景下，目标是跨所有站点的100%冗余，已经不是一个需求，意图是为任何指定的最终用户有最大化内容可用，最小的网络hop跳数。尽管他们不一样，存储冗余配置和地理位置冗余负载均衡用例有明显的重叠。

In this example, the application utilizing this multi-site OpenStack install that is location aware would launch web server or content serving instances on the compute cluster in each site. Requests from clients are first sent to a global services load balancer that determines the location of the client, then routes the request to the closest OpenStack site where the application completes the request.


Load
Balancer
Site1
Horizon
Site2
API Endpoints
API Endpoints
openstack
openstack
DB
LDAP
Controller
DB
LDAP
Controller
Compute
Compute

第 7 章 混合云

目录

混合云,依据定义，意思就是跨多个云的设计。举个这种类型架构的例子，设计包含多个OpenStack云的情景(例如，一个是基于OpenStack的私有云，另外一个是基于OpenStack的公有云)，又或许是一个OpenStack云和一个非OpenStack云的互操作的情况(例如，一个基于OpenStack的私有云和Amazon Web Services互相操作).突破 到外部的云，当私有云没有可用的资源时，系统可以自动到外部的云去建立新的实例。

一些包含混合云架构的场景如下:

- 从私有云突破到公有云
- 灾难恢复
- 开发和测试
- 联合云,允许用户从不同的提供商那选择资源
- 构建混合云以支持原来的系统，从而过渡到云

作为混合云设计所处理的系统，其实已经不是云架构或组织所能控制的，一个混合云架构需要考虑架构的方面可能没有其他云所必须的。举例，设计也许需要处理硬件、软件、以及其他组织控制下的应用程序接口。

同样的，基于OpenStack的混合云架构在多大程度上影响着云的运维者和云的消费者，让他们可以使用OpenStack本身的工具去完成一些任务。根据定义，此中状况没有那个单一的云可以提供所有的功能。为了管理整个系统，用户、运维人员、最终用户会需要一个总体工具，即众所周知的云管理平台(CMP)。任何组织只要使用了混合云，都会有一个CMP，有些CMP还需要运维人员登录外部的门户以及创建一个公有云的实例。

There are commercially available options, such as Rightscale, and open source options, such as ManageIQ (http://manageiq.org), but there is no

single CMP that can address all needs in all scenarios. Whereas most of the sections of this book talk about the aspects of OpenStack, an architect needs to consider when designing an OpenStack architecture. This section will also discuss the things the architect must address when choosing or building a CMP to run a hybrid cloud design, even if the CMP will be a manually built solution.

用户需求

Hybrid cloud architectures are complex, especially those that use heterogeneous cloud platforms. It is important to make sure that design choices match requirements in such a way that the benefits outweigh the inherent additional complexity and risks.

Business considerations when designing a hybrid cloud deployment include:

成本	混合云架构涉及到多个提供商和技术架构。这些架构也许为部署和维护付出高额的费用。运维花费更高的原因在于相比与其它架构需要更多复杂的编排以及额外的工具。相比之下，整体运营成本可能在最具成本效益的平台中使用云运营工具部署负载会降低。
赢利空间	Revenue opportunities vary greatly based on the intent and use case of the cloud. As a commercial, customer-facing product, you must consider whether building over multiple platforms makes the design more attractive to customers.
上线时间	One of the most common reasons to use cloud platforms is to improve the time-to-market of a new product or application. For example, using multiple cloud platforms is viable because there is an existing investment in several applications. It is faster to tie the investments together rather than migrate the components and refactoring them to a single platform.
业务或技术的多样性	Organizations leveraging cloud-based services can embrace business diversity and utilize a hybrid cloud design to spread their workloads across multiple cloud providers. This ensures that no single cloud provider is the sole host for an application.
应用增长	Businesses with existing applications may find that it is more cost effective to integrate applications on multiple cloud platforms than migrating them to a single platform.

法律需求

很多辖区对于云环境中的数据的保管及管理都有相关的法律上或者监管上的要求。这些规章的常见领域包括：

- 确保持久化数据的保管和记录管理以符合数据档案化需求的数据保留政策。
- 管理数据的所有权和责任的数据所有权政策。
- 管理位于外国或者其它辖区的数据存储问题的数据独立性政策。
- 管理由于监管上的问题因而特定类型的数据必须存放在特定的地点或者更重要的，由于相同的原因，不能够存放在其它地点的数据管理政策。

Examples of such legal frameworks include the data protection framework of the European Union (Reform of data protection legislation) and the requirements of the Financial Industry Regulatory Authority (FINRA Rules) in the United States. Consult a local regulatory body for more information.

负载考虑

A workload can be a single application or a suite of applications that work together. It can also be a duplicate set of applications that need to run on multiple cloud environments. In a hybrid cloud deployment, the same workload often needs to function equally well on radically different public and private cloud environments. The architecture needs to address these potential conflicts, complexity, and platform incompatibilities. Some possible use cases for a hybrid cloud architecture include:

Dynamic resource expansion or "bursting"	An application that requires additional resources is another common reason you might use a multiple cloud architecture. For example, a retailer needs additional resources during the holiday retail season, but does not want to build expensive cloud resources to meet the peak demand. The user might have an OpenStack private cloud but want to burst to AWS or some other public cloud for these peak load periods. These bursts could be for long

	or short cycles ranging from hourly to yearly.
Disaster recovery-business continuity	The cheaper storage and instance management makes a good case for using the cloud as a secondary site. Using OpenStack public or private cloud in combination with the public cloud for these purposes is popular.
Federated hypervisor-instance management	Adding self-service, charge back and transparent delivery of the right resources from a federated pool can be cost effective. In a hybrid cloud environment, this is a particularly important consideration. Look for a cloud that provides cross-platform hypervisor support and robust instance management tools.
Application portfolio integration	An enterprise cloud delivers efficient application portfolio management and deployments by leveraging self-service features and rules for deployments based on types of use. Stitching together multiple existing cloud environments that are already in production or development is a common driver when building hybrid cloud architectures.
Migration scenarios	A common reason to create a hybrid cloud architecture is to allow the migration of applications between different clouds. Permanent migration of the application to a new platform is one reason, or another might be because the application requires support on multiple platforms.
高可用	Another important reason for wanting a multiple cloud architecture is to address the needs for high availability. Using a combination of multiple locations and platforms,

	a design can achieve a level of availability that is not possible with a single platform. This approach does add a significant amount of complexity.

In addition to thinking about how the workload works on a single cloud, the design must accommodate the added complexity of needing the workload to run on multiple cloud platforms. We recommend exploring the complexity of transferring workloads across clouds at the application, instance, cloud platform, hypervisor, and network levels.

工具考量

When working with designs spanning multiple clouds, the design must incorporate tools to facilitate working across those multiple clouds. Some of the user requirements drive the need for tools that perform the following functions:

Broker between clouds	Since the multiple cloud architecture assumes that there are at least two different and possibly incompatible platforms that are likely to have different costs, brokering software evaluates relative costs between different cloud platforms. The name for these solutions is Cloud Management Platforms (CMPs). Examples include Rightscale, Gravitent, Scalr, CloudForms, and ManageIQ. These tools allow the designer to determine the right location for the workload based on predetermined criteria.
Facilitate orchestration across the clouds	CMPs are tools are used to tie everything together. Using cloud orchestration tools improves the management of IT application portfolios as they migrate onto public, private, and hybrid cloud platforms. We recommend using cloud orchestration tools for managing a diverse portfolio of installed systems across multiple cloud platforms. The typical enterprise IT application portfolio

is still comprised of a few thousand applications scattered over legacy hardware, virtualized infrastructure, and now dozens of disjointed shadow public Infrastructure-as-a-Service (IaaS) and Software-as-a-Service (SaaS) providers and offerings.

网络考虑

The network services functionality is an important factor to assess when choosing a CMP and cloud provider. Considerations are functionality, security, scalability and HA. Important tasks for the architecture include the verification and ongoing testing of critical features for the cloud endpoint.

- Decide on a network functionality framework and design a minimum functionality test. This ensures testing and functionality persists during and after upgrades.
- Scalability across multiple cloud providers may dictate which underlying network framework you choose in different cloud providers. It is important to present the network API functions and to verify that functionality persists across all cloud endpoints chosen.
- High availability implementations vary in functionality and design. Examples of some common methods are active-hot-standby, active-passive and active-active. Development of high availability and test frameworks is necessary to insure understanding of functionality and limitations.
- Consider the security of data between the client, the endpoint, and any traffic that traverses the multiple clouds.

风险规避和管理考虑

Hybrid cloud architectures introduce additional risk because they add additional complexity and potentially conflicting or incompatible components or tools. However, they also reduce risk by spreading workloads over multiple providers. This means, if one was to go out of business, the organization could remain operational. Heightened risks when using a hybrid cloud architecture include:

供应商可用性或实现细节
: This can range from the company going out of business to the company changing how it delivers its services. The design of a cloud architecture is meant to be flexible

	and changeable; however, the cloud is perceived to be both rock solid and ever flexible at the same time.
服务水平协议比较	Users of hybrid cloud environments potentially encounter some losses through differences in service level agreements. A hybrid cloud design needs to accommodate the different SLAs the various clouds involved in the design offer, and must address the actual enforceability of the providers' SLAs.
安全级别	Securing multiple cloud environments is more complex than securing a single cloud environment. We recommend addressing concerns at the application, network, and cloud platform levels. One issue is that different cloud platforms approach security differently, and a hybrid cloud design must address and compensate for differences in security approaches. For example, AWS uses a relatively simple model that relies on user privilege combined with firewalls.
提供者 API 变化	APIs are crucial in a hybrid cloud environment. As a consumer of a provider's cloud services, an organization rarely has control over provider changes to APIs. Cloud services that might have previously had compatible APIs may no longer work. This is particularly a problem with AWS and OpenStack AWS-compatible APIs. The planning of OpenStack included the maintenance of compatibility with changes in AWS APIs. However, over time, the APIs have become more divergent in functionality. One way to address this issue is to focus on using only the most common and basic APIs to minimize potential conflicts.

技术因素

A hybrid cloud environment requires inspection and understanding of technical issues that are not only outside of an organization's data center, but potentially outside of an organization's control. In many cases, it is necessary to ensure that the architecture and CMP chosen are adaptable. All of these factors influence and add complexity to the design of the hybrid cloud architecture.

Incompatibilities with other cloud platforms are inevitable, however, clouds using the same version and distribution of OpenStack are unlikely to experience any issues.

Clouds that exclusively use the same versions of OpenStack should have no issues, regardless of distribution. The newer the distribution in question, the less likely it is that there will be incompatibilities between versions. An OpenStack community initiative defines core functions that need to remain backward compatible between supported versions.

注意

For example, the DefCore initiative defines basic functions that every distribution must support in order to bear the name OpenStack.

Vendors can add proprietary customization to their distributions. If an application or architecture makes use of these features, it will be difficult to migrate to or use other types of environments.

If an environment includes non-OpenStack clouds, it may experience compatibility problems. CMP tools must account for the differences in the handling of operations and implementation of services. Some situations in which these incompatibilities can arise include differences between the way in which a cloud:

- 部署实例
- 管理网络
- Treats 应用
- 实现服务

容量计划

One of the primary reasons many organizations turn to a hybrid cloud system is to increase capacity without having to make large capital investments.

Capacity, and the placement of workloads, accounts for the design of a mostly internally-operated cloud. The long-term capacity plan for this design must incorporate growth over time to prevent permanent consumption of a more expensive external cloud. In order to avoid this scenario, we recommend accounting for the future applications' capacity requirements and plan growth appropriately.

Unpredictability is a consideration for capacity planning. It is difficult to predict the amount of load a particular application might incur if the

number of users fluctuate, or the application experiences an unexpected increase in popularity. It is possible to define application requirements in terms of vCPU, RAM, bandwidth or other resources and plan appropriately. However, other clouds might not use the same meter or even the same oversubscription rates.

Oversubscription is a method to emulate more capacity than may physically be present. For example, a physical hypervisor node with 32GB RAM may host 24 instances, each provisioned with 2GB RAM. As long as all 24 instances are not concurrently utilizing 2 full gigabytes, this arrangement is a non-issue. However, some hosts take oversubscription to extremes and, as a result, performance can frequently be inconsistent. If at all possible, determine what the oversubscription rates of each host are and plan capacity accordingly.

安全性

Security domains are an important distinction when planning for a hybrid cloud environment and its capabilities. A security domain comprises users, applications, servers or networks that share common trust requirements and expectations within a system.

安全域有：

1. 公有
2. 客户机
3. 管理
4. 数据

You can map the security domains individually to the organization's installation or combine them. For example, some deployment topologies combine both guest and data domains onto one physical network, whereas other topologies physically separate the networks. In each case, the cloud operator should be aware of the appropriate security concerns. We recommend mapping security domains against the specific OpenStack deployment topology. The domains and their trust requirements depend upon whether the cloud instance is public, private, or hybrid.

The public security domain is an entirely untrusted area of the cloud infrastructure. It can refer to the internet as a whole, or simply to networks over which an organization has no authority. Do not trust this domain. For example, in a hybrid cloud deployment, any information

traversing between and beyond the clouds is of the public domain and untrustworthy.

The guest security domain handles compute data. Instances on the cloud generate the data, but not services that support the operation of the cloud, such as API calls. We recommend not to trust this domain if you are a public cloud provider that uses hybrid cloud configurations, or a private cloud provider who does not have controls on instance use and allows unrestricted internet access to instances. Private cloud providers, however, can use this network as an internally trusted network if controls are in place.

The management security domain is where services interact. The networks in this domain transport confidential data such as configuration parameters, user names, and passwords. Trust this domain when it is behind an organization's firewall in deployments.

The data security domain is concerned primarily with information pertaining to the storage services within OpenStack. The data that crosses this network has integrity and confidentiality requirements. Depending on the type of deployment there may also be availability requirements. The trust level of this network is heavily dependent on deployment decisions and does not have a default level of trust.

When operating or utilizing public or private clouds, consider the management of the users. The identity service allows for LDAP to be part of the authentication process. Including these systems in your OpenStack deployments may ease user management if integrating into existing systems.

警告

Be mindful of consistency when utilizing third party clouds to explore authentication options.

Due to the passing of user names, passwords, and tokens between client machines and API endpoints, we recommend the placement of API services behind hardware to perform SSL termination.

The hypervisor also requires a security assessment. In a public cloud, organizations typically do not have control over the choice of hypervisor. For example, Amazon uses its own particular version of Xen. Properly securing your hypervisor is important. Attacks made upon the unsecured hypervisor are called a "hypervisor breakout". Hypervisor breakout describes the event of a compromised or malicious instance

breaking out of the resource controls of the hypervisor and gaining access to the bare metal operating system and hardware resources.

There is not an issue if the security of instances is not important. However, enterprises need to avoid vulnerability. The only way to do this is to avoid the situation where the instances are running on a public cloud. That does not mean that there is a need to own all of the infrastructure on which an OpenStack installation operates; it suggests avoiding situations in which sharing hardware with others occurs.

There are other services worth considering that provide a bare metal instance instead of a cloud. In other cases, it is possible to replicate a second private cloud by integrating with a private Cloud-as-a-Service deployment. The organization does not buy the hardware, but also does not share with other tenants. It is also possible to use a provider that hosts a bare-metal "public" cloud instance for which the hardware is dedicated only to one customer, or a provider that offers private Cloud-as-a-Service.

It is important to realize that each cloud implements services differently. What keeps data secure in one cloud may not do the same in another. Be sure to know the security requirements of every cloud that handles the organization's data or workloads.

More information on OpenStack Security can be found in the OpenStack Security Guide.

量力而行

When it comes to utilization, it is important that the CMP understands what workloads are running, where they are running, and their preferred utilizations. For example, in most cases it is desirable to run as many workloads internally as possible, utilizing other resources only when necessary. On the other hand, situations exist in which the opposite is true. The internal cloud may only be for development and stressing it is undesirable. In most cases, a cost model of various scenarios helps with this decision. However, internal priorities influence this analysis. To improve efficiency, make these decisions programmatically.

The Telemetry module (ceilometer) provides information on the usage of various OpenStack components. There are two limitations to consider:

- If there is to be a large amount of data (for example, if monitoring a large cloud, or a very active one) it is desirable to use a NoSQL back end for Ceilometer, such as MongoDB.

- You must monitor connections to non-OpenStack clouds and report this information to the CMP.

性能

Performance is of the upmost importance in the design of a cloud. When it comes to a hybrid cloud deployment, many of the same issues for multi-site deployments apply, such as network latency between sites. It is also important to think about the speed at which a workload can be spun up in another cloud, and how to reduce the time necessary to accomplish the task. This may mean moving data closer to applications or applications closer to the data they process, including a grouping functionality so that connections that require low latency take place over a single cloud rather than spanning clouds. This may also mean ensuring that the CMP has the intelligence to know which cloud can most efficiently run which types of workloads.

As with utilization, native OpenStack tools are available to assist. Ceilometer can measure performance and, if necessary, the Orchestration (heat) module can react to changes in demand by spinning up more resources.

注意

It is important to note, however, that Orchestration requires special configurations in the client to enable functioning with solution offerings from Amazon Web Services. When dealing with other types of clouds, it is necessary to rely on the features of the CMP.

组件

The number and types of native OpenStack components that are available for use is dependent on whether the deployment is exclusively an OpenStack cloud or not.

Using more than one cloud in any situation requires consideration of four OpenStack tools:

- OpenStack Compute (nova): Regardless of deployment location, hypervisor choice has a direct effect on how difficult it is to integrate with one or more additional clouds. For example, integrating a Hyper-V based OpenStack cloud with Azure has less compatibility issues than if KVM is used.

- Networking (neutron): Whether OpenStack Networking or legacy networking (nova-network) is used, the network is one place where understanding integration capabilities is necessary in order to connect between clouds.
- Telemetry 模块 (ceilometer):在大部分依赖使用Telemetry，可用于云的其他组件。
- Orchestration 模块 (heat): 同样的，Orchestration在编排任务中是一个非常有价值的工具，在基于OpenStack云的云管理平台所需要。

特殊因素

混合云部署经常会遇到的两个在其他云环境的不会发生的问题：

- Image portability: Note that, as of the Kilo release, there is no single common image format that is usable by all clouds. Conversion or the recreation of images is necessary if porting between clouds. To make things simpler, launch the smallest and simplest images feasible, installing only what is necessary preferably using a deployment manager such as Chef or Puppet. Do not use golden images for speeding up the process. However, if you repeat the deployment of the same images, we recommend utilizing this technique instead of provisioning applications on lighter images each time.
- API differences: Avoid using a hybrid cloud deployment with more than just OpenStack (or with different versions of OpenStack). The APIs need to perform certain functions that are different. The CMP needs to know how to handle all necessary versions. To get around this issue, some implementers build portals to achieve a hybrid cloud environment, but a heavily developer-focused organization may benefit more from a hybrid cloud broker SDK such as jClouds.

运营因素

混合云的部署有着复杂操作的挑战。有好多种因素会影响到每个云部署的方法需要被考虑，以及用户和运维人员在每个云中的互操作性。不是所有的云提供商提供一样的实现基础设施组件，这就会导致互操作性的兼容性，无论是负载还是一个特别的云管理平台(CMP)。不同的云提供商提供不同级别的集成，且相互竞争。

当选择一个云管理平台时，其中一个最重要的因素考虑就是监测。 获得有价值的洞察每个云是获得所有云整体视图的关键。在现有的CMP中选择的话，决策的条件有，它是否支持监测所有的云平台，是否有兼容的API可用

于查询必要的信息。一旦所有的关于云的信息能够被收集和存储在一个可搜索的数据库中，数据可以被离线操作，负载就不会受到特别的影响。

敏捷性

实现一个混合云解决方案须提供应用跨不同的云环境和技术的可用性。这种可用性是激活在任何一个单一的云环境中发生灾难时仍然可用。每个云都得提供在发生容量问题或在某单个云中完全失效的情况下能够快速的创建新的实例。

应用准备

理解应用负载将被部署到跨混合云环境中的类型显得非常重要。依赖于可用性的底层基础设施的企业级负载不是为运行在OpenStack而设计的，尽管这些类型的应用可以运行在OpenStack云中，如果应用没有基础设施失效后的容错，那么显然就需要运维人员手动介入来恢复。云负载，尽管时刻牢记失效容错的设计以及应用的SLA也不会和底层基础设施捆绑在一起。理想情况下，云应用将会被设计为当整个机架甚至数据中心的基础设施都失效的情况下可以恢复。

升级

如果一公有云已经部署，就无须去考虑升级的事情。一定要检查正在使用的所有公有云提供商他们所宣传的SLA。

注意

在规模较大时，尽管他们提供非常高百分比的在线时间，但更加关心负载必须在短时间内能够被恢复。

当更新私有云部署时，务必小心谨慎，采用增量的变化最小化失误，同时提供回滚或继续往前的能力，当使用持续交付模式时。

另外一个需要考虑的是，升级CMP需要完成任何混合云升级的协同，这对于当任何时候API变动时是必要的，用于支持新的功能的一个云的解决方案。

网络操作中心

当为一个混合云环境规划网络运维中心(NOC)时，认识到哪里是每个基础设施控制驻留点很重要。如果云的显著部分是由外部管理系统来管理的，就得准备好一下子不能变更所有或大多数的计划时间做不到的情形。另外，冲突的情况也会上升，多个供应商对于基础设施管理和暴露的方法是有不同的视野的。这会导致延迟找到根本原因，分析其中，每个供应商都会坚持对其他供应商撒谎。

确保到位的结构使两个云的联网的连接，以形成一个集成的系统是很重要的，头脑中切记切换的状态，那些切换必须不仅保证尽可能的可靠还得包括尽可能的保持最小延迟，以确保整个系统拥有最佳性能。

可维护性

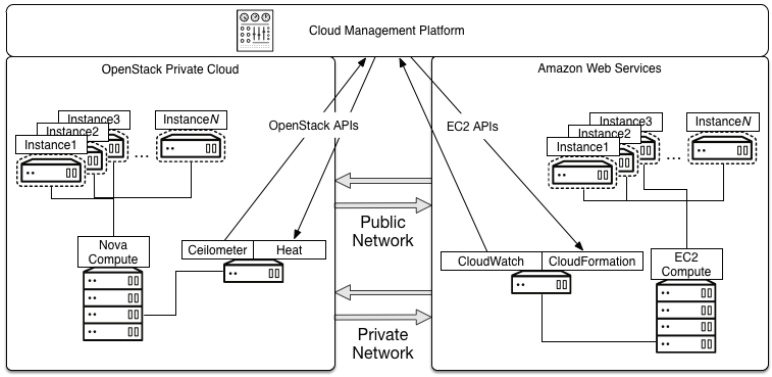
操作混合云是一个非常信赖第三方系统和流程的情况，结果就是对混合云环境的各个环节失去控制，其实也没有必要可能将所有系统保持适当的维护。取而代之的是，用户必须准备好放弃负载且在改进的状态下再生成一次。已经部署完成一个混合云，为上述情形提供敏捷方法，允许迁移负载到替代的云中以响应针对云的特殊问题。

架构

As a first step, map out the dependencies of the expected workloads and the cloud infrastructures that are required to support them. Mapping the applications to targeted cloud environments allows you to architect a solution for the broadest compatibility between cloud platforms, minimizing the need to create workarounds and processes to fill identified gaps.

注意

For your chosen cloud management platform, note the relative levels of support for both monitoring and orchestration.



镜像移植

The majority of cloud workloads currently run on instances using hypervisor technologies such as KVM, Xen, or ESXi. The challenge is that each of these hypervisors uses an image format that may or may not

be compatible with one another. Mitigation in a private or hybrid cloud solution can be standardized on the same hypervisor and instance image format. However this is not always feasible. This is particularly evident if one of the clouds in the architecture is a public cloud that is outside of the control of the designers.

Examples of available conversion tools:

- virt-p2v and virt-v2v
- virt-edit - Edit a file in a virtual machine

The listed tools cannot serve beyond basic cloud instance specifications. Alternatively, build a thin operating system image as the base for new instances. This facilitates rapid creation of cloud instances using cloud orchestration or configuration management tools for more specific templating. Use a commercial image migration tool as another option. If you intend to use the portable images for disaster recovery, application diversity, or high availability, your users could move the images and instances between cloud platforms regularly.

上层服务

很多云都在基本的计算、网络、存储组件之上提供补充的服务，这些额外的服务在云平台中通常用于简单的部署以及管理应用。

When moving workloads from the source to the destination cloud platforms, consider that the destination cloud platform may not have comparable services. Implement workloads in a different way or by using a different technology.

For example, moving an application that uses a NoSQL database service such as MongoDB could cause difficulties in maintaining the application between the platforms.

There are a number of options that are appropriate for the hybrid cloud use case:

- Creating a baseline of upper-layer services that are implemented across all of the cloud platforms. For platforms that do not support a given service, create a service on top of that platform and apply it to the workloads as they are launched on that cloud.

  For example, through the Database service for OpenStack (trove), OpenStack supports MySQL as a service but not NoSQL databases in

production. To either move from or run alongside AWS, a NoSQL workload must use an automation tool, such as the Orchestration module (heat), to recreate the NoSQL database on top of OpenStack.

- Deploying a Platform-as-a-Service (PaaS) technology such as Cloud Foundry or OpenShift that abstracts the upper-layer services from the underlying cloud platform. The unit of application deployment and migration is the PaaS. It leverages the services of the PaaS and only consumes the base infrastructure services of the cloud platform.

- Use automation tools to create the required upper-layer services that are portable across all cloud platforms.

  For example, instead of using any database services that are inherent in the cloud platforms, launch cloud instances and deploy the databases on those instances using scripts or various configuration and application deployment tools.

网络服务

Network services functionality is a barrier for multiple cloud architectures. It could be an important factor to assess when choosing a CMP and cloud provider. Some considerations you should take into account:

- 功能
- 安全性
- 可扩展性
- 高可用(HA)

Verify and test critical cloud endpoint features.

- After selecting the network functionality framework, you must confirm the functionality is compatible. This ensures testing and functionality persists during and after upgrades.

  ### 注意

  Diverse cloud platforms may de-synchronize over time if you do not maintain their mutual compatibility. This is a particular issue with APIs.

- Scalability across multiple cloud providers determines your choice of underlying network framework. It is important to have the network

API functions presented and to verify that the desired functionality persists across all chosen cloud endpoint.

- High availability implementations vary in functionality and design. Examples of some common methods are active-hot-standby, active-passive, and active-active. Develop your high availability implementation and a test framework to understand the functionality and limitations of the environment.

- It is imperative to address security considerations. For example, addressing how data is secured between client and endpoint and any traffic that traverses the multiple clouds. Business and regulatory requirements dictate what security approach to take.

数据

Traditionally, replication has been the best method of protecting object store implementations. A variety of replication methods exist in storage architectures, for example synchronous and asynchronous mirroring. Most object stores and back-end storage systems implement methods for replication at the storage subsystem layer. Object stores also tailor replication techniques to fit a cloud's requirements.

Organizations must find the right balance between data integrity and data availability. Replication strategy may also influence the disaster recovery methods.

Replication across different racks, data centers, and geographical regions has led to the increased focus of determining and ensuring data locality. The ability to guarantee data is accessed from the nearest or fastest storage can be necessary for applications to perform well, for example, Hadoop running in a cloud. The user either runs with a native HDF or on a separate parallel file system. Examples would be Hitachi and IBM.

注意

Take special consideration when running embedded object store methods to not cause extra data replication, which can create unnecessary performance issues.

示例

混合云环境一般被设计为如下场景：

- 从私有云突破负载到公有的OpenStack云
- 从私有云突破负载到公有的非OpenStack云
- 跨云的高可用性(技术多样)

本章讨论这些场景的每个环境实例。

A公司的数据中心运行在危险的低容量环境中。在可预料的将来是不可能扩展此数据中心的。为了满足持续增长的开发资源需求，公司决定使用公有云资源。

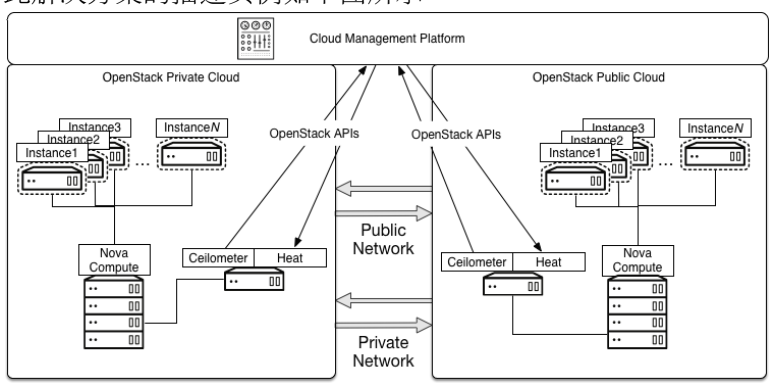
A公司已经拥有显著数量的硬件的数据中心，将负载迁移到公有云中不是可行的方式。

公司已经有个内部的云管理平台，这是选择合适的云的直接需求，当然还依赖于当前本地的容量。

注意

这是一个定制的内部应用，为了特别的目的而写的。

此解决方案的描述实例如下图所示。



此例子展现了两个云，以及一个云管理平台(CMP)，CMP将两个云连接起来。

注意

本书不会尝试去阐释某个特定的CMP，但是上面示意图描述了典型的使用编排组件如何处理负载和Telemetry服务。

其中的私有云是一个基于OpenStack搭建的，有一个或多个控制节点，也有一个或多个计算节点。它包括了有Telemetry模块提供的计费服务，当负载增加时，Telemetry会捕获到，并且将这一消息交由CMP去处理。在私有云

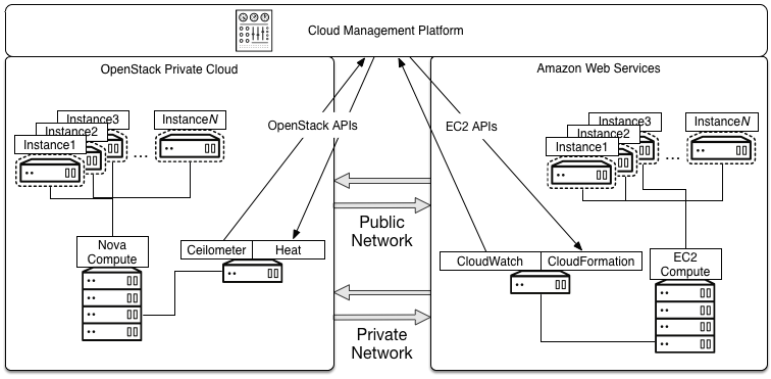
的容量够用时，CMP使用OpenStack提供的API,调 用编排服务来创建实例，从而响应用户的需求。当私有云容量不够用时，CMP会向公有云的编排服务API请求，让之去公有云中创建实例。

在此例中，“A公司”整个的部署都不是直接到外部公有云的，因为公司担心下列情形：对资源缺少控制、整个控制的安全、增加运营费用。

突破到一个不是OpenStack的公有云

第二个例子是从私有云突破负载到非OpenStack公有云中，比如亚马逊web服务(AWS),以增加容量和随需扩展应用。

对于OpenStack-to-AWS的混合云来说，它的架构和下面示意图非常的相像:



公司B的开发者已经在使用AWS来完成他们的一些工作而且不打算切换云提供商。

只要CMP能够使用合适的API连接到外部云提供商，工作流流程仍然和以前的场景没有多大差别。CMP的一些列诸如监测负载，创建>新的实例等动作都是一样的，但是它们必须使用合适的API调用来访问公有云。

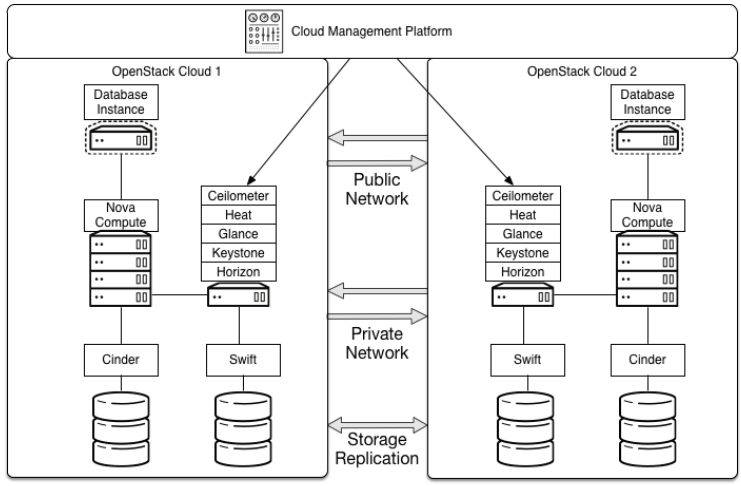
，如果公有云提供商是亚马逊Web服务，CMP须使用EC2 API来创建新的实例以及分配一个弹性IP，而在以前在私有云中IP要添加到HAProxy中。CMP也需要参考AWS特有的工具，如CloudWatch和CloudFormation。

有几款开源工具所构建的CMP已经可用，可以掌控这种类型的转换，它们有ManageIQ,jClouds,以及JumpGate。

高可用/灾难恢复

C公司有着当他们本地的数据中心发生灾难性的事故后能够恢复的需求，一些应用当前运行在他们的私有OpenStack云中，需要保护的数据相关有块存储、对象存储以及数据库，架构的设计支持系统大量组件失效后，仍然能够确保系统继续交付服务。当服务仍然为用户 可用时，系统在后台通过标

准的最佳实践的灾难恢复规则来恢复失效的组件。为了实现这些目标，数据复制到第二个云中，且是远距离的不同地理位置。关于此系统描述的逻辑图见下面示意图：



此例子包含了由一个云管理平台连接的两个私有OpenStack云，来源的云，即1号OpenStack云，包括一个控制器和至少一个实例运行MySQL,也包括至少一个块存储卷和一个对象存储卷，以为用户在所有时间的数据都可用，该方法的用于保护每个数据不同的这些来源的细节。

对象存储是否有复制能力要看对象存储具体的提供商。OpenStack对象存储内嵌此特性，可以创建基于地理位置不同的复制，在每个云中至少要配置一份复制，为了能够在单个磁盘可以工作，云需要被配置使用OpenStack身份且是联合的身份，且和通过Swift代理将多个 云的OpenStack对象存储彼此通信。

For Block Storage, the replication is a little more difficult, and involves tools outside of OpenStack itself. The OpenStack Block Storage volume is not set as the drive itself but as a logical object that points to a physical back end. The disaster recovery is configured for Block Storage for synchronous backup for the highest level of data protection, but asynchronous backup could have been set as an alternative that is not as latency sensitive. For asynchronous backup, the Block Storage API makes it possible to export the data and also the metadata of a particular volume, so that it can be moved and replicated elsewhere. More information can be found here: https://blueprints.launchpad.net/cinder/+spec/cinder-backup-volume-metadata-support.

同步备份，同时在云中创建相同的卷，且选择合适的类型，这样每个云都有相同的后端。这可以通过CMP来完成创建卷，因为CMP知道两个云创建相

同的卷。一旦这被配置，一个解决方案包括DRDB就可以被用户同步真实的物理设备。

数据库组件的备份使用同步备份。MySQL不支持跨地区的复制，所以灾难恢复是由复制文件本身来提供的。在类似MySQL数据库后端使用对象存储是不可能的，所以Swift的复制在这里就没有用武之地。它也不能被决定使用另外其他的地理分层的存储系统来存储数据，诸>如Ceph作为块存储。它必须使用另外层次的保护，其中的一个选择就是使用OpenStack块存储卷来存储数据库，而且和块存储的备份一样来>备份数据库。

第 8 章 可大规模扩展的类型

目录

A massively scalable architecture is a cloud implementation that is either a very large deployment, such as a commercial service provider might build, or one that has the capability to support user requests for large amounts of cloud resources. An example is an infrastructure in which requests to service 500 or more instances at a time is common. A massively scalable infrastructure fulfills such a request without exhausting the available cloud infrastructure resources. While the high capital cost of implementing such a cloud architecture means that it is currently in limited use, many organizations are planning for massive scalability in the future.

A massively scalable OpenStack cloud design presents a unique set of challenges and considerations. For the most part it is similar to a general purpose cloud architecture, as it is built to address a non-specific range of potential use cases or functions. Typically, it is rare that particular workloads determine the design or configuration of massively scalable clouds. Like the general purpose cloud, the massively scalable cloud is most often built as a platform for a variety of workloads. Because private organizations rarely require or have the resources for them, massively scalable OpenStack clouds are generally built as commercial, public cloud offerings.

Services provided by a massively scalable OpenStack cloud include:

- 虚拟机磁盘镜像库
- Raw 块存储
- 文件或对象存储
- 防火墙功能
- 负载均衡功能
- 私有(不可路由到达)及公有(浮动) IP 地址

- 虚拟化网络拓扑
- 软件集合
- 虚拟计算资源

Like a general purpose cloud, the instances deployed in a massively scalable OpenStack cloud do not necessarily use any specific aspect of the cloud offering (compute, network, or storage). As the cloud grows in scale, the number of workloads can cause stress on all the cloud components. This adds further stresses to supporting infrastructure such as databases and message brokers. The architecture design for such a cloud must account for these performance pressures without negatively impacting user experience.

用户需求

比起其它的场景，为可大规模扩展的 OpenStack 设计架构定义用户需求更是意味着需要从不同的，有时候甚至是相反的两个方面来审视我们的设计：云用户的方面，以及云运营者的方面。对于可大规模扩展的 OpenStack 云中的资源的消耗和管理等这些问题，从用户角度来看与从管理者角度来看，其期望和观念都大不相同。

很多辖区对于云环境中的数据的保管及管理都有相关的法律上或者监管上的要求。这些规章的常见领域包括：

- 确保持久化数据的保管和记录管理以符合数据档案化需求的数据保留政策。
- 管理数据的所有权和责任的数据所有权政策。
- 管理位于外国或者其它辖区的数据存储问题的数据独立性政策。
- 管理由于监管上的问题因而特定类型的数据必须存放在特定的地点或者更重要的，由于相同的原因，不能够存放在其它地点的数据管理政策。

Examples of such legal frameworks include the data protection framework of the European Union and the requirements of the Financial Industry Regulatory Authority in the United States. Consult a local regulatory body for more information.

用户需求

可大规模扩展的 OpenStack 云有如下的一些用户需求：

- 云用户希望对云资源进行启动和部署有可重复的、可靠的以及可确定的操作过程。这些功能可以通过基于 web 的接口或者公开可用的 API 入口抛出。对云资源进行请求的所有相应选项应该通过某种类型的用户接口展现给用户，比如命令行接口(CLI)或者API 入口。

- 云用户同样希望拥有一个完全自服务的和按需消费的模型。当一个 OpenStack 云达到“可大规模扩展”的大小时，意味着它也是被希望以每一方面都“作为服务”来进行消费的。

- 对于一个可大规模扩展的 OpenStack 公有云的用户来说，对于安全性、性能以及可用性的控制需求并没有那么强烈。只有与 API 服务的正常运行时间相关的 SLA，以及所提供的服务非常基本的 SLA，是所需要的。用户明白解决这些问题是他们自己的责任。这种期望的例外是一个非常罕见的场景：该可大规模扩展的云是为了一个有特别需求的私有或者政府组织而构建的。

可能与想像中一致，用以确定设计方案的云用户的需求以及期望，都是关注于消费模型之上的。用户希望能够以一种自动化的和确定的方式来使用云中的资源，而不需要以任何对容量、可扩展性或者其它关于该云的底层基础设施的属性的了解作为前提。

运营者的需求

用户对于云的底层基础设施以及属性应该是完全不清楚的，然而，运营者却必须能够构建并且支持该基础设施，也应该了解在大规模的情况下如何操作它。这从运营者的角度提出了关于构建这样一个云的一系列相当强烈的需求：

- 首要的问题是，所有东西都必须能够自动化，从新硬件，包括计算硬件、存储硬件以及网络硬件的部署，到支撑软件的安装及配置，所有的这些都需要能够被自动化完成。手动操作的过程在一个可大规模扩展的 OpenStack 设计架构中是完全无法满足需要的。

- 云运营者要求在云堆栈的所有层次上的资本输出(CapEx)最小化。可大规模扩展的 OpenStack 云的运营者需要使用可靠的商业硬件以及能够自由获取的开源软件组件以减小部署成本和运营费用。像 OpenCompute (关于这个项目更多的信息可以参见 http://www.opencompute.org)这样的项目提供了更多相关的信息和指导意见。很多运营商牺牲了冗余性以减小成本，比如，冗余的电源供应、冗余的网络连接以及冗余的柜顶交换机。

- 运营可大规模扩展云的公司，同样也要求业务费用(OpEx)尽可能最小化。需要对运营开销进行管理的时候，我们则建议使用为云场景进行过优化的硬件。还有一些需要考虑的因素包括电源、冷却系统，以及甚至是底架的物理设计。由于这类实现的规模之大，定制硬件和系统以确保它们是优化过的适合完成相关类型的工作，是非常可能的一件事情。

- 可大规模扩展的 OpenStack 云需要全面的测量及监控功能，通过保持运营人员能够清楚知悉基础设施的状态，以达到最大化业务效率的目的。这包括对硬件和软件状态的全面测度。同样的也需要一个相应的日志和警报框架，用以保存由测量和监控解决方案所提供的数据，并允许针对测量得出的数据采取相应的动作。云运营商还需要一个能够使用测量和监控方案所提供的数据进行容量计划以及容量趋势分析的解决方案。

- 一个可大规模扩展的 OpenStack 云应该是一个多点的云。因此，对于多点 OpenStack 架构设计的用户和运营者需求，在这里也适用。这还包括一系列的关于数据存储、数据存放的地点，以及数据保管等法律上的要求；其它的行政法规和监管的要求；镜像的一致性和可用性；存储的复制和可用性(块存储以及文件/对象存储)；以及认证、授权和审计(AAA)等等；还有很多很多。参考第 6 章 多区域 [111]以了解更多关于多点类型的 OpenStack 云的需求和设计考虑因素。

- 关于诸如空间、底部负重、机柜高度及类型、环境性因素、电源使用及其使用效率(PUE)，以及物理上的安全性等等相关的物理设施的考虑因素，也应该在可大规模扩展的 OpenStack 云的设计架构中一并进行考虑并解决。

技术因素

将一个现存的为其他目的而设计的 OpenStack 环境改造成为可大规模扩展的类型，是一项艰巨的任务。当从头建造一个可大规模扩展的环境时，需要确保初始的部署也是依据不管环境如何增长依然能够适用的原则和选择而建造的。举例来说，在只在第一个地点进行部署的时候，就将整个环境当作是一个多点环境进行部署，就是一个比较好的方式，因为这使得相同的部署及隔离方法，随着环境的扩展，也能够在其它不同的地点被使用，这些地点之间通过专门的线路或者广域网进行连接。在超大规模的环境中，扩展胜过冗余。这种场景下的应用必须依据这个原则进行修改，依赖于整个环境的规模和扩展性和同质性以提供可靠性，而不是使用非商品的硬件解决方案提供的冗余基础设施。

基础设施隔离

幸运的是，OpenStack 的服务是被设计为能够支持水平上大规模的环境的。需要清楚的是这并不是整个支撑基础设施的问题。准确的说，这只是好些 OpenStack 的服务进行数据存储以及远程过程调用通信所用到的数据库管理系统和消息队列的问题。

传统的集群化技术这种环境中也通常被用以提供高可用性和一些额外的扩展性。然而，在大规模的环境下，要对这些组件采取一些额外的配置步骤，以减轻它们所面对的性能压力，避免它们对整个环境的整体性能造

成负面的影响。很重要的一点是需要确保所有的组件上的负载是相对平均的，这样子万一在整个可大规模扩展的环境出现问题时，所有的组件都在，或者至少接近它们的最大负载容量上工作。

OpenStack 中的区域用以隔离完全独立但只由认证模块和面板模块（可选）连接起来的的部署环境。服务在每个区域之中都以独立的 API 入口进行安装，与独立的数据库和消息队列的部署共同组成一套完整的环境。这让用户能够知道环境中发生故障的域，并给予他们一些能力以保证某种程度上的应用弹性，同时又强制要求他们必须选择操作应该应用到哪个区域中去。

在大规模场景下运行的环境需要更加细分其区域和站点，同时又不能要求用户指定故障域。者提供了将整个部署更加细分到故障域的能力，同时也为维护和新硬件的添加提供了逻辑上的单元。在超大规模的环境中，管理员可能一次性添加整个机柜或者甚至是一组机柜上的机器，而不是添加单台计算节点。这样子的节点添加过程，将会用到这里所提到的隔离概念。

单元提供了对一个 OpenStack 环境，也包括区域中的计算部分进行细分的功能，同时保持对外的展现仍然为单个入口点。在每个区域中将会为一系列实际承担负载的计算单元创建一个 API 单元。每个单元拥有其自己的数据库和消息队列(理想情况下是集群化的)，并提供将负载细分到这些子系统中的功能，以提高整体性能。

每个计算单元提供一整个完整的计算环境，包括完整的数据库及消息队列部署、调度器，管理器以及多个计算主机。单元调度器将会把从单个 API 入口点上收到的用户请求，安排到从可用的单元中选出的一个特定单元上。单元内部常规的过滤调度器将负责其内部的这种安排。

使用单元的缺点是这种解决方案在 OpenStack 的服务中，只有计算服务支持得比较好。并且，这种方案也不能够支持一些相对基础的 OpenStack 功能，例如安全组和主机聚合。由于这种方案比较新，并且用途相对特别，在 OpenStack 之中对这种方案的测试也相对比较有限。即使存在种种的这些问题，单元还是在一些著名的大规模 OpenStack 环境中被使用了，包括 CERN 和 Rackspace 中的那些。

主机聚合

主机聚合使得能够将 OpenStack 计算部署划分成逻辑上的分组以实现负载均衡和实例的分布。主机聚合也可以被用于对一个可用域进行更进一步的划分，想象一下一个云环境可能使用主机聚合将可用域中的主机进行分组，分组依据的规则可能是主机共享资源，比如存储和网络，或者主机有特别的属性，比如可信计算硬件。主机聚合并不是明显面向用户的，相反，是通过选择实例类型来实现的，这些实例的类型都带有额外的规格信息，映射到主机聚合的元数据上。

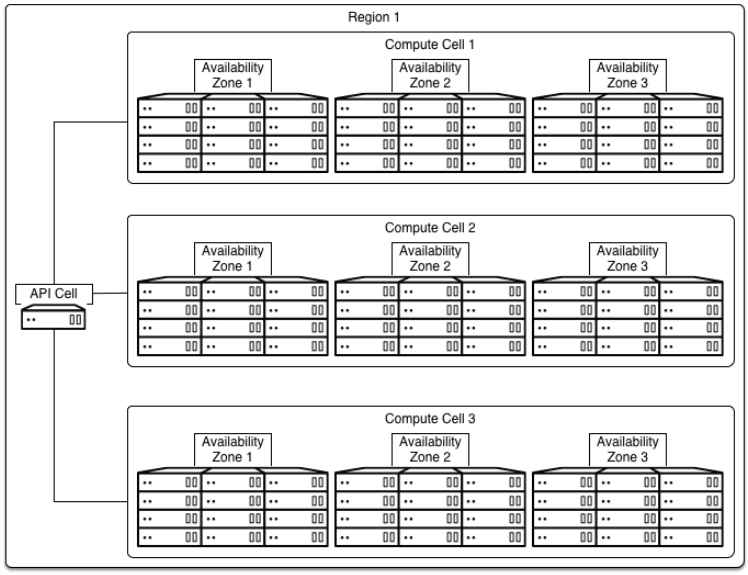
可用域

可用域为细分一个 OpenStack 部署或者区域提供了另外一种机制。实际上，这是明确展现给用户(可选项)的主机聚合。

不同于单元，可用区域并没有自己的数据库服务器以及消息队列代理，它只是简单地代表一个抽象的计算节点组。通常说来，将计算节点分到同一个可用域中依据的是它们可能处于同一个故障域之中，因为他们共享了例如电源、物理网络连接等等的物理特性。可用域对用户来说是可见的，因为它们是能够被定位到的，但是，用户并不怎么需要去确定可用域的位置。另外，运维人员还可以设置默认的可用域，系统将会把实例调度到这个可用域之上，而不是系统中默认的 nova 的可用域。

隔离的例子

在这个例子中，整个云被划分为两个区域，每个地点各一个，每个区域之中又有两个基于数据中心中的电源安排而划分的可用域。其中一系列的主机聚合也被定义以允许通过使用实例规格来指向虚拟机实例，这要求目标主机需要共享诸如 SSD、10GbE 网络或者显卡等特别的硬件和功能。



运营因素

想要在极大规模的情况下正常运营一个 OpenStack 云，对尽可能多的操作过程做自动化的计划就显得尤为重要。自动化包括了准备实例时的配置、监控和警报系统。还有一部分的自动化过程包括了确定什么时候需要人工

干预以及谁应该采取行动的能力。目标是将运行中的系统与运维人员数目的比值尽可能增大，以减少维护成本。在一个扩展到极大规模的环境中，想要运维人员对每个系统都进行单独照看是不可能的。

类似 Puppet 和 Chef 等配置管理工具允许运维人员将系统按照它们的角色进行分组，然后通过配置准备系统，对它们分别创建配置文件以及保证系统的状态。由于错误或者故障而离开预定义的状态的系统很快就会被从活动节点池中移除，并被替换。

在大规模部署的条件下，对单台发生故障系统的进行诊断和调试所耗费的资源成本要比直接替换它高得多。直接将发生故障的系统替换成一个能够被自动部署和配置并且能够很快地被重新加入到活动节点池中的系统，显然要更加经济。通过将劳力密集的、重复性的以及操作难度大的任务自动化，云运营团队就能够更高效的进行管理，因为这种保姆式的工作所需要的资源就更少了。于是，管理者便有时间来解决无法被容易地自动化的以及对业务有着长期影响的任务，比如说容量计划等等。

最前沿

在大规模的场景下运行 OpenStack 需要在稳定性与功能之间做好平衡。比如说，选择比较旧的稳定版本的 OpenStack 以便使得部署更容易看起来比较令人动心。然而在大规模部署的场景之下，对小规模部署造成的困扰不大或者甚至没有什么影响的已知问题，对于大规模的部署来说都可能是缺陷。假如该问题是广为人知的，在通常情况下它可能在较新的发布版本中被解决了。OpenStack 社区能够运用 OpenStack 社区开发者的集体智慧，帮助解决报告到社区中的问题。

当出现问题的时候，在差不多规模场景下运行 OpenStack 的组织的数量，相对于整个 OpenStack 社区来说，是极小的一个比例，因此很重要的一件事情是要与社区分享所遇到的问题，并且在社区中积极倡导将这些问题解决。有些问题可能只在大规模部署的场景下才会出现，所以能够重现和验证该问题的组织是为数不多的，因此将问题良好地文档化并且为问题的解决贡献一些必要的资源，是尤为重要的。

某种情况下，问题的最终解决办法会是部署一套更新版本的 OpenStack。所幸的是，当在一个不可能整个推倒重建的生产环境要解决这样的问题的时候，还可以只重新部署能够解决问题或者能够使得性能明显提高的底层组件的新版本。虽然这乍看起来像是可能给部署带来更高的风险和不稳定性，但是大多数情况下这并不是什么问题。

我们的建议是组织一个开发和运营的团队，由他们来负责开发所需要的特性，调试以及解决问题，并建造用以进行大规模持续集成测试以及持续部署的基础设施。这能够及早地发现缺陷以及使得部署更快和更加简单。除了开发的资源之外，我们也建议招聘消息队列、数据库、分布式系统、网络、云以及存储方面的专家人员。

增长和容量计划

在大规模场景下运行 OpenStack 还有一个重要的考虑因素，是要对增长和利用率趋势进行规划，从而为短期和长期计划资本性支出。这需要计算、网络以及存储等资源的利用率的测量数据，以及这些数据的历史记录。固定的大客户租户可能造成所有资源的利用率有个迅速的增长，在一个组织内部的对其内部云，或者在公有云上的用户对公开提供的服务等的稳定增长的部署及使用，则会使得利用率出现一个稳定的增长趋势。

技能和培训

对存储、网络以及计算等资源的增长进行规划只是为大规模运行 OpenStack 进行的扩展规划的一个方面。对于开发及运维人员的增长以及能力的培养，也是一个重要的考虑因素。让团队的成员参加 OpenStack 大型会议和聚会，鼓励团队成员积极参与邮件列表以及委员会的讨论等，都是让他们保持技能领先并与社区建立良好关系的非常重要的方式。另外，这里还有一个市场上提供 OpenStack 相关技能培训的机构的列表：http://www.openstack.org/marketplace/training/.

第 9 章 特殊场景

目录

尽管大多数的 OpenStack 架构设计都能归类于其它章节中描述的七种主要场景(计算密集型、网络密集型、存储密集型、通用设计、多站点、混合云以及可大规模扩展)，仍然存在其他一些场景独特到无法归类入主要场景中的任何一个之中。本章将讨论一些这种特殊的应用场景，以及每种场景的细节和设计的考虑因素。

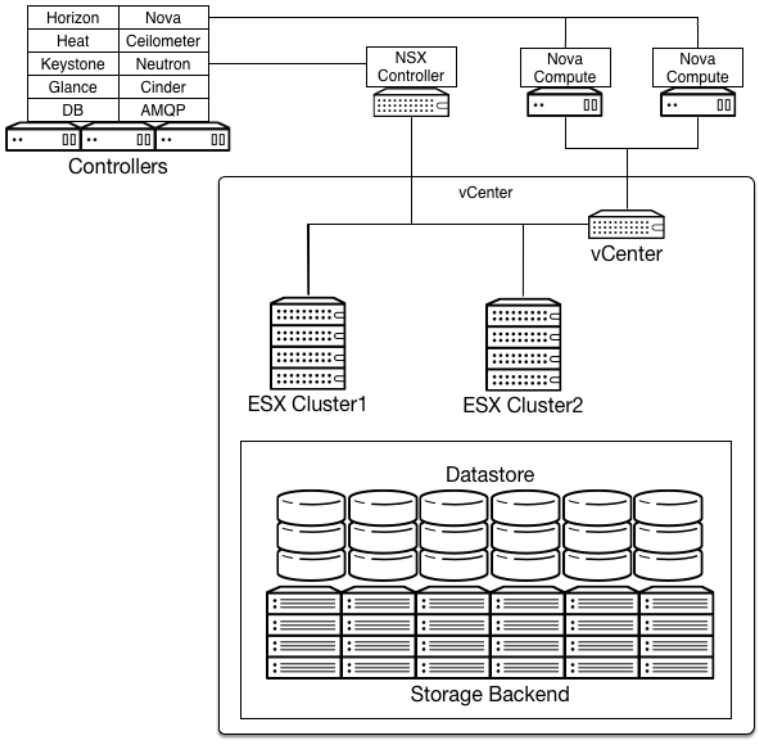
- 特殊的网络应用：此节介绍运行可能涉及直接从网线上读取数据包或者参与路由协议的面向联网的软件。
- 软件定义网络(SDN)：这种场景详细介绍了在 OpenStack 之中运行 SDN 控制器以及 OpenStack 加入到一个软件定义的网络中的情况。
- 桌面即服务：这是为希望在云环境中运行(桌面即服务)所准备的，在私有云以及公有云的情景下都可用。
- OpenStack 上的 OpenStack：一些机构认为通过在一个 OpenStack 部署之上运行 OpenStack 的方式来构建多层次的云有其技术上的意义。
- 专门的硬件：有一些非常特别的情况可能需要在 OpenStack 环境中使用专门的硬件设备。

多种类型宿主机的例子

一个金融公司需要将其应用从一个传统的虚拟化环境中迁移到一个新的由 API 驱动的合理组织的环境中。该公司的一些应用有着严格的支持需求，这限制了在哪些宿主机上它们能够被支持。然而另外的一些应用没有这些限制也不需要相同的特性。根据这些需求，整个目标环境就需要多种类型的宿主机。

当前的环境是一个有 20 台 VMware ESXi 宿主机，支撑着 300 个不同大小的实例的 vSphere 环境。大约有 50 个实例必须在 ESXi 上运行，剩余的则有着更为灵活的需求。

该公司决定将对整个系统的管理迁移到一个由 OpenStack 提供的普通平台中。



解决方法是为一般用途的实例运行由一个 KVM 宿主机组成的主机集群，同时为需要 ESXi 的实例运行一个单独的主机集群。这种方式下，必须运行在 ESXi 之上的负载可以调度到那些宿主机上，同时其它的那些可以调度到 KVM 宿主机之上。

OpenStack 镜像服务中的镜像随附着特别的宿主机元数据，因此当用户请求特定的镜像时，该实例将会在相应的集群中启动。ESXi 中的镜像以 VMDK 格式储存。QEMU 磁盘镜像能够转换至 VMDK 格式，包括精简置备、厚置备、厚置备延迟置零以及厚置备置零等 VMFS 平坦磁盘。请注意，一个 VMFS 精简置备的磁盘一旦从 VMFS 导出到一个非 VMFS 的位置，例如 OpenStack 镜像服务时，它就会变成一个预分配的平坦磁盘。这将影响从 OpenStack 镜像服务到数据存储的传输时间，因为完全预分配的磁盘而不是一个精简置备的磁盘需要被传输。

此例子中有个额外的复杂之处在于，VMware 主机集群的计算节点需要与 vCenter 通信，然后 vCenter 才请求将实例调度到在一个 ESXi 宿主机上运行，而并非通过简单地使用镜像的元数据调用特定的主机集群从而让实例在宿主机上直接启动。在 Icehouse 版本中，这个功能需要 VMware 分布式资源调度器(DRS)在集群中被启用并且设置为“全自动”。

由于 DRS 的需求，请注意 vSphere 需要共享存储(DRS 使用需要共享存储的 vMotion)。整个解决方案使用共享存储为 KVM 实例提供块存储功能，同时为 vSphere 提供存储。环境中使用一个专用的数据网络来支持这个存储功能，因此计算主机上需要有专用的网卡设备来承载这些流量。vSphere 支持使用 OpenStack 块存储将 VMFS 存储展现给实例作为存储，因此在这个架构中对块存储的使用同时支持两种类型的宿主机。

在这个场景中，网络连接由配置了 VMware NSX 插件驱动的 OpenStack 联网提供。系统还可以采用传统的联网方式(nova-network)，也同样被这个设计中的两种类型的宿主机所支持，但是有一些局限性。具体说来，采用传统联网方式时，安全组在 vSphere 上是不被支持的。既然设计中包含了 Vmware NSX 作为其中的一部分，当用户在任何一个主机集群中启动实例时，该实例将会被连接至由用户定义的相应的覆盖逻辑网络之中。

请注意这个解决方案部署时需要小心进行，因为有一些 OpenStack 计算集成相关的设计因素。当在 OpenStack 中使用 vSphere 时，nova-compute 服务将会被配置为与 vCenter 进行通信并将整个 ESXi 集群显示为单台巨大的宿主机(可以运行多个 nova-compute 实例以表示多个 ESXi 集群或者连接至多个 vCenter 服务器)。如果运行 nova-compute 服务的进程崩溃，到 vCenter 服务器以及在该服务器后端的 ESXi 集群的连接将会被切断，并且将不能够在该 vCenter 上产生更多的实例，尽管 vSphere 本身可以被配置为具有高可用功能。因此，监控连接至 vSphere 的 nova-compute 服务的中断就显得尤为重要。

特殊网络应用的例子

有些与网络进行互动的应用需要更加专门的连接。类似于 looking glass 之类的应用需要连接到 BGP 节点，或者路由参与者应用可能需要在2层上加入一个网络。

挑战

将特殊的网络应用连接至它们所需要的资源能够改变 OpenStack 部署的设计。基于覆盖网络的部署是无法支持路由参与者应用的，而且也可能阻挡二层网络监听者应用。

可能的解决方案

使用带有提供商网络的 OpenStack 联网方式进行 OpenStack 的部署可以允许直接到上游网络设备的二层网络连接。这种设计提供了通过中间系统到中间系统(ISIS)协议进行通信，或者传输由 OpenFlow 控制器所控制的网络包等功能所需要的二层联网要求。使用比如 Open vSwitch 这类的带有代理程序的多种二层网络插件能够允许通过 VLAN 直接到三层设备上的特定端口的私有连接。这使得之后会加入自治系统的 BGP 点对点连接能够存在。应该尽量避免使用三层网络的插件，因为它们会分隔广播域并且阻止邻接路由器的形成。

软件定义网络

软件定义网络(SDN)指的是网络数据转发平面以及控制平面的隔离。SDN 已经成为在网络中管理及控制网络包流的流行方案。SDN 使用覆盖网络或者直接控制的-2层网络设备来监测网络流的路径，这对云环境提出了一些挑战。有些设计者可能希望在 OpenStack 环境中运行他们的控制器。另外一些则可能希望将他们的 OpenStack 环境加入到一个 由 SDN 方式进行控制的网络之中。

挑战

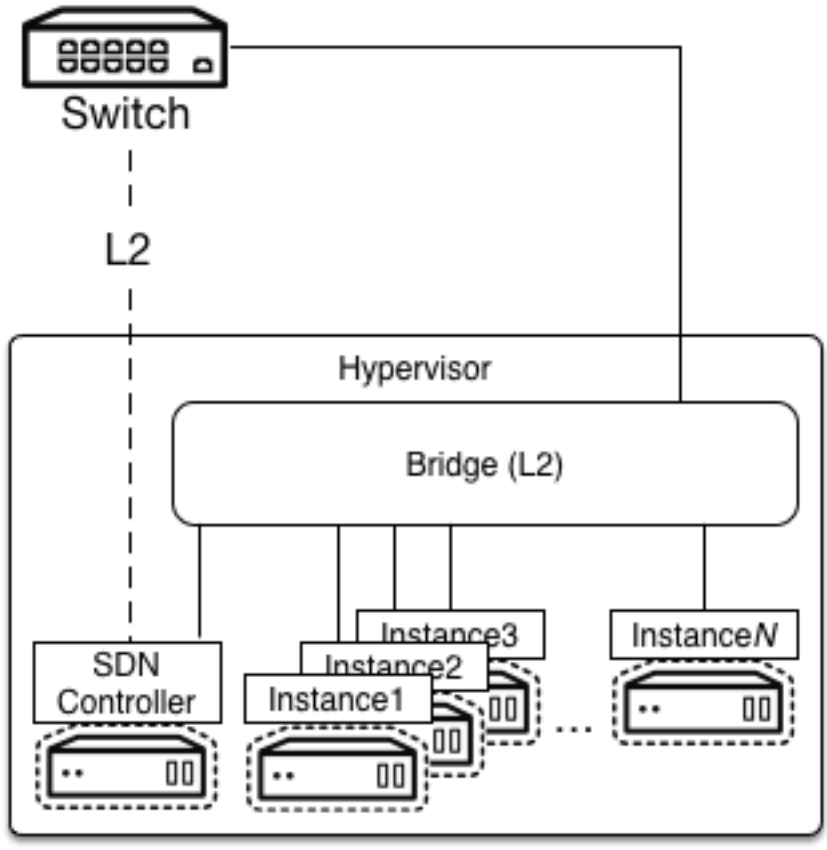
相对来说，SDN 是一个比较新的，仍未被标准化的概念，所以 SDN 系统可能来自很多不同的具体实现。因此，一个真正意义上示范性架构是目前无法给出的。相反的，我们只能够分析当前或者目标 OpenStack 设计中的各种不同，并确定哪些地方将会出现潜在的冲突或者还存在差距。

可能的解决方案

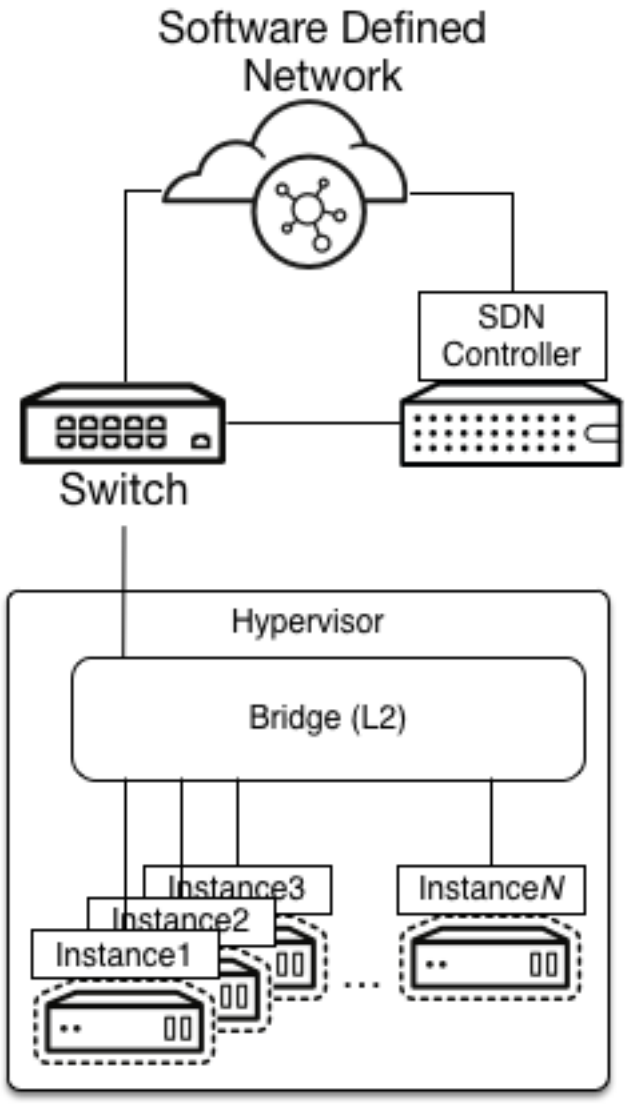
如果一个 SDN 的实现由于它需要直接管理和操作交换机因而要求-2层网络的连接，那么就不建议运行覆盖网络或者-3层网络的代理程序。假如 SDN 控制器运行在 OpenStack 环境之中，则可能需要创建一个 ML2 插件并且将该控制器实例调度到能够连接至能够直接与交换机硬件进行通信的租户 VLAN。另一个可能的方式是，基于外部硬件设备的支持情况，使用一端终结在交换机硬件之上的网络隧道。

图示

OpenStack 上运行 SDN 控制器：



OpenStack 加入到 SDN 控制器所控制的网络中：



桌面即服务

虚拟桌面基础设施(VDI)是在远程服务器上提供用户桌面环境的服务。此类应用对于网络延迟非常敏感并且需要一个高性能的计算环境。传统上这类环境并未成放到云环境之中，因为极少的云环境会支持如此程度暴露给终端用户的高要求负载。近来，随着云环境的稳定性越来越高，云厂商们开始提供能够在云中运行虚拟桌面的服务。在不远的将来，OpenStack 便能够作为运行虚拟桌面基础设施的底层设施，不管是内部的，还是在云端。

挑战

设计一个适用于运行虚拟桌面的基础设施是一个与为其它大部分虚拟化任务进行设计大不相同的工作。该基础设施设计时必须考虑到各种因素，比如以下例子：

- 启动风暴，再很短的时间内发生了大量的虚拟机同时启动的事
- 在提供的虚拟桌面中运行的应用的性能
- 操作系统以及和 OpenStack hypervisor的兼容性

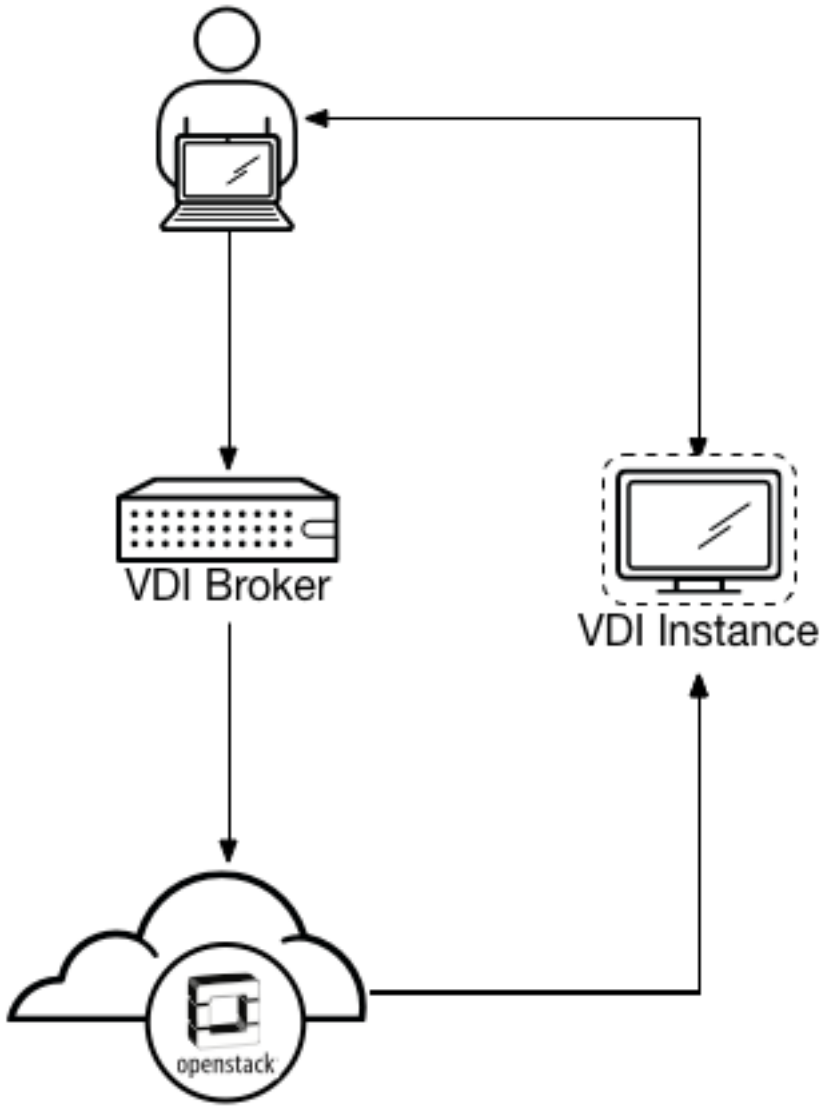
代理

连接代理是架构中的核心组件，其决定了哪个虚拟桌面会被分配至或者连接至用户。这种代理程序一般都是比较成熟全面的管理产品，能够支持远程桌面的自动部署及准备工作。

可能的解决方案

目前有多种商业产品能够提供这种代理的解决方案，但 OpenStack 项目中没有原生的支持。不提供这样一个代理也是一个选择，但是手动管理这些功能并不能够满足作为一个大规模的企业级解决方案的要求。

图示



OpenStack 上的 OpenStack

某些情况下可能需要运行一个嵌套在另外的 OpenStack 云上的 OpenStack。这种场景使得整个 OpenStack 环境都由运行在底层 OpenStack 云所控制的宿主机和服务器上的实例来管理和准备。公有云提供商可以使用这项技术在完全基于 OpenStack 的云上来高效地管理升级和维护的过

程。开发和测试 OpenStack 的人员也可以在不管公有还是私有的可用的 OpenStack 计算资源上，根据这个指引部署他们自己的 OpenStack 环境。

挑战

网络方面是这个嵌套云架构的最复杂部分。由于底层裸金属架构的云拥有所有的硬件设备，当使用 VLAN 的时候，VLAN 需要被暴露到底层云上的物理端口，但是它们也同样需要被展现到嵌套层中。作为备选方案，网络覆盖技术能够在上层云，运行在 OpenStack 之上的 OpenStack， 被用以为部署提供所需要的软件定义联网。

虚拟机管理程序

这种场景中需要解决的一个关键问题是决定采用何种方式为嵌套的 OpenStack 环境提供宿主机。这个决定将会影响在嵌套 OpenStack 部署中哪些操作系统可以被使用。

可能的解决方案：部署

部署整个 OpenStack 环境是比较麻烦的，不过这个困难可以被轻松化解，通过创建一个 Heat 模板来部署整个环境或者使用一个配置管理系统。一旦 Heat 模板创建完成，部署新的 OpenStack 环境就变得非常简单并且可以使用自动化方式完成。

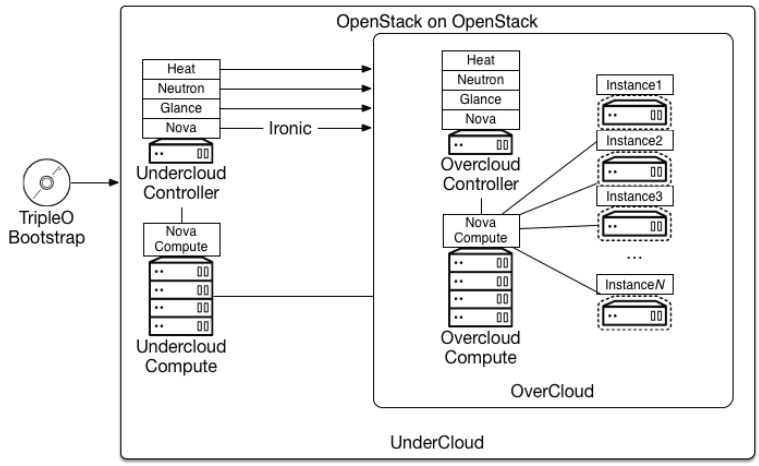
目前，OpenStack-on-OpenStack 项目(TripleO) https://wiki.openstack.org/wiki/TripleO

可能的解决方案：宿主机

在运行 TripleO 的场景下，底层的 OpenStack 环境以裸金属的方式部署计算节点。然后 OpenStack 将会被部署在这些计算的裸金属服务器上，并使用诸如 KVM 之类的管理程序。

如果是要为测试目的运行小规模的 OpenStack 云环境，并且性能不是关键的考虑因素的情况下，则 QEMU 可以被作为替代使用。在实例中运行一个 KVM 的宿主机是可能的(参考 http://davejingtian.org/2014/03/30/nested-kvm-just-for-fun/)，但这不是被支持的配置方式，并且对这样一个使用场景来说，也会是一个复杂的解决方案。

图示



专门的硬件

有些特定的负载需要难以虚拟化或者无法共享的特殊的硬件设备。例如负载均衡器、高并行暴力运算以及与网线直接相连的联网等应用，都可能需要基本 OpenStack 组件不提供的功能。

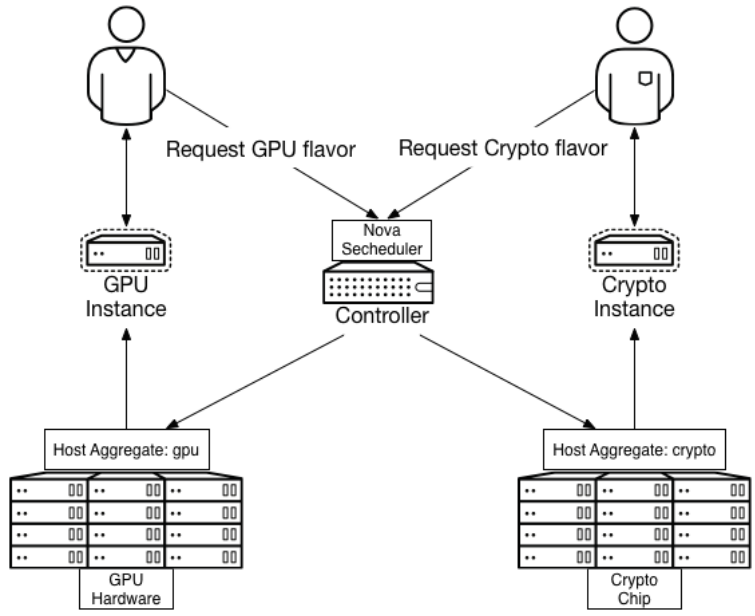
挑战

一些应用需要使用硬件设备以提高性能或者提供非虚拟 CPU、内存、网络或者存储等的功能。这些可以是共享的资源，例如加密处理器，或者专用的资源，例如图形处理器。对于其中一些设备，OpenStack 有办法能够直接使用它们，而对于另外一些设备，则可能需要完成额外的工作。

解决方案

要为一组实例提供加密卸载，可以通过使用镜像服务的配置选项将加密芯片分配至客户机中的一个设备节点。OpenStack 命令行参考中的镜像服务属性键一章包含了关于配置这个解决方案的更多信息。但是这个方案允许所有的客户机都通过该已配置的镜像来使用宿主机的加密设备。

如果需要直接使用某个特定的设备，可以通过使用 PCI 穿透技术将设备指定至每个宿主机上的单个实例。OpenStack 管理员需要定义一个明确具有 PCI 设备以便适当调度实例的实例类别。关于 PCI 穿透的更多信息，包括实施与使用的说明，请参考 https://wiki.openstack.org/wiki/Pci_passthrough。


Request GPU flavor
Request Crypto flavor
Nova Secheduler
Controller
GPU Instance
Crypto Instance
Host Aggregate: gpu
Host Aggregate: crypto
GPU Hardware
Crypto Chip

第 10 章 参考

Data Protection framework of the European Union: Guidance on Data Protection laws governed by the EU.

Depletion of IPv4 Addresses: describing how IPv4 addresses and the migration to IPv6 is inevitable.

Ethernet Switch Reliability: Research white paper on Ethernet Switch reliability.

Financial Industry Regulatory Authority: Requirements of the Financial Industry Regulatory Authority in the USA.

Image Service property keys: Glance API property keys allows the administrator to attach custom characteristics to images.

LibGuestFS Documentation: Official LibGuestFS documentation.

Logging and Monitoring: Official OpenStack Operations documentation.

ManageIQ Cloud Management Platform: An Open Source Cloud Management Platform for managing multiple clouds.

N-Tron Network Availability: Research white paper on network availability.

Nested KVM: Post on how to nest KVM under KVM.

Open Compute Project: The Open Compute Project Foundation's mission is to design and enable the delivery of the most efficient server, storage and data center hardware designs for scalable computing.

OpenStack Flavors: Official OpenStack documentation.

OpenStack High Availability Guide: Information on how to provide redundancy for the OpenStack components.

OpenStack Hypervisor Support Matrix: Matrix of supported hypervisors and capabilities when used with OpenStack.

OpenStack Object Store (Swift) Replication Reference: Developer documentation of Swift replication.

OpenStack Operations Guide: The OpenStack Operations Guide provides information on setting up and installing OpenStack.

OpenStack Security Guide: The OpenStack Security Guide provides information on securing OpenStack deployments.

OpenStack Training Marketplace: The OpenStack Market for training and Vendors providing training on OpenStack.

PCI passthrough: The PCI API patches extend the servers/os-hypervisor to show PCI information for instance and compute node, and also provides a resource endpoint to show PCI information.

TripleO: TripleO is a program aimed at installing, upgrading and operating OpenStack clouds using OpenStack's own cloud facilities as the foundation.

附录 A. 社区支持

目录

以下可用的资源是帮助用户运行和使用OpenStack。OpenStack社区会经常性的改进和增加OpenStack的主要特性，如果用户有问题，请不要在提问题方面犹豫。使用下面列出的资源，以获得OpenStack社区的支持，也能得到一些安装/使用时一些解决问题的思路和方法。

文档

For the available OpenStack documentation, see docs.openstack.org.

To provide feedback on documentation, join and use the <openstack-docs@lists.openstack.org> mailing list at OpenStack Documentation Mailing List, or report a bug.

以下书籍解释了如何安装一个基于OpenStack云及其相关的组件

- Installation Guide for openSUSE 13.2 and SUSE Linux Enterprise Server 12
- Installation Guide for Red Hat Enterprise Linux 7, CentOS 7, and Fedora 21
- Installation Guide for Ubuntu 14.04 (LTS)

以下书籍解释了如何配置和运行一个基于OpenStack的云：

- 架构设计指南
- 云计算平台管理员手册

- 配置参考手册
- 实战指南
- Networking Guide
- 高可用指南
- 安全指南
- 虚拟机镜像指南

以下书籍解释了如何使用OpenStack图形界面和命令行客户端：

- 应用程序接口快速入门
- 用户指南
- 管理员手册
- 命令行参考

下面文档提供了OpenStack 应用程序接口的参考和向导：

- OpenStack应用程序接口完全参考(HTML)
- OpenStack应用程序接口完全参考(PDF)

问答论坛

During the set up or testing of OpenStack, you might have questions about how a specific task is completed or be in a situation where a feature does not work correctly. Use the ask.openstack.org site to ask questions and get answers. When you visit the https://ask.openstack.org site, scan the recently asked questions to see whether your question has already been answered. If not, ask a new question. Be sure to give a clear, concise summary in the title and provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.

OpenStack 邮件列表

A great way to get answers and insights is to post your question or problematic scenario to the OpenStack mailing list. You can learn from and help others who might have similar issues. To subscribe or view the archives, go to http://lists.openstack.org/cgi-bin/mailman/listinfo/

openstack. You might be interested in the other mailing lists for specific projects or development, which you can find on the wiki. A description of all mailing lists is available at https://wiki.openstack.org/wiki/MailingLists.

OpenStack 维基百科

The OpenStack wiki contains a broad range of topics but some of the information can be difficult to find or is a few pages deep. Fortunately, the wiki search feature enables you to search by title or content. If you search for specific information, such as about networking or OpenStack Compute, you can find a large amount of relevant material. More is being added all the time, so be sure to check back often. You can find the search box in the upper-right corner of any OpenStack wiki page.

Launchpad的Bug区

The OpenStack community values your set up and testing efforts and wants your feedback. To log a bug, you must sign up for a Launchpad account at https://launchpad.net/+login. You can view existing bugs and report bugs in the Launchpad Bugs area. Use the search feature to determine whether the bug has already been reported or already been fixed. If it still seems like your bug is unreported, fill out a bug report.

一些小贴士：

- 提供清晰、简洁的语法。
- 尽可能提供详细的细节描述。将命令行的输出或者trace输出粘贴出来，如果是截图请贴链接，以及其他任何有用的信息。
- 确保包含了软件和包的版本信息，尤其是使用的正在开发中的分支，诸如"Juno release" vs git commit bc79c3ecc55929bac585d04a03475b72e06a3208，这样的描述。
- 任何特别的部署信息都是有用的。例如用户使用的是Ubuntu 14.04，或者多节点安装。

以下列出Launchpad Bug区：

- Bugs: OpenStack 块存储 (cinder)
- Bugs: OpenStack 计算 (nova)
- Bugs: OpenStack 仪表盘 (horizon)

- Bugs: OpenStack 认证 (keystone)
- Bugs: OpenStack Image service (glance)
- Bugs: OpenStack 网络 (neutron)
- Bugs: OpenStack 对象存储 (swift)
- Bugs: 裸金属服务 (ironic)
- Bugs: 数据处理服务 (sahara)
- Bugs: Database service (trove)
- Bugs: 编排 (heat)
- Bugs: 计量 (ceilometer)
- Bugs: 消息服务 (zaqar)
- Bugs: OpenStack 应用程序接口文档 (developer.openstack.org)
- Bugs: OpenStack 文档 (docs.openstack.org)

The OpenStack 在线聊天室频道

The OpenStack community lives in the #openstack IRC channel on the Freenode network. You can hang out, ask questions, or get immediate feedback for urgent and pressing issues. To install an IRC client or use a browser-based client, go to https://webchat.freenode.net/. You can also use Colloquy (Mac OS X, http://colloquy.info/), mIRC (Windows, http://www.mirc.com/), or XChat (Linux). When you are in the IRC channel and want to share code or command output, the generally accepted method is to use a Paste Bin. The OpenStack project has one at http://paste.openstack.org. Just paste your longer amounts of text or logs in the web form and you get a URL that you can paste into the channel. The OpenStack IRC channel is #openstack on irc.freenode.net. You can find a list of all OpenStack IRC channels at https://wiki.openstack.org/wiki/IRC.

文档反馈

To provide feedback on documentation, join and use the <openstack-docs@lists.openstack.org> mailing list at OpenStack Documentation Mailing List, or report a bug.

OpenStack分发包

以下是Linux发行版针对OpenStack的社区支持：

- Debian: https://wiki.debian.org/OpenStack
- CentOS, Fedora, 以及 Red Hat Enterprise Linux: https://www.rdoproject.org/
- openSUSE 和 SUSE Linux Enterprise Server: https://en.opensuse.org/Portal:OpenStack
- Ubuntu: ubuntu官方服务器团队之OpenStack云

术语表

6to4
一种可以在IPv4的网络中传输IPv6包的机制，提供了一种迁移到IPv6的策略。

块存储
OpenStack核心项目，它管理卷、卷快照，以及卷类型。块存储的项目名称叫做cinder。

ceilometer
是遥测服务的项目名称，此服务整合OpenStack中提供计量和测量的项目。

单元
Provides logical partitioning of Compute resources in a child and parent relationship. Requests are passed from parent cells to child cells if the parent cannot provide the requested resource.

cinder
OpenStack核心项目，为虚拟机提供块存储服务。

计算
OpenStack核心项目，提供计算服务。项目名称为nova。

仪表盘
OpenStack基于web的管理接口。项目名称为horizon。

Database service
An integrated project that provide scalable and reliable Cloud Database-as-a-Service functionality for both relational and non-relational database engines. The project name of Database service is trove.

桌面即服务
A platform that provides a suite of desktop environments that users access to receive a desktop experience from any location. This may provide general use, development, or even homogeneous testing environments.

encapsulation
The practice of placing one packet type within another for the purposes of abstracting or securing data. Examples include GRE, MPLS, or IPsec.

glance
A core project that provides the OpenStack Image service.

heat
一个集成的项目，目标是为OpenStack编排多种云应用。

horizon
: 提供web接口的仪表盘的OpenStack项目。

混合云
: 混合云即是2个或多个云的组合(这些云可以是公有，私有，或者社区)，它们彼此独立运行但是是绑定到一起的，拥有多部署模式的优点。混合云还拥有连接托管云资源、被管理云资源或专有的云资源的能力。

IaaS
: 基础设施即服务。IaaS是一种配置模式，将数据中心的物理组件，如存储、硬件、服务器以及网络等以组织外包的方式提供。服务运营商提供设备，负责机房以及操作和维护。用户只需要按需使用并付费即可。IaaS是云服务模式的一种。

IOPS
: IOPS(每秒输入/输出操作)是一种常见的性能测量基准，针对于计算机存储设备，例如硬盘，固态硬盘，存储区域网络等。

keystone
: OpenStack验证服务的项目。

网络
: A virtual network that provides connectivity between entities. For example, a collection of virtual ports that share network connectivity. In Networking terminology, a network is always a layer-2 network.

网络
: A core OpenStack project that provides a network connectivity abstraction layer to OpenStack Compute. The project name of Networking is neutron.

neutron
: OpenStack核心项目之一，为OpenStack计算提供网络连接抽象层。

nova
: OpenStack核心项目之一，提供计算服务。

对象存储
: OpenStack核心项目之一，提供一致性的、冗余的存储、可恢复的数字内容。OpenStack对象存储的项目名称是swift。

Open vSwitch
: Open vSwitch是一款产品级的，多层的虚拟交换机，基于开源Apache2.0许可证分发。被设计用于基于可编程扩展的大规模网络自动化，支持标准的管理接口和协议(例如NetFlow, sFlow, SPAN, RSPAN, CLI, LACP, 802.1ag)。

OpenStack
: OpenStack是一个云操作系统，通过数据中心可控制大型的计算、存储、网络等资源池。所有的管理通过前端界面管理员就可以完成，同样也可以通过web接口让最

终用户部署资源。OpenStack是一个开放源代码的项目，基于Apeche许可证2.0发布。

swift
OpenStack核心项目之一，提供对象存储服务。

Telemetry
An integrated project that provides metering and measuring facilities for OpenStack. The project name of Telemetry is ceilometer.

trove
OpenStack为提供数据库服务的应用程序项目。

Xen
Xen是一种hypervisor,使用微内核设计，提供在同一台硬件计算机并行的运行多个计算机操作系统的服务。