OpenStack 架构设计指南

当前最新 (2015-06-01)

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。

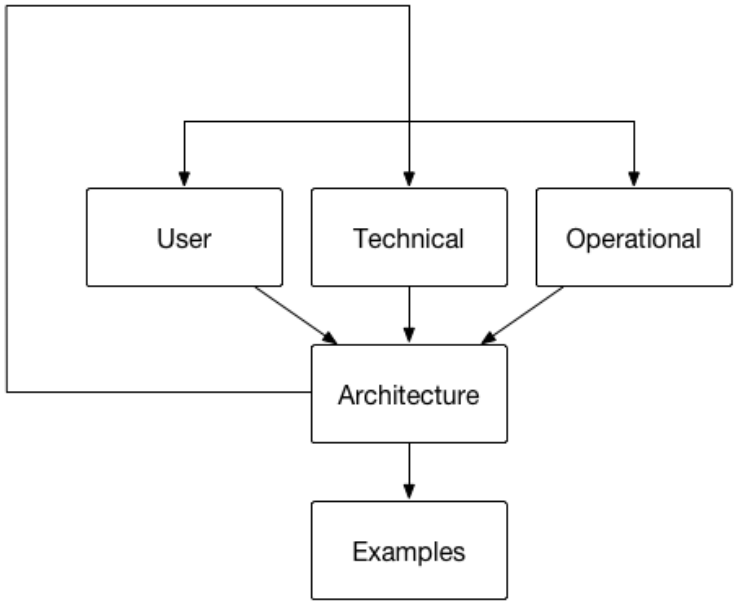
我们非常感谢VMware的盛情款待，以及我们的雇主，Cisco, Cloudscaling,Comcast,EMC,Mirantis,Rackspace,Red Hat, Verizon和VMware,能够让我们花时间做点有意义的事情。尤其感谢Anne Gentle 和 Kenneth Hui，由于二位的领导和组织，才有此书诞生的机会。

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。

相关的法律制度包括了有欧盟的数据保护框架(http://ec.europa.eu/justice/data-protection/)以及美国金融业监督管理局的要求(http://www.finra.org/Industry/Regulation/FINRARules)。请咨询当地的监管机构以了解更多相关信息。

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 [103]。

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https://wiki.openstack.org/wiki/HypervisorSupportMatrix页面找到。

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 n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OpenStack 运维指南。

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 [83]。

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 [61]。

示例

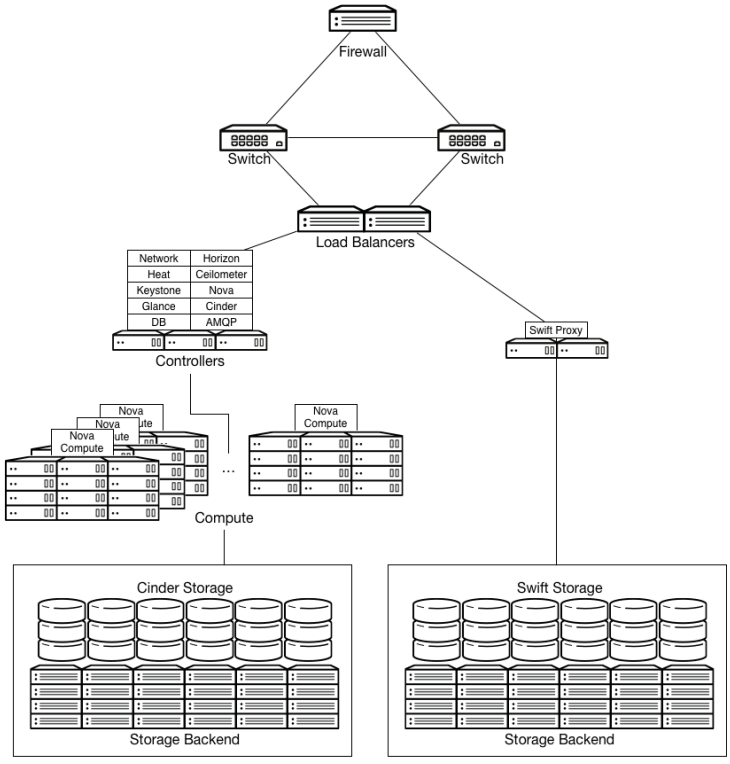
一家在线的广告公司，名称暂时保密，打算基于私有云方式运行他们的web应用，属于网站典型的架构：Tomcat + Nginx + MariaDB。 为了迎合他们的合规性需求，云基础设施运行在他们自己的数据中心。公司对负载需求有过预测，但是仍然提出了预防突发性的需求而能够灵活扩展。他们目前的环境不具有灵活的调整目标到运行开源的应用程序接口环境。目前的环境是如下面这样：

- 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)

根据用例的需求，此类云也许需要提供额外的服务，诸如虚拟机磁盘库，文件或对象存储，防火墙，负载均衡，IP地址以及虚拟本地局域网(VLAN)。计算型的OpenStack云一般不会使用raw块存储服务，因为托管在计算型OpenStack云的应用通常不需要持久性块存储。

用户需求

计算密集性负载是由他们对CPU，内存，或二者的高利用率所定义。用户的需求将决定一个计算型云须要构建成为满足预期的性能要求。

成本　开销对于计算型云来书通常不是主要担忧的因素，尽管一些组织也许会想着如何减少成本。理由现有资源来处理计算密集型任务，而不是获取更多的资源，可以有效的降低成本。

上线时间　计算型云可用于更加快速的交付产品，举例，通过构建产品和应用以加速一家公司的软件开发生命周期(SDLC)。

赢利空间 企业从计算型云获利就得构建加强计算资源的服务或产品。例如包括大数据分析(通过Hadoop或Cassandra)，或者是完成计算密集型的任务诸如渲染、科学计算、模拟现实等。

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— 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。

相关的法律制度包括了有欧盟的数据保护框架以及美国金融业监督管理局的要求。请咨询当地的监管机构以了解更多相关信息。

技术因素

下列一些技术需求需要被设计架构时纳入。

性能 如果云环境的主要技术是用于交付高性能计算的能力，那么一个计算型云的设计显然会被选择，因为它本身就被设计为主机计算密集型。

持久负载 负载可以是短时间的，也可以是长久运行的。短时间的负载比如持续集成和持续部署(CI-CD)任务，大量的计算实例被并行创建以处理计算密集型任务。结果会在实例销毁之前被拷贝出来存放到可长时间保存的存储中。长久运行的负载，例如Hadoop或高性能计算(HPC)集群，通常摄取大量的数据集，然后计算处理这些数据集，最后将结果存放到长时间保存的存储中。和短时间负载不一样的是，当计算工作结束时，它会保持空闲知道下一个任务推给它们。长时间运行的负载通常规模大而且复杂性高，所以尽力保持它们在任务之间的活跃，可以缓减构建的压力。另外一个长时间负载的例子是旧的应用通常都会持续一段时间。

存储 目标为负载的计算型OpenStack云通常不需要任何的持久性块存储(虽然一些使用Hadoop的应用，用HDFS来作为持久性的块存储)。一个共享文件系统或对象存储就可以担当起初始化数据集(s)以及服务于保存计算结果。 通过避免输入 - 输出(IO)的开销，工作负载性

能被显著增强。取决于数据集()的大小，也许扩展对象存储或共享文件系统以满足存储需求是必要的。

用户界面 和其他类型的云架构一样，计算型OpenStack云需要按需服务和自助服务的用户界面。最终用户必须可以简单而又灵活的部署计算资源的诸如电力、存储、网络和软件。这也包括扩展基础设施到大规模级别而无须中断主机的操作。

安全性 安全在业务的需求下将成为高度依赖的服务。举例，一个需要密集计算的药物发现应用相比于为处理市场零售数据所设计的云有着更高的安全需求。作为云的一般入门，关于安全建议和向导，我们在《OpenStack安全指南》这本书有更加详尽的描述。

运营因素

从运营的角度讲计算型云和通用型有着同样的需求，更多关于运营的需求可在通用型设计的章节中找到。

技术因素

在一个计算型OpenStack云中，已经部署的实例负载类型直接影响到技术的决策。举例，一个要求多个短时间运行的任务和一个要求长时间运行的任务是有着不同的需求的，尽管它们被认为都是“专注于计算“的。

无论是公有云还是私有云，都对确定的容量规划有需求，以支持动态的成长，这都是为满足用户预期的服务水平协议。确定容量规划是给定流程保持一贯的高性能的预测和费用的途径。此流程之所以重要是因为，当服务成为用户基础设施的不可或缺的一部分时，衡量一个服务的性能不再是其平均的速度，而是速度的一贯如此。

容量规划有两个方面的考虑因素：规划初始部署，规划扩张一定要保持领先用户的需求。

规划OpenStack初始化部署的典型做法是基于现有的基础设施负载和基于预期的占用。

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：

- 期望同时在线的实例数量：(超分配比例 x 核数)/每实例的虚拟核数
- 需要多大的存储：flavor 磁盘大小 x 实例数

这些比例可用于计算支撑云需要增加的基础设施量，例如，考虑一个场景是，用户需要1600个实例，每个实例有2 vCPU和50GB存储，假设其使用默认的超分配比例 16:1，结果就可以利用如下数学公司计算出来：

- 1600 = (16 × (number of physical cores)) / 2

- storage required = 50 GB × 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。

举例，想一下下面二者之间的区别，一个是支持web托管平台的管理，另外一个是运行集成测试的，会在每次提交代码就会去创建一个实例。前者创建实例的大量工作仅在隔几个月才会发生，而后者会将大的负载不断的给云控制器。实例的生命周期必须被考虑，也就是说有充足的理由不要给云控制器那么大的工作负载。

除了创建和销毁实例之外，用户还需要考虑什么时候访问服务，特别是访问nova-api和其关联的数据库。列出实例亦需要处理大量的信息，以及有那些用户在指定的频率下运行此操作，一个云用户大量的用户会显著的增加负载。有时候，这些都在无意之下就发生了。例如，OpenStaclk GUI界面中的实例列表会在每30秒刷新一次，所以浏览器如果一直保持打开状态，就会产生意料之外的负载。

考虑这些因素可以帮助决定云控制器需要多少个核。一台拥有8CPU核和8GB内存的服务器，满足于上述的计算节点就绰绰有余。

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).

”云资源计算器是一个很有用的工具，用来考核不同硬件和实例负载的结果。可从下面链接获取：https://github.com/noslzzp/cloud-resource-calculator/blob/master/cloud-resource-calculator.ods

扩展计划

当规划扩展云的计算服务时面临一个关键的挑战，那就是云基础设施要求的弹性。在过去，新的用户或客户被强制计划，而且必须提前提出对基础设施的需求，然后进入采购流程。云计算时代的用户，按照他们所需要的，及时的获得资源。所以，这也意味着规划只能交付典型常规的使用，但是更为重要的是，能够经受的住突然来临的突破性的使用。

扩展规划是一个颇为微妙的平衡行为。规划过于保守会导致想不到的超额认购，还会导致用户很不满。规划过去激进又会导致没有有效利用云，以及资金花在了一个未被有效使用的运营基础设施上。

关键是认真的监测云运行时的使用量的高峰和低谷。目的是测量那些服务可以被无障碍的交付，并非什么平均速度或云的容量。基于此信息来模拟容量结果，确保用户更加准确的决定当前和未来的云的容量。

CPU 和内存

(节选自: http://docs.openstack.org/openstack-ops/content/compute_nodes.html#cpu_choice)

在当前的一代CPU,可以达到12个核，如果是英特尔CPU支持超线程的话，核还翻倍到24个。如果购买的服务器支持多CPU的话，核的数量则是再乘以CPU的个数。超线程技术是英特尔模拟多线程的实现，用于改善CPU的并行处理能力。打开超线程可以改善多线程程序的性能。

用户是否打开CPU的超线程取决于用户实际情况。举例来说，关闭超线程有利于密集型的计算环境。分别在打开和关闭超线程的情况下运行本地负载，对之做性能测试，可以有助于对任何的情况作出更加恰当的决定。

如果用例中使用的驱动是 Libvirt/KVM hypervisor的话，计算节点中的CPU必须支持虚拟化才能提供完整的性能表现，在英特尔的CPU芯片中是 VT-x扩展，在AMD的CPU芯片中是AMD-v扩展。

OpenStack支持在计算节点中使用CPU和内存超分配。这将允许在云中运行更多的实例，可在某种程度上降低成本。OpenStack计算默认使用下面的超分配比例：

- 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)。

基于特定的用例必须选择合适的CPU和内存分配比例。

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：

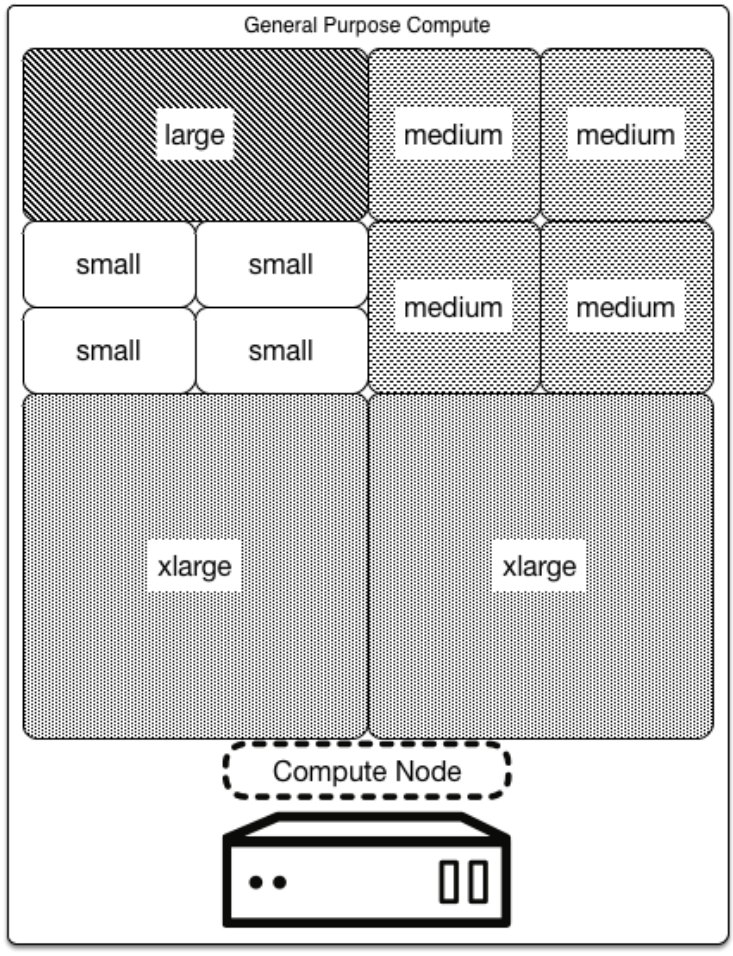
- 使用图形处理单元(GPU)，可大大有益于高性能计算任务。
- Cryptographic routines 受益于硬件随机数生成器，以避免entropy starvation。
- 给临时存储使用SSD硬盘，可大大有益于数据库管理系统，可加速其读写速度。

主机聚合是用于一组共享相似特性的主机，其中包括硬件的相似性。在一个云部署中增加特殊的硬件，其实会给每个节点增加开销。所以仔细的考虑好是否给所有的硬件都增加，或者是为那些有需要的创建单独的flavor，使用这些flavor的实例运行在定制过的计算主机中。

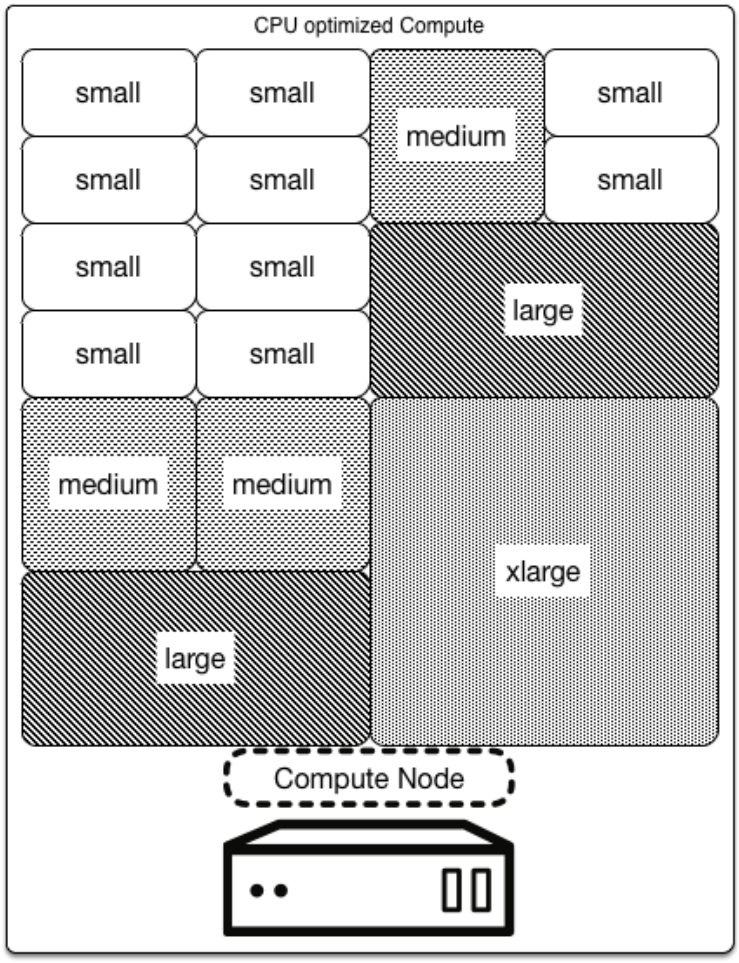
量力而行

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。

为了便利的填充虚拟机到物理主机中，默认的类型选择的结构是，在每一个尺寸中最大的类型比次之的类型大一倍。一倍的vCPU,一倍的内存，以及一倍的临时磁盘空间。次大的再比再次大的大一倍，以此类推。为通用型计算填充一个服务器的概念看起来结果就是类似这个示意图 ：



另一方面，为CPU优化的填充的服务器看起来如下图所示：



这些默认的类型非常适合于典型的为商业服务器硬件负载。欲最大化的利用，就有必要定制类型或者干脆建议一个新的，为可用的硬件更好的调整实例的大小。

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

云的基础设施不应该被共享，因为有可能发生资源不足的情形，所以要保证资源的可用并能用于负载，而且备用于提供有朝一日的大规模扩展。

批处理的持续时间取决于启动的分离任务。时间限制区间可以是秒，可以是分钟，也可以是小时，这样的结果就是增加了考虑的难度，比如预测什么时候资源会被使用？使用多久？那些资源会被用到？

安全性

此场景下的安全因素需求和本书中其它场景所探讨的是类似的。

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：

1. 公有

2. 客户机

3. 管理

4. 数据

这些安全域可以映射给分开的0penStack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0penStack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0penStack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0penStack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。包括将LDAP在部署时整合用户原有系统的话，可减少管理用户的压力。

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，请参考 OpenStack安全指南

OpenStack 组件

由于在计算型云场景中使用的负载的性质，很多组件都会用的到。下面是典型的OpenStack组件：

- OpenStack 计算(nova)
- OpenStack 镜像服务(glance)
- OpenStack 认证(keystone)

也会考虑一些特别的组件：

- Orchestration 模块 (heat)

可以很安全的假定，在此场景中给定应用的性质，这些将是非常高度自动化的部署。在此类用例中Orchestration可以大显身手。批量部署实例、自动化运行测试单元脚本化，都可以使用Orchestration来掌控。

- Telemetry 模块 (ceilometer)

使用Orchestration支持自动扩展实例离不开Telemetry和预警系统。用户不部署Telemetry模块就无法使用Orchestration模块，也许选择使用其他的解决方案来满足计费和监测的需求。

请参考: http://docs.openstack.org/openstack-ops/content/logging_monitoring.html

- OpenStack 块存储(cinder)

在计算型云中主要使用的是内存和CPU，因为其面向批处理等服务，其负载、应用、实例都是围绕此而存在，所以为每个实例增加存储并非是强制的。但这并不意味着OpenStack 块存储(cinder)不会在基础设施中使用，只是不会当作核心组件罢了。

- 网络

当选择网络平台时，要确保其可正常工作在所有的hypervisor和容器技术下，以及OpenStack的driver,后者ML2机制driver的实现。网络平台可以提供ML2机制driver的混合。

运营因素

从运营的角度讲，有许多考虑的因素会影响到计算型OpenStack云的设计。例如对API可用有强制严格的需求，理解和处理失效的场景，又或者是管理主机维护计划等。

服务水平协议(SLA)是一种合同义务，它给服务的可用性提供一个保证。因此，当设计一个OpenStack云时，兑现合同上的承诺就是实现可用性，确保冗余和弹性的水平。

- 保证API的可用性，暗示着多个基础设施服务是由高可用、负载均衡支撑着的。
- 网络运行时间的保证会影响到交换机的设计，或许要求冗余的交换机及电力供应。
- 网络安全规则的要求需要被考虑到部署流程。

如果存在SLA的话，它其中的条款必定包含，知道何时何地实现冗余和高可用(HA)。

支持和维护

为了较好的支持和维护、安装，OpenStack云管理需求运维人员理解和搞懂架构设计的内容。运维人员和工程师人员的技能级别，区分他们，取决于安装的规模和目的。大型云服务提供商，或者电信级提供商，拥有特别的培训计划，有专门的运维队伍。对于小型组织来说，就要求支持团队要精，即是工程师、设计师、运维一起做。

维护OpenStack安装需要大量的技术技能。其中一些技能也许需要调试Python日志输出的能力，这还仅仅是最基本的，另外拥有网络相关方面的理解也很重要。

考虑一下结合架构和设计的内部功能将有助于减少运维的负担。比如一些包含自动化方面的运维功能，又或者是在管理OpenStack部署中使用了第三方的管理工具的一些特别的技巧，能够前期探索一番。

监控

和其它的基础设施部署一样，OpenStack云需要一个合适的监测平台，来确保错误能够及时捕获以及正确的响应。考虑充分利用已有的监控系统，看它是否能够有效地监视OpenStack环境。OpenStack云的方方面面都需要被监测，一些特别的指标的捕获异常的重要，比如镜像磁盘使用率，访问计算API的响应时间等。

计划内和计划外服务器宕机时间

天有不测风云，服务器也会宕机，且一定会宕机。保证SLA的情况下，如何设计缩短恢复时间才是应该考虑的。恢复一个失效的主机，或者是从快照中重載实例，又或者是从另外可用的主机重新启动一个实例。这些后果都得考虑到，然后设计运行在OpenStack云中的所有应用。

设计一个计算型云时，是可以接受实例没有从一台主机迁移到另外主机的能力的。因为应用程序的开发者们希望能够在应用程序内部掌控失效状况。相反的，一个计算型云应根据业务需求部署可提供外加的弹性，在此场景下，希望外加的支撑服务已经部署，诸如给主机挂接共享存储以提供紧急恢复和弹性服务，来满足服务水平协议。

容量计划

为OpenStack云增加额外的容量是很轻松横向扩展的过程，像其它配置好的节点一样会自动挂接到OpenStack云。请记住，尽管如此，但也请在必要的时候才去增加节点到合适的可用域和主机聚合。在增加额外的节点到现有的环境中时，我们建议尽可能的使用相同(或类似)的CPU型号的主机，否则原来支持的在线迁移就可能失效。横向扩展hypervisor主机也会直接影响到网络和其它数据中心的资源，所以得考虑这种增长，例如当达到机架容量上限或需要额外的网络交换机时。

计算主机有时也需要根据需求来增加内部组件，这一过程通常叫做纵向扩展。换一个更多核的CPU，或者给服务器增加内存，可以帮助增加额外需要的容量，当然具体得看所运行的应用程序是需要多少CPU和内存(正如计算型云所期望的)。

另外一个属性是评估平均负载和增加运行在计算环境中的实例数量，即通过调整超分配比例。但这仅适合于某些环境，更改CPU超分配比例会带来负面影响而且会增加影响到其他组件的风险，记住这点很重要。增加超分配

比例带来的风险有，当一个计算主机宕机会引发除其本机运行的实例之外更多的实例会失效。在计算型OpenStack设计架构中，增加CPU超分配比例会增加破坏系统的风险，这是我们不建议的。

架构

硬件选择涵盖三个方面：

- 计算
- 网络
- 存储

对于计算型OpenStack云的硬件选择来说，必须能够响应计算密集的负载。计算型的定义就是对处理器和内存资源的极限需求。计算型OpenStack架构设计的硬件选择必须能够体现出处理计算密集的负载能力，要知道它不是存储密集也不是重型的网络负载。也许在载入数据到计算集群时会对网络和存储产生很重的负载，但是这不能当作存储型或网络型，它们有另外的考虑。

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　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　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。

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。

为一个计算型云选择硬件,重点要看服务器硬件是否可以提供更多的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 PowerEgde M1000e),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。

- 1U机架式服务器(服务器仅占用一个机架单元)要比刀片服务解决方案提供更大的服务器密度。它可以在一个机柜中安放40台服务器，相当于在一个机柜中置放32台“全宽”或“全高”的刀片服务器加上顶部空间(ToR)的交换机。但是会有双CPU插槽、多核CPU配置的限制，记住这点，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。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，非一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。

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　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。

计算型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。

存储硬件选择

对于计算型OpenStack架构设计来说，选择存储硬件不是关键，因为它并非主要标准，虽然如此，但还是比较重要的。云架构师必须考虑如下几种不同的因素：

成本 在解决方案的整个开销中，扮演主要角色的是选择采用什么样的存储架构(及存储硬件)。

性能 性能的解决方案也是一重量级的角色，可以通过存储I-O的延迟需求来测量。在计算型OpenStack云，存储延迟是个主要考虑的对象。在一些计算密集型负载中，从存储中获取数据时的最小延迟CPU都会影响到应用的整个性能。

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。记住一点，规模扩展性和通常所说的扩展性有关系，但是不一样，扩展性更加强调的是在规模扩展的过程中表现出来的性能衡量。

对于计算型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(如果可以的话)，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。

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 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 确定使用场景是否有固定的还是很容易变化的延迟。

吞吐量 为了改善整体的性能，要确保存储解决方案是整个被优化过的。虽然从一个计算型云的角度讲不太可能有从存储过来的重型数据，考虑到这是一个重要的因素就够了。

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。

当实例有高可用的需求时，或者实例需要拥有在主机间迁移的能力，使用共享存储或共享文件系统存放实例历史数据，确保计算服务在节点宕机的情况下不会中断。

选择网络硬件

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 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 网络硬件必须支持建议的网速： 1 GbE, 10 GbE, 或 40 GbE (甚至是 100 GbE).

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。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(EoR)交换机就可能发生问题，假如电力不足的话。

首先了解附加因素和用户是非常重要的，因为这些附加因素会严重影响到云的网络架构。一旦这些关键因素已经决定，适当的网络可以被设计的更好，以满足云中的工作负载。

我们建议设计网络架构时使用扩展性网络模式，让增加容量和带宽变得更加容易。一个较好的例子是使用leaf-spline模式，在此类型网络设计中，它可轻松增加额外的带宽以及横向的扩展。选择网络硬件非常重要，是否能满足端口数的需求？端口速度？以及端口密度。评估网络架构是否值得提供冗余也很重要，毕竟增加网络的可用性和冗余是要花钱的，因此，我们建议用户作出取舍，一边是钱，另一边是部署冗余的网络交换机以及主机级别的网卡bond。

软件选择

对于计算型OpenStack云架构设计的软件选择必须包括以下三类：

- 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

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	取决于所选择的hypervisor,运维人员需要经过适当的培训和掌握相关的知识才能支持所选择的操作系统和Hypervisor组合。如果运维人员没有经过培训以及相关知识的话，那么就需要提供给他们培训，这在设计中是需要额外的开销的。
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	决定什么特性是OpenStack所需要的。这通常是由所选择的操作系统-hypervisor组合来决定的。一些特性仅在某些特定的操作系统或hypervisor中存在。例如，如果某个特性无法实现，设计就得更改，因为用户需求最大。
互操作性	在整个设计中(若有必要)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

在设计和部署中选择包含什么OpenStack组件是着显著的影响。有一些组件是必选的(例如计算和镜像服务)，还有一些组件是可选的，举例来说，一些设计中不需要Orchestration模块，忽略掉Heat组件对整个系统没啥影响。还有其他情况，如架构中使用的存储组件不是OpenStack 对象存储，这就会对设计中的其它组件有着很大的影响。

对于计算型OpenStack设计体系架构来说，下面组件需要使用：

- 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)

OpenStack块存储可能不被纳入计算型设计中，由于持久性块存储并非在此类型负载中有显著的需求，此类型是指在计算型云中运行的实例。尽管如此，有一些情况需要突出性能的表现，那么块存储组件的使用也许可以改善数据I-O。

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

话说OpenStack是一个为构建云服务平台的软件项目，完全公平的收集各类软件，所以有大家都在用的其它软件包，可以在给定的OpenStack设计中按需添加。

联网软件

OpenStack网络为实例提供各种各样的网络服务。另外有许多网络软件包用于管理OpenStack组件自身。这里举一些例子，可提供负载均衡，网络冗余协议或路由守护进程，关于这些软件包更加详细的描述细节都在OpenStack高可用指南 (http://docs.openstack.org/high-availability-guide/content) 中找到。

对于计算型OpenStack云来说，OpenStack基础设施组件需要高可用。如果在设计中没有考虑硬件的负载均衡，那么就得考虑网络软件包，如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。

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。

如果任何这些软件包被需要，那么设计就可以当作额外的资源计入用的到它们的(举例，为日志聚合解决方案所需要的CPU，内存，存储以及网络带宽)。其它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

大部分的OpenStack组件都需要访问后端的数据库服务，以存放状态和配置信息。选择一个合适的后端数据库可以满足可用性和容错是OpenStack服务所必须的。OpenStack服务支持连接由SQLAlchemy所支持的任何数据库，尽管如此，绝大多数的数据库部署还是使用MySQL或其变种。我们建议提供后端服务的数据库做到高可用，使用一些可用的技术手段，多数使用的软件解决方案是使用包括Galera,MariaDB>以及多主节点复制的MySQL。

示例

欧洲理事会核物理研究中心(CERN)，以研究核物理而闻名的欧盟组织，提供粒子加速器和其他基础设施的高能量物理研究。

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

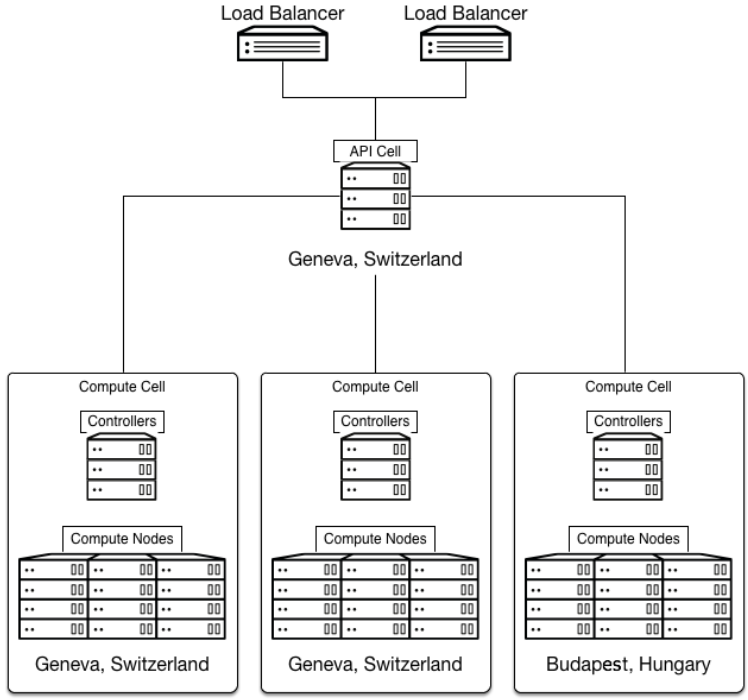
数据中心	大体的容量
	·6 PB 硬盘

为了支持越来越多的相关大型强子对撞机(LHC)实验的计算重度使用者,CERN最终选用了使用SCientific Linux和 RDO部署的OpenStack云。这些努力的目的是简化中心计算资源的管理，在2013年增加了额外的数据中心，计算能力增长了一倍，但是维护人员确依然是原有的水平。

CERN使用 cells来实现计算资源隔离以及在不同数据中心透明的扩展。这个决定意味着在支持安全组与在线迁移的权衡。其他一些细节诸如flavor需要跨cell时手动复制。尽管cell有这些缺点，但是它还是承担起了所需的规模。同时为用户暴露出一单个公用API端点。

原来的二个数据中心，每个创建为一个计算cell，第三个cell是2013年新增的新数据中心。每个cell包含了3个可用区域以进一步隔离计算资源，还有至少3个RabbitMQ 消息代理，并被配置为镜像队列的集群，这样就实现了高可用。

API cell ,是置放在瑞士的数据中心，在负载均衡器HAProxy后面，它可使用定制过的各种cell调度器去直接调用计算cell的API。定制化的调度器允许指定负载到特定的数据中心，或者是所有的数据中心，但它会根据那个cell还有内存可用来决定选择那个cell。



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。

每个cell里的MySQL数据库服务都由中心的数据库团队管理，且均配置为主/备模式，后端使用NetApp存储。每隔6小时做一次备份。

网络架构

为了集成到原有CERN网络基础设施，他们对legacy 网络(nova-network)进行了定制。以驱动的形式集成到CERN现有的数据库，用来追踪MAC和IP地址的分配。

该驱动有助于为新的实例选择一个MAC地址和IP地址，基于调度器会将实例在那个计算节点上启动。

该驱动接受调度器分配的实例到计算节点，然后在数据库中拿到预先注册列表中的MAC和IP地址分配到这个节点，然后数据库更新实例所被分配的地址。

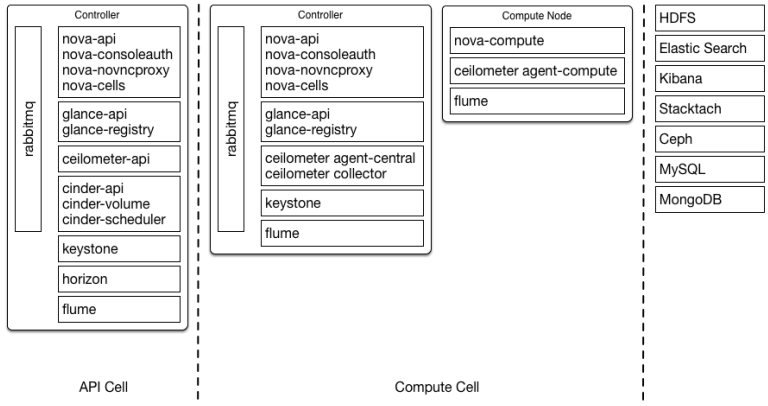
存储架构

OpenStack镜像服务被部署在API单元中且配置了版本1(V1)的API。也需要镜像注册服务。后端的存储使用的Ceph集群，大小有3PB。

由Scientific Linux 5和 6组成的一个小型的“黄金”集合，使用Orchestration工具可以将应用依次放置。Puppet用于实例配置管理和定制化，但是也有预期Orchestration部署。

监控

尽管没有直接的付费需求，Telemetry模块还是用到了，在这里拿到测量数据主要目的是调整项目的配额。也用到了一个分片的、复制的后端MongoDB。为了扩展API负载，nova-api服务被部署到子的cell中，此cell即是一个个的实例组成，Telemetry队列运行于此。这也意味着一些支撑服务包括keystone,glance-api以及glance-registry需要也可以在子的cell中配置。



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

相关的法律制度包括了有欧盟的数据保护框架(http://ec.europa.eu/justice/data-protection/)以及美国金融业监

督管理局的要求(http://www.finra.org/Industry/Regulation/FINRARules)。请咨询当地的监管机构以了解更多相关信息。

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

规模扩展性和通常所说的扩展性有关系。

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 网络硬件必须支持建议的网速： 1 GbE, 10 GbE, 或 40 GbE (甚至是 100 GbE)。

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

在设计和部署中选择包含什么OpenStack组件是着显著的影响。有一些组件是必选的，例如计算和镜像服务，还有一些组件是可选的，举例来说，一些设计中不需要Orchestration模块，忽略掉Heat组件对整个系统没啥影响。还有其他情况，如架构中使用的存储组件不是OpenStack 对象存储，这就会对设计中的其它组件有着很大的影响。

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 镜像服务(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网络 (neutron)为实例提供各种各样的网络服务。另外有许多网络软件包用于管理OpenStack组件自身。这里举一些例子，可提供负载均衡，网络冗余协议或路由守护进程(如 Quagga)，关于这些软件包更加详细的描述细节都在OpenStack 高可用指南中找到 网络控制器集群。

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。

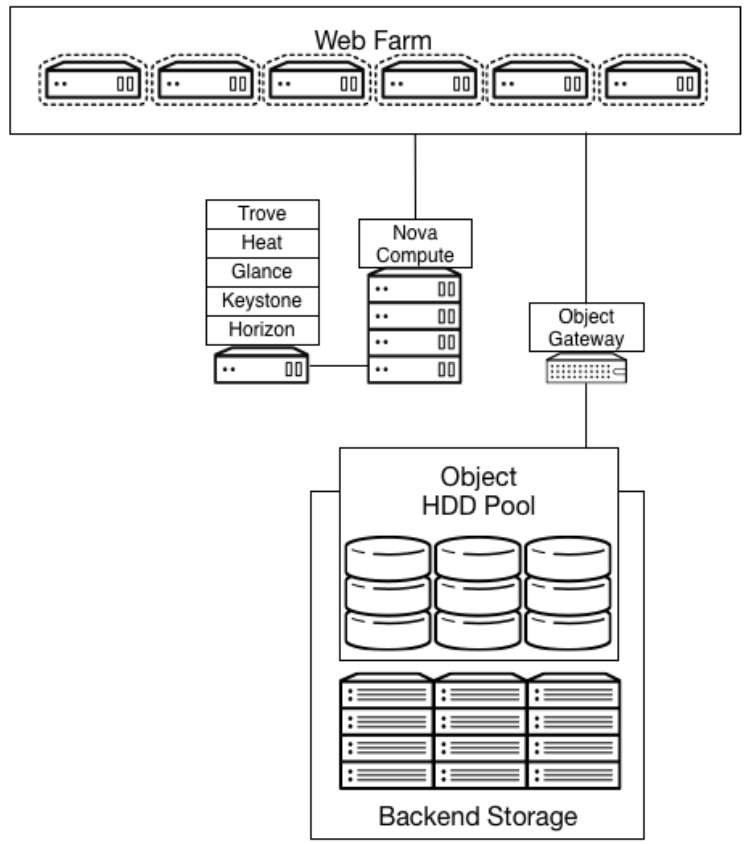
示例

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 台存储服务器，每台拥有 12x4 TB 磁盘，一共 480 TB 的总空间，使用复制的配置之后，大约为 160 TB 的可用空间。

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

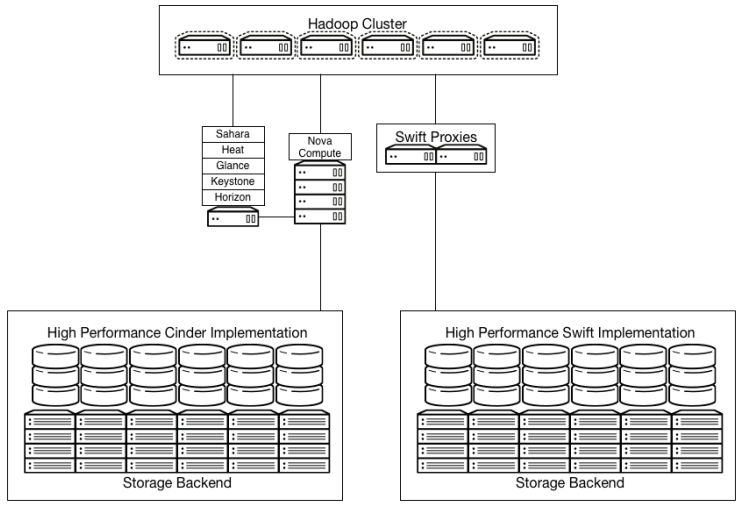
对于一些应用来说，可能需要引入第3方的缓存层以实现合意的性能。

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：



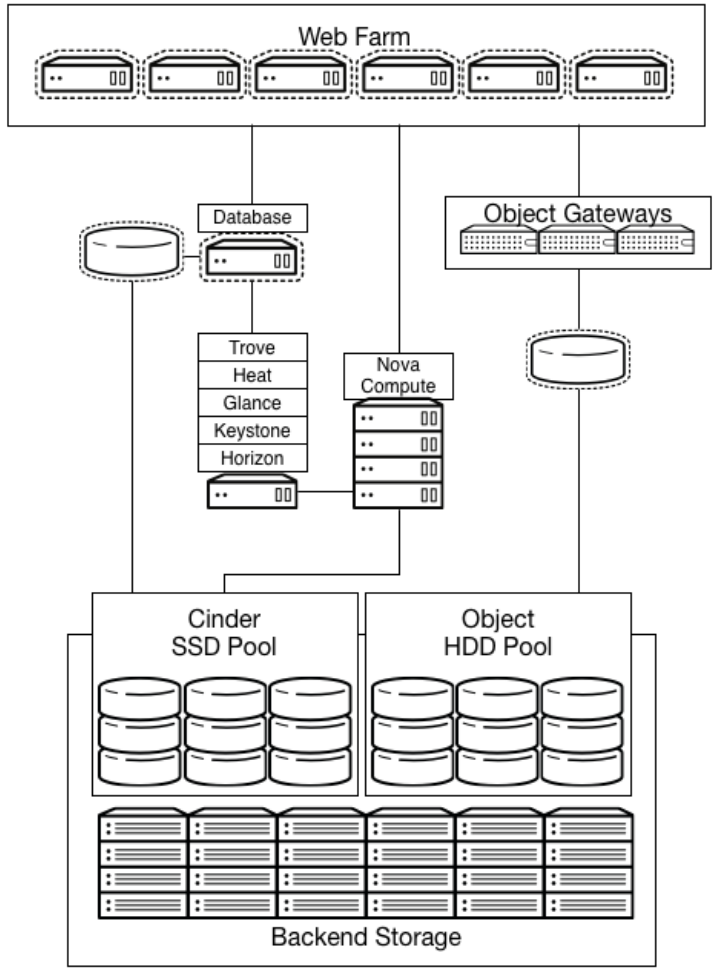
实际的硬件需求以及配置与下面的高性能数据库例子相似。在这个场景中，采用的架构使用了 Ceph 的 Swift 兼容 REST 接口，以及允许连接至一个缓存池以加速展现出来的池的特性。

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 台存储服务器，每台拥有 12x4 TB 磁盘，一共 480 TB 的总空间，使用 3 份复制的配置之后，大约为 160 TB 的可用空间。

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

由于其基于服务的本质，所有的 OpenStack 部署，在一定程度上都依赖于网络通信才能够正常工作。然而，在某些情况下，用例方案决定了网络服务比其它初级的基础设施更为重要。本章讨论了更依赖或者更关注网络服务的架构设计。这类型架构十分依赖于网络基础设施，并且需要被仔细架构以使得网络服务正常工作且可靠，从而满足用户和应用的要求。

可能的用例方案包括：

内容分发网络	这包括分发至大数量终端用户的流视频、相片以及其它任何云数据仓库。影响延迟、带宽以及实例分布的网络配置将在很大程度上影响大规模的流视频传输。并非所有的视频流都是关注用户的。例如多播视频（用于媒体、记者招待会、集团演讲、web 会议服务等等）也能够使用内容分发网络。内容分发受视频仓库的物理位置以及其与终端用户的关系所影响。性能也受后端系统的网络吞吐量，以及 WAN 架构和缓存方法的影响。
网络管理功能	用户建立起一个提供网络服务功能的云以支持诸如 DNS、NTP 或者 SNMP 等的后端网络服务的提供，用于公司内部的网络管理。
网络服务提供	云能够被用以运行面向客户的网络工具以实现对客户的服务。例如 VPN、MPLS 私有网络、GRE 隧道等等。
web 门户或 web 服务	web 服务器对于云服务来说是一个常见的应用，我们建议对于网络需求要有一定的理解。网络必须能够横向扩展以满足用户需求并以最小延迟分发网页。必须根据门户架构的细节，对内部东西向和南北向的网络带宽进行考虑。

高速及大量数据的事务性系统	此类型应用对于网络配置非常敏感。相关的例子包括很多金融系统、信用卡交易应用、交易系统以及其它需要传输非常大量数据的系统。这些系统对于网络的抖动及延迟都很敏感。同时这些系统也都有很大量的东西向及南北向网络流量，需要在这些流量中取得一个平衡使得数据传输的效率最大化。很多这类型的系统都需要连接至一个大型高性能的数据库后端。
高可用	此类应用十分依赖于恰当地改变网络大小以维护站点之间的数据复制，从而实现高可用。如果一个站点转变为不可用状态，其它的站点便能够取代该站点进行服务直到原先的站点恢复正常。恰当改变网络的大小使得能够处理所需要的负载就显得尤为重要了。
大数据	用于管理以及收集大数据(数据获取)的云对网络资源有着显著的需求。大数据应用经常使用数据的部分副本以在大规模分布式云中维护数据完整性。其它需要大量的网络资源的大数据应用包括 Hadoop、Cassandra、NuoDB、RIAK 以及其它 No-SQL 和分布式的数据库。
虚拟桌面基础设施(VDI)	此类应用对于网络拥挤、延迟、抖动以及其他网络特性都非常敏感。与视频流传输相似的是用户体验非常重要，然而，不同之处在于使用缓存不再是一个能够抵消网络问题的选项。VDI 同时需要上行与下行的网络流量并且不能够依靠缓存来让该应用给终端用户使用。
IP 语音(VoIP)	此类应用对于网络拥挤、延迟、抖动以及其他网络特性都非常敏感。VoIP 具有上下行流量对称的特点，需要网络服务质量(QoS)保证最优表现。它也可能需要一个有效的队列管理实现来保证发送。用户对延迟和抖动都非常敏感，并且能在极其轻微的情况下发现它们。
视频会议和 web 会议	此类应用同样对于网络拥挤、延迟、抖动以及其他网络缺陷都非常敏感。视频会议同样具有上下行流量对称的特点，但是除非网络是在一个 MPLS 私有网络之上，其无法使用网络服务质量(QoS)机制来提高性能。与 VoIP 一样，用户对于即使是非常轻微的网络性能问题也很敏感。
高性能计算(HPC)	此类应用较为复杂，需要对网络数据流和使用特点进行仔细考虑以处理云集群的需要。分布式计算具有东

西向流量比较大的特点，但是针对特定应用，也存在南北向的流量很大的可能。

用户需求

网络型的架构不同于通用设计。这种架构在很大程度上受到特定的受网络影响较大的应用子集所影响。一些可能影响此种设计的商业需求包括：

- 用户体验：缓慢的网页载入、低速的视频流传输以及低质量的 VoIP 会话，这些都是网络延迟影响用户体验的方式。而通常用户并不知道网络的设计和架构究竟是如何影响到他们的体验的。企业客户和终端用户都依赖于使用网络来使用他们的应用。网络的性能问题可能造成终端用户的不良体验，同样的，也会造成生产力和经济上的损失。

- 监管上的需求：网络设计需要考虑对于在网络上传输的数据的具体物理位置的管理需求。举个例子，加拿大的医疗记录是不能够传输到加拿大主权领土范围之外的。另一个值得考虑的因素是维护私有数据流的网络隔离和确保在有必要的情况下云端之间的网络是加密的。由于数据会经过各种各样的网络进行传输，对这些传输数据的加密和保护这一类的监管需求将会影响网络架构的设计。

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，数据不能够存放在其它地点的数据管理政策。

相关的法律制度包括了有欧盟的数据保护框架(http://ec.europa.eu/justice/data-protection/)以及美国金融业监督管理局的要求(http://www.finra.org/Industry/Regulation/FINRARules)。请咨询当地的监管机构以了解更多相关信息。

高可用问题

对于网络有着高需求的 OpenStack 部署，也可能有着由具体应用和使用场景所确定的对高可用的需求。金融交易系统相对于用于进行开发的应用来说，对高可用有着更高的要求。网络可用性的状态，例如服务质量(QoS)机制，能够被用于提高诸如 VoIP 和视频流传输这一类敏感应用的网络性能。

通常来说，高性能系统有对能够保证启动时间、延迟以及带宽的最小服务质量的 SLA 要求。SLA 的级别会在很大程度上影响网络架构以及对整个系统中冗余的要求。

风险

错误的网络配置	配置了错误的 IP 地址、VLAN 或者路由可能引起网络中的某个区域，或者，在最坏的情况下，整个网络无法正常工作。错误的配置可能引起破坏性的问题，因此配置应该被自动化以使得发生操作错误的机会最小化。
容量计划	必须对云网络的容量和随着时间的增长进行管理，这其中存在一个风险是网络的增长可能不足以支持负载的增长。关于容量的计划，包括购买新的网络线路和硬件，可能需要几个月甚至更多地时间。
网络调优	必须对云网络进行相应的配置以最小化链路失效、丢包、封包风暴、广播风暴以及环路等发生的可能。
单点故障(SPOF)	甚至在物理层面以及环境层面上也需要考虑高可用。假如由于只有一条上行链路或者只有一个电源而发生单点故障，系统故障将无法避免。
复杂性	一个过于复杂的网络设计将使得维护和解决问题变得困难。虽然使用能够处理覆盖网络级别或者设备级别的配置的自动化工具可以减轻这个困难，在功能之间以及专门的设备之间的非传统连接需要被很好的文档化或者直接避免使用这些连接以防止故障发生。
非标准特性	对网络进行配置使得能够获益于某厂商提供的非标准特性可能带来额外的风险。一个例子是用以在网络的聚合交换机层面上提供冗余的多链路聚合(MLAG)技术。MLAG 并非一个标准，因此，每家厂商都有其专利的对该特性的实现。MLAG 架构在不同交换机厂商之间无法彼此协作，这使得对厂商的选择受到限制，也可能在升级零件时引起延迟或者无法工作。

安全性

安全性因素常被忽略，或者在设计被具体实施之后才添加到其中去。必须在设计物理的和逻辑的网络拓扑之前就考虑安全性相关的因素和要求。需要解决的问题包括确定网络被适当地进行隔离，以及流量能够发送到正确的目的地而不用穿过不必要的地方。需要考虑的因素的一些例子如下：

- 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

另一个需要考虑的安全性弱点是网络应该如何被附加到宿主机之上。假如一个网络无论如何都需要与其他的系统隔离开来，那么就有可能需要调度该网络上的实例至专用的计算节点之上。这也可能用以避免从中转实例利用宿主机进行突破从而使得攻击者能够进入其它网络。

技术因素

当您设计 OpenStack 网络架构时，你必须考虑二层及三层网络相关的问题。二层网络上的决定包括在数据链路层所作出的那些，比如决定是使用以太网络还是令牌环网络。三层网络上则包括关于协议层所作出的决定以及使用 IP 的时间点。例如，一个完全内部使用的 OpenStack 网络能够完全在二层存在并忽略三层网络，然而，要让流量能从云中出去到另外一个网络或者因特网中，却需要一个三层的路由或者交换机。

过去几年间，网络架构出现了两种相互竞争的倾向。其中之一倾向于基于二层网络来构建数据中心的网络架构。另一种则倾向于认为云环境的网络本质上是一个缩小版的因特网。这一方式与用于开发及测试环境中的网络架构方式完全不同：因特网是完全基于三层路由而非二层交换的。

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。

- 往以太网帧中加入更多的层次只会使得联网过程变慢。这被称为节点处理延迟。

- 额外的网络特性，例如服务类别(CoS)控制或者多播，也能够与添加至 IP 网络中一样容易地添加至以太网中。

- VLAN 是一个简单的用于网络隔离的机制。

大多数信息开始并结束于以太网帧中。目前对于数据、语音(例如 VoIP)和视频(例如 web 摄像头)来说都是如此。一般认为，假如从源端到目标端的信息传输能更多地以以太网帧的形式完成，以太网就能够在网络中更加充

分地发挥其优势。尽管不能作为一个 IP 联网的替代，二层联网能够作为 IP 联网的一个强大补充。

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
- 系统进行迁移时不需要记录地址信息。尽管简单的二层协议可能在一个有几百台物理机器的数据中心中良好地进行工作，云数据中心中却有记录所有虚拟机地址及网络信息的额外负担。因为在这些数据中心中，一台物理节点支撑 30 到 40 台实例并不罕见。

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。
- 维护一系列四层网络设备以完成流量控制的需求必须得到满足。
- 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。
- 在大规模的二层网络上配置 ARP 会很麻烦。
- 所有的网络设备必须知道所有的 MAC，甚至是实例的 MAC。因此随着实例的启动和停止，MAC 地址表以及网络状态都经常发生改变。

- 如果 ARP 表的超时没有被正确地设置，迁移 MAC(实例迁移)到另外的的物理位置可能引起问题。

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。

二层网络中，所有的设备都知道所有的 MAC 地址信息，甚至属于实例的设备也如此。主干网络上的网络状态信息随着实例的启动和停止在时刻发生着改变。结果是，主干交换机的 MAC 地址表的变动会远多于正常情况。

三层架构的优势

在三层架构的情况下，实例的启动和停止不会引起路由表的变动。路由状态的改变只会在柜顶(ToR)交换机发生错误或者在主干网络链接发生错误的情况下才会发生。其它使用三层网络架构的优势包括：

- 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。
- 三层网络能够被配置成使用 BGP 联合来实现可扩展性，因此核心路由所保存的状态信息是与机柜数目成比例增长的，而非与服务器或者实例的数量成比例。
- 路由功能保证了网络核心之外的实例 MAC 和 IP 地址减少了对网络状态造成的变动。路由状态的改变只发生在柜顶(ToR)交换机发生错误或者主干网络链接发生错误时。
- 有很多经过完善测试的工具，例如 ICMP，来监控和管理流量。
- 三层网络架构允许使用服务质量(QoS)机制来管理网络性能。

三层架构的局限性

三层架构主要的局限在于没有类似于二层网络架构中的 VLAN 之类的内生的隔离机制。此外，IP 地址的层级性质也意味着实例将会位于与其所在的物

理宿主机相同的子网中。这意味着迁移实例至子网之外并不容易。基于以上原因，网络虚拟化需要在终端宿主机上使用 IP 封装 技术以及软件来实现网络隔离，以及物理层面和虚拟层面上的地址管理分离。其它三层网络架构的潜在缺点包括需要设计一个 IP 地址管理计划，而不是仅依靠交换机自动记录 MAC 地址信息，以及需要在交换机中配置内部网关路由协议。

网络建议的总结

OpenStack 由于各种原因对联网有着复杂的要求。很多组件在系统栈的不同级别上进行互动，这添加了复杂程度。数据流也一样复杂。OpenStack 中的数据可能在网络中的实例之间流动(东西向)，也可能是流出或者流入系统(南北向)。物理服务器节点有着独立于用于实例的节点的联网要求，必须被从核心网络中隔离出来以便实现可扩展性。同样的，出于安全目的，也建议根据用途划分网络，并且通过流量整形调整网络性能。

计划和设计一个 OpenStack 网络时有许多重要的普遍的技术和商业要素需要被考虑到。包括：

- 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。

在以上要求的前提下，可以作出下列关于网络设计的建议：

- 应优于二层架构，先考虑三层网络架构的设计。
- 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

设计一个网络型的 OpenStack 云有很多因素需要考虑。

OpenStack 联网方式对传统联网(nova-network)的考虑

对要实施的网络技术类型的选择依赖于很多要素。OpenStack 联网方式(neutron)和传统联网(nova-network)都有其各自的优势和劣势。他们都是有效的方式，各自所支持的用于满足不同使用场景的选项在下表中列出。

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
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

联网的一个技术性考虑因素是数据中心的交换设备必须有备用的交换机以应对硬件故障的方案。

关于交换机发生硬件故障的平均间隔时间(MTBF)的研究表明，此平均时间大概介于100,000 至 200,000 个小时之间。该数字还与>交换机所在的数据中心的环境温度相关。当采用适当的制冷及维护时，有 11 至 22 年的时间交换机才可能发生硬件故障。即使在最差情况 的通风条件及数据中心的高环境温度下，该发生故障的平均间隔时间仍然为 2 到 3 年。以上内容基于公开发表的研究 http://www.garrettcom.com/techsupport/papers/ethernet_switch_reliability.pdf 和 http://www.n-tron.com/pdf/network_availability.pdf。

大多数情况下，使用单独一台交换机的同时预留几台空闲的交换机以便替换坏了的设备，要比在整个数据中心中部署冗余的交换机的方式更加经

济。相关的应用也应该能够在不影响正常运营的情况下，容忍机柜级别的故障，这是由于网络及计算资源是比较容易供应的也相对充足。

为将来做准备：IPv6 支持

目前，一个网络方面的重要话题是 IPv4 地址即将用尽。2014 年初，ICANN 宣布他们已经开始分配最后的 IPv4 地址段给区域性>因特网注册(http://www.internetsociety.org/deploy360/blog/2014/05/goodbye-ipv4-iana-starts-allocating-final-address-blocks/)。这意味着 IPv4 的地址空间已经接近被完全分配了。其结果是，由于未分配的 IPv4 地址段的匮乏，很快我们随着应用成>熟度提高或者由于扩展目的要为应用分配更多的 IPv4 地址就会变得困难。

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 联网方式支持 IPv6，进行配置之后就可以享受到该特性带来的好处。要启用该特性，只需要简单地在网络模块中创建一个 IPv6 子网，并且在创建安全组时使用 IPv6 前缀即可。

非对称连接

当设计一个网络架构时，相关应用的网络流量特点将会在很大程度上影响总带宽的分配以及用以发送和接收流量的连接数目。为客户提供文件存储的应用将更偏向分配用于流入流量的带宽和连接，与此同时，视频流传输应用将更偏向分配用于流出数据的带宽和连接。

性能

当设计一个用以支持面向网络的应用的环境时，对应用对于网络延时及抖动的容忍程度的分析便很重要。特定类型的应用，例如 VoIP，对于网络延时及抖动的容忍性更低。当需要关注延时及抖动时，特定类型的应用可能需要对 QoS 参数和网络设备队列进行调整以保证它们的流量进入马上发送的队列之中，或者得到最小带宽的保证。由于目前 OpenStack 并不支持这些功能，所选的网络插件是否支持这些功能便也成为一个考虑因素了。

服务具体的物理位置可能影响该应用或者用户体验。假如一个应用的目标是对不同用户提供不同的内容，则它必须被设计为能够恰当地将连接指向那些特定的位置。在适当的情况下，使用多站点的部署方式。

联网可以通过两种不同的方式实现。传统的联网方式(nova-network)提供了一个平坦的 DHCP 网络以及一个单一的广播域。这种实现不支持租户隔离网络，也不支持高级的插件，但它却是当前唯一的能够使用多主机配置来实

现分布式三层代理的方式。OpenStack 联网方式(neutron) 是正式的网络连接实现，提供了一个支持多种多样的网络连接方式的可插入架构。其中包括了只使用二层网络的提供商网络模型，外部设备插件，甚至是 OpenFlow 控制器。

大规模的联网将带来一系列的边界相关的问题。关于二层网络域大小具体需要多大的确定，必须基于该域中节点的数目以及域中实例之间的广播流量的大小来作出判断。打破二层网络的边界可能要求覆盖网络或者隧道的使用。该决定是在对更小的总开销的要求和对更小的域的要求之间所作出的平衡。

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

网络型的 OpenStack 云有一系列运营上的考虑因素会影响所选择的设计。相关的主题包括但不限于，动态路由或者静态路由，服务等级协议以及用户管理的所有权等，这些都需要被考虑。

最先要作出的决定中的其中一个是对电信公司或者传输提供商的选择。在网络需求包括外部的或者站点到站点之间的网络连接时，这更是明确的。

关于监控以及警报功能，也需要作出设计决定。这可以是内部的责任，也可能是外部提供商的责任。在使用外部提供商的情况下，则可能采用服务等级协议(SLA)。另外，其它运营上的考虑因素，例如带宽、延迟以及网络抖动等都可以被包含在服务等级协议之中。

升级基础架构的能力是另外一个需要考虑的主题。随着对网络资源需求的增长，管理员将需要添加更多的 IP 地址块以及更多的带宽能力。避免对租户的服务中断的同时，管理硬件以及软件的生命周期事件例如升级、退役以及断电等，也需要被考虑到。

可维护性也是整个网络设计中的因素之一。这包括管理和维护 IP 地址，以及对包括VLAN 标记 ID、GRE 隧道 ID 和 MPLS 标记等覆盖网络的标识符等的使用的管理和维护。比如说，假如整个网络中的所有 IP 地址都要被更改，这是被称为“重新编号”的一个过程，那么设计就需要支持能够这么做。

在考虑特定的运营上的现实问题的同时，网络型的应用本身也要被考虑到。例如，即将到来的 IPv4 地址耗尽问题，到 IPv6 的迁移，以及对用以隔离应用接收或者发出的不同类型的流量的私有网络的使用等。在 IPv4 到

IPv6 迁移的情况下，应用必须遵守保存 IP 地址的最佳实践。此外，还建议避免依赖于未在 IPv6 协议中保留的或者实现上有区别的 IPv4 特性。

使用私有网络以隔离流量的时候，应用需要为数据库和数据存储网络的流量创建私有租户网络，并为面向客户的流量使用公共网络。通过这样子对流量进行隔离，可以做出服务质量和安全性方面的决定以确保每个网络都对应其所需的服务级别。

最后，还要做出关于网络流量路由方面的决定。对于一些应用来说，必须为路由采用更为复杂的策略框架。在昂贵的链路和更便宜的链路上传输流量的经济成本，加上对带宽、延迟以及网络抖动等的需求，都将被用以生成能够满足商业上的需求的路由策略。

如何对网络事件进行响应也是考虑因素之一。例如，在故障场景下某个链路上的负载如何转移到另外的链路上，也是设计中需要考虑的因素。假如没有正确计划网络的能力，发生故障的流量可能使得其它的端口和网络链路也无法正常工作，从而造成连环式的故障场景。这种情形下，转移到某链路上的流量使得该链路无法正常工作，然后又转移到之后链路上，直到整个网络的流量都停止。

架构

网络型的 OpenStack 架构与其他 OpenStack 架构场景具有很多相似之处。但是当设计一个以网络为主的或者重度依赖网络的应用环境时，有一些特别的因素需要考虑。

网络是作为用以在系统之间传输数据的媒介存在的。一个 OpenStack 的架构设计不可避免地有着对 OpenStack 的非网络部分的内部依赖以及对外部系统的依赖。根据负载场景的不同，可能与 OpenStack 内部以及外部的存储系统有着较多的互动。例如，假如是一个内容分发网络，则与存储之间的互动将是双向的。南北向上将会有很多流量出入存储阵列以存放和提供内容。另外，在东西向上也将有用于复制的网络流量。

重度依赖计算的负载也可能与网络有相互的影响。比如一些高性能计算应用依赖基于网络的内存映射和数据共享，其结果是当实例之间传输结果和数据集的时候会产生较高的网络负载。其他的一些应用也可能是高度事务性的，非常频繁的加上事务性锁，执行相应的工作，然后解锁。这也会影响到网络的性能。

有些网络方面的依赖会是在 OpenStack 环境以外的。虽然 OpenStack 联网方式已经能够提供网络端口，IP 地址管理，某种程度上的路由功能，以及覆盖网络，仍然有一些功能是其无法提供的。对于这些 OpenStack 提供不了的功能，我们就需要使用外部系统或者设备来填补功能上的空白。硬件的负载均衡器就是此类设备中的一种，需要它来分配负载或者完成部分特定的上层功能。请注意，直到 Icehouse 版本发布之时，OpenStack 中的动

态路由功能仍未成熟，且可能需要使用外部的设备或者 OpenStack 中专门的服务实例来实现。OpenStack 联网提供了隧道功能，然而该功能却只能在 OpenStack 联网所管理的范围内使用。如果需要将隧道从 OpenStack 的范围扩展至另外一个区域或者外部系统，则有必要在 OpenStack 之外使用能够将隧道或者整个覆盖网络映射到外部的隧道的其它隧道管理系统来实现隧道功能。OpenStack 目前并不提供网络资源配额功能。如果需要对网络资源进行配额管理，则有必要在 OpenStack 之外采用其它的服务质量管理机制。很多类似的情形下，将需要类似流量整形或者其它网络功能的解决方案。

根据所选择的设计的不同，OpenStack 联网方式有可能完全无法支持所需要的三层网络功能。如果选择提供商网络模式而不运行三层网络代理程序，则您必须安装一个外部的路由器来提供到外部系统的三层网络连接。

毫无疑问，在大规模部署中肯定需要用到编排服务。编排模块能够按照模板中的定义来分配映射到租户网络上的网络资源以及完成端口创建，还能够分配浮动 IP 地址。如果有需要使用编排模块来定义和管理网络资源，则建议一开始的设计就应该包含编排模块以满足用户的需要。

对设计的影响

有很多类型的因素能够影响到网络型的 OpenStack 架构。有些因素与一般的使用场景的架构所需要考虑的因素是相同的，而另外还有一些特别的与网络需求相关的负载会影响网络设计的决定。

其中一个需要决定的事情是是否使用网络地址转换(NAT)以及在哪里具体实施。假如需要的是浮动 IP 地址而不是使用固定的公共地址的话，那么就需要使用 NAT。这种情况发生在依赖 IP 端点的网络管理应用上。一个例子是 DHCP 中继需要知道真正的 DHCP 服务器的 IP。这些情况下，让基础设施能够自动将目标 IP 地址应用到新的实例之上，要比为每个新实例都重新配置传统的或者外部的系统要来得容易。

为 OpenStack 联网所管理的浮动地址而使用的 NAT 通常都在宿主机上，不过也有其它用途的 NAT 会在别的地方运行。假如出现 IPv4 地址不足的情况，有两种常见的方式来在 OpenStack 之外减轻这个问题的影响。其一是以一个实例的方式在 OpenStack 内部，或者使用外部的负载均衡方案来运行负载均衡器。内部的场景下，诸如 HAproxy 这一类的负载均衡软件，能由 OpenStack 联网的负载均衡即服务(LBaas)进行管理。具体说来，这是在使用 HAproxy 实例的双路连接将外部网络和所有内容服务器所在的租户私有网络连接起来的同时，对虚拟 IP(VIP)进行管理来实现的。使用外部的负载均衡方案的场景下，负载均衡器需要为该虚拟 IP 服务，并且通过外部的方法连接到租户的覆盖网络之中，或者通过私有地址路由到其中。

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。

云应用的负载类型会影响底层网络架构的设计。假如负载需要网络级别上的冗余，那么路由和交换架构就得适应这种需求。基于不同网络硬件的选择，硬件的性能差异，以及要部署的是哪种联网模型这些因素，有不同的方法可以提供这种冗余。比如说链路聚合(LAG)或者热备份路由器协议(HSRP)的使用。还有个考虑因素是部署 OpenStack 联网还是传统联网(nova-network)，以及如果使用 OpenStack 联网方式的话，要选择哪些插件。如果使用外部的系统，必须将联网方式配置成能够运行带有提供商网络配置方式的二层网络。比如说，可能需要使用 HSRP 来终结三层网络连接。

根据目标负载的不同，覆盖网络可能是，也有可能不是推荐的配置。当应用的网络连接是小规模的，短暂的或者非连续的，运行一个动态的覆盖网络能产生相当于它承载的网络包一样多的流量，也有可能引起足够的延迟从而产生其它的问题。同样也会影响到产生到覆盖网络上的流量的设备，在大多数部署中该设备是是宿主机。这将导致每秒网络包数和每秒产生连接的速度这些相关的指标发生性能降低。

覆盖网络还有另外一个可能使得其合适或者不合适特定的负载的选项。默认情况下，覆盖网络是以一种完全网状的形式存在的，但是由于这对于某些负载来说可能会产生多余的开销，所以有时可能要关掉这个功能。相对的，其它的负载也有可能能在这种情况下正常工作。比如说，大多数的 web 服务应用在完全网状的覆盖网络上没有什么严重的问题，但是一些网络监控工具或者存储复制的负载，在这种情况下却有可能因为吞吐量或者多余的广播流量而产生性能问题。

很多人忽略了一个很重要的设计因素：三层网络协议的选择。虽然 OpenStack 一开始只有 IPv4 的支持，但是现在 OpenStack 的联网已经支持 IPv6 以及同时使用两种协议的双栈网络。请注意直到 Icehouse 版本之时，这只包括了无状态地址自动配置的机制。但是，对无状态和有状态的 DHCPv6 和不需要使用 NAT 的 IPv6 浮动地址的支持的工作正在进行中。由于使用 IPv6 以及例如 NAT64、DNS64 或者 6to4 之类的 IPv6 和 IPv4 的相互转换机制的这些选项变得可行，使得承载某些类型的负载变得可能。这将改变单栈情景下的地址计划的需求，并且，过渡性的 IPv6 部署也能够减少对于 IPv4 地址的需求。

直到 Icehouse 版本的发布，OpenStack 对于动态路由的支持都很有限。然而，通过集成第三方的解决方案，包括网络设备、硬件节点以及实例，有很多的选择可以在云中实现路由。有一些类型的负载只需要在三层网络的终结点处设置好静态路由和默认的网关的情况下就可以很好地工作了。大多数情况下这是足够的，但是有些使用场景却需要至少一种动态路由协议的使用，如果不是要求多种协议同时使用的话。在一个 OpenStack 部署中，使得某种内部网关协议(IGP)对于实例可用，能够让服务使用单播路由注入作为地理位置或者故障转移机制成为可能。其它的应用则可能直接参

与到路由协议之中，作为一个类似 looking glass 一样的被动的观察者，或者作为一个路由反射器这样的主动参与者。由于一个实例可能具有大量的计算和内存资源，存放整个未分区的路由表并使用它来提供类似于给其它应用使用的网络路径可见性，或者监控工具等的服务，并不会是什么大问题。

路径最大传输单元(MTU)故障不怎么出名，但却难以判断。MTU 必须足够大以满足传输正常流量，覆盖网络的开销以及需要的三层网络协议。当假如外部的隧道时，MTU 包的大小被减小。这种情况下，您可能需要注意下最大可能的 MTU 大小，因为有些系统被配置为忽略或者丢弃用于路径 MTU 发现的网络包。

可调联网组件

为网络资源密集型的负载进行设计时，需要考虑包括 MTU 和 QoS 在内的，与 OpenStack 架构设计有关的可配置的联网组件。有些负载，基于其对传输大块数据的需求，可能要求比平常更大的 MTU。当为类似视频流传输以及存储复制等应用提供网络服务时，建议在可能的情况下，确保 OpenStack 硬件节点以及支撑网络设备都为巨型帧进行了配置。这使得对可用的带宽的利用率更高。对巨型帧的配置需要在整个网络包会经过的路径上都完成。假如其中一个网络组件不能够处理巨型帧，则整个路径将会恢复到使用默认 MTU。

服务质量(QoS)同样对网络资源密集型的复杂有着很大的影响，通过为可能受低网络性能影响的因而具有更高优先级的网络包提供及时的服务。在类似于 IP 语音(VoIP)这种服务中，差异化服务几乎是是保证正常运营的必要条件。QoS 还能以相反的方式用以将负载混合，避免诸如备份服务、视频会议或者文件共享等低优先级但却需要高带宽的应用阻塞其它服务正常运营所需要的带宽。可以通过将用于文件存储的带宽标记为类似“尽力而为”或者“清道夫”之类的更低优先级，以允许更高优先级的流量通过。在云中的区域可能分布在不同的地理地点的情况下，可能需要相应地作出计划，使用 WAN 优化来对抗延迟以及丢包。

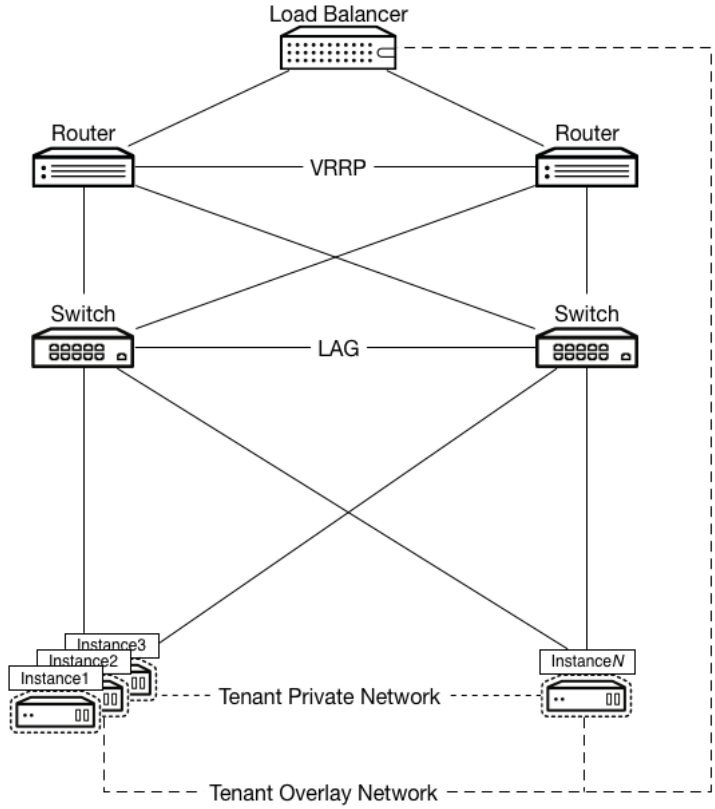
示例

一个大型的 web 应用被以能够在云中运行的想法设计出来。该应用被设计为能够以爆炸性的速度横向扩展并且会产生大量的实例。该应用要求一个 SSL 连接以保护数据安全并且要求不能够丢失到单独服务器的连接状态。

这类型负载的示例设计如下图所示。在这个例子中，硬件负载均衡器被配置成为提供 SSL 功能并且连接至租户网络以减少地址的消耗。由于其将为应用提供虚拟 IP，负载均衡器被连接至路由架构。路由器和负载均衡器都以应用的租户网络的 GRE 隧道 ID 进行配置并且都配置有属于租户子网之中

但是在地址池之外的 IP 地址。这确保了负载均衡器能够在不需要使用公有 IP 地址的情况下，与应用的 HTTP 服务器进行通信。

由于在会话关闭之前其将一直存在，路由和交换架构必须面向高可用进行设计。交换机和宿主机以及交换机之间组成网状结构，并且使用了 MLAG 以确保二层网络连接不会发生故障。路由器使用 VRRP 进行配置并且与交换机充分地组成网状结构以保证三层网络的连接。由于 GRE 技术作为一个覆盖网络被使用，联网方式被部署并配置成使用选择 GRE 隧道模式的 Open vSwitch 代理程序。这保证了所有的设备都能够连接到其它设备上，并且能够为使用私有地址到负载均衡器的连接创建租户网络。



web 服务的架构有很多选择，也有很多可选的组件。因此，这种架构也能够符合其它大量的 OpenStack 设计。然而，一些关键的组件还是需要存在以处理与大多数 web 类型应用具有相类似特点的工作。用户需要以下的组件：

- 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

在通常的认证、计算、镜像服务以及对象存储组件之外，还推荐使用编排模块以正确处理将系统根据需求进行扩展的操作。由于有自动调节规模大小的需要，此设计也使用了计量模块。web 服务在负载上通常都是不连续的，并且有着非常明确的峰值和谷值的用量特征，因此能够获益于根据网络用量自动调节实例数目大小。在网络层面上，分离的网络的结构能很好地应对位于私有租户网络上的数据库，因为这些不会引起大量的广播流量，而且可能需要连接至一些数据库获取内容。

负载均衡

在这个设计中，使用了负载均衡技术来使请求分布至多个实例。这能够很好的横向扩展至大量的实例。这使得实例不需要能够公开路由到的地址便可以运行，并且仅仅依赖于负载均衡器来使得服务在任何地方都可用。很多这类型的服务都不要求服务器直接返回。这同时简化了地址的计划以及在调节规模时对地址的利用，因为只有虚拟 IP (VIP)需要在公网上。

覆盖网络

覆盖网络功能的设计包括了使用 Open vSwitch GRE 隧道模式的 OpenStack 联网方式。在这个例子中，三层网络通往外部的路由器使用 VRRP 结对，交换机也应该使用一个具体的 MLAG 实现进行结对，从而保证了到上游路由设施的连接不会丢失。

性能调优

这类型负载在网络层面上的调整是很小的。服务质量将根据已经确定的策略被应用于此类型的负载之上形成一个中性的类别选择器，高于尽力而为型的队列但低于加速转发或者保证转发型的队列。由于这类型的应用将产生更大的包和存在时间更长的连接，对于带宽的利用可以针对持续时间较长的 TCP 连接进行优化。常用的带宽计划，即通过测定一个会话的使用量乘以预期将同时进行的会话数目并加上其他开销来进行判断的方式，在这里可以派上用场。

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。

此类型配置的支撑网络需要具备低延迟和均匀分布流量的能力。在使用这些服务的用户本地部署这些服务是较好的。通过部署相当多的应用副本来尽可能在更加靠近客户的地方处理客户的需求的多站点部署方法也常被使用。由于此类型的应用可以独立工作，它们并不需要使用覆盖网络去连接租户网络。在这种情况下使用覆盖网络也有其缺点，即是由于频繁的进行流规则的设置而引起的性能不加，另外在很大量的小网络包传输的情况下也会造成太多的额外开销，因此并不推荐使用。

对于一些情形来说，QoS 是有必要的，以保证数据的发送。DNS 查询对于其他服务的加载时间有比较大的影响，因此必须可靠并且能够提供快速的响应。因此需要在上游的网络设备上配置规则将 DNS 查询数据通过类型选择器设置为更高的优先级以保证快速转发，或者使其在队列算法中占据更佳的位置。

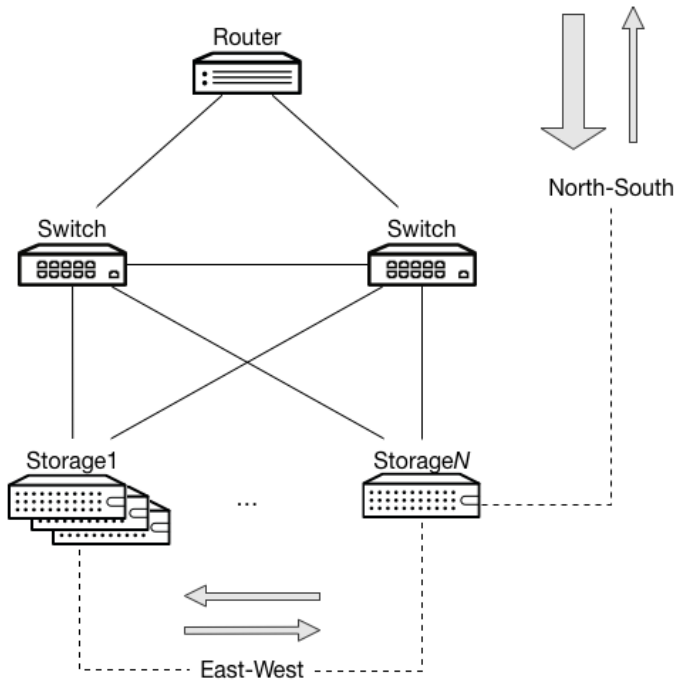
云存储

另一个常见的 OpenStack 环境的应用场景是提供云端的文件存储和共享服务。您可能认为这是一个存储型的场景，但是其在网络方面的需求确实使得其成为一个网络型的场景。

举个例子，想象一下一个云备份应用。这种类型的负载有两个明确的行为将对网络产生影响。由于这种类型的负载包括一个面向外部的服务，以及一个在内部进行复制的应用，对南北向的流量以及东西向的流量都有相关的考虑因素，如下：

南北向流量
: 当用户上传并存储内容，该内容就进入 OpenStack 环境中。当用户下载该内容，该内容就从 OpenStack 环境中发送出去。由于这个服务主要是作为备份服务使用，大多数流量就都是南向的流入环境之中。在这种情况下，配置一个非对称的主要用于下行的网络将较有好处，因为进入 OpenStack 环境的流量要比从环境中出去的流量更大。

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。

这种情况下的网络设计对可用性的依赖程度较低，而对处理高带宽的能力依赖程度较高。显然的结果是，放弃链路冗余而把链路进行绑定的效果会更好。因为这提高了可用带宽。将路径上的所有设备，以及 OpenStack 在内，配置成为能够产生并允许巨型帧通过也是有好处的。

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。

相关的法律制度包括了有欧盟的数据保护框架(http://ec.europa.eu/justice/data-protection/)以及美国金融业监督管理局的要求(http://www.finra.org/Industry/Regulation/FINRARules)。请咨询当地的监管机构以了解更多相关信息。

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。

在跨不同的站点有必要保持部署实例的一致性，这需要构建基础设施，如果为镜像服务提供的后端存储是OpenStack对象存储的话，这很轻松的就可跨多个站点建立一致性镜像的仓库。具有中央端点与多个存储节点将允许为每一个站点有一致的集中存储。

不使用中心化的对象存储将会带来运维方面的问题，例如一致性的镜像库就需要维护，包括开发复制机制以掌控传输镜像和跨多站点的镜像变化。

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。

OpenStack 高可用指南：关于如何为OpenStack 组件提供冗余性的资料信息。

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。

维护对象可用性的能力会显著影响到对象存储的设计和实现，也会影响到站点之间的广域网网络设计。

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。
- 当一个站点失效时，理解在站点之间的对象和数据复制会发生什么很重要。如果这导致队列开始建立，要考虑到多长时间这些队列到什么事情爆炸发生之前可安全完成。
- 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

理想中最好是有一个单一的认证域而不是在每个站点中分离的去实现。这当然会要求认证机制是高可用和分布式的确定持续可操作。局部认证服务也需要好好的规划。

技术因素

设计一个多区域OpenStack有许多的技术因素需要被纳入有关的账单。一个OpenStack云有很多中办法来设计，去掌控分离的应用需求，一个多区域的部署相比于单一站点的安装会带来额外的挑战，也因此将会是一个更加复杂的解决方案。

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。

站点间的链接能力描述的是在不同的OpenStack站点之间连通性的能力。这里包含的参数有带宽、延迟，线路是否是专用、以及任>何连接的业务规则认可。站点之间连接的能力和数量将决定部署时何种属性可用。例如，如果在两个站点之间有一组高带宽的连接可用，也许就想配置一个在两个站点之间分离的存储复制网络，以支持单一的Swift端点和共享的对象存储。（此技术的一个更好的例子，详细的配>置，请参阅 http://docs.openstack.org/developer/swift/replication_network.html#dedicated-replication-network）。此场景的另外一>个属性是构建专用的一组租户私有网络，在第二条线路使用覆盖网络，一个第三方的映射在彼此的站点覆盖。

应用的行为驱动着站点之间线路能力的需求，如果线路的延迟太高，某些应用使用大量的小包，比如RPC调用，就会遭遇彼此之间的通信问题和正确操作的问题。另外，OpenStack会遇到类似的问题，为了降低这些问题的发生几率，优化认证服务调用的超时时间就非常的有必要，这样可以防止认证在中心化身份服务的冲突。

另外关于容量的考虑因素，当它涉及到多站点部署网络的可用量时，以及对于租户网络的负载网络的性能。如果使用了跨zone的共享租户网络，那么将外部覆盖管理或控制器映射到一起就很迫切。非常有必要确保zone之间的ID量一致。记住，在Icehouse版本发布时，OpenStack网络还没有在安装时拥有管理隧道ID的能力，这意味着如果其中一个站点用完了ID，但是另外一个站点没有用完，租户网络将无法达到其它站点。

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。

负载均衡是安装多站点OpenStack另外一个常见的问题，它可能运行着负载均衡即服务的HAproxy实例，这些可能是本地一个特定的region，一些应用也许通过内部机制来应对这种情况，另外，也许需要外部的系统来实现，诸如全局服务负载均衡器或者anycast-advertised DNS。

取决于在设计站点时的选择的存储模式，存储的复制和可用性也是最终用户所关心的。如果一个应用有理解region的能力，那么它可能会保持通过region分离的对象存储系统，在此种情形下，用户需要在多个region访问对象的话就需要去自己去跨站点复制，基于中心化的swift代理，用户也许需要后端对象存储的复制基准时间。测试基准允许运维人员提供给用户可以理解的存储需求的总计时间或者是更改对象使之在整个环境中可用。

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。

在一个多区域部署中，存储可用性也会影响到架构。一个中心化的对象存储服务需要在region中为实例本地存取对象中频繁的使用，在对象没有建立的地方。一些应用也许为此而需要作出相应的调整。块存储目前还没有跨多个region的复制数据的方法，所以应用若依赖可用的块存储的话，需要手动的复制，由于这种限制导致在每个region都要创建重复的块存储。

安全性

多区域OpenStack安装的安全会来带额外的挑战，租户会为了安全会建立一个租户创建的网络。在多区域站点安装中也许没有需求去使用非私有网络来连接站点，这意味着流量对于第三方来说是可见的，假如应用对安全有需求，这就是个问题需要去解决，安全一VPN或者将连接加密在站点之间，在此情形下就非常有必要。

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

大多数的OpenStack安装需要一个最小集合的功能，这些包括提供认证的OpenStack 认证(keystone)、OpenStack计算(nova)、提供镜像存储的OpenStack镜像服务(glance)、提供网络服务的OpenStack 网络(neutron)、以及可能的对象存储OpenStack对象存储(swift)。多站点的建设会带来额外的组件需求，如为了region之间的协同完成某任务，中心化的认证服务是提供单点认证的必要服务，中心化的GUI界面提供单点登录和API/CLI访问的统一入口。如果有必要，一个中心化的对象存储服务也大有用处，当然这会要求安装swift代理服务。

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，自从Icehouse版本发布后，实现此特性的文档一直都在写作中，详情请浏览： 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

日志和监测对于多区域OpenStack云和其他类型没有任何显著的差别。在OpenStack运维手册中 日志和监测 章节中提到的工具依然适用于多区域OpenStack云。日志和监测可同时提供per-site基础和通用的中心位置。

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

多区域OpenStack云的部署使用region为单位，即每个站点。尽管每个站点都是相对独立安装的，但是如认证服务，是中心化的服>务，所有的站点共享使用，这就又将它们彼此联系起来了。关于升级分离的OpenStack环境，运维的高级别建议参考OpenStack>运维手册 中的(>升级)章节，获取更多详情。

1. 升级OpenStack认证服务 (keystone).
2. 升级 OpenStack 镜像服务 (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. 在每个区域升级 OpenStack 镜像服务(glance) 。

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。

关于管理配额的更多参考信息请阅读 OpenStack 运维手册书中的管理项目和用户 章节。

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。

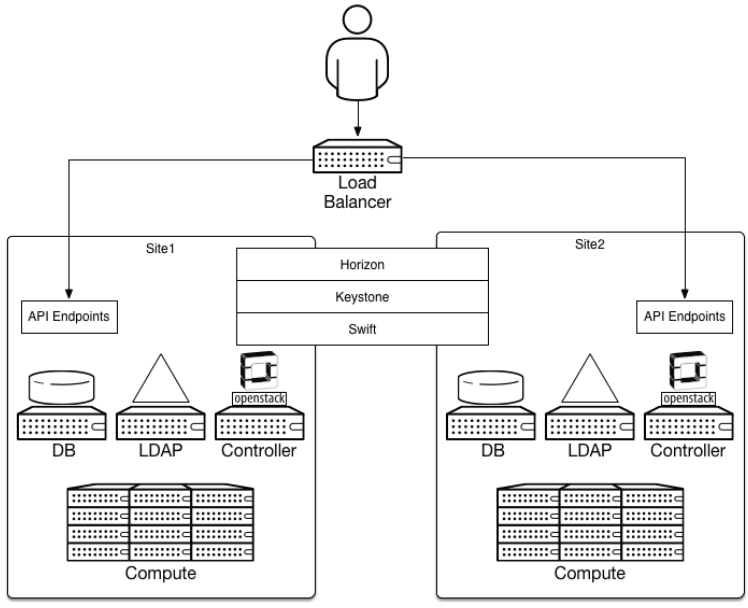
文档

用户必须自行去操作云基础设施以及在环境中部署新的资源。用户查阅用户文档来操作云，这样的话能够给出准确的文档就很重要。举例来说，默

认情况下OpenStack自动的调度实例到某个计算节点，但是，当多个region均可用时，就得用户自行决定在哪个region中调度新的实例。GUI界面显示的是用户配置的第一个region，而API和CLI工具若不指定合法的region就不会执行。因此提供文档很重要，文档中要描述region的层次以及特定region的配额等，如果用户在一个region中达到了配额的上限，OpenStack是不会自动到另外的region中构建新的实例的。文档中描述特别的例子将会帮助用户理解如何操作云，因此而减少电话呼救和现场维修。

架构

此图一个多区域OpenStack架构的高度抽象示意图。每个站点都是一个OpenStack云，但是根据架构的需要可能是不同的版本。例如，如果第二个站点的使命就是替代第一个站点，他们就可以不同。另为通用的设计是一个私有的OpenStack云是完全复制的站点将用来做高可用或灾难恢复。最重要的设计决定是如何配置存储，既可以配置为单个共享池也可以是分离的池，这取决于用户和技术的需求。



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

对于一个多区域OpenStack安装架构来说，依赖于很多的因素，其中一个主要的依赖需要考虑的就是存储。当设计存储系统时，存储的机制需要决定。一旦存储的类型决定了，如何访问就成了重点。例如，我们建议存储使用专用网络。另外一个需要考虑的是如果配置存储来保护数据。例如恢复点目标(RPO)和恢复时间目标(RTO),如何快速的从完全失效中恢复，将决定复制数据如何频繁的需求。确保有足够的存储来分配，以支持数据保护策略。

网络决定包括用于租户网络的封装机制，广播域有多大，以及内部联通的服务水平协议。

示例

基于所需要专注的负载，构建多区域OpenStack安装有好几种办法。下面的架构例子基于不同的需求，这些例子仅仅用作参考，对于部署不是硬性和快速的规则。使用前面本章前几节来帮助基于特殊的需求选择特定的组件和实现。

一个大型的内容提供商需要为分散在不同地理位置的客户交付内容，此负载对延迟极为敏感，而且还得快速响应最终用户。经过用户的回顾，技术和运营的考虑，最后决定构建有利于本地客户边缘的几个region，在此用例中，没有选择构建几个大型的、中心化的数据中心，而是架构的意图很明确，在接近最终用户的地方提供几个小型的数据中心，在此用例中，取代传统计算负载扩展的方式，而使用不同的横向扩展快速扩充应用，为了确保用户需求的快速响应时间，为贴近用户最迫切的需求，专注于通过创建更多应用的复制的扩展。此供应商会在每4个所选择的region部署两个数据中心，此设计的实现是围绕这每个远端的region复制资源的方法。Swift对象存储服务，Glance镜像服务，以及块存储在每个region都需要手动的复制，这也许有利于一些系统，比如内容服务的用例，他们的一些内容仅出现在部分而不是全部的region。建议建立一个中心化的Keystone，确保认证服务和API端点访问可轻松管理。

强烈建议安装一个自动的DNS系统，比如Designate。除非有外部的动态DNS系统可用，应用管理员会需要一个管理办法，在每个region中映射那个应用复制出现以及如何访问它们。Desginate将会将这些流程自动化，而且会填充记录到每个region的区。

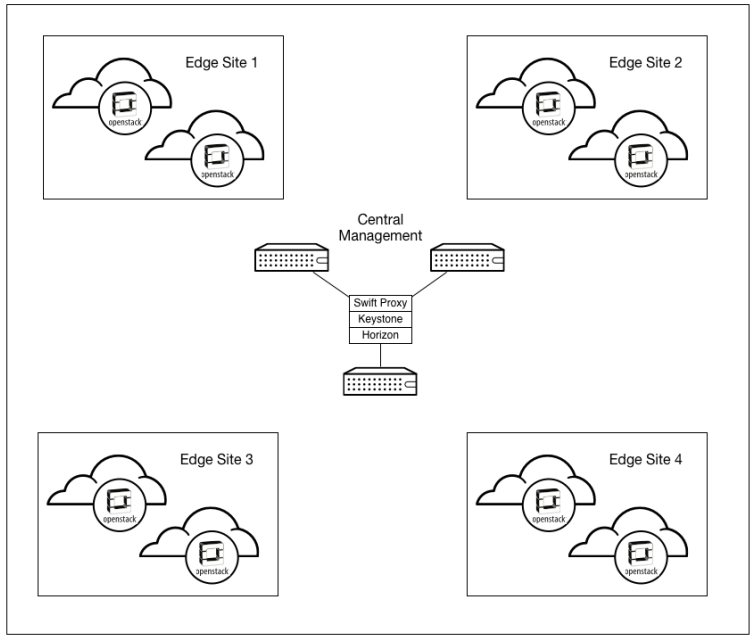
Telemetry也会部署到每个region中，虽然每个region的增长可能不一样或者使用的频率也不同。Ceilometer将运行在每个region中的监测数据，然后回报到中心地点。这在OpenStack环境中对于最终用户也好，对于系统管理员也好，都有很大的用处，使最终用户根据某些局部的负载高于其他地方

做出决定而成为可能，然后采取相应的措施。使系统管理员在每个region预测增长成为可能，而不是去盲目的同等看待所有region而去扩展它们，因此对于多区域设计是最大化的节省资源。

运行此类基础设施的一个关键决定是是否提供冗余模式。在此配置中有两类冗余和高可用模式需要被实现。第一类围绕中心化OpenStack组件的可用性，Keystone会基于三个中心化的数据中心而形成高可用，而成为中心化的OpenStack组件。这会防止任何的region失效而导致服务不可用。这也会为在每个region运行着中央存储仓库作为主要缓存，用于分发内容，带来额外的好处。

第二类容易是边缘数据中心本身。在每个边缘region地带的第二个数据中心会在靠近第一个region边上承载起第二个region的功用。这确保应用不会遭受延迟和可用性方面的性能下降。

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。
- 运行着OpenStack控制器服务的：网络服务、GUI界面、块存储以及计算服务，都运行在每三个region的本地。其他的服务：认证、Orchestration、Telemetry、镜像服务以及对象存储会安装到每个region的中心节点，通过全局来提供冗余的OpenStack控制器。
- OpenStack计算节点运行着KVM的hypervisor。
- OpenStack对象存储为静态内容提供服务，例如图片，确保所有的图片都通过标准化的存于所有的region，且使用定期复制。
- 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。
- 基于地理位置的负载均衡服务用于顾客过去使用的需求。

在三个region中都使用自动伸缩heat模板来部署应用。此模板包括：

- 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。

另外一个自动伸缩的Heat模板会用于部署分布式MongoDB shard，在全部的三个地点存储全局可用的swift容器数据。基于数据库服务所使用的和负载，以及Telemetry定义的阀值，来决定是否增加额外的shard。

这里选择三个region的原因是由于担心偶然的异常负载在单个region会引发失效。两个数据中心足够可以满足需求。

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对象存储在这里的使用，是为镜像服务提供后端支撑，而镜像存储最合适的解决方案就是全局的分布式存储解决方案，>而对象存储恰恰满

足其自身的复制机制。本土的解决方案也已经被使用，包括复制的处理，但都没有选择，因为对象存储已经是基础设施和成熟的解决方案的复杂部分。

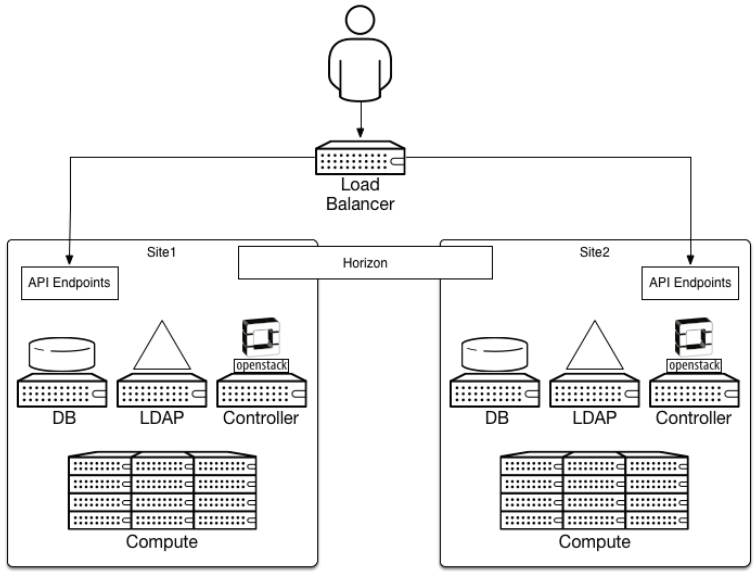
一个额外的负载均衡服务将被使用，而不是OpenStack的LBaaS,因为OpenStack的LBaaS在OpenStack中不是冗余的解决方案而且不具有地理位置的任何特征。

本地服务

多区域OpenStack部署常见的一个场景是创建内容分发网络CDN，一个应用使用本地架构会有低网络延迟和接近用户的需求，为了提供给用户极致的体验，为了减少带宽和传送的费用，因此将内容输送到更加接近客户，而替代中心化的内容存储这种需要跨国连接的高额费用。

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。

在此例中，应用利用此多区域OpenStack基于位置敏感的安装，在每个站点的计算集群中启动web服务和内容服务的实例，来自客户的请求会第一时间发送到全局服务负载均衡器，然后负载均衡器会决定客户的位置，当应用完成请求，路由会将请求发给最近的OpenStack站点。


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。

CMP有商业产品，如Rightscale,也有开源产品，如ManageIQ(http://manageiq.org),但是么有那个单个的CMP可以掌控所有的场景、所有的需求。本书的大部分章节都是在探讨OpenStack方面，以及设计OpenStack架

构时需要考虑的内容。这章内容讨论的是当选择或构建一个CMP运行混合云时，设计此架构时考虑的内容。哪怕是CMP是用户自行构建的解决方案。

用户需求

混合云架构引入额外的复杂性，特别是那些使用不同种类的云计算平台。结果就是，确保设计满足需求的同时不要增加额外的复杂性和风险显得很重要。

设计一个混合云的部署时对于商业作出的考虑包括：

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
: 赢利空间着眼于将云用来干什么。如果构建为一个面向商业用户的产品，考虑到驱动构建它在多个平台和是否使用多个平台，使设计更具吸引力的目标客户，从而提高收入的机会。

上线时间
: 使用云平台的一个常见的原因就是可以加速新产品或新应用的上市时间。一业务需求要使用多云平台，原因是他们已经在多个应用上投入了，且将此与云平台混合起来要比迁移和重构为单一平台节省很多时间。

业务或技术的多样性
: 组织已经借用基于云的服务是希望拥抱更多的业务，且利用混合云设计来跨云提供商来解决负载问题，所以没有应用是托管在单个的云服务提供商那里的。

应用增长
: 一家公司已经拥有在多云环境下的应用，且用于生产，那么也许在寻找更加省钱的集成方案，而不是迁移到单一的平台中。

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。

相关的法律制度包括了有欧盟的数据保护框架(http://ec.europa.eu/justice/data-protection/)以及美国金融业监督管理局的要求(http://www.finra.org/Industry/Regulation/FINRARules)。请咨询当地的监管机构以了解更多相关信息。

负载考虑

准确定义“负载”这个词在混合云环境的上下文中的意思显得很重要。负载可以定义为利用系统专注于做某些事，这通常被称作“用例”。一个负载可以是单一的应用也可以是一组一起工作的应用。它也可以是需要运行在多个云环境中的一组应用的多个复制。在混合云部署中，同样的负载需要等同的功能适用不同的公有和私有云环境。那么架构就需要处理它们潜在的冲突，复杂性以及平台兼容性。

一些可能用到混合云架构的用例有：

- 动态资源扩展或“突破”:另外使用多云架构的通用原因是在某时”突破“应用需要额外的资源。举个案例，一个零售商在假期销售高峰季需要额外的资源，但是又不愿意构建昂贵的云资源来仅仅满足高峰需求，因为多数时候不需要。他们也许有个OpenStack私有云，为了高峰期负载期间需要突破到AWS或其他公有云。突破的周期可长可短，可以是小时、可以是月、也可以是年为单位。

- 容灾恢复及业务连续性：在第二个站点的云中使用廉价的存储和实例管理就是一个很好的案例。公有云通常就有为这样的场景服务。

- 统一管理hypervisor及其实例：增加自服务，费用回收和透明交付正确的资源，这些在一个混合hypervisor的池中是有成本效益的。在一个混合云环境中，这是特别重要的考虑因素。寻找一个可以提供跨平台管理hypervisor和健壮的实例管理工具的云。

- 应用组合集成：一个企业级云可以交付更好的应用组合管理，通过利用自服务特性可更高效的完成部署，基于类型使用的部署规则。通常构建一个混合云架构的驱动力是，将已经是产品化的云环境无缝组合共同工作。

- 迁移场景：建立一个混合云架构的通用原因是允许在不同的云之间迁移应用。这也许是因为应用将永久迁移到新的平台，或者是因为应用为了走的更远需要被多个平台所支持。

- 高可用：使用混合云架构的另外一个重要的原因是交付需要高可用。通过使用多地点和多平台的组合，此设计可以是很高的一个可用级别，是

不可能在单一的平台下实现的。这种做法确实显著的增加了架构的复杂度。

另外思考的内容是关于负载是如何在单一的云中工作的，设计必须纳入将负载运行在多个云平台时所增加的复杂度。跨云传递负载的复杂度需要遍历应用、实例、云平台、hypervisor以及网络层。

工具考量

当设计跨多个云时，设计必须包含那些跨多个云工作的工具。一些由用户需求所驱动的工具需要具备下面功能：

- 云之间的代理: 在混合云架构中，由于至少包括两个不同的，还可能不兼容的平台，这就导致就有两种不同的成本，鉴于此，代理软件被设计为评估不同云平台之间的相对成本。这些解决方案有时可参考云管理平台(CMP)。这里举几个云管理平台的例子，Rightscale,Gravitent,Scalr,CloudForms以及ManageIQ.这些工具允许设计者根据预先定义的工作量以确定最佳位置。

- 跨云的轻松编排：CMP是用于将一切连在一起工作的工具。云编排工具是用于改进IT应用组合的管理，尤其是这些应用迁移到公有，私有或混合云平台。这些工具需要慎重考虑。云编排工具用于管理在跨多个云平台的已安装的系统中的多种组合应用。典型的企业级IT应用组合仍然是由散落在古老的硬件，虚拟化基础设施中的几千个应用，现在则由公有的基础设施即服务(IaaS)和软件即服务(SaaS)提供商来接管。

网络考虑

当选择一家CMP或云供应商时，网络服务的功能是一个重要的因素。考虑的有功能、安全、扩展性和高可用。验证和测试云端点的临界的特性，对于架构来说是很重要的任务。

- 一旦网络功能框架决定了，最小的功能测试应该被提及。这将能够确保测试期间和升级后的功能仍然可用。

- 跨多个云供应商的可扩展性也许主导了用户选择哪个底层网络框架，拥有网络API功能和验证这些功能在跨所有的云端点的持久有效都是非常重要的。

- 高可用能实现不同的功能以及可以有不同的设计。一些通常使用的模式如主-热-备，主-备，及主-主。高可用及其测试框架需要被开发以确保功能和局限都能被理解。

- 安全考虑因素包括如何在客户端和端点之间保护数据，在多个云之间的任何流经之地，以及窃听，还有拒绝服务攻击。

风险规避和管理考虑

混合云架构接入额外的风险是因为引进了复杂性、潜在的冲突或不兼容的组件或工具。尽管如此，它也同样降低了风险，可以跨多个供应商分担负载，这意味着，如一个供应商业务挂了，不会影响到整体的业务。

使用混合云架构可能增加风险的因素包括：

供应商可用性或实现细节	从公司不能提供业务到公司变动的情况下如何交付他们的服务。云架构集成设计的灵活性和可变性。矛盾的是，云不可能同时提供健壮性和灵活性。
服务水平协议比较	混合云计算环境的用户通过服务级别协议的差异可能会遇到一些损失。混合云设计需要，以适应由参与设计的各种云提供的不同服务水平协议，并且必须解决供应商的服务水平协议的实际可执行性。
安全级别	混合云环境的安全要比单个云环境的安全更加的复杂。以下内容需要关注，但也不仅仅限于这几项，它们有：应用、网络以及云平台级别。这里有一个问题，那就是不同的云平台实现安全的途径不同。而混合云的设计恰恰被言中，必须面对不同的安全实现途径。举例来说，AWS使用了一个简单模式，依赖于用户权限和防火墙混合。
提供者 API 变化	API在混合云环境中的位置非常之关键。作为供应商云服务的消费者，一个组织需掌控供应商对API的变化。云服务也许是某种原因，依赖兼容的API不再支持了，这个问题尤其在AWS和OpenStack的AWS兼容API之间非常的明显。OpenStack最初计划维护AWS API的兼容性。但是，随着时间的推移，API在功能上越走越远。一个通用的解决此类问题的办法就是尽可能的使用通用的和基础的API去最小化的减少冲突。

技术因素

一个混合云环境需要仔细的诊断，且要理解技术问题不仅是它不在企业组织的数据中心内，而且可能都不在企业组织的控制范围，在多数场景下，有必要确保架构和CMP的选择是适合的，不仅是不同的环境，还有可能的变化。在此情形下，应用均是跨不同平台且可能被定位到不同的位置。所有这些因素会影响和增加混合云架构设计的复杂性。

基于同个版本或者同个发行版的OpenStack不会因为云平台的兼容性而引发问题，这是唯一的情况，否则不兼容的情形必然会发生。

应该为云尽可能的较少不兼容的问题，其中一个办法就是完全使用同一个版本的OpenStack,哪怕是使用不同的发行吧。越新的发行版，不同版本的不兼容性越少。这得归功于OpenStack社区所建立的主动定义了核心功能需要在支持的版本中向后兼容。DefCore定义了基本功能，每个发行版必须支持，否则的话那还叫什么”OpenStack “。

一些供应商，在他们的发行版中增加了商业的定制，如果应用或架构中用到了这些特性，就会导致迁移到或使用其他类型的环境很困难。任何人考虑旧版本OpenStack互操作性，在Havana时代尤其是，在版本之间的互操作性功能确认之前要仔细的考虑。旧的版本之间的内部差异要远远好于不同平台，比如OpenStack与亚马逊web 服务或者是OpenStack与微软的Azure。

如果从一开始就纳入使用不同的云平台情况就会更好的预测。如果另外的云不是基于OpenStack的话，就几乎没有什么兼容性可言，CMP工具就需要计入掌控无数不同操作和服务的实现。一些情况会增加他们的不兼容性，包括在云中使用不同的方法：

- 部署实例
- 管理网络
- Treats 应用
- 实现服务

容量计划

许多组织切换到混合云系统的一个主要原因是为了扩容但是又不想投入太多钱。尽管如此，容量计划在设计一个OpenStack安装时仍然有必要，即使是已经有了外部的云。

特别地，当为一个有偶尔爆发能力的内部操作云设计时，整个的容量和负载的置放都需要计入考虑。一个长期的容量规划，诸如设计一个随时间推移而逐步增长，以放置被外部昂贵的云吞噬并占据。为避免此种情形，请认证思考将来的应用、容量需求以及进行合适的增长规划。

容量规划的一个缺点就是不可预测。很难预测特定应用的负载可能承载的量，例如用户数量的波动或者是应用遇到的意外的增加。使用术语vCPU,RAM，带宽或其他资源来定义应用需求和进行合适的规划是可能的，但是其他的云未必有相同的元素或者相同的超额认购率。

超额认购是一种模拟多于其本身物理容量的方法。举例说明，一个物理hypervisor节点拥有32GB内存托管24个实例，每个实例赋予 2GB内存，只要不是每个实例全部并行的达到满2GB，这样就不会出现什么问题。然而，一些主机极限的使用超额认购，结果就是，性能经常 是不一致的。如果可能的话，确定每个主机的超额认购率和容量规划相适应。

安全性

一个混合云环境的本身特性就是对整个基础设施没有完全的控制权。安全就成了一个强烈的需求，因为在云中的数据或应用超越了企业组织的控制范围。当规划一个混合云环境以及它的容量时，安全域就成了一个重要的范畴。一个安全域包括用户，应用程序，服务器或网络共享公共信任需求，且满足系统的期望。

安全域有：

1. 公有

2. 客户机

3. 管理

4. 数据

这些安全域既可以单独的映射到企业组织的安装也可以合并，举例说明，一些部署拓扑将客户和数据域混合到一个物理网络，另外一些拓扑会将他俩分开不同的物理网络。无论那种情景，云运维人员需要意识到合适的安全关注。安全域需要映射特别的OpenStack部署拓扑，域以及他的信任需求依赖于云实例是否是公有？私有？还是混合？

公共安全域是一个完全不可信任的区域，对于云基础设施来说。可参考的对象就是整个互联网，或者是说企业组织对这个网络没有任何权力。此域须永远认为是不可被信任的。当考虑混合云部署时，任何流量经由多个云之间的，都被认为是在此安全域，因此不可信。

典型的在单一数据中心用于实例对实例的流量，客户机安全域掌控的是在云中的实例所生成的计算数据，而不是操作云的诸如API调用所支持的服务。公有云提供商用于混合云配置是企业组织无法控制的，私有云提供商对实例的使用或什么人被容许访问某实例没有绝对的控制，所以认为此域是不被信任的。私有云提供商可认为此网络作为内网而因此可信任，也仅强调可控制的地方，实例和租户是可信任的。

管理安全域是位于服务的内部操作，通常可被称做“控制面板”，在此域的网络传输有私密数据诸如配置参数、用户名、密码等。部署在企业组织的防火墙后面，此域被认为是可信任的。在一个公有云模式是属于架构的一部分，这需要评估公有云提供者理解控制的程度。

数据安全域是主要涉及有关的信息的OpenStack存储服务，通过此网络的多数数据具有高完整性和保密性需求，且依赖于部署的类型它们还有高可用的需求。此网络的信任级别非常依赖于部署的决定且类似的是不会赋予默认的信任级别的。

务必仔细考虑好管理用户的系统，无论是操作或采用公有/私有云。身份服务允许LDAP是认证流程的一部分，若包含了此系统，则可以将OpenStack的用户管理去掉了。请注意当采用了第3方云来浏览认证的属性，适用于安装，以帮助管理和保持用户认证的一致性。

介于在客户机器和API端点之间传输的是用户名称、密码以及生成的令牌，强烈建议在API服务后端放置SSL硬件终端。

往内看云组件自身，另外需要安全审查的是hypervisor，在公有云中，企业组织是无法控制选择Hypervisor的。(例如，亚马逊使用其自身开发的Xen变种。)在一些案例中，hypervisor经常会成为典型的攻击对象，称之为"hypervisor爆发"，假如没有预防的话，这将会发生！Hypervisor爆发形容的是破坏或恶意升级到hypervisor资源控制之外且获得裸金属操作系统的操作权限以及硬件资源的事件。

如果实例的安全不被认为是重要的，就没有那么多问题了。大多数情况下，企业级需要避免这种脆弱性，且避免此种情形唯一的做法就是将实例运行在公有云中。这并不意味着一个OpenStack安装操作就需要拥有所有的基础设施，建议避免的情形是硬件可共享给其他人。

其他值得考虑的就是能够提供裸金属实例，在一些场景中，有可能是通过集成私有的云即服务部署来复制一个私有云，这样企业还无须购买硬件，且不需要分享给另外的租户。也有可能是使用这样的供应商：仅为一个用户提供专用的硬件，提供裸金属公有云实例；或者是提供私有云即服务的提供商。

千万要意识到每一个云实现的服务是不同的，这点很重要！在这个云里可以保证数据的安全，在另外一个就不一定安全。一定要知道，每个云处理其组织的数据或负载对于安全的需求是不一样的。

关于OpenStack安全的更多信息，请参考OpenStack 安全指南。

量力而行

当到了采用这个环节时，能够让CMP理解是什么负载运行，在哪里运行，以及他们优先采用什么，就显得重要起来。例如，大多数的场景都希望尽可能在内部运行大多数的负载，采用其他资源则是在需要的时候才考虑。另一方面，实际情形与此确正好相反，内部云仅仅用于开发且着重强调不被使用。大多数情况是，多种场景的开销模式帮助作出决定，尽管这些分析会严重影响到内部的优先级。 最重要的是一个编程能力基础上有效地做出这些决策的能力。

Telemetry模块(ceilometer)被设计为提供多个OpenStack组件的使用信息。有两个局限性因素需要考虑：第一，如果使用的是非常大的数据(例如，监测一个巨型的云，或者是非常活跃的),Ceilometer需要在后端使用NoSQL数

据库，例如MongoDB。第二，当连接到非OpenStack云时，Ceilometer有必要监测它的使用，且要为CMP的后端提供监测数据。

性能

在设计云时，性能是主要的因素。当在混合云部署考虑此时，会遇到和多区域部署同样的很多问题，例如在站点之间的网络延迟问题。另外需要重点考虑是一个应用在另外的云中启动的速度，以及完成必要的任务所能节省的时间。这也就意味着将数据迁移到离应用最近的地方，或者相反，将应用移到离数据近来处理。也可以将应用分组来分配到单一的云中而不是多个云，这样对于那些有低延迟需求的应用有好处。这同样意味着CMP的集成能够知道那个云适合那种类型负载的应用。

对于采用组件来说，OpenStack本身的工具即可提供帮助。Ceilometer可以衡量性能，如果有必要，Orchestration模块可以用于应对需求的变动，自动配生更多的资源。记住下面这点很重要，Orchestration需要在客户端进行特殊的配置，以便由Amazon Web Services所提供的功能才能被使用。当处理其他类型的云时，有必要先考察下CMP是否支持某些特性。

组件

使用多少或什么类型的OpenStack组件取决于是不是部署专门的OpenStack云。如果是的话，所有的OpenStack组件都配得上用场，且和多站点部署一样需要考虑多个方面的问题。

也就是说，在多余一个云的使用的任何情况下，至少四个OpenStack组件是需要考虑的：

- OpenStack计算(nova):先不说部署地点，hypervisor的选择会直接影响到集成一个或多个云的难度。举例，集成基于Hyper-V的OpenStack和Azure，相比使用KVM来说要省很多事。

- 网络：无论是OpenStack网络(neutron)还是遗留网络(nova-network)一旦使用了，该网络需要在云之间连接并可相互理解的整合在一起的能力。

- 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：

镜像移植：记住，在Icehouse发布后，就没有一个通用的镜像格式用于所有的云。这意味着当在云之间运行时镜像需要被转换或重新创建。为了使事情变得更为简单，启动最小且最精简的镜像，使用部署管理工具诸如Chef或Puppet安装必须的组件，这也意味着不要使用重量级镜像去加速流程，如果同样的镜像需要重复部署，使用此技术会更有意义，而不是每次都为一个镜像部署应用。

不同的API：当使用混合云部署了不止是OpenStack(或OpenStack的不同版本)时，多数早发现的问题是无法避免的，因为API的不同。云管理平台需要知道如何掌握所有必须的版本。为解决此类问题，一些实现者构建了门户，以实现混合云环境，但是大型的专注于开发的组织会使用混合云管理SDK，如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。

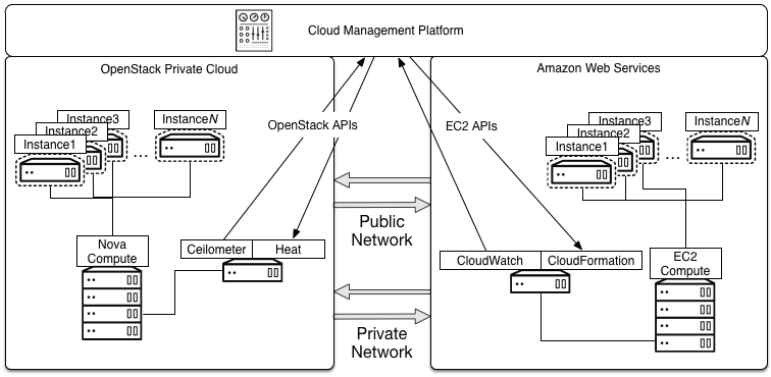
可维护性

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

一旦业务需求和应用需求都定义好了，设计混合云解决方案的第一步就是制定出预期的负载和多种云基础设施之间的依赖，以支持好业务和应用需求。将应用和目标云环境对应起来，用户需要一个解决方案架构，尽可能的是云平台之间的兼容性良好，最小化减少创建问题解决，以及填补空白

的流程。记住：在每个可用的云平台都要评估监测和orchestration API,以及选择云管理平台对他们支持的相关级别。



镜像移植

当前运行实例的云的主要负载均使用hypervisor技术实现，如KVM,Xen,或ESXi。挑战来自于这些hypervisor使用的镜像格式虽然大部分兼容于彼此，但不是完全兼容。在私有或混合云解决方案中，可强制指定标准使用同一种hypervisor或实例镜像格式，但是这不太现实。特别是如果在架构中存在一个云是公有云时，这就根本不在设计者的控制之内。

有一些转换工具比如virt-v2v (http://libguestfs.org/virt-v2v) 以及virt-edit (http://libguestfs.org/virt-edit.1.html) 在某些时候也许可以解决上述场景中的问题，但是不太适合甚至是最基本的云实例。一个替代办法是构建一个最小的操作系统镜像为新实例提供服务。这大大有助于迅速创建云实例，使用云orchestration或由CMP所驱动的配置管理工具，但这需要更多特定的模板。另外一个昂贵的选择是使用商业镜像迁移工具。问题是镜像移植并非说是一次就搞定，如果使用多个云的功能是灾难恢复、应用多样或高可用的话，镜像和实例会经常的在不同的云之间移动，而没一次移动都得进行相应的转换。

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。

考虑到迁移负载的需求也许有上层服务的依赖，原有的云平台和目标云平台也许拥有完全不同的服务，换个角度讲，用户使用不同的方法实现或者使用了完全不同的技术。举例来说，原有云交付的服务是使用NoSQL数据库服务MongoDB，而目标云不提供此服务或者使用的是关系性数据库MySQL的服务，迁移此应用将不是很现实，有大部分的数据、应用需要维护。

也许较适合混合云用例的几点属性：

- 创建跨所有云平台所实现上层服务的基准，对于平台来说不支持给定的服务，在平台的顶部创建一个这样的服务，且关注他们在云中所承载的负载。举例来说，OpenStack的数据库服务是(trove)，OpenStack支持MySQL为服务，而在生产环境不支持NoSQL数据库，无论从AWS迁移过来还是在AWS时运行，一个NoSQL的应用必须使用自动化的工具，诸如Orchestration模块(heat)，在OpenStack的顶部重新创建NoSQL数据库。

- 部署诸如Cloud Foundry或OpenShift平台即服务(PaaS)技术，是从底层云平台抽象上层服务。应用部署和迁移的单位是PaaS和利用了的PaaS的服务，并只消耗了云平台的基础设施服务。这样做的缺点的方法是，将PAAS本身则潜在地变成锁定的来源。

- 跨越所有云平台仅仅使用最基本的基础设施服务很平常，使用自动化工具来创建需要的可跨所有云平台移植的上层服务，举例来说，替代继承云平台内部的使用任何数据库服务的是，使用脚本或各种配置和应用部署工具来启动云实例以及将数据库部署到这些实例中。

网络服务

网络服务功能在多云架构中是有巨大障碍的，当选择CMP和云提供商时此因素非常的重要，考虑因素：功能、安全、扩展以及高可用(HA)。验证和不断的测试云端点的一些临界特性，用于架构设计是非常重要的任务。

- 一旦网络功能框架已经决定，起码得设计一个功能测试确保功能的兼容性。此测试可以确保在组件升级前后的功能一致性。需要注意的是随着时间的推移，多样化的云平台如果不去用心去维护兼容性，会出现同步异常，这尤其在API上是常见的问题。

- 跨多个云提供商的可扩展性要求所选择的覆盖网络框架须满足不同云提供商。网络API功能很重要，而且需要验证其跨所有云端点的与预料中的功能一致。

- 高可用(HA)有不同的功能和设计实现，举几个常见的方法，活动的-热点-备用，活动-非活动，以及双活等。高可用和测试框架需要被开发，确保其功能和局限都很好理解。

- 安全考虑，诸如数据如何在客户端和端点之间保证安全，以及任何的流量在多个云之间的传送，从窃听到 拒绝服务攻击行为等都得考虑到，业务和监管需要采取安全的方法。

数据

复制是实现保护对象存储的传统做法，在已经存在的存储架构中有许多种不同的实现，这方面的例子有同步或异步的镜像，多数的对象存储和后端

存储系统都有在存储子系统中事项复制的方法。对象存储通常会实现复制技术，而这正是云的需求而量身定制的，一个组织必须找到正确的平衡，在数据完整性和数据可用性之间。复制策略也可能影响到灾难恢复的实现方法。

复制跨不同的机架，不同的数据中心，甚至是不同地理位置的region，目的是增加确定和确保数据局部性。保证从最近或最快的存储获取数据的能力对于应用的性能表现良好是有必要的，举例来说Hadoop运行在云中，用户不仅运行着HDFS，还使用了并行文件系统，类似的提供商有日立和IBM。一个需要特别考虑的情况是，当运行嵌入式的对象存储方法毋须额外的数据复制，那样的话会带来不必要的性能问题。另外一个确保数据局部性的例子是使用Ceph,Ceph拥有一个数据容器抽象，叫做池，池可以被复制的创建，基于复制的池有些规则，用来定义数据写入本地硬件集合，那可能是主要的访问和修改点。

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。

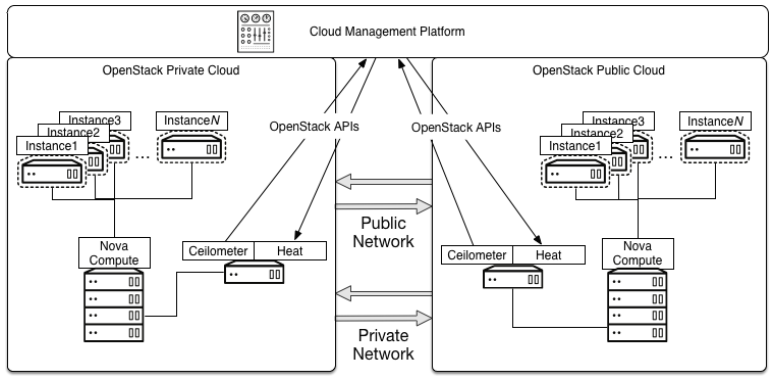
A公司已经拥有显著数量的硬件的数据中心，将负载迁移到公有云中不是可行的方式。

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。

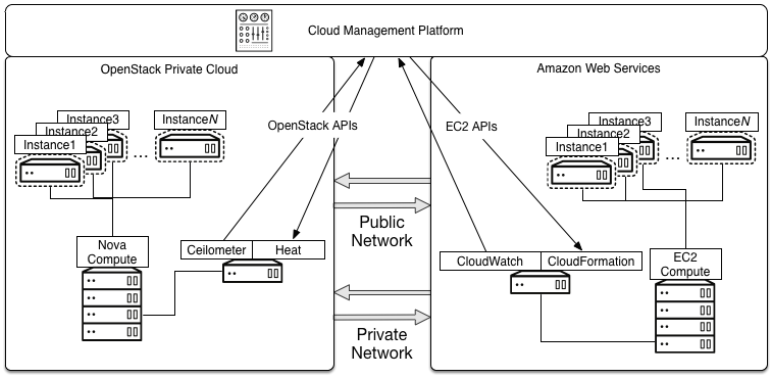
其中的私有云是一个基于OpenStack搭建的，有一个或多个控制节点，也有一个或多个计算节点。它包括了有Telemetry模块提供的计费服务，当负载增加时，Telemetry会捕获到，并且将这一消息交由CMP去处理。在私有云的容量够用时，CMP使用OpenStack提供的API,调 用编排服务来创建实例，从而响应用户的需求。当私有云容量不够用时，CMP会向公有云的编排服务API请求，让之去公有云中创建实例。

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。

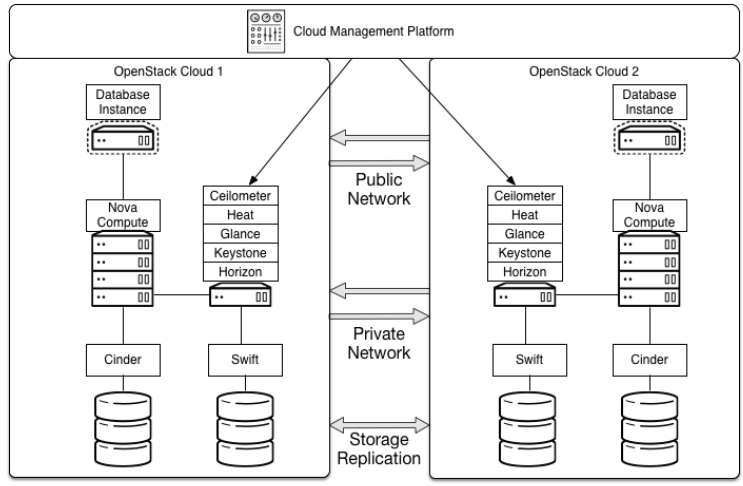
只要CMP能够使用合适的API连接到外部云提供商，工作流流程仍然和以前的场景没有多大差别。CMP的一些列诸如监测负载，创建>新的实例等动作都是一样的，但是它们必须使用合适的API调用来访问公有云。

，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。

对于块存储来说，复制多少有一些困难，且OpenStack本身不提供这样的工具。OpenStack块存储卷并非是驱动器本身而是指向后端 物理设备的一个逻辑对象，为块存储配置灾难恢复就得同步备份，就得是高级别的数据保护，但是异步备份只能是为延迟敏感的替代方案， n对于一步备份来说，块存储API须能够做到导出数据以及特别卷的元数据，所以它可以迁移和复制到任何地方。关于此方面更多信息请参>考 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

一个可大规模扩展的架构被定义为如下形式的云架构实现：其要么是一个非常大规模的部署，例如商业性的服务提供商所建造的那种云架构；要么是一个拥有能够支持用户对大量云资源进行请求的能力的云架构。比如说一个在其中，即使是同时请求对 500 个或者以上数目的实例进行服务也并不罕见的基础设施，就是这样的一个例子。在一个可大规模扩展的基础设施中，这样子的请求甚至都不用完全消耗所有可用的云资源便可以被满足。虽然部署这样一个云架构的资金成本可能高到令人望而却步，因而只有少数的组织能够冲在最前面进行尝试，很多其他的组织也都在为未来做部署这种类型云架构的计划。

可大规模扩展的 OpenStack 云的设计提出了一系列独特的挑战以及相关的考虑因素。这样一个云是用以服务不确定范围的潜在使用场景或者功能的，因此大多部分的考虑因素都与通用类型的云架构设计相类似。通常情况下，可大规模扩展的云为专门的负载工作进行设计和定制的情况是很少见的。一般来说，可大规模扩展的 OpenStack 云是作为商业性质的公有云服务产品构建的，因为单个私有组织很难得有这样多的资源或者如此规模的需求。

可大规模扩展类型的 OpenStack 云提供的服务包括：

- 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

如同通用型的云，在可大规模扩展的 OpenStack 云中部署的实例并不一定使用云服务的特定某一个方面的资源(计算、网络或者存储)。随着云规模的扩大，云上所支撑的负载的增长可能对所有的云组件都造成一定压力。额外的压力还可能来自需要对诸如数据库和消息代理这类型的基础设施进行支持。为这样一种云进行的架构设计必须考虑如何到如何处理这些性能压力，并且不对用户体验造成负面的影响。

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。

相关的法律制度包括了有欧盟的数据保护框架(http://ec.europa.eu/justice/data-protection/)以及美国金融业监督管理局的要求(http://www.finra.org/Industry/Regulation/FINRARules)。请咨询当地的监管机构以了解更多相关信息。

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 [103]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(Region)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。

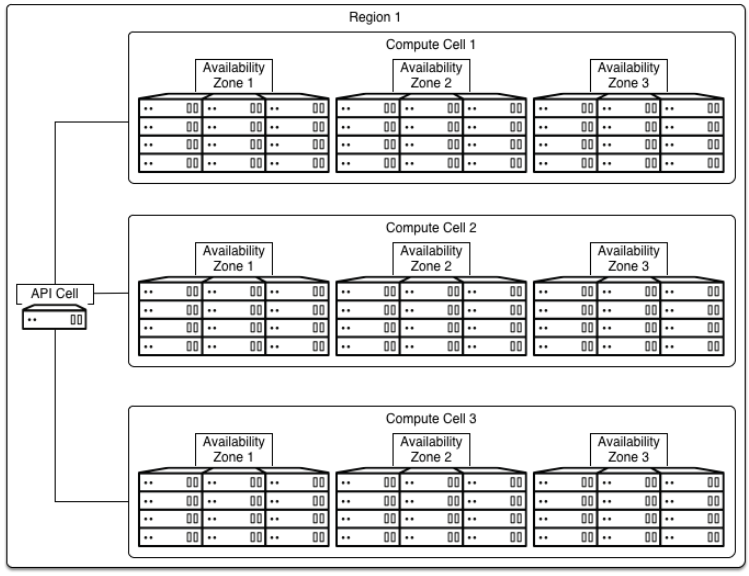
可用域

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，主机聚合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。

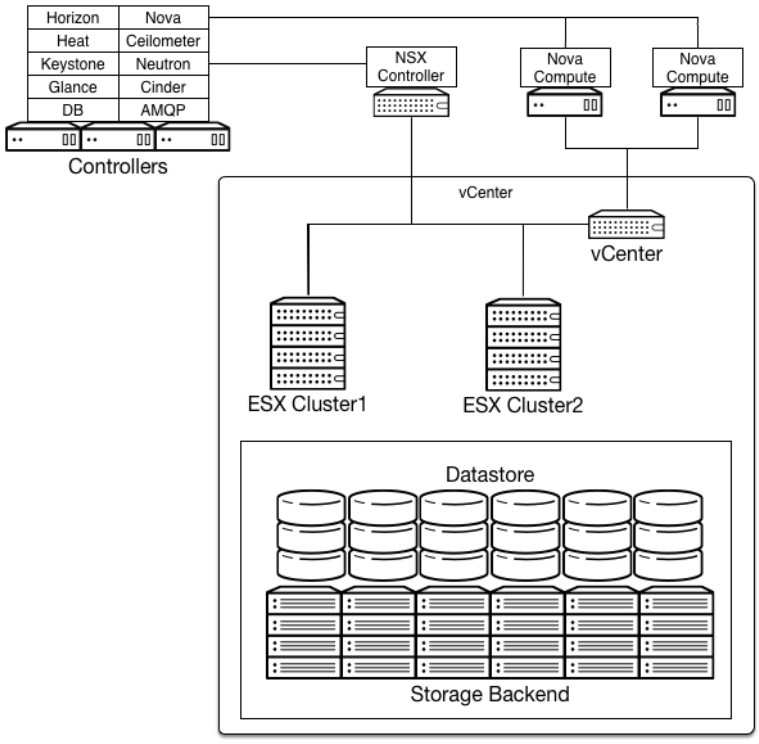
- 特殊的网络应用：此节介绍运行可能涉及直接从网线上读取数据包或者参与路由协议的面向联网的软件。
- 软件定义网络(SDN)：这种场景详细介绍了在 OpenStack 之中运行 SDN 控制器以及 OpenStack 加入到一个软件定义的网络中的情况。
- 桌面即服务：这是为希望在云环境中运行虚拟桌面环境的组织所准备的，在私有云以及公有云的情景下都可用。
- OpenStack 上的 OpenStack：一些机构认为通过在一个 OpenStack 部署之上运行 OpenStack 的方式来构建多层次的云有其技术上的意义。
- 专门的硬件：有一些非常特别的情况可能需要在 OpenStack 环境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二层上加入一个网络。

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二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

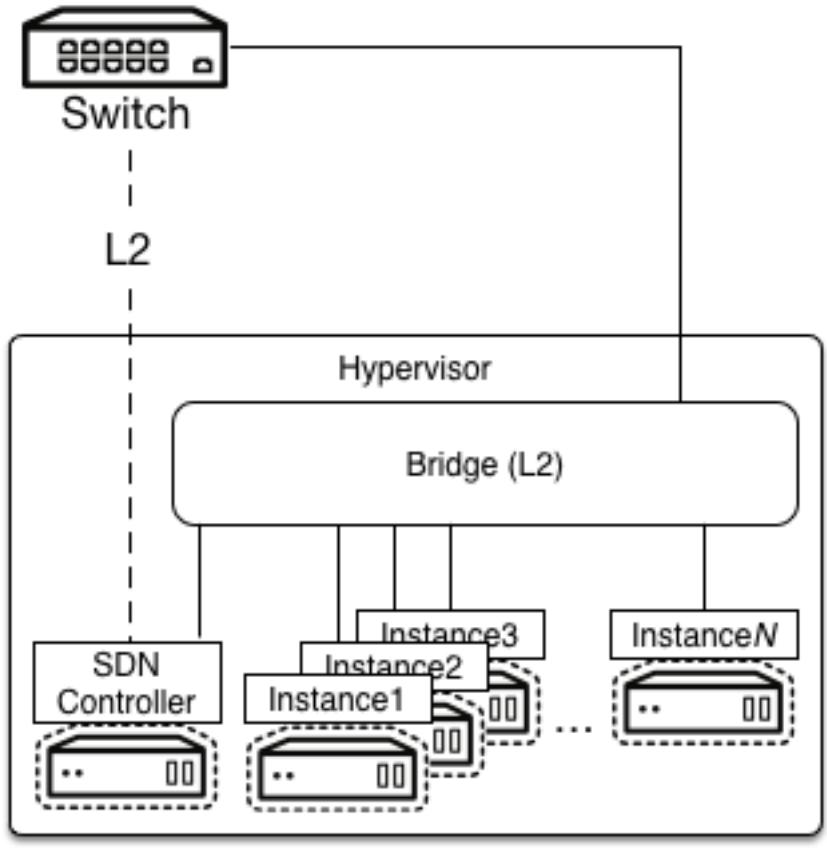
相对来说，SDN 是一个比较新的，仍未被标准化的概念，所以 SDN 系统可能来自很多不同的具体实现。因此，一个真正意义上示范性架构是目前无法给出的。相反的，我们只能够分析当前或者目标 OpenStack 设计中的各种不同，并确定哪些地方将会出现潜在的冲突或者还存在差距。

可能的解决方案

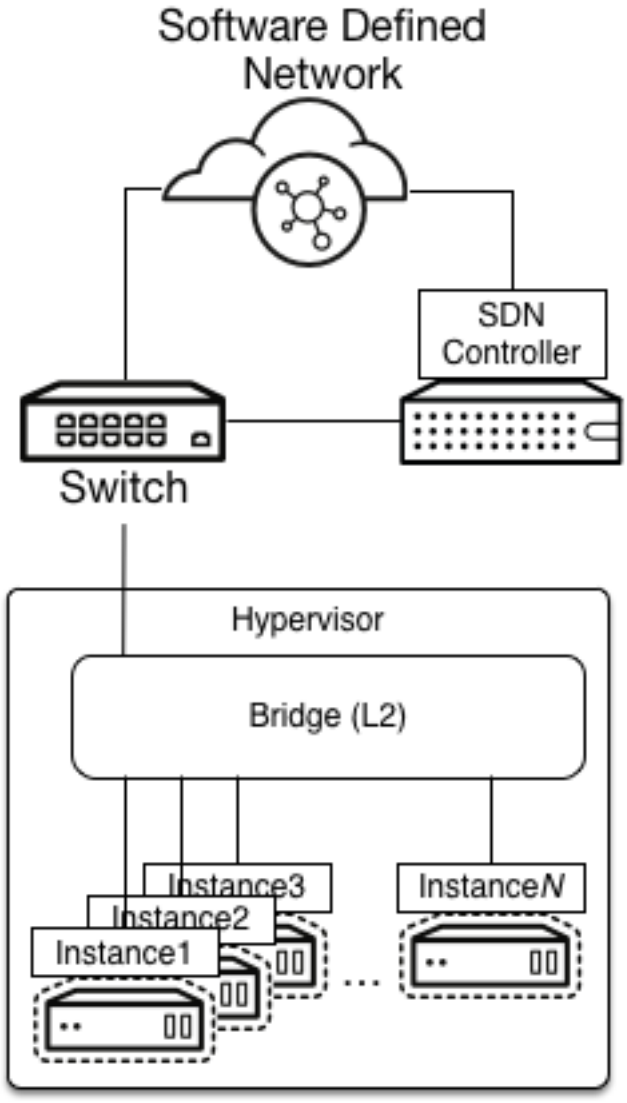
如果一个 SDN 的实现由于它需要直接管理和操作交换机因而要求二层网络的连接，那么就不建议运行覆盖网络或者三层网络的代理程序。假如 SDN 控制器运行在 OpenStack 环境之中，则可能需要创建一个 ML2 插件并且将该控制器实例调度到能够连接至能够直接与交换机硬件进行通信的租户 VLAN。另一个可能的方式是，基于外部硬件设备的支持情况，使用一端终结在交换机硬件之上的网络隧道。

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：当工作开始时成百成千的用户同时登录时发生的情况，影响存储的设计。
- 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 宿主机的兼容性

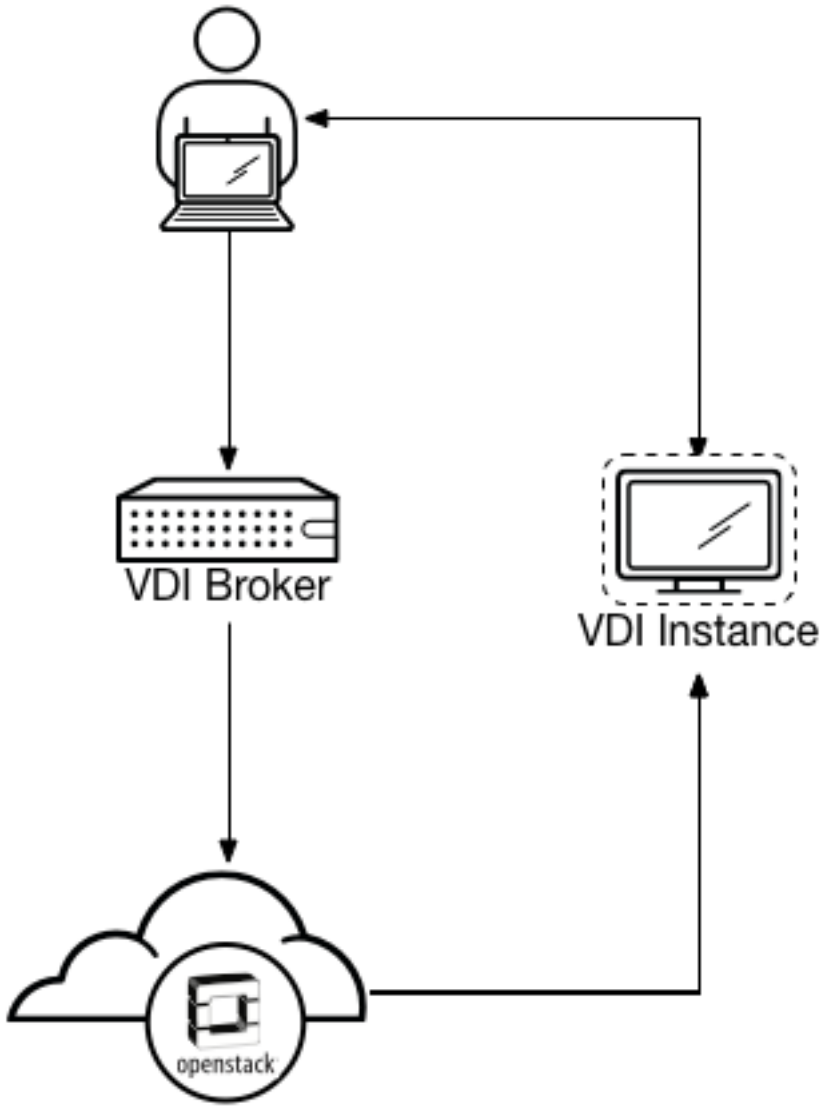
代理

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瓦解，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。

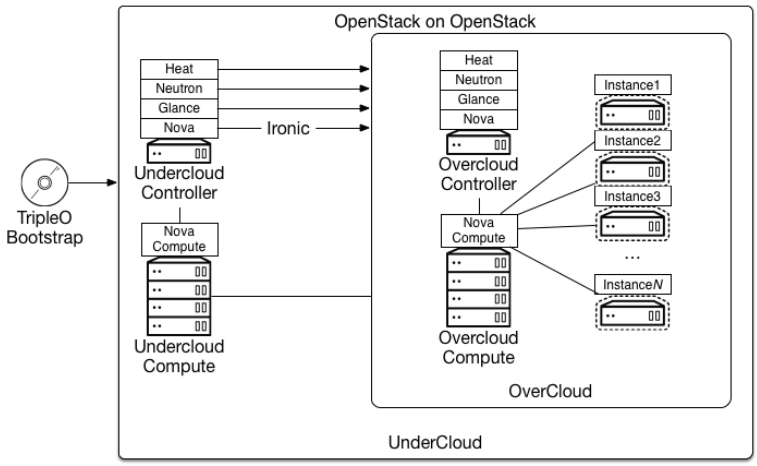
目前，OpenStack-on-OpenStack 项目(TripleO)能够解决这个问题，然而，这个项目并未完全覆盖嵌套 OpenStack 的场景。请参考 https://wiki.openstack.org/wiki/TripleO.

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。

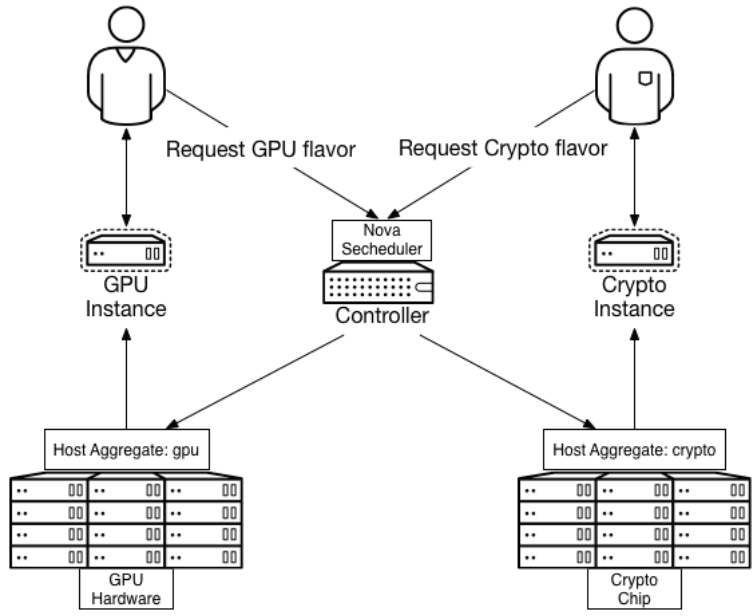
挑战

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

欧盟的数据保护框架：欧盟关于数据保护法律的指导>意见。

IPv4 地址正在耗尽：解释了如何从 IPv4 地址迁移到 IPv6，以及这个情况是不可避免的。

以太网交换机可靠性：关于以太网交换机可靠性研究的白皮书。

金融业监督管理局：美国金融业监督>管理局的要求。

镜像服务属性键：Glance API 的属性键，允许管理员附加自定义的特性至镜像之上。

LibGuestFS 文档：LibGuestFS 官方文档。

日志和监控：OpenStack 官方的运维文档。

ManageIQ 云管理平台：一个开源的能够管理多个云的管理平台。

N-Tron 网络可用性：关于网络可用性的研究白 皮书。

嵌套的 KVM：关于如何在 KVM 之上嵌 套 KVM 的日志文章。

Open Compute 项目：Open Compute 项目基金会的任务是设计以及传播为 可扩展的计算而生的最有效的服务器、存储以及数据中心硬件的设计方案。

OpenStack 实例类型：OpenStack >官方文档。

OpenStack 高可用指南：关于如何为 OpenStack 组件提供冗余性的资料信息。

OpenStack 宿主机类型支持表格: 关于在 OpenStack 对各种宿主机和特性的支持情况的表格。

OpenStack 对象存储(Swift)复制参>考：Swift 复制的开发者文档。

OpenStack 运维指南：OpenStack 运维指南提供了关于部>署和安装 OpenStack 的相关信息。

OpenStack 安全性指南：OpenStack 安全性指南提供了关 于强化 OpenStack 部署的安全性的相关信息。

OpenStack 培训服务市场：提供关于 OpenStack 的>培训及提供此类培训的厂商的信息。

PCI 穿透：扩展服务器/宿主机功能的 PCI API 补丁，该补丁使得宿主机能够为实例以及计算节点显示 PCI 的信息，并提供了用以显示 PCI 信息的资源入口。

TripleO：TripleO 是一个程序，其目标是使用 OpenStack 自身提供的设施作为基础安装、升级以及运营另外的 OpenStack 云。

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

关于OpenStack文档，请访问 docs.openstack.org。

为文档提供反馈，请使用<openstack-docs@lists.openstack.org>加入OpenStack 文档邮件列表，或者报告 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

- 基于openSUSE 13.1 或 SUSE Linux Enterprise Server 11 SP3的安装向导
- 基于Red Hat Enterprise Linux 7, CentOS 7, 以及 Fedora 20的安装向导
- 基于Ubuntu 14.04的安装向导

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
- 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)
- OpenStack 块存储服务 API v2 参考
- OpenStack 计算服务 API V2 以及其扩展参考
- OpenStack 认证服务 API v2.0 参考
- OpenStack 镜像服务 API v2 参考
- OpenStack 网络服务 API v2.0 参考
- OpenStack 对象存储 API v1参考

培训指南提供了针>对云管理的软件培训信息。

问答论坛

用户在部署或测试OpenStack的过程中，难免会遇到一些问题，诸
如如何完成特定的任务，或一些特性无法正常的运行，于是可能>
想问，其他人是否也遇到相同的问题？或者期待牛人帮助解决.那么
请访问 ask.openstack.org来提问并获得答案。当作为用户的你访问
ask.openstack.org时，请先浏览>下近期已经提问的问题，看下你遇到的问

题是否已经得到解决。如果没有，再去提交新的提问。请确保提交清晰、简洁的题目，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。

OpenStack 邮件列表

到OpenStack的邮件列表中寻求答案，将问题或出问题的场景提交到邮件列表，亦是不错的方法之一。用户可以从中学习，或者帮>助他人解决和自己遇到的类似问题。订阅或查看归档请访问http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack。也许用户对OpenStack其他项目或者开发感兴趣，OpenStack所有的邮件列表可以访问：维基百科页面，这里也有对各邮件列表详细的介绍。

OpenStack 维基百科

OpenStack 维基百科 有着大量的条目，这样带来的问题就是查找信息有点困难，或者说一些页面非常的深。幸运的是，维基百科的搜索可以解决，它支持不仅仅是标题，还支持内容检索。如果用户查找特定的信息，比如网络或者是计算服务，用户得到的结果是非常多的相关项。由于看到的是最后的页面，所以要经常查阅历史。用户可以在OpenStack>维基百科页面的右上角看到搜索框。

Launchpad的Bug区

OpenStack社区的价值之一就是通过用户创建和测试能够反馈问题。要提交bug，用户首先需要一个Launchpad的帐号，请访问 这里。登录后，用户可以查看已经提交的bug列表，也可以在Launchpad的Bug区提交自己遇到的bug。使用搜索可以查找特定的已经被报告的bug，或者已经修复的bug。当然在提交之前，请确认是否其他人已经提交过了，以免>重复。

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 镜像服务 (glance)
- Bugs: OpenStack 网络 (neutron)
- Bugs: OpenStack 对象存储 (swift)
- Bugs: 裸金属 (ironic)
- Bugs: 数据处理服务 (sahara)
- Bugs: 数据库服务 (trove)
- Bugs: 编排 (heat)
- Bugs: 计量 (ceilometer)
- Bugs: 队列服务 (marconi)
- 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

OpenStack社区的聊天室频道是托管在Freenode上，频道名为：#openstack。用户可以一直挂着，问问题，可以得到及时相应的反>馈，这对于比较着急的问题尤其重要。登录聊天室需要在本地安装客户端，或者使用网页版的 http://webchat.freenode.net/，客户端，如果用户是Windows，则安装mIRC(http://www.mirc.com/),如果是Mac OS X,则安装Colloquy (http://colloquy.info/),Linux的话就用xchat。当在聊天室中要分享代码或者是命令行输出，一个通常的办法是使用粘贴板。OpenStack也有一个，请访问http://paste.openstack.org。用户只需要复制文本，粘贴到指定的输入框，最后系统会生成>一个固定的URL，只需要将这个分享到聊天室即可。OpenStack主要的频道名是#openstack 在irc.freenode.net上。用户可以到维基百科https://wiki.openstack.org/wiki/IRC中找到所有项目的聊天室频道。

文档反馈

为文档提供反馈，请使用<openstack-docs@lists.openstack.org>加入OpenStack 文档邮件列表，或者报告 bug.

OpenStack分发包

以下是Linux发行版针对OpenStack的社区支持：

- Debian: Debian官方维基百科的OpenStack板块
- CentOS, Fedora, 以及 Red Hat Enterprise Linux: 红帽的RDO社区
- openSUSE 和 SUSE Linux Enterprise Server: openSUSE的关于OpenStack的板块
- 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
: 在父子关系中提供计算资源的逻辑分区。如果父的单元不能满足请求那么就会传递给其子单元。

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。

桌面即服务
: 能够为用户提供不论其来自何地都能登录后使用桌面环境及体验的平台。这也能够提供通用，开发甚至是 同质的测试环境。

glance
: OpenStack核心项目，提供镜像服务。

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。

镜像服务
: OpenStack核心项目之一，提供发现、注册和交付磁盘和服务器镜像的服务。项目的名称叫glance。

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。

三层网络
: 来自OSI网络架构的术语，即网络层。网络层响应报文转发，包括从一个节点到其它节点的路由。

网络
: 一个虚拟网络，能够提供两个实体之间的连接，例如，一组虚拟的端口可以共享给网络连接。在网络属于中，一个网络永远是2层网络。

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
: 一个集成项目，它能够为OpenStack提供计量和测量。Telemetry项目的代号是ceilometer。

TripleO
: OpenStack-on-OpenStack程序，为OpenStack开发程序使用的项目。

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。