

Broadview
www.broadview.com.cn

TryStack
99cloud
苏宁云商

联合力荐

· 跃上云端——OpenStack企业应用之路 ·

OpenStack

企业云平台架构与实践

张小斌 著



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

跃上云端——OpenStack

OpenStack

企业云平台架构与实践

张小斌 著



电子工业出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书结合作者亲身经历的各类 OpenStack 的咨询、规划和实施经验,以循序渐进的方式,从理论和工程角度,讲述了如何将 OpenStack (本质上只是一堆相关的进程和服务)变成企业可靠的、托管企业各类生产环境的云平台的方方面面,让 OpenStack 真正变成我们身边默默无闻但又实实在在的环境的一分子。本书分为 10 章,分别介绍了 OpenStack 与云,OpenStack 社区,OpenStack 与 AWS、VMware、虚拟化管理工具,虚拟机管理程序与典型应用,OpenStack 架构与组件,OpenStack 部分组件安装示例,系统定制技术,OpenStack 部署,第三方工具搭建 OpenStack 运行环境,九州云 Animbus 融合架构一体机解决方案等内容。

本书面向广大 OpenStack 工程人员、技术专家、IT 管理人员、云计算架构师等。对于初学者也有很强的参考意义。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有,侵权必究。

图书在版编目(CIP)数据

OpenStack 企业云平台架构与实践 / 张小斌著. —北京:电子工业出版社, 2015.1

(跃上云端: OpenStack 企业应用之路)

ISBN 978-7-121-24690-6

I. ①O… II. ①张… III. ①计算机网络—研究IV. ①TP393

中国版本图书馆 CIP 数据核字(2014)第 256610 号

策划编辑:张春雨 付 睿

责任编辑:付 睿

印 刷:北京京科印刷有限公司

装 订:三河市鹏成印业有限公司

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱 邮编:100036

开 本:787×1092 1/16 印张:25.25 字数:485 千字

版 次:2015 年 1 月第 1 版

印 次:2015 年 1 月第 1 次印刷

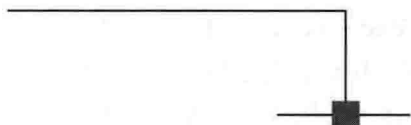
印 数:3000 册 定价:69.00 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888。

质量投诉请发邮件至 zlts@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线:(010) 88258888。

序 1



OpenStack 这两年在中国得到了众多企业和社区的关注，记得 2011 年年初，我和国内一些朋友刚刚开始在国内推广和普及 OpenStack 时很多企业都还在观望。最初我们联合英特尔、上海交通大学广州电信研究院等几家单位，在上海软件产业促进中心的支持下，邀请了该项目的发起单位 Rackspace 和 Nebula（前 NASA CTO 创建的公司），还请来了 Intel 和 Dell 的海外研发团队来国内布道，共同组织了“首届 OpenStack 上海峰会”（后来由于和基金会的 OpenStack 峰会在名称上容易混淆而改成“OpenStack 用户组大会”）。国内也只有中国电信广州研究院和上海交通大学几个为数不多的机构和部门开始研究 OpenStack。可以看出，当时已经有国内企业和机构开始在 PoC（Prove-of-Concept）或测试环境中部署 OpenStack，但生产环境大规模部署还不成熟，这对于推广 OpenStack 也是一个非常大的挑战。所以我当时就四处打探国内到底有谁在实际使用 OpenStack，也就是在这个时候我认识了新浪 SAE（Sina App Engine）团队的负责人丛磊，在他的引荐和帮助下找到了原 SAE 团队技术经理程辉，并邀请他们到社区分享经验与心得。虽然当时新浪的规模并不大，但他们确实已经在生产环境中用到了 OpenStack 中的 Nova 和 Swift 两个核心项目，并用 Swift 替换了原有的 SAE 存储，另外他们还内部尝试做了计费 Dough 和监控 Kanyun 两个开源项目（可惜后来没有了下文）。

接下来国内 OpenStack 用户群体逐渐发展了起来，先是瞬联软件、趣游、网易等分别开始尝试基于 OpenStack 开发部署自己的云平台，之后爱奇艺、用友、京东、百度、360、美团等纷纷选用 OpenStack。但总的来说初期大多数用户都将 OpenStack 作为私有云服务，并且有发展成为 OpenStack 云服务提供商的趋势。当然也有部分玩家希望先基于 OpenStack 做公有云，认为如果 OpenStack 公有云平台能支撑住成千上万的虚拟机，做私有云就没有

太大的问题。大部分私有云也并没有采用直接替换现有系统的方式，例如携程（花旗银行之前的报告也曾提到过提供 OpenStack 发行版的价值要低于交钥匙的方案，还预测得私有云用户者得天下）。

2013 年年初我和我的团队开始为部分国内企业客户提供基于 OpenStack 的私有云服务，所以接触到一些付费或有意向的用户。如互联网数据中心（Internet Data Center, IDC）客户，因为随着微软 Azure 与 AWS（Amazon Web Services）的进入，外资领先的云服务商可能对国内 IDC 企业造成毁灭性打击，IDC 正在转型的关键节点，迫使他们必须从卖资源转型到卖服务。而目前国内公有云的现状是：没有成本优势，规模无法实现赢利，不计成本很难持久，几乎没有提供相关的 API（应用程序接口），不能给用户提供按需付费的交付方式。所以说 AWS 模式很难在国内复制，特别是 AWS 采用的 DevOps 模式，要知道有 600 多人在 AWS 产品线，并且他们无须客服人员，用户自助、互助服务。在国内，如果没有客服，你都不敢想象会是什么样子。

还有一些金融客户，他们面临大数据的机遇与挑战，正在努力寻找突破。相对来说，互联网企业投入其实并不大，几个人的团队就已经足够为云平台提供运维支撑，出了问题也是可以自己扛的。金融客户重点在于关注自身的业务，而底层的基础设施服务希望能够通过第三方服务提供商来提供保障，进而可以加快项目实施进度。政府和教育行业也同样，他们面临定制化开发风险，面对升级维护等问题时也倾向于选择第三方服务提供商来实施自己的云平台项目。

而安全可靠成了企业级用户在迁移到云平台时首先关注的问题。另外，如何基于开源的 OpenStack 技术对企业应用进行优化，并为 OpenStack 核心组件设计高可靠和高可用方案，甚至要为多租户提供分布式虚拟化防火墙，基于 SDN（软件定义网络）技术为上层应用提供按需的网络资源及服务，完善监控和管理周期，实现全面自动化等；如何提高消息队列服务的高可靠性，避免因为发送消息到写入消息之间的延迟导致的信息丢失；如何实现 MySQL 的高可用集群等，都是企业客户比较关心的问题。我们在为客户实施时通过支持 OpenStack 的防火墙为租户网络隔离提供支持，通过 OpenFlow 交换机与协议对网络进行编程控制，打破了现有网络对业务封闭的问题；利用网络流量的可视化及灵活的编排，将流量导入特定的业务服务器、防火墙或者 IDS/IPS（入侵检测系统/入侵防御系统）设备，确保应用程序以及流量的正确流向及安全可靠。

这些成功部署了 OpenStack 平台的企业大致可以分为两类：一类企业属于内部有较强的技术实力，对开源软件有着强烈的好感，他们下载源码之后几乎可以通过 DIY（Do It

Yourself) 来解决自己的问题 (不过这可不像下载几部电影那么简单, 你真的要这样做吗? 想好怎么升级了吗? 谁来为你提供技术服务?); 另一类企业则选择了开源服务供应商, 例如我们团队。

不管是 OpenStack 发行版还是各种 OpenStack 解决方案都需要对 OpenStack 有一定的了解和掌握, 而目前国内这方面的资料还不是很多, 大多数散落在一些个人博客上, 本书在某种程度上填补了这个空白, 对于国内 OpenStack 爱好者来说是本详尽的参考书, 并且张小斌先生也参与了不少 OpenStack 项目的实施, 一直奋战在第一线, 积累了不少经验, 值得推广, 希望本书的出版对于国内 OpenStack 社区的发展也能起到很好的促进作用, 帮助更多用户和开发者了解 OpenStack。

杜玉杰

OpenStack 基金会独立董事

杜玉杰是 OpenStack 基金会独立董事、开源社区顾问、OpenStack 中国社区发起人、TryStack 执行董事、企业级云计算联盟 (ECA) 副秘书长, 目前主要负责 OpenStack 在中国的推广工作。先后为 IBM、Intel、Dell、HP、华为等企业提供过专业的开源咨询和培训服务, 帮助更多企业深入了解开源社区的运作, 并为客户提供基于开源的信息化顾问咨询服务。

序 2

很荣幸作为本书的第一批读者，并受张小斌先生之邀作序。小斌先生作为 IT 领域资深的技术专家，所展现出来的沉稳、深厚的技术能力，商业洞察力，面对问题的从容解决之道，均来自他本人多年的技术积累和经验提炼。本书是来自他实践的结晶和心血，会给读者带来耳目一新的感觉。

随着近年来云计算和大数据领域的持续升温，云计算、云服务得到了越来越多的人关注，云计算相关技术也已经被越来越多的企业所了解、掌握、应用和创新。按需使用、灵活计费，云计算无疑成为水/电/煤之后又一大众资源。

在传统的 IaaS (Infrastructure as a Service, 基础设施即服务)、PaaS (Platform as a Service, 平台即服务)、SaaS (Software as a Service, 软件即服务) 之外，还有 DaaS (Data as a Service, 数据即服务)、BaaS (Backend as a Service, 后端即服务)、CaaS (Communication as a Service, 通信即服务)、MaaS (Monitor as a Service, 监测即服务) 等，从而出现了 XaaS (X as a Service, Everything as a Service, 一切皆服务)。

当所有的基础设施、基本的功能服务以及面向不同领域的产品，都可以被包装成随时随地、按需使用的服务模式后，社会分工将进一步细化，服务质量将进一步提升，整体生产效率也将大幅度提高。在每个细分的领域，将不断涌现出更好的服务和产品，从而带动整个社会的进步。云服务的社会化意义将远远大于目前它所展现出来的技术层面的能力。

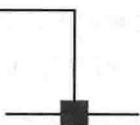
对于个人和企业来说，越早地了解 and 掌握云计算相关技术和理念，在不久的未来，就越有可能在各个领域不断创新，颠覆传统的产品模式，创造更大的价值。

小斌先生适时地推出了本书，便于大家了解和学习最新的云计算领域的动态和相关技术，无论作为技术人员还是非技术人员，真诚推荐大家选取相应部分进行阅读、交流和思考。

金龙

苏宁云开发总监

序 3



云计算技术作为目前计算机和互联网领域的研究热点，得到了国内众多研究机构、企业和高校的关注，其虚拟化、可扩展及按需服务等特性颠覆了传统的技术模式和商业模式，极大地改变了人们的生活形态。

云计算不仅为企业提供了很好的发展平台，也对高校培养高素质的复合型和创新型高等工程技术与管理人才提出了新的要求。作为高校的实验教学和管理人员，我们一直也在关注云平台相关的发展动向以及云平台在高校的应用状况，希望能设计一个针对本科生实验教学的云平台。遗憾的是，目前国内高校搭建的云平台大多数用于特定的科研项目，鲜有面向本科生教学的云平台案例可以借鉴。

我有幸与张小斌先生带领的 IBM OpenStack 设计和实施团队合作，共同探讨以本科生实验教学为目标的云平台相关实验项目，并由张小斌、郭晋兵和姚岩炜三位技术专家实施搭建了一个云计算教学和实验平台，以满足云计算相关课程的实验教学要求。从高校本科教学的角度来讲，我们希望云计算不仅仅停留在空泛的理论学习，而是通过实践操作培养学生的工程实践综合能力。针对教学的云平台和其他企业用于生产环境的云平台需求有所不同，实验教学的云平台要求根据课程能快速创建实验环境，教师能方便地对云计算实验所需的资源、环境、流程以及实验成果进行管理，使复杂实验环境的管理通过简单操作就能完成。此外，为给后续大数据应用课程相关实验的开设提供环境支持，还要求在该云平台部署 Hadoop，让学生在云计算实践环境中学习大数据处理技术。

要搭建一个满足用户需求的云平台，无论是在硬件还是软件方面，都需要足够的专业知识进行配置和优化，IBM OpenStack 设计和实施团队的专家们以严谨务实的工作态度和扎实的专业知识保障了云平台的顺利实施和使用。而如何搭建一个可用的云平台，不仅是

云计算相关工程师关心的问题，也是高等院校云计算教学的主要内容。因此，我们亟需一本介绍如何搭建云平台的具有较强实战性的书籍，帮助学生了解和掌握云计算的理论知识以及 Openstack 的整体架构，并为实验教学管理和平台维护提供实践指导。

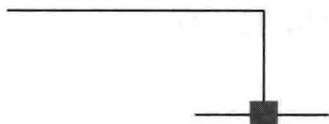
张小斌先生作为长期从事 OpenStack 研究和开发的技术专家，在云计算及云平台搭建方面有着丰富的专业知识和实践经验，在教学培训方面也具有独特的见解。由张小斌先生执笔的这本书内容涵盖了云计算的架构、模式和应用，系统地介绍了云平台的搭建以及要解决的问题，不仅可为云平台的设计实施者提供指导，也是一本适用于教学的参考用书。

OpenStack 作为一个开源项目，正处于一个不断发展完善的过程，采用 OpenStack 搭建的云平台在使用中也会出现一些意想不到的问题，如网络不稳定，大批量部署虚拟机后导致系统性能下降，用户界面操作功能有待完善等，我们相信，随着 OpenStack 的不断更新发展，这些问题也会逐步得以解决。希望张小斌先生今后能持续不断地将自己的专业知识以及实践经验以著书的形式奉献给读者，为在云平台搭建和使用中碰到难题的技术人员提供参考，也帮助更多的用户体验云平台所带来的便利服务。

金永霞

河海大学云计算与大数据老师

前言



云计算的世界正在变成现实。每天早晨起床，我们便拿起手机或者 iPad，开始浏览订阅的新闻，打开微博查看那些大牛又分享了什么，通过微信记录和分享自己的感受，然后会用一小时时间来处理各种邮件，完成一些文档。

我们的生活每天都离不开网络。互联网和我们生活的连接成为常态，无论我们在百度上的搜索、下单购物、开车远行的地图导航，还是商家给我们精准的广告投递，背后的它们都存储和运行在远处一个个数据中心的分布式服务器里，就好像所有的信息交互都来自于手里的那个设备一样。而我们所需要做的，就是提交我们的订单，然后就像支付每月的电费水费账单一样，为得到的服务买单。无论我们走到哪里，只要有网络，我们就可以访问到我们需要的服务。

在今天的公司里，需要一台服务器变得如此简单：在企业自己的私有云平台，只需要简单输入一些信息并提交，短短几分钟，我们便拥有了一台“真正”的 Linux 或 Windows 服务器，我们需要的软件已经自动安装配置好，我们只需要输入 IP 地址，便可以登录到这台计算机里。当我们使用完毕，只需要执行几个操作，这台服务器便被回收回去，不再挂在自己名下的公司资产里。我们在云平台搭建我们的开发测试环境，运行企业的 Web 服务器和数据库，就在办公环境的桌面也是云平台提供的。当我们有大量文档、图片和视频数据需要存放和管理时，只需要打开浏览器，把它们上传到云存储里去，不用再担心空间不够或者磁盘损坏怎么办。当我们有许多数据要进行分析，可以很容易地启动一些大数据的作业，无论是计算的节点，还是算法本身，都只需用指尖轻触键盘，一切均在弹指之间完成。

一切即服务，是我们今天听到最多的话语，基础设施即服务、平台即服务、软件即服务、Windows 即服务、大数据即服务、安全即服务、电源即服务。很早就进入了云计算世

界的企业，他们所解读的云计算，就是这一种技术，所有需要的资源都会作为一项服务提供给我们：无论是我们需要的主机，还是运行计算任务的大数据引擎，甚至一个复杂的开发测试环境和工作流。用软件定义的方法去重新诠释资源，让我们的业务变得如此敏捷。一切即服务为我们带来巨大的便捷，让我们专心做我们该做的事，而不像传统 IT 环境，需要花费几周甚至几个月的时间去等待漫长的采购流程、审批流程、环境搭建流程……

许多企业都开始采用云计算技术来降低成本，提高效率。

云时代的操作系统

今天我们介绍的 OpenStack 正是这样一项搭建云平台的技术。OpenStack 是一个开源的平台、开放的设计、开放的开发、开放的社区。它让你在商用硬件平台之上，搭建自己的 IaaS，无论大小。OpenStack 包括许多相关的项目，提供了各种各样的软件组件，可以非常容易地让你搭建自己的公有云或私有云平台。OpenStack 项目由全球顶尖的开发者、公司和云计算技术协作，推动着开放标准的、适合公有云与私有云的云计算平台。

OpenStack 在数据中心管理，通过软件定义的网络、软件定义的存储、软件定义的数据中心、软件定义的环境，管理各种大量的计算、网络 and 存储资源池。OpenStack 也是高度可扩展的，意味着你可以非常容易地随着时间增加新的服务器、存储和网络资源，让你的云平台长大。OpenStack 关注易用性、大规模可扩展，其插件式的设计可以非常容易地集成各个服务器、网络 and 存储厂商的产品和技术，而无任何厂商锁定。

许多企业都采用 OpenStack 来搭建自己的云平台。根据 2013 年 OpenStack 香港大会前做的用户调研，共有全球 539 家公司提交了回应，目前已知的 OpenStack 部署分布在 56 个国家的 200 多个城市中，分布在互联网、学术教育、电信、制片与多媒体、政府国防、制造业、零售、医疗、金融、消费品市场等行业。

在部署 OpenStack 的企业中，有 5 000 人以上的大企业占到了 29% 的比例，而 100 人以下的小微型企业则占到了 42% 的比例。

按照部署类型划分，60% 的 OpenStack 部署都是作为企业内部私有云，17% 是托管的私有云，公共云部署占 15%，混合云部署占 6%，剩下的 2% 是用于维护 OpenStack 社区自身运作的。

那么，从用户的角度，云平台能带来什么样的价值呢？让我们看看用户为什么选择 OpenStack 解决方案。下面是按照关注程度排序，用户为什么选择 OpenStack 的一些原因：

- 节省成本；
- 提高运维效率；
- 开放的平台；
- 无厂商锁定；
- 底层可以有比较多技术选择的自由；
- 能够帮助企业更好地创新。

IDG Connect 代表红帽公司进行了一项网上调研，来了解用户对私有云，特别是 OpenStack 的看法。共有 200 位来自美国公司（员工人数在 1 000 人以上的公司）的 IT 决策者积极参与了这项调研，其中有 35% 的受访者来自员工人数超过 5 000 人的大型企业。访者主要来自：

- 制造业；
- 金融服务；
- 医疗保健/医药；
- 零售在内的垂直行业；
- 公共部门。

调查结果显示，私有云的发展面临着很多方面的挑战，包括：资源管理（占到 21%）、简化 IT 管理（占到 18%）、应用软件管理（占到 18%）、应用软件迁移（占到 18%）。随着企业用户可以越来越重视地解决这些问题，企业用户正在或有计划地向 OpenStack 私有云迁移。有 60% 的受访者表示，他们正处在配置 OpenStack 的初期阶段，有的尚未完成，或者还在实施过程当中。有 84% 的受访者表示 OpenStack 是他们未来云计划的一部分。

参与调研的受访者认为 OpenStack 成为私有云替代品所具备的独特优势包括：

- 管理的可见性（占 73%）；
- 配置速度（占 72%）；
- 平台的灵活性（占 69%）；
- 更好的灵活性（占 69%）；
- 竞争优势（占 67%）。

“跃上云端——OpenStack 企业应用之路”丛书

OpenStack 在工程实践中，对许多初学的用户来说，面临着许多困难，首先其涉及的内

容非常庞大而复杂；其次，虽然网上有许多资料，但都是针对具体某个技术和专题而展开的，缺乏系统、全局的视野来看待 OpenStack 的范围、难点和挑战，也缺乏具体实践指导，如何将 OpenStack 变成一个真实运行、稳定而性能良好的云平台，不管是几个节点的实验环境，还是几十甚至上百个节点的研发测试生产环境。本书很好地总结和回答了这些问题。另外，许多人往往关注 OpenStack 的具体某项技术，但往往忽略了，一个可以运行的 OpenStack 云平台，往往涉及大量系统方面的知识和经验，如物理网络规划、部署规划、具体参数和配置对性能的影响，大规模环境规划中面临的网络地址管理，网络分离设计，存储方案的选择，一步步的实施计划等。这些内容在网上都是很难找到的。

本书面向广大的 OpenStack 初学者、系统开发人员、云平台实施人员，还面向系统架构师和 IT 管理人员，不但涵盖 OpenStack 学习和使用中遇到知识的方方面面，还包括许多规划方面的第一手实践，对各个知识系统的相关性也进行了很多阐述。通过本书，希望能让广大的读者能融会贯通，对面临的广度和复杂度有一个清晰的视野，又有具体的工程实践方面的第一手资料可供参考。

“跃上云端——OpenStack 企业应用之路”系列丛书，由一些长期从事 OpenStack 研究、设计、开发、部署、实施的一线技术专家和工程人员共同编写和校阅。其中 IBM OpenStack Solution 架构师张小斌对丛书进行策划和总体内容的规划，并主要编写丛书中的第一本图书，其他人员还包括 IBM OpenStack 和 X86 实施团队技术专家郭晋兵，IBM OpenStack 网络存储专家姚岩炜，北大软件学院天才的研究生蓝启嵩、郝树伟，北京邮电大学林潇俊等。本丛书得到了弯曲评论首席陈怀临先生，OpenStack 社区独立董事、TryStack 社区主席杜玉杰先生，99 云总裁张淳先生的技术指导与大力支持，在此一并表示感谢。

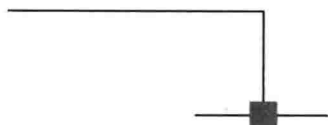
参与本书编写和审阅的其他朋友还包括夏庆梅、张小立、杨永胜、张娟、杨飞、高凯、苏沛沛、王婷、张小轶、刘业成等。

“跃上云端——OpenStack 企业应用之路”丛书系列包括：

- 《OpenStack 企业云平台架构与实践》
- 《OpenStack 企业应用托管与演进》

由于云平台 and OpenStack 所涉及的知识和技术非常庞大而复杂，本人水平有限，错误在所难免，如有疑问请与作者本人直接联系和讨论（作者邮箱：xiaobinz@gmail.com），非常感谢！

目 录



第 1 章	OpenStack 与云	1
1.1	云计算	2
1.2	云服务模式	3
1.2.1	IaaS: 基础设施即服务	3
1.2.2	PaaS: 平台即服务	4
1.2.3	SaaS: 软件即服务	4
1.3	云应用形式	4
1.3.1	私有云	4
1.3.2	云存储	5
1.3.3	云游戏	5
1.3.4	云物联	5
1.3.5	云安全	5
1.3.6	公有云服务	6
1.3.7	混合云	6
1.3.8	云计算的安全风险	6
1.4	云管理与虚拟化管理	7
1.5	私有云与公有云	8
1.6	传统应用与云感知应用	9
1.7	什么是 OpenStack	11
1.8	开源云平台比较	13

1.9 术语	15
第 2 章 OpenStack 社区	17
2.1 OPENSTACK 基金会	21
2.2 白金会员	22
2.3 黄金会员	22
2.4 OPENSTACK 设计原则	23
2.5 开源而开放的原则	23
2.6 OpenStack 版本管理	23
2.7 OpenStack 用户	24
2.8 OpenStack 的误区	27
2.9 部署 OpenStack 的技术需求	30
第 3 章 OpenStack 与 AWS、VMware、虚拟化管理工具	31
3.1 OpenStack 与 AWS 的比较	32
3.2 OpenStack 与 VMware 对比	39
3.2.1 VMware vMotion 与 OpenStack 动态迁移、块迁移	44
3.2.2 VMware DRS、DPM 与 OpenStack 调度器	45
3.2.3 VMware 与 OpenStack 的高可用	45
3.2.4 VMware 与 OpenStack 的容错 (Fault Tolerance)	46
3.2.5 总结	46
3.3 虚拟化与虚拟化管理工具	47
3.3.1 服务器虚拟化	47
3.3.2 网络虚拟化	48
3.3.3 存储虚拟化	48
3.3.4 虚拟化工具 VirtualBox	48
3.3.5 虚拟化工具 Virt-Manager	53
第 4 章 虚拟机管理程序与典型应用	56
4.1 开放虚拟化技术 KVM	57
4.1.1 libvirt 介绍	58
4.1.2 域配置文件	59

4.1.3 使用 Libvirt 创建和管理 KVM 虚拟机	60
4.2 Linux 容器	62
4.2.1 LXC	65
4.2.2 Docker	71
4.3 裸机	77
4.4 LXC/Docker 与 KVM/Xen 的选择	78
4.5 OpenStack 与 Linux	79
4.6 OpenStack 与 KVM	79
4.7 OpenStack 与 VDI	80
4.7.1 基于 OpenStack 的 VDI 典型架构	80
4.7.2 Spice 协议	81
4.7.3 开发桌面虚拟化应用的功能需求	84
4.8 OPENSTACK 与 HADOOP	85
4.8.1 云平台/虚拟化对大数据计算的益处	86
4.8.2 OpenStack 对 Savanna 的支持	86
4.8.3 Savanna 的使用简介	87
第 5 章 OpenStack 架构与组件	92
5.1 OpenStack 项目与组件	93
5.2 IaaS 模型与 OpenStack 组件对应关系	95
5.2.1 OpenStack 功能待提高的方面	100
5.2.2 节点与网络类型	100
5.3 消息总线 and 数据库	104
5.4 多租户	105
5.5 Keystone	107
5.6 Glance	112
5.7 Nova	119
5.7.1 nova-api	119
5.7.2 nova-scheduler	120
5.7.3 nova-schedule 过滤器	122
5.7.4 nova-volume	128
5.7.5 nova-compute	128

5.7.6 nova-network	132
5.7.7 nova-conductor	135
5.7.8 服务横向扩展	136
5.8 存储	138
5.8.1 对象存储	139
5.8.2 块存储	139
5.8.3 Cinder	144
5.8.4 卷 (Volume) 操作	145
5.9 Neutron	148
5.9.1 nova-network 的局限性	148
5.9.2 Neutron 功能特点	149
5.9.3 ML2	149
5.9.4 Open vSwitch 虚拟网络	151
5.10 Ceilometer	154
5.10.1 计算 (Nova)	155
5.10.2 网络 (Neutron)	156
5.10.3 镜像 (Glance)	157
5.10.4 块存储 (Cinder)	158
5.10.5 对象存储 (Swift)	158
5.10.6 编排 (Heat)	158
5.10.7 能源 (Kwapi)	159
5.10.8 网络 (SDN 控制器)	159
5.10.9 计量数据收集的结构、交互图	161
5.11 Heat	161
第 6 章 OpenStack 部分组件安装示例	163
6.1 安装拓扑	164
6.2 服务器远程安装配置	165
6.3 软件包与仓库	167
6.3.1 软件包	167
6.3.2 软件仓库	169
6.3.3 依赖关系	169

6.3.4	软件包名称	170
6.3.5	只下载软件包的方法	171
6.3.6	RPM 常用命令	172
6.4	ISO	173
6.5	安装 OpenStack 组件——Keystone、Glance 和 Quantum	175
6.5.1	控制节点	175
6.5.2	计算节点	187
6.5.3	设置 iptables 规则	187
6.6	大规模安装技术与工具	187
6.6.1	Chef	187
6.6.2	Puppet	190
6.6.3	Chef 与 Puppet 的比较	191
6.6.4	IBM xCAT	192
第 7 章	系统定制技术	193
7.1	系统环境的定制	194
7.1.1	KVM 的检查与安装	194
7.1.2	网络时间协议 (NTP) 服务的设置	196
7.1.3	SSH 无密码登录	197
7.1.4	自动运行定制化程序	199
7.1.5	简单备份	200
7.2	网络	201
7.2.1	ifconfig 命令使用及结果分析	201
7.2.2	静态 IP 地址的配置	202
7.2.3	网卡绑定	204
7.2.4	网桥模式的配置	207
7.2.5	Access、Hybrid 和 Trunk 三种模式	209
7.3	安装与打包技术	211
7.3.1	制作 RPM	211
7.3.2	Kickstart 快速安装	218
7.3.3	编辑可引导的 ISO	230
7.3.4	制作一个定制化可引导的 ISO	231

第 8 章 OpenStack 部署	240
8.1 来自实际客户的困惑	241
8.2 企业云环境规划	245
8.2.1 理解企业业务需求和预期	245
8.2.2 云平台规划	247
8.3 区域和可用区	252
8.4 典型部署拓扑	255
8.4.1 基于传统网络的基本部署架构	255
8.4.2 基于 OpenStack Neutron 的部署架构	257
8.4.3 基于 Ceph 统一存储的部署架构	257
8.4.4 中型企业私有云部署架构	258
8.4.5 中型企业差异化资源池、多种存储池的部署	260
8.4.6 融合传统硬件的部署方案	260
8.5 云平台硬件选择	261
8.5.1 试验环境推荐配置	261
8.5.2 标准部署推荐配置	262
8.6 控制节点的设计	263
8.6.1 硬件抉择	263
8.6.2 服务分离式部署	264
8.6.3 数据库	264
8.6.4 消息中间件	264
8.6.5 认证与授权	265
8.6.6 网络	265
8.6.7 计算节点的设计	265
8.6.8 存储的选择	270
8.6.9 网络的选择	276
8.6.10 计算硬件需求	279
8.6.11 软件部分检查清单	284
8.6.12 与企业现有系统集成	285
8.6.13 云环境扩展需求	286
8.7 生产环境 问题和对策	288

8.7.1	计算资源隔离和流量控制	288
8.7.2	调度策略	288
8.7.3	负载均衡	288
8.7.4	OpenStack 的实施	289
8.8	云平台监控	291
8.8.1	性能监控	291
8.8.2	服务与资源监控	293
8.8.3	消息中间件监控	295
8.8.4	日志分析	298
第 9 章	第三方工具搭建 OpenStack 运行环境	300
9.1	DevStack	302
9.1.1	环境准备	302
9.1.2	安装	303
9.2	IBM OpenStack Solution for System X	309
9.2.1	OpenStack Solution 客户服务生命周期	310
9.2.2	OpenStack Solution 的功能特点与优势	311
9.2.3	云平台内部的修补、优化与定制	313
9.2.4	模块化和自动化的云平台搭建技术	315
9.2.5	控制节点的设计	316
9.2.6	控制服务部署场景	316
9.2.7	安装	317
9.3	Red Hat RDO	322
9.3.1	环境准备	322
9.3.2	设置源	324
9.3.3	安装 PackStack	326
9.3.4	定制 PackStack 的 answer 文件	326
9.3.5	安装	338
9.3.6	增加计算节点	342
9.4	Mirantis Fuel	343
9.4.1	Fuel 简介	343
9.4.2	Mirantis 支持的架构	344

9.4.3	Fuel 安装	349
9.5	Dell Crowbar	364
9.5.1	Crowbar 安装 OpenStack	365
9.5.2	界面	366
第 10 章	九州云 Animbus 融合架构一体机解决方案	371
10.1	产品背景	372
10.2	九州云计算存储云一体机	375

第 1 章

OpenStack 与云

1.1 云计算

云计算（Cloud Computing）包括狭义的云计算和广义的云计算，狭义的云计算指 IT 基础设施的交付和使用模式，通过网络以按需、易扩展的方式获得所需资源；广义的云计算指服务的交付和使用模式，通过网络以按需、易扩展的方式获得所需服务，通常通过互联网来提供动态易扩展且经常是虚拟化的资源。这种服务可以是提供 IT 基础设施、软件或互联网相关的各种服务，也可以是其他类型的服务。云是网络、互联网的一种比喻说法，也用来表示互联网和底层基础设施的抽象。它意味着计算能力也可以作为一种商品通过互联网进行流通。

通过这种资源和服务的交付方式，共享的软硬件资源和信息可以按需提供给计算机和其他设备。典型的云计算提供商往往提供通用的网络业务应用，可以通过浏览器等软件或者其他 Web 服务来访问，而软件和数据都存储在服务器上。

通过使计算分布在大量的分布式计算机上，而非本地计算机或远程服务器中，企业数据中心的运行将与互联网更相似。这使得企业能够将资源切换到需要的应用上，根据需求访问计算机和存储系统，类似于从古老的单台发电机模式转向电厂集中供电的模式。它意味着计算能力也可以作为一种商品进行流通，就像煤气、水电一样，取用方便，费用低廉。最大的不同在于，它是通过互联网进行传输的。

云计算可以满足大规模计算需求，也可以通过虚拟化服务器，以更多地利用现有的硬件和从潜在的服务中释放旧的硬件来整合工作。人们还可以通过搭建高可用云计算平台，提供软件套件，用于文字处理、数学运算和电子邮件通信。云计算还可以把额外的存储资源通过云平台共享出来，将用户数据保存在云端，避免了每个用户的桌面上需要额外的硬盘驱动器，并能在云中提供巨大的在线数据存储容量。

互联网上的云计算服务特征和自然界的云、水循环具有一定的相似性，通常云计算服务应该具备以下几条特征。

- 基于虚拟化技术快速部署资源或获取服务；
- 可以动态地、可伸缩地扩展；
- 按需求提供资源，按使用量付费；
- 通过互联网提供面向海量信息的处理；

- 用户可以方便地参与使用；
- 减少用户终端的处理负担；
- 降低用户对于 IT 专业知识的依赖；
- 虚拟资源池为用户提供弹性服务。

云计算的含义包含以下两个方面。

(1) IT 资源的云化，或者说 IT 资源的一种组织形态，称为 IT 资源池。这个池也是一种 IT 系统，但这个池中的 IT 资源不是孤立的，而是构成一个有机体，可以动态配置、灵活扩展、自动化管理。这个池用“云”这个概念来表示。

(2) IT 资源的使用模式，即服务化。过去 IT 资源是在用户端本地部署和使用的，现在是部署在云端，并且以服务的方式对用户提供了 IT 资源。用户通过网络访问这些服务。这种使用模式的好处是服务可以随时、随地、按需地获得，根据资源使用情况付费。这种使用模式用“云服务”这个概念来表示。

人们也可以从如下方面理解云计算。

- **按需自服务**：用户可以自己申请并获得计算、存储和网络资源，几乎不需要人工干预。
- **网络访问**：任何计算功能都可以通过网络实现，有许多不同的设备可以访问。
- **资源池化**：多个用户可以同时访问云，根据需求分配计算资源，来服务于消费者。
- **快速弹性可扩展**：根据需要，云平台可以随时在线扩容，而用户是感知不到的。
- **计量或测量服务**：就像按小时支付的公用事业，云应该优化资源利用和控制它的服务或服务器，如存储或处理能力，并按计算类型的级别提供不同的服务模式，满足不同消费者消费能力的需要。
- **多租户架构**：云平台资源是共享的，而且使用是相互隔离的。

1.2 云服务模式

1.2.1 IaaS：基础设施即服务

IaaS (Infrastructure as a Service)：基础设施即服务。消费者通过 Internet 可以从完善的计算机基础设施获得服务。该层为用户提供虚拟机，包括网络连接和存储，人们能够运行

任何软件或操作系统。

在这种服务模型中，服务商以服务的形式提供虚拟硬件资源，如虚拟主机、存储、网络等资源。用户无须购买物理服务器、网络设备、存储设备，只需要通过互联网租赁就可以搭建自己的应用系统。典型应用有亚马逊的 AWS（Amazon Web Services）等。

1.2.2 PaaS：平台即服务

PaaS（Platform as a Service）：平台即服务。PaaS 实际上是指将软件研发的平台作为一种服务，以 SaaS 的模式提交给用户。消费者通过一门编程语言，或通过云平台提供商支持的工具体来部署应用程序，如 Eclipse/Java 编程平台。因此，PaaS 也是 SaaS 模式的一种应用。但是，PaaS 的出现可以加快 SaaS 的发展，尤其是加快 SaaS 应用的开发速度。

在这种服务模型中，服务商提供应用服务引擎，如互联网应用编程接口/运行平台等。用户基于这些应用服务引擎，就可以搭建自己的应用。典型应用有 Google AppEngine、Force.com、微软 Azure 等服务平台。

1.2.3 SaaS：软件即服务

SaaS（Software as a Service）：软件即服务。它是一种通过 Internet 提供软件的模式，消费者使用部署于云平台中的软件，如基于 Web 的电子邮件、客户关系系统、订单管理系统、HR 系统等。用户无须购买软件，而是向提供商租用基于 Web 的软件，来管理企业经营活动。

在这种服务模型中，用户通过 Internet 使用软件，而不必购买，只需要按需租赁软件即可。典型应用有 Google Doc、Salesforce.com、Oracle CRM On Demand、Office Live Workspace 等。

1.3 云应用形式

1.3.1 私有云

私有云（Private Cloud）是将云基础设施与软硬件资源创建在防火墙内，以供机构或企

业内各部门共享数据中心内的资源。服务形式通常为大型企业在企业内部数据中心按照云计算的架构搭建平台，面向企业内部用户、面向内部需求来提供云计算服务等。

创建私有云，除了硬件资源外，一般还需要云平台（IaaS）软件；开放源代码的云平台软件主要有 OpenStack、CloudStack、Eucalyptus 等。

1.3.2 云存储

云存储是在云计算概念上延伸和发展出来的一个新的概念，是指通过集群应用、分布式文件系统等功能，将网络中大量各种不同类型的存储设备通过应用软件集合起来协同工作，共同对外提供数据存储和业务访问功能的一个系统。当云计算系统运算和处理的核心是大量数据的存储和管理时，云计算系统中就需要配置大量的存储设备，那么云计算系统就转变成为一个云存储系统，所以云存储是一个以数据存储和管理为核心的云计算系统。

当前云存储的应用主要在网盘领域，而网盘又分为企业网盘和个人网盘。

企业也搭建云存储平台，用以存放海量的文档、电子邮件、视频、图片、数据备份等。

1.3.3 云游戏

云游戏是以云计算为基础设施的游戏平台，在云游戏的运行模式下，所有游戏都在服务器端运行，并将渲染完毕后的游戏画面压缩后通过网络传送给用户。在客户端，用户的游戏设备不需要任何高端处理器和显卡，只需要基本的视频解压能力即可。

1.3.4 云物联

“物联网就是物物相连的互联网”。这有两层含义：第一，物联网的核心和基础仍然是互联网，是在互联网基础上延伸和扩展的网络；第二，其用户端延伸和扩展到了在任何物品与物品之间进行信息交换和通信。而互联网往往就是云计算平台。

1.3.5 云安全

云安全（Cloud Security）是一个从“云计算”演变而来的新名词。云安全的策略构想是：使用者越多，每个使用者就越安全，因为如此庞大的用户群，足以覆盖互联网的每个

角落，只要某个网站被挂马或某个新木马病毒出现，就会立刻被截获。

云安全通过网状的大量客户端监测网络中软件行为的异常，获取互联网中木马、恶意程序的最新信息，推送到服务器端进行自动分析和处理，再把病毒和木马的解决方案分发到每一个客户端。

1.3.6 公有云服务

公有云服务的形式为：管理员面向外部用户需求，通过开放网络提供云计算服务。大型互联网企业是目前国内主要的云计算服务提供商，业务形式以 IaaS+PaaS 形式的开放平台服务为主，其中 IaaS 服务相对较为成熟，PaaS 服务初具雏形。国内大型互联网企业开发了云主机、云存储、开放数据库等基础 IT 资源服务，以及网站云、游戏云等一站式托管服务。一些互联网公司自主推出了 PaaS 云平台，并向企业和开发者开放。一些例子包括 IDC、GoogleApp、Salesforce 在线 CRM。

1.3.7 混合云

云平台提供者兼顾以上公有云与私有云两种情况的云计算服务。典型的例子包括 AWS 等既为企业内部又为外部用户提供云计算服务的云平台。混合云也是企业内部私有云与外部公有云平台的连接，可以在公有云与私有云之间调度资源的使用形式，另外也有人将跨虚拟化管理程序的云平台称为混合云。

1.3.8 云计算的安全风险

云计算意味着数据被转移到用户掌控范围外的机器上，也就是云计算服务提供商的手中，潜在地会存在如下安全问题。

- 云平台管理特权用户访问；
- 是否遵从数据安全用户隐私的法律法规；
- 数据位置变得模糊和不可追踪；
- 对不同安全级别的数据隔离；
- 服务和数据的可用性；
- 灾难恢复需要验证；

- 审计比较困难；
- 云平台提供商的存活能力；
- 降低风险方面的支持；
- 服务中断事故将使消费者意识到服务质量的差异。

安全总是相对的。最重要的是，企业和组织要有实际的安全保障措施，以及审计和确保这些措施正常运行的程序。不管是自己建立安全系统，还是向第三方供应商租赁安全服务，都是如此。建立或租赁安全系统也并不会获得完全意义的安全保护。

云计算的安全风险对私有云并不严重，许多组织和个人仍认为私有云很安全。然而，公有云则潜在具有相当的风险，当迁移到公有云平台时，一定要对各种安全风险做谨慎的评估。

1.4 云管理与虚拟化管理

在 IaaS 这个层面，云管理和虚拟化管理的概念非常接近，但是有一些细微的差别。

虚拟化是指在同一台物理机器上提供多台虚拟机器（包括 CPU、内存、存储、网络等计算资源）的能力。每一台虚拟机器都能够像普通的物理机器一样运行完整的操作系统以及执行正常的应用程序。当需要管理的物理机器数量较小时，虚拟机生命周期管理（资源配置、启动、关闭等）可以通过手工去操作。当需要管理的物理机器数量较大时，就需要写一些脚本和程序来提高虚拟机生命周期管理的自动化程度。以管理和调度大量物理和虚拟计算资源为目的的系统，属于虚拟化管理系统。这样一个系统，通常用于管理企业内部的计算资源。

云计算是指通过网络访问物理和虚拟计算机并利用其计算资源的活动。通常来讲，云计算提供商以虚拟机的方式向用户提供计算资源。用户无须了解虚拟机背后实际的物理资源状况，只需了解自己所能够使用的计算资源配额。因此，虚拟化技术是云计算的基础。绝大多数云计算管理平台都是构建在虚拟化管理平台的基础之上的。

1.5 私有云与公有云

公有云和私有云之间的界限，就像“内部/外部”和“部门/合作伙伴”的概念一样，并不十分明显。根据项目需求的不同，可能会有不同的解释。表 1-1 对它们进行了简单的对比。

表 1-1 公有云与私有云的对比

	服 务 对 象	特 点	理 想 用 户
私有云	仅对某个集团或企业内部提供服务	<ul style="list-style-type: none">• 强调虚拟资源调度的灵活性。• 系统管理员需要为不同的部门（或者应用）定制不同的虚拟机，根据部门（或者应用）对计算资源的需求对分配给某些虚拟机的计算资源进行调整。• OpenQRM、XenServer、Oracle VM、CloudStack 和 ConVirt 比较适合提供私有云服务	各种小微组织、社团、创业企业、中小服务企业
公有云	对公众提供服务	<ul style="list-style-type: none">• 强调虚拟资源的标准性。• 通过将计算资源切割成标准化的虚拟机配置（多个系列的产品，每个产品配置相同数量的 CPU、内存、磁盘空间、网络流量配额），公有云提供商可以通过标准的服务合同（Service Level Agreement, SLA）以标准的价格出售计算资源• 由于 Amazon EC2 是目前比较成功的公有云提供商，大部分云管理平台都在某种程度上模仿 Amazon EC2 的构架。从这个意义上讲，Eucalyptus、OpenNebula 和 OpenStack 提供了与 Amazon EC2 兼容或者类似的接口，比较适合提供公有云服务	中小规模及以上，有自己数据中心或机房，有一定技术人员或储备的企业

业界对私有云的前景有许多独到的见解，但争论还在继续。下面是一些比较中肯的看法，然而各个企业和组织还是要根据自己的实际情况进行评估。这里没有标准答案。

（1）私有云不会消失，而会获得大发展。

许多人认为，私有云并不是“真正的云”，而公共云才是改善 IT 部门服务的真正方法。

公共云服务提供商甚至认为，私有云并不该存在，因为它缺乏灵活性，而且价格昂贵。但实际上，IT 部门使用私有云就是为了给它们的组织提供灵活的、可随时调用的计算环境。而且，组织在现有的数据中心部署私有云服务实际上更为便捷，也更为便宜。在将来，私有云不仅不会消失，而且会获得更大的发展。

事实上，许多原来搭建在亚马逊云平台之上的企业，在发展到一定程度，或企业的更多需求无法满足时，都转而搭建自己的云平台。

(2) 混合云仍将是大多数组织的务实选择。

公共云的使用在计算领域掀起了一场革命，对于难以预测的面向消费者的应用程序来说尤其如此。对于需要处理各种规章制度、标准和其他非技术问题的组织来说，私有云不可或缺。这取决于企业的实际需要。但是，混合云服务，即综合有公共云和私有云的服务，仍将是大多数组织的务实选择。

在 2011 年，亚马逊广为人知的服务中断事故给很多人敲响了警钟，因为很多人曾经错误地认为云服务能够保证始终正常运行。对于云计算来说，细节问题非常重要，你不知道的东西可能会损害你的云应用程序。云服务的某些特性，例如高可用性，可通过投资基础设施、人力资源和 workflows 来实现。这也是区分云服务提供商的关键所在。在高可用性方面投资较少或几乎没有投资的云服务，表面上看起来可能很便宜，但是你必须自己投资时间和费用从零开始设计、编写和操作自己的可用性系统。

对于公有云来说，有三大门槛：资金门槛、运营门槛，以及技术门槛。在三个门槛中，技术和运营更为关键，因为技术和运营需要长期的积累，而且是大型云平台实战的经验积累。选择公有云平台时，也需要对云平台所有者进行相应的调查和评估。

1.6 传统应用与云感知应用

什么样的应用可以跑在云平台之上呢？或者云平台是否支持企业现有的应用呢？

每个人都有许多和应用程序打交道的经历。当云兴起并逐渐成熟，当考虑是否将企业生产或关键应用迁移到云端时，一个现实的问题是：原来那些依赖于基础架构来保证的高可用、容错、灾难恢复等机制，在云端还能保证吗？或如何保证呢？

如图 1-1 所示，有一个形象的比喻是：“你是在养宠物，还是在养牛？”在传统的服务

模型下，你的机器以及运行的应用就像是“宠物”，你需要精心照顾它们。今天许多传统应用依然是这种模型。在云感知服务模型中，虚拟机就像“牛（牲口）”，每个虚拟机都是一样的，当资源不够时，只需要增加新的虚拟机；当它们出现故障时，只需要重启或创建新的虚拟机即可。今天互联网企业里基本都是这样的应用模型。



图 1-1 传统应用与云感知应用的对比

云感知应用能够自己解决高可靠性、容错和灾难恢复的场景；而传统应用需要基础架构来提供对高可靠性和灾难恢复等的支持，图 1-2 从架构层面说明了这种区别。

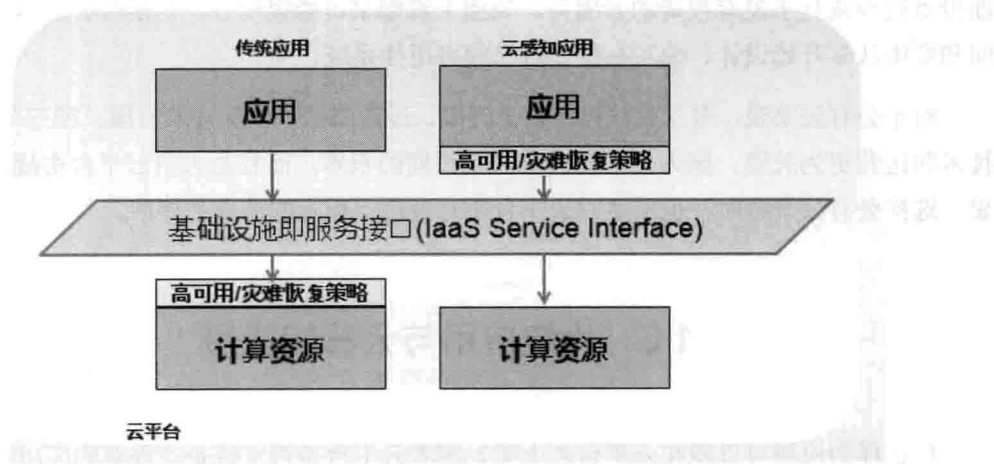


图 1-2 传统应用与云感知应用的架构区别

云感知应用的一些共同特征如下。

- 分布式的；
- 无状态的；

- 应用设计中已包含故障转移;
- 应用设计中已包含可扩展性。

而传统应用的一些共同特征表现为:

- 客户-服务器架构;
- 典型的是贯穿的逻辑和代码, 很难水平扩展;
- 故障转移依赖于基础设施;
- 可扩展性也依赖于基础设施。

传统应用需要云平台提供诸如容错、虚拟机层的高可靠性、自动的病毒检测等, 而云感知应用则不需要, 当前虚拟机失效时, 只需要重新创建一个新的虚拟机代替即可。

传统的应用通常需要更强的硬件来扩展(垂直扩展), 而基于云的应用则通过更多的硬件来扩展(水平扩展)。扩展时, 不是通过切换到更强、更大的服务器, 而是通过购买更多硬件, 简单安装同样配置的服务程序。对于相同的服务, 通过负载均衡来调度这些服务“集群”。

1.7 什么是 OpenStack

OpenStack 最早由美国国家航空航天局(NASA)研发的 Nova 和 Rackspace 研发的 Swift 组成; 后来以 Apache 许可证授权, 是一个旨在为公共及私有云的建设, 面向云平台管理的自由软件和开放源代码项目。这个项目由几个主要的组件协作起来完成一些具体的工作。

现在除了美国航空航天局(NASA), Rackspace 在内有大量公司加入了 OpenStack 基金会, 成为它的成员。表 1-2 列出了已经加入 OpenStack 基金会, 并从事 OpenStack 功能设计、开发, 或提供方案咨询、设计与实施的著名的 IT 巨头们。

表 1-2 OpenStack 的 IT 巨头们

前三服务器厂商	IBM、HP、Dell
前三 Linux 发行厂商	Red Hat、Canonical、SUSE
前三网络设备厂商	Cisco、Juniper、Alcatel-Lucent
前三存储厂商	IBM、EMC、NetApp

在今天的数据中心中，许多计算机在计算能力和网络带宽方面都遭遇到同样的使用率偏低的问题。例如，项目可能需要大量的计算能力来完成计算，但在完成计算后就不再需要那么大量的计算能力。OpenStack 正是用来更有效率地管理各类计算、存储、网络资源的一个开源项目和许多软件组件的集合。

OpenStack 项目的首要任务是简化云的部署过程并带来良好的可扩展性，帮助服务商和企业内部实现类似于 Amazon EC2 和 S3 的云基础架构服务 (IaaS)。OpenStack 包含 8 个核心项目，还有许多孵化项目。其中，核心项目如下。

- Nova: 计算。
- Keystone: 认证。
- Glance: 镜像。
- Cinder: 块存储。
- Neutron: 网络。
- Swift: 对象存储。
- Heat: 编排。
- Ceilometer: 计量。

每个项目都包括许多不同组件，可以部署在商用硬件之上。其部署非常灵活，可以支持各种部署拓扑，绝大部分服务都可以水平扩展，以提高性能、可靠性和服务能力。

作为构建云平台的一项重要技术，OpenStack 在这些年里发展迅猛。然而搭建一个基于 OpenStack 的云平台却不是那么容易：在软件方面，OpenStack 云平台包含大量软件组件（大多数是开源的），需要足够的专业知识去对软件进行选择、配置以及调优；在硬件方面，配置一个合理的硬件平台也需要掌握足够的软件和测试等专业知识。

尽管几年前 OpenStack 在国内外有许多部署环境，然而早期的版本功能非常有限，而且代码中存在许多问题。OpenStack 社区在三年来修补了许多代码中的问题，增加了许多功能，也吸收了许多从部署和生产环境得到的反馈和经验教训。同时还有一些新的项目正在孵化并走向成熟，例如，Savana，在 OpenStack 平台之上运行大数据作业；Murano，在 OpenStack 平台之上运行 Windows Server 2008 或 Windows Server 2012 操作系统。类似的项目还有很多，许多新的项目正在以孵化的形式演进，等到一定阶段，就会进入核心项目之列，供用户使用。

现在 OpenStack 无论从功能还是代码质量上都足以走进企业，为企业搭建各种云平台环境，从几台服务器规模的环境，到成百上千台服务器的数据中心。

1.8 开源云平台比较

目前有四种主流的开源云平台软件，表 1-3 对他们进行了简单的对比，OpenStack 是其中的佼佼者。

表 1-3 开源云平台的对比

	授权协议	许可证管理	商业模式	项目历史与运营团队
OpenStack	Apache 2.0 授权协议	不需要许可证	免费使用	是服务器托管公司 Rackspace 与 NASA 共同发起的开放源代码项目。目前已经获得国际上绝大多数主流 IT 巨头公司的支持
OpenNebula	Apache 2.0 授权协议	不需要许可证	<ul style="list-style-type: none"> • 社区版免费使用 • 企业版将社区版重新打包，提供补丁等程序的访问权限，使得用户能够更容易地安装、配置和管理，以订阅的模式提供服务 	是 2005 年启动的研究性项目，2008 年年初发布第一个开放源代码版本，2010 年年初大力推进开源社区的建设
Eucalyptus	<ul style="list-style-type: none"> • 社区版采用 GPLv3 授权协议 • 企业版使用自定义的商业授权协议 	<ul style="list-style-type: none"> • 社区版不需要安装许可证 • 企业版需要在云控制器 (CLC) 节点上安装许可证 	<ul style="list-style-type: none"> • 社区版免费使用 • 企业版按处理器核心总数收费，用户购买的许可证针对特定版本永久有效 	最初是 UCSB 的 HPC(高性能计算)研究项目，2009 年年初成立公司来支持该项目的商业化运营
CloudStack	<ul style="list-style-type: none"> • 社区版采用 GPLv3 授权协议 • 企业版使用自定义的商业授权协议 	<ul style="list-style-type: none"> • 社区版不需要安装许可证 • 企业版需要在管理服务器上安装许可证 	<ul style="list-style-type: none"> • 社区版免费使用。 • 企业版提供增强功能和技术支持，收费模式不详 	源于 2008 年成立的 VMOps 公司，2010 年 5 月启用 cloud.com 域名，后来被 citrix 收购并被 citrix 开源，并交给 Apache 软件基金会管理

下列数据摘自蒋清野先生的云平台开源社区活跃度的统计和分析报告。蒋清野先生通过收集和分析 OpenStack、OpenNebula、Eucalyptus 和 CloudStack 项目的论坛和邮件列表的各种原始数据，对这些社区的活跃度进行了分析和比较。

如图 1-3 所示为这四种开源云平台每个月新增加的社区人口数量。从 2012 年 4 月以来，CloudStack 与 OpenStack 的社区人口增长速度基本相当，但是从 2013 年 4 月以来，OpenStack 社区人数剧增，这与大量 IT 巨头的加入密不可分。与 CloudStack 和 OpenStack 相比较，Eucalyptus 和 OpenNebula 的社区人口增长较为缓慢。

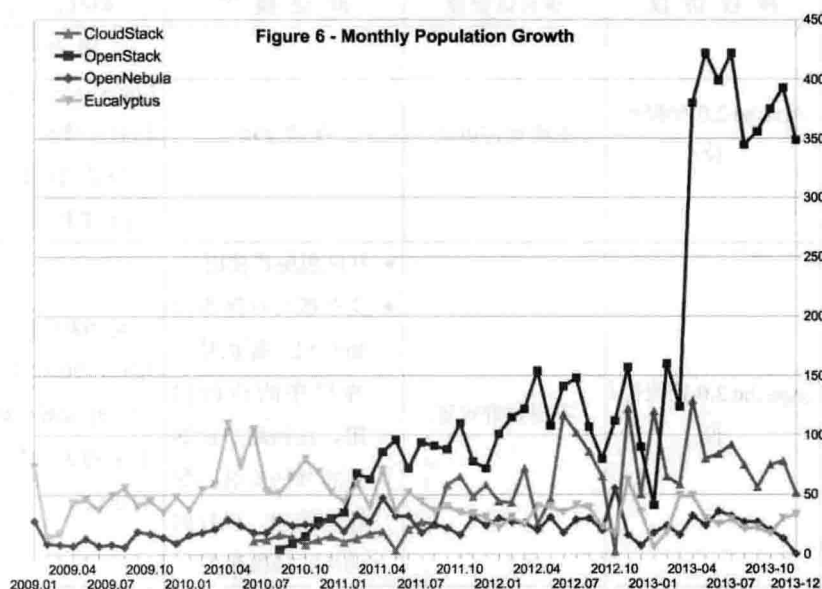


图 1-3 四种开源云平台社区人口数量对比

图 1-4 中实线部分表示的是每个月参与论坛或者邮件列表讨论的人数，虚线部分表示的是每个月新加入论坛或者邮件列表的人数。

从 2013 年年底看，在过去 12 个月中，OpenStack 与 CloudStack 项目的新增人口占当月活跃用户的 30% 左右，OpenNebula 与 Eucalyptus 项目大概是 50%。如果不考虑社区人口的规模的话，可以认为 OpenStack 与 CloudStack 社区的黏性大于 OpenNebula 与 Eucalyptus 社区。

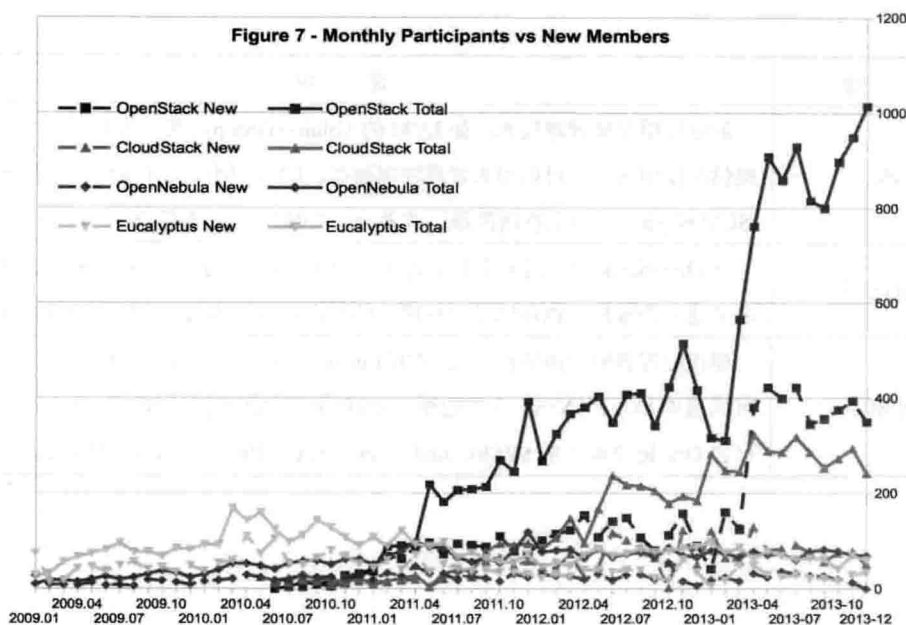


图 1-4 四种开源云平台总共和新增成员数量对比

1.9 术语

表 1-4 中列举了本书中常用的一些术语。

表 1-4 常用术语

术 语	说 明
节点/主机	单台物理机或虚拟机，如果用于计算，即运行虚拟机，则称为计算节点；如果提供网络服务，则称为网络节点；如果提供存储空间，则称为存储节点
虚拟机管理程序 (VMM, Hypervisor)	运行于物理机上一层软件，可以将一个物理机节点虚拟化成多个虚拟机，每个虚拟机可以运行不同的操作系统，有不同的配置
域	在虚拟机管理程序提供的虚拟机里运行的一个操作系统实例
卷	一个存储卷，可以挂载到单个虚拟机，或者用来生成资源池。一个卷可以是一个块设备、一个 Raw 格式文件，或其他格式文件。

续表

术 语	说 明
池	通过使用存储管理技术，如 LVM 的 Volume Group，然后从里面再切分成卷，提供给使用者。池可以用来管理物理磁盘、NFS（网络文件系统）服务器、一个 iSCSI target、一个主机适配器，或者一个 LVM（逻辑卷管理）。
项目/租户	在 OpenStack 里，这两个名字为同一个含义，原来指在公有云里以企业或组织名义建立的账户，也可以是一些用户的集合。这些构成了管理用户的单元
QEMU	模拟处理器的自由软件，在 GNU/Linux 平台上使用广泛，默认支持多种架构。可以模拟 IA-32（X86）个人电脑、AMD 64 个人电脑、MIPS R4000、Sun（目前已由 Oracle 收购）的 SPARCsun3 与 PowerPC（PReP 及 Power Macintosh）架构

第 2 章

OpenStack 社区



OpenStack 起源于 Rackspace 和美国航空航天局（NASA）共同开发的云计算平台，用于帮助服务商和企业内部实现类似于 Amazon EC2 和 S3 的云基础架构服务。OpenStack 开始时包含两个主要模块：Nova 和 Swift。前者是 NASA 开发的虚拟服务器部署和业务计算模块；后者是 Rackspace 开发的分布式云存储模块，两者可以一起用，也可以分开单独用。OpenStack 是开源项目，除了有 Rackspace 和 NASA 的大力支持外，还有如 Dell、Citrix、Cisco、Canonical 这些重量级公司的贡献和支持，发展速度非常快，有取代另一个业界领先开源云平台 Eucalyptus 的态势。另外，许多原来使用 CloudStack 开源云平台的企业也纷纷将云平台移到 OpenStack 上。

OpenStack 在 2011 年 10 月宣布将于 2012 年成立基金会的计划，在 2011 年剩下的时间里则探寻着发展模式，交流着成立社区的想法。2012 年 1 月，OpenStack 基金会的筹备工作启动，包括制定基金会的使命、组织结构和资金资助模型。2012 年 4 月，就已经有 19 家公司加入，大家承诺遵守组织结构和基金方面的要求，并开始起草法律文件用于审阅。2012 年 7 月，法律文件终于完成，而 Rackspace 则将 OpenStack 转交给 OpenStack 基金会来管理。

OpenStack 基金会成立于 2012 年 9 月，设有三个分支：技术委员会、用户委员会和董事会。董事会由 24 名成员组成，负责为该组织提供战略和财务监督；技术委员会是之前的项目政策委员会的延续，该委员会继续负责定义并指导个别项目的软件开发；新的用户委员会主要代表最终用户的利益。OpenStack 基金会作为一个独立组织，将确保 OpenStack 在长期内继续得到开发和支持。基金会总是希望在强势而希图关注自己业务与利益的大公司与投入许多时间并贡献代码的个人开发者中间保持平衡。图 2-1 为 OpenStack 官网对几个分支的一些简要说明和统计。



图 2-1 OpenStack 基金会组成

OpenStack 基金会是一个独立的实体，不属于任何一个公司、组织和个人。基金会的使命是提供共享的资源，来保护、培育和提升 OpenStack 软件和社区，包括周围的用户、开发者和整个生态系统。简而言之，OpenStack 基金会是 OpenStack 的联合国大会，通过资助现金和一定公开的选举、民主决策来决定接纳新的会员，决定它的发展方向、发展策略。例如，VMware、华为加入 OpenStack 的申请都需要基金会的讨论通过。基金会的董事会既有如 IBM、Red Hat 等公司成员，也有如独立董事杜玉杰先生一类的个人成员。最近的一次“民主”选举就发生在 2014 年 1 月份，全世界的 OpenStack 人都可以投票，也有许多候选人。但是加入 OpenStack 社区则是完全免费和开放的，并没有准入门槛和其他方面的限制。

今天，OpenStack 社区已经有超过 100 个国家的近 9 500 名个人开发者、贡献者，还有超过 850 个全球公司、厂商、各类组织参与。在社区的推动上，“技术委员会”负责整体地思考和管理全部 OpenStack 项目，而选举产生的“项目技术负责人”(Project Technical Lead)则管理项目内的事务并做出许多影响项目本身的决定。

OpenStack 社区贯彻着这样的原则。

- 技术专家负责技术，并做出技术的决定；
- 专门的资源来创建社区和整个生态系统；
- 一个生态系统才能创造财富；
- 鼓励并对各种贡献都进行奖励，这些贡献包括测试、文档、翻译、集成、扩展、教育、财务、培训、支持、协调、设计等。

OpenStack 于 2013 年 11 月在香港召开了全球峰会，这是全球峰会第一次在亚洲召开。在为期四天的会议中，有约 5 000 人参与大会，250 多场主题演讲。图 2-2 为香港峰会的宣传图片。

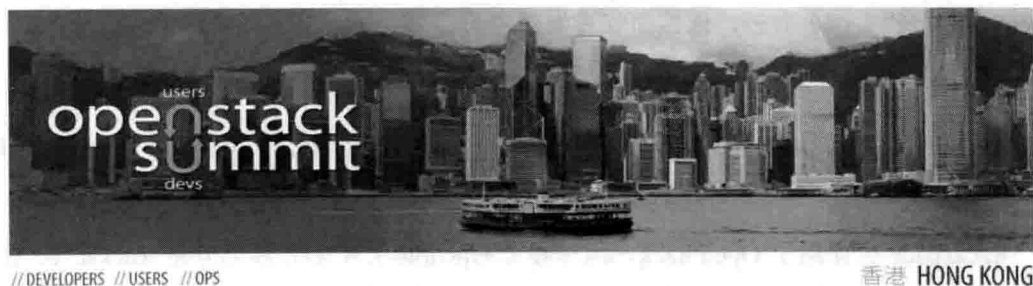


图 2-2 OpenStack 香港峰会的宣传图片

在香港大会，通过用户调研，得到下列一些数据。

- 超过半数的部署采用了 G 版或 H 版（Havanna）；
- 采用 OpenStack 的动机主要为降低成本、提升运行效率和开放；
- 美国、印度、中国、法国都是 OpenStack 受到极大关注的地区；
- 用户主要来自信息科技、电信、政府、健康等领域；
- 主要业务应用包括 Web 服务、测试环境、数据库、科研、持续集成自动化测试、存储和备份、虚拟桌面、数据挖掘/Hadoop、管理监控系统、HPC，部署类型主要为私有云服务；
- KVM（Kernel-based Virtual Machine）依然是主要的 Hypervisor，比例超过六成，LXC（Linux Container）以 4% 的比例排名第 5；
- 部署工具方面，Puppet、Devstack、Chef、Packstack、Crowbar 排名前五。

下一届 OpenStack 全球峰会将在美国亚特兰大举行，在 OpenStack 官网可以看到如图 2-3 所示的宣传图片。在 OpenStack 社区官网上可以查看到会议的详情。

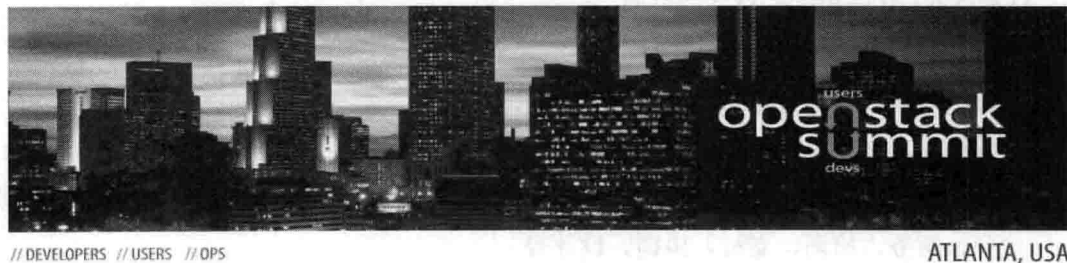


图 2-3 OpenStack 亚特兰大峰会的宣传图片

由于有一系列大公司的支持，加上本身非常良好的架构设计，最近几年，OpenStack 社区非常活跃，发展也非常快速，为云计算的落地推广起到了很大的作用。但是如果用 OpenStack 搭建大型公有云，依然面临着如下考验。

（1）目前全世界没有特别成功的 OpenStack 公有云案例。

AWS、Google、Windows Azure、阿里云使用的是自己研发的云平台和产品体系，而使用 OpenStack 进行大规模部署的只有 HP Cloud，但 HP Cloud 目前还在早期发展阶段。

Rackspace 也使用了 OpenStack，但实际上 OpenStack 中的存储部分是 Rackspace 提供的，目前还说明不了 Rackspace 全部是用 OpenStack 搭建的。

(2) OpenStack 是一个软件，不是服务，而公有云是服务。

实际上，做公有云，不但需要软件，还需要强大的运营来做支撑。

基于互联网运营的经验，用户看到的功能只是互联网产品里面的 20%，还有 80% 是用户看不到，但是运营需要的功能，如多账号体系、计费体系、安全体系、后台的自动化管理、版本发布、扩容等。

(3) 与开源项目协调的问题。

假设用 OpenStack 搭建公有云并运行起来，如果此时用户有需求，而这个需求是 OpenStack 现有的功能无法解决的，那么就将面临两难的境地：是去自行开发这样一个功能，开发之后，提交给社区，还是自己私有呢？如果提交了这个功能，社区又接受了，那么可以扩大影响力，两全其美。如果提交了功能，但是社区不接受；或者如果不提交这个功能，那么会面临升级、维护难的问题。将自行研发的功能，跟社区发布的新版 OpenStack 进行融合，会是比较困难的事情，而这个难度会随着时间推移而变得严重。

这些问题不但是用户需要考虑的，而且也是社区需要非常慎重思考的。

2.1 OpenStack 基金会

OpenStack 基金会包括以下三类会员。

- **个人会员**：包括那些以个人名义加入社区，或者代表公司加入的个人。加入基金会是免费的，而且个人会员可以竞选或者投票来参与领导角色。
- **白金会员**：包括那些以战略性角度参与并投入资金和资源的公司。每个白金会员可以指定一个董事会的代表来参与重大决策。
- **黄金会员**：包括那些投入资金和资源，但是比白金会员低一级的公司。它们作为一类群体来选举董事会。

董事会的选举规则：

- 个人会员选举 1/3 席位；
- 黄金会员选举 1/3 席位；
- 白金会员任命 1/3 席位。

因此，没有一个公司可以控制超过两个的董事会席位。

2.2 白金会员

如图 2-4 所示为 OpenStack 基金会的白金会员。



图 2-4 OpenStack 白金会员

2.3 黄金会员

图 2-5 中列举了 OpenStack 基金会的黄金会员。



图 2-5 OpenStack 黄金会员

2.4 OpenStack 设计原则

OpenStack 的所有项目，在涉及时都必须遵守如下一些原则。

- 可扩展性和伸缩性是我们的主要目标；
- 任何影响到可扩展性和伸缩性的功能都必须是可选的；
- 所有的环节必须是异步的，如果不能异步实现，参考第二条设计原则；
- 所有的基础组件必须能横向扩展；
- 始终使用无共享的架构，如果不能实现，参考第二条设计原则；
- 所有的组件部署形式都是支持分布式的；
- 接受最终一致性，并在适合的条件下使用；
- 测试一切。

2.5 开源而开放的原则

开源意味着自由，开源引领着创新，这是由下列精心设计的一些原则和机制保证的。

- **Open Source:** Apache 2.0 许可，企业友好。
- **Open Design:** 6 个月一次，基本与 Ubuntu 同步。
- **Open Development:** 社会化研发。
- **Open Community:** OpenStack 基金会。

2.6 OpenStack 版本管理

OpenStack 每 6 个月发布一个新版本，如表 2-1 所示为 OpenStack 版本列表。

表 2-1 OpenStack 版本列表

版本代号	时 间	内 容
Austin	2010 年 10 月	Nova, Swift
Bexar	2011 年 2 月	Nova, Swift
Cactus	2011 年 4 月	Glance
Diablo	2011 年 9 月	第一个 “Prouction Ready”
Essex	2012 年 4 月	Horizon, Keystone
Folsom	2012 年 9 月	Quantum, Volume
Grizzly	2013 年 4 月	Neutron
Havanna	2013 年 10 月	Heat, Ceilometer
Icehouse	2014 年 4 月	Sahara (原 Savanna)

2.7 OpenStack 用户

在 OpenStack 官网 (<http://www.openstack.org>) 上有专门的版面，收集用户部署方面的信息，如图 2-6 所示为社区收录的分行行业统计。如图 2-7 所示的是其中一部分典型的大家比较熟悉的公司用户。事实上，到今天 OpenStack 已经得到非常广泛的应用。在国内，许多公司正在从 CloudStack 平台往 OpenStack 平台迁移。另外，由于 VMware 高昂的许可 (license) 费用，不少寻找替代方案的用户也在考虑迁移到 OpenStack 云平台。在笔者参与的不少项目中，客户都提出了迁移的需求，希望能帮助他们以一种可靠而且自动化的方式完成迁移到 OpenStack 的工作。

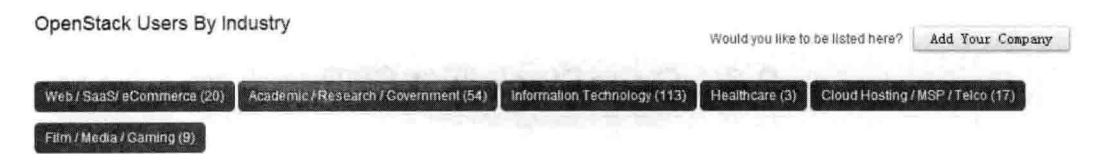


图 2-6 OpenStack 社区收录的部署分行行业统计



图 2-7 OpenStack 典型客户

国内典型客户及使用案例

图 2-8 中列举了 OpenStack 在国内的一些非常有影响力的客户,表 2-2 中列举了部分客户的典型应用场景。这些资料都可以从互联网上公开获得。



图 2-8 OpenStack 国内知名客户

表 2-2 国内知名客户及应用场景

公 司	应 用 场 景
瞬联	<p>瞬联私有云：</p> <ul style="list-style-type: none">需求一 开发、测试环境的需求。<ul style="list-style-type: none">占据瞬联业务重要部分的就是 Saap 的模式，在瞬联内部有大量为不同客户项目提供解决方案的团队，涵盖电信、移动互联网等诸多方面，其中也包括企业私有云解决方案团队。通过 OpenStack 提供的镜像服务以及虚拟机服务可以方便、快速地保存、重建开发与测试环境，大大缩短了环境重建的周期，提高了效率，也提高了资源的利用率。需求二 共享存储——私有网盘。<ul style="list-style-type: none">基于群组的文件共享也是重要需求之一。开发了针对 pdf、Office 文档及图片等常见文件格式的在线浏览功能，同时还提供群组分享、内容检索等服务。需求三 VDI 远程桌面。<ul style="list-style-type: none">远程桌面系统，用来解决远程办公、软件共享、安全控制等功能
趣游	<p>2011 年开始“趣云”</p> <ul style="list-style-type: none">基于 Swift 实现了一个图片存储的服务：<ul style="list-style-type: none">图片存储数据备份虚拟化现在是在使用 KVM+CentOS 6。部署架构是经典的架构图（控制节点+计算节点+FlatDHCP），提供统一的 UI，然后来提供计算服务，包括：<ul style="list-style-type: none">游戏服务开发与测试论坛网站服务部署（chef+cobbler）
携程	<p>“技术要永远领先于业务一步”</p> <p>因为前端业务增长太快，如各种新业务的不断涌现，网站流量急剧增长（该公司网站流量一年增长一倍），给后端的支撑平台带来了成本、效率、可靠性等各方面的巨大压力，迫使其不得不考虑使用云计算等新兴技术。</p>

续表

公 司	应 用 场 景
携程	<p>因此,携程从2012年开始搭建私有云。系统选型时,在CloudStack、OpenStack 和 OpenNebula 中,该公司选择了 OpenStack, 现在的部署基于 G 版本。同时,该公司还基于 OpenStack 部署了超过 13 000 个虚拟桌面。“我们可能是第一个用 OpenStack 技术部署虚拟桌面的。”技术副总裁和首席架构师叶亚明说。而携程之所以选择 OpenStack, 主要原因还是因为它是一个被广泛使用, 拥有广泛生态系统的开源云平台。</p> <p>私有云和虚拟桌面给携程带来的好处是显而易见的, 如成本的节约、敏捷与弹性的提升等</p>
360	<p>360 从 2012 年 Q3 开始构建其 OpenStack 的环境, 最早基于 Folsom 稳定版, 采用 CentOS 5.4 作为宿主机, Xen 3.4.4 作为虚拟化系统。目前, 该环境的规模达到了 4 000 多个实例, 分布在 20 多个数据中心上, 为 10 多个产品线服务, 包括云杀毒、开发测试环境、在线游戏、build、Hadoop 客户端、Web 应用、基础架构工具等。目前由 OpenStack 系统管理的实例数量的比例在 360 内部已经达到了 40%, 预计在 2014 年年底会达到 60%</p>
文思海辉	<p>整个公有云目前用到的物理机在几百台的规模, 分布在北京、佛山、香港等地的多个机房中, 集群分布还在不断向各个主要国内节点的机房扩展。根据目前的增长速度, 预测到明年将达到约 30 000 个实例的规模。</p> <p>HSCloud 之前基于 OpenStack Essex 版本进行主要扩展。最初选型的时候其实考虑过 CloudStack, 不过因为觉得 OpenStack 的支持厂家更多, 社区更活跃, 在中国的 OpenStacker 人数又比较多, 所以选择了 OpenStack。目前看来选择是正确的</p>

2.8 OpenStack 的误区

(1) OpenStack 是开源云平台管理软件, 为什么需要预算?

OpenStack 是开源而免费的, 没有许可费用。然而, OpenStack 是非常庞大而复杂的一套云平台管理软件, 上千的全球工程师努力工作来提高代码, 增加功能, 所以它是在天天变化的, 使用一个最新的组件同时也需要更新其所依赖的全部组件。

问题还在于, 即使最新的代码也是不稳定的, 社区在不断修补严重的代码问题。有时候, 你需要自己修改某些问题, 而不能等到社区来修复。所以即使代码本身是免费的, 但总是在变化, 而时间就是金钱。

(2) 我可以自己做。

如果你的整个云平台足够小，你可以自己来实施一个基于 OpenStack 的云平台。然而，你总是需要一些帮助的。但往往，很多云平台并不是这么简单，你必须理解你需要的。你是需要一个公有云，私有云，还是混合云？你的应用是需要多租户，长期运行，短期运行，专用的，非持久化的，稳定的，还是突发式的，或者包括上述部分或全部？

另外，云平台还不是问题的全部，再看看你现在的应用，它们是需要继续在现有环境中运行，还是需要迁移到云平台？

这些都不是可以简单做出决定的。

(3) 每个人都理解术语。

可以认为每个人都理解这些术语，然而，非常关键的是，你需要把你的需求，如“谁、做什么、为什么去做、什么时候去做、如何做”等作为一个整体去考虑。

即使在 OpenStack 和云业界，对一个术语和概念的理解，也总是充满着争论。

(4) 旧有系统将会消失/被迁移。

非常乐观的观点认为，在云平台搭建和迁移完成后，旧有系统将会消失。然而这不是现实，即使最古老的 COBOL 程序员依然可以找到工作。旧有应用不会消失，一个旧有系统，如数据存储类、交易类、财务类、保险类应用，还没有准备好迁移到云平台。这些迁移也会影响公司业务流程，需要修改公司业务规则。

即使旧有系统会被迁移，速度也不会很快。

(5) 全部需要的就是负载均衡。

这些观点来自于一些乐观的思考，认为云平台是一个大型路由器，运行了一些无状态应用，所需要的是考虑将这些应用放在哪里运行，以及出问题，只需要重启该应用即可。

然而在云平台的设计规划的时候，我们需要考虑在云平台运行什么样的应用？是开发测试环境？是否可以随时启动/停止？可以在紧急时停止应用？是一个组件，还是一群组件？很多时候，往往并不能通过复制应用来扩展应用。不是所有应用都可以通过复制来保持一致性和状态，除非它们以前就是这样设计的。

(6) 我们不需要开发。

在 OpenStack 云平台，应用会有比在传统环境多得多的控制手段，需要更多的协同，

需要透彻地了解架构、技术和操作流程。

为了更好地管理和运行 OpenStack 云平台，管理员、操作员和开发者需要更多的协同。

(7) 我们员工有需要的技术。

很多人基于使用 Linux 的经验，认为 OpenStack 是个非常简单的软件。如果你的公司拥有专家级别的，熟悉开源 IP 网络、Hypervisor 管理、存储冗余和优化、开源代码管理、安全与加密、驱动优化、分布式应用架构，和许多其他技术的人才，那么你是对的。然而，很多情况下，你总会缺少一部分或很多这类的技术人才储备。

任何人都会用 Linux，但不是每个人都是内核专家，也许会有这类专家，但不是一夜就会有的。

(8) 我们可以不需要花钱。

很可能，你需要购买新硬件，这些设备并不便宜，你还需要培训员工，需要建立一个数据中心。

而且你可能需要一个新的业务模式，你现有的基础设施在搭建时，是基于不同的功能和业务属性的。假设某天，他们都不存在了，你的业务还能运转吗？

理解云平台的价值，首先需要理解你的用户的业务和经济模型。

(9) 云是自修复的。

云平台虽然是高度容错的，但还没有这么神奇。它可以自修复，但是你得确保有合适的监控和冗余，特别是负载已经接近阈值时。往往在系统出问题且不能自我修复时，你才知道问题发生了。

(10) 故障不是选项。

在传统应用场景里，故障不是个选项，人们用各种技术手段来避免故障。但是在云世界里，故障是个必然，是云平台设计的一个核心理念和原则。你只需要让你的系统和应用准备某种故障发生，让云平台来替你调整。

“自动化所有事情”，特别是感知故障发生。这样，即使有故障发生，你依然可以享受到云平台这种新技术带来的快乐。这也是云平台为你带来的最大的收益。

2.9 部署 OpenStack 的技术需求

学习和部署 OpenStack 需要大量的系统方面的知识。

- 熟悉需要使用的 Linux，包括安装、系统管理、文件系统、网络等。
- SQL 数据库的安装、配置、管理和调优。
- 虚拟化的知识和一些使用经验。
- 网络和多个 Linux 主机之间的网络：
 - DHCP;
 - Linux Bridge;
 - VLAN;
 - Iptables。
- 网络：
 - 交换机;
 - 路由器;
 - 防火墙。
- 存储，包括：
 - LVM;
 - 存储设别;
 - 分布式存储;
 - 块存储;
 - 对象存储。
- 物理机/BIOS。
- 脚本编程经验。

第 3 章

A horizontal line extends from the right side of the chapter number, followed by a vertical line that ends in a small solid black square.

OpenStack 与 AWS、VMware、 虚拟化管理工具

A horizontal line spans the width of the page, ending with a large solid black square on the right side.

3.1 OpenStack 与 AWS 的比较

亚马逊在经过多年的发展并取得极大的成功之后，事实上已经成为公有云的标准。尽可能吸纳亚马逊 AWS 的功能，兼容 AWS 的接口成为各个正在成长的云平台的目标。图 3-1 为亚马逊公有云功能首页。



图 3-1 亚马逊公有云功能首页

尽管在 OpenStack 世界里，没有任何组件称为 EC2 或者 S3 或者其他，但事实上，OpenStack 无论在功能上，还是 API 接口上，都尽最大可能地与 AWS 保持一致。

从功能上，OpenStack 尽管目前远弱于 AWS，因为 AWS 已经是一个正在运营，并且已经在全球建立多个数据中心的超大型公有云，已经开发了非常完善的、对应于各种用户需求的功能集合；然而 OpenStack 定位在同时满足公有云和私有云，并不是来自于一个具体的企业或服务商的部署环境，所以在一些辅助或次要功能，或在运营管理上，注意力远远不够。

我们完全可以找到它们的对应关系。比如，EC2 在 OpenStack 里是由 Nova 项目和其组件提供的，VPC 是由 OpenStack 里的 Neutron 及其各种组件、插件共同提供的。

即使在一些不常用的功能，如 SQS、OpenStack 里也有正在孵化的项目在努力追赶。

在接口上，OpenStack 则保持了完全的兼容。也就是说，如果客户开发了基于亚马逊

API 的管理业务逻辑，当把企业应用迁移到自己搭建的基于 OpenStack 的云里时，这些管理业务逻辑也是完全可以重用的。反之，如果客户既有自己的基于 OpenStack 的私有云，又把一部分业务部署于亚马逊 AWS 里，也完全可以用同样的接口开发管理业务逻辑。

表 3-1 列举了一些主要功能的对应关系。需要说明的是，空白或“*”标示的栏目并不表示 AWS 或 OpenStack 里没有该功能，只是在名称或具体功能上有差异，我们不想强制地说明它们是否具有对应关系。AWS 是全世界最大的公有云平台，其功能和业务之多，也许远远超出我们的想象。我们希望能避免人们产生某种误解或争执。

表 3-1 OpenStack 与 AWS 功能对比

AWS	OpenStack	功 能
EC2	Nova	计算，即虚拟化管理程序，根据需求，创建虚拟机并调度到计算节点。支持多种虚拟化软件，如 KVM、Hyper-V、Xen、LXC、ESXi
VPC	Neutron	网络管理的组件，根据需求，管理网络地址空间，并分配 IP 到指定的虚拟机，其他还包括虚拟网桥、三层路由等
S3	Swift	对象存储的组件，是 OpenStack 所有组件中最成熟的
IAM	Keystone	提供身份认证和授权的组件
EBS	Cinder	存储管理的组件，主要是指虚拟机的存储管理
CloudFormation	Heat	该项目解决虚拟机的各种配置软件部署的问题
Console	Horizon	图形化操作控制界面
DynamoDB, RDS	Trove	Database as a Service，该项目的目的是为关系型和非关系型数据库引擎提供可扩展的、可靠的云数据库作为服务的定制与配置功能，并持续改善和提高开源框架，提供更全和可扩展的功能集
Elastic MapReduce	Savana	OpenStack 关于大数据的方案
	Murano	用于支持 Windows 和 Linux 平台企业级系统
ELB	LBaaS	实现负载均衡的项目，目前这个项目集成在 Neutron 里，不过基本算是一个独立的组件
	FWaaS	
*	Ceilometer	实现监控和计量的组件
*	Moniker	实现 DNS 功能的组件
*	Marconi	用于解决 OpenStack 消息队列的扩展问题

下面我们会用一些屏幕截图来给大家一些直观的认识和感觉。

图 3-2 是亚马逊 AWS 创建新的虚拟机过程的其中一个界面。请注意其中需要填写的信息字段。在不同的虚拟化或不同技术实现的云平台管理端，这些信息都是大同小异的。

图 3-2 AWS 创建虚拟机实例

不同界面，如亚马逊 AWS、VMware vCenter，以及 OpenStack Dashboard，对创建虚拟机的流程差异比较大，但从基本信息角度来看，无外乎选择虚拟机名称、数量、操作系统，或者虚拟机模板、网络信息、安全访问信息等。

另外，亚马逊提供的选项更多一些，因为全球有多个数据中心，还需要选择虚拟机运行于哪个数据中心，是否采用网络直连（需要事先部署 1G/10G 的专用网络连接），是否需要 Auto-Scaling 服务（它可以在停止虚拟机或其他问题时，自动地重新启动虚拟机，是典型的云感知应用场景）。

OpenStack 也可以允许用户在创建虚拟机时定义一些硬件特性信息，主要用于调度算法。

VMware vCenter 做得更精细些，可以让用户根据运行主机的硬件特性做一些深度的配置，另外还可以配置如高可用性、容错方面的设置。

图 3-3 是 OpenStack Dashboard 云主机管理界面，目前默认提供两个视图，分别为管理员看到的系统全部功能操作界面，以及只有项目（租户）登录后的界面。

图 3-3 中显示的界面，包括虚拟机属于哪个项目、名称、使用的镜像名称、IP 地址、状态等。在最右侧则有一个下拉式的操作列表，可以暂停虚拟机、终止虚拟机、做快照等。



图 3-3 OpenStack Dashboard 云主机管理界面

图 3-4 为 OpenStack Dashboard 的“启动云主机”界面，需要填写要创建的虚拟机信息，包括区域、名称、数量、启动方式等。另外，在后续界面里，还可以配置安全访问、网络，以及创建后初始运行的一些初始化脚本程序等。

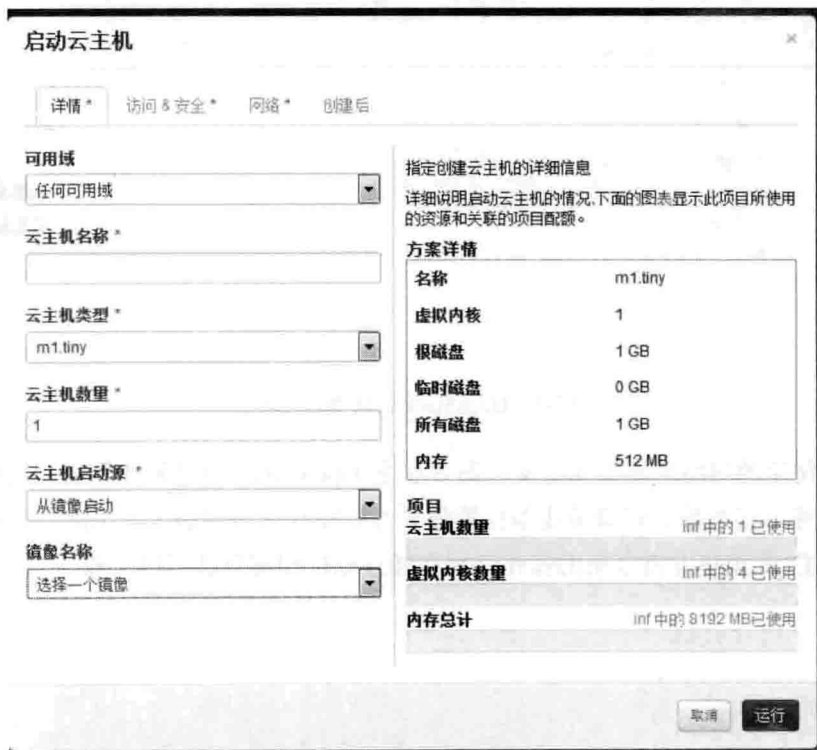


图 3-4 OpenStack Dashboard 的“启动云主机”界面

OpenStack 用“主机类型 (flavor)”来定义虚拟机的规格，这个也是和亚马逊相似的，包括 vCPU 数目、内存、硬盘大小等信息。对于 OpenStack 的 flavor，用户可以增加一些扩展信息，如硬件平台信息，或扩展的采用键值对形式的自定义内容，在虚拟机调度时使用。

图 3-5 是 OpenStack 云硬盘管理界面，显示了项目、虚拟当前运行物理主机、大小、状态等信息。和传统的计算机一样，一个虚拟机也可以“挂”多个硬盘，在 OpenStack 里称为“卷 (Volume)”。从虚拟机看到的硬盘和从物理机看到的硬盘一样，可以格式化成各种文件系统，在上面存储数据。

云硬盘类型则可以对云平台后端提供“卷”的存储做进一步的区分，比如从性能、转速或本地硬盘/存储服务等角度来定义，不同的类型对应着不同的客户服务等级。

云硬盘也可以创建、挂载、下载和删除；其他操作还有快照、克隆、迁移等。



图 3-5 OpenStack 云硬盘管理界面

虚拟机使用的网络需要事先定义，图 3-6 是 OpenStack 网络管理界面，包括项目、网络名称、子网、状态等。管理员或项目管理员可以在网络里增加多个子网，创建路由器。这些在传统 IT 环境里非常复杂的操作，现在通过鼠标和键盘就可以完成。



图 3-6 OpenStack 网络管理界面

图 3-7 为 OpenStack 虚拟机管理器界面，显示的是所有计算节点，以及每个计算节点的资源使用情况，包括虚拟内核总数、当前使用的数量、内存总计和当前使用量、存储总计和当前使用量，以及每个计算节点当前运行的虚拟机实例数量等。



图 3-7 OpenStack 虚拟机管理器界面

如图 3-8 所示，在这个界面可以查看系统各种信息，如当前运行的服务列表，提供计算服务的节点（主机），所在域、状态和持续运行时间等。如果系统某些功能运行不正常，可以先到这里查看这些服务是否处于“down”的状态。这种状态可能是服务本身有问题，

但也有可能是该节点和主控节点的网络连接出了问题。

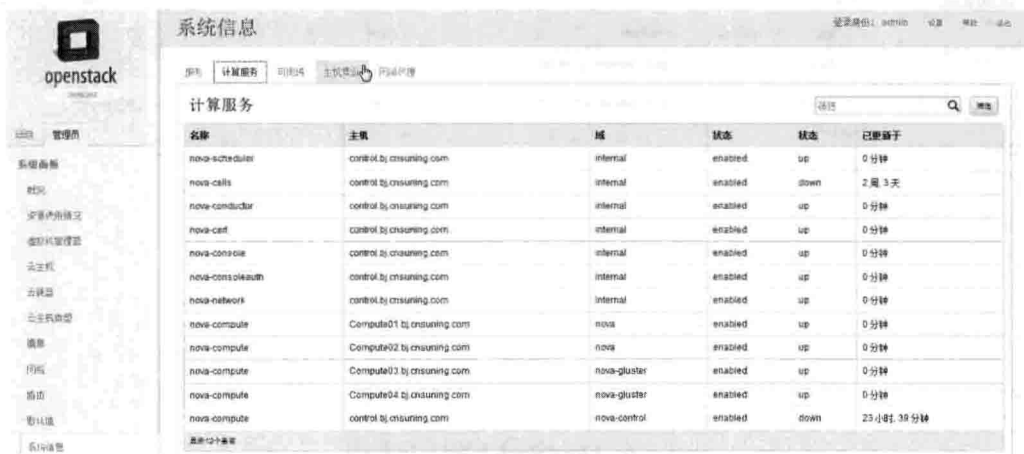


图 3-8 OpenStack 系统信息界面

图 3-9 是 OpenStack 里当前部署的代理 (Agent) 列表。这些代理包括网络三层、二层、DHCP 服务等。网络问题在任何环境都是大问题，网络不通，可能的原因非常多，但直接后果是用户对环境无法使用，常常影响面非常大。这里可以初步查看各种代理的信息以及运行状况。



图 3-9 OpenStack 界面中的代理信息

在 OpenStack 里，还可以显示各种网络拓扑。图 3-10 显示了，在一个网络里有多个虚拟机存在。还可以显示多级、复杂的网络，包括软件定义路由器、浮动 IP、子网等。



图 3-10 OpenStack 的网络拓扑视图

3.2 OpenStack 与 VMware 对比

企业用户已经在大量使用 VMware 的产品，所以经常会将 VMware 的 vCenter/ESX 和 OpenStack 做比较。然而，两者是完全不同的东西，就像把苹果和香蕉做比较一样，其结果无法说明什么。

VMware 提供了非常完善的虚拟化管理软件和工具集，比较好地解决了企业问题，而 OpenStack 则是一个开源项目，一组搭建云平台的项目/组件集合。两者提供的功能也有很大差异。

如果一定要进行对比的话，则 VMware 的虚拟化管理类似于 OpenStack 的 Nova 项目，但在具体功能和实现方式上仍然差异很大。专家们比较一致的看法是：两者并不具有太多可比性。

一个令人欣喜的消息是，VMware 也加入了 OpenStack 基金会，并成为其黄金会员。VMware 在 OpenStack 里的工作，主要集中在从 Nova 角度支持其 Hypervisor ESX/ESXi，还有 SDN。对 ESX/ESXi 的支持还在进行中，但有希望在几个月后，能在统一的 OpenStack 部署平台，同时运行 Linux 的 KVM、VMware 的 ESX/ESXi、微软的 Hyper-V 和 IBM 的 Power。

VMware 的 vCenter 和 ESX/ESXi 提供了非常完善的虚拟化支持。图 3-11 为其围绕着虚

拟化和虚拟机提供的功能。

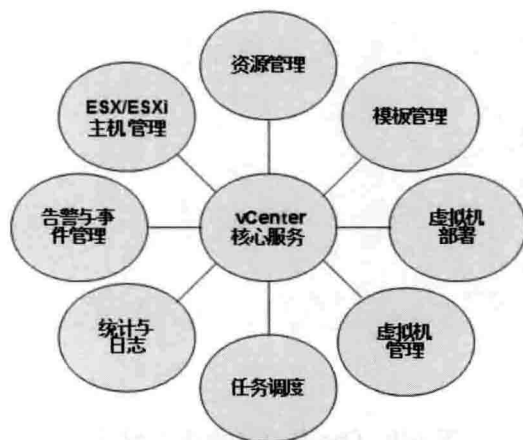


图 3-11 VMware vCenter & ESXi 功能

如图 3-12 所示，在用 VMware 虚拟化管理程序创建虚拟机后，在本地目录会生成一些文件。这些文件因虚拟化管理程序而异。但基本包括磁盘镜像文件、配置文件，以及其他如元数据文件等。

名称	修改日期	类型	大小
cache	2014/7/5 22:54	文件夹	
CentOS-RDO.vmx.lck	2014/7/6 12:53	文件夹	
CentOS-RDO	2014/7/6 9:08	VMware 虚拟机 BIOS	9 KB
CentOS-RDO	2014/7/6 9:08	VMware 虚拟磁盘文件	20,737,72...
CentOS-RDO	2014/7/5 6:34	VMware 快照元数据	0 KB
CentOS-RDO	2014/7/6 9:08	VMware 虚拟机配置	3 KB
CentOS-RDO	2014/7/5 6:54	VMware 组成员	4 KB
vmware	2014/7/6 9:08	文本文档	168 KB
vmware-0	2014/7/6 8:25	文本文档	166 KB
vmware-1	2014/7/6 0:24	文本文档	332 KB
vmware-2	2014/7/5 6:58	文本文档	227 KB
vprintproxy	2014/7/6 9:08	文本文档	13 KB

图 3-12 VMware 虚拟化管理程序创建的本地文件

图 3-13 为 VMware 虚拟机管理配置界面。在今天，各个虚拟化管理工具，从基本的虚拟机管理、启停、配置等角度，功能方面的差距已经很小了。但是 VMware 的虚拟化管理，作为一个商业产品，其各方面，从界面、操作到信息显示，都代表了一个高度，其占据虚

拟化市场很长时间，高达 60% 以上，绝对是名至实归的。



图 3-13 VMware Workstation 虚拟机管理配置界面

图 3-14 展示了在 VMware 虚拟化管理程序中可以非常灵活地定义网络，配置 QoS 参数等。OpenStack 社区提供的界面中并没有这样的界面，很多功能都必须通过命令行来进行，而且命令和参数非常多，对 OpenStack 不太熟悉的用户往往是很难操作的。

如图 3-15 所示，在对虚拟机配置完网络之后，宿主机就可以看到一些网络设备，这些设备有多种方式，有 NAT，网桥形式的，也有主机模式的。不同网络模式，访问外部网络方式不同，如 NAT 则无法从外部访问虚拟机，而主机模式可以容易地在宿主机和虚拟机之间通过网络互访。



图 3-14 VMware 管理界面管理网络和 QoS

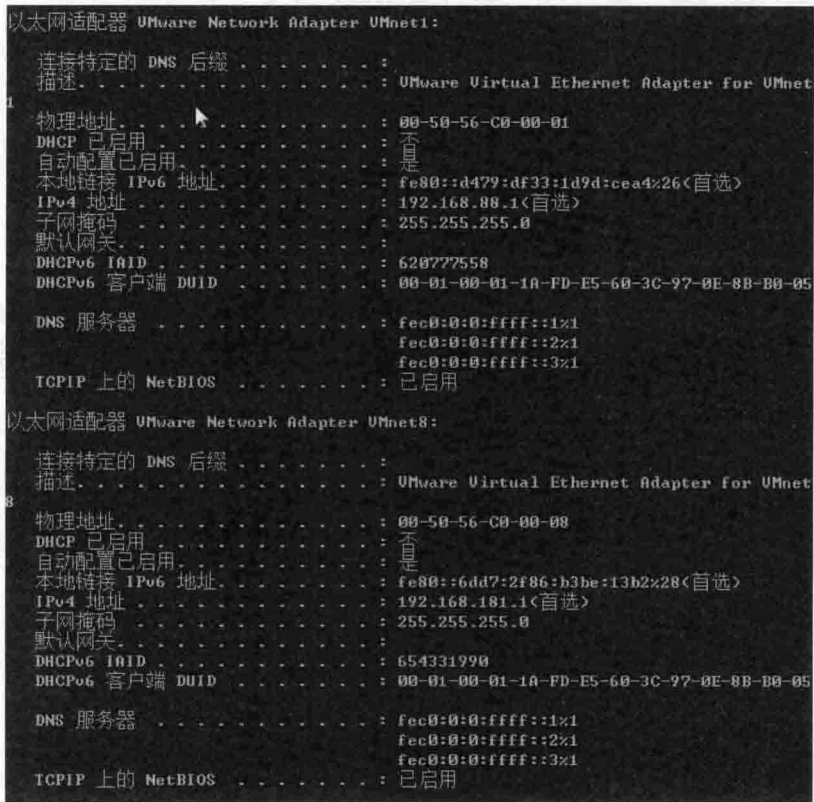


图 3-15 VMware 创建的网络

图 3-16 是虚拟机管理程序的“选项”界面。对于一个虚拟机管理工具来说，管理和启停虚拟机是远远不够的。这里涉及和宿主机共享文件、电源、快照等各种其他功能，也是用户常常需要的。

VMware 产品在这些方面也做得非常精细，特别是 VMware Tools，通过在虚拟机里部署代理（agent），对于从主机管理虚拟机、抽取虚拟机信息，非常实用。



图 3-16 VMware 虚拟机管理程序的“选项”界面

VMware 的产品可以清晰地划分为以下四层。

- (1) **ESXi Hypervisor**: 裸机上运行的 Hypervisor，类似于 Citrix XenServer 或者 KVM。
- (2) **vSphere**: vSphere 基本上是个商业化的 ESXi，包括了一系列非常优秀的企业级功能，诸如热迁移、高级安全、网络 and 存储 I/O 控制等。
- (3) **vCenter**: vCenter 是一个集中的编排/控制面板（即操作界面），可以管理许多节点。
- (4) **vCloud Director**: vCloud Director 可以把基于 VMware 虚拟化的数据中心变成类似 IaaS 的服务，其主要功能包括自服务界面、计费 etc。

根据最新的信息, VMware 基本上加入 OpenStack 阵营, vCloud Director 完全可以被 OpenStack 的类似功能替代, 因此可以忽略。

对 OpenStack 和 VMware 的比较很难, OpenStack 的 Nova 计算、Glance 镜像、Neutron 网络和 Dashboard 控制面板基本上覆盖了 VMware #2、#3 层的功能; 而 Swift 对象存储、虚拟负载均衡器则无法对应到 VMware 产品系列去, 目前没有发现相对应的产品或功能。

3.2.1 VMware vMotion 与 OpenStack 动态迁移、块迁移

vMotion 是 vSphere DRS、DPM 和主机维护三大功能的合集。其中虚拟机动态迁移允许将一台虚拟机在不关机的情况下由一台宿主机迁移到另一台上, 这原本是需要共享存储的支持的, 但在 vSphere 5.1 中, VMware 已经不需要通过共享存储实现动态迁移了。当一台虚拟机由一个宿主机迁移到另一个上时, 虚拟机的内存状态和数据都要同步迁移过去。如果是共享存储的情况, 实际上数据是不需要进行迁移的, 只需要变化指向数据存储的链接而已。这在加速了迁移速度的同时也减少了在复制过程中网络的负载。

OpenStack 中, KVM 的动态迁移依赖于共享存储, 允许一个虚拟机由一个虚拟机管理器迁移到另一个, 一台虚拟机也可以来来回回在 AMD 架构主机与 Intel 架构主机上进行迁移, 但是需要注意的是, 64 位的虚拟主机只能被迁移到 64 位的宿主机上, 但是 32 位的则有 32 位和 64 位两种选择。在动态迁移过程中, 不能再对虚拟机进行操作, 但是虚拟机内的用户还是可以在虚拟机内部继续进行工作的。

共享存储, 可以是外部 SAN/NAS 存储, 但各个节点必须部署并行文件系统, 如 IBM GPFS; 或者是本地硬盘, 通过如 GlusterFS、GFS 等分布式集群文件系统来提供。

动态迁移需求如下。

- 虚拟机存储需要放在分布式文件系统之上, 如 NFS 或在 GlusterFS;
- Libvirt 必须要开启 listen flag;
- 每一个计算节点(虚拟机管理器)都必须在同一个网络/子网当中;
- 计算节点间的认证必须提前配置。

分布式文件系统/共享存储的挂载节点在每一个计算节点必须保持一致。

在 OpenStack 当中, KVM 也支持块存储迁移, 这也就是说虚拟机迁移共享存储并不是必须需要的。在块迁移的场景下, 虚拟机的内存状态与数据都将被迁移, 但是迁移操作也

需要消耗两端的 CPU 资源并且操作花费时间比共享存储要长。在某些用户场景当中，如果比较关注主机的可维护性，并且不想花费过多费用，那么应用块存储迁移将是好的解决方案。同时，如果在没有共享存储的环境中，需要对计算节点进行内核维护、安全升级，那么保证虚拟机服务不被打断，块存储迁移也是理想选择。

3.2.2 VMware DRS、DPM 与 OpenStack 调度器

基于 vMotion，DRS 可以动态监控虚拟机及宿主机的当前使用状况，并且为宿主机的负载均衡提供支持。包括：

- 部署阶段：可以对监控虚拟机执行自定义自动化脚本；
- 监控阶段：DRS 可以在 ESX(i)主机上分布式部署虚拟机，并且持续监控活跃虚拟机和可用资源，以动态迁移虚拟机来实现资源利用率最大化。

基于 vMotion，DPM 将虚拟机从低负载宿主机迁移掉，并且关闭以达到减少电能损耗。当负载增长，DPM 将宿主机重启，并且部署新的虚拟机以满足负载需要。

OpenStack 包含了对于计算和存储资源的调度器，通过一系列的管理员设定的规则参数和过滤器来进行，OpenStack 调度器将虚拟机部署到合适的宿主机上。在过滤器方面，调度器是非常灵活的，用户可以自己完成 JSON 格式的过滤器，并且过滤器还包含很多预定义的过滤器。虽然 OpenStack 调度器非常灵活，但是还是不能完全替代 DRS，原因如下。

- 调度器选择哪个宿主机进行虚拟机部署，其静态参考数据来源于 Nova 的数据库。也就是说，如果发现宿主机已经有了 4 台虚拟机了，那么调度器要选择一个新的宿主机去部署下一台虚拟机。
- 调度器只能在虚拟机部署阶段影响部署的位置，一旦部署完成，虚拟机运行后则无法挪动虚拟机了。如果需要基于动态数据进行调度，那么调度器需要与外部监控解决方案如 Nagios 合作。总而言之，目前 OpenStack 调度器将只会对部署虚拟机环节有影响。

3.2.3 VMware 与 OpenStack 的高可用

在 vSphere 中，虚拟机级别的高可用性允许在虚拟机或者 ESX(i)主机出错时，在不同宿主机部署相同的虚拟机。高可用是在硬件出问题的时候保证虚拟机的正常工作，如果真

的出错了，那么只能在不同的 ESX(i)主机上启动虚拟机，这也可能造成服务的中断。

OpenStack 目前并没有官方声明支持的虚拟机级别的高可用性，这个特性在 Folsom 版本被提出，但是后续又被放弃了。目前 OpenStack 已经提供了基本功能，称为 Host Evacuate，其作用是为 OpenStack 提供虚拟机级别高可用支持。但对于触发此功能，还需要自己集成对集群内主机状态的监控和触发的动作。

3.2.4 VMware 与 OpenStack 的容错 (Fault Tolerance)

VMware 容错机制是通过监控虚拟机的状态和所有变化，将这些变化同步到第二台备份 ESX(i)服务器之上。容错的概念在于无论是主还是从宿主机出现问题，只要一方能正常工作，那么宿主机上的虚拟机都保持正常工作。这种机制还是无法解决虚拟机中的应用程序崩溃，因为一旦一方崩溃，则这个变化也会同步到从节点，当停止虚拟机的服务去修复它，从节点也会同样停止服务。所以这个机制只能保证单点失效的问题不再出现，而真正的应用层面的容错则需要其他工具配合才能解决。考虑到其他方面如最高资源使用、内存、硬盘、CPU、带宽的容错，这些方面都有局限性，并且是使用量相对较小的功能。如实现这些则需要双倍的内存，由于内存不能跨主机复制，并且还需要 CPU lockstepping 去同步每一个 CPU 指令，这将导致只有单独一个虚拟 CPU 被容错机制所监控。

在 OpenStack 中没有针对于容错的功能，并且截至目前也没有计划去完成这些功能。未来，KVM 也不再支持镜像操作功能。

3.2.5 总结

可以看到，在功能的支持方面和功能细节，OpenStack 与 VMware 还是有差距的，但是这对 OpenStack 还是有优势的，因为相比 VMware 的昂贵价格，OpenStack 免费、开放的优势显现出来。VMware 高投入带来的功能，OpenStack 大部分可以免费提供给客户。

从 VMware 在功能方面的领先可以看出，VMware 还在继续研发除了 vMotion、高可用、容错以外其他的新功能去保护虚拟机；OpenStack 一方面跟随 VMware 的脚步，另一方面投入精力在支持更多硬件厂商解决方案上。

VMware 在将虚拟化扎根于企业级做了非常出色的工作，随着虚拟化使用越来越多，VMware 不断地围绕 Hypervisor 追加新的功能，VMware 的利润依然主要来自于此，基本上

还是虚拟化市场。因此，VMware 不是云，是基础架构自动化或者数据中心自动化，而不是基础设施服务。但是 VMware 针对企业级市场，提供了非常符合企业级需要的功能，如应用负载如何处理的问题，但不是弹性计算或存储，以及水平扩展的问题。因此，两者解决基础设施自动化的途径不但是技术上的差别，而且偏离很远。

基础架构云或者 OpenStack 并不是围绕 Hypervisor 来追加功能的产品，OpenStack 项目着眼于用逻辑或软件定义的方式来封装基础架构的资源，并可以被上层应用所使用，因此有下列这些概念：计算、网络、块存储、对象存储等。

3.3 虚拟化与虚拟化管理工具

我们先来简单回顾一下虚拟化的概念，以及一些具体的虚拟化方向、工具等。鉴于这些知识已经成熟并使用，因此在这里不展开讨论。

虚拟化通常是指计算元件在虚拟的基础上而不是真实的基础上运行。虚拟化技术可以扩大硬件的容量，简化软件的重新配置过程。CPU 的虚拟化技术可以令单 CPU 模拟多 CPU 并行，允许一个平台同时运行多个操作系统，并且应用程序都可以在相互独立的空间内运行而互不影响，从而显著提高计算机的工作效率。

虚拟化包括很多方面，在云平台中，直接相关的是服务器虚拟化、网络虚拟化和存储虚拟化。

3.3.1 服务器虚拟化

当今的 X86 服务器的体系结构使其在同一时间只能运行一个操作系统。大多数服务器的容量利用率不足 15%；不但效率非常低下，而且还导致了服务器数量剧增并增加了复杂性。通过抽象化物理硬件的操作系统和应用资源，服务器虚拟化摆脱了 X86 服务器传统的一对一体系结构模式，实现了更加经济高效、更敏捷、更简单的服务器环境。借助服务器虚拟化，多个操作系统能够以虚拟机方式运行在一个物理服务器上，每个虚拟机均可访问底层服务器计算资源。

服务器虚拟化释放出了当今 X86 服务器的强大潜能。在同一物理主机之上，可以运行多个不同的操作系统，用于不同的应用环境，极大地提高了单个服务器资源的使用效率。

3.3.2 网络虚拟化

网络虚拟化以软件方式完整再现了物理网络。虚拟网络不仅可以提供与物理网络相同的功能特性和保证，而且还具有虚拟化的硬件独立性和运营优势，包括快速调配、无中断部署、自动维护、支持旧版应用和新应用等。

网络虚拟化将逻辑网络连接设备和服务（逻辑端口、交换机、路由器、防火墙、负载均衡器和 VPN 等）提供给已连接的工作负载。应用在虚拟网络上的运行与在物理网络上完全相同。可以创建高度可扩展的网络结构，提供更高水平的运营效率和敏捷性、更快的调配、故障排除和克隆，同时进行监控并确保服务质量和安全保护。

3.3.3 存储虚拟化

存储虚拟化是软件定义存储的一部分，可在不增加存储硬件的前提下改善性能和空间效率。

它必须支持快速调配，以便这种高性能、高空间效率存储能够像如今的虚拟机一样快速启动并投入使用。它必须提供以虚拟机为中心的存储管理模式，这种模式对于在虚拟环境中承担更多存储管理任务的虚拟管理员来说必须非常直观。

3.3.4 虚拟化工具 VirtualBox

VirtualBox 是一款开源虚拟机软件。VirtualBox 是由德国 Innotek 公司开发，由 Sun Microsystems 公司出品的软件，使用 Qt 编写，在 Sun 被 Oracle 收购后正式更名为 Oracle VM VirtualBox。Innotek 以 GNU General Public License (GPL) 释出 VirtualBox，并提供二进制版本及 OSE 版本的代码。使用者可以在 VirtualBox 上安装并且执行 Solaris、Windows、DOS、Linux、OS/2 Warp、BSD 等系统作为客户端操作系统。

VirtualBox 的主要特点如下。

- 支持 64 位客户端操作系统，即使主机使用 32 位 CPU；
- 支持 SATA 硬盘 NCQ（原生命令队列）技术；
- 虚拟硬盘快照；
- 无缝视窗模式（须安装客户端驱动）；
- 能够在主机端与客户端共享剪贴簿（须安装客户端驱动）；

- 在主机端与客户端间建立分享文件夹（须安装客户端驱动）；
- 内建远端桌面服务器，实现单机多用户，支持 VMware VMDK 磁盘档及 Virtual PC VHD 磁盘档格式；
- 3D 虚拟化技术支持 OpenGL（2.1 版后支持）、Direct3D（3.0 版后支持）、WDDM（4.1 版后支持）；
- 最多虚拟 32 颗 CPU（3.0 版后支持）；
- 支持 VT-x 与 AMD-V 硬件虚拟化技术；
- 支持 iSCSI；
- 支持 USB 与 USB 2.0。

下面对其基本功能进行简单介绍。

使用 VirtualBox，在一个宿主机为 Windows 7 的操作系统上，创建一个 Ubuntu 64 位的虚拟机，该虚拟机可以通过 NAT（网络地址转换）方式访问外网。图 3-17 为在笔记本电脑运行 VirtualBox，并管理多个虚拟机的界面。

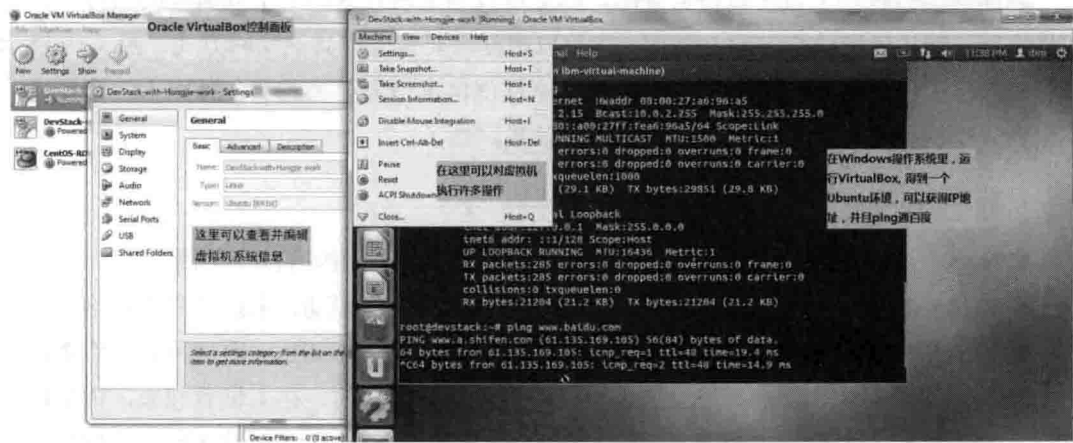


图 3-17 VirtualBox 运行虚拟机

虚拟机可以灵活地启动/停止，而且可以根据需要修改硬件设备和资源量。例如，可以随时增加一个 HDD 硬盘、一个网卡，或者增大 CPU。图 3-18 展示了可以对虚拟机进行的各种操作，包括设置虚拟机基本硬件信息和配置、启停虚拟机等。

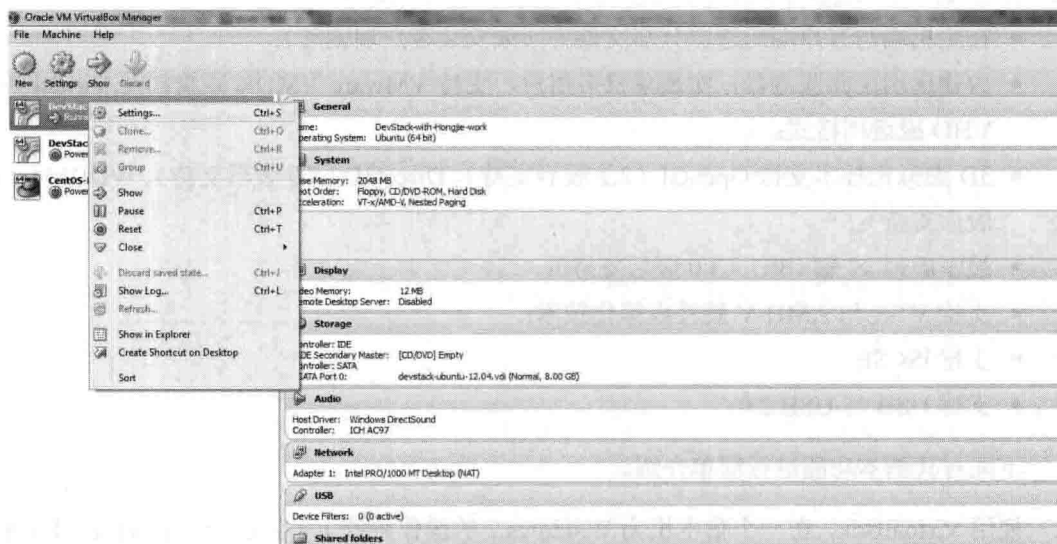


图 3-18 VirtualBox 管理虚拟机

创建的虚拟机马上就可以访问外网，因此可以马上从互联网上下载需要的软件，并在本机执行。默认使用的是 NAT，然而，也可以根据需要修改成其他联网方式。图 3-19 展示了可以对虚拟机网络连接进行的一些配置。这样就再也不需要去购买各种网卡、交换机，不用到轰鸣的机房里去寻找需要的机器，再插拔网线了。

目前 VirtualBox 为虚拟机提供的网络方式如下。

- **NAT:** 通过地址转换，虚拟机可以单向访问主机可以访问的外部网络。
- **Bridged Adapter:** 桥接的方式，利用一个创建的主机驱动，来过滤物理网络上的网络流量，因此也可以成为一个“网络过滤”驱动，VirtualBox 用这种方式将虚拟机的网络数据注入到虚拟机；当虚拟机使用这种方式通信，在主机看起来，虚拟机直接连接到了主机所在的物理网络。而主机也可以与虚拟机之间发送和接收数据。
- **Internal Network:** 虚拟机可以与在同一主机的其他虚拟机进行通信；如果不希望虚拟机流量在主机网络上出现并且被一些工具监听，虚拟机之间的流量需要更高的安全，可以选择这种连接方式。
- **Host-Only Adapter:** 可以认为是 Bridged 和 internal network 的结合，像桥接方式的网络，虚拟机可以相互连通，并且和主机通信，就像连接到同一个物理以太网交换机；如同内部网络，并不需要一个物理的网络接口，因此虚拟机不能和主机所在的外部网络通信，因为它们并没有连接到物理网络接口。

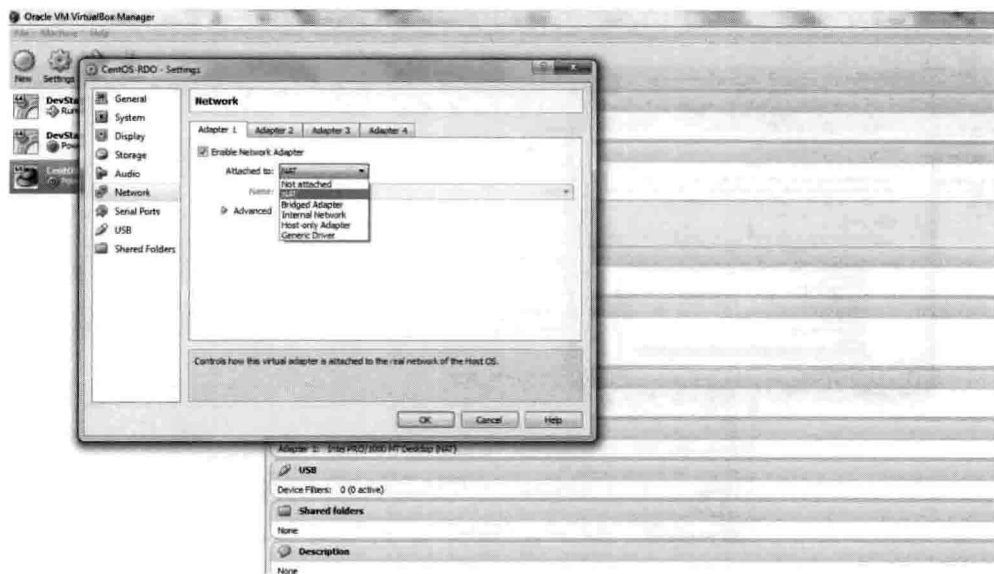


图 3-19 VirtualBox 配置虚拟机网络

图 3-20 展示了在 VirtualBox 里，配置全局的 NAT 和桥接网络的例子。

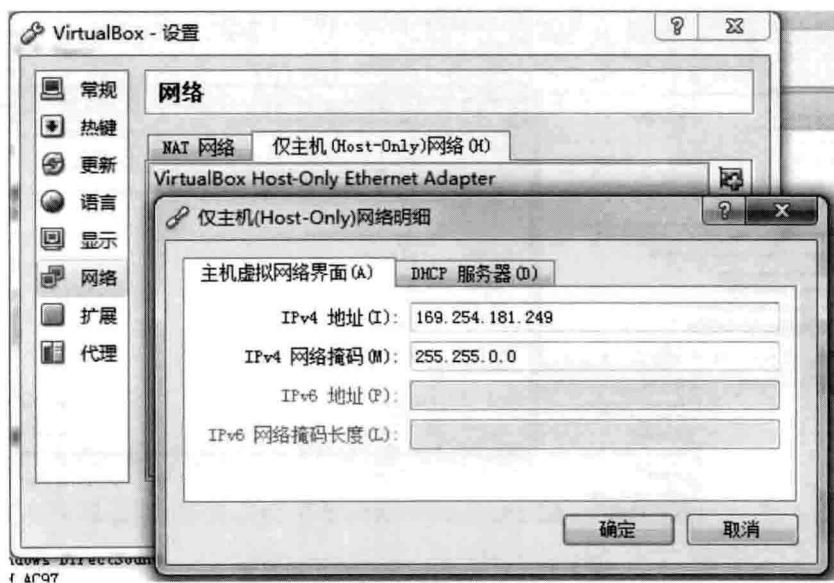


图 3-20 VirtualBox 配置全局的 NAT 和主机网络

图 3-21 为创建虚拟机的界面。创建新的虚拟机非常简单，选择操作系统，定义硬件参数，选择镜像文件。几步之后，我们就有了一个正在运行的虚拟机。

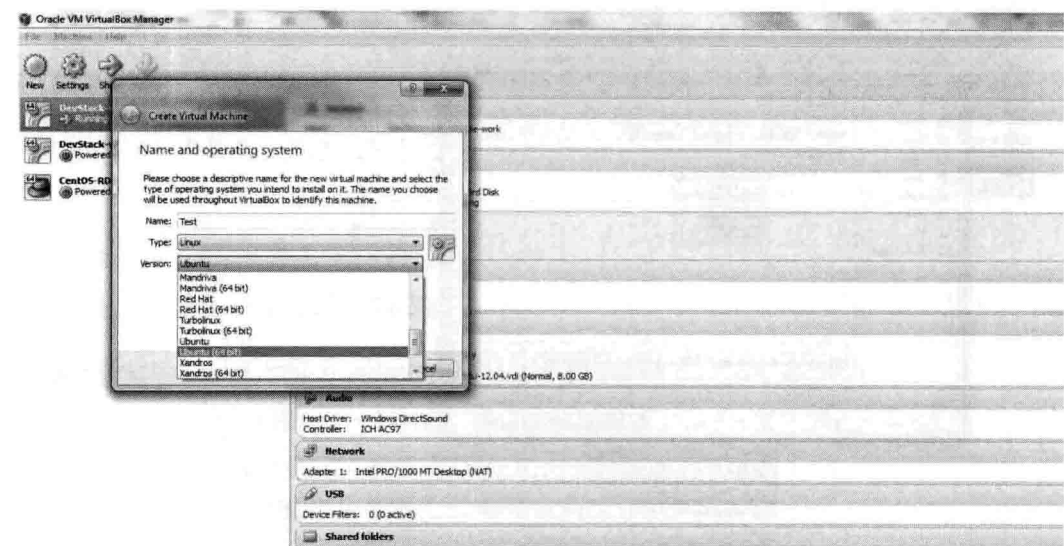


图 3-21 VirtualBox 创建虚拟机

在图 3-22 中选择本地一个虚拟机镜像文件，不需要安装操作系统，就马上拥有了一个定制过的虚拟机。虚拟机镜像是非常方便地发布定制化虚拟机的方式之一。

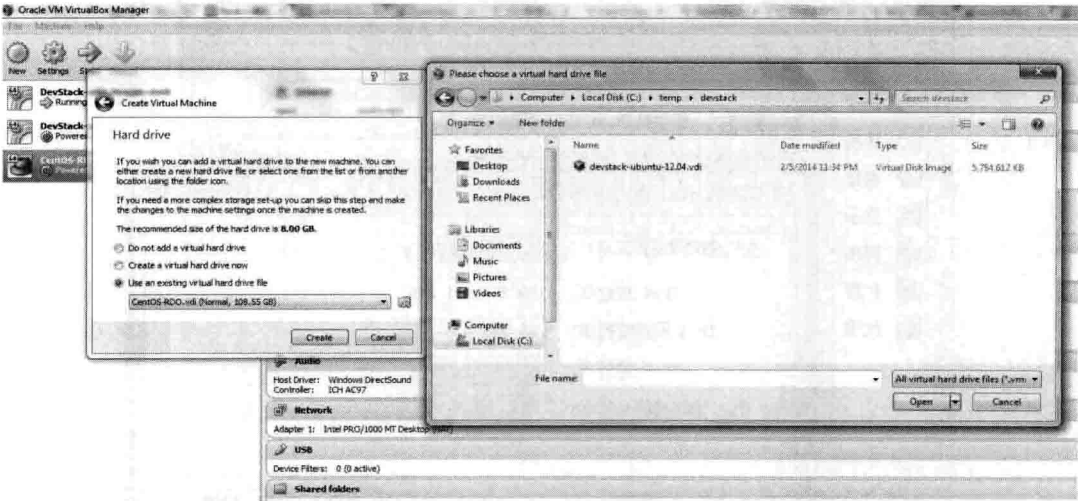


图 3-22 VirtualBox 载入虚拟机模板

3.3.5 虚拟化工具 Virt-Manager

Virt-Manager 是 Red Hat 共享的一个开源虚拟化管理软件，它是用 Python 编写的 GUI 程序。底层使用 Libvirt 对各类 Hypervisor 进行管理。图 3-23 为在 Linux 环境运行的 Virt-Manager 创建虚拟机的界面。

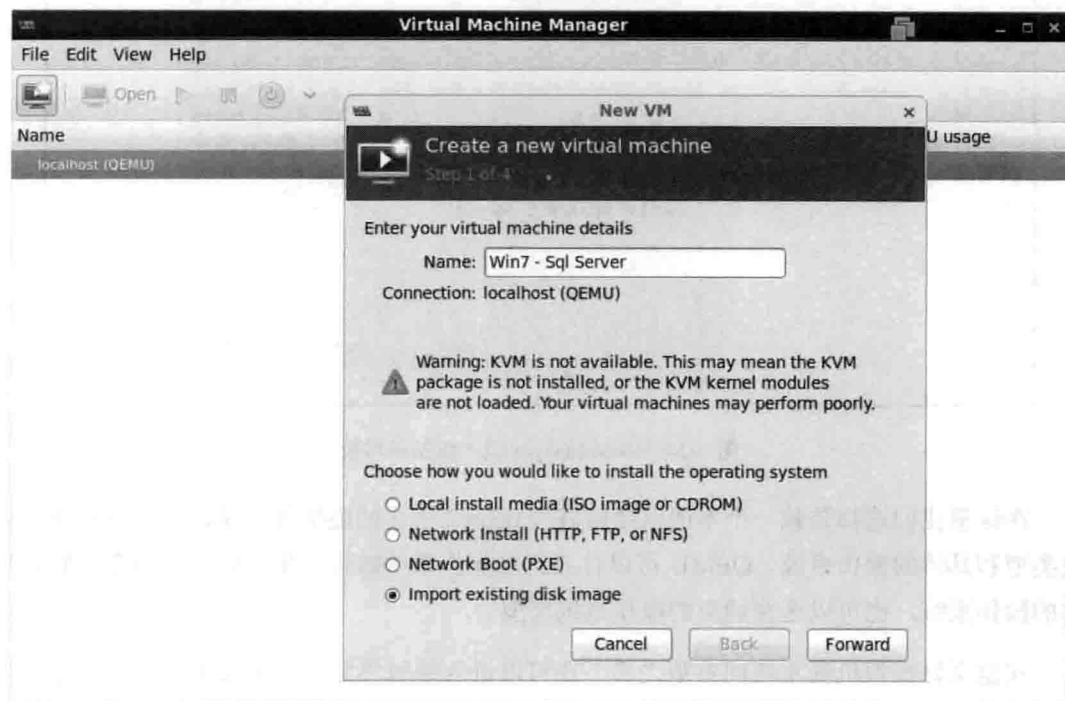


图 3-23 Virt-Manager 创建虚拟机

图 3-24 为在 Virt-Manager 里创建一个运行 Windows 7 操作系统的虚拟机，在给新虚拟机命名之后，我们可以选择从本地的 ISO/CDROM 来安装，也可以从远程的 HTTP、FTP 或 NFS 安装，或者通过 PXE（预启动执行环境）启动的方式从网络安装。

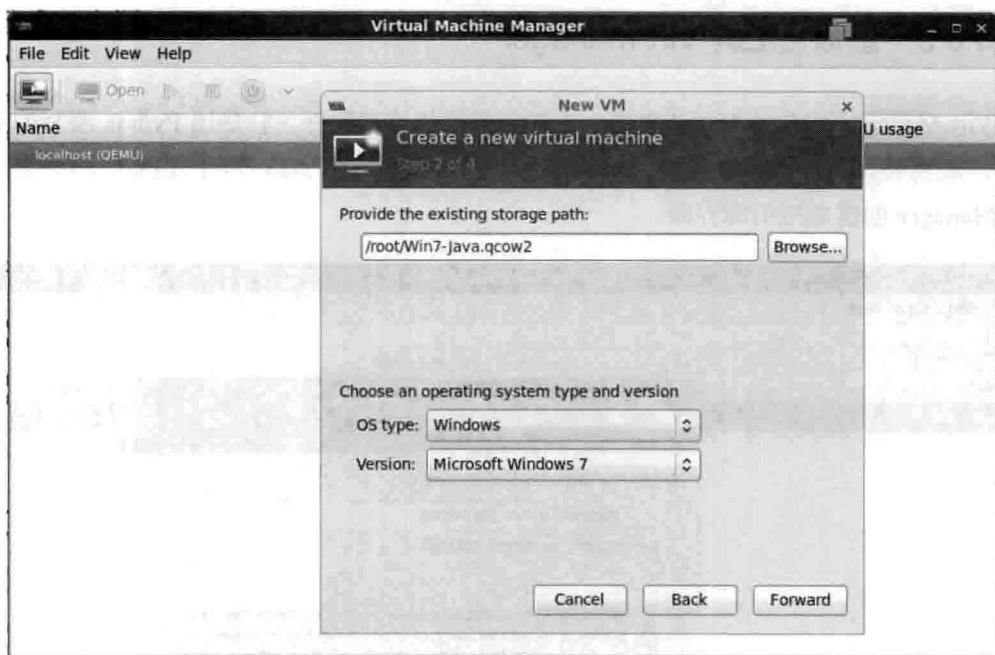


图 3-24 Virt-Manager 载入虚拟机模板

在这里我们选择装载一个本地已经做好的 qcow2 格式的虚拟机镜像，然后选择操作系统类型和具体的操作系统。QEMU 可以针对具体操作系统做进一步的优化。如果不知道具体的操作系统，也可以选择通用的操作系统类型。

在定义好虚拟机基本相同参数之后，还可以进入编辑界面，对虚拟机用到的具体硬件参数进行调整，或再次增加/删除需要的硬件。这些硬件包括 CPU、内存大小、启动顺序、网卡以及网络具体配置参数、鼠标、显示器、VNC（虚拟网络计算机）等。图 3-25 显示了为虚拟机配置网络的界面。

在选择运行之后，我们就有了一个在 Virt-Manager 里运行的 Windows 7 操作系统环境，受制于系统资源，虽然可能会运行得比较慢，但是它是一个“真正”的 Windows 操作系统，你可以在这里做一切想在 Windows 环境里做的事情。

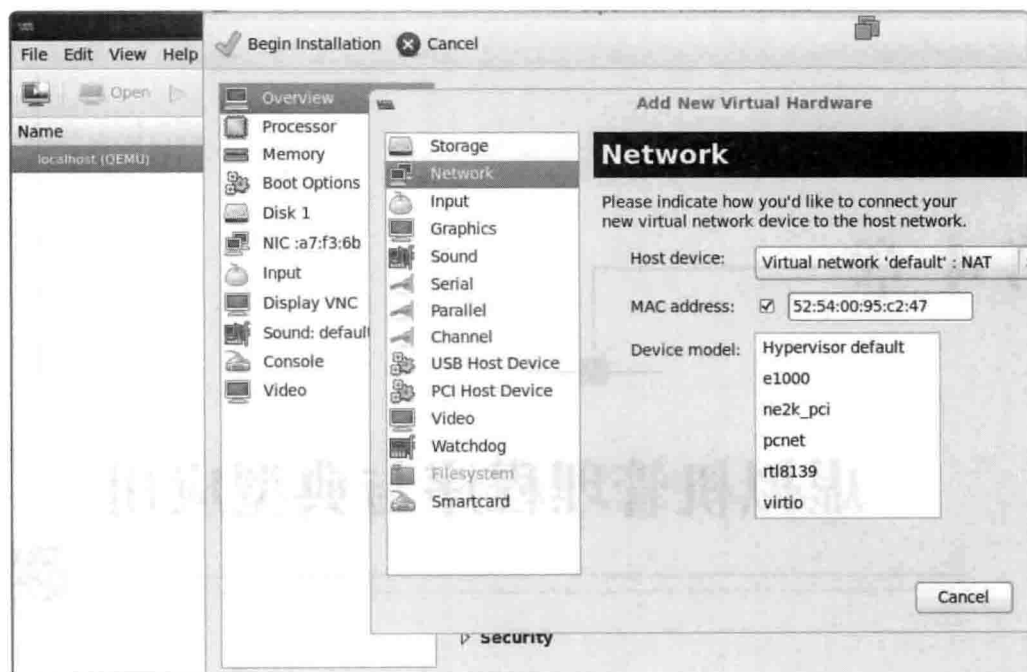



图 3-25 为虚拟机配置网络的界面

需要注意的是，尽管这些虚拟化工具都申请支持许多种虚拟机镜像文件类型，但经过实际使用，VirtualBox 支持最好的是 VDI 格式的镜像文件，而 Virt-Manager 支持的是 qcow2 和 raw 格式。尽管有一些工具可以在相互之间进行转换，但由于一些具体的不兼容问题，转换后的格式未必可以使用。所以在使用这些虚拟机镜像文件时，必须考虑到格式不兼容带来的影响，需要多做一些端到端的测试。

第 4 章



虚拟机管理程序与典型应用



根据 2014 年 4 月亚特兰大 OpenStack 全球峰会前的全球用户调研, 在 Hypervisor 中, 部署里基于 KVM 的云占到全部的 62%; 在研发测试环境中 Xen 占到 12%; 第三为 ESX, 占到 8%。但明显, 在准生产的评估环境和生产环境中, 第二则为 ESX 所代替。图 4-1 显示了每种 Hypervisor 的使用比例。

在这里, 有必要对 KVM 以及其与 OpenStack 的渊源进行一些介绍。

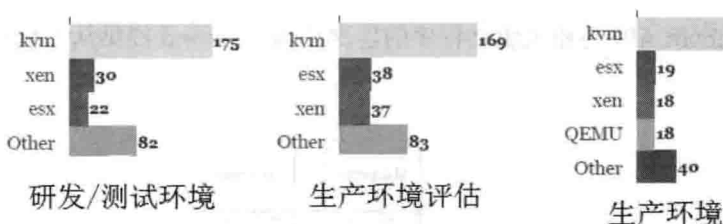


图 4-1 亚特兰大峰会对 Hypervisor 使用分布的调查

4.1 开放虚拟化技术 KVM

Hypervisor 是一种特殊的程序, 允许多个操作系统同时共享一台硬件主机。每一个操作系统都是在 Hypervisor 管理的虚拟机中运行的, 每个虚拟机被分配一部分主机的物理资源。Linux 下的 KVM (Kernel-based Virtual Machine), 是一个开源的系统虚拟化模块, 它使用 Linux 自身的调度器进行管理, 所以相对于 Xen, 其核心源码很少。

KVM 的设计原则如下。

- 全面利用英特尔虚拟化技术 (VT) 和 AMD 提供的所有硬件辅助虚拟化功能;
- 发掘硬件能力和虚拟化扩展, 同时保持 KVM 虚拟化开销减至最低;
- 完全集成到 Linux 操作系统内核, 自 Linux 2.6.20 之后集成在 Linux 的各个主要发行版本中。

KVM 以高性能、扩展性、高安全, 以及低成本面向用户。

KVM 用一个个进程来运行虚拟机。基于 KVM, 可以执行热迁移, 将一个运行的虚拟机从一台物理主机移到另外一台物理主机, 而虚拟机里的运行一点儿也不受影响; 可以保存当前虚拟机的运行状态到硬盘, 然后可以重新启动虚拟机, 这时虚拟机的运行状态和之前状态一样。

4.1.1 libvirt 介绍

Libvirt 是一个便于用户管理虚拟机和其他网络、存储等虚拟化功能的软件集合。它包括一个 API 库、一个守护进程 (libvirtd) 和一个命令行工具 (virsh)。它支持各种虚拟机监控程序 (Hypervisor)，包括 Xen、KVM 和 QEMU，以及用于其他操作系统的一些虚拟产品。它为虚拟机监控程序的常用功能提供一套通用的 API。

图 4-2 展示了 Libvirt API 与相关驱动程序的层次结构。libvirtd 提供从远程应用程序访问本地域的方式。

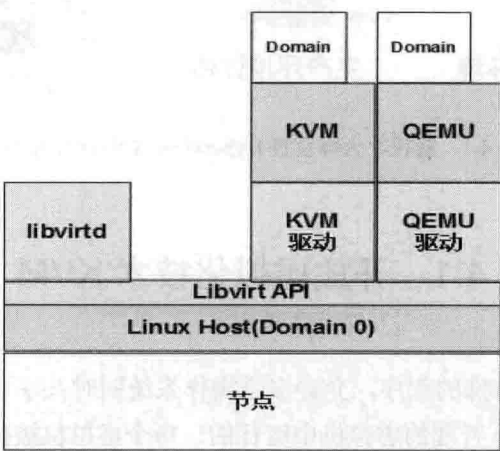


图 4-2 KVM 架构

Libvirt 的主要目标是为各种虚拟化工具提供一套方便、可靠的编程接口，用一种一致的方式管理各种不同的虚拟化提供方式。

Libvirt 的主要功能如下。

(1) 虚拟机管理：对虚拟机进行各种管理操作，包括启动、停止、暂停、保存、恢复和迁移等。支持多种设备类型的热插拔，如磁盘、网卡、内存和 CPU。

(2) 远程访问支持：只要一台主机上运行了 Libvirt daemon，所有的 Libvirt 功能就都可以访问和使用。支持多种网络远程传输，使用最简单的 SSH，不需要额外配置工作。比如，example.com 运行了 Libvirt，而且允许 SSH 访问，下面的命令行就可以在远程的主机上使用 virsh 命令行：

```
virsh -connect qemu+ssh://root@example.com/system
```

(3) **存储管理**: 任何运行了 Libvirt daemon 的主机都可以用来管理不同类型的存储, 如创建不同格式的文件映像 (qcow2、vmdk、raw 等)、挂接 NFS 共享、列出现有的 LVM 卷组、创建新的 LVM (Logical Volume Manager) 卷组和逻辑卷、对未处理过的磁盘设备分区、挂接 iSCSI 共享, 等等。因为 Libvirt 可以远程工作, 所有这些都可以通过远程主机来使用。

(4) **网络接口管理**: 任何运行了 Libvirt daemon 的主机都可以用来管理物理和逻辑的网络接口, 可以列出现有的接口、配置参数、桥接、VLAN (虚拟局域网) 和关联设备等, 也可以创建新的网络接口。

(5) **虚拟 NAT 和基于路由的网络**: 任何运行了 Libvirt daemon 的主机都可以用来管理和创建虚拟网络。Libvirt 虚拟网络使用防火墙规则作为路由器, 让虚拟机可以透明访问主机的网络。

4.1.2 域配置文件

每个 KVM 虚拟机都需要一个域配置文件来说明其硬件信息。域在这里来自于英文术语 domain, 是一个从 Xen 沿袭下来的概念。这里指一个虚拟机, 而不是指 Windows 平台的管理域的概念。域配置文件如下:

```
<domain type='kvm'>
  <name><虚拟机名字></name>
  <memory><虚拟机使用内存大小></memory>
  <vcpu><虚拟机使用 vCPU 个数></vcpu>
  <os>
    <type arch='x86_64' machine='rhel6.3.0'>hvm</type>
    <boot dev='hd' />
  </os>
  <features>
    <acpi />
    <apic />
    <pae />
  </features>
  <clock offset='utc' />
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type='file' device='disk'>
```

定义了虚拟机可使用的资源大小

定义了虚拟机使用的虚拟机镜像格式和路径。虚拟化软件根据此信息来启动具体的虚拟机; 虚拟机镜像格式中已经定义好“硬盘”, 在虚拟机运行之后, 便可发现多块如/dev/vda 等的硬盘

```

    <driver name='qemu' type='qcow2' />
    <source file='<虚拟机使用镜像文件路径>' />
    <target dev='hda' bus='ide' />
    <address type='drive' controller='0' bus='0' unit='0' />
</disk>
<interface type='bridge'>
    <mac address='<虚拟机网卡 MAC 地址>' />
    <source bridge='<虚拟机使用网桥名称>' />
    <model type='virtio' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03'
function='0x0' />
</interface>
<serial type='pty'>
    <target port='0' />
</serial>
<console type='pty'>
    <target type='serial' port='0' />
</console>
<video>
    <model type='cirrus' vram='9216' heads='1' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02'
function='0x0' />
</video>
<memballoon model='virtio'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05'
function='0x0' />
</memballoon>
</devices>
</domain>

```

定义了虚拟机网卡 MAC 地址，以及连接到什么交换设备。在这里连接的是一个虚拟网桥

4.1.3 使用 Libvirt 创建和管理 KVM 虚拟机

下面步骤说明了从安装 KVM，然后部署一个 KVM 虚拟机，通过命令启动/停止，并登录到创建的虚拟机的过程。无论是图形化界面，还是命令行，创建和管理 KVM 虚拟机都非常简单。

(1) 安装 KVM，并加载 KVM 内核，如下：

```

[root@target ~]# yum install -y qemu-kvm.x86_64 qemu-kvm-tools.x86_64
#安装 kvm 内核
[root@target ~]# yum install libvirt.x86_64 libvirt-cim.x86_64
libvirt-client.x86_64 libvirt-java.noarch libvirt-python.x86_64
#安装 virt 管理工具
[root@target ~]# modprobe kvm

```

#加载 kvm 内核

```
[root@target ~]# modprobe kvm-intel
#intel cpu 加载 kvm-intel 内核, 要支持全虚拟化, 必须开启: 要求
#cpu 支持, 通过 bios 可设置
[root@target ~]# modprobe kvm-amd                #amd cpu 加载 kvm-intel
[root@target ~]# modprobe -ls | grep kvm          #查看内核是否开启
kernel/arch/x86/kvm/kvm.ko
kernel/arch/x86/kvm/kvm-intel.ko
kernel/arch/x86/kvm/kvm-amd.ko
[root@target ~]# modprobe -ls | grep kvm-intel
kernel/arch/x86/kvm/kvm-intel.ko
```

(2) 通过命令行安装虚拟机, 如下:

```
[root@target ~]# yum install virt-viewer          #开启图形控制台, 安装虚拟机需要
[root@target ~]# virt-install \                   #安装选项可用 virt-install --help 查看
--name node4 \                                   #虚拟机名称
--ram=1024 \                                     #分配内存大小以 MB 为单位
--arch=x86_64 \                                  #模拟的 CPU 构架
--vcpus=1 \                                       #配置虚拟机的 vcpu 数目
--check-cpu \                                    #检查确定 vcpu 是否超过物理 CPU 数目, 如果超过则发出警告
--os-type=linux \                                #要安装的操作系统类型, 例如: 'linux'、'unix'、'windows'
--os-variant=rhel5 \                             #操作系统版本, 如 'Fedora6'、'rhel5'、'solaris10'、'win2k'
--disk path=/virhost/node7.img,device=disk,bus=virtio,size=20,sparse=true \
#虚拟机所用磁盘或镜像文件, size 以 G 为单位
--bridge=br0 \                                   #指定网络, 采用网桥
--noautoconsole \                                #不自动开启控制台
--pxe                                             #通过网络安装
```

(3) 通过命令行开关虚拟机, 如下:

```
[root@target ~]# virsh start node2               #开机
[root@target ~]# virsh shutdown node2            #关机
[root@target ~]# virsh destroy node2             #强制关闭电源
[root@target ~]# virsh list --all                #查看虚拟机状态

Id 名称      状态
-----
18 node2     running
- node1      关闭
- win8       关闭
```

(4) 远程管理虚拟机 (qemu+ssh 连接), 如下:

```
[root@target ~]# yum install virt-viewer
[root@target ~]# export DISPLAY=192.168.10.8:0.0
[root@target ~]# virt-viewer -c qemu:///system node2
#本地管理虚拟机, system: 获取 system 权限, 注意 qemu 后是三个/
[root@manager ~]# virt-viewer -c qemu+ssh://root@192.16.12.2/system node2 #
远程 Linux 通过 virt-viewer+ssh 管理虚拟机
```

```
Xlib: extension "RANDR" missing on display "192.168.12.2:0.0".
root@192.168.12.2's password:
root@192.168.12.2's password:
#然后弹出 virt-viewer 的管理界面
```

4.2 Linux 容器

在服务器上，最常用的应用架构通常为 Web 服务类型，包括前端的 Web 服务器，中间为应用服务器，后端为一到多个数据库。这类应用可以占到我们日常应用系统的绝大多数。只是每个模块采用的具体技术不同、规模不同等。在虚拟化和云平台上，可能多个 Web 服务器运行于同一个物理主机上，它们需要根据访问量做线性扩展，需要极高的响应速度，数据库需要很高的吞吐等。

在服务器上部署、运行这样一个系统，从开发到测试到上线，特别是反复的迭代，我们可能会遇到下面的问题和挑战。

- **易于安装：**任何一个应用，通常都不是一个孤立的系统，涉及前端、应用和数据库、系统库、运维脚本等。跨度从开发到测试到生产环境，涉及开发、测试和运维团队。特别是，有可能目标生产系统环境和开发测试环境有差异。安装在这个过程中尤为关键，中间任何环境出问题，都是需要花费很多人力和时间去检查和调试的。
- **隔离性：**如果我们已在同一个物理主机上部署了不同的网站，已有的网站只能在 Nginx 上或 Apache 上运行，此时我们就会有麻烦，即它们都监听 80 端口。同时运行两个网站是可以的，但需要调整配置（修改监听端口），设置反向代理等。库引用也会出现类似的冲突，如果有的老应用仍然依赖 PHP4，而新的则依赖于 PHP5 就会出问题，同时运行 PHP4 和 PHP5 会变得非常困难。运行在同一个服务器上的应用在文件系统级别和网络级别如果没有互相隔离，它们可能会互相冲突。
- **安全性：**对于外部用户访问的应用系统来说，安全总是非常重要的。任何一个软件/应用，总是会有安全缺陷。所以我们还要给它创建沙箱，至少在黑客入侵时不会影响其他运行中的应用。
- **资源限制：**如果我们的应用耗费很多 CPU 资源，就会导致其他应用无法做任何事情；如果它用尽了全部可用的内存呢？或者写日志导致磁盘阻塞呢？一个良好的系统需要能限制应用的可用资源，比如 CPU、内存和磁盘空间。另外，一个虚拟机可能会占用大量的网络带宽，导致其他服务无法访问。

- **升级、降级：**应用升级一般会覆盖现有文件。升级过程中会发生什么？如果升级失败，我们怎样能快速回退到先前的版本？系统如果要关闭，该如何操作？
- **快照、备份：**一旦所有的内容都设置好，就要给系统创建一个“快照”，以便能备份快照，甚至能移到另一个服务器上再次启动，或者复制到多个服务器上以备不时之需。
- **重复性：**系统出新版本之后，比较好的做法是先在测试环境中自动部署并测试，然后再发布到生产系统。通常会利用诸如 Chef、Puppet 等工具在服务器上自动安装软件包，等一切内容都测试并稳定可靠之后，再在生产系统上运行相同的部署脚本。通常绝大部分情况下都没有问题，但总是会有意外发生：在从测试环境到生产环境之间的时间跨度里，可能依赖的包有了更新，而新版本并没有充分测试因此并不兼容，从而导致生产环境的设置和测试环境不同。更严重的情况下，可能破坏生产系统。假如没有控制部署的每一个方面（例如自己的 APT (Advanced Package Tool) 或 YUM (Yellowdog Updater Modified) 仓库），持续在多个阶段（比如测试、预演、生产环境）重复搭建出完全相同的系统就很困难。
- **易于移除：**软件应该能轻松、干净地移除，不留痕迹，但部署应用通常要调整已有的配置文件、设置状态（MySQL 数据库的数据、日志），使得完全移除应用变得不那么容易。

解决这些问题目前最好的途径是使用虚拟机。如果在单独的虚拟机上运行独立的应用，则上述问题会变得非常容易解决。

- **易于安装：**如果包应用包装在虚拟机镜像中，则部署会变得非常容易，而且可以在开发测试阶段制作好，然后移到云平台。在传输过程中，应用和运行环境没有任何改变。在云平台上，只要单击一个按钮就能实例化应用，而启动只需要几分钟。
- **安全性：**由于应用在单独的虚拟机里，因此相互间完全隔离。如果某个应用遭受攻击，其余的基础设施并不会受到影响。
- **隔离性：**在不同的虚拟机上安装不同的应用，应用是完全独立的，除非攻击者攻破虚拟化管理程序，侵入主机。
- **重复性：**应用完全封装在单独的虚拟机镜像里，可以用各种方便的方式传输和搭建系统，然后创建访问点和访问空盒子，可以根据需要随意实例化多个实例，搭建负载均衡器，自动线性扩展。这个过程是完全可重现的。
- **资源限制：**每个虚拟机会分配特定的配额，包括 CPU 周期、可用内存和磁盘空间。没有调整配额的话就无法超额。

- **易于移除**：当不需要某个应用的时候，直接销毁虚拟机就可以，非常干净和方便，只需要单击一个按钮。
- **升级、降级**：升级和降级只需要虚拟机镜像和虚拟机实例的替换，新版本发布后，只需要创建新的虚拟机，部署新版本，然后让负载均衡器指向部署了新版本的虚拟机即可。
- **快照、备份**：单击一个按钮（或者调用一下 API），就能制作一个虚拟机或卷的快照，然后备份起来。

虚拟机已经得到了非常广泛的应用，人们基于 VMware、KVM 制作和部署了许多这样的应用。在虚拟机类的应用已经得到广泛的应用之后，人们还在继续寻找更好的方法继续提高性能，降低时间和其他开销。经过分析，虚拟机在下列几个方面还是比较昂贵的。

- **金钱**：虚拟机虽然带来很多好处，但仍然消耗很多系统资源，从硬件仿真或指令集转换到内存管理等。在一个大型环境中，比如“双十一”或情人节，为了应对网络访问的高峰，需要准备大量的计算资源，而此之后，许多资源又处于闲置。
- **时间**：虚拟机相关的操作大多数都很慢，启动要几分钟，捕捉快照要几分钟，创建镜像也需要几分钟。能否进一步降低时间的开销呢？

容器技术提供了非常轻量级的虚拟化，无须依赖硬件虚拟化所需要的指令级封装机制和其他复杂的全虚拟化实现，并且也非常容易提供进程和资源的隔离。容器技术可以非常有效地将单个操作系统管理的资源切分成多个隔离的组，来更好地平衡各个隔离的组之间对资源需求的冲突。和传统虚拟化的不同之处在于，无须指令级的仿真和内存页之间复杂的映射。容器可以运行和宿主机同样的 CPU 指令，无须任何指令集的转换，也不需要复杂的虚拟化、半虚拟化或者系统调用。

操作系统通过提供容器这种轻量级的技术，让一个应用程序感觉到还是运行于一个单独的机器中，同时却共享着许多底层的系统资源。例如，一些如 glibc 等需要的页缓存，就为所有的容器所共享，因为所有的容器使用同一个内核；并且取决于容器的配置，它们经常使用同样的其他系统库资源。共享的范围还可以扩展到目录里其他一些不需要写入的文件。

通过这些共享来节省资源开销，同时又能提供一定的隔离，意味着容器与其他那些虚拟化管理程序相比，对系统和资源的开销更低，也能提供更高的性能。

容器技术其实由来已久，如 Solaris Zones 和 BSD jails 都是一些非 Linux 操作系统上的容器的例子。这些容器技术也遗传到 Linux 操作系统，如 Linux-Vserver、OpenVZ 和 FreeVPS 等。尽管这些技术都比较成熟，但一直没有将其集成到主流的 Linux 内核里来。

Linux Resource Containers 项目（由 IBM 公司的 Daniel Lezcano 开发和维护）正在努力通过贡献于主流的 Linux 内核的方式来实现和支持容器，以推动在 Linux 操作系统上发展出一种比较成熟的容器实现。

Docker 是一个开源的、高度可移植的、自包含的自动化部署应用的引擎。它独立于底层的硬件、语言、框架、打包系统和主机。Docker 扩展了 LXC，包括一组高层的 API 来提供轻量级的虚拟化功能，提供了在一个单台节点中管理 LXC 容器的方式，让进程运行于隔离的环境中。它提供了一种可以安全和可重复地自动化部署软件的环境。一个标准的 Docker 环境包括软件组件、全部依赖的二进制文件、库文件、配置文件、脚本、Jars 文件、gems 文件和 tar 文件。Docker 可以运行于任何支持 cgroups 和 aufs 的 x86_64 Linux 内核中。但是运行了 OpenStack 计算服务的环境使得 Docker 可以提供更多的功能。

4.2.1 LXC

LXC（也称为 Linux 容器）是一种操作系统层的虚拟化技术，这种技术不同于其他硬件虚拟化技术的虚拟化管理程序，如 KVM、Xen 和 VMware。如果你有一些不支持硬件虚拟化的主机，LXC 可以提供比 QEMU 更高性能的应用环境。而且，如果你的虚拟机需要访问一些特殊的硬件，如 GPU，也可以很容易地通过 LXC 来办到。另外，LXC 不是一种适合于多租户环境的安全的虚拟化技术，特别地，容器技术可能影响到运行于同一主机上的其他容器的资源配额。基于这些原因，这种虚拟化技术需要在一些特定的用户场景下才推荐使用。

目前，有两个比较方便管理容器用户空间的工具：Libvirt 和 LXC。Libvirt 通过“lxc:///”像管理其他虚拟机驱动一样管理 LXC 虚拟机；另一个是与 Libvirt 完全独立的 LXC，它定义了一系列的命令，可以更灵活地管理和使用 LXC。

一、OpenStack 的支持

LXC 现在属于 Nova 项目，通过调用 Libvirt 来实现。当将 LXC 作为一种虚拟化管理程序使用时，一些 OpenStack 计算服务定义的功能还不具备，在使用前需要查看 OpenStack 社区的“虚拟化管理程序支持列表”来了解哪些功能还不具备，以便确定是否满足应用需求。

如果要使用 LXC，在运行 nova-compute 服务的节点的/etc/nova/nova.conf 文件中需要做如下改动：

```
compute_driver=libvirt.LibvirtDriver  
libvirt_type=lxc
```

在 Ubuntu 12.04 操作系统中，如果要使用 LXC，需要安装 `nova-compute-lxc` 软件包。

二、获取、制作和安装 LXC

LXC 项目包括一个加到 Linux 内核的补丁程序和一些运行于用户空间的工具组成。这些运行于用户空间的工具依赖于内核补丁的功能来提供一组精简的功能，用来操作和控制容器。为了能够使用 LXC，需要下载 Linux 内核代码，将 LXC 补丁加入进去，然后编译打包、安装并启动操作系统，在此之后下载 LXC 工具，编译、打包，就可以安装使用了。

1. 工具

下面是 LXC 需要的一些工具。

(1) Liblxc

下载并展开 Liblxc，然后在 liblxc 目录执行：

```
./configure --prefix=/  
make  
make install
```

同样地，还可以用 RPM (Red Hat Package Manager) 来安装它们。

(2) iproute2

为了管理容器的网络接口，需要版本 2.6.26 或之后的 iproute2 软件包。如果当前使用的 Linux 发行版没有这个包，则需要下载、配置、make 和安装这个包。

2. 配置网络

在容器需要的众多功能中，网络访问是最重要的部分之一，网桥目前是连接一个容器到网络最好的方法之一。为了使用 LXC，我们需要创建一个网桥，用它来连接容器的网口到真实的网络设备接口上去。

创建一个命名为 `br0` 的网桥包括：

```
brctl addbr br0  
brctl setfd br0 0
```

将前面已经在使用的网络接口 IP 地址赋予这个新创建的网桥接口，并启动它：

```
ifconfig br0 10.10.2.15 promisc up
```

将物理网络接口（在这里是 `eth0`）加到这个网桥中，并去除在此之前已经赋予的 IP 地址：

```
brctl addif br0 eth0
ifconfig eth0 0.0.0.0 up
```

最后，执行下列命令，确保默认路由可以把网络数据包发送到正确的网关：

```
route add -net default gw 10.10.2.2 br0
```

以后，当配置容器的时候，需要指定 `br0` 是连接外部网络的通路。

三、安装文件系统

除了网络，容器往往还需要它们自己的文件系统。下面给出两种为容器添加文件系统的方法。

- 构建一个定制的 Debian 容器；
- 运行一个 SSH 容器。

使用 `debootstrap` 命令构建一个定制的 Debian 容器非常简单：

```
debootstrap sid rootfs http://debian.osuosl.org/debian/
```

如果要构建大量的容器，首先将包下载到一个 `tarball` 中可以节省时间，例如：`debootstrap --make-tarball sid.packages.tgz sid http://debian.osuosl.org/debian/`

这将产生一个 `.tar` 文件，这个文件约 71MB（压缩了 52MB），而一个根目录约 200MB。然后开始在 `rootfs` 中构建根目录：

```
debootstrap --unpack-tarball sid.packages.tgz sidrootfs
```

（`debootstrap` 主页上有更多关于构建更小的或更适合的容器的信息。）这将生成一个宿主容器高度冗余的环境。

运行一个 SSH 容器可以让你显著地减少容器文件系统所消耗的磁盘空间。例如，这种方法可以为不同的容器运行多个绑定到端口 22 的 SSH 守护进程，仅仅消耗几 KB 的空间。从技术实现的角度，这是因为容器通过只读的方法来绑定需要使用的一些系统中比较关键的根目录，包括 `/bin/`、`/sbin/`、`/lib` 等。为了与已经存在的 Linux 系统共享 SSHD 软件包，需要使用到网络的命名空间，并创建基本的读写内容。

这些可以产生如此轻量级容器所使用到的技术，原来主要是用来生成 `chroot` 环境的。不同之处在于，它使用了只读绑定等来访问系统的一些关键目录，还使用到了命名空间来

增强 chroot 环境的隔离，使其变成一个非常有效的容器。

四、连接容器

取决于不同的配置，有以下几种方法可以用来连接所创建的容器。

- SSH。
- VNC (GUI)。
- VT: tty (text)。
- VT: X (GUI)。

如果不需要一个图形化界面来访问容器，则 SSH 是最好的选择。在这种情况下，一个简单的 SSH 连接就足够了。这种方法的优点在于，可依赖于 IP 地址等来创建任意数目的容器。

如果 SSH 连接等待很长时间后才看到密码输入提示，那往往可能是 Avahi Multicast DNS/Service Discovery 守护进程在 DNS 查找时超时的原因。

通过 VNC 连接容器可以为容器添加一个图形化的访问方式。在这种方式下，需要安装 vnc4server 来服务 VNC 客户端请求，可以安装 vnc4server 并从/etc/rc.local 启动来运行，如：

```
echo '/usr/bin/vnc4server :0 -geometry 1024x768 -depth 24' >> rootfs/etc/rc.local
```

这将在容器启动后，生成一个 24 位颜色、1024×768 分辨率的显示窗口。然后连接到这个容器变得非常容易：

```
vncviewer <ip>:<display>
```

五、运行 LXC 工具

LXC 使用 cgroup 文件系统来管理容器。在使用前，用户必须首先挂载这个文件系统到 LXC：

```
mount -t cgroup cgroup /cgroup
```

在使用 LXC 之前，首先必须挂载这个文件系统：

```
mount -tcgroupcgroup /cgroup
```

可以将 cgroup 文件系统挂载到任何地方。LXC 将使用/etc/mtab 中挂载的第一个 cgroup 文件系统。

1. 生成 LXC 容器

创建一个容器包括将一个容器的名字和一个配置文件关联起来。通过这个名字可以来管理一个具体的容器。下面的命令可以创建一个容器：

```
lxc-create -n name -f configfile
```

上述命令允许多个容器共用同一个配置文件。在配置文件里，可以指定许多容器的属性，包括主机名、网络、根文件系统和 `fstab` 等。在运行了 `lxc-sshd` 命令之后，一个 SSH 容器的配置如下所示：

```
lxc.utsname = my_ssh_container
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = br0
lxc.network.ipv4 = 10.0.2.16/24
lxc.network.name = eth0
lxc.mount = ./fstab
lxc.rootfs = ./rootfs
```

无论配置文件如何，每个由 LXC 工具启动的容器看到的都是自己系统里的进程、自己挂载的文件系统树、自己的进程间通信资源。

当容器开始运行，所有在配置文件里没有提到的资源，容器都认为是和主机共享的。这可以让管理员以非常紧凑的方式来说明主机与容器间的最关键的区别，让容器的配置具有最大的可移植性。

查询已创建容器的信息，对于管理容器非常关键。下面的命令用于显示一个指定容器的状态：

```
lxc-info -n name
```

下面的命令用来显示属于某个容器的进程：

```
lxc-ps
```

2. 启动

LXC 区别对待两种不同类型的容器：系统容器和应用容器。系统容器像一个虚拟机，和真正的虚拟机的不同之处在于，它们以降低隔离为代价，降低了对主机系统的开销，这是因为每个容器都使用同一个 Linux 内核。为了像一个真正的虚拟机，一个系统容器的开始点和一个 Linux 系统的开始点都一样，需要运行 `init` 程序：

```
lxc-start -n name init
```

和系统容器相反，一个应用容器仅仅创建足可以隔离单个应用的单独的命名空间。要运行一个应用容器，壳执行如下命令：

```
lxc-execute -n name cmd
```

3. 发送信号

如果要给容器里运行的所有进程发送信号，可执行如下命令：

```
lxc-kill -n name -s SIGNAL
```

4. 暂停

暂停一个容器，从概念上相当于发送 SIGSTOP 信号给容器里所有进程，但是发送危险的 spurious SIGSTOP 信号可能会混淆一些程序。所以 LXC 使用通过 cgroup 接口可用的 Linux process freezer。要暂停一个容器，可执行如下命令：

```
lxc-freeze -n name
```

5. 恢复

要恢复一个暂停的容器，可执行如下命令：

```
lxc-unfreeze -n name
```

6. 停止

停止一个容器包括停止所有容器里的进程，清理容器使用的资源。要停止一个容器，可执行如下命令：

```
lxc-stop -n name
```

7. 销毁

销毁一个容器包括删除其所使用的配置文件，和用 lxc-create 生成的关联到名字的元数据。要销毁一个容器，可执行如下命令：

```
lxc-destroy -n name
```

8. 其他

查看或调整容器的优先级的命令如下：

```
lxc-priority -n name
```

```
lxc-priority -n name -p priority
```

持续地观察一个容器的状态和优先级的变化的命令如下：

```
lxc-monitor -n name
```

按 Ctrl+C 组合键可以停止对容器的监控。

也可以等待容器进入一些用 “|” 分割的多个状态中的某个，命令如下：

```
lxc-wait -n name -s states
```

例如，等待进入除了 RUNNING 之外的其他状态：

```
lxc-wait -n name -s 'STOPPED|STARTING|STOPPING|ABORTING|FREEZING|FROZEN'
```

另外，LXC 使用 cgroup 文件系统来管理容器，因此可以通过 LXC 来对 cgroup 文件系统进行读取和一些操作。例如，管理 CPU 的使用情况也可以通过读取和调整容器的 CPU 配额来实现：

```
lxc-cgroup -n name cpu.shares
```

```
lxc-cgroup -n name cpu.shares howmany
```

4.2.2 Docker

Docker 是一种在 Linux 容器里运行应用的开源工具，一种轻量级的虚拟机。除了运行应用，Docker 还提供了一些工具，借助 Docker Index 或自己托管的 Docker 注册表对进行了容器处理的应用进行分发，从而简化复杂应用的部署过程。

Docker（主要用 Go 语言编写）是由公共 PaaS 提供商 dotCloud 发起的开源项目，它试图简化两种已有技术的使用。

- **LXC**：Linux 容器，允许独立进程在比普通 UNIX 进程更高的隔离级别上运行。使用的技术术语是集装箱化，一个容器里运行一个进程。容器支持的隔离级别如下。
 - **文件系统**：容器只能访问自己的沙箱文件系统（类似于 chroot），否则要专门挂载到容器的文件系统中才能访问。
 - **用户名字空间**：容器有自己的用户数据库（也就是容器的 root 不等于主机的 root 用户）。
 - **进程名字空间**：只有容器里的进程才是可见的（ps aux 的输出会非常简洁）。
 - **网络名字空间**：每个容器都有自己的虚拟网络设备和虚拟 IP（因此它可以绑定任意端口，不用占用主机端口）。
- **AUFS**：高级多层的统一文件系统，可用来创建联合、写时复制的文件系统。

Docker 可以安装在任何支持 AUFS 和内核版本大于等于 3.8 的 Linux 系统上。但从概

念上来说它并不依赖于这些技术，以后也可以和类似的技术一起运行，例如 Solaris 的 Zones 或 BSD jails，并将 ZFS (Zetabyte File System) 作为文件系统。不过目前只能选择 Linux 3.8+ 和 AUFS。

Docker 非常轻量。启动虚拟机通常是个大动作，需要占用大量内存；而启动 Docker 容器只耗费很少的 CPU 和内存，并且非常快，几乎和启动一个常规进程没什么区别。不仅运行容器快，构建镜像、捕获文件系统的快照也很快。

它可以运行在已经虚拟化过的环境中。也就是说，我们可以在 KVM 实例中或 VirtualBox 里运行 Docker。事实上，在 Mac 和 Windows 上使用 Docker 的首选方式是使用 Vagrant。Docker 容器能移植到任何可以运行 Docker 的操作系统上，无论是 Ubuntu 还是 CentOS，只要 Docker 运行着，你的容器就能运行。

Docker 比较好地解决了部署、操作等问题。

- **隔离性：**Docker 在文件系统和网络级别隔离了应用。从这个意义上讲很像在运行“真正的”虚拟机。
- **重复性：**用自己熟悉的方式准备系统（如登录并在所有软件里执行 `apt-get` 命令，或者使用 Dockerfile），然后把修改提交到镜像中。可以随意实例化若干个实例，或者把镜像传输到另一台机器，完全是同样的设置。
- **安全性：**Docker 容器比普通的进程隔离更为安全。Docker 团队也已经确定了一些安全问题，并正在着手解决。
- **资源约束：**Docker 现在能限制 CPU 的使用率和内存用量，但目前还不能直接限制磁盘的使用情况。
- **易于安装：**Docker 有一个 Docker Index，这个仓库存储了已有的 Docker 镜像，用一条命令就可以完成实例化。比如要使用 Clojure REPL 镜像，只要运行 `docker run -t -i images/clojure-repl` 命令就能自动获取并运行该镜像。
- **易于移除：**只需要销毁容器即可。
- **升级、降级：**和其他虚拟化管理程序一样，先启用新版本，然后把负载均衡器切换到新的端口。
- **快照、备份：**Docker 能提交镜像并给镜像打标签，并生成快照。

尽管 Docker 有助于系统的可靠部署，但它本身并不是个完全成熟的部署系统。它操作的是容器里运行的应用。至于哪个容器安装在哪个服务器上，以及如何启动它们，则超出了 Docker 的范围。

同样地, Docker 也不处理跨越多个容器(可能在多个物理服务器上,也可能在多个虚拟机上)运行的应用。要让容器互相通信,需要某些发现机制,来找出哪些 IP 和端口上的其他应用。这和跨越虚拟机的服务发现原理非常相似。Etc 等工具或者其他的服务发现机制都能用来解决这个问题。

Docker 可以从它的网站上获取并免费使用。它的交互式入门指南也很不错。Docker 将会被包装进 Fedora 项目里,成为未来 Fedora 发布版的一部分。基于 Fedora 和 Red Hat Enterprise Linux (RHEL) 的关系, Docker 最终会变成 RHEL 的一部分。Red Hat 将会帮助 Docker 移除对 AUFS (Advanced Multi Layer Unification Filesystem) 的依赖。目前 AUFS 由于它的 relabeling 闲置,还和 SELinux 不兼容,所以 Docker 还不能运行于 Fedora、RHEL 或 CentOS。

1. OpenStack 里的支持

在 OpenStack 的计算服务里, Docker 驱动是从 Havana 版本加入的一种虚拟化管理程序的驱动,可以在几台主机里管理数百个这样的容器。目前, Docker 项目优先解决和 OpenStack 的兼容性问题,不是去作为虚拟机的一个替代品,而是针对一些特定应用场景的补充。目前通过虚拟化管理程序来管理运行虚拟机已经很成熟,但容器还没有达到这样的程度,特别是使用 Libvirt/LXC 的方式仅仅是个开始。

为了在 OpenStack 计算里运行 Docker,需要在所有运行了 nova-compute 服务的主机的 /etc/nova/novacompute.conf 文件里配置下列配置项:

```
compute_driver=docker.DockerDriver
```

Glance 的 /etc/glance-api.conf 也需要配置来支持 Docker 容器格式:

```
container_formats = ami,ari,aki,bare,ovf,docker
docker_registry_default_port=5042 (IntOpt) Default TCP port to find the
docker-registry container
```

2. Docker 操作

如果已经安装了 Docker,要在 Ubuntu 容器中运行 bash,只要执行如下命令:

```
docker run -t -i ubuntu /bin/bash
```

根据“ubuntu”镜像的下载情况, Docker 会选择下载或者使用本地可用的副本,然后在 Ubuntu 容器里运行 /bin/bash。接着就能在容器里执行几乎所有典型的 Ubuntu 操作,比如安装一个新的软件包。下面来安装一个 hello 程序:

```
$ docker run -t -i ubuntu /bin/bash
root@78b96377e546:/# apt-get install hello
Reading package lists... Done
Building dependency tree... Done
The following NEW packages will be installed:
  hello
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 26.1 kB of archives.
After this operation, 102 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu/ precise/main hello amd64 2.7-2 [26.1 kB]
Fetched 26.1 kB in 0s (390 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package hello.
(Reading database ... 7545 files and directories currently installed.)
Unpacking hello (from .../archives/hello_2.7-2_amd64.deb) ...
Setting up hello (2.7-2) ...

root@78b96377e546:/# hello
Hello, world!
```

现在退出，然后再运行一次相同的 Docker 命令：

```
root@78b96377e546:/# exit
exit
$ docker run -t -i ubuntu /bin/bash
root@e5e9cd16021:/# hello
bash: hello: command not found
```

此时刚才运行的那个 `hello` 程序不见了。这是因为刚刚我们根据干净的 Ubuntu 镜像启动了一个新的容器。要继续先前那个 Docker，我们必须把它提交到仓库中。我们退出这个容器，查看下先前启动的容器 ID：

```
$ docker ps -a
```

ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
e5e9cd16021	ubuntu:12.04	/bin/bash	About a minute ago	Exit 127	
78b96377e546	ubuntu:12.04	/bin/bash	2 minutes ago	Exit 0	

`docker ps` 命令能列出当前运行的容器，`docker ps -a` 还会显示已经退出的容器。每个容器都有一个唯一的 ID，类似于 Git 提交的哈希值。该命令也列出了容器基于的镜像、运行的命令、创建时间、当前状态，以及容器暴露的端口和与主机端口之间的映射。

上面列出的是我们第二次启动的容器，它不包含“hello”；如果需要能重用一个容器，我们提交其到仓库里，再用它创建一个新的容器：

```
$ docker commit 78b96377e546 images/ubuntu
356e4d516681
```

```
$ docker run -t -i images/ubuntu /bin/bash
root@0d7898bbf8cd:/# hello
Hello, world!
```

上面例子中，用容器 ID 把容器提交到了仓库中。仓库类似于 Git 仓库，包含一或多个打了标签的镜像。如果没有指定标签名称，标签会被命名为“latest”。运行 `docker images` 命令可以查看本地安装的所有镜像。

Docker 提供了一些基础镜像（比如 Ubuntu 和 CentOS），用户也可以创建自己的镜像。用户仓库的命名模型和 Github 的类似：在 Docker 用户名后面跟一个斜线，然后再跟仓库名称。

创建 Docker 镜像更简洁的方式是使用 Dockerfile。

3. 使用 Dockerfile 构建镜像

Dockerfile 是个简单的文本文件，说明了如何从基础镜像构建镜像。下面的 Dockerfile 文件用来运行、安装一个 SSH 服务器：

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y openssh-server
RUN mkdir /var/run/sshd
RUN echo "root:123456" | chpasswd
EXPOSE 22
```

FROM 命令定义了基础镜像，基础镜像可以是官方的，也可以是新创建的 `images/ubuntu`。RUN 命令用来配置镜像。在这里，我们更新了 APT 包仓库，安装了 `openssh-server`，创建了一个目录，然后给 root 账户设置了一个简单的密码。EXPOSE 命令会向外暴露 22 端口（SSH 端口）。

4. 构建并实例化 Dockerfile

第一步是构建一个镜像。在包含 Dockerfile 的目录下运行如下命令：

```
$ docker build -t images/ssh
```

这步会创建一个 `images/ssh` 仓库，包含新的 SSH 镜像。如果创建成功，就能进行实例化了：

```
$ docker run -d images/ssh /usr/sbin/sshd -D
```

和前面的命令不一样，`-d` 表示会在后台运行容器，而不是运行 `bash`，所以我们用前台模式（用 `-D` 参数指定）运行了 `SSHD` 守护进程。

现在查看运行中的容器，看看命令做了些什么：

```
$ docker ps
ID                IMAGE                COMMAND              CREATED
STATUS           PORTS
23ee5acf5c91     zefhemel/ssh:latest  /usr/sbin/sshd -D    3 seconds ago
Up 2 seconds     49354->22
```

可以看到新的容器启动着。`PORTS` 头下为 `49354->22`：这是因为前面我们 `EXPOSE` 了 22 端口，这个端口现在映射到了主机里的一个端口 49154。让我们看看它能否运行：

```
$ ssh root@localhost -p 49354
The authenticity of host '[localhost]:49154 ([127.0.0.1]:49354)' can't be
established.
ECDSA key fingerprint is f3:cc:c1:0b:e9:e4:49:f2:98:9a:af:3b:30:59:77:35.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:49354' (ECDSA) to the list of known hosts.
root@localhost's password: <I typed in 'root' here>
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.8.0-27-generic x86_64)
```

```
* Documentation: https://help.ubuntu.com/
```

```
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
```

```
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

```
root@23ee5acf5c91:~#
```

登录成功，现在有了一个运行的 `SSH` 服务器，而且能登录它。要终止这个容器，可执行如下命令：

```
$ docker kill 23ee5acf5c91
```

容器的 22 端口映射到了 49354 端口，但这是完全随机的。要把它映射到特定端口，运行命令时需要传入 `-p` 参数：

```
docker run -p 2222:22 -d zefhemel/ssh /usr/sbin/sshd -D
```

现在，如果 2222 端口可用，则 `SSH` 端口就会映射到 2222 上。

我们在 Dockerfile 的结尾再添加一行内容，以便我们的镜像对用户更加友好：

```
CMD /usr/sbin/sshd -D
```

CMD 表示构建镜像时并不会运行命令，在实例化时才运行。所以不传递其他参数时就执行 `/usr/sbin/sshd -D`。

然后我们可以直接运行：

```
docker run -p 2222:22 -d images/ssh
```

得到的结果和前面一样。

要发布新创建的镜像，只要运行 `docker push` 即可：

```
docker push images/ssh
```

登录之后，镜像就可用了。

4.3 裸机

在许多云平台里，除了提供虚拟化和虚拟机之外，还提供传统的物理机的服务。虽然虚拟机已经得到非常广泛的使用，但还是无法满足企业里一些业务系统对性能的要求。在 OpenStack 里，可以将裸机（Baremetal）与其他部署了虚拟化管理程序的节点通过不同的计算池（Availability Zone）一起管理。

裸机驱动是 OpenStack 计算服务使用的一种驱动。在 OpenStack 框架里，它和其他的虚拟化管理程序（如 Libvirt、KVM、Xen 等）有着同样的角色，但不一样的是，它的硬件是没有被虚拟化的。在物理硬件和租户之间，没有虚拟化层。它通过 OpenStack API，还有插件式的驱动，来提供一些硬件管理的功能，包括主机镜像的部署（PXE）、电源控制（IPMI）等。有了这些，配置和管理物理硬件可以通过普通的云平台 API 和一些工具来完成。如 OpenStack 的 heat（编排）或者 salt-cloud。然而，因为裸机是非常特殊的，所以使用裸机驱动需要一些额外的环境配置。

为了使裸机驱动可以被正确地加载和工作，需要修改计算节点（即运行 nova-compute 服务的主机）的配置文件 `/etc/nova/nova.conf`，涉及下面一些配置项：

```
[default]
compute_driver=nova.virt.baremetal.driver.BareMetalDriver
firewall_driver = nova.virt.firewall.NoopFirewallDriver
scheduler_host_manager=nova.scheduler.baremetal_host_manager.
BaremetalHostManager
ram_allocation_ratio=1.0
reserved_host_memory_mb=0
```

这里的一些配置项是裸机驱动所特有的，另外还需要一些额外的步骤，如制作部署用的 ramdisk 等。其他许多配置参数，请参考 OpenStack 社区的 `bk-config-guide-Havanna.pdf` 文档。

4.4 LXC/Docker 与 KVM/Xen 的选择

虚拟化管理程序提供了更好的进程隔离，呈现出一个完整的系统；而 Docker/LXC 除了一些基本的隔离，并没有提供足够的虚拟化管理功能，缺乏很多安全机制。如果运行了 SELinux，情况会好很多，但仍然依赖于共享的内核。而虚拟化提供了更高层的资源隔离。另外，共享的内核也限制了一些需要的功能，严重地依赖于主机的内核。如果需要的是一个非 Linux 的系统，就无法提供。另外，容器目前也不支持热迁移。

在一些情景下，基于容器的方案无法提供足够的灵活性，例如：

基于容器的方案无法运行和主机内核不同的其他内核。这可能和客户希望运行的操作系统内核不相容。

基于容器的方案无法运行一个完全不同的操作系统。人们往往希望在云里运行 Windows 操作系统，或其他非 Linux 的操作系统，如 FreeBSD 等。

作为一个替代物，目前 OpenStack 社区对容器的驱动的支持还没有像其他虚拟化管理程序那么成熟，还缺少一些功能或者稳定性。

目前，Docker 仍然无法替代虚拟化管理程序。虚拟化管理程序定义了许多标准操作，在 KVM、RHEV 或 VMware 中支持得很好，但容器只支持较少的部分。

在最基本的使用之后，Docker 仍然看起来像围绕 Linux 命名空间的一些工具集。

4.5 OpenStack 与 Linux

如图 4-3 所示, 根据 2014 年 4 月亚特兰大 OpenStack 全球峰会前的全球用户调研, 在已经部署的 OpenStack 环境里, 使用的操作系统 Ubuntu 占 47%, CentOS 占 17%, RHEL 占 11%。



图 4-3 亚特兰大峰会前对 OpenStack 使用的操作系统的分布

部署 OpenStack 的客户中, 有些明确要求操作系统是 CentOS, 对于政府、通信行业, 因为客户已经购买 license, 则要求必须是 RHEL。但是, 一些韩国、澳大利亚的售前人员曾经气急败坏地说, 如果不支持 Ubuntu, 那么别想进入他们国家的任何高校或其他企业。

因此, 很明显, 操作系统的选择在一定程度上反映出地域特点。不同国家的客户倾向不同。但毫无疑问, 这三个操作系统是 OpenStack 部署中最主要的三种操作系统。部署案例增多, 意味着在这种操作系统中, 大量的问题会被发现和修正, 也就是越安全。

4.6 OpenStack 与 KVM

OpenStack 被某些热心支持者称为云时代的 Linux, 是公有云与私有云的开源操作系统, OpenStack 炙手可热, 它如同 Linux 一样, 旨在构建一个内核, 所有的软件厂商都围绕着它进行工作。

KVM 集成在 Linux 的各个主要发行版本中, 使用 Linux 自身的调度器进行管理。KVM 专注于成为最好的虚拟机监控器, 是使用 Linux 的企业的不二选择, 加上它还支持 Windows 平台, 所以也是异构环境的最佳选择。

KVM 是一个脱颖而出的开放虚拟化技术。它是由一个大型的、活跃的开放社区共同开发的，Red Hat、IBM、SUSE 等都是其成员。2011 年，IBM、Red Hat、英特尔与惠普等建立开放虚拟化联盟（OVA），帮助构建 KVM 生态系统，提升 KVM 采用率。如今，OVA 已经拥有超过 250 名成员公司，其中，IBM 有 60 多位程序员专门工作于 KVM 开源社区。

OpenStack 几乎支持所有的虚拟化管理程序，不论是开源的（Xen 与 KVM）还是厂商的（Hyper-V 与 VMware）。但在早期，OpenStack 是基于 KVM 开发的，KVM 常常成为默认的虚拟机管理程序。两者都使用相同的开放源理念与开发方法。

如今，多数企业用户在 IT 环境中使用了超过一种的虚拟化软件，有一半的用户选择将开源产品作为性价比更高的虚拟化替代方案。IDC 报道中指出，OpenStack 是 KVM 增长的一个巨大机会。OpenStack 是一个具有巨大的行业发展动力，并拥有一个充满活力的社区的云计算平台，有 95% 的 OpenStack 平台由 KVM 驱动。因此，随着 OpenStack 的增长，KVM 也会相应增长。

4.7 OpenStack 与 VDI

在部署了 OpenStack 环境之后，许多客户都表示了对桌面虚拟化（VDI）的兴趣。虽然桌面虚拟化并不是客户首要和紧急的需求，但是很多客户都想在 OpenStack 平台上尝试这个功能，寻找它对企业的业务结合点和更多的价值。

4.7.1 基于 OpenStack 的 VDI 典型架构

图 4-4 是一个基本的在 OpenStack 平台支持 VDI 的架构。其中，如桌面管理、用户管理、模板管理、连接管理等功能都需要进行相应扩展，才可以是一个典型而完整的 VDI 方案 and 平台。OpenStack 的架构很好地满足了虚拟桌面化的需求，而且默认地就带有 VNC 的支持，可以通过 tigerVNC 等客户端连接远程桌面，效果类似于 Windows 的 Remote Desktop。但其分辨率较差，功能也很弱。通常用于如虚拟机启动异常，可以远程查看虚拟机的启动情况，或进行一些简单的命令行操作。OpenStack 的 Cinder 服务可以为远程桌面的虚拟机提供卷存储服务，如果不使用卷存储服务，也可以将虚拟机运行于非持久化存储或者共享存储里，可以对虚拟机进行热迁移等操作，而远程桌面的使用不受影响。

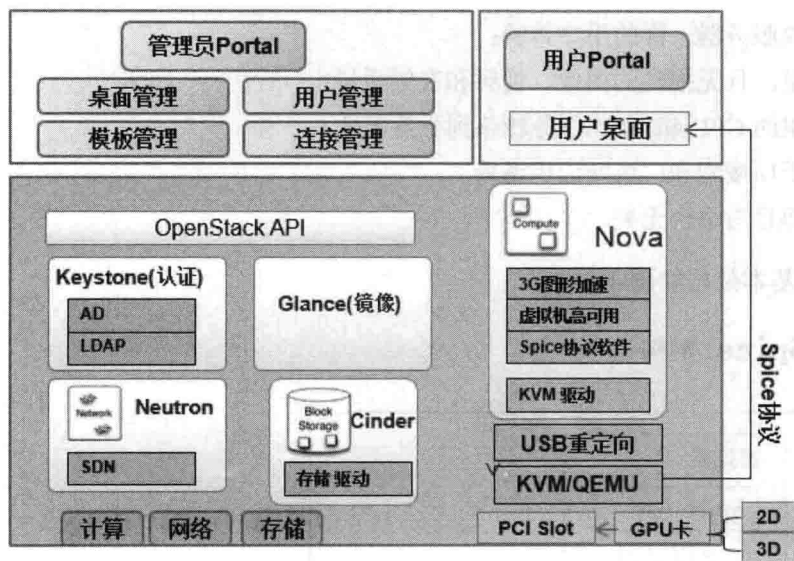


图 4-4 基于 OpenStack 的典型 VDI 架构

Glance 服务用来保存和管理远程桌面虚拟机的各种镜像。但 Glance 的功能相对于企业多样的需求往往难以满足，因此还需要额外的定制或开发。Keystone 提供了认证服务，但 OpenStack 默认的用户管理不但层级不够，而且缺少很多用户相关的元数据，因此一般也需要额外的开发来满足需求。

网络在远程桌面的云环境里要求相对较弱，扁平网络一般都可以满足需要。

Nova 里运行远程桌面的虚拟机，在客户端也需要相应的程序或浏览器插件，显示和操作远程桌面。基本的行为 OpenStack 已经满足。但在 OpenStack 环境里，往往用户会选择 Spice 协议和库来支持远程桌面应用类型。

4.7.2 Spice 协议

Spice 是一个开源协议，也是基于 OpenStack 桌面虚拟化的首选，用于客户端与服务器端通信，例如传输图形化对象、键盘和鼠标事件、光标信息、音频回放和录音，以及控制命令。Spice 具有如下特点。

- 一个开源的远程计算解决方案；
- 可以允许客户端远程访问服务器的显示器、键盘、鼠标和音箱；
- 适合于虚拟化环境，不特别依赖于虚拟机网络；

- 和本地服务器一样的用户体验;
- 高质量, 且无损耗的图像、视频和音频质量;
- 将密集的 CPU 和 GPU 任务转移到服务器端;
- 适用于局域网和广域网应用场景;
- 通用而且与平台无关。

Spice 的基本架构如图 4-5 所示。

Spice 架构

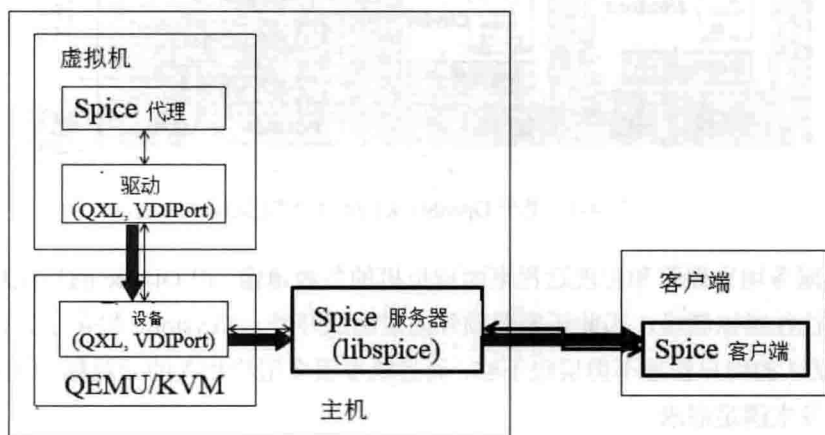


图 4-5 Spice 基本架构

架构图显示了基于 QEMU 并且运行了 libspice 库的基本组件, 以及虚拟机到客户端的图形操作命令的数据流, libspice 也可以运行于任何兼容的 VDI 应用中。图形操作命令开始于一个用户应用请求操作系统的图形引擎执行一个显示渲染操作, 图形引擎将命令传到 QXL 驱动, 它将这些命令翻译成操作系统命令, 然后发送到一个命令环的数据结构中。这个命令环运行于设备的内存里。然后 libspice 从环里取得命令, 把它们加到图形命令树里, 命令树包括一组命令, 执行这些命令将复现显示器显示的内容。这个树被 libspice 用来优化传输到客户端的命令传输, 可以消除那些被其他命令遮掩的显示/渲染命令。这个命令树也可以用于视频流的检测。libspice 还包括一个发送到客户端命令的队列, 来更新显示。当一个命令从队列里拿出, 准备发送到客户端时, 它就会被翻译成 Spice 协议的消息, 此时这个命令就从树里拿出, 并且从发送队列里删除。当 libspice 不再需要某个命令时, 它会被送到一个释放环里。驱动程序用这个环来释放命令占用的资源。当客户端收到一个图形命令, 则更新显示的内容。

1. Spice 代理

Spice 代理是一个软件，运行于虚拟机里，用于提升用户体验，执行一些面向虚拟机的任务。Spice 服务器和客户端用这个代理来执行一些任务，如配置客户端的显示设置，当使用客户端鼠标模式时，将鼠标位置和状态信息注入虚拟机等。通信是通过一个 VDI Port 的设备和一个虚拟机驱动来完成的。消息既可以是客户端产生的，如配置虚拟机显示设置；也可以是服务器端产生的，如鼠标运动；还包括代理本身，如配置应答等。

2. Spice 客户端

Spice 客户端是一个跨平台的用户交互的接口。它定义了一些非常通用的接口，例如 Platform 类定义了许多底层服务，如计时、光标操作等，同时在平行的目录里维护和平台相关的实现。

客户端和服务器端通信是通过一些通道来进行的。每个通道都是 TCP（传输控制协议）连接，可以是明文的或者 SSL（安全套接层）加密的，且每个通道用于特定数据的传输。在客户端，每个通道有一个独立的线程，这样可以在每个通道设置独立的 QoS（服务质量）参数，如不同线程的优先级。

这些通道具体如下。

- **RedClient (Main):** 是一个主要的通道，它拥有并控制所有其他的通道，包括创建、连接、断开连接等，并且处理控制需要的一些配置和迁移等。
- **Display:** 处理图形相关的命令、图像和视频数据流。
- **Inputs:** 键盘和鼠标输入。
- **Cursor:** 光标，指示鼠标位置、可见性和光标形状。
- **Playback:** 服务器端收到的客户端需要播放的音频。
- **Record:** 需要录制的客户端的音频。

3. Spice 服务器

Spice 服务器是由 libspice，一个虚拟设备接口（VDI）插件式的库来实现的。VDI 提供了一个标准的方式来通过软件提供虚拟设备。这可以使得其他软件模块通过 API 的方式与这些设备交互。Spice 服务器同时与远程客户端通过 Spice 协议通信，同时也与 VDI 上的应用通信。

为了远程显示的目的，这个服务器维护了一个命令的队列和一个树来管理当前对象的

依赖关系和隐藏。QXL 命令经过处理和翻译，然后用 Spice 协议发送到远程客户端。

Spice 总是试图将显示渲染的任务尽可能地发送到客户端，利用其硬件加速的功能。而在主机端通过软件 GPU 处理渲染，则作为最后的选择。服务器端保留那些可以合成当前显示内容的命令，只有当某个命令完全被另外的命令覆盖并且没有依赖关系时，才释放这个命令。

4. QXL 设备

Spice 服务器端支持 QXL VDI 接口，当 QEMU 用到 libspice 的时候，一个特殊的 QEMU QXL PCI 设备就被创建，用来提升远程显示的性能和增强图形处理能力。QXL 设备需要在虚拟机里安装 QXL 驱动来使用全部的功能。如果没有安装驱动，则使用标准的 VGA。这种模式也可以显示虚拟机从启动开始的界面。这个设备与驱动通过命令和光标环来交互，根据显示、光标和 I 端口的事件产生中断。

5. QXL 驱动

虚拟机里要使用与平台相关的驱动，来和 QXL 设备交互。Windows 虚拟机的驱动由一个显示驱动来和图形设备接口（GDI）交互，一个迷你端口驱动来处理内存映射、端口和中断。

4.7.3 开发桌面虚拟化应用的功能需求

开发一个桌面虚拟化平台，有许多事情要做。一个桌面虚拟化平台的设计与开发至少需要包括下列功能。

- 支持什么协议。
- 支持瘦客户端。
- 3D。
- 多媒体重定向（MMR）。
- 用户管理。
- 主镜像管理。
- 集成管理界面。
- 存储加速。
- Ovf 镜像文件打包。

- 网关 IP 虚拟化。
- 打印。
- USB 访问。
- 视频/音频。
- 多监视器。
- Windows AD (ActiveDirectories) 认证集成。
- LDAP (轻量级目录访问协议) 集成。
- 虚拟机高可用。
- 虚拟机负载均衡。
- 单点登录。

4.8 OpenStack 与 Hadoop

一般来说,虚拟化云平台与大数据在资源的使用上是有所不同的。虚拟化技术将一个池的资源进一步切分;而大数据则是用多个资源合起来进行超大型的计算。然而,通过移动计算,将大数据进行分片处理,以及数据尽可能本地化,通过并行 I/O 来降低网络通信,可以显著降低虚拟化对大数据计算带来的不利影响。图 4-6 说明了这种区分。

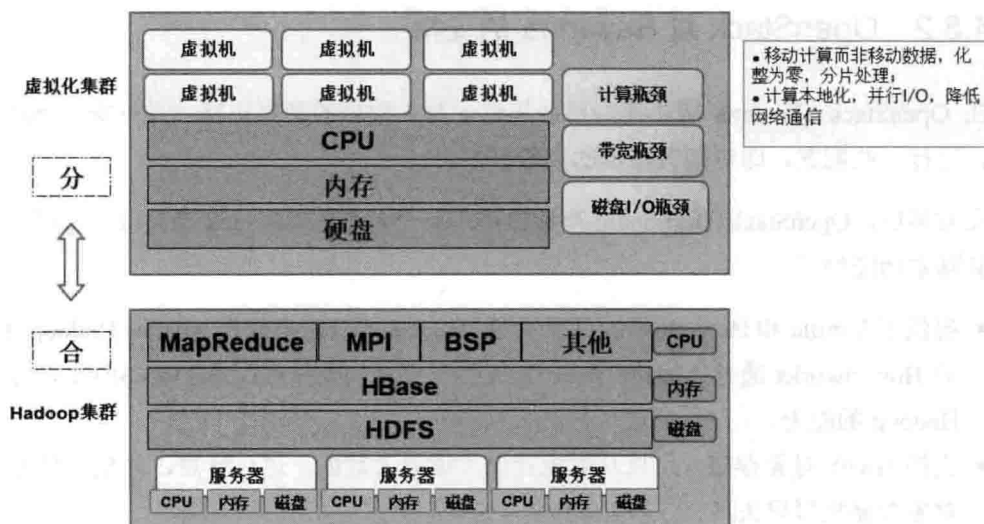


图 4-6 虚拟化与大数据

4.8.1 云平台/虚拟化对大数据计算的益处

- **提升 Hadoop 部署速度：**云平台 and 虚拟化可以提升 Hadoop 大数据集群的部署速度，易于维护。一份虚拟机镜像模板可以克隆分布到各类节点，分别启动，然后就可以得到一个大数据计算环境。
- **提供 Hadoop 高可用和容错能力：**通过云平台的各种高可用特性、热迁移等，可以非常容易地自动或人工来迁移虚拟机，或当主机出现故障时，自动地寻找其他可用服务器来运行虚拟机。
- **提升 Hadoop 环境资源利用率：**一个云平台里，虚拟机是随时可以创建和销毁的。所以可以白天运行如 ERP、各种企业其他应用，晚上则运行各种大数据计算。另外还可以同时运行多个大数据，提高集群的资源利用率。
- **Hadoop 云端多租户和安全隔离让 Hadoop 落地更安全：**云平台里，可以创建多个虚拟网络，在不同网络里运行不同应用。网络隔离使得计算更加安全。
- **集群易于维护和迁移：**同样利用云平台的热迁移、冷迁移，以及各种高可靠性等功能，非常容易实现大数据计算的维护和迁移。
- **使用异构集群实现高密度存储和计算：**云平台本身可以将异构的计算资源和存储资源整合成统一的资源池，但同时在调度上又可以做进一步的区分，显著提高了资源利用率，同时可以提供差异化的服务。

4.8.2 OpenStack 对 Savanna 的支持

在 OpenStack Havana 版本里，已经提供了对大数据的基本支持，通过安装 Savanna 的包，进行一些配置，即可拥有基本的大数据平台。

在安装后，OpenStack Dashboard 里就出现了一个单独的 Savanna 控制台，如图 4-7 所示。其基本功能如下。

- 提供了 Vanilla 和 Hortonworks 两种插件。Vanilla 可以用来操作 Apache Hadoop 集群，而 Hortonworks 通过 Apache Ambari 提供了更强的编排和配置 OpenStack 平台上的 Hadoop 的能力。
- 支持 Swift 对象存储。可以从对象存储里读取大数据，进行计算，并把结果返回到对象存储集群里去。

- 快速管理和部署 Hadoop 集群。集群可以方便地进行扩展，以提供更强的计算能力。所有操作只是在界面上的一些单击而已。



图 4-7 Savanna 管理界面

4.8.3 Savanna 的使用简介

1. 注册镜像

首先使用 `glance image-create` 命令上传镜像，镜像必须用已经做好的 Ubuntu 镜像，如图 4-8 所示。

在试验中可以采用一个社区配置好的 Ubuntu 镜像：`savanna-0.3-vanilla-1.2.1-ubuntu-13.04.qcow2`。

单击 Savanna 标签，选择 Image Registry，在页面右上角单击 Register Image，选择需要注册的镜像。

填写用户名: ubuntu。

添加标签: vanilla 1.2.1。

2. 创建 Node Group Templates

单击 Savanna 标签, 选择 Node Group Templates, 在页面右上角单击 Create Templates, 如图 4-9 所示。

在 Hadoop Cluster 中分 Master Node 和 Worker Node, 所以要创建两种模板。

(1) Master Node 要选中 Namenode 和 Jobtracker, 如果要使用 Swift 建立 EDP, 还需要选中 Oozie。

(2) Worker Node 要选中 Datanode 和 Tasktracker。

Register Image

Image *

ubuntu

User Name *

ubuntu

Description

Additional information here...

vanilla 1.2.1

Register tags required for the Plugin with specified Hadoop Version

Plugin	Version
vanilla	1.2.1

Add

Add custom tag

Image Registry tool:

Image Registry is used to provide additional information about images for Savanna

Specified username will be used by Savanna to apply configs and manage processes on instances.

Tags are used for filtering images suitable for each plugin and each hadoop version. To add required tags, select a plugin with Hadoop version and click "Add all" button.

You may also add any custom tag.

Unnecessary tags may be removed by clicking a cross near tag's name.

Cancel Done

图 4-8 注册镜像

Create Node Group Template

Configure Node Group TemplateHDFS Parameters *MapReduce Parameters *

Template Name *

savanna-master

Description

Additional information here...

OpenStack Flavor *

m1.small

Storage location *

Ephemeral Drive

Processes *

☒ namenode

☐ datanode

☐ secondarynamenode

☐ oozie

☐ tasktracker

☒ jobtracker

☐ hiveserver

This Node Group Template will be created for:
Plugin: vanilla
Hadoop version: 1.2.1

The Node Group Template object should specify processes that will be launched on each instance. Also an OpenStack flavor is required to boot VMs.

Savanna provides different storage location options. You may choose Ephemeral Drive or a Cinder Volume to be attached to instances.

When processes are selected, you may set **node** scoped Hadoop configurations on corresponding tabs.

Cancel

Create

图 4-9 创建节点集群模板

3. 创建 Cluster Templates

单击 Savanna 标签, 选择 Cluster Templates, 在页面右上角单击 Create Templates, 填写 Cluster Templates 的名字。在 Details 中选择 Cluster 中的 Node Group, 一般情况下是一个 Master 多个 Workers Node Group。

在如图 4-10 所示的界面中, 可以配置集群的一些基本信息, 包括模板名字, 使用到哪些节点类型; 而在如图 4-11 所示界面中, 可以配置使用到的具体各类节点数量。

Create Cluster Template

Details Node Groups General Parameters HDFS Parameters MapReduce Parameters

Template Name *
savanna-cluster

Description
Additional information here...

Use anti-affinity groups for:

- ☒ namenode
- ☒ datanode
- ☒ secondarynamenode
- ☒ oozie
- ☒ tasktracker
- ☒ jobtracker
- ☒ hiveserver

This Cluster Template will be created for:
Plugin: vanilla
Hadoop version: 1.2.1
 The Cluster Template object should specify Node Group Templates that will be used to build a Hadoop Cluster. You can add Node Groups using Node Group Templates on a "Node Groups" tab.
 You may set **cluster** scoped Hadoop configurations on corresponding tabs.
 The Cluster Template object may specify a list of processes in anti-affinity group. That means these processes may not be launched more than once on a single host.

Cancel Create

图 4-10 创建集群-详细信息

Create Cluster Template

Details Node Groups General Parameters HDFS Parameters MapReduce Parameters

savanna-worker

Group Name	Template	Count	
savanna-master	savanna-master	1	- + Remove
savanna-worker	savanna-worker	2	- + Remove

Cancel Create

图 4-11 创建集群-节点数量

4. 启动 Cluster（如图 4-12 所示）

单击 Savanna 标签，选择 Clusters，在页面右上角单击 Launch Cluster，输入 Cluster 的名字；选择步骤 3 中创建的模板，选择步骤 1 中注册的镜像，选择一个创建好的 keypair，选择创建好的 flat 网络，单击 Create。

Launch Cluster

Cluster Name *
cluster-savanna

Description
Additional information here..

Cluster Template
savanna-cluster

Base Image *
savanna

Keypair
savanna

Neutron Management Network *
flatnet

This Cluster will be started with:
Plugin: vanilla
Hadoop version: 1.2.1
Cluster can be launched using existing Cluster Templates.
The Cluster object should specify OpenStack Image to boot instances for Hadoop Cluster.
User has to choose a keypair to have access to clusters instances.

Cancel Create

图 4-12 启动集群

5. 执行作业登录以后切换到 Hadoop，用户就可以执行作业了。

比如要执行一个 wordcount 的作业，流程如下：

```
su hadoop
cd /home/hadoop
mkdir file
cd file
echo "hello savanna" > file1.txt
echo "hello hadoop" > file2.txt
cd /home/hadoop
```

在 HDFS 上创建输入文件夹。

```
hadoop fs -mkdir input
```

上传本地文件到集群：

```
hadoop fs -put ~/file/file*.txt input
```

运行 wordcount 程序即可。

第5章

OpenStack 架构与组件

5.1 OpenStack 项目与组件

OpenStack 由下列三种项目构成。

(1) 核心项目。这里包括我们所熟知的 Nova、Keystone、Glance、Neutron、Cinder、Ceilometer、Heat 等。许多用户已经基于它们搭建了自己的企业私有云。

(2) 孵化项目。孵化项目正在开发、完善和测试中，一旦它们成熟，经过社区的批准，则会转变成社区的核心项目，包括 Ironic、TripleO、Savanna 等。

(3) 其他相关项目。

表 5-1 至表 5-4 简要介绍了目前 OpenStack 的核心项目、存储相关项目、共享服务项目和高层服务项目，它们组成了目前 OpenStack 提供的各种功能。

表 5-1 OpenStack 核心项目

服 务	项 目 名 称	描 述
Dashboard	Horizon	提供了一个基于 Web 的自服务界面，来交互其他 OpenStack 服务，例如，创建一个虚拟机、分配网络、配置访问权限等
计算	Nova	管理 OpenStack 环境里虚拟机的生命周期，负责根据请求创建、调度、销毁虚拟机
网络	Neutron	实现网络作为服务，提供其他 OpenStack 网络服务。提供一套 API，用户可以定义网络并且分配网络等。在实现上，提供了一套可以嵌入的框架，其他网络硬件/软件提供商可以基于此框架提供自己产品的实现

表 5-2 OpenStack 存储相关项目

服 务	项 目 名 称	描 述
对象存储	Swift	通过 RESTful HTTP API（而不是传统通过 mount 文件卷，通过文件接口）来保存和访问任意非结构化数据，通过数据的自动复制和高度可扩展架构，达到数据的高度容错和可靠性
块存储	Cinder	提供了持久化的存储，来给运行的虚拟机提供块存储卷。它是一个可以通过 pluggable 驱动的架构，来提供创建和管理块存储服务的框架

表 5-3 OpenStack 共享服务项目

服 务	项 目 名 称	描 述
认证服务	Keystone	对所有 OpenStack 服务提供了认证和授权，对所有 OpenStack 服务提供了一个目录，可以通过该目录，来查询和访问部署的各种服务
镜像服务	Glance	可以保存和访问各种虚拟机镜像文件。OpenStack 计算服务通过该服务来在虚拟机创建时，安装各种不同用途的应用虚拟机
计费服务	Ceilometer	收集各种 OpenStack Cloud 资源的使用数据，包括 Nova、Cinder、Neutron、SDN 控制器、Swift、Glance，可以用来实现云平台的计费功能，同时也可以用于性能监控，以及其他各种统计的目的

表 5-4 OpenStack 高层服务项目

服 务	项 目 名 称	描 述
编排服务	heat	通过 OpenStack 本身的 REST API，或者亚马逊 AWS CloudFormation 兼容的 API，创建和维护 OpenStack 自有的模板格式，或者 AWS CloudFormation 模板格式，来编排各种复合的云应用（各种复杂网络、存储，还可以将各种环境和用户数据注入创建的虚拟机）

每个项目由一系列进程、命令行脚本、数据库表结构和其他脚本组成，其含义如表 5-5 所示。这些进程完全是分布式的，通过数据库和消息中间件来耦合到一起。进程包括接收和调解 API 调用的 WSGI 应用程序（nova-api、glance-api 等），进行编排任务的守护进程（nova-compute、nova-network、nova-scheduler）。

表 5-5 OpenStack 项目组成

类 型	描 述
Daemon（守护进程）	作为系统守护进程运行，在 Linux 平台，一个守护进程通常安装为一个服务来运行
CLI（命令行工具）	一组工具集，在命令行中执行并调用 OpenStack 各个服务的 API，将用户作业提交到 OpenStack 的各种服务进程

OpenStack 中还包含两个组件：消息队列服务和数据库。这两个组件通过消息传递和信息共享来异步编排复杂的任务。

OpenStack 对象存储是一个在具有内置冗余和容错的大容量系统中存储对象的系统。对象存储有各种应用模式，如备份或存档数据，存储图形或视频（流媒体数据传输到用户的浏览器），存储二级或三级静态数据，开发与数据存储集成的新应用程序，当难以预测存储容量时存储数据，搭建弹性和灵活的云存储 Web 应用程序。

5.2 IaaS 模型与 OpenStack 组件对应关系

OpenStack 是一组开源软件和技术，可用于搭建大规模可扩展云平台的操作系统，可以认为它是帮助你搭建一个类似于亚马逊 AWS 的 Infrastructure as a Service (IaaS) 的软件。

如果我们要构建自己的 IaaS 云环境并提供服务给用户，需要提供以下几个功能。

- 基本客户关系管理功能：允许需要云服务的用户注册云服务、查看资源使用情况以及使用的账单。
- 基本应用搭建功能：允许开发人员和运维人员创建和存储用于他们应用的定制镜像。
- 基本运行管理功能：允许开发人员和运维人员启动、监控、停止虚拟机实例。
- 允许云平台管理人员配置和管理云基础设施。

尽管仍然有很多很多功能需要提供，然而上述四项功能是最基本的。

在如图 5-1 所示的模型中，我们定义了与云交互的四种人员，他们需要与云平台交互，各自有自己对应的功能集合。

- 开发者：在云端虚拟机开发实现应用业务逻辑。
- 运维人员：对本租户的虚拟机和应用逻辑，及其运行的网络、存储进行定制和自动化。
- 应用使用者：使用上述人员提供的云平台环境。
- Cloud 管理员。

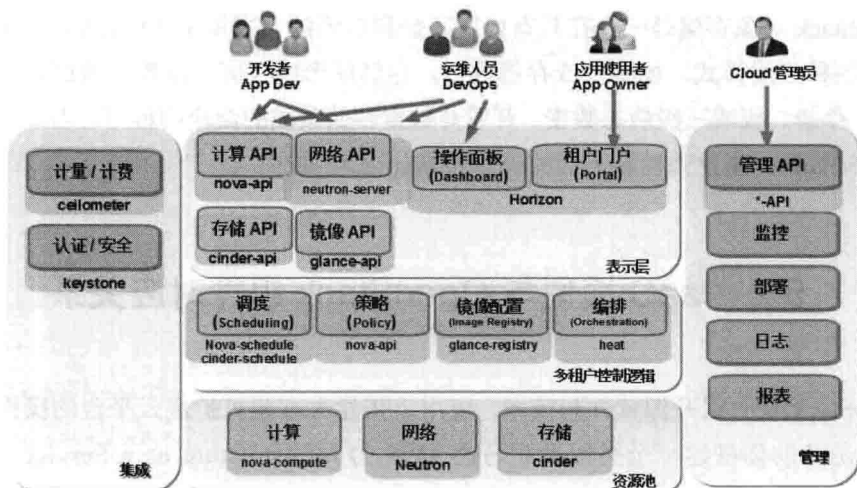


图 5-1 OpenStack 服务模型

在这里还有类似传统应用的三层架构（表现、逻辑、资源）和两个领域（集成和管理）。

- **表示层**：在这一层，各个组件与用户交互，接受用户输入，并呈现用户需要的信息。在这一层，非开发人员看到并操作的是一个 Web 图形界面，开发人员使用的是云服务的各种 API。在这一层，还提供了负载均衡、控制台代理、安全、命名服务等各项功能。
- **逻辑层**：提供了各种云服务的智能和控制功能逻辑。本层包括业务流程控制（用于复杂任务的工作流）、调度（确定把任务调度到什么资源去）、策略（包括配额、隔离等）、镜像（配置如虚拟机镜像的元数据）、编排（管理复杂虚拟机初始化，虚拟机集群的创建和生命周期等）。
- **资源层**：对于云计算，我们仍然需要物理的计算、网络 and 存储资源来进行计算任务，包括物理主机服务器、网络交换机、存储设备等。

在搭建云平台时，绝大多数服务提供商已有现成的客户身份管理系统、计费系统等。任何云架构将需要与这些系统集成。

对于云计算这样大型而复杂的环境，我们需要一个管理层，来管理和维护云平台的硬件和软件资源与运行环境。这包括提供给云管理员的、通过 API 调用的管理功能，以及各种监控工具等，而这些监控工具很可能也需要与已有系统集成。对于一个真实的云平台，还包括大量其他功能，如部署、配置、告警、备份、迁移等。

无论私有云，还是公有云，最核心的服务是提供虚拟机资源供我们使用。一个虚拟机，

包括计算、网络和存储资源，在 OpenStack 里，完全是软件定义的，由不同的项目和服务来提供。

因为目前 Havana 是最新的发布版本，在 Havana 版本中，核心项目已经有 Nova、Glance、Keystone、Cinder、Neutron、Ceilometer、Heat 等，图 5-2 是以虚拟机 (Virtual Machine, VM) 为中心的视图。可以看到虚拟机是由 Nova、Glance、Neutron、Cinder 等一起交互的结果。

- Nova 为我们需要的虚拟机提供了计算资源，包括 vCPU、内存，是我们的应用可以运行的环境。其作用就是我们传统所拥有的物理机，包括内存、CPU、各种外部设备、接口等，但不包括显示器、键盘和鼠标部分。
- Glance 为我们提供镜像服务。Glance 提供了一系列的功能，既可以让我们非常容易地拥有一个安装了操作系统的运行环境，也可以使其快速地变成任意定制化的开发/测试/运行环境，比如，安装了 Apache 或 Tomcat 的环境。
- Cinder 为我们提供了传统计算机的磁盘，或者卷。它是个裸设备，需要我们手工或自动地将其格式化成我们需要的文件系统，才可以使用。
- Neutron 既为我们的虚拟机提供了网络配置，也提供了我们可以从办公室或家里访问远程云中环境的网络通道。

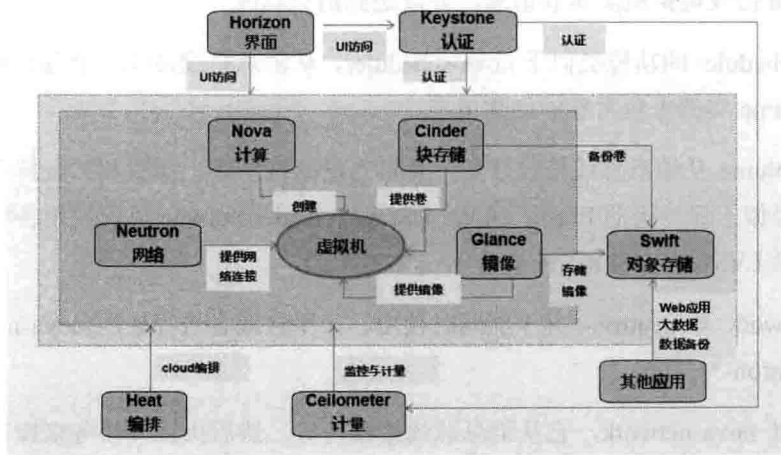


图 5-2 OpenStack 项目交互关系（以虚拟机为中心的视图）

图 5-2 展示了 OpenStack 项目之间的交互关系。云平台用户（开发者与运维人员，甚至包括其他 OpenStack 组件）通过 nova-api 等来与其他 OpenStack 服务交互，而这些 OpenStack 服务守护进程通过消息总线（动作）和数据库（信息）来执行 API 请求。

OpenStack Glance、Neutron、Cinder 都是完全独立、可以独立部署的服务，其服务入口分别为 glance-api、neutron-server、cinder-api。Nova 必须通过相应服务的 API 接口来调用和使用其服务。

nova-api 进程是 Nova 服务的中心点，它提供了所有 API 调用的服务端点，可发起绝大部分云服务 workflow 和编排任务，并且执行一些策略检查，如配额等。

nova-scheduler 进程是 OpenStack Nova 中实现最简单的代码：从消息总线接收一个虚拟机请求，决定在哪里创建并运行该虚拟机（更具体点，在哪个计算节点去运行）。然而，在实际环境中，这也许会变成最复杂的部分，因为它需要了解整个云环境的当前状态，并运用非常复杂的算法，来保证云平台的效率。

nova-scheduler 实现了一个插件式的架构，可以让用户选择或编写自己的算法用于调度。当前，已经包括多个调度算法，如简单、机会等。它将是一个未来版本 OpenStack 开发的热门领域。

nova-compute 进程是一个服务进程，用于创建和终止虚拟机实例。该进程的工作非常复杂，但最基本的是：从消息总线接收任务，然后执行一系列系统命令，如创建 KVM 虚拟机，并执行，最后在数据库中更新状态。

cinder-api 接收虚拟机实例卷请求，并发送到消息总线。

cinder-scheduler 的功能类似于 nova-scheduler，从消息总线接收一个卷请求，决定由哪个 cinder-volume 实例来负责卷的创建请求。

cinder-volume 从消息总线接收任务，负责为虚拟机创建、挂载和卸载持久化虚拟磁盘卷。其功能类似于亚马逊的 Elastic Block Storage。cinder-volume 可以使用各种存储后端提供的卷，包括 LVM、iSCSI 等。

nova-network 与 neutron-* 并不能同时使用，云平台或者选择使用 nova-network，或者选择使用 neutron-*。

如果使用 nova-network，它从消息总线接收任务，然后执行该任务来操作网络，如设置网桥接口、修改 iptables 规则等。

如果云平台部署的是 neutron-*，则 neutron-server 接收创建网络、子网、路由器等任务，而 openvswitch、neutron-*-agent 执行类似于 nova-network 的同样任务。但是 neutron-* 提供了多租户网络隔离以及许多高级网络管理功能。

消息总线（Message Queue）为所有这些守护服务进程提供了一个中心的消息机制。通过“主题”，发送者和消息的接收者相互交换任务和数据进行通信，协同完成各种云平台功能。消息总线将各个服务进程解耦，所有进程可以任意分布式部署而协同工作在一起。目前 RabbitMQ 是默认的消息总线实现，但理论上，任何支持 AMQP（异步消息队列协议）的实现都可以使用。

SQL 数据库保存了云平台绝大多数创建和运行时的状态，包括可用的虚拟机实例类型，正在使用的实例，可用的网络、项目等。理论上，OpenStack 可以使用任意支持 SQL-Alchemy 的数据库，当前大量使用的为 sqlite3（只用于测试和开发工作）、MySQL 和 PostgreSQL。

OpenStack Glance 包括三部分：glance-api、glance-registry 和镜像仓库。glance-api 就像 nova-api 一样，接收 API 请求，而镜像本身，则存储于镜像仓库中。glance-registry 保存和检索镜像的元数据信息。镜像仓库可以是文件系统、对象存储等，是可配置的。

最后，未在图中画出的是 Dashboard。Dashboard 提供了一个基本界面，管理员可以管理网络，上传镜像，分配虚拟磁盘卷，创建、启动和停止虚拟机等，用户也可以自己创建、启动和停止自己的虚拟机，查看云平台使用数据等。

管理和使用是走不同的网络通道的。管理必须要经由 *-api 转发过去，或者在控制台通过命令来完成；而用户使用时，则可以通过网络直接连接计算节点上的虚拟机。

图 5-3 是一个运行时的关系图。

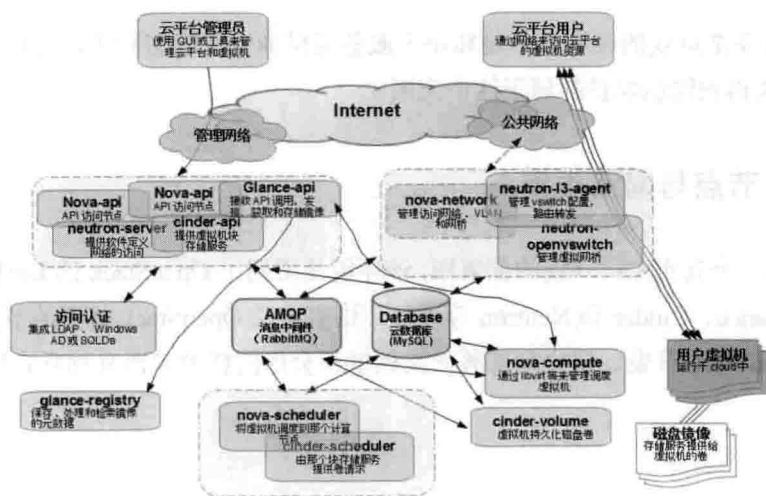


图 5-3 OpenStack 运行示例

5.2.1 OpenStack 功能待提高的方面

目前，从云平台角度，最大的差距在于日志，包括系统日志和用户访问日志等，并没有一个特别友好的方法管理和提取这些信息。而云平台管理员需要能够获取这类信息，以对云平台的安全和故障进行审计。

而计费是通过 OpenStack Ceilometer 来完成的，目前虽然已经实现了对虚拟机计算、网络、存储等的的数据收集，但依然缺失很多客户关心的指标数据。

Neutron 依然有很多问题，各个开发者忙于根据框架支持其自己的网络插件，但 neutron-server 及其核心组件依然有很多问题，基本的 Linux Bridge 和 OpenvSwitch 在使用过程中，也会出现许多意想不到的问题。如果在一定规模使用，又不关心 SDN 一类的新功能，则 nova-network 是一个非常稳定可靠的选择。

认证也是会有很大问题的项目。除非我们运行一个简单的环境，将用户、租户、权限信息存放到数据库中。在生产环境中，我们还是需要和我们现在的认证系统集成，包括 LDAP、Windows AD，甚至包括其他，或者开发自己的中间件。

门户 Portal 也是一个需要集成的部分。虽然 OpenStack 提供了 Dashboard（运行虚拟机，创建虚拟机），但它没有任何让客户签约云服务、查看账单或提交故障的功能。

监控也是一个非常弱的功能，目前严重依赖于第三方工具来监控服务状态、资源使用情况等。而计算节点到虚拟机的管理和监控，与客户希望的仍然有很大的差距。

策略也是非常重要的领域，但是和每个服务提供商业务紧密相关，包括从配额（已经支持）到 QoS 再到隐私保护都属于这个范围。

5.2.2 节点与网络类型

图 5-4 是一个真实生产环境的部署图，该环境使用到了 OpenStack 的 Dashboard、Nova、Keystone、Glance、Cinder 和 Neutron 等项目。这里涉及 OpenStack 部署的节点和网络，而所有关于 OpenStack 开发、研究和部署的文档里都会用同样的术语来称呼它们。

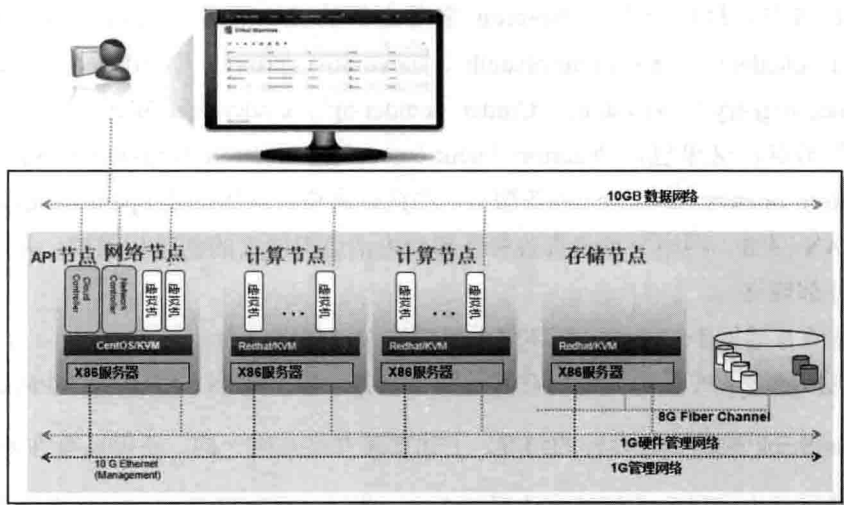


图 5-4 OpenStack 典型部署拓扑

1. OpenStack 部署的节点类型

图 5-5 中涉及的节点类型如下。

- **控制 (API) 节点**：安装并运行各种 OpenStack 控制服务进程，具体如下。
 - 提供了一个用户访问入口和环境中所有其他服务来访问的 API 服务。
 - 可以使用 Pacemaker 和 HAProxy 来以高可用的方式运行多个服务，使用 virtual IP 提供负载均衡调度。
 - 通过 MySQL 和 QPID 来把各种服务关联到一起，提供一个高可用的基础设施。
- **计算节点**：运行 KVM Hypervisor，包括 Nova 的 nova-compute、Neutron (openvswitch、neutron-openvswitch-agent) 等服务；该节点将创建并运行用户虚拟机，同时为虚拟机分配网络。
 - 运行最少而必须的一些服务，来辅助虚拟机的创建和运行。
 - 可以使用本机存储空间来给虚拟机提供非持久性存储，但虚拟机此时不能迁移，或者当节点失效时不能恢复。
- **存储节点**：运行 cinder-volume 服务，安装有存储的驱动程序，为计算节点的虚拟机提供持久化卷服务。存储节点存储环境需要的所有数据，包括磁盘镜像、虚拟机持久性卷。存储节点可以使用如 GlusterFS 来提供高可靠性和可扩展性。

在图 5-4 中，控制节点如下。

- API 节点，这里指去除 Neutron 服务之外的控制服务，包括 Nova (nova-api、nova-scheduler、nova-consoleauth、nova-novncproxy)、Glance (glance-api、glance-registry)、Keystone、Cinder (cinder-api、cinder-scheduler) 等。
- 网络节点：这里包括 Neutron (neutron-server、neutron-dhcp-agent、openvswitch、neutron-openvswitch-agent) 等服务。在此部署中，使用的是 openvswitch，二层则是 VLAN 网络。网络节点负责各种外部和内部虚拟网络的创建和维护，并连接虚拟机到外部网络。
 - 是所有运行于 OpenStack 环境虚拟机的入口和出口点。
 - 包括网络 API 服务之外所有其他网络服务，如 FWaaS、LBaaS、VPNaaS。

从 OpenStack 松耦合性设计的角度，上述部署并不是唯一的，还可以有许多其他变化。

- 存储节点也可以合并到某个计算节点去。此时，因为 cinder-volume 服务是管理存储设备，所以在当前的部署中，能且只能有一个服务进程运行。
- nova-compute 与 Neutron (openvswitch、neutron-openvswitch-agent) 服务必须在一起运行，否则虚拟机将无法得到网络，使得无法访问。
- 控制节点。
 - API 节点与网络节点可以合二为一，这样并不影响任何功能，唯一需要考虑的是性能和可维护性。
 - 前面没有提到 AMQP (高级消息队列协议) 和数据库，它们作为单独的服务进程，可以运行于 API 节点或网络节点之一，这样不影响任何功能。
 - 同样，Keystone、Glance、Cinder 服务，也不是必须在 API 节点运行，也可以在网络节点运行，或在一个新的控制节点运行，或可以单独创建认证节点来运行 Keystone、镜像节点以运行 Glance 服务。
 - 然而，cinder-api 与 cinder-scheduler 通常需要在同一节点运行，共用同一个配置文件；同样，nova-api、nova-scheduler、nova-novncproxy 等服务也要在同一节点运行，共用同一配置文件。

OpenStack 松耦合和弹性的设计，可以使它适应各种业务需要的部署场景，而技术不变。

2. OpenStack 部署的网络类型

表 5-6 列举了在一个典型的 OpenStack 部署中涉及的网络。

表 5-6 OpenStack 典型网络类型

网 络 类 型	说 明
管理网络	提供 OpenStack 组件间的内部通信，IP 地址通常只能在数据中心内部访问。用单独的网络，确保系统管理和监控访问于虚拟机网络分离，避免来自于用户虚拟机的监听和攻击，确保云平台的安全性
数据网络	在云环境中，供虚拟机之间相互通信，当虚拟机跨越物理主机时，由于物理机的网卡被设置为 trunk 模式，所以虚拟机相互之间的二层数据包通过物理交换机后直接可到，相互就像连接到同一个交换机上
外部网络	在一些部署场景里，提供了虚拟机访问 Internet。任何 Internet 上的用户都可以到达这个网络上的 IP 地址
API 网络	用来提供所有 OpenStack API 给租户。在这个网络上的 IP 地址应该是 Internet 上所有用户可达的。这个网络通常和外部网络是同一个，因为通常可以只分配外部网络的子网供其使用
存储网络	在很多部署环境中，搭建单独的二层网络，供存储设备与 cinder-volume 之间使用（这里指基于 IP 的协议）；也可以用于大型分布式/共享文件系统间
硬件管理网络	在大型数据中心，通常建立单独的硬件管理网络，用于管理员远程配置 BIOS，远程启动或关闭硬件服务器，查询系统信息，而不用到机房的控制台操作

OpenStack 的网络也有许多变化方案。

- 上述全部网络可以合并成一个网络，OpenStack 依然可以支持；这种环境通常用于 OpenStack 环境的原型验证（PoC），或开发测试。
- 通常，管理网络和硬件管理网络可以合二为一。两个网络从访问的角度，通常只局限于数据中心内部。
- 外部网络和 API 网络可以合二为一，通常这两个网络都是为外部用户服务和访问的。
- 管理网络可以和存储网络合二为一，管理网络通常网络流量很低。
- 外部网络、API 网络和数据网络可以合并，减少多个路由器等设备，降低网络的复杂度。

5.3 消息总线 and 数据库

Nova 使用的是一种“无共享，基于消息”的架构，组件间通过消息队列进行通信和信息共享来完成各种复杂任务的编排。只要支持 AMQP 协议的任何消息队列服务器（Message Queue Server）都可以作为 Nova 组件间通信的管道，当前官方推荐用 RabbitMQ。另外，为了提高用户体验，Nova 使用“回调”（call-back）机制发送消息。图 5-5 说明了它们之间的关系。

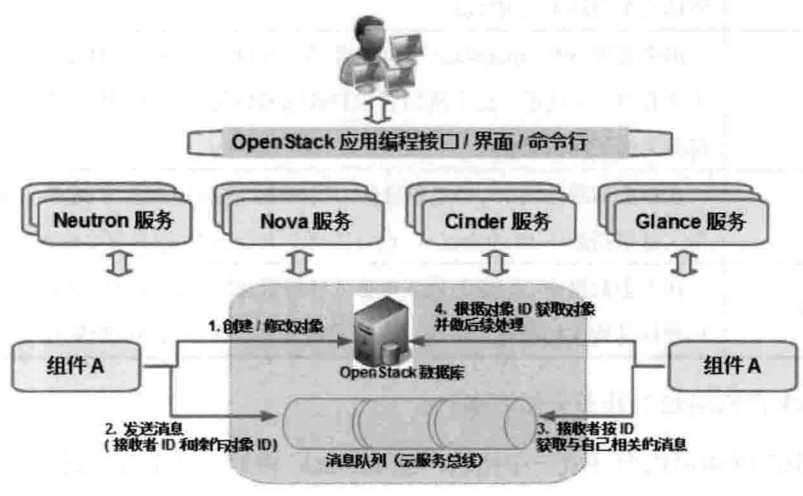


图 5-5 OpenStack 共享组件

AMQP 是一个异步的面向消息中间件的开放应用层协议。它是基于队列的，对消息提供路由（包括点对点和发布者-订阅者等方式）及提供一套消息安全可靠传递的机制。

AMQP 的基本数据单元是“数据帧”。目前有 9 种 AMQP 帧被定义，包括发起、控制和结束消息传递等。

- open
- begin
- attach
- transfer
- flow

- disposition
- detach
- end
- close

OpenStack 默认使用的消息总线是 RabbitMQ，也可以配置为 QPID。然而理论上可以是 Python 的 amqp 支持的任何 AMQP 消息队列。

SQL 数据库保存云基础设施搭建和运行时的状态，包括可用的实例类型、正在使用中的实例、可用的网络和项目。

在安装完 OpenStack 后，系统将会为每个项目创建一个单独的数据库，包括独立的用户名和密码，定义有独立的 schema，而且各个数据库之间没有关系，完全是独立、可以分开安装的。下面是基本安装后，需要创建的数据库：

```
+-----+
| Database      |
+-----+
| information_schema |
| ceilometer    |
| cinder        |
| glance        |
| heat          |
| keystone       |
| mysql         |
| nova          |
| ovs_neutron   |
| savanna       |
| test          |
| zabbix        |
+-----+
```

5.4 多租户

需要提醒的是，IaaS 是多租户的，也就是每个租户在 IaaS 里看到的是完全属于自己的计算、网络和存储资源。一个租户完全看不到其他租户正在和自己共享云平台资源。

一个租户可以是一个用户、一个组织、一个部门或一个企业。每个租户可以有多个用户，有自己的私有网络，当使用资源的时候，有自己可用的配额。

在早期的 OpenStack 里，“项目”一词被大量使用，一些命令行仍然使用 `project_id`，而不是 `--tenant_id` 来索引租户。为了和 EC2 保持一致项目也叫租户，也就是说在现在的 OpenStack 中，两者是完全一致和可以互换的。

一个项目就是一组被隔离的资源的组合，通常一个项目都具有：

- 单独的 VLAN；
- 卷；
- 虚拟机实例；
- 镜像；
- 密钥；
- 用户。

一个项目一般都有自己的配额限制：

- 可创建的存储卷总数；
- 存储卷总空间的大小（G）；
- 可运行 VM 总数；
- CPU 最大核数；
- 公网 IP 最大数；
- 角色。

一个私有云，可以配置下列角色。但是如果是默认安装时只有云管理员和用户两种角色。其余角色需要用户根据扩展规则来自己添加。表 5-7 列举了一些主要的角色和他们的含义。

表 5-7 OpenStack 典型角色

角 色	范 围	默 认	职 责
云管理员 (Cloud Administrator)	全局	是	拥有整个云的全部资源和操作的管理权限，其作用相当于 Linux 的系统管理员，具有对系统无限的控制权力
用户 (User)	Project	是	普通的项目用户，可以申请、创建、启动、停止虚拟机等
IT Security	全局	否	该角色仅限 IT 人员使用，可以隔离任意项目的任意 VM 添加这个角色需要增加新的角色 (Role)，并修改配置文件，如 <code>policy.json</code> 来实现

续表

角 色	范 围	缺 省	职 责
网络管理员 (Network Administrator)	项目	否	<ul style="list-style-type: none"> • 分配公网 IP (不一定是访问 Internet 的 IP) • 创建/修改 FW 规则等
项目管理员 (Project Manager)	项目	否	<ul style="list-style-type: none"> • 增加用户 • 操作镜像文件 • 运行/终止 VM 等

5.5 Keystone

OpenStack 里的认证和授权与传统的概念非常相似, 用户必须有密码或证书来证明自己, 用户可以是一个或多个组织(项目或租户)的成员。

例如, 一个云管理员可以列出所有的虚拟机实例, 而一个用户只能看到当前自己组织的虚拟机实例、可以使用的 CPU 核数、资源配额等。

认证服务(Keystone)是一个做出认证决定和提供用户信息的访问点, 被其他服务用来决定是否提供收取使用服务。策略信息保存在 policy.json 文件中。

OpenStack 认证服务提供下列功能。

- 用户管理, 记录用户以及他们的权限。
- 服务目录, 提供一个可用服务的目录, 以及访问它们的调用端点(endpoints)。

Keystone 包含下列重要定义。

- 用户: 一个用户代表一个使用云的个人, 还包括相关的信息, 如用户名、密码、电子邮件地址等。
- 租户(项目): 一个租户可以是一个项目、一个组织、一个用户群。对于任何向 OpenStack 的请求, 必须包括一个“租户”的信息。
- 角色: 一个角色包含在一个给定租户里, 哪些操作允许该用户去运行。
- 服务: OpenStack 服务, 如计算服务(Nova)、对象存储服务(Swift)、镜像服务(Glance)。

- 服务端点：某个服务的 URL 信息，用户通过它们来访问资源、执行操作。
- 令牌：用户通过认证后，由 Keystone 生成并分配给用户，Token 中包含着许多信息（“你是谁？你有哪些权限？”），如用户名称/ID、租户名称/ID、用户的角色信息等。此后的用户请求中都需要附上令牌。

图 5-6 说明了这种典型的权限模型的内部关系。

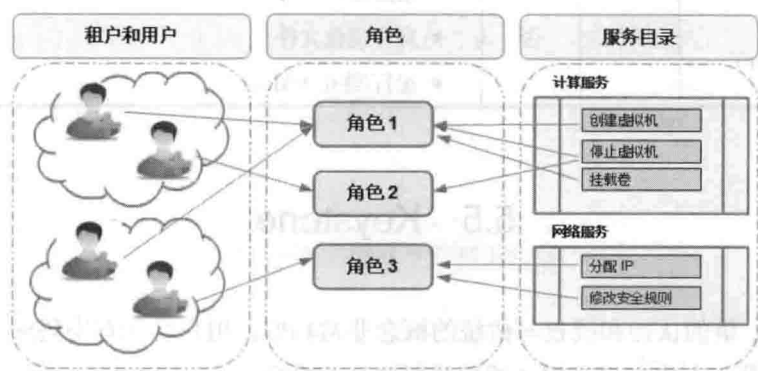


图 5-6 OpenStack 典型权限模型

表 5-8 说明了认证服务的基本信息。

表 5-8 认证服务基本信息

服务名	openstack-keystone
进程名	keystone
进程	keystone 2610 1 0 Jan25 ? 01:52:09 /usr/bin/python/usr/bin/keystone-all--config-file/etc/keystone/keystone.conf
功能	用户管理，包括用户和权限；服务目录，提供可用服务的 endpoint
TCP Port	5 000, 35 357, 8 774, 8 776, 8 773, 8 888
安装目录	/usr/lib/python2.6/site-packages/keystone
命令	/usr/bin/keystone
配置	/etc/keystone
日志	/var/log/keystone

运行时进程列表如下：

```
[root@NetworkController ~]# ps -ef | grep keystone
keystone 2610 1 0 Jan25 ? 01:27:37 /usr/bin/python /usr/bin/keystone-all --config-file /etc/keystone/keystone.conf
root 14011 13938 0 22:33 pts/0 00:00:00 grep keystone
[root@NetworkController ~]#
```

OpenStack 安装后，会创建默认的租户、权限和用户。除了管理用户之外，其他用户都是对应于系统某个服务的，如安装 Nova 组件，会创建 Nova 用户，安装 Cinder 组件，会创建 Cinder 这个用户，他们都属于 service 租户（或项目）的。表 5-9 列举了这些默认的租户（项目）和用户。需要注意的是，这些用户既不能修改，也不应该被删除，否则系统将无法正常工作。

表 5-9 OpenStack 默认租户

租 户	包 含 用 户	权 限	说 明
admin	admin	admin	系统超级用户，拥有对系统进行配置修改的一切权限，通常为系统管理员所用
service	keystone、glance、nova、cinder、neutron、ceilometer、heat	admin	由 OpenStack 每个项目的进程所拥有，对数据库进行操作，对相应的服务进行访问

使用 keystone 可以获得一系列系统用户和权限信息。下面是用 keystone 命令查看系统有哪些用户：

```
[root@CloudController ~]# keystone user-list
```

```
+-----+-----+-----+-----+
|          id          |  name  | enabled | email |
+-----+-----+-----+-----+
| 449f3d8c463541728ac574eb1950e064 |  admin  |   True  |      |
| 655d1054d47f47548212149fcabd9e9d | ceilometer |   True  |      |
| d933138c58c44a54ab4e599bebc879a5 |   ec2   |   True  |      |
| 5be4e6175c31471cb5b29190e56d5788 | glance  |   True  |      |
| d569657618604f08a494912460ae2c72 |   heat  |   True  |      |
| f4827d3b1b1741a0ab812bd809b8f1ae | neutron |   True  |      |
| 2ecec2381bf141aca859660174476875 |   nova  |   True  |      |
| 445c5706638f4b9aa9250ca6e50e75c9 |  swift  |   True  |      |
+-----+-----+-----+-----+
```

查看系统有哪些角色：

```
[root@CloudController ~]# keystone role-list
```

```
+-----+-----+
|          id          |  name  |
+-----+-----+
| 9fe2ff9ee4384b1894a90878d3e92bab | _member_ |
| 765628cf7147449f863acaf9746e95f8 |   admin  |
| 39c33291d48c4e6280594cf91d8c144b | heat_stack_user |
+-----+-----+
```

查看系统有哪些租户:

```
[root@CloudController ~]# keystone tenant-list
```

id	name	enabled
f62963ebea08405590c742655c3a0d86	admin	True
dc6e59e426c548d8bac3a801ae67b65c	service	True

查看系统提供哪些服务:

```
[root@CloudController ~]# keystone service-list
```

id	name	type	description
8369ef7bef5c4dafb343d8e954bb0dd6	ceilometer	metering	OpenStack Ceilometer service
37edc0c22a4142cc95778ce3b0ed22c3	cinder	volume	Cinder Volume Service
0abd3ce9838d45c6a86d9a448483dd2a	ec2	ec2	EC2 Compatibility Layer
a3ceaac37c3249db8b7526d3bc3a61a2	glance	image	Glance Image Service
6f12fdb4ed6e4e4d8ca7309558d3becd	heat	orchestration	Heat API
524f46d6f3854491974b0ce8f03609c6	heat-cfn	cloudformation	Heat CloudFormation API
155151a4dd934ac2bfe4cc3ffc9b9d02	keystone	identity	Keystone Identity Service
1c8a7b55602a4e85abe6909cd36cc4a7	neutron	network	QUANTUM Networking Service
475ff22b3e9346adb3173e8db49b707c	nova	compute	Nova Compute Service

查看系统有哪些服务访问端点:

```
[root@CloudController ~]# keystone endpoint-list
```

id	region	publicurl
----	--------	-----------

service_id	internalurl	adminurl
067e3327aeld41e8a58cb21c9e311f2e	RegionOne	
http://10.103.0.3:5000/v2.0		http://10.103.0.3:5000/v2.0
http://10.103.0.3:35357/v2.0	155151a4dd934ac2bfe4cc3ffc9b9d02	
2ee0fe1250a34c8289fc8ff4af44e02f	RegionOne	http://10.103.0.3:9292
http://10.103.0.3:9292		http://10.103.0.3:9292
a3ceaac37c3249db8b7526d3bc3a61a2		
49f83d8358894fa0875bdd48b29310d9	RegionOne	http://10.103.0.2:8777
http://10.103.0.2:8777		http://10.103.0.2:8777
8369ef7bef5c4dafb343d8e954bb0dd6		
521304828f78415e81c9cda52fb84b69	RegionOne	
http://10.103.0.2:8000/v1		http://10.103.0.2:8000/v1
http://10.103.0.2:8000/v1	524f46d6f3854491974b0ce8f03609c6	
6f2caa0dc1e44c64876blcdf866f52ac	RegionOne	
http://10.103.0.2:8774/v2/(tenant_id)s		
http://10.103.0.2:8774/v2/(tenant_id)s		
http://10.103.0.2:8774/v2/(tenant_id)s	475ff22b3e9346adb3173e8db49b707c	
8150788e250a4b159e187ffe933a009b	RegionOne	http://10.103.0.3:9696/
http://10.103.0.3:9696/		http://10.103.0.3:9696/
1c8a7b55602a4e85abe6909cd36cc4a7		
8d92a6459321491797e5e6e39d8a76e7	RegionOne	
http://10.103.0.2:8773/services/Cloud		
http://10.103.0.2:8773/services/Cloud		
http://10.103.0.2:8773/services/Admin	0abd3ce9838d45c6a86d9a448483dd2a	
f0a502cd49cd49bf835b3b764ef9d75d	RegionOne	
http://10.103.0.2:8004/v1/(tenant_id)s		
http://10.103.0.2:8000/v1/(tenant_id)s		
http://10.103.0.2:8000/v1/(tenant_id)s	6f12fdb4ed6e4e4d8ca7309558d3becd	
fb4ed36f5c43456f9e9b41b605977865	RegionOne	
http://10.103.0.2:8776/v1/(tenant_id)s		
http://10.103.0.2:8776/v1/(tenant_id)s		
http://10.103.0.2:8776/v1/(tenant_id)s	37edc0c22a4142cc95778ce3b0ed22c3	

keystone 和 LDAP

除了使用数据库来存储用户、租户和权限信息，认证服务也可以用如 LDAP、Windows AD 来进行认证。

然而，通常的用户场景是：企业用户并不允许 OpenStack 直接操作企业级 LDAP 服务，而是用 OpenStack 数据库来保存基本的用户信息，通过企业级 LDAP 或 AD 来提供认证，即 OpenStack 对企业级认证服务器只具有“查询”的功能。

5.6 Glance

Glance 是独立组件，供 nova-compute 下载镜像使用。功能包括镜像模板的注册、查找、存储和检索。该服务有如下特点。

- 支持多种镜像格式；
- 支持多种存储类型。

下面是镜像服务运行时的进程列表：

```
root@NetworkController:~# ps -ef | grep glance
glance    2534      1  0 Jan25  ?    00:00:00 /usr/bin/python /usr/bin/glance-api --config-file /etc/glance/glance-api.conf
glance    2602      1  0 Jan25  ?    00:00:00 /usr/bin/python /usr/bin/glance-registry --config-file /etc/glance/glance-registry.conf
glance    2770    2602  0 Jan25  ?    00:00:20 /usr/bin/python /usr/bin/glance-registry --config-file /etc/glance/glance-registry.conf
glance    2777    2602  0 Jan25  ?    00:01:04 /usr/bin/python /usr/bin/glance-api --config-file /etc/glance/glance-api.conf
root     14373  19945  0 22:31  pts/0    00:00:00 grep glance
```

虚拟机镜像可以保存在多种后端，甚至包括亚马逊 S3，对于 Glance 来说，用户只需要在配置文件中配置一下存储后端即可，Glance 会自动连接这些后台存储，为用户提供了极大的选择和灵活性。图 5-7 说明了这些后端。

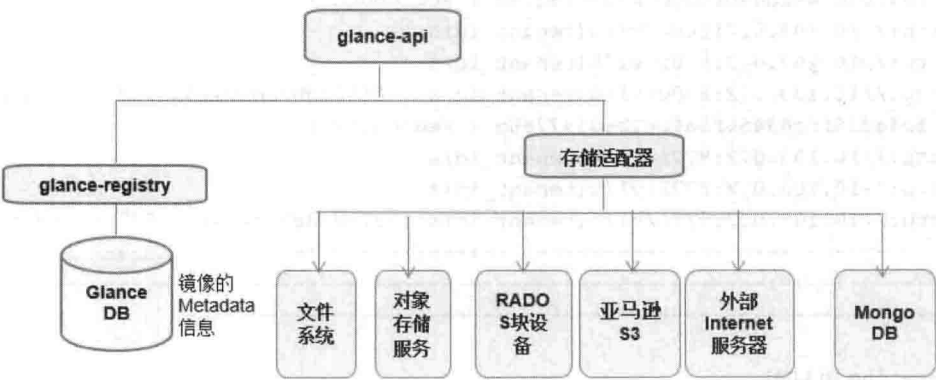


图 5-7 Glance 模块及存储后端

表 5-10 列举了 Glance 服务支持的镜像格式。

表 5-10 Glance 服务支持的镜像格式

格 式	描 述
raw	未做任何处理的镜像格式，等同于用 dd 命令对磁盘做的复制 这种格式的优点是简单，可以很容易地导出到其硬件模拟器中去。如果你的文件系统支持稀疏文件，如 Linux 的 ext2、ext3，或者 Windows 的 NTFS，那么只有写入的字段会占用空间。
vdh	由 VMware、Xen、Microsoft、VirtualBox 和许多其他 Hypervisor 支持的磁盘镜像格式
vmdk	许多 Hypervisor 支持的磁盘镜像格式
vdi	VirtualBox 和 QEMU 仿真器支持的磁盘镜像格式
iso	光盘（CD、ROM）数据归档格式
qcow2	QEMU 仿真器支持的镜像格式，可以动态扩大，支持写时复制。该格式是最万能的格式，即使是在不支持稀疏文件的文件系统上，也可以获得最小空间的镜像文件，还有 AES 加密、zlib 压缩、支持虚拟机快照等。
aki	亚马逊 kernel 镜像格式
ari	亚马逊 ramdisk 镜像格式
ami	亚马逊虚拟机镜像格式

表 5-11 是虚拟机和常用的镜像格式对应表。

表 5-11 虚拟机和常用镜像格式对应表

虚 拟 机	镜像文件格式
KVM	qcow2
XEN	qcow2
VMware	vmdk
Hyper-V	hdx
VirtualBox	vdi

表 5-12 是 glance-api 服务信息。该服务也是一个 API 服务访问点。

表 5-12 glance-api 基本信息

服务名	openstack-glance-api
进程名	glance-api
进程	Glance 2594 1 0 Jan25 ? 00:00:00/usr/bin/python/usr/bin/glance-api--config-file/etc/glance/ glance-api.conf

什么是虚拟机镜像

Glance 是镜像服务。在这里简短介绍下镜像本身。镜像文件和 ZIP 压缩包类似，它将特定的一系列文件按照一定的格式制作成单一的文件，以方便用户下载和使用，例如一个测试版的操作系统、游戏等。

虚拟机管理程序 (Hypervisor) 可以模拟出一台完整的计算机，而计算机需要操作系统。虚拟机镜像文件，就是给这个虚拟机模拟的计算机安装一个操作系统后保存的文件。

虚拟机硬盘通常是用一个物理机上的文件或一组文件来虚拟硬盘，对虚拟机硬盘的所有读写都转换为对物理机上文件的读写。虚拟机硬盘提供了很多智能操作，还带高速缓存。比如格式化硬盘，实际上并没有对物理硬盘格式化，而是简单地改写一下物理机上的文件，这个操作就很快。

虚拟光驱的镜像 (ISO)，就是把一张光盘，做成一个文件 (后缀名一般是 iso)，当用某些程序打开这个文件之后，里面的内容是和原来的光盘完全相同的 (这里指的完全相同是指数据排列方式相同，比如盘里有加密内容一类的东西也可以复制下来。和简单的复制、粘贴文件是不一样的)。这个文件就可以叫作这个光盘的一个镜像。同样地，也可以把硬盘上的某个分区，甚至整块硬盘，做成一个镜像，有很多工具可以完成这个功能。

1. raw 和 qcow2 比较

raw 的优势如下。

- 简单，是转为其他虚拟机格式的通用中间格式，并能够导出到其他硬件模拟器中去。
- 在支持稀疏文件的系统上，根据实际使用量来占用空间，而非原先设定的最大值 (比如设定最高 20GB，而实际只使用 3GB)。
- 以后能够改变空间最大值 (把最高值 20GB 提高到 200GB，qcow2 也可以，不过要转为 raw)。
- 能够直接被宿主机挂载，不用开虚拟机即可在宿主和虚拟机间进行数据传输 (注意，此时虚拟机不能开机)。
- 直接读写虚拟机硬盘里面的文件。
- qcow2 性能更好。

qcow2 的优势如下。

- 即使在不支持稀疏文件的系统上，也可以获得更小的存储空间。
- 可以动态扩大，支持写时复制。

- AES 加密。
- zlib 压缩。
- 支持多重虚拟机快照等。

然而 raw 的很多优势，必须在关机状态才可以使用，如直接挂载。而这些都是生产环境下所无法具备的条件，生产环境大都需要的是要在线具备的高级硬盘特性、稳定性。

2. vmdk 虚拟机镜像文件格式

如果使用像 WinSCP 或者 Datastore Browser 这样内置的文件浏览器在 ESX 主机上查看虚拟机的根目录，会看见与虚拟机相关的文件清单。然而并不是所有时刻都可以看见所有的文件类型，例如，当虚拟机开启时只出现.vswp 文件，当虚拟机暂停时只出现.vmx 文件。

那么组成虚拟机的所有这些文件是什么，它们有什么用？

- **.nvram 文件**：这个小型文件包括虚拟机启动过程一部分的 Phoenix BIOS。它类似于拥有 BIOS 芯片的物理服务器，能够设置硬件配置选项。一台虚拟机也应该在 NVRAM 文件里有虚拟 BIOS。当虚拟机首次启动时，按 F2 键可以访问 BIOS。不管虚拟机的硬盘配置发生了什么变化，都会保存在 NVRAM 文件里。
- **.vmx 文件**：这个文件包括虚拟机所有配置信息与硬件设置。不管对虚拟机的设置做了何种编辑，所有的信息都会以文本形式保存在这个文件里。这个文件包括与虚拟机有关的多种信息，如特殊硬件配置（例如 RAM 大小、网络接口卡信息、硬盘驱动信息和串行与并行信息）、高级能源与资源设置、VMware 工具选项以及能源管理选项。
- **VMDK 文件**：所有的虚拟磁盘由两个文件组成，一个与虚拟磁盘大小相等的大型数据文件和一个小型文本磁盘描述文件，这个描述文件描述虚拟磁盘文件的大小与形状，也包括指向大型数据文件，以及虚拟磁盘驱动扇区数、磁头数、柱面数及磁盘适配器的信息。有三种不同类型的虚拟磁盘数据文件可用于虚拟机中。
 - **flat.vmdk 文件**：这是个默认的大型虚拟磁盘数据文件，创建于添加虚拟硬盘驱动到虚拟机时（另外还有 RAM Disk Mapping (RDM) 文件，让虚拟机通过映射直接使用物理磁盘）。当使用厚磁盘时，这个文件的大小相当于创建虚拟硬盘驱动时所指定的大小。
 - **delta.vmdk 文件**：这些虚拟磁盘数据文件只用于创建虚拟机快照时。当创建了快照，对原始 flat.vmdk 的所有写入都停止，并变成只读；然后这些对虚拟磁盘的更

改将写入 `delta` 文件。每为虚拟机创建一个快照，就会生成一个 `delta` 文件，并且它们的文件名以数字递增。当快照融合到原始-`flat.vmdk` 文件后再删除时，这些文件将自动删除。

- **rdm.vmdk 文件**：这是 RDM 映射文件，用来管理 RDM 设备的映射信息。映射文件作为一般磁盘文件呈现给 ESX 主机，可用于一般的文件系统操作。不过，对于虚拟机，存储虚拟化层将映射设备作为虚拟 SCSI（小型计算机系统接口）设备呈现。映射文件的元数据包括映射设备的位置（名称解析）和映射设备的锁定状态。每在虚拟机上创建一个 RDM，就会生成一个 `-rdm.vmdk` 文件。
- **.vswp 文件**：在启动虚拟机时，如果 ESX 主机由于过量使用而消耗光其物理内存时，会创建一个内存交换文件代替物理主机内存。这些文件的大小等于分配给虚拟机的内存大小，再减去任何内存预留（默认是 0），这些文件通常创建在虚拟机里，不过只有当主机耗尽所有物理内存时才使用。由于虚拟机内存读或写入磁盘没有物理主机 RAM 快，如果虚拟机开始使用这个文件的话，性能会有所降低。这些文件会占用 VMFS（虚拟机文件系统）卷上非常大的磁盘空间，因此要确保有足够的可用空间。当虚拟机关闭或暂停时，这些文件将被删除。
- **.vmss 文件**：这个文件用于虚拟机暂停时，保存虚拟机的存储内容，以便在重新开始时继续运行。当虚拟机再次运行时，这个文件的内容将写回主机服务器的物理内存，不过，这个文件不会自动删除，除非关闭虚拟机（操作系统重启不管用）。当虚拟机再次暂停时，如果先前的暂停文件存在的话，这个文件将再次使用而不会删除和重新创建。当暂停虚拟机时，这个文件删除的话，那么虚拟机将正常启动，而不是从暂停状态启动。
- **.vmsd 文件**：这个文件与快照一起使用，用于存储元数据和其他活动在虚拟机里的每个快照的信息。这个文本文件在创建快照之前的初始大小是 0 字节，并在每次创建或删除快照时更新信息。这些现有文件中只有一个文件，不管快照运行的数量。
- **.vmsn 文件**：这个文件与快照一起使用，用于存储虚拟机在进行快照时的状态。每在虚拟机上创建一个快照就会生成一个 `.vmsn` 文件，在删除快照时，文件自动删除。这个文件的大小基于你是否选择在快照里存储虚拟机的内存状态。如果你选择要存储内存状态，那么这个文件比分配给虚拟机的 RAM 大得多。如果选择不存储快照的内存状态，那么这个文件非常小（小于 32KB）。当然，在暂停虚拟机时，这个文件的情况类似于 `.vmss`。
- **.log 文件**：这些文件创建用于存储虚拟机的日志信息，并常常用于故障检查。在虚拟机目录里，有大量这样的文件。在虚拟机关闭或重新启动时，或者如果日志文件

达到了所限制的最大值，就会创建一个新的日志文件。所保留的日志文件的数量和所限制的最大值都由虚拟机高级配置参数来定义。

- **.vmxf 文件**：这是一个附加配置文件，不是用于 ESX，而是用于与 Workstation 兼容的目的。这个文件是文本格式，Workstation 用来聚合虚拟机（VM teaming），将多个虚拟机分配成一组，作为一个单一对象开启或关闭、暂停或恢复它们。

3. qemu-img 介绍

qemu-img 命令行工具是 Xen 和 KVM 用来格式化各种文件系统的，可使用 qemu-img 格式化虚拟客户端映像、附加存储设备以及网络存储。qemu-img 的选项及用法如下。

(1) 创建新磁盘映像文件名为 `kvm_img`，格式为 `format`。

```
# qemu-img create [-s size] [-e] [-b base_image] [-f format] filename [kvm_img]
```

例如，创建一个 10MB 的镜像文件：`sina_kvm.img`，文件格式为：`raw`。

(2) 将现有映像转换成另一种格式，转换选项是将可识别格式转换为另一个映像格式。

命令格式：

```
# qemu-img convert [-c] [-e] [-f format] filename [-O output_format]
output_filename
```

例如，将 `raw` 镜像格式转换为 `qcow2` 镜像格式，转换时间由基础镜像大小决定。

(3) 获得映像信息。

`info` 参数显示磁盘映像信息。`info` 选项的格式如下：

```
# qemu-img info [-f format] filename
```

这个命令可以给出磁盘映像文件名、大小、格式等信息。使用该命令可以获得在磁盘中保留的空间的大小，但可能与显示的大小有所不同。如果在磁盘映像中保存有虚拟机快照，则此时也会显示。

qemu-img 目前支持的镜像格式包括 `raw`、`qcow`、`qcow2`、`vmdk` 等。

5.7 Nova

Nova 根据要求提供虚拟服务。这与 Rackspace 云服务器或亚马逊 EC2 类似。用户通过 nova-api 接口与 OpenStack 计算交互，计算服务的守护进程通过队列交换信息（行动）和数据库（信息）来响应请求。

图 5-8 是 Nova 主要服务以及一些分类和描述信息。

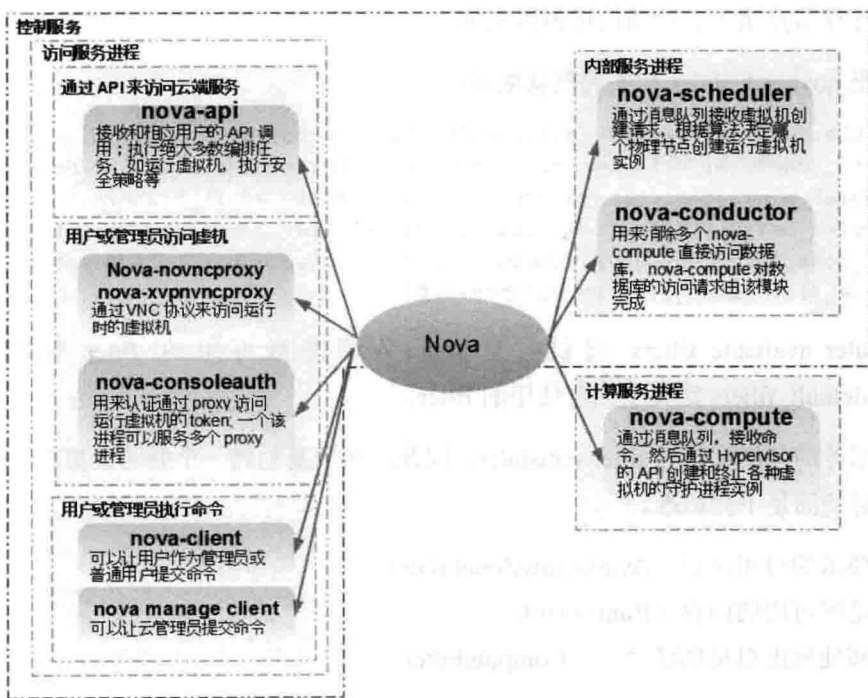


图 5-8 Nova 组件模块分解

5.7.1 nova-api

nova-api 是 OpenStack 对外服务最主要的接口。它提供了一个集中的可以查询所有的 API（OpenStack API 或 EC2 API）的端点。调用 nova-api 的接口会引发许多业务流程的活动（如运行一个实例），并实施一些管理策略（主要包括配额检查）。nova-api 组件实现了

RESTful API 功能，是外部访问 Nova 的唯一途径。nova-api 接收外部的请求并通过消息总线将请求发送给其他的服务组件，该组件也兼容 EC2 API，所以也可以用 EC2 的管理工具对 nova 进行日常管理。如果要了解详细的管理调度逻辑，可参考代码 trunk/nova/api/ec2/cloud.py 中的实现。

5.7.2 nova-scheduler

nova-scheduler 根据当前资源使用情况，决定计算节点分布到哪台计算节点上，支持插件方式扩展。nova-scheduler 作为一个后台进程运行，它会根据一定的算法从计算资源池中选择一个计算节点用于启动新的虚拟机实例

下面是 nova-scheduler 相关的默认配置：

```
scheduler_driver=nova.scheduler.multi.MultiScheduler
compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler
scheduler_available_filters=nova.scheduler.filters.all_filters
scheduler_default_filters=AvailabilityZoneFilter,RamFilter,ComputeFilter
least_cost_functions=nova.scheduler.least_cost.compute_fill_first_cost_fn
compute_fill_first_cost_fn_weight=-1.0
```

scheduler_available_filters 提供了计算服务调度器可用的 filter 的列表，而 scheduler_default_filters 定义了当前使用的 filter。

计算服务的调度配置为 filter scheduler，因此，如果要创建一个新的虚拟机，被选择的计算节点必须满足下列标准。

- 在要求的可用区里 (AvailabilityZoneFilter)。
- 有足够可用的内存 (RamFilter)。
- 能够处理虚拟机创建请求 (ComputeFilter)。

增加一个过滤器，则只需要往 scheduler_default_filters 里写入该过滤器名称即可：

```
scheduler_default_filters=AggregateInstanceExtraSpecsFilter,
AvailabilityZoneFilter,RamFilter,ComputeFilter
```

另外，计算服务也可以选择为 Chance Scheduler：

```
Chance Scheduler, nova.scheduler.chance.ChanceScheduler
```

两者的区别在于：在根据 filter 过滤完之后，FilterScheduler 会根据各个计算节点当前的权重来选择主机，而 ChanceScheduler 会随机选择一个计算节点。

图 5-9 说明了主机资源与调度器之间的关系。

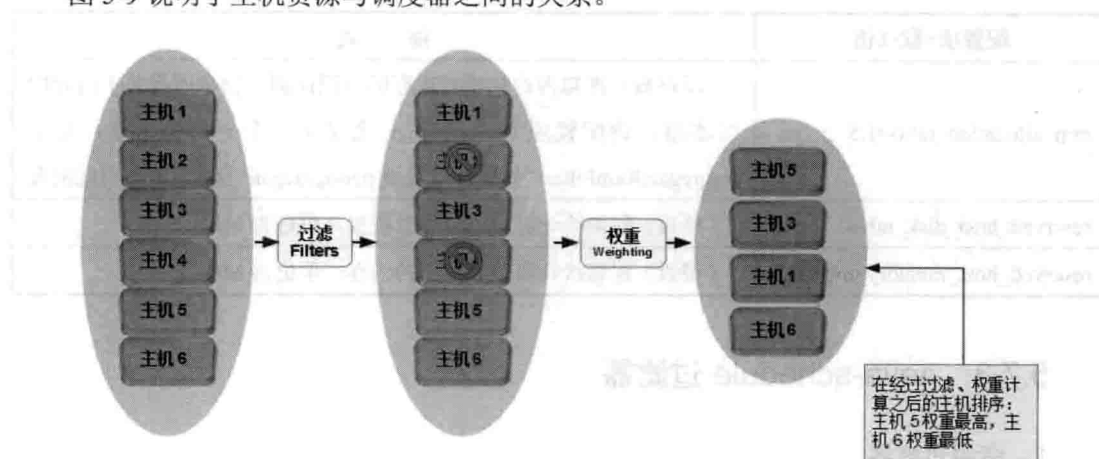


图 5-9 OpenStack 调度模型与算法

以下是 Nova-Scheduler 中的权重设计。

调度器将根据 scheduler_weight_classes 的配置来获取权重算法，计算每个主机的权重。默认为 nova.scheduler.weights.all_weighters，它将选取目前唯一可用的算法：RamWeigher。权重高的主机将被优先选择来运行虚拟机。下面是在 nova.conf 中常用的一些配置：

```
scheduler_weight_classes=nova.scheduler.weights.all_weighters
ram_weight_multiplier=1.0
```

默认值为 1.0，可以将虚拟机平均分布到各个主机。设置该项为负数，则会将虚拟机集中到某个或某些主机。表 5-14 列举了一些常用的配置项及与资源直接的关系。

表 5-14 调度算法与资源关系

配置项=默认值	描 述
cpu_allocation_ratio=16.0	(浮点数) 虚拟 CPU 到物理 CPU 的分配比例，影响所有 CPU 过滤器。该项为 CoreFilter 定义了一个全局的比例；对于 Aggregate CoreFilter，如果没有 per-aggregate 设置，则使用此配置
disk_allocation_ratio=1.0	(浮点数) 虚拟磁盘到物理磁盘的分配比例
isolated_hosts=	(列表) 计算节点，保留给特定的镜像
max_instances_per_host=50	(整数) 当某计算节点上运行的虚拟机实例超出此数时，则忽略该节点
max_io_ops_per_host=8	(整数) 忽略有太多此类操作的节点：builds/resizes/snaps/ migrations

续表

配置项=默认值	描 述
ram_allocation_ratio=1.5	（浮点数）虚拟内存到物理内存的分配比例，影响所有关于内存的过滤器。该配置项为 RamFilter 定义了一个全局的比率；对于 AggregateRamFilter，如果没有发现 per-aggregate 设置，则使用此配置
reserved_host_disk_mb=0	（整数）保留给计算节点使用的磁盘，单位为 MB
reserved_host_memory_mb=512	（整数）保留给计算节点使用的内存，单位为 MB

5.7.3 nova-schedule 过滤器

1. 资源过滤器

(1) CoreFilter

根据可用的 CPU 核来调度虚拟机实例。如果该过滤器没有设置，则调度器将会从 CPU 核的角度过载某个计算节点。该过滤器的默认配置为：

```
cpu_allocation_ratio=16.0
```

该配置允许，如果一个主机有 8 个核，则最多可以运行 128 vCPU 的虚拟机实例。

修改为下列值来关闭此过滤器：

```
cpu_allocation_ratio=1.0
```

(2) RamFilter

根据可用的内存来调度虚拟机创建。如果该过滤器没有设置，则调度器会从内存角度过载某个计算节点。

该配置允许对内存过载设置一个固定的比率：

```
ram_allocation_ratio=1.5
```

(3) DiskFilter

根据可用的磁盘空间来调度虚拟机实例，确保足够的磁盘空间可用于根分区和非持久化存储。这个过滤器可以配置，允许一定的磁盘空间过载存在。该过滤器的默认配置为：

```
disk_allocation_ratio=1.0
```

调整该值大于 1 将会允许一定的磁盘空间分配过载出现。当使用的虚拟机磁盘镜像格

式是稀疏的,或者写时复制技术(Copy-On-Write, COW)时有用,因为这些虚拟机不需要按照 1:1 来分配磁盘空间。

2. 主机聚合过滤器

(1) 设置主机聚合过滤

一个很常用的场景是:我们有许多主机,其中一部分具有某些硬件特性,我们需要把虚拟机调度到这一部分主机去。例如,如果用户需要快速的磁盘 I/O,而我们有部分主机安装了 SSD 硬盘,或者某些用户需要 GPU 加速,而部分主机安装了 GPU 卡。

为了能让调度器使用主机聚合(Host Aggregate),在运行 nova-scheduler 的主机配置文件 nova.conf 中,对配置项 scheduler_default_filters,除了原来的过滤器外,还必须包括 AggregateInstanceExtraSpecsFilter:

```
scheduler_default_filters=AggregateInstanceExtraSpecsFilter,
AvailabilityZoneFilter,RamFilter,ComputeFilter
```

例子:选择安装了 SSD 的主机。

这个例子配置计算服务,允许用户请求安装了 SSD 的主机。我们在某个可用区创建了一个 fast-io host 的主机聚合。然后,在主机聚合中增加 `ssd=true` 的键值对到主机聚合,然后增加主机 `node1`、`node2` 到该主机聚合中去:

```
$ nova aggregate-create fast-io nova
+-----+-----+-----+-----+-----+
| Id | Name | Availability Zone | Hosts | Metadata |
+-----+-----+-----+-----+-----+
| 1 | fast-io | nova | | |
+-----+-----+-----+-----+
$ nova aggregate-set-metadata 1 ssd=true
+-----+-----+-----+-----+-----+
| Id | Name | Availability Zone | Hosts | Metadata |
+-----+-----+-----+-----+-----+
| 1 | fast-io | nova | [] | {u'ssd': u'true'} |
+-----+-----+-----+-----+
$ nova aggregate-add-host 1 node1
+-----+-----+-----+-----+-----+
| Id | Name | Availability Zone | Hosts | Metadata |
+-----+-----+-----+-----+-----+
| 1 | fast-io | nova | [u'node1'] | {u'ssd': u'true'} |
+-----+-----+-----+-----+
$ nova aggregate-add-host 1 node2
```

```

+-----+-----+-----+-----+-----+
--+
| Id | Name      | Availability Zone | Hosts                | Metadata |
+-----+-----+-----+-----+-----+
--+
| 1  | fast-io  | nova              | [u'node1', u'node2'] | {u'ssd': u'true'} |
+-----+-----+-----+-----+-----+
--+

```

用命令 `nova flavor-create` 来创建一个 SSD flavor, ID 为 6, 8GB 内存, 80GB 根分区, 4 vCPUs:

```

$ nova flavor-create ssd.large 6 8192 80 4
+-----+-----+-----+-----+-----+-----+-----+
--+
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor | Is_Public | extra_specs |
+-----+-----+-----+-----+-----+-----+-----+
--+
| 6  | ssd.large | 8192      | 80  | 0          |      | 4     | 1            | True      | {}          |
+-----+-----+-----+-----+-----+-----+
--+

```

当创建了 flavor 之后, 指定一个或多个键值对, 并且匹配在主机聚合里设置的键值对。在这个例子中为 `ssd=true` 键值对。可使用 `nova flavor-key set_key` 命令来设置:

```
#nova flavor-key set_key -name=ssd.large -key=ssd -value=true
```

设置好之后, 就可以看到 `ssd.large` flavor 的 `extra_specs` 属性包含 `ssd`, 并且其值为 `true`:

```

$ nova flavor-show ssd.large
+-----+-----+-----+
| Property                | Value                |
+-----+-----+-----+
| OS-FLV-DISABLED:disabled | False                |
| OS-FLV-EXT-DATA:ephemeral | 0                    |
| disk                     | 80                   |
| extra_specs              | {u'ssd': u'true'}   |
| id                       | 6                    |
| name                     | ssd.large            |
| os-flavor-access:is_public | True                 |
| ram                      | 8192                 |
| rxtx_factor              | 1.0                  |
| swap                     |                      |
| vcpus                    | 4                    |
+-----+-----+-----+

```

现在,当用户希望以 `ssd.large flavor` 来创建虚拟机时,调度器将只考虑那些具有 `ssd=true` 键值对的主机。在本例中为 `node1` 和 `node2`。

(2) AggregateInstanceExtraSpecsFilter

匹配虚拟机类型的 `extra specs` 中的属性,和管理员定义的主机聚合中的属性。

3. 租户与主机聚合过滤器

(1) AggregateMultiTenancyIsolation

将租户隔离到指定的主机聚合中去。如果一个主机所在的主机聚合中,配置中包括了键值 `filter_tenant_id`,那么该主机将只为指定的租户创建虚拟机。一个主机可以在不同的主机聚合中。如果该主机不属于任何具有该键值的主机聚合,则可以为所有组合创建虚拟机。

(2) 主机聚合资源过滤器

① 设置 per-aggregate

```
nova aggregate-set-metadata <aggregate id> cpu_allocation_ratio=<some value>
```

② AggregateCoreFilter

实现了 `per-aggregate-resource-ratio` 功能。如果未发现 `per-aggregate` 值,则使用全局设置的 CPU 设置值。

③ AggregateRamFilter

实现了 `per-aggregate-resource-ratio` 功能。如果未发现 `per-aggregate` 值,则使用默认值。

4. 可用区过滤器

根据可用区 (Availability Zone) 过滤,当请求中包含可用区信息时,该过滤器必须设置。

5. 主机过滤器

(1) AllHostsFilter

类似于空指令,不会过滤任何主机。

(2) ComputeFilter

如果计算节点是可操作的和启用 (`enabled`) 的,则选出。通常这个过滤器总是要加入

的。

(3) DifferentHostFilter

对一组虚拟机请求，调度到不同计算节点上。为了使用该功能，需要在请求中传递一个调度暗示：用 `different_host` 作为键值，一系列虚拟机实例的 UUID 作为值。该过滤器和 `SameHostFilter` 是相反的。

使用 `nova` 命令行工具的 `--hint` 标志的例子如下：

```
$ nova boot --image cedef40a-ed67-4d10-800e-17455edce175 --flavor 1 \
--hint different_host=a0cf03a5-d921-4877-bb5c-86d26cf818e1 \
--hint different_host=8c19174f-4220-44f0-824a-cd1eeef10287 server-1
```

(4) SameHostFilter

对一组虚拟机请求，调度到相同计算节点上。为了使用该功能，需要在请求中传递一个调度暗示：用 `same_host` 作为键值，一系列虚拟机实例的 UUID 作为值。该过滤器和 `DifferentHostFilter` 是相反的。

使用 `nova` 命令行工具的 `--hint` 标志的例子如下：

```
$ nova boot --image cedef40a-ed67-4d10-800e-17455edce175 --flavor 1 \
--hint same_host=a0cf03a5-d921-4877-bb5c-86d26cf818e1 \
--hint same_host=8c19174f-4220-44f0-824a-cd1eeef10287 server-1
```

(5) ComputeCapabilitiesFilter

需要匹配虚拟机实例 `extra specs` 定义的属性和计算节点定义的属性，只有匹配，该计算节点才可以被选出。

在计算节点的 `nova.conf` 中可以定义如下属性：

```
instance_type_extra_specs= (ListOpt) a list of additional capabilities
corresponding
to instance_type_extra_specs for this compute host to advertise. Valid entries
are name=value, pairs For example, "key1:val1, key2:val2"
```

而虚拟机实例的 `extra specs` 可以定义如下属性：

```
hypervisor=xenserver;kvm,os=linux;windows
```

6. 镜像属性过滤器

(1) IsolatedHostsFilter

管理员可以定义一些特殊的镜像和一些特殊的计算节点，这些特殊的镜像创建的虚拟

机只可以被调度到这些特殊的计算节点上去。

管理员需要在 `nova.conf` 中定义这些特殊镜像和特殊计算节点的列表：

```
isolated_hosts=server1,server2
isolated_images=342b492c-128f-4a42-8d3a-c5088cf27d13,ebd267a6-ca86-4d6c-9a0e-
-bd132d6b7d09
```

(2) ImagePropertiesFilter

根据主机可以支持的属性和虚拟机的属性来进行过滤。属性包括：CPU 架构、Hypervisor 类型、虚拟机模式等。只有当主机可以支持虚拟机包含的镜像的这些属性时，才会被选出。

虚拟机镜像可以用如下方式来定义属性：

```
$ glance image-update img-uuid --property architecture=arm --property
hypervisor_type=qemu
```

过滤器检查的镜像属性包括如下内容。

- **架构**：架构描述了镜像要求的虚拟机架构，如 `i686`、`X86_64`、`arm`、`ppc64`。
- **hypervisor_type**：Hypervisor 描述了镜像要求的 hypervisor 类型，如 `xen`、`kvm`、`qemu`、`xenapi`、`powervm`。
- **vm_mode**：虚拟机模式描述了虚拟机镜像模板要求满足的虚拟机管理程序应用二进制接口（hypervisor application binary interface (ABI)）。如 Xen 3.0 paravirtual ABI 的 `'xen'`、native ABI 的 `'hvm'`、User Mode Linux paravirtual ABI 的 `'uml'`、container virt executable ABI 的 `"exe"`。

7. IP 网段过滤器

根据 IP 子网范围来调度创建虚拟机。为了使用该功能，请求中必须指定有效的 CIDR 格式的 IP 范围。有两个调度暗示可用。

- **build_near_host_ip**：子网里第一个可用的 IP 地址。
- **cidr**：指定了子网。

使用 `nova` 命令行工具指定子网范围 `192.168.1.1/24` 的例子如下：

```
$ nova boot --image cedef40a-ed67-4d10-800e-17455edce175 --flavor 1 \
--hint build_near_host_ip=192.168.1.1 --hint cidr=/24 server-1
```

8. 定制化过滤器

JsonFilter 允许用户构造一个定制化的过滤条件，并传递到调度器。过滤条件是 JSON 格式。支持下列操作：

- =
- <
- >
- in
- <=
- >=
- not
- or
- and

过滤条件支持下列变量：

- \$free_ram_mb
- \$free_disk_mb
- \$total_usable_ram_mb
- \$vcpus_total
- \$vcpus_used

使用 nova 的命令行例子如下：

```
$ nova boot --image 827d564a-e636-4fc4-a376-d36f7ebel1747 \  
--flavor 1 --hint query='[">=", "$free_ram_mb", 1024]' server1
```

5.7.4 nova-volume

nova-volume 给虚拟机分配额外持久化的存储，管理持久卷到计算实例的创建、连接和分离。该服务功能现在已经完全被 Cinder 服务所替代，因此不建议使用。

5.7.5 nova-compute

nova-compute 接收队列中的动作，然后执行一系列的命令（如启动 KVM 实例），

同时更新数据库中的状态。**nova-compute** 并不包含任何虚拟化软件，而是定义了在主操作系统上运行虚拟化的底层交互机制的驱动程序，以及一套基于 Web API 的接口。

nova-compute 一般运行在计算节点上，通过消息总线接收虚拟机生命周期管理指令并实施具体的管理工作，如虚拟机的创建、终止、迁移或改变资源大小等操作。

对于如何选择 Hypervisor，必须根据预算、资源、需要支持的功能和技术规范等来决定。目前社区对虚拟化管理程序的支持，主要围绕 KVM 和 Xen 来进行。

图 5-10 说明了一个 OpenStack 管理多个主机组成的计算资源池，其中包括三种虚拟机管理程序：VMware ESX/ESXi、Microsoft Hyper-V、KVM。

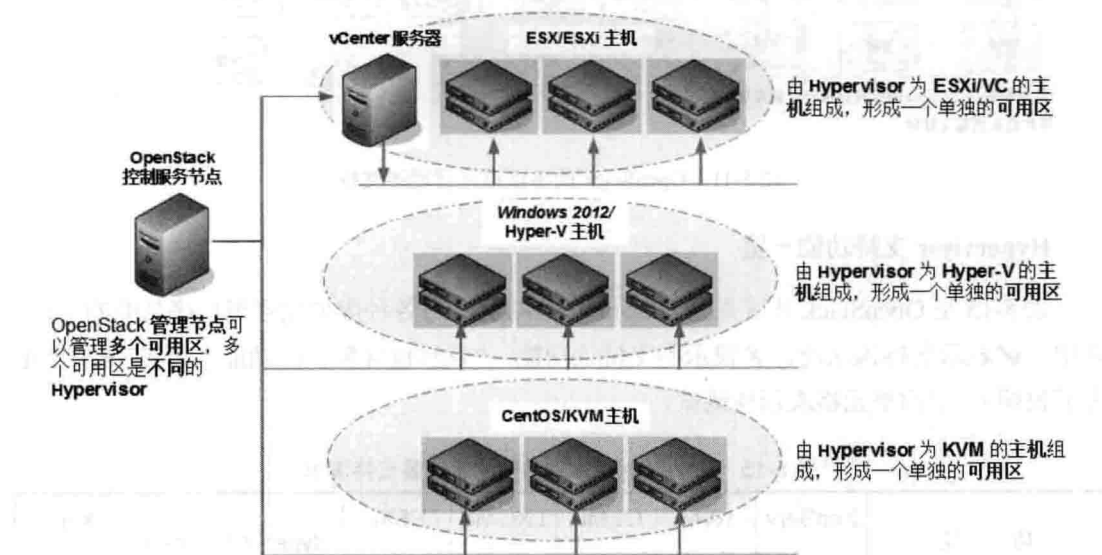


图 5-10 OpenStack 管理混合的虚拟化管理器

Nova 并不提供虚拟化功能，但通过 Libvirt API 可对如下虚拟化平台进行管理。

- **KVM:** Kernel-based Virtual Machine。
- **LXC:** Linux 容器。
- **QEMU:** Quick EMUlator。
- **UML:** User Mode Linux。
- VMware ESX/ESXi 4.1 update 1。
- **Xen:** XenServer 5.5, Xen Cloud Platform (XCP)。

通过计算服务，可以在不同的可用区里，用多种虚拟机管理程序来编排云服务。

图 5-11 列举了 OpenStack 可以支持的虚拟化程序。

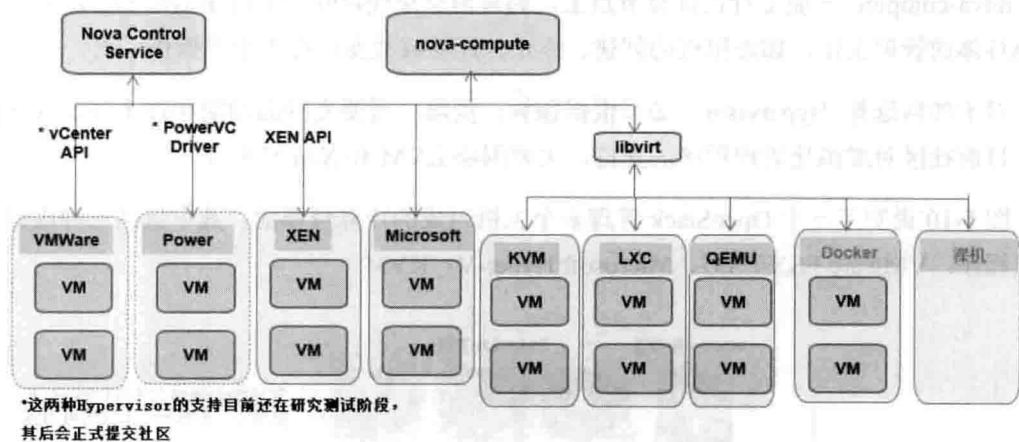


图 5-11 OpenStack 的多虚拟化管理器支持

Hypervisor 支持功能一览

表 5-15 是 OpenStack 社区直到 Havana 版本为止,对各种虚拟化管理程序功能的支持。其中，✓表示支持该功能，✗表示不支持该功能，*表示包含警告的功能（警告/提示参见表后说明），空白单元格表示待测试。

表 5-15 Havana 版本对虚拟化管理器支持清单

功 能	XenServ er/XCP	KVM on X86	QEMU on X86	LXC via libvirt	ESXi/ VC	Hyper-V	Docker	Xen/ Libvirt
Launch	✓	✓	✓	✓	✓	✓	✓	✓* ^①
Reboot	✓	✓	✓	✓	✓	✓	✓	✓
Terminate	✓	✓	✓	✓	✓	✓	✓	✓
Resize	✓	✓	✓		✓	✓	✗	✓
Rescue	✓	✓	✓		✓		✗	
Pause	✓	✓	✓		✗	✓	✗	✓
Un-pause	✓	✓	✓		✗	✓	✗	✓
Resume	✓	✓	✓		✓	✓	✗	✓
Inject Networking	✓* ^②	✓* ^①	✓* ^①		✓*	✓		

续表

功 能	XenServer/XCP	KVM on X86	QEMU on X86	LXC via libvirt	ESXi/VC	Hyper-V	Docker	Xen/Libvirt
Inject File	✓	✓*	✓*			X ^{*⑧}	X	
Serial Console Output	✓ ^③	✓	✓	X	✓	X	✓	
VNC Console	✓*	✓*	✓*	✓	✓		X	
SPICE Console	X	✓	✓		X	X	X	
Attach Volume	✓	✓	✓	X	✓	✓	X	✓
Detach Volume	✓	✓	✓	X	✓	✓	X	✓
Live Migration	✓	✓	✓		X ^⑨	✓	X	✓
Snapshot	✓	✓	✓		✓	✓	✓	✓ ^{*⑪}
iSCSI	✓	✓	✓		✓	✓		
iSCSI CHAP		✓	✓					
Fibre Channel		✓	✓					
Set Admin Pass	✓	X	X		X	X ^{*⑧}	X	
Get Guest Info	✓	✓	✓		✓	✓	✓	
Get Host Info	✓	✓	✓		✓	✓	✓	
Glance Integration	✓	✓	✓		✓	✓	✓	✓
Service Control	✓	✓	✓		✓			
VLAN Networking	✓	✓	✓	✓	✓	X ^{*⑦}		✓
Flat Networking	✓	✓	✓	✓	✓	✓		✓
Security Groups	✓	✓	✓	✓	X ^{*⑧}			✓
Firewall Rules	✓	✓	✓			X ^{*⑫}		✓
Routing	✓	✓	✓		✓	X ^{*⑦}		✓
nova diagnostics	✓	✓	✓				X	✓
Config Drive	✓	✓	✓	✓	✓	✓	X	✓
Auto configure disk	✓						X	
Evacuate	✓	✓						✓
Volume swap	X	✓	✓		X	X	X	
Volume rate limiting	X	✓	✓		X	X	X	

注：① 网络注入只适用于 nova-network 的 Flat mode，且只适用于 Debian/ Ubuntu 的虚拟机，而且只适用于开机启动时。

② XenServer 的集成也支持使用一个虚拟机 Agent 来注入虚拟机的网络设置；这种注入可以在任意时间通过管理接口的扩展注入，只需要在虚拟机里安装 Agent 就可以实现。

③ （未使用）

④ XenAPI 在 Havana 版本添加了对串行控制台的支持。

⑤ （未使用）

⑥ Windows 是由 cloud-init 提供的：<http://www.cloudbase.it/cloud-init-for-windows-instances/>。

⑦ 在 Hyper-V 虚拟化管理程序中，VLAN 和路由支持必须通过使用 Quantum 的 Hyper-V 插件，Nova-network 中不支持 VlanManager 模式。

⑧ 在使用 VC/ ESX 时，安全组（Security Groups）需要 Quantum 的 NVP 插件。

⑨ <https://bugs.launchpad.net/nova/+bug/1192192>。

⑩ 不支持写入时复制（Copy-On-Write）镜像。

⑪ 只有冷快照可用。

⑫ <https://bugs.launchpad.net/nova/+bug/1269448>。

5.7.6 nova-network

nova-network 执行网络管理，给虚拟机分配网络，创建路由，使外部 PC 可以直接访问。它接受队列中的网络任务，然后执行任务操作网络（如设立桥接接口或更改 iptables 规则）。图 5-12 说明了 Nova 组件里与 nova-network 相关的一些服务及其信息。

如图 5-13 所示，nova-network 包括三个网络管理模块，可以用来创建三种基本的网络拓扑，分别是扁平网络（Flat Net Work）、扁平 DHCP 网络（Flat DHCP Network）、VLAN 网络（VLAN Network）等。VLAN Network Manager 这种方式较适合于共有云，但也在企业私有云方面得到广泛使用。在私有云方面，如果 IP 充足，而且为了方便地互联互通，简单的扁平结构网络比较适合。OpenStack 支持浮动 IP，该特性可以方便地通过更改 IP 来容错转移（Failover）或者迁移。

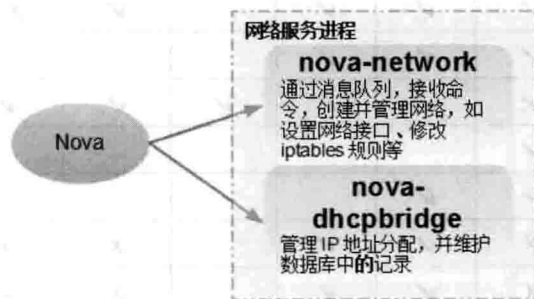


图 5-12 Nova-network 组成

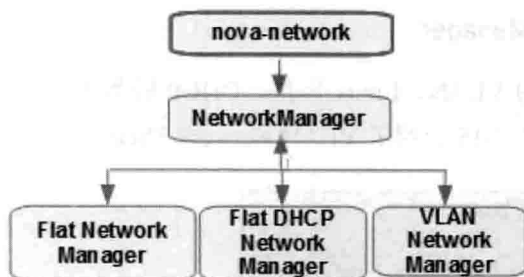


图 5-13 Nova-network 支持的网络管理器

1. Flat Network Manager

这种模式是 Nova-network 里最简单的网络模式，具有下列特点。

- 不使用 VLAN，仅支持一个网络。
- 需要在各节点上手动创建桥设备。
- 虚拟机通过一个地址池来获得固定 IP 地址。
- 虚拟机的网络配置是在启动前被注入到 `/etc/network/interfaces` 文件中的，所以该模式仅支持 Linux 系列的虚拟机。

2. Flat DHCP Network Manager

该模式下，需要启动一个 DHCP 服务器为 VM 分配固定 IP 地址，除此之外，基本和第一种模式相同。

图 5-14 对两种网络管理进行了对比。可以看出，它们的区别只在于 DHCP 功能。

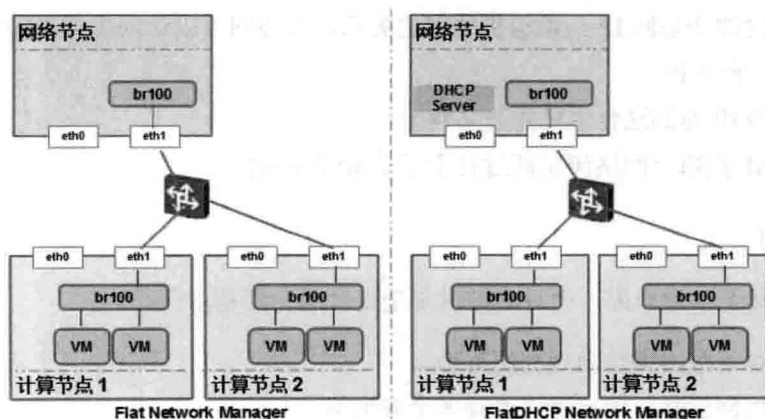


图 5-14 FlatNetwork 与 FlatDHCPNetwork 对比

3. VLAN Network Manager

每个项目均有自己的 VLAN、Linux 网桥、DHCP 服务器，所有属于同一 VLAN 的虚拟机连接到同一网桥。图 5-15 介绍了 VLAN Network Manager 的内部网络组件结构。

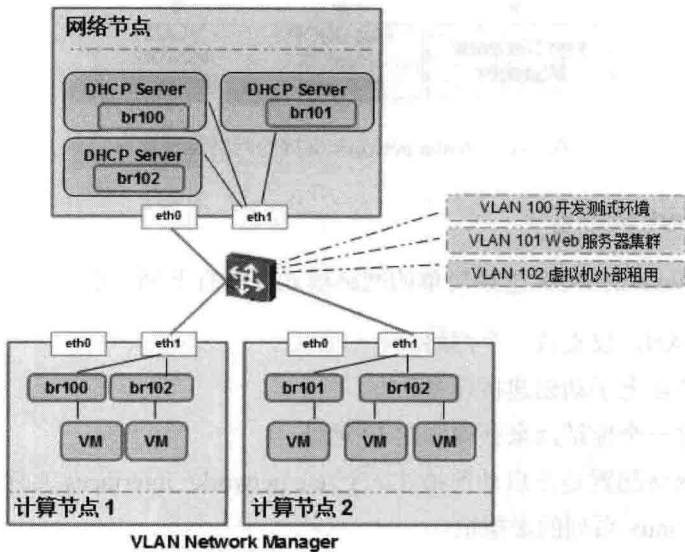


图 5-15 VLAN Network Manager

4. 浮动 IP

使用浮动 IP 是基于下列原因。

- 每个项目可使用的公网 IP 数量总是受限。
- 需要随时改变公网 IP 与虚拟机的绑定关系，使公网可以访问虚拟机的服务。
 - 确定一台主机。
 - 把浮动 IP 加到这台主机的公共网卡。
 - 用 NAT 把所有网络流量通过这个浮动 IP 来路由。

5. 安全组

安全组是基于下列原因，而对虚拟机安全提供的一层基本保护。

- 安全组是一些网络访问规则的集合。
- 在虚拟机创建时，用户可以选择多个安全组。
- 如果没有指定安全组，则使用默认的安全组。

- 安全组在主机曾生效。
- 安全组只提供了入口流量的过滤，而没有提供出口流量的过滤。

6. nova-network 的局限性

- 计算服务与网络服务强耦合，与 OpenStack 的本意不符。OpenStack 是云服务的集合。
- 支持的网络服务十分有限，而且可扩展性很差，只提供了最基本的一些技术。
 - 只有 VLAN 可以用于多租户场景。
 - 使用的是最简单的 Linux 网桥没有高级的 QoS、ACL（访问控制列表）或监控。
 - 网络控制节点是一个集中的单点故障点。
- 对多厂家、不同技术支持有限。
- 无法根据多租户对网络进行控制。
- 云计算时代的网络与传统网络不同。
 - 大规模、高密度、多租户接入。
 - 构建大二层网络。
 - Vmotion 时网络配置自动跟随。

5.7.7 nova-conductor

在早期的 nova-compute 设计时，nova-compute 是直接访问数据库的。这有两个问题：安全和性能。对于安全来说，如果一个计算节点被攻破，则攻击者可以直接访问数据库；对于性能来说，nova-compute 对数据库的访问是单线程和阻塞式的，而数据库处理则是串行的，而不是平行的，这变成一个瓶颈。

nova-conductor 服务解决了这些问题。它成为 nova-compute 对数据库操作的一个代理，现在 nova-compute 将首先访问 nova-conductor 服务，然后 nova-conductor 访问数据库。因为 nova-compute 不再直接访问数据库，安全问题就不存在了。而 nova-conductor 本身实现为非阻塞式，所以从各个计算节点来的访问请求也变成并行。

引入 nova-compute 后，会带来以下几个方面的影响。

(1) 安全。

- 优点：因为 compute 节点通常会运行不可信的用户负载，一旦服务被攻击或用户虚拟机的流量溢出，则数据库会面临直接暴露的风险。通过增加 nova-conductor 会提

高安全性。

- **局限性：**虽然 nova-conductor 限制了直接数据库访问，但 compute 节点仍然可以通过 nova-conductor 获取所有节点的虚拟机数据，执行删除虚拟机等操作。特别是如果使用 E 版的 multihost 部署模式，使用到 nova-api-metadata 和 nova-network 等服务时不能使用 nova-conductor。这样计算节点仍然是不安全的。

(2) 方便升级。

- **优点：**将 nova-compute 与数据库解耦的同时，也会与模式 (schema) 解耦，因此就不用担心旧的代码访问新的数据库模式。

(3) 性能

- **优点：**当使用 green thread 时，所有的 DB 访问是阻塞的。而如果使用 nova-conductor，可以多线程使用 RPC（远程过程调用）访问。
- **局限性：**通过 RPC 访问 nova-conductor 会受网络时延的影响。同时，nova-conductor 的 DB 访问也是阻塞的，如果 nova-conductor 实例较少，效果反而会更差。

如果使用的是 nova-network 并且是 multi-host 模式，nova-compute 还是必须直接访问数据库。如果 nova-compute 希望继续直接访问 DB，需要进行如下配置：

```
[conductor]
use_local=true
```

5.7.8 服务横向扩展

像数据库和消息总线一样，nova-api、nova-conductor 这些服务也是可以水平扩展的。可以有多个 nova-api、nova-conductor 在同一服务器或不同服务器上运行，以提高服务能力。传统的类似 HTTP 的负载均衡技术可以用来提高这些服务的高可靠性和容错性。

nova-scheduler 也是可以水平扩展的。所以为了高可靠性，或者服务大规模的安装，可以考虑运行多个 nova-scheduler 实例。这些实例是监听共享的消息总线，所以不需要其他的负载均衡器。

1. 经典部署方式

如图 5-16 所示，在经典的部署方式（一个控制节点，多个计算节点）中，可以将 nova-conductor 运行在计算节点，这也就意味着控制节点将运行更多的任务，负载变重，因

此有必要对负载做监控，必要时需要扩展控制服务。比如当在多个节点运行 nova-api 时，就需要在前端做负载均衡；当在多个节点运行 nova-scheduler 或 nova-conductor 时，负载均衡的任务就由消息队列服务器来实现。

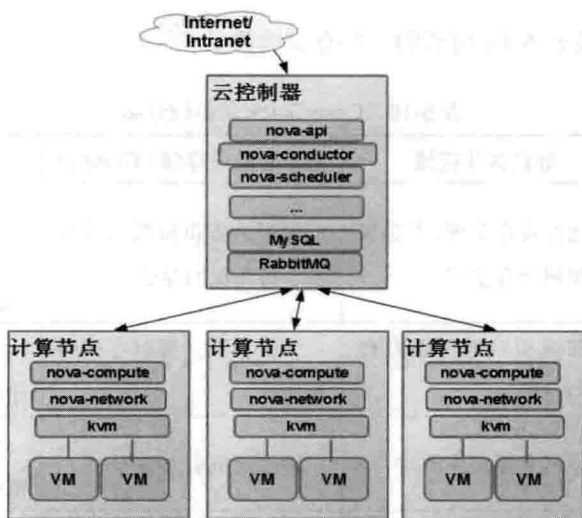


图 5-16 经典部署方式

2. 负载均衡部署

图 5-17 说明了负载均衡部署模型的内部关系。

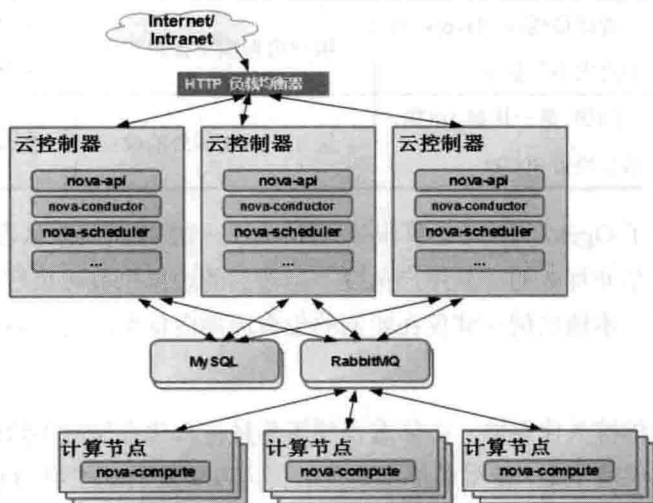


图 5-17 负载均衡部署模型

5.8 存储

OpenStack 里涉及表 5-16 列举的一些存储类型。

表 5-16 OpenStack 的存储分类

	非持久化存储	基于卷的块存储（Cinder）	对象存储（Swift）
用来……	运行操作系统，并提供虚拟机暂存空间	用来为虚拟机增加额外的、持久化的存储	用来保存虚拟机镜像和数据，也可以用于大数据、Web 数据等
一直存在直到……	在虚拟机被销毁时自动被释放	一直存在，直到卷被明确删除	一直存在，直到被明确删除
如何访问到……	通过虚拟机来访问	通过虚拟机来访问	可以独立被访问，而不依赖于通过虚拟机的方式
通过什么来访问……	在部署了 OpenStack 计算服务的节点，以文件系统的形式出现	通过 OpenStack 块存储协议来 mount 上（如 iSCSI）	REST API
被谁管理……	OpenStack 计算(Nova)	OpenStack 块存储服务（Cinder）	OpenStack 对象存储服务（Swift）
如何确定大小	管理员通过 flavors 来指定大小等信息	用户根据需要指定大小	可以非常容易地根据需要扩展
典型的使用例子	例如，第一块盘 10GB，第二块盘 30GB	例如，1TB 额外的硬盘	例如，200TB 数据存储

如果只是部署了 OpenStack 的计算服务（Nova），则虚拟机是默认没有持久化存储的，虚拟机看到的磁盘是非持久的。从用户角度，当他们的虚拟机被终止后，那些磁盘和磁盘上的数据都消失了。本地实例存储仅在实例的生命周期内保留，这是启动实例的一种经济方式。

为了给用户提供持久化存储，必须选择使用那种持久化存储提供给用户，数据将独立于实例的生命周期保留下来，使得在停止实例后仍可以重新启动使用。OpenStack 提供两种类型的持久化存储——对象存储和块存储。

5.8.1 对象存储

OpenStack Swift 不像传统的硬盘，而是为了解除传统的 POSIX-style 风格文件系统的限制。它的访问全部是基于 API 的，如 http。如果不需要原子操作，可以很容易地扩展存储系统，并且避免单点故障。它类似于亚马逊的 S3 或者 Rackspace 的 Cloud Files。OpenStack 对象存储可以不依赖于计算服务（Nova）而独立工作。

除了可以保存各种文档、图像、电子邮件、大数据等数据文件，Swift 还支持跨区域数据中心的数据复制，对于搭建企业备份容灾系统也是最好的选择。

当使用对象存储时，用户将通过一套 REST API 来访问自己的二进制对象。如果希望为用户提供保存和管理大型数据集合的功能，对象存储是一个很好的选择。另外，OpenStack 也可以将虚拟机镜像保存于对象存储中。

Swift 对象存储已经大量地运用于很多生产系统，是用户完全可以信赖的存储系统之一。其优点如下。

- 非常容易扩容。
- 非常容易线性扩展。
- 无中心架构（最终一致性哈希环）。
- 无单点故障。
- 开源且容易搭建。

5.8.2 块存储

块存储（有时也叫卷存储），就是提供了块设备存储的接口，为用户提供一个块设备。用户在使用虚拟机时，需要将这个块（卷）挂载到虚拟机才可以使用。这些卷是持久化的，它们可以从虚拟机卸载，然后重新挂载到其他虚拟机上，数据依然保持完整。

首先，一个硬盘是一个块设备。内核检测到硬盘后，在 /dev/ 下会看到 /dev/sda/。为了用一个硬盘来得到不同的分区做不同的事，我们使用 fdisk 工具得到 /dev/sda1、/dev/sda2 等。这种方式通过直接写入分区表来规定和切分硬盘，是最死板的分区方式。

块存储服务，在 OpenStack 里是由 Cinder 项目来提供的，Cinder 支持多种后端，需要相应的驱动来支持。当选择后端时，必须确保有相应的块存储驱动来支持 Cinder。

这些驱动和传统的块存储驱动工作原理有些不同，在基于 NFS 和 GlusterFS 的文件系统中，一个单独的文件被创建，然后作为一个“虚拟”的卷，挂载到虚拟机上。大多数存储驱动允许虚拟机实例直接访问底层存储硬件的块驱动，来提高读写 I/O。

块存储服务提供的基本资源如下。

- **卷：**分配的块存储资源，可以挂载到虚拟机作为额外的存储资源，或者可以用作 root 来存放启动的虚拟机文件。卷为持久化的可读/写块存储设备，通常通过 iSCSI 在计算节点挂载到虚拟机。
- **快照：**一个只读的卷在某个时刻的副本，可以在卷当前正在使用或者在可用状态时候创建。快照可以用来创建新的卷。
- **备份：**可以归档的卷的副本。

1. LVM

LVM 是一种逻辑卷管理器。通过 LVM 来对硬盘创建逻辑卷组和得到逻辑卷，要比 fdisk 方式更加弹性。LVM 基于 Device-mapper 用户程序实现。Device-mapper 是一种支持逻辑卷管理的通用设备映射机制，为存储资源管理的块设备驱动提供了一个高度模块化的内核架构。

2. SAN & iSCSI

SAN（存储区域网络）是主流的企业级存储方式。

大部分 SAN 使用 SCSI 协议在服务器和存储设备之间传输和沟通，通过在 SCSI 之上建立不同镜像层，可以实现存储网络的连接。常见的有 iSCSI、FCP、Fibre Channel over Ethernet 等。

SAN 通常需要在专用存储设备中建立，而 iSCSI 是基于 TCP/IP 的 SCSI 映射，通过 iSCSI 协议和 Linux iSCSI 项目，我们可以在常见的 PC 机上建立 SAN 存储。

3. 分布式块存储

在面对极具弹性的存储需求和性能的要求下，单机或者独立的 SAN 越来越不能满足企业的需要。如同数据库系统一样，块存储在向上扩展的瓶颈下也面临着横向扩展的需要。我们可以用以下几个特点来描述分布式块存储系统的概念：

- 分布式块存储可以为任何物理机或者虚拟机提供持久化的块存储设备。

- 分布式块存储系统管理块设备的创建、删除、挂载和卸载。
- 分布式块存储支持强大的快照功能，快照可以用来恢复或者创建新的块设备。
- 分布式存储系统能够提供不同 I/O 性能要求的块设备。

(1) GlusterFS

GlusterFS 是一个开源的、可扩展的分布式文件系统，基于可堆叠的用户空间设计，可为各种不同的数据负载提供优异的性能。GlusterFs 目前已经在许多生产系统中得到非常广泛的应用，不过首要应用类型是作为分布式共享文件系统，存储海量企业级文件，或者作为虚拟机非持久化存储用于热迁移。GlusterFS 是由 Red Hat 参与社区并主要推动的，也用于 Red Hat 的内部私有云平台。

GlusterFS 支持运行在任何标准 IP 网络上标准应用程序的标准客户端，用户可以在全局统一的命名空间中使用 NFS/CIFS（通用 Internet 文件系统）等标准协议来访问应用数据。GlusterFS 使得用户可摆脱原有的独立、高成本的封闭存储系统，能够利用普通廉价的存储设备来部署可集中管理、横向扩展、虚拟化的存储池，存储容量可扩展至 TB/PB 级。GlusterFS 的主要特征如下。

- **扩展性和高性能。**GlusterFS 利用双重特性来提供几 TB 至数 PB 的高扩展存储解决方案。Scale-Out 架构允许通过简单地增加资源来提高存储容量和性能，磁盘、计算和 I/O 资源都可以独立增加，支持 10GbE 和 InfiniBand 等高速网络互联。Gluster 弹性哈希（Elastic Hash）解除了 GlusterFS 对元数据服务器的需求，消除了单点故障和性能瓶颈，真正实现了并行化数据访问。
- **高可用性。**GlusterFS 可以对文件进行自动复制，如镜像或多次复制，从而确保数据总是可以访问的，甚至是在硬件故障的情况下也能正常访问的。自我修复功能能够把数据恢复到正确的状态，而且修复是以增量的方式在后台执行的，几乎不会产生性能负载。GlusterFS 没有设计自己的私有数据文件格式，而是采用操作系统中主流标准的磁盘文件系统（如 EXT3、ZFS）来存储文件，因此数据可以使用各种标准工具进行复制和访问。
- **全局统一命名空间。**全局统一命名空间将磁盘和内存资源聚集成一个单一的虚拟存储池，对上层用户和应用屏蔽了底层的物理硬件。存储资源可以根据需要在虚拟存储池中进行弹性扩展，比如扩容或收缩。当存储虚拟机映像时，存储的虚拟映像文件没有数量限制，成千虚拟机均通过单一挂载点进行数据共享。虚拟机 I/O 可在命名空间内的所有服务器上自动进行负载均衡，消除了 SAN 环境中经常发生的访问热

点和性能瓶颈问题。

- **弹性哈希算法。**GlusterFS 采用弹性哈希算法在存储池中定位数据，而不是采用集中式或分布式元数据服务器索引。在其他的 Scale-Out 存储系统中，元数据服务器通常会导致 I/O 性能瓶颈和单点故障问题。GlusterFS 中，所有在 Scale-Out 存储配置中的存储系统都可以智能地定位任意数据分片，不需要查看索引或者向其他服务器查询。这种设计机制完全并行化了数据访问，实现了真正的线性性能扩展。
- **弹性卷管理。**数据储存在逻辑卷中，逻辑卷可以对虚拟化的物理存储池进行独立逻辑划分而得到。存储服务器可以在线进行增加和移除，不会导致应用中断。逻辑卷可以在所有配置服务器中增长和缩减，可以在不同服务器迁移进行容量均衡，或者增加和移除系统，这些操作都可在线进行。文件系统配置更改也可以实时在线进行并应用，从而适应工作负载条件变化或在线性能调优。
- **基于标准协议。**GlusterFS 存储服务支持 NFS、CIFS、HTTP、FTP 以及 Gluster 原生协议，完全与 POSIX（可移植操作系统接口）标准兼容。现有应用程序不需要做任何修改或使用专用 API，就可以对 GlusterFS 中的数据进行访问。这在公有云环境中部署 GlusterFS 时非常有用，GlusterFS 对云服务提供商专用 API 进行抽象，然后提供标准 POSIX 接口。

配置 GlusterFS 支持块存储非常简单，在安装并配置完 GlusterFS 集群，并安装好驱动后，在 cinder.conf 文件中加入如下代码即可：

```
Volume_driver=cinder.volume.drivers.glusterfs.GlusterfsDriver
```

额外地，加入 GlusterFS 需要的配置信息如表 5-17 所示。

表 5-17 Gluster 的配置

配置项=默认值	描 述
glusterfs_disk_util=df	（字符串）计算空余空间时，使用 du 或者 df
glusterfs_mount_point_base=\$state_path/mnt	（字符串）包含 GlusterFS 共享的父目录
glusterfs_qcow2_volumes=False	（布尔值）创建的卷是 qcow2 格式，而不是 raw 格式
glusterfs_shares_config=/etc/cinder/glusterfs_shares	（字符串）包含 GlusterFS 共享列表的文件
glusterfs_sparsed_volumes=True	（布尔值）将卷创建为稀疏文件，稀疏文件几乎不占用空间。如果设置成 False，则创建成普通文件时不但卷本身占用很大空间，而且卷的创建也会占用许多时间

(2) Ceph

Ceph 是开源实现的 PB 级分布式文件系统，其分布式对象存储机制为上层提供了文件接口、块存储接口和对象存储接口。Inktank 是 Ceph 的主要支持商，也是目前 Ceph 开源社区的主要力量。

Ceph 目前是 OpenStack 的开源块存储实现系统（即 Cinder 项目的存储后端驱动（backend driver）之一），其实现分为三个部分：OSD、Monitor、MDS。OSD 是底层对象存储系统，Monitor 是集群管理系统，MDS 是用来支持 POSIX 文件接口的元数据服务器（Metadata Server）。Ceph 专注于扩展性、高可用性和容错性。它放弃了传统的元数据（Metadata）查表方式（HDFS），而改用算法（CRUSH）去定位具体的文件块（block）。

利用 Ceph 提供的 RULES 可以弹性地制定存储策略和存储池（Pool）的选择，Monitor 作为集群管理系统掌握了全部的 Cluster Map，客户端在没有 Map 的情况下需要先向 Monitor 请求得到，然后通过 Object id 计算相应的 OSD Server。

配置 Ceph，需要在 cinder.conf 中加入需要的配置信息，如表 5-18 所示。

表 5-18 Ceph 的配置

配置项=默认值	描 述
rbd_ceph_conf=	（字符串）Ceph 配置文件路径
rbd_flatten_volume_from_snapshot=False	（布尔值）将从快照来的卷变成扁平普通文件，去除依赖
rbd_max_clone_depth=5	（整形值）一个卷在把它扁平化之前，用来创建复制的复合节点可用的最大深度
rbd_pool=rbd	（字符串）用来保存块设备（RBD）卷的分布式对象存储（RADOS）池
rbd_secret_uuid=None	（字符串）rbd_uservolumes 的 libvirt uuid 使用的密钥
bd_user=None	（字符串）RADOS 客户名，用来访问 RBD 卷； 只在使用 Ceph 认证时使用
volume_tmp_dir=None	（字符串）如果驱动没有写进 Ceph 卷，临时目录用来存放镜像文件

5.8.3 Cinder

图 5-18 介绍了 Cinder 的工作模型。Cinder 是 OpenStack 中提供类似于 EBS（弹性块存储）块存储服务的 API 框架。它并没有实现对块设备的管理和实际服务，而是为后端不同的存储结构提供了统一的接口，不同的块设备服务厂商在 Cinder 中实现其驱动支持来和 OpenStack 进行整合。后端的存储可以是 DAS（直接附加存储）、NAS（网络附加存储）、SAN（存储区域网络）、对象存储或者分布式文件系统。也就是说，Cinder 的块存储数据完整性、可用性保障是由后端存储提供的。在 CinderSupportMatrix 中可以看到众多存储厂商如 NetAPP、IBM、SolidFire、EMC 和众多开源块存储系统对 Cinder 的支持。图 5-19 列举了到 Havana 版本为止支持的后端存储类型。

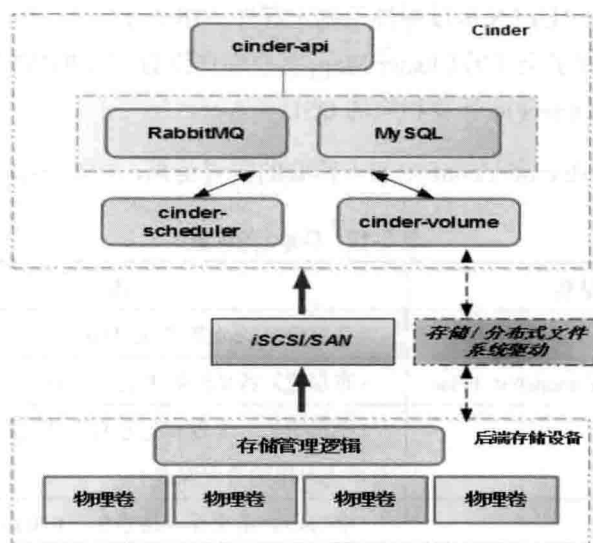


图 5-18 Cinder 工作模型

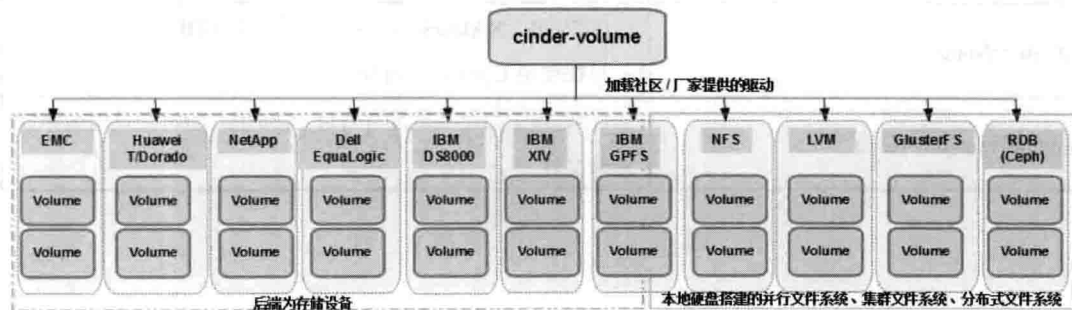


图 5-19 Cinder 支持的后端存储类型

Cinder 只是提供了一层抽象，然后通过其后端支持的驱动（driver）实现发出命令来得到回应。关于块存储的分配信息以及选项配置等会被保存到 OpenStack 统一的 DB 中。

图 5-20 是 Cinder 包含的进程信息。

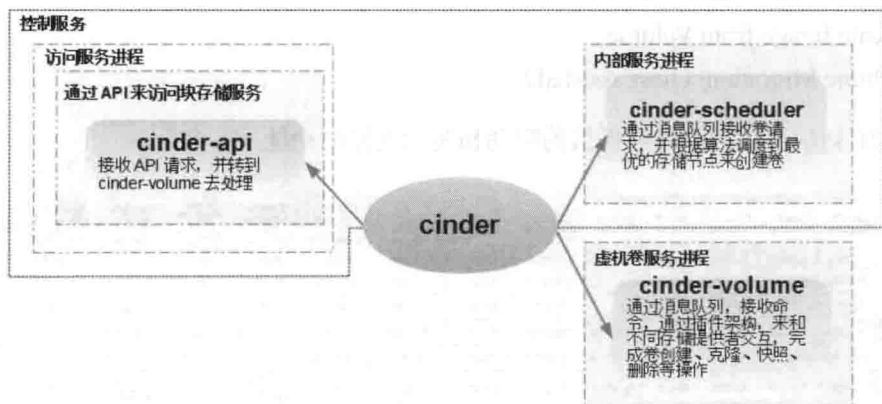


图 5-20 Cinder 组成

cinder-volume 一般运行在存储节点上（类似于 Agent 的作用），主要执行卷（Volume）相关的功能，如创建卷、为虚拟机绑定卷或解绑定卷等。下面是 Cinder 安装运行于同一个节点后运行的服务。

```
[root@CloudController rc2.d]# ps -ef | grep cinder
root      1379      754  0 22:27 pts/0    00:00:00 grep cinder
cinder    2961      1  0 Jan25  ?      00:02:32 /usr/bin/python /usr/bin/cinder-api --config-file /etc/cinder/cinder.conf --
cinder    2977      1  0 Jan25  ?      00:14:59 /usr/bin/python /usr/bin/cinder-scheduler --config-file /etc/cinder/cinder.c
cinder    4525      1  1 Jan25  ?      01:58:37 /usr/bin/python /usr/bin/cinder-volume --config-file /etc/cinder/cinder.conf
cinder    4531  4526  0 Jan25  ?      00:15:24 /usr/bin/python /usr/bin/cinder-volume --config-file /etc/cinder/cinder.conf
```

5.8.4 卷（Volume）操作

对于卷，可以进行下列各种操作。

- Create Volume
- Delete Volume
- Attach Volume
- Detach Volume
- Extend Volume
- Create Snapshot
- Delete Snapshot

- List Snapshot
- Create Volume from Snapshot
- Create Volume from Image
- Create Volume from Volume (Clone)
- Create Image from Volume
- Volume Migration (host assisted)

图 5-21 列举了社区中已经提供的驱动和驱动支持的功能。

Driver	Create Volume	Delete Volume	Attach Volume	Detach Volume	Extend Volume	Create Snapshot	Delete Snapshot	List Snapshots	Create Volume from Snapshot	Create Volume from Image	Create Volume from Volume (Clone)	Create Image from Volume	Volume Migration (host assisted)	Supported Protocols
Coraid	Grizzly	Grizzly	Grizzly	Grizzly	Havana	Grizzly	Grizzly	Grizzly	Grizzly	Havana	Havana	Havana	Havana	AOE (Grizzly)
Dell EqualLogic	Havana	Havana	Havana	Havana		Havana	Havana	Havana	Havana	Havana	Havana	Havana	Havana	ISCSI (Havana)
EMC VMAX/VNX	Grizzly	Grizzly	Grizzly	Grizzly		Grizzly	Grizzly	Grizzly	Grizzly	Grizzly	Grizzly	Grizzly	Havana	ISCSI (Grizzly)
GlusterFS	Grizzly	Grizzly	Grizzly	Grizzly	Havana	Havana	Havana	Havana	Havana	Havana	Havana	Havana	Havana	GlusterFS (Grizzly)
HDS (HUS)	Havana	Havana	Havana	Havana	Havana	Havana	Havana	Havana	Havana	Havana	Havana	Havana	Havana	ISCSI (Havana)
HP 3PAR (StoreServ)	Grizzly	Grizzly	Grizzly	Grizzly	Havana	Grizzly	Grizzly	Grizzly	Grizzly	Grizzly (Havana/FC)	Grizzly	Grizzly (Havana/FC)	Havana	ISCSI (Grizzly), FC (Grizzly)
HP Lethand (StoreVirtual)	Diablo	Diablo	Diablo	Diablo	Havana	Havana	Havana	Havana	Havana	Grizzly		Grizzly	Havana	ISCSI (Diablo)
Huawei T Dorado	Grizzly	Grizzly	Grizzly	Grizzly		Grizzly	Grizzly	Grizzly	Grizzly	Grizzly	Havana	Grizzly	Havana	ISCSI (Grizzly), FC (Havana)
Huawei HVS	Havana	Havana	Havana	Havana		Havana	Havana	Havana	Havana	Havana	Havana	Havana	Havana	ISCSI (Havana), FC (Havana)
IBM GPFS	Havana	Havana	Havana	Havana		Havana	Havana	Havana	Havana	Havana	Havana	Havana	Havana	GPFS NSD
IBM Storwize family/SVC	Folsom	Folsom	Folsom	Folsom	Havana	Folsom	Folsom	Folsom	Folsom	Havana	Grizzly	Grizzly	Havana	ISCSI (Folsom), FC (Grizzly)
IBM XIV	Folsom	Folsom	Folsom	Folsom	Havana	Folsom	Folsom	Folsom	Folsom	Grizzly	Havana	Havana	Havana	ISCSI (Folsom), FC (Grizzly)
IBM DS8000	Havana	Havana	Havana	Havana	Havana	Havana	Havana	Havana	Folsom	Havana	Havana	Havana	Havana	FC (Havana)
LVM (Reference)	Folsom	Folsom	Folsom	Folsom	Havana	Folsom	Folsom	Folsom	Folsom	Grizzly	Grizzly	Grizzly	Havana	ISCSI (Folsom)
NetApp	Essex	Essex	Essex	Essex	Havana	Folsom	Folsom	Essex	Essex	Grizzly	Grizzly	Grizzly	Havana	ISCSI (Folsom), NFS (Folsom)
Nexenta	Essex	Essex	Essex	Essex	Havana	Essex	Essex	Essex	Essex	Havana	Havana	Grizzly	Havana	ISCSI (Essex), NFS (Havana)
NFS (Reference)										Folsom		Havana	Havana	NFS (Folsom)
RBD (Ceph)	Cactus	Cactus	Cactus	Cactus	Havana	Diablo	Diablo	Diablo	Folsom	Grizzly	Havana	Grizzly		RBD (Cactus)
Scalify	Grizzly	Grizzly	Grizzly	Grizzly		Grizzly	Grizzly	Grizzly	Grizzly	Grizzly	Havana	Grizzly		Scalify (Grizzly)

图 5-21 Cinder 存储后端支持一览（资源：<https://wiki.openstack.org/wiki/CinderSupportMatrix>）

块存储服务的后端，也可以是一些分布式/集群/并行文件系统。下面将对这类文件系统进行一些简单介绍

1. 分布式/集群/并行文件系统

(1) 分布式文件系统

自然地，“分布式”是重点，它是相对于本地文件系统而言的。分布式文件系统通常指 C/S 架构或网络文件系统，用户数据没有直接连接到本地主机，而是存储在远程存储服务上。NFS/CIFS 是最为常见的分布式文件系统，这就是我们说的 NAS 系统。分布式文件

系统中, 存储服务器的节点数可以是一个(如传统 NAS), 也可以是多个(如集群 NAS)。对于单个节点的分布式文件系统来说, 存在单点故障和性能瓶颈问题。除了 NAS 以外, 典型的分布式文件系统还有 AFS(Andrew 文件系统), 以及集群文件系统(如 Lustre、GlusterFS、PVFS2 等)。

(2) 集群文件系统

“集群”主要分为高性能集群(High Performance Cluster, HPC)、高可用集群(High Availability Cluster, HAC)和负载均衡集群(Load Balancing Cluster, LBC)。集群文件系统是指协同多个节点提供高性能、高可用或负载均衡的文件系统, 它是分布式文件系统的子集, 消除了单点故障和性能瓶颈问题。对于客户端来说集群是透明的, 它看到的是一个单一的全局命名空间, 用户文件访问请求被分散到所有集群上进行处理。此外, 可扩展性(包括 Scale-Up 和 Scale-Out)、可靠性、易管理性等也是集群文件系统追求的目标。在元数据管理方面, 可以采用专用的服务器, 也可以采用服务器集群, 或者采用完全对等分布的无专用元数据服务器架构。目前典型的集群文件系统有 SONAS、ISILON、IBRIX、NetAPP-GX、Lustre、PVFS2、GlusterFS、Google File System、LoongStore、CZSS 等。

(3) 并行文件系统

并行文件系统能够支持并行应用, 比如 MPI(多点接口)。在并行文件系统环境下, 所有客户端可以在同一时间并发读写同一个文件。并发读, 大部分文件系统都能够实现。并发写实现起来则要复杂许多, 既要保证数据一致性, 又要最大限度提高并行性, 因此在锁机制方面需要特别设计, 如细粒度的字节锁。通常 SAN 共享文件系统都是并行文件系统, 如 GPFS、StorNext、GFS、BWFS, 集群文件系统大多数也是并行文件系统, 如 Lustre、Panasas 等。

(4) 如何区分各种文件系统

区分以上三种文件系统的重点是“分布式”、“集群”、“并行”三个前缀关键字。简单来说, 非本地直连的、通过网络连接的, 为分布式文件系统; 分布式文件系统中, 服务器节点由多个组成的, 为集群文件系统; 支持并行应用(如 MPI)的, 为并行文件系统。在上面所举的例子中也可以看出, 这三个概念之间具有重叠之处, 比如 Lustre, 它既是分布式文件系统, 也是集群和并行文件系统。但是, 它们也有不同之处。集群文件系统是分布式文件系统, 但反之则不成立, 如 NAS、AFS。SAN 文件系统是并行文件系统, 但可能不是集群文件系统, 如 StorNext。GFS、HDFS 之类, 它们是集群文件系统, 但可能不是并行文件系统。

5.9 Neutron

通常来说，IaaS 云通过虚拟化技术，可以租赁计算资源，包括计算、网络、存储等给各种租户。而网络虚拟化可以提供从二层到七层网络服务给客户，就和传统网络一样。

Neutron 提供了虚拟机与计算之间的软件定义网络功能。它也引入了插件（plug-in），一种 OpenStack 网络定义 API 的后端实现。一个插件可以使用各种技术，来实现逻辑 API 请求创建或操作各种软件定义的网络设备。一些插件可能使用基本的 Linux VLANs 和 IP tables，另外一些插件可能使用了更高级的技术，例如 L2-in-L3 隧道或者 OpenFlow。

目前提供了插件的网络代理如图 5-22 所示。

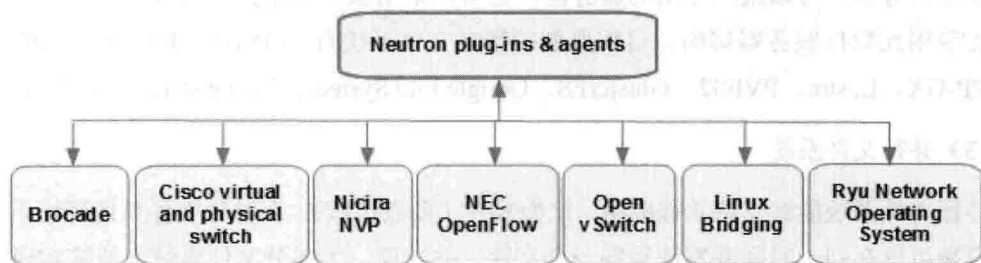


图 5-22 提供了插件的网络代理

5.9.1 nova-network 的局限性

- (1) L2 只提供了一种插件：Linux Bridge。
- (2) L3 通过 Linux 内核的 IPv4 转发功能，提供了静态路由功能，通过 dnsmasp 提供了 DHCP 功能。
- (3) 没有提供 L4~L7 层功能，但 nova-compute 可以支持安全组功能。
- (4) 不但简单，而且可以水平扩展，提供高可用能力。

5.9.2 Neutron 功能特点

(1) L2 支持更多的插件, 包括 Linux Bridge、OpenvSwitch、Hyper-V, 以及一些特定厂商的插件。

(2) L2 支持隧道技术, 除了 nova-network 支持的扁平化和 VLAN 网络之外, 还包括 GRE、VXLAN (虚拟可扩展 LAN)。

(3) L3 通过 Linux 内核的 IPv4 转发功能, 提供了静态路由功能, 通过 dnsmasq 提供了 DHCP 功能; 但 L3 代理依然不支持多主机功能。

(4) L4~L7 是一个服务框架 (但服务链还没有实现), 现在支持负载均衡即服务 (LbaaS)、防火墙即服务 (FwaaS)、IPSec VPNaaS、计费服务、l3_router 服务等。

(5) 支持同时使用不同的插件和不同的拓扑。

5.9.3 ML2

通常 Neutron 中的一个网络服务包括两部分: 一个是插件, 主要用来更新数据库, 另一个是代理, 只习惯真正的网络服务操作。ML2 (Modular Level 2) 是一个通过万能地支持所有网络服务插件来减少代码冗余的插件。

除了减少代码冗余, 更强大的是, ML2 可以让当前所有插件 (openvswitch, linuxbridge 和 hyperv L2 代理等) 和所有网络拓扑 (扁平网络、vlan、gre、vxlan) 一起同时工作。

图 5-23 为使用 OpenvSwitch 时的进程和每个进程的功能。

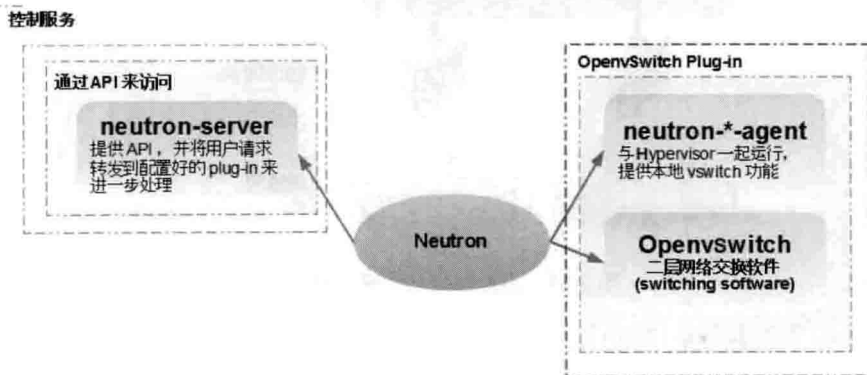


图 5-23 Neutron 与 OpenvSwitch

图 5-24 为三层及以上提供的服务与功能。而图 5-25 说明了虚拟网络与主机物理网络设备的关系。

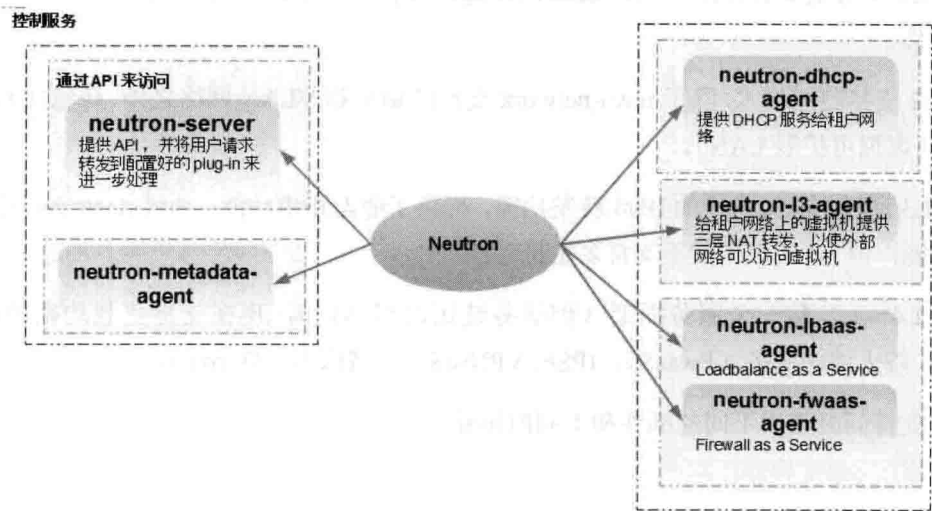


图 5-24 Neutron 三层及以上提供的服务与功能

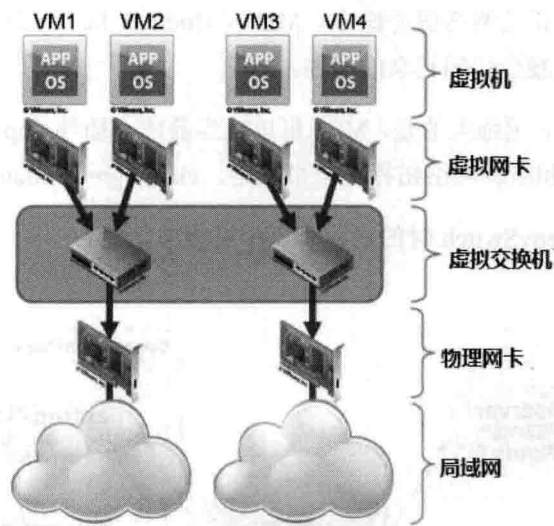


图 5-25 虚拟网络与物理网络

5.9.4 Open vSwitch 虚拟网络

下面将以 OpenvSwitch 为例，来说明软件定义网络是如何工作的。

租户 A 和租户 B 各自有一个子网，它们都通过一个路由器连接到公网上去。网络如图 5-26 所示。

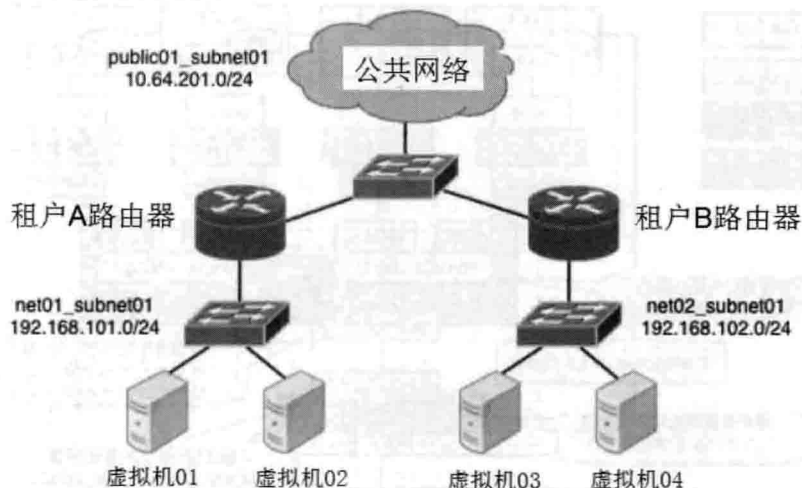


图 5-26 两个租户与两个 VLAN

1. 计算节点内的基于 Open vSwitch 的虚拟网络

图 5-27 是官方文档的一个图，非常清楚地说明了各层网桥和相互之间的关系。本书在其中增加了对各种设备类型的说明。这张图说明了在多租户环境下，计算节点内部的虚拟网络及其关系。在 OpenStack 网络里，有四种截然不同的虚拟网络设备：TAP 设备、veth 对、Linux 网桥和 Open vSwitch 网桥。对于一个以太网数据包，若要从虚拟机 vm01 的网口 eth0 到物理网络去，它必须通过主机内部 9 个虚拟网络设备：TAP vnet0、Linux 网桥 qbrnnn、veth 对 (qvbnnn、qvonnn，这里 nnn 是创建网络端口时的 UUID 号，对应于图 5-27 的 XXX、YYY 等)，Open vSwitch 网桥 br-int、veth 对 (intbr-eth1、phy-br-eth1)，最后是物理网口 eth1。

- **TAP 设备：**vnet0 是虚拟机管理程序（如 KVM 和 Xen）实现的虚拟网络网卡（通常叫作 VIF 或者 vNIC）。发送到 TAP 的以太网数据包由虚拟机的操作系统接收到。
- **veth 对：**直接相连的一对虚拟网络接口。发送到 veth 对一端的以太网数据包由 veth 对的另一端接收。在 OpenStack 网络里，使用 veth 对来连接两个虚拟网桥。
- **Linux 网桥：**就像一个传统的 hub，可以连接多个网络接口设备（虚拟的或者物理的）

到这个 Linux 网桥，任何以太网数据包，如果发送到这个网桥的一个接口，就会被转发到该网桥上的所有其他接口。

- **Open vSwitch 网桥：**虚拟网桥，网络接口设备连接到 vSwitch 网桥的端口。这些端口可以像物理交换机的端口一样进行配置，包括配置 VLAN。

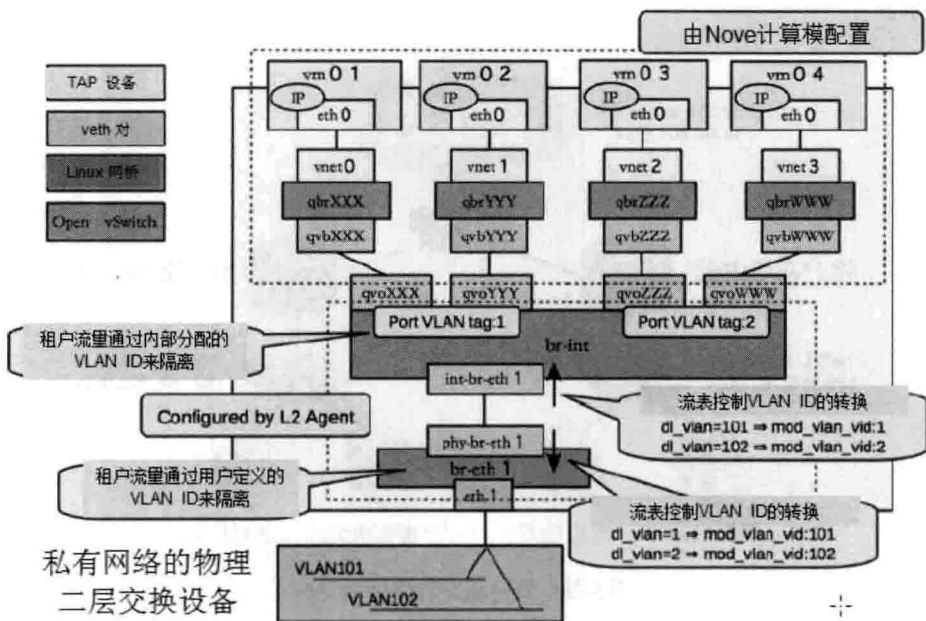


图 5-27 多租户的计算节点虚拟网络

在计算节点，需要创建一个 Open vSwitch 集成网桥（br-int），另一个 Open vSwitch 网桥连接到数据网络。这两个网桥由一对虚拟端口相连，neutron-openvswitch-plugin-agent 配置这两个网口来做 VLAN 的转换。

（1）集成网桥（br-int）

所有运行于计算节点的虚拟机连接到这个网桥。OpenStack 网络通过配置在 br-int 上的端口来实现虚拟机的隔离。

（2）物理连接网桥

br-eth1 网桥通过 eth1 来连接物理网卡。它通过 veth 对（int-br-eth1、phy-br-eth1）连接到集成网桥。

(3) VLAN 转换

在上述网络的例子中，net01 和 net02 在内部的 VLAN 号分别是 1 和 2，然而物理网络只支持 VLAN IDs 范围为 101~110。Open vSwitch 代理负责配置 br-int 和 br-eth1 上的流规则来做 VLAN 转换。当 br-eth1 上的端口 phy-br-eth1 收到一个标记为 VLAN ID 1 的数据包时，它修改包中的 VLAN ID 为 101。相似地，当 br-int 上的端口 int-br-eth1 收到 VLAN 标记为 101 的数据包时，它将 VLAN ID 修改为 1。

(4) 安全组：iptables 和 Linux 网桥

理想的情况是，TAP 设备和 vnet0 应该直接连到网桥 br-int，但这因为当前 OpenStack 安全组的实现而变得不可能。OpenStack 在 TAP 设备如 vnet0 上利用 iptables 规则来实现安全组规则，但是 Open vSwitch 与连接到 Open vSwitch 端口的 TAP 设备的 iptables 规则不兼容。所以 OpenStack 网络使用了一个额外的 Linux 网桥和一个 veth 对来作为应对的办法。

那个物理网络用于什么目的，需要在配置文件中说明。下面是在 ovs_neutron_plugin.ini 文件中，针对 open vswitch 和 VLAN 这种网络模式的配置，physnet2 为数据网络使用到的物理网络。

```
[ovs]
tenant_network_type = vlan
network_vlan_ranges = physnet2:101:110
integration_bridge = br-int
bridge_mappings =physnet2:br-eth1
```

2. 网络节点内的基于 Open vSwitch 的虚拟网络

图 5-28 来自于官方文档，非常清楚地说明了网络节点内部的虚拟网络及其连接关系。在网络节点，假定 eth0 网卡连接到外部网络，而 eth1 连接到数据网络，则需要在 ovs_neutron_plugin.ini 文件中做如下配置。在下例中，使用的是 VLAN 网络类型，可用的 VLAN 范围为 101~110，创建了一个集成网桥 br-int，而网桥 br-ex 和 br-eth1 用来连接物理网口。具体如下：

```
[ovs]
tenant_network_type = vlan
network_vlan_ranges = physnet2:101:110
integration_bridge = br-int
bridge_mappings = physnet1:br-ex,physnet2:br-eth1
```

在网络节点，一个 Open vSwitch 网桥（br-ex）连接到物理网口，并通过该网口（eth0）连接到外部网络。集成网桥和这个网桥也通过两个虚拟端口相连，利用三层交换来将网络

数据包从内部网络路由到外部网络去。

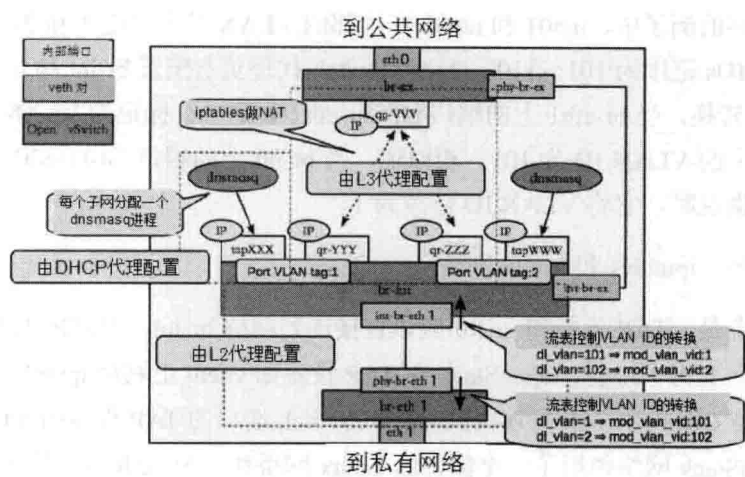


图 5-28 多租户的网络节点虚拟网络及其连接关系

(1) Open vSwitch 内部端口

网桥上有多个内部端口，这些内部端口可以分配一个到多个 IP 地址到 Open vSwitch 网桥。在上面的例子中，网桥 br-int 有四个内部端口：tapXXX、qr-YYY、qr-ZZZ、tapWWW。每个内部端口都有一个单独的 IP 地址。另有一个内部端口 qq-VVV 在 br-ex 网桥上。

默认地，OpenStack 网络 DHCP 代理运行一个叫作 dnsmasq 的程序来提供 DHCP 服务给虚拟机。OpenStack 网络必须在每个需要 DHCP 服务的网络上创建一个内部端口，并把一个 dnsmasq 进程和这个端口管理起来。

(2) L3 代理（路由）

OpenStack 网络通过使用 Open vSwitch 内部端口实现 L3 路由功能。只要系统打开了 IP 转发功能，因为这些端口对于网络节点操作系统都是可见的，它将把数据包通过各个网络接口路由到合适的网络去。L3 代理将使用 iptables 来实现浮动 IP 到其他网络的转化（NAT）。

5.10 Ceilometer

Ceilometer 可以用来跟踪 OpenStack 集群，首要目的是为了计费，但是整个框架非常通

用，可以扩展到其他用途，如资源监控、性能报告、问题调试、容量分析、智能调度、优化等。

计费通常包括以下三步。

- (1) 收集运行数据。
- (2) 将运行数据转换为可以计费的条目，并计算费用。
- (3) 生成计费表单，收集付费。

数据收集的过程通常如下。

- (1) 从 OpenStack 组件收集计量数据。
- (2) 如果需要，将一种计量数据转换成另外一种。
- (3) 发布计量数据到任何目的端（包括 Ceilometer 本身）。
- (4) 保存收到的计量数据到存储。
- (5) 通过 Ceilometer REST API 来读取数据。

到 OpenStack Havana 版本为止，OpenStack 已提供了大量的计量内容，分别由表 5-19 到 5-26 来说明。

5.10.1 计算（Nova）

表 5-19 列举了到 Havana 版本为止，Ceilometer 实现的针对计算方面的计量类型。

表 5-19 计算里的计量

名 称	单 位	说 明
Instance	个数	创建了总共多少个虚拟机
Instance:<type>	个数	创建了总共多少虚拟机<类型>
内存	MB	以 MB 为单位的内存使用容量
CPU	ns	使用到的 CPU 时间
cpu_util	%	平均 CPU 使用率
vcpus	vcpu	vCPU 的个数
disk.read.requests	请求	读请求的个数

续表

名 称	单 位	说 明
disk.read.requests.rate	请求/秒	平均每秒的读请求个数
disk.write.requests	请求	写请求的个数
disk.write.requests.rate	请求/秒	平均每秒的写请求个数
disk.read.bytes	字节	以字节为单位的读的容量
disk.read.bytes.rate	字节/秒	每秒平均的读的字节数
disk.write.bytes	字节	以字节为单位的写的容量
disk.write.bytes.rate	字节/秒	每秒平均的写的字节数
disk.root.size	GB	根分区的大小，以 GB 为单位
disk.ephemeral.size	GB	非持久性存储的大小，以 GB 为单位
network.incoming.bytes	字节	在单个虚拟机网络虚拟网口的进入字节数
network.incoming.bytes.rate	字节/秒	在单个虚拟机网络虚拟网口以秒为单位的平均流入字节数
network.outgoing.bytes	字节	在单个虚拟机网络虚拟网口的出去字节数
network.outgoing.bytes.rate	字节/秒	在单个虚拟机网络虚拟网口以秒为单位的平均流出字节数
network.incoming.packets	包	在单个虚拟机网络虚拟网口流入的数据包个数
network.incoming.packets.rate	包/秒	在单个虚拟机网络虚拟网口以秒为单位的平均流入数据包个数
network.outgoing.packets	包	在单个虚拟机网络虚拟网口流出的数据包个数
network.outgoing.packets.rate	包/秒	在单个虚拟机网络虚拟网口以秒为单位的平均流出数据包个数

到目前为止，这些计量只实现了以 Libvirt 为前端的虚拟化管理程序，其他类型的虚拟化管理程序还在开发中。

5.10.2 网络 (Neutron)

表 5-20 列举了到 Havana 版本为止，Ceilometer 实现的网络方面的计量类型。

表 5-20 网络的计量

名 称	单 位	说 明
network	个数	网络个数
network.create	个数	这个网络创建的请求数
network.update	个数	这个网络更新的请求数
subnet	个数	子网个数
subnet.create	个数	子网创建的请求数
subnet.update	个数	子网更新的请求数
port	个数	端口数量
port.create	个数	端口创建的请求数
port.update	个数	端口更新的请求数
router	个数	路由器个数
router.create	个数	路由器创建的请求数
router.update	个数	路由器更新的请求数
ip.floating	个数	浮动 IP 的个数
ip.floating.create	个数	浮动 IP 创建的请求数
ip.floating.update	个数	浮动 IP 更新的请求数

5.10.3 镜像（Glance）

表 5-21 列举了到 Havana 版本为止，Ceilometer 实现的镜像管理方面的计量类型。

表 5-21 镜像服务的计量

名 称	单 位	说 明
image	个数	镜像个数
Image.size	字节	上传的镜像大小
Image.update	个数	更新的镜像个数
Image.upload	个数	上传的镜像个数
Image.delete	个数	删除的镜像个数
Image.download	字节	镜像下载的数量
Image.serve	字节	用于创建虚拟机总共使用的字节数

5.10.4 块存储 (Cinder)

表 5-22 列举了到 Havana 版本为止, Ceilometer 实现的块存储管理方面的可计量类型。

表 5-22 块存储的计量

名 称	单 位	说 明
volume	个数	卷的个数
volume size	GB	卷的大小

5.10.5 对象存储 (Swift)

表 5-23 列举了到 Havana 版本为止, Ceilometer 实现的对象存储管理方面的可计量类型。

表 5-23 对象存储的计量

名 称	单 位	说 明
storage.objects	个数	对象的个数
storage.objects.size	字节	全部对象的大小
storage.objects.containers	个数	容器的个数
storage.objects.incoming.bytes	字节	流入字节数
storage.objects.outgoing.bytes	字节	流出字节数
storage.api.request	个数	API 请求的个数
storage.containers.objects	个数	容器里对象的个数
storage.containers.objects.size	字节	容器里保存的对象的总共大小

5.10.6 编排 (Heat)

表 5-24 列举了到 Havana 版本为止, Ceilometer 实现的编排方面的可计量类型。

表 5-24 编排服务的计量

名 称	单 位	说 明
stack.create	个数	成功创建的编排的个数
stack.update	个数	成功更新的编排的个数

续表

名 称	单 位	说 明
stack.delete	个数	成功删除的编排的个数
stack.resume	个数	成功恢复的编排的个数
stack.suspend	个数	成功挂起的编排的个数

5.10.7 能源 (Kwapi)

表 5-25 列举了到 Havana 版本为止, Ceilometer 实现的能源管理方面的可计量类型。

表 5-25 能源服务的计量

名 称	单 位	说 明
energy	kWH	消耗能源的总量
power	W	电源消耗

5.10.8 网络 (SDN 控制器)

表 5-26 列举了到 Havana 版本为止, Ceilometer 实现的网络 SDN 控制器方面的可计量类型。

表 5-26 SDN 控制器的计量

名 称	单 位	说 明
switch	个数	交换机个数
switch.port	个数	端口个数
switch.port.receive.packets	个数	收到的数据包个数
switch.port.transmit.packets	个数	发送的数据包个数
switch.port.receive.bytes	字节	收到字节数
switch.port.transmit.bytes	字节	发送字节数
switch.port.receive.drops	个数	接收丢弃的包个数
switch.port.transmit.drops	个数	发送丢弃的包个数
switch.port.receive.errors	个数	接收错误数据包个数
switch.port.transmit.errors	个数	发送错误数据包个数

续表

名 称	单 位	说 明
switch.port.receive.frame_error	个数	接收到的数据包对齐错误个数
switch.port.receive.overrun_error	个数	接收溢出个数
switch.port.receive.crc_error	个数	接收到的循环冗余错误包个数
switch.port.collision.count	计数	冲突次数
switch.table	个数	表的个数
switch.table.active.entries	个数	活跃的表单个数
switch.table.lookup.packets	个数	查询的数据包个数
switch.table.matched.packets	个数	匹配的数据包个数
switch.flow	个数	流表个数
switch.flow.duration.seconds	秒	持续时间
switch.flow.duration.nanoseconds	纳秒	持续时间
switch.flow.packets	个数	收到的数据包个数
switch.flow.bytes	字节	收到的字节数

OpenStack Ceilometer 包括的组件如图 5-29 所示。需要注意的是,不同的组件需要安装在相应的节点上。

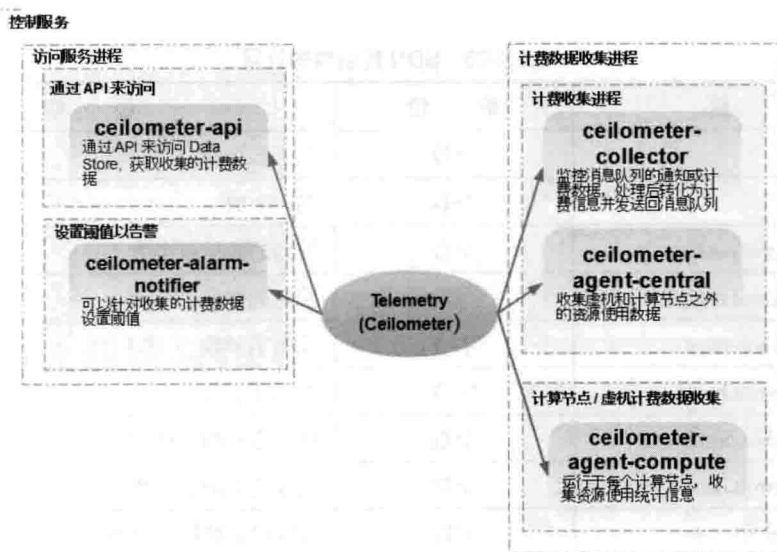


图 5-29 计量服务的组成

5.10.9 计量数据收集的结构、交互图

图 5-30 说明了计量各个组件及其相互之间的交互关系。

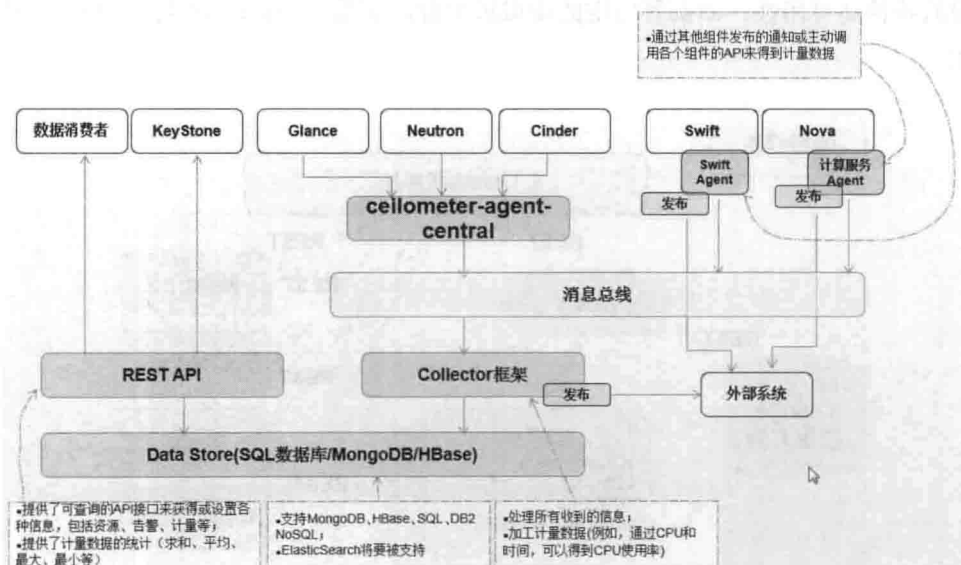


图 5-30 计量服务的架构

运行时的进程如下：

```

[root@CloudController rc2.d]# ps -ef | grep cellometer
root      1733      784   0 22:31 pts/0    00:00:00 grep cellometer
root      2186      1   0 Jan25  7 00:11:24 /usr/bin/python /usr/bin/cellometer-api --config-file /etc/cellometer/cellometer.conf --
root      2196      1   0 Jan25  7 00:02:31 /usr/bin/python /usr/bin/cellometer-agent-central --config-file /etc/cellometer/cellome
root      2204      1   0 Jan25  7 00:05:16 /usr/bin/python /usr/bin/cellometer-collector --config-file /etc/cellometer/cellometer-
root      2212      1   0 Jan25  7 00:01:57 /usr/bin/python /usr/bin/cellometer-agent-compute --config-file /etc/cellometer/cellome

```

5.11 Heat

Heat（编排服务）可以提供如各种应用的高可用配置、负载均衡和自动线性扩展等功能，是一个非常强大而有用的工具。图 5-31 说明了 Heat 与其他各个项目之间的交互关系。

Heat 开始于 2012 年 3 月，在 Grizzly 版本时处于孵化阶段，而在 Havana 版本中变成一个正式和单独的项目。它可以和 OpenStack 的核心项目交互，编排虚拟机的部署，提供类似于 AWS Cloudformation 功能和一些 REST API，并且和 AWS Cloudformation 兼容。Heat 将服务的配置抽象成一个模板，然后将这个 JSON 格式的模板转换成一个云端应用。另外，

模板还可以嵌入其他模板，提供了非常好的模块性。部署的过程是完全可以重复的和自动化的。另外它还可以配置虚拟机实例，使其可以根据负载而自动扩展，比如，虚拟机集群在节假日自动扩大服务的虚拟机集群，在流量降低时，自动减少虚拟机集群的规模；还包括应用的各种高可用性，即部署应用的虚拟机集群，前端有负载均衡器；还有服务的监控的功能。

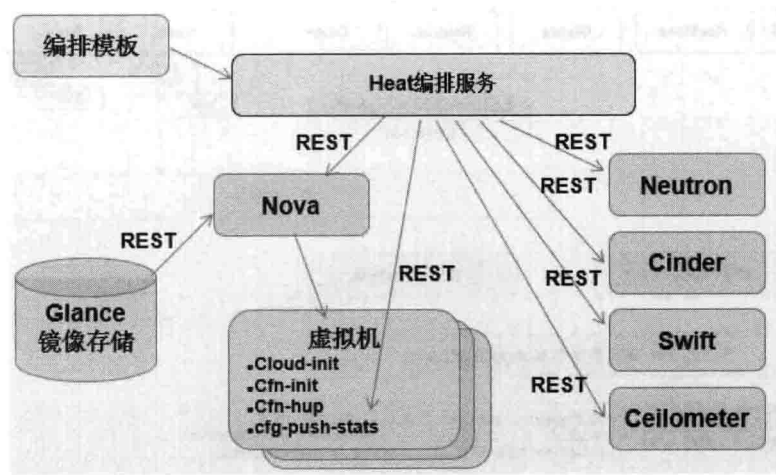


图 5-31 Heat 交互逻辑

Heat 包括如图 5-32 所示的一些进程，这些服务都属于控制服务，并且运行于控制节点。

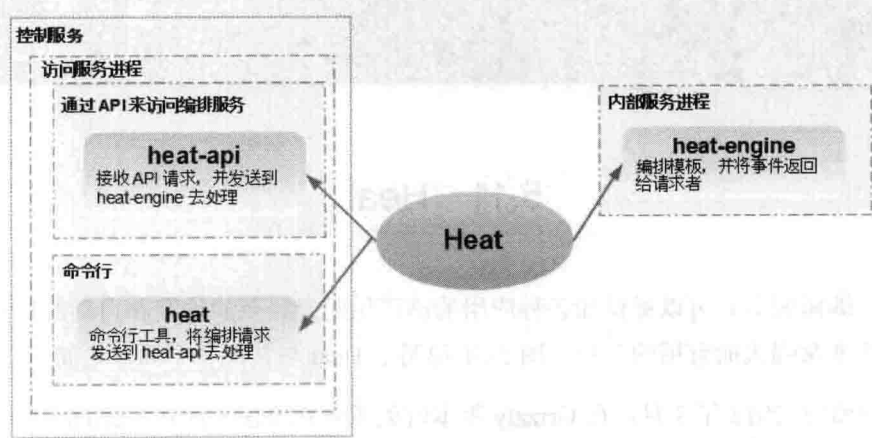


图 5-32 Heat 的组成

第 6 章

OpenStack 部分组件安装示例

本章将演示如何手工安装一些 OpenStack 组件。

完整搭建一个多节点，满足企业生产环境可用的 OpenStack 环境，需要很多艰苦的工作，包括决定要使用的操作系统、获取稳定的 OpenStack 安装包、设计安装拓扑、规划 IP 地址、配置物理交换机、配置存储设备、安装操作系统、安装 OpenStack 服务包、安装第三方软件/工具、配置系统、配置网络、配置存储等。

一个系统化的、可重复的过程和工具对云平台部署和管理至关重要。

在下列的步骤中，假设用户已经收集到需要的安装包，并放在本地目录，并且物理机的操作系统已经安装好，使用的是 CentOS 6.4。

6.1 安装拓扑

在本例中，示范的是安装一个控制节点和一个计算节点。然而，同样的步骤可以用来安装多个计算节点。图 6-1 是本例的安装拓扑、组件选择和一些其他功能说明。

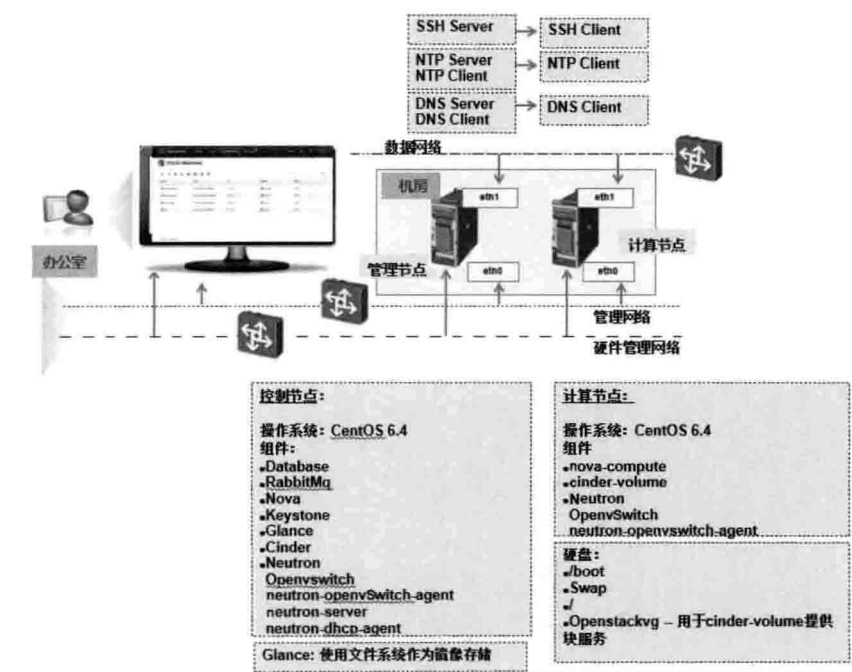


图 6-1 示例安装拓扑与配置

另外限于篇幅，下列服务的安装没有包括在内，可以参考相应的文档或网上资料来解决。

- Nova
- Cinder
- Heat
- Ceilometer
- Dashboard
- DNS & NTP

在安装 OpenStack 服务时，一定要保证 DNS（域名系统）和 NTP（网络时间协议）服务已经工作。分布式服务严重依赖于这些服务。

6.2 服务器远程安装配置

数据中心，通常有几十、几百到数千台服务器。往往数据中心管理员需要对硬件或 BIOS 进行一些设置。传统的方法是管理员需要到机房里去，先刷卡或者按指纹，通过机房的安全认证之后，进入机房，在机器的轰鸣声中，寻找自己需要管理的服务器，在 console 上进行操作。数据中心服务器通常是没有显示器、键盘、鼠标的。管理员需要找到空闲的这些设备，连接到要管理的服务器上，然后执行管理操作。

现在的服务器通常都提供几种方式，让管理员可以在办公室里远程管理这些服务器，即使是像 console 这样的操作，也可以远程，或者通过浏览器，或者通过网络用命令行来执行。

当然，远程管理并不能解决所有问题，比如，网线松动或者硬件故障，还必须要到机房去解决。

如图 6-2 所示为通过浏览器访问远程服务器的 BIOS 界面，在这里可以管理硬件设备、禁止或打开某个设备、修改参数、固件升级等。

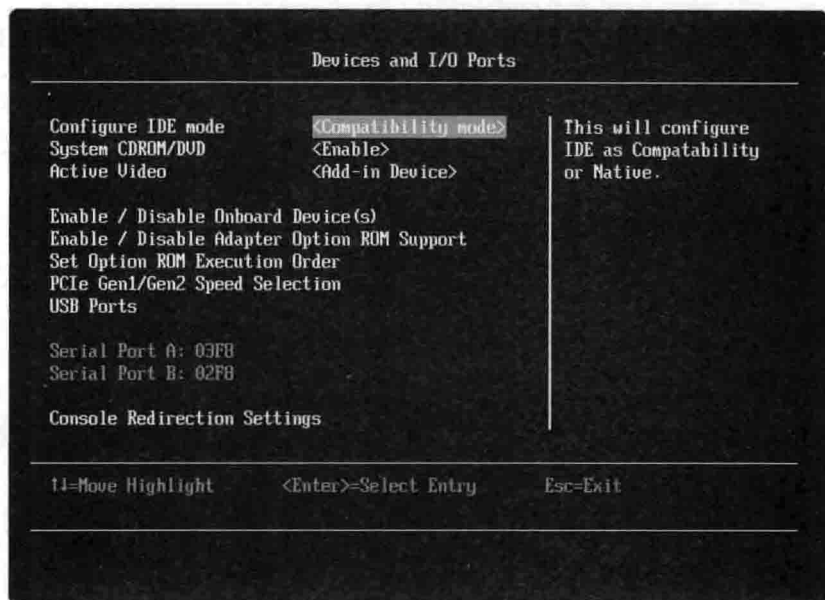


图 6-2 BIOS 界面

在系统安装后，正常情况下服务器会设置成从硬盘启动。然而我们可以修改这种启动顺序，让它从光盘（CD/DVD）启动，或者通过网络来启动。另外许多服务器还支持从 USB 启动，从共享的外部存储启动。灵活的启动方式，也是各个服务器的特色之一。图 6-3 展示了可以远程修改系统的启动顺序。

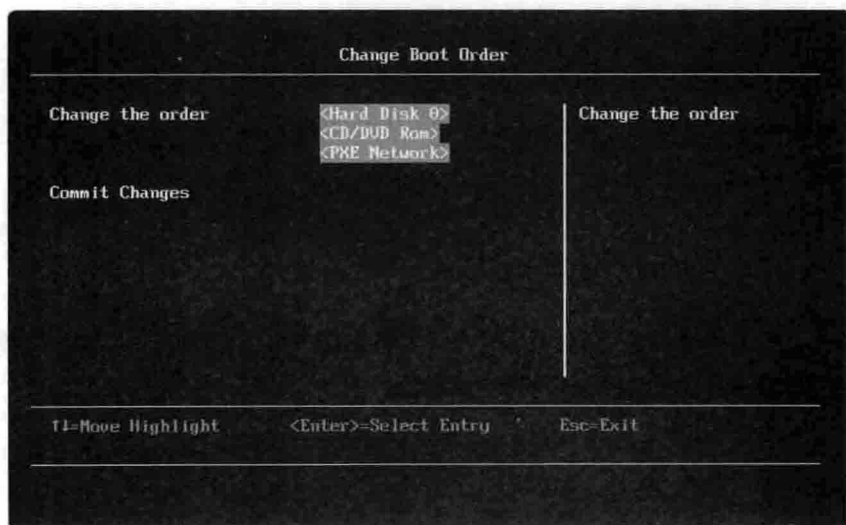


图 6-3 在 BIOS 里修改启动顺序

如果选择从 CD/DVD 启动，我们需要光驱来放入启动媒介。在远程的方式下，我们也可以打开远程媒介（Remote Media）窗口，然后选择要安装的 ISO 文件，挂载。其结果就和在光驱里放入一张物理光盘一样。图 6-4 是一个远程挂载 ISO 操作系统来安装主机的界面。除了传统的光驱、USB，现在可以远程挂载本地的一个安装媒介，如 ISO 来安装系统，再也不需要到轰鸣的机房里去做这些了。

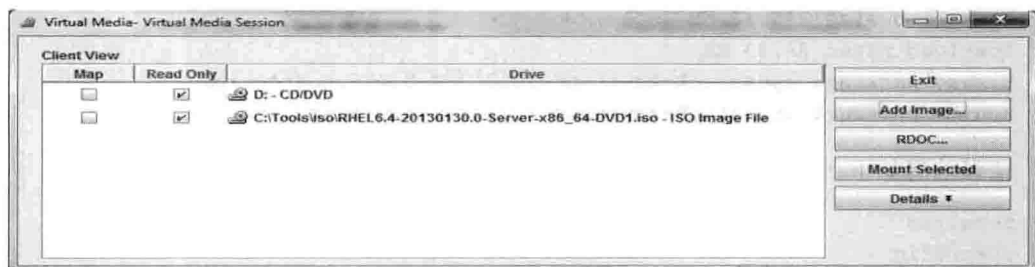


图 6-4 挂载虚拟安装介质

在此之后，只要选择让服务器从光驱启动，则服务器会自动从本地文件系统读取 ISO 文件，开始安装过程。

6.3 软件包与仓库

6.3.1 软件包

Fedora、Red Hat 和 CentOS 等使用到的软件和文档以一种称为 RPM 软件包文件的方式提供。每个软件包是一个压缩的文档，包含了内容信息、应用程序文件、图标、文档和管理用的脚本。管理程序利用这些内容来安全地定位、安装和卸载软件。例如，Fedora 安装过程使用随 Fedora Core 附带的软件包来构建或升级符合你所需要的系统。

软件包也包含一个数字签名，以验证它们的来源。软件管理工具通过 GPG 公钥来验证这个签名。yum 和 RPM 工具共享同一个 keyring，它保存了所有有保障的软件包来源的公钥。

下面为一个 RPM 包的信息：

```
openstack-keystone-2012.2.4-5.el6.noarch.rpm
```

openstack-keystone - OpenStack Identity Service

Distribution: CentOS 6
Repository: EPEL i386
Package name: openstack-keystone
Package version: 2012.2.4
Package architecture: noarch
Package type: rpm
Installed size: 45,12 KB
Download size: 28,53 KB
Binary package: openstack-keystone-2012.2.4-5.el6.noarch.rpm
Source package: openstack-keystone-2012.2.4-5.el6.src.rpm

依赖信息如下:

Requires
chkconfig
chkconfig
initscripts
python-keystone = 2012.2.4-5.el6
python-keystoneclient >= 2012.1-0.4.e4
shadow-utils

包中的文件列表按照安装后的目录形式呈现:

Files
/etc/keystone/
/etc/keystone/default_catalog.templates
/etc/keystone/keystone.conf
/etc/keystone/logging.conf
/etc/keystone/policy.json
/etc/logrotate.d/openstack-keystone
/etc/rc.d/init.d/openstack-keystone
/usr/bin/keystone-all
/usr/bin/keystone-manage
/usr/bin/openstack-keystone-sample-data
/usr/share/doc/openstack-keystone-2012.2.4/
/usr/share/doc/openstack-keystone-2012.2.4/LICENSE
/usr/share/doc/openstack-keystone-2012.2.4/README.rst
/usr/share/man/man1/keystone-all.1.gz
/usr/share/man/man1/keystone-manage.1.gz
/usr/share/openstack-keystone/
/usr/share/openstack-keystone/openstack-keystone.upstart
/usr/share/openstack-keystone/sample_data.sh
/var/lib/keystone/
/var/log/keystone/
/var/run/keystone/

6.3.2 软件仓库

仓库 (Repository) 是一个准备好的目录或是一个网站, 包含了软件包和索引文件。软件管理工具如 yum, 可以在仓库中自动地定位并获取正确的 RPM 软件包。这样, 用户就不必手动搜索和安装新应用程序和升级补丁了。只需一个命令, 用户就可以更新系统中所有软件, 也可以根据指定的搜索目标来查找安装新软件。

有一系列的服务器, 为每个版本的 Fedora/Red Hat/CentOS 分别提供了一些仓库。

对于 Red Hat, 只有购买了 license 的用户并注册后, 才可以从官方仓库下载新的软件包。

下面是一个配置了 yum 源的文件: vi /etc/yum.repos.d/rhel-debuginfo.repo。

```
[base]
name=CentOS-$releasever - Base
baseurl=http://mirrors.163.com/centos/6.0/os/$basearch/
gpgcheck=1
gpgkey=http://mirrors.163.com/centos/RPM-GPG-KEY-CentOS-6

#released updates
[updates]
name=CentOS-$releasever - Updates
baseurl=http://mirrors.163.com/centos/6.0/updates/$basearch/
gpgcheck=1
gpgkey=http://mirrors.163.com/centos/RPM-GPG-KEY-CentOS-6

#packages used/produced in the build but not released
#[addons]
#name=CentOS-$releasever - Addons
#baseurl=http://mirrors.163.com/centos/$releasever/addons/$basearch/
#gpgcheck=1
#gpgkey=http://mirrors.163.com/centos/RPM-GPG-KEY-CentOS-6
#additional packages that may be useful
```

6.3.3 依赖关系

RPM 中某些包属于库, 它为多个应用程序提供功能。如果一个应用程序需要某个特定的库, 那么这个库就是一个依赖。要正常地安装一个软件包, 必须首先满足它的依赖关系。一个 RPM 软件包的依赖信息储存在这个 RPM 文件中。

yum 工具使用软件包依赖关系数据来保证一个应用程序在安装前所有的要求都已满足。它自动地安装依赖的软件包，如果系统中没有的话，或者某个新的应用程序的要求与现有的软件冲突，yum 会放弃安装，而不对系统做任何修改。

6.3.4 软件包名称

每个软件包文件都有一个很长的名字，包含了几个最重要的信息。下面是 OpenStack Keystone 包的全名：openstack-keystone-2012.2.4-5.el6.noarch.rpm。

管理工具处理软件包时，通常使用以下三种格式之一。

- 软件包名称 openstack-keystone。
- 带有版本号和发行版本的软件包名称：openstack-keystone-2012.2.4-5。
- 带有硬件架构的软件包名称：openstack-keystone-2012.2.4-5.el6.noarch。

为了清楚表明 yum 以名称架构的格式来列出软件包，仓库通常也将软件包存储在以架构区分的目录中。每次为软件包指定架构的时候，实际指定的是此软件对机器架构的最低要求。表 6-1 说明了硬件架构格式的含义。

表 6-1 硬件架构格式对应表

硬 件 架 构	说 明
i386	适合于任何现有的 X86 计算机
noarch	适合于所有架构
ppc	适合于 PowerPC 系统，例如 Apple Power Macintosh
X86_64	适合于 64 位 Intel 处理器，例如 Opteron

一些软件会为特殊类型的处理器优化，为 i386、i586、i686 和 X86_64 计算机提供不同的软件包。如果机器中有 Intel 奔腾、VIA C3 或其他兼容的处理器，那么可以使用 i586 软件包。如果机器中有 Intel 高能奔腾或更强劲的处理器，或是有时兴的 AMD 处理器，那么可以使用 i686 软件包。

在 yum 的命令行中，应当使用软件包的短名称，yum 会自动在符合你的机器架构的仓库中，选择版本最新的软件包。

用其他命名格式指定软件包，可以避免默认行为，强制 yum 使用指定版本或架构的软件包。只有当你知道默认选择有问题，不适宜安装时才应当这样做。

6.3.5 只下载软件包的方法

有很多时候，我们想只下载软件包，而不是直接把它们装到当前这台计算机上，但我们又不想一个个地下载，因为这样，一些依赖关系就需要我们自己去解决。针对此问题，这里介绍几种办法。

1. 方法一：downloadonly 插件

使用 yum 的插件 downloadonly，安装该工具后即可只下载不安装。

(1) 安装插件：

```
yum install yum-download
```

(2) 下载：

```
yum update httpd -y --downloadonly
```

这样 httpd 的 RPM 就被下载到/var/cache/yum/中去了。

也可以指定一个目录存放下载的文件：

```
yum update httpd -y --downloadonly --downloadaddir=/opt
```

值得注意的是，downloadonly 插件不但适用于 yum update，也适用于 yum install。

2. 方法二：yum-utils 中的 yumdownloader

yum-utils 中包含着一系列的 yum 工具，如 debuginfo-install、package-cleanup、reporclosure、repodiff、repo-graph、repomanage、repoquery、repo-rss、reposync、repotrack、verifytree、yum-builddep、yum-complete-transaction、yumdownloader、yum-debug-dump 和 yum-groups-manager。

(1) 安装 yum-utils.noarch：

```
yum -y install yum-utils
```

(2) 使用 yumdownloader：

```
yumdownloader httpd
```

3. 方法三：利用 yum 的缓存功能

用 yum 安装了某个工具后，我们想要这个工具的包，而此时 yum 安装的过程其实就已经把包给下载了，只是没有保存而已。所以，我们要做的是将其缓存功能打开。

(1) `vi /etc/yum.conf`

将其中的 `keepcache=0` 改为 `keepcache=1`，保存退出。

(2) `/etc/init.d/yum-updatesd restart`

(3) `yum install httpd`

(4) `cat /etc/yum.conf |grep cachedir`

```
cachedir=/var/cache/yum
```

(5) 跳到上述目录

```
cd cachedir=/var/cache/yum && tree ./
```

(6) 这个时候的目录树中应该可以找到所需要的安装包了。

6.3.6 RPM 常用命令

下面列出一些常用的 RPM 目录。对于每个命令，这里不做详细的解释，从互联网上可以找到许多说明和例子。

- 安装一个包：

```
# rpm -ivh
```

- 升级一个包：

```
# rpm -Uvh
```

- 移走一个包：

```
# rpm -e
```

- 安装参数。

`--force`：即使覆盖属于其他包的文件也强迫安装。

`--nodeps`：如果该 RPM 包的安装依赖于其他包，即使其他包没装，也强迫安装。

- 查询一个包是否被安装：

```
# rpm -q < rpm package name>
```

- 得到被安装的包的信息：

```
# rpm -qi < rpm package name>
```

- 列出该包中有哪些文件：

```
# rpm -ql < rpm package name>
```

- 列出服务器上的一个文件属于哪一个 RPM 包：

```
#rpm -qf
```

- 可组合好几个参数一起用，如

```
# rpm -qil < rpm package name>
```

- 列出所有被安装的 RPM 包：

```
# rpm -qa
```

- 列出一个未被安装进系统的 RPM 包文件中包含有哪些文件：

```
# rpm -qilp < rpm package name>
```

6.4 ISO

在安装系统或软件时，我们都需要使用 ISO 文件。在这里，我们对 ISO 文件做一些简短的说明。后面的章节中会说明如何做一个自己的 ISO 文件来发布。

ISO 文件可以挂载到 Linux 系统：

```
cd /root/source/ && mkdir redhat1
Mount -o loop RHEL-6.3-x86_64-bin-DVD1.iso /root/source/redhat1
```

挂载后可以查看 ISO 包中的内容：

```
[root@Code-Server redhat1]# ll
total 3428
dr-xr-xr-x 3 root root 2048 Jan 31 2013 EFI
lr-xr-xr-x 1 root root 7 Jan 31 2013 EULA -> EULA_en
-r--r--r-- 3 root root 10726 Nov 7 2012 EULA_de
-r--r--r-- 3 root root 8724 Nov 7 2012 EULA_en
-r--r--r-- 3 root root 10846 Nov 7 2012 EULA_es
-r--r--r-- 3 root root 10682 Nov 7 2012 EULA_fr
-r--r--r-- 3 root root 10497 Nov 7 2012 EULA_it
-r--r--r-- 3 root root 13173 Nov 7 2012 EULA_ja
-r--r--r-- 3 root root 9841 Nov 7 2012 EULA_ko
-r--r--r-- 3 root root 10033 Nov 7 2012 EULA_pt
-r--r--r-- 3 root root 7306 Nov 7 2012 EULA_zh
-r--r--r-- 3 root root 18092 Jun 30 2010 GPL
dr-xr-xr-x 3 root root 2048 Jan 31 2013 HighAvailability
dr-xr-xr-x 3 root root 2048 Jan 31 2013 images
dr-xr-xr-x 2 root root 2048 Jan 31 2013 isolinux
dr-xr-xr-x 3 root root 2048 Jan 31 2013 LoadBalancer
-r--r--r-- 2 root root 114 Jan 31 2013 media.repo
dr-xr-xr-x 2 root root 671744 Jan 31 2013 Packages
-r--r--r-- 2 root root 16435 Sep 2 2010 README
-r--r--r-- 3 root root 142742 Jan 22 2013 RELEASE-NOTES-as-IN.html
-r--r--r-- 3 root root 144051 Jan 22 2013 RELEASE-NOTES-bn-IN.html
...
```

```

-r--r--r-- 3 root root 161492 Jan 22 2013 RELEASE-NOTES-zh-CN.html
-r--r--r-- 3 root root 158904 Jan 22 2013 RELEASE-NOTES-zh-TW.html
dr-xr-xr-x 2 root root 4096 Jan 31 2013 repodata
dr-xr-xr-x 3 root root 2048 Jan 31 2013 ResilientStorage
-r--r--r-- 3 root root 3375 Jan 29 2013 RPM-GPG-KEY-redhat-beta
-r--r--r-- 3 root root 3211 Jan 29 2013 RPM-GPG-KEY-redhat-release
dr-xr-xr-x 3 root root 2048 Jan 31 2013 ScalableFileSystem
dr-xr-xr-x 3 root root 2048 Jan 31 2013 Server
-r--r--r-- 1 root root 11414 Jan 31 2013 TRANS.TBL

```

如下目录里显示了我们安装时选择安装类型的文件内容定义:

```

[root@Code-Server BOOT]# pwd
/repo/ISO/redhat1/EFI/BOOT
[root@Code-Server BOOT]# cat BOOTX64.conf
#debug --graphics
default=0
splashimage=/EFI/BOOT/splash.xpm.gz
timeout 5
hiddenmenu
title Red Hat Enterprise Linux 6.4
    kernel /images/pxeboot/vmlinuz
    initrd /images/pxeboot/initrd.img
title Install system with basic video driver
    kernel /images/pxeboot/vmlinuz xdriver=vesa nomodeset askmethod
    initrd /images/pxeboot/initrd.img
title rescue
    kernel /images/pxeboot/vmlinuz rescue askmethod
    initrd /images/pxeboot/initrd.img
[root@Code-Server BOOT]#

```

ISO 中的包都保存在 **packages** 目录下, 如果没有的话, 那么从 ISO 里寻找某个包的 **yum** 命令也会失败, 这里找不到需要的包的话, 那就是在这个 ISO 中不存在。

```

[root@Code-Server Packages]# pwd
/repo/ISO/redhat1/Packages
[root@Code-Server Packages]# ll *vnc*
-r--r--r-- 116 root root 96524 Aug 17 2010 gtk-vnc-0.3.10-3.el6.i686.rpm
-r--r--r-- 81 root root 97512 Aug 17 2010 gtk-vnc-0.3.10-3.el6.x86_64.rpm
-r--r--r-- 81 root root 17588 Aug 17 2010
gtk-vnc-python-0.3.10-3.el6.x86_64.rpm
-r--r--r-- 65 root root 163068 Aug 17 2010
libvncserver-0.9.7-4.el6.x86_64.rpm
-r--r--r-- 19 root root 263768 Jan 23 2013 tigervnc-1.1.0-5.el6.x86_64.rpm
-r--r--r-- 19 root root 1090492 Jan 23 2013
tigervnc-server-1.1.0-5.el6.x86_64.rpm
[root@Code-Server Packages]#

```

6.5 安装 OpenStack 组件——Keystone、Glance 和 Quantum

通用的安装过程如图 6-5 所示。



图 6-5 通用的安装过程

6.5.1 控制节点

1. 设置 yum 源

在后面小节列出的步骤里，我们可以直接执行 `yum install` 命令来安装，让 `yum` 自动寻找需要的包并安装。就像 Eclipse，我们只需要下载插件，而插件会自动到合适的网站去寻找插件的包和库文件。下面的例子可以将一个 ISO 文件设置为要安装的 `yum` 源：

```
cd /root/source/ && mkdir os
Mount -o loop RHEL-6.3-x86_64-bin-DVD1.iso /root/source/os
[root@NetworkController quantum]# cat /etc/yum.repos.d/rhel-source.repo
[rhel-source]
```

```
name=Red Hat Enterprise Linux $releasever - $basearch - Source
#baseurl=ftp://ftp.redhat.com/pub/redhat/linux/enterprise/$releasever/en/os/
SRPMS/
baseurl=file:///root/source/os/Server
enabled=1
gpgcheck=0
#gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
```

2. 安装数据库

下面安装 MySQL 数据库，和安装标准的 MySQL 一样：

```
yum install -y mysql mysql-server MySQL-python
chkconfig --level 2345 mysqld on
service mysqld start
```

这一步无须修改密码，默认为空。

3. 安装 RabbitMQ

RabbitMQ 用于各个组件间无共享地通信。它被安装到一台服务器上，通常是控制节点：

```
yum install -y tk
yum -y install mysql-connector-odbc
rpm -ivh rpm-dependency-for-RabbitMQ-OS-RHEL/*.rpm
rpm --import http://www.rabbitmq.com/rabbitmq-signing-key-public.asc
rpm -ivh rabbitmq-server-3.0.3-1.noarch.rpm
chkconfig rabbitmq-server on && service rabbitmq-server start
```

4. 安装 Keystone

安装 OpenStack 组件时，还需要初始化数据库。每个项目有一个数据库，用于保存本项目需要的管理逻辑或状态数据。该数据库有自己专门的用户名和密码来访问。

在此之后，要把服务变成自启动，这样在断电重启后，系统可以正常工作：

```
cd $path/packages/openstack
rpm -ivh openstack-utils-2013.1-1.el6.noarch.rpm
yum -y install python-paste python-decorator python-sqlalchemy python-tempita
cd $path/packages/openstack/rpm-dependency-for-keystone-OS-RHEL
rpm -ivh *.rpm
yum -y install PyPAM python-memcached
cd $path/packages/openstack/python
rpm -ivh python-keystone-2012.2.1-1.el6.noarch.rpm
python-keystone-2012.2.1-1.el6.noarch.rpm
python-keystoneclient-0.1.3.27-1.el6.noarch.rpm
cd $path/packages/openstack
```



```
rpm -ivh openstack-keystone-2012.2.1-1.el6.noarch.rpm
# 初始化数据库
openstack-db --service keystone --init -r
openstack-config --set /etc/keystone/keystone.conf DEFAULT admin_token ADMIN
# 启动服务并在系统重启时也生效
service openstack-keystone start && chkconfig openstack-keystone on
keystone-manage db_sync
service openstack-keystone restart
```

OpenStack 是用 Python 开发的，安装后，可以到系统的/usr/lib/python2.6/site-packages 目录下找到各个项目的源代码，每个项目一个目录，以本项目名为目录名。

不同的操作系统，安装后目录也不一样。下面展示的是在 CentOS 上安装后目录的部分文件列表：

```
[root@CloudController-sec site-packages]# pwd
/usr/lib/python2.6/site-packages
[root@CloudController-sec site-packages]# ll keystone
total 224
drwxr-sr-x 3 root root 4096 Feb 10 2014 assignment
drwxr-sr-x 3 root root 4096 Feb 10 2014 auth
drwxr-sr-x 3 root root 4096 Feb 10 2014 catalog
-rw-r--r-- 1 root root 2650 Nov 26 15:17 clean.py
-rw-r--r-- 2 root root 2905 Nov 26 15:18 clean.pyc
-rw-r--r-- 2 root root 2905 Nov 26 15:18 clean.pyo
-rw-r--r-- 1 root root 7104 Nov 26 15:17 cli.py
-rw-r--r-- 2 root root 7950 Nov 26 15:18 cli.pyc
-rw-r--r-- 2 root root 7950 Nov 26 15:18 cli.pyo
drwxr-sr-x 6 root root 4096 Feb 10 2014 common
-rw-r--r-- 1 root root 1952 Nov 26 15:17 config.py
-rw-r--r-- 2 root root 1394 Nov 26 15:18 config.pyc
-rw-r--r-- 2 root root 1394 Nov 26 15:18 config.pyo
drwxr-sr-x 11 root root 4096 Feb 10 2014 contrib
-rw-r--r-- 1 root root 5356 Nov 26 15:17 controllers.py
-rw-r--r-- 2 root root 5485 Nov 26 15:18 controllers.pyc
-rw-r--r-- 2 root root 5485 Nov 26 15:18 controllers.pyo
```

OpenStack 组件安装后，所有配置文件都在/etc 目录，以本项目名为子目录名，寻找起来非常方便：

```
[root@CloudController-sec keystone]# ll /etc/keystone
total 36
-rw-r----- 1 root keystone 1539 Nov 26 11:23 default_catalog.templates
-rw-r----- 1 root keystone 15628 Nov 26 15:17 keystone.conf
-rw-r----- 1 root keystone 2880 Nov 26 11:23 keystone-paste.ini
-rw-r----- 1 root keystone 1046 Nov 26 11:23 logging.conf
```

```
-rw-r----- 1 keystone keystone 5203 Nov 26 11:23 policy.json
[root@CloudController-sec keystone]#
```

安装之后，需要对组件进行配置。每个配置文件取决于项目，有多少不等的配置参数。但基本上，每个项目都需要对数据库的访问信息进行配置，包括数据库所在服务器地址、端口、用户名和密码等。下文展示了在配置文件里配置数据库连接的例子：

```
[sql]
connection = mysql://keystone:keystone@localhost/keystone
# The SQLAlchemy connection string used to connect to the database
# connection = sqlite:///keystone.db
# the timeout before idle sql connections are reaped
# idle_timeout = 200
[identity]
driver = keystone.identity.backends.sql.Identity
# driver = keystone.identity.backends.sql.Identity
```

在 Keystone 安装后，需要创建项目/租户、角色、用户，并把它们关联起来：

```
#
# Service tenant
#
SERVICE_TENANT=$(get_id keystone tenant-create --name=service \
--description "Service Tenant")

GLANCE_USER=$(get_id keystone user-create --name=glance \
--pass=*****)
keystone user-role-add --user-id $GLANCE_USER \
--role-id $ADMIN_ROLE \
--tenant-id $SERVICE_TENANT
NOVA_USER=$(get_id keystone user-create --name=nova \
--pass=Passw0rd \
--tenant-id $SERVICE_TENANT)
keystone user-role-add --user-id $NOVA_USER \
--role-id $ADMIN_ROLE \
--tenant-id $SERVICE_TENANT
```

还需要根据需要安装的项目，分别创建相应的服务访问端点：

```
#
# Keystone service
#
KEYSTONE_SERVICE=$(get_id \
keystone service-create --name=keystone \
--type=identity \
--description="Keystone Identity Service")
if [[ -z "$DISABLE_ENDPOINTS" ]]; then
```

```
keystone endpoint-create --region RegionOne --service-id $KEYSTONE_SERVICE \
    --publicurl "http://NC_IP:5000/v2.0" \
    --adminurl "http://NC_IP:35357/v2.0" \
    --internalurl "http://NC_IP:5000/v2.0"
fi
```

5. 安装 Glance

Glance 的安装非常简单，包括安装软件包、生成数据库，并启动服务：

```
### INSTALL Glance
cd $path/packages/openstack/rpm-dependency-for-Glance-OS-RHEL
rpm -ivh *.rpm
cd $path/packages/openstack/python
rpm -ivh python-swiftclient-1.2.0-2.el6.noarch.rpm\
    python-glance-2012.2.3-1.el6.noarch.rpm\
    python-glanceclient-0.5.1-1.el6.noarch.rpm
cd $path/packages/openstack
rpm -ivh openstack-glance-2012.2.3-1.el6.noarch.rpm
# 初始化数据库
openstack-db --service glance --init -r
glance-manage db_sync
# 启动服务并在系统重启时也生效
chkconfig openstack-glance-registry on
chkconfig openstack-glance-api on
service openstack-glance-registry start
service openstack-glance-api start
```

如果想看源代码的话，可以到相应目录里。下面展示了在 `/usr/lib/python2.6/site-packages/glance` 目录里查看 Glance 的代码。感兴趣的人可以到这些目录里查看并调试 Glance 的代码。

```
[root@CloudController-pri ~]# ll /usr/lib/python2.6/site-packages/glance
total 120
drwxr-sr-x 5 root root 4096 Feb 10 2014 api
drwxr-sr-x 2 root root 4096 Feb 10 2014 cmd
drwxr-sr-x 2 root root 4096 Feb 10 2014 common
-rw-r--r-- 1 root root 3133 Nov 26 11:25 context.py
-rw-r--r-- 2 root root 2745 Nov 26 15:25 context.pyc
-rw-r--r-- 2 root root 2745 Nov 26 15:25 context.pyo
drwxr-sr-x 5 root root 4096 Feb 10 2014 db
drwxr-sr-x 2 root root 4096 Feb 10 2014 domain
-rw-r--r-- 1 root root 4026 Nov 26 11:25 gateway.py
-rw-r--r-- 2 root root 3014 Nov 26 15:25 gateway.pyc
-rw-r--r-- 2 root root 3014 Nov 26 15:25 gateway.pyo
drwxr-sr-x 3 root root 4096 Feb 10 2014 image_cache
```

不同于其他项目，Glance 全部的两个服务都需要端口，有自己的配置文件。前面讲过，glance-registry 用于管理和保存镜像的元数据，而 glance-api 接收请求，并将镜像保存到配置好的存储后端里去。

```
[root@CloudController-pri ~]# ll /etc/glance/
total 48
-rw-r--r-- 1 root root 11634 Jan 28 11:32 glance-api.conf
-rw-r----- 1 root glance 2603 Nov 26 15:25 glance-api-paste.ini
-rw-r----- 1 root glance 6686 Nov 26 11:25 glance-cache.conf
-rw-r----- 1 root glance 3703 Nov 26 15:25 glance-registry.conf
-rw-r----- 1 root glance 866 Nov 26 15:25 glance-registry-paste.ini
-rw-r----- 1 root glance 2246 Nov 26 11:25 glance-scrubber.conf
-rw-r----- 1 root glance 982 Nov 26 11:25 logging.cnf.sample
-rw-r----- 1 root glance 101 Nov 26 11:25 policy.json
-rw-r----- 1 root glance 1259 Nov 26 11:25 schema-image.json
```

下面是 glance-api 的配置信息。如果是简单的使用，其实里面大部分配置都可以直接使用默认的，只需要关注服务端口，绑定服务主机、数据库和日志信息等。

```
# Address to bind the API server
bind_host = nn.nn.nn.nn
# Port the bind the API server to
bind_port = 9292
# Log to this file. Make sure you do not set the same log
# file for both the API and registry servers!
log_file = /var/log/glance/api.log
# Backlog requests when creating socket
backlog = 4096
# TCP_KEEPIDLE value in seconds when creating socket.
# Not supported on OS X.
#tcp_keepidle = 600
# SQLAlchemy connection string for the reference implementation
# registry server. Any valid SQLAlchemy connection string is fine.
# See:
http://www.sqlalchemy.org/docs/05/reference/sqlalchemy/connections.html#sqla
lchemy.create_engine
sql_connection = mysql://glance:*****@nn.nn.nn.nn/glance
```

在这里，glance-api 需要知道 glance-registry 服务的位置。Glance 的这两个服务可以分开部署，也可以部署于同一个节点：

```
# ===== Registry Options =====
# Address to find the registry server
registry_host = nn.nn.nn.nn
# Port the registry server is listening on
registry_port = 9191
```

现在该配置消息中间件了。绝大部分 OpenStack 服务都需要配置消息中间件的位置和访问信息。有了消息中间件，这个云平台才可以流动起来。

```
# ===== Notification System Options =====
# Notifications can be sent when images are create, updated or deleted.
# There are three methods of sending notifications, logging (via the
# log_file directive), rabbit (via a rabbitmq queue), qpid (via a Qpid
# message queue), or noop (no notifications sent, the default)
notifier_strategy = rabbit
# Configuration options if sending notifications via rabbitmq (these are
# the defaults)
rabbit_host = nn.nn.nn.nn
rabbit_port = 5672
rabbit_use_ssl = false
rabbit_userid = admin
rabbit_password = *****
rabbit_virtual_host = /
rabbit_notification_exchange = glance
rabbit_notification_topic = glance_notifications
rabbit_durable_queues = False
```

下面指明镜像将被存储在哪里。本地文件系统是最省力气的地方，可以满足绝大部分用户场景的需要。

```
# ===== Filesystem Store Options =====

# Directory that the Filesystem backend store
# writes image data to
filesystem_store_datadir = /var/lib/glance/images/
```

下面是 glance-registry 的配置文件。其内容类似于 glance-api 的配置。但由于这两个服务可以分布式部署，所以配置文件也是单独的，也需要进行配置。

```
# Address to bind the registry server
bind_host = 0.0.0.0
# Port the bind the registry server to
bind_port = 9191
# Log to this file. Make sure you do not set the same log
# file for both the API and registry servers!
log_file = /var/log/glance/registry.log
# Backlog requests when creating socket
backlog = 4096
# TCP_KEEPIIDLE value in seconds when creating socket.
# Not supported on OS X.
#tcp_keepidle = 600
# SQLAlchemy connection string for the reference implementation
```

```
# registry server. Any valid SQLAlchemy connection string is fine.
# See:
http://www.sqlalchemy.org/docs/05/reference/sqlalchemy/connections.html#sqla
lchemy.create_engine
sql_connection = mysql://glance:*****@nn.nn.nn.nn/glance
```

下面是用于 Keystone 的认证信息。这里密码是未加密的，因此会有一定的安全隐患。

```
[keystone_authtoken]
auth_host = nn.nn.nn.nn
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = glance
admin_password = *****
```

下面是 Glance 进程。需要注意的是，每个服务启动后可能会有多个进程同时运行。

```
[root@NetworkController-pri ~]# ps -ef | grep glance
glance    2540    1  0 Jan28 ?        00:00:00 /usr/bin/python
/usr/bin/glance-api --config-file /etc/glance/glance-api.conf
glance    2750  2540  0 Jan28 ?        00:00:00 /usr/bin/python
/usr/bin/glance-api --config-file /etc/glance/glance-api.conf
glance    2781    1  0 Jan28 ?        00:00:00 /usr/bin/python
/usr/bin/glance-registry --config-file /etc/glance/glance-registry.conf
--debug --verbose
glance    2797  2781  0 Jan28 ?        00:00:00 /usr/bin/python
/usr/bin/glance-registry --config-file /etc/glance/glance-registry.conf
--debug --verbose
```

6. 安装 Neutron

Neutron 包括下列组件：

- neutron-server
- neutron-l2-agent
- neutron-dhcp-agent
- neutron-l3-agent

在本过程中，我们使用 OpenvSwitch 来实现二层交换：

```
# install neutron
cd $openstackpath/neutron
rpm -Uvh --replacepks *.rpm
# 启动 openvswitch 服务
service openvswitch start
```

```
# 初始化 neutron server
/bin/cp $path/nc_automation/neutron-server-setup
/usr/bin/neutron-server-setup
neutron-server-setup --plugin openvswitch -r
# 启动服务并在系统重启时也生效
chkconfig neutron-server on
service neutron-server start
# 初始化本机节点服务
/bin/cp $path/nc_automation/neutron-node-setup /usr/bin/neutron-node-setup
neutron-node-setup --plugin openvswitch -q localhost
# 在这里我们增加了两个 openvswitch 网桥
ovs-vsctl add-br br-int
ovs-vsctl add-br br-phy
# 启动服务并在系统重启时也生效
chkconfig neutron-openvswitch-agent on
service neutron-openvswitch-agent start
# 初始化本机节点服务
neutron-dhcp-setup --plugin openvswitch -q localhost
# 启动服务并在系统重启时也生效
chkconfig neutron-dhcp-agent on
service neutron-dhcp-agent start
# 增加 openvswitch 网桥
ovs-vsctl add-br br-ex
```

其中:

- **neutron-server-setup** 为选择的 Neutron 服务组件安装数据库信息;
- **neutron-node-setup** 为选择的组件安装和本节点相关的数据库信息。

OpenStack 里定义了一些通用的组件,特别是网络。在各个文档中都可以看到如下一些网络组件的名称。

- **br-int:** OpenvSwitch 集成网桥,所有在本机运行的虚拟机都会连接到这个网桥上。网络服务在各个虚拟机之间通过配置 **br-int** 的端口实现了隔离。
- **br-ex:** 外部网桥,连接到外部网络去。
- **br-eth1:** 虚拟机通信用的网桥,需要通过物理机网口如 **eth1** 连接到物理网络的交换机,它也通过 **veth** 对 (**int-br-eth1**、**phy-br-eth1**) 连接到集成网桥 (**br-int**)。

下面展示了 **neutron** 所有涉及的配置文件:

```
[root@ControlNode-pri openvswitch]# cd /etc/neutron/
[root@ControlNode-pri neutron]# ll
total 68
-rw-r--r--  1 root  root    761 Jan 28 00:02 api-paste.ini
```

```
-rw-r-----. 1 root    neutron 2771 Nov 28 02:35 dhcp_agent.ini
-rw-r--r--. 1 root    root      2324 Jan 28 00:02 l3_agent.ini
-rw-r-----. 1 root    neutron 1104 Nov 28 02:35 lbaas_agent.ini
-rw-r-----. 1 root    neutron 1003 Nov 28 02:35 metadata_agent.ini
-rw-r-----. 1 root    neutron  407 Nov 28 02:35 metering_agent.ini
-rw-r--r--. 1 root    root     13470 Jan 28 00:02 neutron.conf
drwxr-sr-x. 16 root    root      4096 Jan 23 15:58 plugins
-rw-r-----. 1 root    neutron 5853 Nov 28 02:35 policy.json
-rw-r--r--. 1 root    root       84 Nov 28 02:53 release
-rw-r--r--. 1 root    root     1214 Nov 28 02:35 rootwrap.conf
drw-r-----. 2 neutron root      4096 Nov 28 02:53 rootwrap.d
-rw-r-----. 1 root    neutron  432 Nov 28 02:35 vpn_agent.ini
```

下面是部分 `neutron.conf` 中的配置:

```
core_plugin =
neutron.plugins.openvswitch.ovs_neutron_plugin.OVSNeutronPluginV2
service_provider=LOADBALANCER:Haproxy:neutron.services.loadbalancer.drivers.
haproxy.plugin_driver.HaproxyOnHostPluginDriver:default
```

从 Havanna 版本开始, 安装 Openstack 之后, 现有支持的每种网络插件都会被安装到如下目录里:

```
[root@ControlNode-pri neutron]# cd plugins/
[root@ControlNode-pri plugins]# ll
total 56
drwxr-sr-x. 2 root root 4096 Jan 23 15:58 bigswitch
drwxr-sr-x. 2 root root 4096 Jan 23 15:58 brocade
drwxr-sr-x. 2 root root 4096 Jan 23 15:58 cisco
drwxr-sr-x. 2 root root 4096 Jan 23 15:58 hyperv
drwxr-sr-x. 2 root root 4096 Jan 23 15:58 linuxbridge
drwxr-sr-x. 2 root root 4096 Jan 23 15:58 metaplugin
drwxr-sr-x. 2 root root 4096 Jan 23 15:58 midonet
drwxr-sr-x. 2 root root 4096 Jan 23 15:58 ml2
drwxr-sr-x. 2 root root 4096 Jan 23 15:58 mlnx
drwxr-sr-x. 2 root root 4096 Jan 23 15:58 nec
drwxr-sr-x. 2 root root 4096 Jan 23 15:58 nicira
drwxr-sr-x. 2 root root 4096 Jan 28 00:02 openvswitch
drwxr-sr-x. 2 root root 4096 Jan 23 15:58 plumgrid
drwxr-sr-x. 2 root root 4096 Jan 23 15:58 ryu
```

`ovs_neutron_plugin.ini` 配置文件的例子:

```
[ovs]
# (StrOpt) Type of network to allocate for tenant networks. The default value
'local' is useful only for single-box testing and provides no connectivity
between hosts. You MUST either change this to 'vlan' and configure
network_vlan_ranges below or change this to
# 'gre' or 'vxlan' and configure tunnel_id_ranges below in order for tenant
```



```

networks to provide connectivity between hosts. Set to 'none'
# to disable creation of tenant networks.
#
tenant_network_type = vlan
# Example: tenant_network_type = gre
# Example: tenant_network_type = vxlan
# (ListOpt) Comma-separated list of <physical_network>[:<vlan_min>:<vlan_max>]
tuples enumerating ranges of VLAN IDs on named physical networks that are available
for allocation. All physical networks listed are available for flat and VLAN
provider network creation. Specified ranges of VLAN IDs are available for tenant
network allocation if tenant_network_type is 'vlan'. If empty, only gre, vxlan
and local networks may be created.
#
network_vlan_ranges = physnet1:1:4094
# Example: network_vlan_ranges = physnet1:1000:2999
# Do not change this parameter unless you have a good reason to. This is the name
of the OVS integration bridge. There is one per hypervisor.
# The integration bridge acts as a virtual "patch bay". All VM VIFs are attached
to this bridge and then "patched" according to their network
# connectivity.
#
integration_bridge = br-int
# (ListOpt) Comma-separated list of <physical_network>:<bridge> tuples mapping
physical network names to the agent's node-specific OVS
# bridge names to be used for flat and VLAN networks. The length of bridge names
should be no more than 11. Each bridge must exist, and should have a physical
network interface configured as a port. All physical networks listed in
network_vlan_ranges on the server should have mappings to appropriate bridges
on each agent.
#
bridge_mappings = physnet1:br-phy
[agent]
# Agent's polling interval in seconds
polling_interval = 2
[securitygroup]
# Firewall driver for realizing neutron security group function.
firewall_driver =
neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
# 1. With VLANs on eth1.
[database]
connection = mysql://neutron:*****@nn.nn.nn.nn:3306/ovs_neutron

```

通过 OpenStack vSwitch 命令可以查看各种软件交换机和端口的信息，包括哪些端口连接到哪个虚拟网桥上：

```
[root@NetworkController-pri ~]# ovs-vsctl show
```

```
d8e8c635-6822-4308-8f1e-d6de647af504
  Bridge br-ex
    Port br-ex
      Interface br-ex
        type: internal
  Bridge br-int
    Port br-int
      Interface br-int
        type: internal
    Port int-br-phy
      Interface int-br-phy
  Bridge br-phy
    Port phy-br-phy
      Interface phy-br-phy
    Port br-phy
      Interface br-phy
        type: internal
  ovs_version: "1.10.0"
```

查看有哪些虚拟网桥:

```
[root@NetworkController-pri ~]# ovs-vsctl list-br
br-ex
br-int
br-phy
```

查看某个网桥上有哪些端口:

```
[root@NetworkController-pri ~]# ovs-vsctl list-ports br-int
int-br-phy
```

使用 `ifconfig` 则可以看到各个端口, 不管是物理的, 还是虚拟的, 不管是 Linux Bridge 的, 还是 OpenvSwitch 的:

```
[root@NetworkController-pri ~]# ifconfig
br-ex  Link encap:Ethernet HWaddr 3A:8B:46:1B:F4:4B
        inet6 addr: fe80::c41:e8ff:fe86:ad36/64 Scope:Link
        UP BROADCAST RUNNING MTU:1500 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:0 (0.0 b) TX bytes:468 (468.0 b)

br-int Link encap:Ethernet HWaddr 96:67:E9:36:71:43
        inet6 addr: fe80::5848:95ff:fe63:7db1/64 Scope:Link
        UP BROADCAST RUNNING MTU:1500 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
```

```

RX bytes:0 (0.0 b) TX bytes:468 (468.0 b)
br-phy    Link encap:Ethernet HWaddr C2:8D:AD:E2:A3:43
          inet6 addr: fe80::3c88:clff:feef:998d/64 Scope:Link
          UP BROADCAST RUNNING MTU:1500 Metric:1

```

6.5.2 计算节点

在计算节点，只需要安装 Quantum-*agent 的相关服务，包括如下步骤。

- 安装 RPM 包。
- 启动 OpenvSwitch 服务。
- 启动 neutron-openvswitch-agent 服务。
- 创建网桥。
 - ovs-vsctl add-br br-int
 - ovs-vsctl add-br br-phy

6.5.3 设置 iptables 规则

另外，还需要设置 iptables 规则：

```

# MYSQL 3306
iptables -I INPUT 1 -p tcp --dport 3306 -j ACCEPT
# KEYSTONE 5000 35357
iptables -I INPUT 1 -p tcp --dport 5000 -j ACCEPT
iptables -I INPUT 1 -p tcp --dport 35357 -j ACCEPT
# GLANCE 9191 9292
iptables -I INPUT 1 -p tcp --dport 9191 -j ACCEPT
...

```

6.6 大规模安装技术与工具

6.6.1 Chef

Chef 是由 Ruby 开发的服务器配置管理工具。Chef 属于 Apache License Version，本身是一个自动化平台，又是一个云平台基础设施自动化框架，不论基础设施的大小和规模，

可以非常容易地部署服务器和应用到任何物理、虚拟的节点，或者云中指定的位置。利用 Chef 可以很容易而快速地交付产品和服务，可以管理复杂且需要快速扩展的基础设施来满足各种需求。

如果我们现在需要搭建一台 MySQL 主从服务器，第一次安装过程我们手动操作了。没多久，我们需要搭建第二台，这时候我们会想，如果安装第一台之后，把操作过程执行的命令写成脚本，那么安装第二台时运行一下脚本就行了，节约时间而且不容易出错。

Chef 就相当于这样的一个脚本管理工具，但功能要强大得多，可定制性强。Chef 将脚本命令代码化，定制时只需要修改代码，安装的过程就是执行代码的过程。比如，Chef 就像一个制作玩具的工厂，它可以把一些原材料做成漂亮的玩具，它有一些模板。当把原材料放进去，选择一个模板，它就会制造出这个玩具。服务器的配置也是这样，一台还没有配置的服务器，当给它指定一个模板（role 或 recipe），Chef 就会把它配置成想要的线上服务器。

这个只是 Chef 的一方面，因为在安装好系统后执行一些脚本也可以达到同样的目的。Chef 还有另一方面功能是脚本达不到的，那就是 Chef 对经过配置的服务器有远程控制的能力，它可以随时对系统进行进一步的配置或修改，就像前面的玩具工厂可以随时改变它的玩具的颜色、大小，也可以通过手动的方式实现这点，但是当服务器比较多时，手动的方式不但工作量很大，且出错概率大增。

Chef 架构如图 6-6 所示。

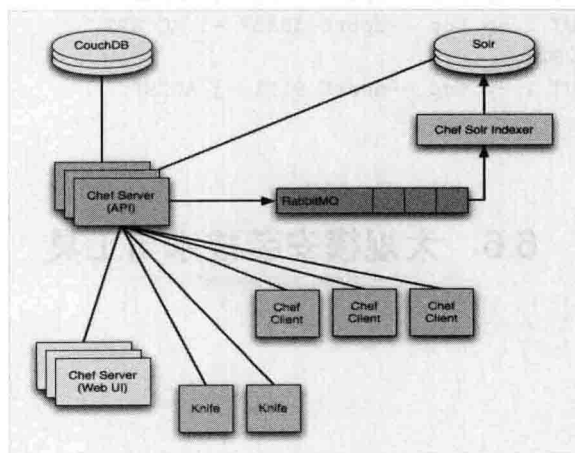


图 6-6 Chef 架构

对图中元素的说明如下。

(1) 有一个中心服务器（运行 `chef-server`）。

- Chef 将数据存储存储在 CouchDB 数据库里面。
- RabbitMQ 和 `chef-solo` 等提供搜索的功能。
- Chef 还提供了图形用户界面（`cher-server-webui`）。

(2) 可以有多个 Workstation（运行 Knife 工具对 Chef 进行配置）。

- Workstation 上有一个 `pem` 文件，Knife 利用它作为认证来和 `chef-server` 通过 REST API 进行通信。
- Workstation 将配置（利用 Recipes 等描述各 Client 应该如何配置自己）上传到服务器。
- Workstation 和中心服务器可以在同一台机器。

(3) 可以有多个 Client（被配置节点）。

- Client 上有一个 `pem` 文件，`chef-client` 利用它来认证并和 `chef-server` 通过 REST API 进行通信。
- 当新加一个 Client 的时候，需要从中心服务器上复制 `validator.pem` 到新加的 Client 节点。
- Client 利用这个 `pem` 进行注册得到自己的 `client.pem` 进行以后的认证。
- Client 连到 Chef Server，然后进行自我配置。

Chef 服务器是一个中心的管理控制节点，有一个数据库用来保存关于云平台/基础设施的角色、类型、cookbook 等信息。Chef 服务器可以通过推送的方式将配置发送到各个节点，或者由节点来主动地获取配置信息。另外，Chef 服务器还有 Web UI，除了管理功能，还可以通过拖曳的方式来修改配置或推送配置到节点。

节点是被 Chef 服务器管理的物理或虚拟服务器，例如，在 OpenStack 里，运行了 `nova-compute` 或者 `cinder-volume` 的服务器都是节点，可以被 Chef 服务器来管理。

工作台是用户管理 Chef 服务器的工作节点，这里，可以执行一些列管理命令，制作 cookbook，并提交到 Chef 服务器，查询各个节点的状态或角色等。

部署和使用 Chef 需要了解 Ruby 编程，这给 Chef 的推广增加了一些麻烦，然而也可以用 Shell 脚本来编写逻辑。同任何一门语言一样，Chef 强大到可以让你用自己熟悉的语言编写逻辑并集成进去，只要比较熟悉 Chef 的工作原理。

部署 Chef 时，很多工作都在编写和维护 cookbook，但是在网上已经有了许多开源的 Chef cookbook，事实上并不需要从头做起，除非自己想做。

Chef 节点可以通过 run-list 来直接在节点上产生配置，通过对应用、资源和管理抽象来减少管理的复杂度，并且通过版本管理来保存各种配置信息。

recipes 是资源的集合，而 cookbook 则包括 recipes、模板、文件、配置的资源等。

1. Chef 管理的资源

下面列出一些被 Chef 抽象和管理的资源的例子。

- 网络。
- 文件。
- 目录。
- 符号链接。
- 挂载点。
- 路由。
- 用户。
- 组。
- 任务。
- 软件包。
- 软件。
- 服务。
- 配置。
- 其他。

2. Chef 用户

Chef 用户包括 IBM、Intel、Rackspace、HP、DreamHost、BaremetalCloud、Mirantis、Dell、CloudScaling、Calxeda 等著名公司。

6.6.2 Puppet

Puppet 是一种 Linux、UNIX、Windows 平台的集中配置管理系统，使用自有的 Puppet 描述语言，可管理配置文件、用户、cron 任务、软件包、系统服务等。Puppet 把这些系统

实体称为资源，Puppet 的设计目标是简化对这些资源的管理以及妥善处理资源间的依赖关系。

Puppet 采用客户/服务器架构，所有的客户端和一个或几个服务器交互。每个客户端周期（默认半个小时）向服务器发送请求，获得其最新的配置信息，保证和该配置信息同步。每个 Puppet 客户端每 30 分钟（可以设置）连接一次服务器端，下载最新的配置文件，并且严格按照配置文件来配置服务器。配置完成以后，Puppet 客户端可以反馈给服务器端一个消息，如果出错也会给服务器端反馈一个消息。

开发 Puppet 是为了让系统管理员可以相互交流和共享成熟的工具，避免重复的劳动。它通过以下两个特性来实现这一目标。

- 提供一个简洁的但是强大的框架来完成系统管理任务。
- 系统管理任务可以描述成 Puppet 语言，因此可以相互分享代码，就像分享其他语言的代码一样。

因此，作为系统管理员可以更快地完成工作，因为可以用 Puppet 来处理所有的管理细节，甚至还可以下载其他管理员的 Puppet 代码来让自己的工作完成得更快。

Puppet 与其他手工操作工具最大的区别是 Puppet 的配置具有稳定性，因此可以多次执行 Puppet，一旦更新了自己的配置文件，Puppet 就会根据配置文件来更改机器配置，通常每 30 分钟检查一次。Puppet 会让系统状态同配置文件所要求的状态保持一致，比如配置文件里面要求 SSH 服务必须开启，假如不小心 SSH 服务被关闭了，那么下一次执行 Puppet 的时候，Puppet 会发现这个异常，然后会开启 SSH 服务，让系统状态和配置文件保持一致。

可以使用 Puppet 管理服务器的整个生命周期，从初始化到退役。不同于传统的例如 Sun 的 Jumpstart 或者 Red Hat 的 Kickstart，Puppet 可以持续地让服务器保持最新状态，只要一开始正确地配置它们，然后就再也不用去管了。通常 Puppet 用户只需要给机器安装好 Puppet 并让它们运行，然后剩余的工作都由 Puppet 来完成。

6.6.3 Chef 与 Puppet 的比较

表 6-2 列举了 Chef 与 Puppet 的一些基本差别。

表 6-2 Chef 与 Puppet 对比

	Puppet	Chef
历史	有一些	新
用户	多, 不乏知名公司, 如 Mirantis、Red Hat	较少, 包括 IBM、Rackspace、Dell
开发活跃度	中等	活跃
依存关系表达	运行次序是根据状况由系统端决定的, 类似于 Makefile	基本上是书写顺序, 相比 Puppet 更具脚本风格
必要中间件	没有	服务端需要安装有 CouchDB、RabbitMQ
安装	简单, 用 gem 安装就可以	比较复杂
与其他系统交互	基本上没有	因为使用 RESTful 的服务 API, 用 JSON 可以取值, 所以能做许多事

6.6.4 IBM xCAT

IBM xCAT 是一个开源的、具有十多年历史的部署工具，原来用于高性能计算集群部署，但其本身的部署功能也可以用来部署物理机与虚拟机。因为历史悠久，其功能也非常强大而可靠。它具有硬件二次发现、丰富的命令行工具、图形化界面等许多优秀而稳定的功能，具体如下。

- 安装和配置物理机和虚拟机运行环境。
- 各种虚拟化管理程序。
- 部署 OpenStack。
- 部署 KVM。
- 配置和部署集群。
- 部署和配置 HPC 集群。
- 部署 IBM Platform LSF。
- 部署许多第三方应用集群。
- 部署分析类软件。
- 部署 IBM Platform Symphony。
- 部署 Hadoop。
- 部署 Linux 和 Windows 集群来支持远程桌面应用。

第7章

系统定制技术

7.1 系统环境的定制

7.1.1 KVM 的检查与安装

KVM 是 OpenStack 默认的虚拟机管理程序。大量的 OpenStack 部署使用的是 KVM。KVM 以 Linux 进程的形式运行。下面简单介绍安装 KVM 的过程。

(1) 首先确定主机的 CPU 是否支持 vmx 或者 svm 虚拟化, vmx 属于 Inter 处理器, svm 属于 AMD 处理器; 或者用 `cpu-z` 查看处理器是否支持 vt-x 虚拟化, BIOS 中需要开启 vt 支持。下面命令可以用来查看是否支持虚拟化:

```
~]#egrep 'vmx|svm' /proc/cpuinfo//
```

(2) 利用 Kickstart 安装 KVM, 最主要是添加 4 个组包。

- ① `virtualization` //提供虚拟机的环境, 主要包含 `qemu-kvm`。
- ② `virtualization-client` //管理和安装虚拟机实例的客户端, 主要有 `python-virtinst`、`virt-manager`、`virt-viewer`。
- ③ `virtualization-platform` //提供访问和控制虚拟客户端的接口, 主要有 `libvirt`、`libvirt-client`。
- ④ `@virtualization-tools` //管理离线虚拟机镜像的工具, 主要有 `libguestfs` 等。

用户根据需求选择安装的软件包, 一般都安装①、②、③。可以利用 `yum groupinstall "Virtualization"` 或者 `"Virtualization Client"` 或者 `"Virtualization Platform"` 来通过命令行安装这些软件包。

以下说明已经加载了 `kvm` 和 `kvm_intel` 的模块。

```
~]#lsmod|grep kvm
kvm_intel          52570  3
kvm                314739  1 kvm_intel
```

(3) 使用下列命令检查 KVM 是否成功安装。

```
~]#virsh -c qemu:///system list //以下表示正常
将会显示如下结果:
```

```
Id Name          State
-----
```

(4) 查看 libvirtd API 工具是否启动。

```
~]# service libvirtd status
libvirtd (pid 1607) is running...
```

(5) 查看 libvirtd 开机启动 runlevel 为 3、4、5 级别。

```
~]# chkconfig --list libvirtd
libvirtd      0:off 1:off 2:off 3:on  4:on  5:on  6:off
SELinux
```

SELinux 是一种访问控制体系，在这种访问控制体系的限制下，进程只能访问那些在它的任务中所需要的文件。然而在通常企业里部署，该功能反而会带来很多问题。通常的做法是关闭它。

如果改变了模式则需要系统重启；如果由 enforcing（强制）或 permissive（宽容）改成 disabled（关闭），或由 disabled（关闭）改成其他两个，那也必须重新开机。这是因为 SELinux 是整合到核心里面去的，用户只可以在 SELinux 运作下在 enforcing（强制）或 permissive（宽容）模式下切换，不能够直接关闭 SELinux。

SELinux 的启动、关闭与查看

(1) 并非所有的 Linux distributions 都支持 SELinux，目前 SELinux 支持三种模式，分别如下。

- **enforcing**: 强制模式，代表 SELinux 运行中，并且已开始正确地限制 domain/type 了。
- **permissive**: 宽容模式，代表 SELinux 运作中，不过仅会有警告讯息并不会实际限制 domain/type 的存取。这种模式可以用来作为 SELinux 设置的 debug 之用。
- **disabled**: 关闭模式，代表 SELinux 并没有实际运行。

(2) 查看 SELinux 当前的模式，下面命令显示出当前的模式为 Enforcing:

```
~]# getenforce
Enforcing
```

或者使用如下命令，如果 SELinux status 参数为 enabled 即为开启状态:

```
~]#/usr/sbin/sestatus -v
SELinux status:                enabled
```

(3) 在打开模式下查看 SELinux 模式 (Policy):

```
~]# sestatus
SELinux status:                enabled          #是否启动 SELinux
SELinuxfs mount:              /selinux        #SELinux 的相关文件挂载点
Current mode:                  enforcing         #目前的模式
Mode from config file:         enforcing         #设定指定的模式
Policy version:                21
Policy from config file:       targeted         #目前的政策为谁服务
```

(4) 临时关闭 (不用重启机器):

```
~]# setenforce 0                #设置 SELinux 为 permissive 模式
~]# setenforce 1                #设置 SELinux 为 enforcing 模式
```

(5) 通过配置文件调整 SELinux 的参数

```
~]# vi /etc/selinux/config
SELINUX=enforcing              #可以设置为三个值: enforcing|disabled|permissive
SELINUXTYPE=targeted           #目前仅有 targeted 与 strict
```

7.1.2 网络时间协议 (NTP) 服务的设置

时间的同步既可以通过命令在本地设置, 也可以通过 NTP 服务与远程的一个 NTP 服务进行同步。例如, 一次性的同步, 可以如下操作。

首先检查要使用的 NTP 服务器是否可访问:

```
~]# ntpdate -q server_address
```

例如:

```
~]# ntpdate -q 0.rhel.pool.ntp.org
```

当发现了一个可以使用的服务器后, 运行 `ntpdate` 命令, 后面跟一个或多个服务器的地址:

```
~]# ntpdate server_address...
```

例如:

```
~]# ntpdate 0.rhel.pool.ntp.org 1.rhel.pool.ntp.org
```

除非当前控制台显示一个错误信息, 否则, 时间现在被同步了。可以通过 `date` 命令, 但后面不跟任何参数来检查同步后的系统时间。

如果一个或多个服务一直要使用正确的时间，那么需要在系统启动时进行时间的同步，可以运行如下命令来完成：

```
~]# chkconfig ntpdate on
```

如果和时间服务器的同步总是失败，例如，在/var/log/boot.log 系统日志文件中发现错误消息，需要把下列一行加入到/etc/sysconfig/network 文件中：

```
NETWORKWAIT=1
```

然而更方便的方式是运行 ntpd 守护进程，在系统启动时自动同步时间。打开 NTP 服务的配置文件/etc/ntp.conf，如果不存在该文件，也可以创建一个：

```
~]# nano /etc/ntp.conf
```

然后在 ntp.conf 文件中增加或修改，主要部分为一列 NTP 服务的地址。如果使用的是 Red Hat Enterprise Linux 6，下面是该文件的一个例子：

```
server 0.rhel.pool.ntp.org iburst
server 1.rhel.pool.ntp.org iburst
server 2.rhel.pool.ntp.org iburst
server 3.rhel.pool.ntp.org iburst
```

每行尾部的 iburst 指令是为了加速初始的同步。在 Red Hat Enterprise Linux 6.5 中，它会被自动增加。在增加或编辑完毕后，需要设置正确的访问权限，只有本机可以访问：

```
restrict default kod nomodify notrap nopeer noquery
restrict -6 default kod nomodify notrap nopeer noquery
restrict 127.0.0.1
restrict -6 ::1
```

保存完内容，退出编辑器后，需要重启 NTP 守护进程：

```
~]# service ntpd restart
```

确保 ntpd 在系统启动时开始运行：

```
~]# chkconfig ntpd on
```

7.1.3 SSH 无密码登录

在服务器之间建立信任关系，对于监控服务器管理，特别是 OpenStack 云平台管理，以及块迁移等功能很有必要。这里介绍 SSH 的无密码登录建立过程。

(1) 生成公私钥文件。

本机 IP:192.168.1.6

```
~]# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
/root/.ssh/id_rsa already exists.
Overwrite (y/n)? y (原来已有上次生成的 key 文件, 所以系统提示是否覆盖原来的文件)
Enter passphrase (empty for no passphrase): (直接回车无须输入密钥)
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
04:37:13:2a:4b:10:af:c1:2b:03:3f:6b:27:ce:b9:62 root@xiaobin
```

(2) 查看生成的文件:

```
~]# cd .ssh/
~]# ll
-rw----- 1 root root 883 Apr 25 17:51 id_rsa
-rw-r--r-- 1 root root 221 Apr 25 17:51 id_rsa.pub
-rw-r--r-- 1 root root 442 Apr 25 17:37 known_hosts
```

其中, id_rsa 是密钥文件, id_rsa.pub 是公钥文件。

(3) 复制公钥文件到目标主机:

```
~]# scp id_rsa.pub 192.168.1.4:/root/.ssh/192.168.1.6
root@192.168.1.4's password:
id_rsa.pub                                100% 221    0.2KB/s  00:00
```

这里把公钥文件取名为本机的 IP 地址就是为了以后和更多的机器建立信任关系而不发生混淆。

(4) 目标机的操作。

登录到 192.168.1.4 机器

```
~]# cd .ssh/
~]# cat 192.168.1.6 >> authorized_keys
```

(5) 无密码登录。

回到 192.168.1.6 机器:

```
~]# ssh 192.168.1.4
Last login: Wed Aug 8 12:14:42 2007 from 192.168.1.6
```

通过以上步骤即可, 里面偶尔涉及权限问题。一般 .ssh 文件夹是 755, authorized_keys

文件为 600 或者 644。

当第一次登录到目标主机时，目标主机将发送一个 `fingerprint` 给当前主机。下面命令行参数可以进一步关闭此显示：

```
ssh -O StrictHostKeyChecking=no root@9.125.13.254
```

复制目标主机的 `id_rsa.pub` 和 `authorized_keys` 到各个主机，则同一套公钥/私钥可以访问多台主机。

7.1.4 自动运行定制化程序

在搭建一个可用的生产环境或者正式环境时，往往需要运行一些自己的逻辑，这些逻辑很多是在第一次系统启动时执行的，也有些是需要周期性执行的。掌握下列技巧会简化很多这类工作。

(1) 开机启动时自动运行程序

Linux 加载后，它将初始化硬件和设备驱动，然后运行第一个进程 `init`。`init` 根据配置文件继续引导启动过程启动其他进程。通常情况下，修改放置在 `/etc/rc` 或 `/etc/rc.d` 或 `/etc/rc?.d` 目录下的脚本文件，可以使 `init` 自动启动其他程序。例如，编辑 `/etc/rc.d/rc.local` 文件，在文件最末加上一行 “`xinit`” 或 “`startx`”，可以在开机启动后直接进入 X-Window。

(2) 登录时自动运行程序

用户登录时，`bash` 首先自动执行系统管理员建立的全局登录脚本：`/etc/profile`。然后 `bash` 在用户起始目录下按顺序查找三个特殊文件中的一个：`/.bash_profile`、`/.bash_login`、`/.profile`，但只执行最先找到的一个。

因此，只需根据实际需要在上述文件中加入命令就可以在用户登录时自动运行某些程序（类似于 DOS 下的 `Autoexec.bat`）。

(3) 退出登录时自动运行程序

退出登录时，`bash` 自动执行个人的退出登录脚本 `/.bash_logout`。例如，在 `/.bash_logout` 中加入命令 “`tar -cvzf c.source.tgz *.c`”，则在每次退出登录时自动执行 “`tar`” 命令备份 `*.c` 文件。

(4) 定期自动运行程序

Linux 中有一个称为 `crond` 的守护程序，其主要功能是周期性地检查 `/var/spool/cron` 目录下的一组命令文件的内容，并在设定的时间执行这些文件中的命令。用户可以通过 `crontab` 命令来建立、修改、删除这些命令文件。

例如，建立文件 `crondFile`，内容为“00 9 23 Jan × HappyBirthday”，运行 `crontab cronFile` 命令后，每当 1 月 23 日上午 9:00，系统会自动执行“HappyBirthday”的程序（“×”表示忽略星期几的信息）。

（5）定时自动运行程序一次

定时执行命令 `at` 与 `crond` 类似（但它只执行一次）：命令在给定的时间执行，但不自动重复。`at` 命令的一般格式为：`at [-f file] time`，在指定的时间执行 `file` 文件中所给出的所有命令。也可直接从键盘输入命令：

```
~]# at 12:00
at>mailto Roger -s "Have a lunch" < plan.txt
at>Ctrl-D
Job 1 at 2000-11-09 12:00
```

2000-11-09 12:00 时自动发一标题为“Have a lunch”、内容为 `plan.txt` 文件内容的邮件给 Roger。

7.1.5 简单备份

出于安全，我们经常会对计算机里存储的重要数据进行备份，如 `/etc`、`/boot`、`/root` 这些目录中的数据。

以备份 `/etc` 目录为例，首先创建备份脚本：

```
#!/bin/bash
day='date +%Y-%m-%d'
cd /opt/backup/etc
tar -zcf etc.${day}.tar.gz /etc/* 2> /dev/null
```

命令为 `etcbakup.sh`，放在 `/opt/tools/bin` 目录下。

然后运行 `vi /etc/crontab`，添加一行：

```
15 18 * * * root /opt/tools/bin/etcbakup.sh
```

这样系统会在每天下午 6 点 15 分的时候将 `etc` 整个目录备份至 `/opt/backup/etc` 目录下，并自动以当天的时间来命名。

7.2 网络

7.2.1 ifconfig 命令使用及结果分析

任何一个系统，网络不通都是非常严重的问题，而 `ifconfig` 是 Linux 里最常用的命令之一。在 Linux 下，第一块、第二块网卡的命名规律为 `eth0`、`eth1`。`lo` 为环回接口，它的 IP 地址固定为 127.0.0.1，掩码 8 位，代表当前主机本身。在有些系统里，网卡命名为 `em1`、`em2`、……

(1) 使用 `ifconfig` 查看网卡的信息：

```
ifconfig [Interface]
```

`Interface` 是可选项，如果不加此选项，则显示系统中所有网卡的信息；如果添加此选项，则显示所指定的网卡信息。

例如：`ifconfig eth0`

```
eth0 Link encap:Ethernet HWaddr 00:0C:29:F3:3B:F2
      inet addr:192.168.0.10 Bcast:192.168.0.255 Mask:255.255.255.0 UP
BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  RX packets:78 errors:0 dropped:0
overruns:0 frame:0  TX packets:104 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:100
      RX bytes:11679 (11.4 Kb)
      TX bytes:14077 (13.7 Kb)
      Interrupt:10 Base address:0x1080
```

第一行信息显示为：连接类型为 Ethernet（以太网）HWaddr（硬件 mac 地址）。

第二行信息显示为：网卡的 IP 地址、子网、掩码。

第三行信息显示为：UP（代表网卡开启状态），RUNNING（代表网卡的网线被接上），MULTICAST（支持组播），MTU:1500（最大传输单元：1500 字节）。

第四、五行信息显示为：接收、发送数据包情况统计。

第七、八行显示为：接收、发送数据字节数统计信息。

(2) 使用 `ifconfig` 配置网卡。

配置网卡的 IP 地址：

```
ifconfig eth0 192.168.0.1 netmask 255.255.255.0
```

在 `eth0` 上配置 192.168.0.1 的 IP 地址及 24 位掩码。如果想再在 `eth0` 上配置一个 192.168.1.1/24 的 IP 地址，可以用下面的命令：

```
ifconfig eth0:0 192.168.1.1 netmask 255.255.255.0
```

这时再用 `ifconfig` 命令查看，就可以看到两个网卡的信息了，分别为 `eth0` 和 `eth0:0`。若还想再增加 IP，那网卡的命名就会是 `eth0:1`、`eth0:2`、……

还可以用该命令来配置网卡硬件地址，配置网卡的硬件地址为：

```
ifconfig eth0 hw ether xx:xx:xx:xx:xx:xx
```

这样即可更改网卡的硬件地址，此时就可以骗过局域网内的 IP 地址绑定了。

(3) 将禁用网卡：

```
ifconfig eth0 down
```

(4) 启用网卡：

```
ifconfig eth0 up
```

`ifconfig` 命令的功能很强大，还可以设置网卡的 MTU、混杂模式等。

需要注意的是，用 `ifconfig` 命令配置的网卡信息，在机器重启后，配置就不存在了。要想将上述配置信息保存起来，就需要修改网卡的配置文件。

7.2.2 静态 IP 地址的配置

如果需要设置一台主机的静态地址，并且在系统重启后依然生效，则需要修改系统配置文件。在 RHEL/Redhat/Fedora/CentOS Linux 系统里，`eth0` 的配置文件在 `/etc/sysconfig/network-scripts/ifcfg-eth0` 里，而 `eth1` 的配置文件在 `/etc/sysconfig/network-scripts/ifcfg-eth1` 里，依次顺推。

在下面的举例中，我们将使用下列 IP 地址、默认网关和 DNS 服务器地址：

```
IP address: 192.168.1.10
Netmask: 255.255.255.0
Hostname: server1.try.com
```

```
Domain name: try.com
Gateway IP: 192.168.1.254
DNS Server IP # 1: 192.168.1.254
DNS Server IP # 2: 8.8.8.8
DNS Server IP # 3: 202.54.2.5
```

要使用静态地址，需要编辑/etc/sysconfig/network 文件如下：

```
# cat /etc/sysconfig/network
```

```
Sample static ip configuration:
```

```
NETWORKING=yes
HOSTNAME=server1.try.com
GATEWAY=192.168.1.254
```

然后编辑/etc/sysconfig/network-scripts/ifcfg-eth0 文件，输入：

```
# cat /etc/sysconfig/network-scripts/ifcfg-eth0
```

```
Sample static ip configuration:
```

```
# Intel Corporation 82573E Gigabit Ethernet Controller (Copper)
DEVICE=eth0
BOOTPROTO=static
DHCPCLASS=
HWADDR=00:30:48:56:A6:2E
IPADDR=192.168.1.10
NETMASK=255.255.255.0
ONBOOT=yes
```

编辑/etc/resolv.conf 来设置 DNS 服务，输入：

```
# cat /etc/resolv.conf
```

```
Sample static IP configurations:
```

```
search try.com
nameserver 192.168.1.254
nameserver 8.8.8.8
nameserver 202.54.2.5
```

最后，需要重新启动网络服务：

```
# /etc/init.d/network restart
```

如果需要验证 eth0 已经有了我们赋予的新的静态 IP 地址，可输入：

```
# ifconfig eth0
```

```
# route -n
# ping 192.168.1.254
# ping google.com
```

7.2.3 网卡绑定

网卡绑定 (Bonding) 就是把同一台服务器上的多个物理网卡, 通过软件绑定成一个虚拟的网卡 (表现为同一个 IP/MAC), 以达到分散负载的目的。虚拟后所有 `eth(#)` 网卡的 IP 和 MAC 将会变成完全相同, 系统会多出一个 `bond0` 的虚拟网卡。对于外部网络而言, 这台服务器只有一个可见的网卡。对于任何应用程序, 以及本服务器所在的网络, 这台服务器只有一个网络链接或者说只有一个可以访问的 IP 地址。

之所以要利用绑定技术, 除了利用多网卡同时工作来提高网络速度以外, 还可以通过绑定实现不同网卡之间的负载均衡 (Load balancing) 和网卡冗余 (Fault tolerance)。

要实现网卡绑定, 硬件需要至少两块以上的网卡, 但不一定要同一款的网卡。

绑定提供了下面几种模式, 但要注意的是其中的 `active-backup(mode=1)`、`balance-tlb(mode=5)`、`balance-alb(mode=6)` 是不需要交换机特别设定来支持的。

其他模式如 `round-robin(mode=0)`、`XOR(mode=2)`、`broadcast(mode=3)`、`802.3ad policies(mode=4)` 则需要特别的交换机来支持。

如果在设定绑定时没有指定, 将会默认使用 `mode 0 balance-rr(round-robin)`。

(1) `balance-rr` 或者 0。

假设将两张网络卡设成 `Balance`, 则绑定包含容错功能。如果一张网卡故障, 而另一张依然工作, 则整个网络服务不受影响。

(2) `active-backup` 或者 1。

假设有两张网卡, 一张是 `PRIMARY`, 另一张是 `STANDBY`。这时网络流量只在 `PRIMARY` 上跑, 当 `PRIMARY` 网卡出现故障时, `STANDBY` 网卡会自动启用并变成 `PRIMARY`, 如果原来的 `PRIMARY` 从故障中恢复, 则会自动变成 `STANDBY`。

此外的 5 种模式, 可以参考 `/usr/src/linux-[version]/Documentation/networking/bonding.txt`。

设置绑定, 首先需要检查内核是否支持。系统需要加载 `bonding` 驱动, 在 `Linux 2.4.12` 之后的核心中, 提供了绑定的驱动, 可以支持把多个网卡集合在一起, 当作一个网卡来使

用。

```
cat /boot/config-kernel-version |grep -i bonding
CONFIG_BONDING=m
```

M——将该功能编译成可以在需要时动态插入到内核中的模块。

如果需要使用绑定，我们需要在/etc/modprobe.conf 中加入一行，这样才可以在设置了绑定后，在系统启动的时候自动加载 bonding 的驱动程序：

```
vi /etc/modules.conf
-----
.....
alias bond0 bonding
options bond0 miimon=100 mode=0
-----
```

也可以直接在下面的命令行中加上模块的参数：

```
modprobe bonding miimon=100 mode=0
miimon=100
```

其中，miimon 是指多长时间要检查网络一次，单位是 ms(毫秒)，这里的 100 是 100ms，即 0.1s，意思是假设其中有一条网络断线，会在 0.1s 内自动发现。

mode=0 balance-rr(round-robin)，这个模式主要是做负载均衡(load-balancing)。

建立虚拟网卡 bond0，需要生成下列文件：

```
cat /etc/sysconfig/network-scripts/ifcfg-bond0
-----
DEVICE=bond0
ONBOOT=yes
BOOTPROTO=static
IPADDR=192.168.0.30
NETMASK=255.255.255.0
GATEWAY=192.168.0.254
USERCTL=no
-----
```

在绑定涉及的所有网络设置文件中，都有 SLAVE 和 MASTER 的定义。例如，如果要让 eth0 (SLAVE) 和 eth1 (SLAVE) 组成 bond0 (MASTER)，它们对应的设置 (ifcfg-eth0 和 ifcfg-eth1) 就要按照下面的内容进行修改：

```
cat /etc/sysconfig/network-scripts/ifcfg-eth0
-----
DEVICE=eth0
```

```
ONBOOT=yes
BOOTPROTO=static
MASTER=bond0
SLAVE=yes
```

```
-----
cat /etc/sysconfig/network-scripts/ifcfg-eth1
```

```
-----
DEVICE=eth1
ONBOOT=yes
BOOTPROTO=static
MASTER=bond0
SLAVE=yes
-----
```

设置完成后，重启系统或是重新启动网络服务，都可以重新启动系统来使 **bond0** 生效：

```
service network restart
```

这样，我们就完成了 **bonding** 的设置。

接下来可以查看目前 **bonding** 的状态：

```
cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v2.6.0 (January 14, 2004)
Bonding Mode: load balancing (round-robin)
MII Status: up
MII Polling Interval (ms): 0
Up Delay (ms): 0
Down Delay (ms): 0
Slave Interface: eth0
MII Status: up
Link Failure Count: 0
Permanent HW addr: 00:0a:5e:3e:77:41
Slave Interface: eth1
MII Status: up
Link Failure Count: 0
Permanent HW addr: 00:0a:5e:3e:7c:30
```

还可以查看目前网络的状态，运行 **ifconfig**：

```
bond0 Link encap:Ethernet HWaddr 00:0A:5E:3E:77:41
inet addr:192.168.0.30 Bcast:192.168.0.255 Mask:255.255.255.0
UP BROADCAST RUNNING MASTER MULTICAST MTU:1500 Metric:1
RX packets:86009442 errors:0 dropped:0 overruns:0 frame:0
TX packets:414270098 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:3749356653 (3575.6 Mb) TX bytes:124489446 (118.7 Mb)
```

```
eth0 Link encap:Ethernet HWaddr 00:0A:5E:3E:77:41
inet addr:192.168.0.30 Bcast:192.168.0.255 Mask:255.255.255.0
UP BROADCAST RUNNING SLAVE MULTICAST MTU:1500 Metric:1
RX packets:6119846 errors:0 dropped:0 overruns:0 frame:0
TX packets:23960950 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:1179793455 (1125.1 Mb) TX bytes:3732737688 (3559.8 Mb)
Interrupt:24
```

```
eth1 Link encap:Ethernet HWaddr 00:0A:5E:3E:77:41
inet addr:192.168.0.30 Bcast:192.168.0.255 Mask:255.255.255.0
UP BROADCAST RUNNING SLAVE MULTICAST MTU:1500 Metric:1
RX packets:5826277 errors:0 dropped:0 overruns:0 frame:0
TX packets:23959600 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:1126390337 (1074.2 Mb) TX bytes:3719587744 (3547.2 Mb)
Interrupt:48
```

bond0 是虚拟网卡，而且 eth0、eth1 的 IP 和 MAC 都和 bond0 一样。

如果是临时使用绑定的话，可以直接用指令方式实现，但是系统重启以后绑定也随即消失：

```
modprobe bonding
ifconfig eth0 down
ifconfig eth1 down

ifconfig bond0 ipaddress
```

让 eth0 (SLAVE) 和 eth1 (SLAVE) 成为 bond0 (MASTER) 的成员，在绑定中的所有网络的设置文件中，都要有 SLAVE 和 MASTER 的定义：

```
ifenslave bond0 eth0
ifenslave bond0 eth1
```

7.2.4 网桥模式的配置

在搭建虚拟机的时候，我们最终关心的网卡是 eth0、br0、vnet0、vnet1……其中，eth0 是原来的物理网卡，br0 是虚拟出来的网桥设备，而 vnet[n] 就是网桥映射到虚拟机里面用到的网卡。当配置完了之后，如果使用 ifconfig 查看，eth0 是没有 IP 地址的，而 br0 仿佛是虚拟出来的一个原来 eth0 的访问接口，它具有 IP 地址，可以代替原来的 eth0 被访问，而 vnet[n] 的地址可以在建立虚拟机之后在虚拟机里面配置，桥接后可以被外部所访问。

当将已有的物理网卡添加到网桥，此时物理网卡工作于混杂模式，所以不再需要 IP 地址了，因为网桥是工作在链路层的。br0 就提供了 IP 地址，来模拟原来的物理网卡的访问接口。混杂模式的工作原理是，在以太网里，数据包是在整个子网里面广播发送的，当网卡发现这个数据帧不是发给自己的也不是广播包的时候，就直接把包丢弃，而不传送到上层内核去处理；而当网卡处于混杂模式的时候，网卡就不会丢弃数据帧，而是全部向上提交到内核，让内核去处理这些数据帧结构。通常情况下，混杂模式是用来进行网络调试的，很多网络监听工具也是使得网卡工作于这个模式。

在一个配置较好的服务器里，可以运行多个虚拟机，通过创建网桥，这些虚拟机都可以访问局域网或被局域网所访问。这些虚拟机可以有各种用途，为工作提供许多便利。下面是搭建网桥的基本方法之一。

一个服务器中，eth0 为物理网络接口，而 br0 为桥接。

查看 ifcfg-br0 配置文件，有下列信息：

```
~]#cat ifcfg-br0
DEVICE=br0                //网卡接口名称
TYPE=Bridge               //网卡类型
BOOTPROTO=static          //启动静态地址协议（bootp 协议和 dhcp 协议）
NM_CONTROLLED=no          //是否允许 NetworkManager 管理
ONBOOT=yes                //启动系统时是否自动加载
IPADDR=192.168.xx.xxx      //网卡 IP 地址
NETMASK=255.255.255.0      //网络掩码
GATEWAY=192.168.xx.xx      //网卡网关地址
```

查看 ifcfg-eth0 配置文件有下列信息：

```
~]##cat ifcfg-eth0
DEVICE=eth0                //网卡接口名称
TYPE=Ethernet              //网卡类型
BOOTPROTO=none             //启动地址协议 static 静态，bootp 协议和 dhcp 协议
HWADDR=8C:89:A5:65:B8:3D    //网卡 MAC 地址
NM_CONTROLLED=no           //是否启动 NetworkManager 管理
ONBOOT=yes                 //启动系统是否自动加载
BRIDGE=br0                 //网桥名字，名字可以自定
```

重启网络让 br0 和 br1 桥生效：

```
~]#service network restart
Shutting down interface br0: [ OK ]
Shut Shutting down interface br0: [ OK ]
Shutting down interface eth1: [ OK ]
Shutting down interface eth1: [ OK ]
```



```
Shutting down loopback interface:      [ OK ]
Bringing up loopback interface:         [ OK ]
Bringing up interface eth0:             [ OK ]
Bringing up interface br0:              [ OK ]
Bringing up interface eth1:             [ OK ]
Bringing up interface br1:              [ OK ]
```

查看全部桥接信息:

```
~]# brctl show
bridge name      bridge id                STP enabled    interfaces
br0              8000.8c89a565b83d        no             eth0
br1              8000.00e04cefb385        no             eth1
virbr0           8000.5254001daa20        yes            virbr0-nicvirbr0
```

7.2.5 Access、Hybrid 和 Trunk 三种模式

tag、untag 以及交换机的各种端口模式是网络工程技术人员调试交换机时接触最多的概念了。untag 就是普通的以太网报文,普通 PC 机的网卡就可以识别这样的报文进行通信;tag 报文结构的变化是在源 MAC 地址和目的 MAC 地址之后,加上了 4 字节的 VLAN 信息,也就是 VLAN tag 头;一般来说,对于这样的报文,普通 PC 机的网卡是不能识别的。

带 802.1Q 的数据帧是在标准以太网帧上插入了 4 个字节的标识,其中包括:

(1) 2 个字节的协议标识符(TPID),当前置为 0x8100 的固定值,说明该帧带有 802.1Q 的标记信息。

(2) 2 个字节的标记控制信息(TCI),包含了三个域。

Priority 域,占 3 比特,表示报文的优先级,取值为 0~7。7 为最高优先级,0 为最低优先级。该域被 802.1p 所采用。

规范格式指示符(CFI)域,占 1 比特,0 表示规范格式,应用于以太网;1 表示非规范格式,应用于令牌环。

VLAN ID 域,占 12 比特,用于标示 VLAN 的归属。

以太网交换机端口有三种链路类型:Access、Hybrid 和 Trunk。

(1) Access 类型的端口只能属于某一个 VLAN,一般用于连接计算机的端口。

(2) Trunk 类型的端口可以允许多个 VLAN 通过,可以发送和接收多个 VLAN 的报文,

一般用于交换机之间的端口互联。

(3) Hybrid 类型的端口可以允许多个 VLAN 通过,可以接收和发送多个 VLAN 的报文,可以用于交换机之间连接,也可以用于连接用户的计算机。

Hybrid 端口和 Trunk 端口在接收数据时,处理方法是一样的,唯一不同之处在于发送数据时,Hybrid 端口可以允许多个 VLAN 的报文发送时不打标签,而 Trunk 端口只允许默认 VLAN 的报文发送时不打标签。

在这里引入了默认 VLAN 的概念。Access 端口只属于一个 VLAN,所以它的默认 VLAN 就是它所在的 VLAN,不用单独设置;Hybrid 端口和 Trunk 端口属于多个 VLAN,所以需要设置默认 VLAN ID。默认情况下,Hybrid 端口和 Trunk 端口的默认 VLAN 为 VLAN 1。如果设置了端口的默认 VLAN ID,当端口接收到不带 VLAN tag 的报文后,则将报文转发到属于默认 VLAN 的端口;当端口发送带有 VLAN tag 的报文时,如果该报文的 VLAN ID 与端口默认的 VLAN ID 相同,则系统将去掉报文的 VLAN tag,然后再发送该报文。

交换机端口出入数据的处理过程如下。

(1) Access 端口。

- **接收报文:** 收到一个报文,判断是否有 VLAN 信息,如果没有则打上端口的 PVID,并进行交换转发;如果有则直接丢弃。
- **发送报文:** 将报文的 VLAN 信息剥离,直接发送出去。

(2) Trunk 端口。

- **接收报文:** 收到一个报文,判断是否有 VLAN 信息,如果没有则打上端口的 PVID,并进行交换转发,如果有则判断该 Trunk 端口是否允许该 VLAN 的数据进入,如果可以则转发,否则丢弃。
- **发送报文,** 比较端口的 PVID 和将要发送报文的 VLAN 信息,如果两者相等则剥离 VLAN 信息,再发送;如果不相等则直接发送。

(3) hybrid 端口

- **接收报文:** 收到一个报文,判断是否有 VLAN 信息,如果没有则打上端口的 PVID,并进行交换转发;如果有则判断该 Hybrid 端口是否允许该 VLAN 的数据进入,如果可以则转发,否则丢弃(此时端口上的 untag 配置是不用考虑的,untag 配置只对发送报文时起作用)。

- 发送报文。

- 判断该 VLAN 在本端口的属性（`disp interface` 即可看到该端口对哪些 VLAN 是 untag，哪些 VLAN 是 tag）。
- 如果是 untag 则剥离 VLAN 信息，再发送；如果是 tag 则直接发送。

7.3 安装与打包技术

7.3.1 制作 RPM

RPM 可以帮助我们许多，极大地简化我们的工作，包括自动安装、升级和卸载安装组件等。RPM 还可以做许多事情，让我们的系统与最新的设计代码保持同步。RPM 被设计为可以非常容易被开发者用来制作许多种类的软件发行包。

1. RPM 的优点

(1) 跟踪初始来源

许多开发者可能用 RPM 来打包自己开发的软件，而另外许多人则用它来打包并非自己开发的、从互联网上下载的软件。因为这个原因，RPM 在设计时就考虑到这些，从而能够非常容易地处理“第三方软件”的问题。

当一个软件打包模块需要打包其他的软件时，这些其他软件通常是从互联网上获得的，并且是用如 GNU Zip 格式压缩的 tar。这是我们平常经常遇到或使用的一种打包形式。

如果打开 tar 文件，查看内容，会发现一些不同之处：应用程序可能是以源代码形式发布的，或者是二进制可执行形式，或者是前述二者的组合。

这些应用程序可能是用 make、imake，或者一个脚本来编译和打包的，或者全部纯粹是以手工方式来打包的。

这些应用可能需要在使用前做一些配置，可能使用的是 GNU configure、一个定制化的配置脚本，或者一些文件，在使用前需要根据当前的运行环境来编辑。

这些应用在设计编写时，就必须安装于某些目录，而这些目录或者在目标环境是不存在的，或者从应用的角度出发是有问题的。

这些应用可能并不支持目标环境，需要各种各样的修改来移植到目标环境去运行。

所以对于一个应用来说，从互联网上下载后，很少情况下可以直接打包然后到目标环境去运行，总是需要很多改动。

RPM 设计原则为跟踪软件的初始来源，而不管它是如何打包和配置的。这样一来，在打包过程的所有修改和初始来源是分离的，它们在分开的包或者补丁里。也许这一点看起来并不起眼，但是考虑一下开发过程，新的版本的应用发布了，而新版本里有一些对前面问题的补丁，这样整个补丁都可以追加到原来的安装里去。新补丁完成后通过一个 RPM 命令就可以打包并安装。如果补丁不能完全追加进去，至少可以追踪到需要做哪些事情才可以将新版本编译和打包。

如果用户有时对软件包做了定制，那么追踪来源的做法也很方便，他们可以很容易地发现哪些补丁是增加的，也可以很容易地增加自己的部分。另外一个好处是可以非常容易地跟踪一个软件包的多个来源。

为了完成上述目标，RPM 需要：

- ① 以 tar 文件存在的初始代码；
- ② 一些需要的补丁，让应用可以被创建；
- ③ 一些控制 RPM 打包创建流程的文件。

有了上述三项，可以很容易地在任意时间点去打包制作安装程序，可以跟踪多个版本中每个版本的上述三个组件，而不是成千上万个打了补丁的源文件。实际上，RPM 还可以创建一个源包，包含上述三个组件。这个源包按照 RPM 标准的命名规则，跟踪和保留重建一个发行包的所有信息。跟踪多个发行包的多个版本，仅仅变成一个保留多个合适的源软件包的过程。所有的东西都可以基于此重建。

(2) 容易制作

RPM 在设计时，就考虑到了处理各种来源的比较通用的第三方软件的打包发布，而不仅局限于一种特定具体的情况。下面的内容讲述了为什么 RPM 是一个非常通用而友好的软件打包发布系统。

- ① 可重现的打包过程。

开发者常见的情况是重现一个特定的打包过程，这也是许多部署了版本控制工具来管

理源代码系统的初衷。

RPM 不能和强大的版本管理工具相比,但是它将一个软件包来制作一个特定版本所需要的信息收集在一处。只要一个命令, RPM 可以打开软件包,抽取源文件,打补丁,执行打包过程,生成一个新的可用的软件包,而且每次生成的包内容都是一样的,因为打包需要的源文件都在一起。

② 无人值守的打包过程。

完整地创建一个软件包只需要一个命令。这个特点可以用来搭建一个自动化的打包制作过程,来一起制作上百个软件包。无论是制作一个应用的软件包,还是搭建整个系统所需要的几百个软件包,都可以用 RPM 来自动制作。

③ 多架构/操作系统支持。

很多时候,软件开发者需要将他们的软件移植到不同的操作系统上,而且很多时候,操作系统运行于不同的平台或硬件架构上。

RPM 可以支持各种架构和操作系统,可以很容易地制作多种操作系统/平台组合的软件包,软件包可以通过配置,来制作一个或多个架构/操作系统。唯一的限制来自于应用自身的可移植性。

(3) 用户友好

虽然前述都描述了 RPM 给开发者带来的好处,但是从使用者角度, RPM 也带来了许多便利。

① 容易升级。

对于开发者和用户来说,最头疼的方面是软件的升级,或者整个操作系统的升级。RPM 可以让升级变成只需要一步的操作。只需要一个命令,一个新的软件包安装就可以完成,而旧的软件包的不相容部分则会被移除。

② 智能的配置文件处理。

所有的应用程序都有配置文件,只是可能起了不同的名字,但是所有的配置文件都用来控制应用的行为。用户常常需要根据自己的环境来配置这些文件。如果在安装、升级或者删除软件包的过程中,他们所做的定制丢失,会是个非常麻烦的事情。

RPM 很好地处理了这个问题。它使用了 MD5 校验,可以确定对于一个配置文件,什

么样的动作合适。如果配置文件被用户修改，必须被替换，则保留一份备份。这样用户对配置的修改就不会丢失。

③ 强大的查询能力。

RPM 使用一个数据库来记录安装的文件。而这个数据库也提供了其他一些好处，如可以容易地获取各种信息。RPM 的查询命令可以很容易地帮助用户查询到各种需要的信息，诸如：

- 这些文件从哪里来？是哪个软件包的？
- 这个软件包是做什么的？
- 当前系统里安装了哪些软件包？

上述只是一些简单的例子，RPM 可以帮助用户获取各种需要的信息。

④ 安装容易验证。

另外一个 RPM 将信息保留在数据库的好处是，提供了一个便捷的途径来验证一个软件包被正确安装了。比如，利用这个能力，RPM 可以很容易地确定在 `/usr/bin` 目录下的哪些软件包被一个命令中的通配符给删除了。另外，RPM 的验证功能可以检测到哪些文件属性被修改了，如文件的权限、拥有者和大小等。

2. RPM 打包安装程序

下面介绍 RPM 创建软件包的流程。创建软件包的过程就和编译代码一样，有输入、一个执行任务的引擎和输出。

(1) 输入

RPM 打包过程使用到三种不同的输入。其中前两种输入在创建安装包的过程中是必须的，第三种在严格意义上说是可选的，但是除非你在打包自己的代码，否则很大可能是需要的。

① 源代码。

源代码总是需要的。如果针对第三方代码，源代码需要和原来发布时的形式一样，也许是一个压缩的 `tar` 文件。RPM 也可以处理其他种类的归档文件，但是需要一点前期工作。

任何情况下，不要尝试去修改任何源代码，如果根据第三方代码来创建安装包，意味着源文件拿到时是什么样子的，还是保留原来的样子。如果是自己的软件，这个决定权在自己。

② 补丁。

前面一直在强调不要尝试去修改源代码结构,因为 RPM 可以帮助你将一个补丁加到以前的安装系统里。通常,补丁包括下面一些类型。

- 补丁是针对目标环境的一个问题。这可能包括修改 makefile 文件来安装应用程序到某个合适的目录;或者解决一个跨平台的冲突,例如替换一个系统的调用。
- 补丁也可以是在安装过程中根据配置步骤来创建的一些文件。很多时候,或者必须手工编辑配置文件,或者用脚本来设置一些值。其他情况还包括在编译前,先运行一个配置工具等。

创建补丁:

上述补丁的类型虽然听起来很抽象,其实制作起来比较简单。下面是一些步骤。

- 解开源文件包。
- 对顶层目录重命名,例如,以“.orig”结尾,这是源软件包的原始副本。
- 再次解开软件包。

再次解开的软件包目录是需要创建安装包的地方。

有时候,在软件安装包制作过程中不需要进行任何修改,但如果需要修改时,必须生成一些补丁。此时,可以用 diff 命令对修改过的目录和原来目录进行递归的比较,就可以得到补丁的部分。

③ 规格文件

规格文件是 RPM 打包过程的关键部分。和 makefile 的机理一样,它包含 RPM 制作安装包的必要信息和让 RPM 如何打包的指令。另外,规格文件也说明了哪些文件是需要被打包的,以及安装的目录等信息。

规格文件的格式比较复杂,然而,它分成几个部分,因此容易维护和处理。

a) 前置部分。

前置部分包含了当用户查询包信息时需要显示的部分,包括包里软件功能的描述、版本号等。另外还包含源代码的行数、显示在图形界面的图标等信息。

b) 准备部分。

准备部分包含真实打包过程开始的部分,包含一些在软件打包开始前的准备工作。一

般来说，任何在打包开始前需要执行的动作，都可以放在这个部分。一般都放置一些将源代码解压的代码。

这些代码通常是些普通的 Shell 脚本，然而，RPM 还定义了一些宏。其中有些宏用来解压压缩文件，然后进入某个目录，另外的宏则将补丁加入打开的源文件。

c) 创建过程。

创建过程包含一些 Shell 脚本，执行任何编译代码过程的一些命令。这个部分可能只是一个简单的 `make` 命令，或者更复杂点的一段脚本。现在很多软件都是用 `make` 来执行创建过程，所以这个部分没有定义宏。

d) 安装过程。

安装过程也包含一些 Shell 脚本，执行真实安装的任务。如果软件作者加入了安装目标，这段将只包括一个 `make install` 命令。否则，需要加入一些如 `cp`、`mv`，或者 `install` 等命令来完成安装。

e) 安装和卸载脚本。

这部分包括当真实安装或卸载软件包时，运行于用户目标系统的脚本。RPM 执行这些脚本的时间可以是：

- 在软件包开始安装前；
- 在软件包安装后；
- 在软件包被删除前；
- 在软件包被删除后。

一个例子是软件包里包括共享库。在这种情况下，在包安装或者卸载后，需要运行 `ldconfig` 命令。另外情况还包括：如果一个包中包含 Shell，则 `/etc/shells` 需要在安装或卸载时被更新。

f) 验证脚本。

这是另外一种在用户环境执行的脚本。它在 RPM 验证软件包是否被正确安装时使用。虽然 RPM 做了很多验证工作，但如果 RPM 无法做到的时候，则需要额外的脚本来完成这些验证。

g) 清除过程。

在创建安装包之后，往往还需要一个脚本来清除创建环境。不过通常情况下，RPM 很好地清除了中间环境，所以这个脚本很少用到。

h) 文件列表。

最后一部分是组成软件包的文件列表。另外，有一些宏来控制文件的属性，标识出哪些文件是文档，哪些包含配置信息等。文件列表非常重要，如果它缺失，则不会生成很好的安装包。

(2) RPM 引擎

所有这些过程的核心是 RPM，它在创建安装包过程中执行一系列步骤。

- ① 执行规格文件中准备阶段定义的命令和宏。
- ② 检查文件列表的内容。
- ③ 执行创建阶段定义的命令和宏。
- ④ 执行安装阶段定义的命令和宏，文件列表定义的宏也在这个阶段执行。
- ⑤ 生成二进制的安装包。
- ⑥ 生成源代码安装包。

通过使用 RPM 命令的不同选项，创建过程可以在上述过程的任意步骤停下来。这使得初始的创建过程变得容易，可以看到在执行下一步之前，是否每步都执行正确。

(3) 输出

创建过程的产物是一个源代码的包和一个二进制代码的包。

① 源代码包。

源代码包是一个特殊格式的归档文件，包含下列文件。

- 原来压缩的 tar 文件；
- 规格文件；
- 补丁文件。

因为源文件包里包含了可以生成二进制安装包的所有内容，所以可以将这个包分发出给各种使用者；另外，这个包也可以用来重新生成一个特定版本的包。

② 二进制包。

二进制包是整个打包过程的产物，也是用户用到的部分。它包含组成应用的各个文件，还包括安装或卸载时需要的各种信息。

7.3.2 Kickstart 快速安装

1. 什么是 Kickstart

许多系统管理员使用自动化的安装方法来安装 Red Hat/CentOS ISO，特别是大规模安装的情况下。为了满足这种需求，Red Hat 创建了 Kickstart 安装方法。通过使用 Kickstart，系统管理员可以创建一个文件，这个文件包含了在典型的安装过程中所遇到的系统选项的答案。

Kickstart 文件可以存放于单一的服务器上，或者就在被安装的镜像文件中，在安装过程中被安装的机器所读取。这种安装方法可以支持使用单一 Kickstart 文件在多台主机上安装 Red Hat/CentOS Linux，对于网络和系统管理员来说是个理想的选择。

2. 如何执行 Kickstart 安装

Kickstart 安装可以使用本地光盘、本地硬盘驱动器，或通过 NFS、FTP、HTTP 等来执行。使用 Kickstart 之前要进行如下准备。

- 创建一个 Kickstart 文件。
- 创建有 Kickstart 文件的引导介质或者使这个文件在网络上可用。
- 准备安装树。
- 开始 Kickstart 安装。

3. 创建 Kickstart 文件

Kickstart 文件是一个简单的文本文件，它包含了一个项目列表，每个项目由一个关键字来识别。使用者可以用「Kickstart 配置」应用程序创建它，或是自己从头编写。Red Hat Linux 安装程序也根据安装过程中用户做的选择，创建一个简单的 Kickstart 文件。这个文件在安装后被写入到/root/anaconda-ks.cfg，可以用任何能够把文件保存为 ASCII 文本的文本编辑器或字处理器来编辑它。

在创建 Kickstart 文件时需要注意下列问题。

(1) 每节必须按顺序指定，除非特别申明，每节内的项目不必按序排列。小节的顺序如下。

- 命令部分，这里应该包括必需的选项。
- %packages 部分，这部分选择需要安装的软件包。
- %pre 和 %post 部分，这两个部分可以按任何顺序排列而且不是必需的。

(2) 不必需的项目可以被省略。

(3) 如果忽略任何必需的项目，安装程序会提示用户输入相关项目的选择，就像用户在典型的安装过程中所遇到的一样。一旦用户进行了选择，安装会以非交互的方式 (unattended) 继续 (除非找到另外一个没有指定的项目)。

(4) 以井号 (“#”) 开头的行被当作注释行并被忽略。

(5) 对于 Kickstart 升级，下列项目是必需的。

- 语言。
- 安装方法。
- 设备规格 (如果这个设备是在安装过程中所需要的)。
- 键盘设置。
- upgrade 关键字。
- 引导装载程序配置。

(6) 如果任何其他的项目被指定为 upgrade，这些项目将被忽略 (注意这包括了软件包选择)。

4. Kickstart 命令选项

在这里我们只列出最常用的选项。基本上，在安装界面里可以选择输入的部分，在这里都有相应的指令来对应。下面的选项可以放入 Kickstart 文件。

(1) 安装类型

- install (可选)

告诉系统安装全新的系统，而不是在现有系统上升级。这是默认的模式，必须指定安装的类型，如 CD、ROM、本地硬盘、NFS 服务器或 URL (FTP 或 HTTP 安装)。install 命令和安装方法命令必须处于不同的行上。

(2) 安装方法

安装方法如上所述, 可以选择 CD、ROM、本地硬盘、NFS 服务器和远程 URL 来安装。下面是 URL 的用法。

- Url

通过 FTP 或 HTTP 从远程服务器上的安装树中安装, 例如, `url --url http://<server>/<dir>`, 或 `url --url ftp://<username>:<password>@<server>/<dir>`。

(3) 语言

- lang (必需)

设置安装过程中使用的语言, 以及系统的默认语言。例如, 要把语言设置为英语, 则 Kickstart 文件应该包含下面一行:

```
lang en_US
```

文件 `/usr/share/system-config-language/locale-list` 里每一行的第一个字段提供了一个有效语言代码的列表, 它是 `system-config-language` 软件包的一部分。文本模式的安装过程不支持某些语言 (主要是中文、日文、韩文和印度的语言)。如果用 `lang` 命令指定这些语言中的一种, 安装过程仍然会使用英语, 但是系统会默认使用指定的语言。

(4) 键盘

- keyboard (必需)

设置系统键盘类型, 通常为 `us`。

(5) 网络

- network (可选)

为系统配置网络信息。如果 Kickstart 安装不要求联网 (即不从 NFS、HTTP 或 FTP 安装), 就不需要为系统配置网络。如果安装要求联网而 Kickstart 文件里没有提供网络信息, 安装程序会假定从 `eth0` 通过动态 IP 地址 (BOOTP/DHCP) 来安装, 并且给安装完的系统配置动态分配的 IP 地址。`network` 选项为通过网络的 Kickstart 安装以及所安装的系统配置网络信息。

网络的参数包括:

--bootproto=, 可以选择 dhcp、bootp 或 static 中的一种。默认值是 dhcp。bootp 和 dhcp 被认为是相同的。static 方法要求在 Kickstart 文件里输入所有的网络信息, 即这些信息是静态的且在安装过程中和安装后是固定的。静态网络的设置行更为复杂, 因为必须包括所有的网络配置信息, 必须指定 IP 地址、网络、网关和名字服务器。

例如 (“\” 表示连续的行):

```
network --bootproto=static --ip=10.0.2.15 --netmask=255.255.255.0 \  
--gateway=10.0.2.254 --nameserver=10.0.2.1
```

--device=, 用来选择用于安装的特定的网络设备。注意, 除非 Kickstart 文件是一个本地文件 (如 ks=floppy), 否则--device=的使用是无效的。这是因为安装程序会配置网络来寻找 Kickstart 文件。

例如:

```
network --bootproto=dhcp --device=eth0  
--ip=, 要安装的机器的 IP 地址。  
--gateway=, IP 地址格式的默认网关。  
--nameserver=, 主名称服务器, IP 地址格式。  
--nodns, 不要配置任何 DNS 服务器。  
--netmask=, 安装的系统的子网掩码。  
--hostname=, 安装的系统的主机名。  
--ethtool=, 指定传给 ethtool 程序的网络设备的其他底层设置。  
--essid=, 无线网络的网络 ID。  
--wepkey=, 无线网络的加密密钥。  
--onboot=, 是否在引导时启用该设备。  
--class=, DHCP 类型。  
--mtu=, 该设备的 MTU。  
--noipv4=, 禁用此设备的 IPv4。  
--noipv6=, 禁用此设备的 IPv6。
```

(6) 安全

• rootpw (必需)

把系统的超级用户密码设置为<password>参数, 其语法格式为:

```
rootpw [--iscrypted] <password>
```

其中: --iscrypted, 如果该选项存在, 口令就会假定已被加密。

• selinux (可选)

在系统里设置 SELinux 状态。在 anaconda 里, SELinux 默认为 enforcing。其语法格式为:

```
selinux [--disabled|--enforcing|--permissive]
```

其中--enforcing, 启用 SELinux, 实施默认的 targeted policy。如果 Kickstart 文件里没有 selinux 选项, SELinux 将被启用并默认设置为--enforcing。

--permissive, 输出基于 SELinux 策略的警告, 但实际上不执行这个策略。

--disabled, 在系统里完全地禁用 SELinux。

- firewall (可选)

这个选项对应安装程序里的“防火墙配置”界面, 其语法格式为:

```
firewall --enabled|--disabled [--trust=] <incoming> [--port=]
```

--enabled 或者--enable, 启动防火墙。在这个主机上, 如果需要使用某些服务, 可以选择允许指定的服务穿过防火墙。

--disabled 或--disable, 不会配置任何 iptables 规则。

--trust=, 在此列出设备, 如 eth0 等。这允许所有经由这个设备的数据包通过防火墙。如果需要列出多个设备, 则使用--trust eth0 --trust eth1 的格式。

<incoming>, 使用以下服务中的一个或多个来替换, 从而允许指定的服务穿过防火墙: --ssh、--telnet、--smtp、--http、--ftp。

--port=, 可以用“端口:协议 (port:protocol)”格式指定允许通过防火墙的端口。

例如, 如果想允许 IMAP 通过防火墙, 可以指定 imap:tcp。还可以具体指定端口号码。例如, 如果要允许 UDP 分组在端口 1234 通过防火墙, 则输入 1234:udp。要指定多个端口, 需要用逗号将它们隔开。

(7) 时区

- timezone (可选)

把系统时区设置为<timezone>, 它可以是 timeconfig 列出的任何时区。其语法格式为:

```
timezone [--utc] <timezone>
```

其中: --utc, 如果存在, 系统就会假定硬件时钟被设置为 UTC (格林威治标准) 时间。

(8) 服务

- services (可选)

修改运行在默认运行级别下的默认的服务集。在 `disabled` 列表里列出的服务将在 `enabled` 列表里的服务启用之前被禁用。

参数包括：

`--disabled`，禁用用逗号隔开的列表里的服务。

`--enabled`，启用用逗号隔开的列表里的服务。

(9) 引导程序安装方式

- `bootloader` (必需)

指定引导装载程序如何被安装。对于安装和升级，这个选项都是必需的。

参数包括：

`--append=`，指定内核参数，要指定多个参数，需要使用空格分隔它们。

例如：

```
bootloader --location=mbr --append="hdd=ide-scsi ide=nodma"
```

`--driveorder`，指定在 BIOS 引导顺序中居首的驱动器。

例如：

```
bootloader --driveorder=sda, hda
```

`--location=`，指定引导记录被写入的位置，有效的值如下：`mbr` (默认)、`partition` (在包含内核的分区的第一个扇区安装引导装载程序) 或 `none` (不安装引导装载程序)。

`--password=`，如果使用 GRUB，把 GRUB 引导装载程序的密码设置到这个选项指定的位置。该功能被用来限制对可以传入任意内核参数的 GRUB Shell 的访问。

`--md5pass=`，如果使用 GRUB，这和 `--password=` 类似，只是密码已经被加密。

`--upgrade`，升级现存的引导装载程序配置，保留其中原有的项目。该选项仅可用于升级。

(10) 磁盘分区处理

- `clearpart` (可选)

在创建新分区之前，从系统上删除分区。默认不会删除任何分区。

注意：如果使用了 `clearpart` 命令，`--onpart` 命令就不能够用在逻辑分区上。

参数包括：

`--all`，删除系统上所有分区。

`--drives=`，指定从哪个驱动器上清除分区。

例如，下面的命令清除了主 IDE 控制器上的前两个驱动器上的所有分区：

```
clearpart --drives=hda, hdb --all
```

`--initlabel`，根据不同体系结构把磁盘标签初始化为默认设置（例如，`msdos` 用于 X86 而 `gpt` 用于 Itanium）。当安装到一个崭新的硬盘时，这很有用，安装程序不会询问是否应该初始化磁盘标签。

`--linux`，删除所有的 Linux 分区。

`--none`（默认），不要删除任何分区。

- `part` 或 `partition`（对于安装是必需的，升级可忽略）

在系统上创建分区。注意，作为安装过程的一部分，所有被创建的分区都会被格式化，除非使用了 `--noformat` 和 `--onpart`。参数包括：

`<mntpoint>`、`<mntpoint>` 是分区的挂载点，它必须是下列形式中的一种：

- `<path>`，例如，`/`、`/usr`、`/home`。
- `swap`，该分区被用作交换空间，要自动决定交换分区的大小，使用 `--recommended` 选项。
- `swap -recommended`。
- 自动生成的交换分区的最小值大于系统内存的数量，但小于系统内存的两倍。
- `raid.<id>`，该分区用于 software RAID（参考 `raid`）。
- `pv.<id>`，该分区用于 LVM（参考 `logvol`）。

`--size=`，以 MB 为单位的分区最小值，在此处指定一个整数值，如 500。不能在数字后面加 MB。

`--grow`，告诉分区使用所有可用空间（若有），或使用设置的最大值。

`--maxsize=`，当分区被设置为可扩充时，以 MB 为单位的分区最大值。在这里指定一个整数值，不能在数字后加 MB。

`--noformat`, 用`--onpart` 命令来告诉安装程序不要格式化分区。

`--onpart=`或`--usepart=`, 把分区放在已存在的设备上。

例如: `partition /home --onpart=hda1`, 把/home 置于必须已经存在的/dev/hda1 上。

`--ondisk=`或`--ondrive=`, 强迫分区在指定磁盘上创建。

例如: `--ondisk=sdb` 把分区置于系统的第二个 SCSI 磁盘上。

`--asprimary`, 强迫把分区分为主分区, 否则提示分区失败。

`--type=` (用 `fstype` 代替), 这个选项不再可用, 应该使用 `fstype`。

`--fstype=`, 为分区设置文件系统类型, 有效的类型为 `ext2`、`ext3`、`swap` 和 `vfat`。

`--start=`, 指定分区的起始柱面, 它要求用`--ondisk=`或`ondrive=`指定驱动器。它也要求用`--end=`指定结束柱面, 或用`--size=`指定分区大小。

`--end=`, 指定分区的结束柱面。它要求用`--start=`指定起始柱面。

`--bytes-per-inode=`, 指定此分区上创建的文件系统的节点大小。不是所有的文件系统都支持这个选项, 所以在其他情况下它都被忽略。

`--recommended`, 自动决定分区的大小。

`--onbiosdisk`, 强迫在 BIOS 找到的特定磁盘上创建分区。

注意: 如果因为某种原因分区失败, 虚拟终端上会显示诊断信息。

- `volgroup` (可选)

用来创建逻辑卷管理 (LVM) 组, 其语法格式为:

```
volgroup <name><partition><options>
```

参数包括:

`--noformat`, 使用一个现存的卷组, 不要格式化它。

`--useexisting`, 使用一个现存的卷组, 重新格式化它。

`--pesize=`, 设置物理分区 (physical extent) 的大小。

- `logvol` (可选)

可使用以下语法来为 LVM 创建逻辑卷:

```
logvol <mntpoint> --vgname=<name> --size=<size> --name=<name><options>
```

参数包括:

--noformat, 使用一个现存的逻辑卷, 不进行格式化。

--useexisting, 使用一个现存的逻辑卷, 重新格式化它。

--fstype=, 为逻辑卷设置文件系统类型。合法值有 ext2、ext3、swap 和 vfat。

--fsoptions=, 申明了挂载文件系统时的许多参数。

--bytes-per-inode=, 指定在逻辑卷上创建的文件系统的节点的大小。因为并不是所有的文件系统都支持这个选项, 所以在其他情况下它都被忽略。

--grow=, 告诉逻辑卷使用所有可用空间 (若有), 或使用设置的最大值。

--maxsize=, 当逻辑卷被设置为可扩充时, 以 MB 为单位的分区最大值。在这里指定一个整数值, 不能在数字后加 MB。

--recommended=, 自动决定逻辑卷的大小。

--percent=, 用卷组里可用空间的百分比来指定逻辑卷的大小。

这些磁盘管理指令需要按照下面顺序来使用: 首先创建分区, 然后创建逻辑卷组, 再创建逻辑卷。例如:

```
part pv.01 --size 3000
volgroup myvg pv.01
logvol / --vgname=myvg --size=2000 --name=rootvol
```

(11) 启停选项

- **reboot** (可选)

在成功完成安装 (没有参数) 后重新启动。通常, Kickstart 会显示信息并等待用户按任意键来重新启动系统。

- **shutdown** (可选)

在成功完成安装后关闭系统。在 Kickstart 安装过程中, 如果没有指定完成方法, 将使用默认的 reboot 选项。

5. Kickstart 软件包选择

在 Kickstart 文件里使用 `%packages` 命令来列出想安装的软件包（仅用于全新安装，升级安装时不支持软件包指令）。

可以指定单独的软件包名或是组，或者使用星号通配符。安装程序可以定义包含相关软件包的组。关于组的列表，可以参考 Red Hat/CentOS Linux 光盘里的 `variant/repodata/comps-*.xml`。每个组都有一个编号、用户可见性的值、名字、描述和软件包列表。在软件包列表里，如果这个组被选择的话，组里标记为“mandatory”的软件包就必须被安装，标记为“default”的软件包默认被选择，而标记为“optional”的软件包必须被明确地选定才会被安装。

多数情况下，只需要列出想安装的组而不是单个的软件包。注意，Core 和 Base 组总是默认被选择，所以并不需要在 `%packages` 部分指定它们。

下面是一个 `%packages` 选择的示例：

```
%packages
@ X Window System
@ GNOME Desktop Environment
@ Sound and Video dhcp
```

如上所看到的，组被指定了。每个组占用一行，用@符号开头，后面是 `comps.xml` 文件里给出的组全名。组也可以用组的 id 指定，如 `gnome-desktop`。不需要额外字符就可以指定单独的软件包（上例里的 `dhcp` 行就是一个单独的软件包）。

`%packages` 指令也支持下面的选项。

`--nobase`，不要安装@Base 组，如果想创建一个很小的系统，可以使用这个选项。

`--resolvedeps`，选项已经被取消了。目前依赖关系可以自动地被解析。

`--ignoredeps`，选项已经被取消了。目前依赖关系可以自动地被解析。

`--ignoremissing`，忽略缺少的软件包或软件包组，而不是暂停安装来向用户询问是中止还是继续安装。

例如：`%packages --ignoremissing`

6. Kickstart 预安装脚本

可以在 `ks.cfg` 文件被解析后马上加入要运行的命令。这个部分必须处于 Kickstart 文件

的最后（在命令部分之后），而且必须用`%pre`命令开头。可以在`%pre`部分访问网络；然而，此时命名服务还未被配置，所以只能使用 IP 地址。

预安装脚本不在改换了的根环境（`chroot`）中运行。

7. Kickstart 安装后脚本

可以加入在系统安装完毕后运行的命令。这部分内容必须在 `Kickstart` 的最后，而且用`%post`命令开头。它被用于实现某些功能，如安装其他的软件和配置其他的命名服务器。

注意：如果用静态 IP 信息和命名服务器配置网络，可以在`%post`部分访问和解析 IP 地址。如果使用 DHCP 配置网络，当安装程序执行到`%post`部分时，`/etc/resolv.conf` 文件还没有准备好。此时，可以访问网络，但是不能解析 IP 地址。因此，如果使用 DHCP，必须在`%post`部分指定 IP 地址。

`post-install` 脚本是在 `chroot` 环境里运行的。因此，某些任务如从安装介质复制脚本或 RPM 将无法执行。

参数包括：

`--nochroot`，允许指定想在 `chroot` 环境之外运行的命令。

下例把`/etc/resolv.conf` 文件复制到刚安装的文件系统里：

```
%post --nochroot cp /etc/resolv.conf /mnt/sysimage/etc/resolv.conf
--interpreter /usr/bin/python
```

允许指定不同的脚本语言，如 Python。把`/usr/bin/python` 替换成想使用的脚本语言。

8. 一个简单完整的实例

下面是一个真实生产环境里使用的 `Kickstart` 文件的例子。在这个例子里，这个主机将从网络 PXE 启动，然后创建基于 LVM 的几个分区，分别存放根分区、交换分区和数据分区等。在系统安装后，会执行两阶段的脚本，在第一阶段，将一些文件从安装媒介里复制到本机；而在第二阶段，则执行一些系统的配置，另外使用工具，将启动顺序从 PXE 网络启动修改为从第一块硬盘启动。因为我们不想在该机重启后，反复从网络安装。

```
# Kickstart file automatically generated by anaconda.

#version=DEVEL
install
cdrom
```

```

lang en_US.UTF-8
keyboard us
network --onboot no --device eth0 --bootproto dhcp --noipv6
rootpw --iscrypted
$6$j34jh/UhCcp2ZSF3$4n0h4Jf19zqAhYteeu0d2lnukfS9szHRZDqOxMb264gN.5F8eRvFeoCN
OgHErIj6a3Y45hL/U7A43xvJkqUFQ0
firewall --service=ssh
authconfig --enableshadow --passalgo=sha512
selinux --disabled
timezone --utc PRC
bootloader --location=mbr --driveorder=sda --append="crashkernel=auto rhgb
quiet"
zerombr
# The following is the partition information you requested
# Note that any partitions you deleted are not expressed
# here so unless you clear all partitions first, this is
# not guaranteed to work
clearpart --all --initlabel

#create partitions for boot 512M
part /boot --fstype=ext4 --size=512 --ondisk=sda

#This is a cheat rootvg
part pv.01 --size=102400 --ondisk=sda

#This is a cheat to allow the partition for the logical volume to grow and consume
the rest of the disk
part pv.02 --grow --size=1

#create volume group rootvg
#then create logical volumes for swap 32G, root 200G
volgroup rootvg --pesize=4096 pv.01
logvol swap --name=lvswap --vgname=rootvg --size=32768
logvol / --fstype=ext4 --name=lvroot --vgname=rootvg --size=51200

#create volume group openstackvg
volgroup openstackvg --pesize=4096 pv.02

%packages
*
%end

%post --nochroot
cp -r /mnt/source/postinstall /mnt/sysimage/opt/postinstall
mv /mnt/sysimage/opt/postinstall/compute_host/* /mnt/sysimage/opt/lbs
%end

```

```
%post
/opt/postinstall/drivers/install.sh
find /opt/lbs -type f -name "*.TBL" -exec rm {} \;
#/opt/ibm/toolscenter/asu/asu64 set BootOrder.BootOrder "Hard Disk 0"="CD/DVD
Rom"="PXE Network" --kcs
%end

reboot
```

7.3.3 编辑可引导的 ISO

很多时候，还可以修改已经制作好的 ISO 的内容。下面是一些小技巧。

(1) 复制 ISO 内容到临时目录：

```
Mount ISO
mkdir /mnt/dvd
mount -o loop,ro RHEL6.1-20110510.1-Server-x86_64-DVD1.iso /mnt/dvd
Copy the files from the DVD to a temporary directory
mkdir /tmp/rhel6-custom
rsync -a /mnt/dvd/ /tmp/rhel6-custom/

Unmount the ISO image
umount /mnt/dvd
```

(2) 加入定制化内容。

例如，修改并将 Kickstart 文件重新放入（调整了安装路径 -- /path/to/）。

```
cp /path/to/ks.cfg /tmp/rhel6-custom
```

(3) 创建新的可引导 ISO

```
cd /tmp/rhel6-custom/
mkisofs -o /tmp/rhel6-custom-install.iso -b isolinux/isolinux.bin -c
isolinux/boot.cat -no-emul-boot -boot-load-size 4 -boot-info-table -r -T -J -V
"RHEL 6.1 Custom Install DVD" .
```

不要忘记命令最后面的“.”。

现在可以使用一些工具来将前面生成的 ISO 文件制作成 DVD 格式，工具如 growisofs、cdrecord，或者 k3b。

```
growisofs -dvd-compat -Z /dev/dvd=/tmp/rhel6-custom-install.iso
Install from the new DVD. At the prompt (you may have to hit TAB), enter linux
```

```
ks=cdrom:/ks.cfg for a kickstart install.
```

7.3.4 制作一个定制化可引导的 ISO

CentOS、Red Hat Enterprise Linux 提供了一个强大的工具来快速部署和管理配置：Kickstart。Kickstart 可以让您生成一个定制化的安装包，不但可以快速而容易地分发软件，而且可以让目标机器安装的和期望的完全一致。比较遗憾的是，Kickstart 的文档不多且不易理解。

可以想象，如果有 30 个 Web 服务器，也许还是混合了 Web 服务器、数据库、邮件服务器等，我们希望在所有机器上安装完全相同的软件包，这样我们可以确定这些机器运行的软件和环境将完全一致。

如果使用的是变准的 DVD，要想达到这个效果可能是有点挑战性的，在安装过程中，很可能为某台机器不小心选择了不同的一组软件包，特别是，如果使用的不是标准的安装类型。特别地，对于生产环境，需要一些不在标准的 CentOS 发布版里的软件包，例如，我们可能需要增加一些其他软件工具，如 `httpperf`、`ganglia`、`iptraf`、`memcached`、`nagios` 和 `swatch`。如果需要在安装后，再增加这些软件包，需要额外花费很多时间，也许可能忽略了某台机器，或某个软件包，而引起系统运行时一些不可预测的行为。

另外，除了软件包，也许还需要对系统和软件做各种配置，例如，可能需要配置 `httpd` 和 `mysql`，或者需要增加一些用户和组，修改 `/etc/group` 文件，或者修改一些密码到 `/etc/passwd`。非常要紧的是，这些改动要在所有机器里完全一致地完成。

解决的方法是事先在 Kickstart 里定义软件包的列表，让 Kickstart 自动地控制安装过程，确保所有机器都安装同样的软件包。更进一步，Kickstart 可以在安装后执行一些脚本逻辑，可以用来安装一些非发行版的软件，执行一些系统配置。为了简化我们的工作，需要把所有这些工作分装到单个的 DVD 里来完成所有这些希望的事情。

1. 准备一个制作系统

我们需要一个安装了 CentOS 6 操作系统的机器来组装需要的软件包和制作定制化的 ISO 文件。这可以是台物理机，也可以是台虚拟机。下载两个 DVD ISO 之后，在安装前验证下 MD5 校验。如果需要装到物理机，则需要将 ISO 烧到 DVD 里；如果是虚拟机，则只需要直接 mount ISO。

用“Desktop”安装类型来安装 CentOS 6 到主机（除非你想要手工定制化安装，那样的话，还需要安装开发工具）。再准备制作 ISO 的主机，创建一个如下结构的目录。顶层目录为~/kickstart_build，子目录包括：

```
~/kickstart_build
+-- isolinux
|   +-- Packages
|   +-- images
|   +-- ks
+-- utils
```

从 CentOS 第一张光盘的 isolinux 目录下复制所有文件到~/kickstart_build/isolinux 目录里。另外，从 CentOS 第一张光盘中复制.discinfo 到~/kickstart_build/isolinux 目录里。

从 CentOS 第一张光盘的 images 目录下复制所有文件到~/kickstart_build/isolinux/images 目录里。

从 repodata 目录里找到 comps.xml 文件。在 RHEL/CentOS 6 里，这个文件不再仅仅叫作“comps.xml”，而是在文件名里加入了一些十六进制的字符串。在 CentOS 6.2 中，它的名字叫 bedb7dc8fd920deffbdc5a70ea0d6d77255656556184f5e996e8a88a63d145c-c6-X86_64-comps.xml.gz。复制这个文件到~/kickstart_build/comps.xml.gz，然后用 gunzip 解开，这样就有了~/kickstart_build/comps.xml 文件。

Kickstart 的配置文件将放到~/kickstart_build/isolinux/ks 下。最好是分配单独的目录给它们，因为这样可以给不同的机器或不同类型的机器创建一个单独的配置文件。

2. 准备起始的配置文件

最好是从一个简单的配置文件开始。CentOS 在这些小的细节方面也为你准备好了。只需要做一个标准的安装，安装后，查看/root/anaconda-ks.cfg 目录，就会发现一份简单的 Kickstart 配置文件。

将这个文件复制为~/kickstart_build/isolinux/ks/ks.cfg。编辑这个文件，找到分区信息的部分，这个部分一般被注释掉了。它包含下列信息：

```
# The following is the partition information you requested
# Note that any partitions you deleted are not expressed
# here so unless you clear all partitions first, this is
# not guaranteed to work
```

把这些和磁盘分区相关的注释去掉，否则，你的 Kickstart 将没有磁盘分区信息，从而

没有地方去安装选择的软件包。

在这里的驱动配置信息反映了当前制作主机的信息，如果这些信息和安装目标机器不一样（例如，当前使用的机器有 IDE 的驱动，而目标主机用的是 SCSI，或者你准备在目标机器中使用软件 RAID 等），必须修改一些行。如果当前使用的制作主机和将来要安装的机器一样，将会使得制作变得更加容易。

3. 收集 RPM 包

尽管这不是绝对需要的，但将所需要的所有 RPM 包收集到一个目录里，将会让后续的工作变得容易。如果是制作定制化的光盘镜像，我们将只复制所需要的那些 RPM 包。但我们接下来可能需要访问各种各样的 RPM 包，最好是把全部的 RPM 包复制到一个临时目录里。从 DVD1 里复制所有的 RPM 到 `~kickstart_build/all_rpms` 目录里，对 DVD2 光盘，重复同样的动作。这样对后续解决依赖问题会很有帮助。

4. 确定要包括的软件包

下面查看 `~kickstart_build/comps.xml` 文件（就是从 CentOS 发布的光盘的第一张 `repodata/comps.xml` 复制来的）。这里定义了软件包和它们的组。

当前的 Kickstart 配置文件在 `%packages` 里列出了要安装的那些包。而组则用一个以 `@` 开头的字段来标记。把 `@core` 和 `@base` 在这个文件里保留，因为这里定义了一些非常关键的软件包。没有它们系统很可能运行不起来。

如果希望精简软件包的列表，从而制作一个非常紧凑的发行版，在这里可以把一些不需要的组删除，或者把组的名字替换成那些组里想要安装的具体包的名字。可以到 `comps.xml` 里查看哪些组里包括了哪些软件包。如果需要明确地去除某个软件包，可以在包名前加一个“-”，在安装时 ISO 时，这个包就不会被安装。

接下来就是最挑战的部分：生成需要的 RPM，并且解决依赖的问题。

在确定要包括哪些软件包之后，接下来需要把它们放到一起。

我们需要哪些 RPM？假定至少需要包括 `@core` 和 `@base` 里的软件包的组（通常必须是），一个方法是查看 `comps.xml` 里，在 `core` 和 `base` 组里的 RPM 列表。这个 XML 看起来如下：

```
<group>
  <id>core</id>
```

```

<name>Core</name>
...
<packagelist>
  <packagereq type="default">Deployment_Guide-en-US</packagereq>
  <packagereq type="mandatory">SysVinit</packagereq>
  <packagereq type="mandatory">authconfig</packagereq>
  <packagereq type="mandatory">basesystem</packagereq>
  <packagereq type="mandatory">bash</packagereq>
  <packagereq type="mandatory">centos-release</packagereq>
  <packagereq type="mandatory">coreutils</packagereq>
  <packagereq type="mandatory">cpio</packagereq>
  ...
</packagelist>
</group>

```

在最少的情況下，我們需要 base 和 core 組里所有需要的包。可以看到在每個組里有 `<packagereq type="mandatory">`和`<packagereq type="default">`這樣的條目。這些條目說明了哪些包要包括進來，這些包的名字和 RPM 的名字是一一对应的。

Red Hat 網站有個 Perl 編寫的腳本，可以讀取 comps.xml 文件，然後收集 core 和 base 組里所有非可選的包。可以搜索並下載 `parse_comps.pl` 腳本，並保存到 `~/kickstart_build/utlis` 目錄里。

在運行 `parse_comps.pl` 之前，還需要安裝一些 perl 的庫：

```

cd ~/kickstart_build/all_rpms
sudo rpm -Uvh \
perl-Compress-Raw-Zlib-2.023-119.el6.x86_64.rpm \
perl-Compress-Zlib-2.020-119.el6.x86_64.rpm \
perl-HTML-Parser-3.64-2.el6.x86_64.rpm \
perl-HTML-Tagset-3.20-4.el6.noarch.rpm \
perl-IO-Compress-Base-2.020-119.el6.x86_64.rpm \
perl-IO-Compress-Zlib-2.020-119.el6.x86_64.rpm \
perl-libwww-perl-5.833-2.el6.noarch.rpm \
perl-URI-1.40-2.el6.noarch.rpm \
perl-XML-Parser-2.36-7.el6.x86_64.rpm \
perl-XML-Simple-2.18-6.el6.noarch.rpm

```

按照下面的方法運行：

```

cd ~/kickstart_build/isolinux/Packages
~/kickstart_build/utlis/parse_comps.pl ~/kickstart_build/comps.xml
~/kickstart_build/all_rpms x86_64

```

5. 增加 RPM

如果希望增加那些不是包括在 `base` 和 `core` 组里的任意 RPM (`httpd`、`php` 或 `mysql`)，可以把那些 RPM 包复制到 `~/kickstart_build/isolinux/Packages` 目录里。

6. 解决依赖关系

现在我们已经把所有 `base` 和 `core` 组里的 RPM 收集到 `~/kickstart_build/isolinux/Packages` 目录里，另外还加入了其他想要安装的应用的 RPM 包。但是很可能，许多依赖的包并没有包含进来。所以下一步，需要跟踪那些依赖关系，把所有需要的包都放到这个目录里。在这里可以用另外一个脚本（可以搜索并从网上下载），来完成依赖关系的检查和清理。

```
cd ~/kickstart_build/isolinux/Packages
~/kickstart_build/utils/follow_deps.pl ~/kickstart_build/all_rpms x86_64
```

(1) 测试依赖

一旦已经收集到需要的 RPM，现在需要测试一下这些 RPM 的依赖关系，看是否依赖的包都已经包含在这里。每个 RPM 包都可能需要其他的 RPM 先被安装，才能安装当前的 RPM 包。如果依赖的 RPM 包也被包含在这些 RPM 里，那么所有的依赖关系就解决了。这一步的目的是收集到完全没有外部依赖的 RPM 的集合用于目标主机的安装。

为了在当前的 RPM 集合里测试依赖关系，需要执行下列命令：

```
mkdir /tmp/testdb
rpm --initdb --dbpath /tmp/testdb
rpm --test --dbpath /tmp/testdb -Uvh *.rpm
```

如果此时发现有没有解决的依赖关系（通常情况下都会有的），可以从两个途径来解决。

- 删除那些没有解决依赖关系的 RPM（例如，这些 RPM 引入了一个依赖关系的链条，而这些本来都不需要，如安装服务器，但包括了 X Windows 的包）。
- 跟踪 RPM 直到依赖关系被满足。这里需要一些判断，来确定哪些 RPM 包可以被去除。通常如果软件包是明确加到 `core` 和 `base` 组里的，那么必须寻找到那些依赖的包。如果真是一些不需要的包，才可以删除它们。

(2) 解决依赖

`follow_deps.pl` 脚本可以自动地寻找到所有依赖的包。但是如果不能的话（特别是如果增加了一些包而引起混乱的话），则需要手工地跟踪依赖关系。

需要 RPM 包来满足依赖关系，可能说起来容易做起来难。有时候，RPM 包里列出来

的依赖和真正的文件名不匹配,例如,可能有一个包 `redhat-lsb-3.1-12.3.EL.el5.centos.X86_64` 需要 `/usr/bin/strip` 这个文件,而它是由 `binutils` 提供的。

如果不确定什么样的 RPM 提供了这些依赖,可以用 RPM 本身来查询正确的 RPM 包(假定在当前系统里,已经正确地安装了这些包的话,通常如果有个机器里收集了所有需要的 RPM 包,则将会非常有用)。

```
rpm -q --whatprovides /usr/bin/strip
```

这是一个不断重复的过程,准备好花费一段时间来跟踪依赖关系,直到得到一个非常良好地解决了所有依赖关系的 RPM 包的集合。

7. 生成软件仓库

安装 `createrepo`, 可以用来创建软件仓库 (Repository)。如果如前所述,已经把所有的 RPM 都复制到 `~kickstart_build/all_rpms` 目录里,则可进行下面的步骤:

```
cd ~kickstart_build/all_rpms
rpm -Uvh createrepo*rpm deltarpm*rpm python-deltarpm*rpm
```

接下来需要为安装光盘生成 `repodata` 文件。这些文件提供了可用软件包的安装信息。

现在使用 `createrepo` 来创建仓库信息:

```
cd ~/kickstart_build/isolinux
declare -x discinfo=`head -1 .discinfo`
createrepo -u "media://$discinfo" -g ~/kickstart_build/comps.xml .
```

这一步将在 `~/kickstart_build/isolinux` 里创建一个 `repodata` 目录,里面会包含仓库的数据文件。

8. 生成 ISO

下面要生成 ISO 文件,然后可以烧制成 DVD:

```
cd ~/kickstart_build
mkisofs -o custom.iso -b isolinux.bin -c boot.cat -no-emul-boot \
    -boot-load-size 4 -boot-info-table -R -J -v -T isolinux /usr/lib/anaconda-
runtime/implantisomd5 custom.iso
```

这样就生成了一个名字为 `custom.iso` 的 ISO 镜像文件。

9. 测试 ISO

此时不需要烧制一个真实的 DVD，然后找台物理机去做真实的安装，可以找个虚拟机来做安装，例如可以使用 VirtualBox 来完成这件事。

当系统启动之后，可以看到标准的 CentOS 安装提示，在继续之前，按 Esc 键，然后输入：

```
linux ks=cdrom:/ks/ks.cfg
```

如果 Kickstart 配置文件是其他的名字，这里只需要将上述的 ks.cfg 替换成相应的名字即可。这里可以根据主机名或者服务类型来命名配置文件，也就是说配置文件的名字完全可以根据需要来自行命名。

安装应该继续直到顺利结束。

此时有了一个工作的而且可以定制的 Kickstart 安装光盘。为了最大限度地利用 Kickstart 的强大功能，现在可以增加一些安装后执行的脚本，来根据需要配置系统。

10. %post 部分

在 Kickstart 配置文件里，可以加入一些命令在安装结束后运行。这些脚本由 %post 指令来表示。这些脚本可以通过读取 /mnt/source 目录来访问安装媒介的文件。

这段脚本有一些格式上的要求。但更重要的是，如果需要访问安装后的系统，这些系统应挂载在 /mnt/sysimage 目录里，而不是通常安装后系统重启运行的 “/”。

默认地，这些在 %post 部分的脚本运行于 chroot 的环境，因此 /mnt/sysimage 此时可以通过 “/” 来读取访问，这样就可以使用正常的路径，如 /etc，而不是 /mnt/sysimage/etc 来访问系统的配置文件。但这里如果需要访问安装媒介上的内容，则在 chroot 的环境里是不可见的。

这个问题可以通过将安装后执行脚本分成两阶段执行来解决。在第一阶段，我们告诉 anaconda 不要 chroot，然后我们可以将需要的文件从 DVD 复制到硬盘：

```
%post --nochroot
#!/bin/sh
```

```
set -x -v
exec 1>/mnt/sysimage/root/kickstart-stagel.log 2>&1
```

```
echo "==> copying files..."
```

```
cp -r /mnt/source/postinstall /mnt/sysimage/root/postinstall
```

在本文的例子中，我们所有在安装后执行的文件都存放在 `~/kickstart_build/isolinux` 目录下的 `postinstall` 目录里。注意，这里我们创建 ISO 的环境里的 `isolinux` 目录会变成我们生成的安装光盘的根目录。所以放在 `postinstall` 目录里的内容，在安装后就被复制到新系统硬盘的 `/root/postinstall` 目录。

接下来我们需要运行 `postinstall` 的第二阶段脚本：

```
%post
#!/bin/sh

set -x -v
exec 1>/root/kickstart-stage2.log 2>&1
```

在这里的两阶段脚本里，都将输出定向到 `root` 目录下的日志文件里。这样如果有上百行脚本代码在安装后执行，会对调试问题很有帮助。

下面列举出了一些可以在安装后执行的事情。

- 增加用户和组。
- 使用 RPM 安装非 CentOS 的应用。
- 使用 tar 文件来安装非 CentOS 的应用。
- 设置各种系统服务的运行 `runlevels`。
- 配置各种服务，如 `apache`、`samba`、`SSHD`、`MySQL`。
- 配置 `bash Shell` 的各种默认行为。
- 其他任何的配置等。

这样系统开始运行之后，就不需要手工来做各种安装配置的事情了。

组织 `postinstall` 文件

下面是文件组织的目录结构（这只是一个建议）：

```
~/kickstart_build/isolinux/postinstall
+-- apps
+-- appconfig
+-- sysconfig
+-- libs
```

把非 CentOS 本身的 RPM 和 tarball 文件放到 `apps` 目录里（对每个应用，创建一个单独的子目录），把应用的配置文件、脚本文件放到 `appconfig` 目录里（同样地，每个应用一

个子目录), 把操作系统配置文件(如网络配置文件)放到 `sysconfig` 目录里, 把通用目的的库文件(这些文件也许不是那个特定的要安装的应用需要的)放到 `libs` 目录里。

其中, 对应用的管理是最重要的, 当你安装非 CentOS 发布版的应用时, 往往需要安装额外的库文件或者工具来满足这些应用的依赖关系。当需要更新 Kickstart 镜像时, 将每个应用已经需要的依赖文件都放到同一个目录将会极大地减轻维护的复杂度。

11. 外部仓库

下面列出一些外部仓库, 里面的内容没有包括在正式的 CentOS 发行版里面, 但是非常可靠。

- **EPEL:** Enterprise Linux 额外的包。
- **Reporforge:** 以前的 RPMforge, DAG repository 现在也指向这里。
- **ATrpms:** 获取 ffmpeg rpms 的好地方。
- **ELrepo:** 硬件驱动的好地方。

第 8 章

OpenStack 部署

OpenStack 的部署规划与设计，需要非常强地理解用户需求，并寻找最佳的方案来满足企业需求。

8.1 来自实际客户的困惑

在与客户交流的过程中，客户有许多实际的问题。对于这些问题，有无答案，或者是否让用户满意，是云平台能否走出社区和一堆开源代码，带来生命力的重要因素。

下面是一些具体的问题。

- 难于取舍 Xen 还是 KVM

多年来许多客户基于 Xen 开发和搭建了自己的“自主知识产权”的企业云平台，或者给其他客户提供产品和解决方案。Xen 本身也是早期比较成熟的一种虚拟化管理程序。然而，在 OpenStack 中，KVM 是默认的虚拟化管理程序，因此针对云平台的各种功能、开发、试验和企业级使用，KVM 得到最高的关注度和使用度，因此各项功能得到最多的验证，问题最少。不可否认的是 Xen 也是 OpenStack 支持的虚拟化管理程序之一，OpenStack 完全可以管理基于 Xen 的虚拟化主机池。造成这种困惑的主要原因之一，或许是因为 KVM 较早集成进入 Linux 的内核，虽然 Xen 有早发的优势，但在集成进入 Linux 内核方面却迟缓许多。而 Linux 是搭建云平台最广泛的操作系统，KVM 本身在内核，无须额外的安装与配置就可以使用，并且运行良好，赢得了开源社区和相当多的用户。

- 选择 OpenStack 还是其他开源云平台技术

在前几年，出现了许多开源的云平台技术和软件，并且都通过社区的方式进行推广。判断哪种云平台技术适合自己，并没有标准的答案，主要还是看哪种技术适合自己，适合企业发展战略，是否有足够的技术能力去掌握他们。对于 OpenStack 来说，这个项目得到全世界绝大部分顶尖的服务器、存储、网络厂商的加入，围绕其已经形成一个完整的生态系统，包括硬件、软件、实施和服务等。

另外需要说明的是，VMware、微软等都是 OpenStack 黄金会员，国内如华为、中兴等也都是黄金会员。换句话说，以后无论是公有云，还是私有云、混合云，你所接触和使用的产品服务厂商中，可能绝大部分都是属于 OpenStack “阵营”的。

另外一方面，OpenStack 从各个角度，都避免了厂商锁定，给用户提供了最大的灵活性和选择面。另外，OpenStack 本身是开源的，其技术是大家都可以自由学习和

掌握的，这也是相对于闭源产品最大的优势之一。

最后，OpenStack 的设计和函数都参考了亚马逊云平台，也就是说，如果采用 OpenStack 并且持续跟踪其最新的技术，有一天，你将自己拥有如亚马逊云平台一样强大的，不但涵盖 IaaS，而且包括许多 PaaS 的云平台。

值得一提的是，现在许多客户都选择了 OpenStack，因为看到了它蓬勃的发展和最广泛的厂家和技术支持。笔者服务过的一个著名的全国级的电商，在部署了 CloudStack 仅仅三个月，在对云平台有了更多了解之后，果断地放弃了 CloudStack，而部署了 OpenStack。

- 最佳部署有哪些

OpenStack 提供了非常灵活的部署模型，对每个功能，都有许多不同的选择，如虚拟化管理程序、网络插件、存储后端、镜像存储等。然而，只有适合自己的才是最佳的，需要基于自己的硬件环境和云平台业务需求来设计。当然，最通用的部署方式也是可以的，它是最安全的，但并不一定是最佳的。

- 需要更多 OpenStack 资料

无论是 OpenStack 社区，还是一些公司网站，包括一些论坛，都提供了许多文档资料，但总体上都是碎片化的，用户需要花费许多时间去分析、学习和理解。另外，国内现在有许多培训，提供的环境可以直接搭建云平台。最后，可以找专业研究和实施的公司团队。开源并不意味着免费，如果方法不对，极端情况下可能花在开源产品上的时间和代价比闭源产品还要高。通常，开源产品都可以将价格大幅降下来，总体是对用户非常有利的。

- 性能，如何保证业务性能不降级或不显著降级

性能调优涉及许多方面，从硬件选型、配置到软件搭配等，还包括系统和虚拟化管理程序的优化，以及服务的线性扩展、分布式并行文件系统等，内容非常庞大博杂。一般地，虚拟化平台经过调优可以达到或超过原来同等级物理平台 90% 以上的性能。

- 可靠性，如何保证数据不丢失，运行业务中断最小

OpenStack 在设计时，就基于一个原则：硬件是不可靠的。因此，OpenStack 服务本身是高度分布式、高度容错和无状态的，需要关注的是用户数据本身的可靠性问题。可靠性包括在线数据安全和离线数据的可靠性。可靠性包括硬件层面如 Raid，也包括软件方面的诸如分布式并行文件系统、存储设备的冗余、实时备份等多种途径的组合，来达到原来一些专用设备（如 SAN 存储）的同等效果。

- 安全，包括传统网络安全与虚拟化带来的安全问题

除了传统的安全威胁，云平台带来了额外的包括虚拟化本身的一些安全问题，如东

西向安全，虚拟化管理程序安全等。

- 虚拟机高可靠性，通常对比 VMware vCenter

OpenStack 社区本身并不关注应用的高可用性问题，这也是最为人诟病和阻挠其进入企业级市场的部分。通过一些非常可靠的、互联网公司使用的技术和工具，基于 OpenStack 的云平台也能提供像 VMware vCenter/ESXi 一样的虚拟机高可用性，但目前 OpenStack 在管理、使用的方便程度上与 VMware 还存在差距。另外，VMware 的容错（Fault Tolerance）功能，OpenStack 目前仍然无法提供。

- 虚拟机是否支持热迁移，通常对比 VMware vCenter

只要 OpenStack 底层使用的虚拟化管理程序支持，则 OpenStack 就支持。如果是 KVM，则提供基于块和共享存储的热迁移，迁移时，用户的虚拟机和里面的业务还在运行，一点儿都不受影响。另外，也提供冷迁移。这个是和 VMware vCenter 同样的，但是共享存储的方案更多，选择面也更广泛。

- 运维自动化

这是需要自行开发的，或由其他专业公司来提供的，OpenStack 本身不提供此类功能，但提供技术和基础架构的支持。

- 业务自动化，支持自动从开发到上线

由于云平台和虚拟化可以说都是软件定义的，即传统的计算、存储和网络都是可以通过编程来控制 and 使用的，因此，云平台的业务自动化反而比传统平台大大简化，现在在开源社区已经有许多基于 Chef 和 Puppet 的工具和项目可以参考并使用，或自主新开发更适合自己的业务自动化平台。

- 特殊需求，如如何在虚拟机共享存储

此类需求，通常是一些传统的应用需要的。在云平台环境，共享存储或共享文件有更多的选择，通常包括在裸盘基本的共享和文件级别的共享。另外，基于网络的共享也有许多方案可选。

- 升级，此类需求国外居多

OpenStack 的跨版本升级本身是非常有挑战性的，另外，如果是生产系统，则难度会上升许多。

在 OpenStack E 版本之后，国外许多高校、公司甚至银行就部署了从几十台到上千台节点的 OpenStack，现在已经几乎两年时间了。OpenStack 的升级有以下几条路径。

- 原来的系统仍然运行于旧版本的 OpenStack 环境，而新系统则搭建在新版本的 OpenStack 环境中。
- 逐版本升级，直到最新版本为止。OpenStack 本身是支持 $N \rightarrow N+1$ 版本升级的，但

如果是跨版本多，则可行性几乎为零。因为不会有实际生产使用的客户选择这种办法。

- 跨版本升级。考虑到不但 OpenStack 项目模块本身发生了许多变化，而且客户业务系统也具有很大的复杂性，这条升级路径也仍然是难度很大、很有挑战性的。
- 旧版本 OpenStack 环境逐步向新版本 OpenStack 环境的迁移。这是一条比较希望的升级路径。考虑到底层实际涉及的是虚拟化管理程序的迁移（或升级）和数据库本身的变化，这条升级路径从技术难度、可行性、对业务系统的影响都可以在预测和控制的范围之中。

- 是否支持 Ubuntu

这通常是国外客户关心的问题。在韩国、澳大利亚、欧洲，许多学校、科研机构和公司部署的 OpenStack 环境是基于 Ubuntu 的。事实上，CentOS/Red Hat, Ubuntu 和 SUSELinux 都是得到大量使用的 Linux 操作系统，但这种选择明显地带有地区性差异，特别是在国内，则以 CentOS/Red Hat 居多。因此操作系统的选择，除了一小部分技术方面的原因，还包括许多其他原因，尤其是从公司战略选择方面的考虑居多，如公司本来就和某操作系统提供商有合作关系，或者被供应商锁定。

- 需要集成企业/公司原有的认证系统

OpenStack 本身的 Keystone 可以很容易地与 LDAP 和 Windows AD 集成起来。集成其他认证系统需要开发一些中间件（Middleware）来支持。

- 容易擦除物理机/虚拟机原有的部署，替换为新的部署包

OpenStack 社区有 TripleO、Ironic 等项目可以提供此类功能，但目前还不完善。除此之外，如 xCat 等工具也可以很容易地实现此类功能。

- 需要好的界面

OpenStack 在 Grizzly 版本里，界面上可用性仍然比较弱，一些后台提供的功能在界面中还是看不到。在 Havana 里，以及后续的 IceHouse 里，界面已经得到很多增强，许多功能界面本身都可以操作。因此，我们可以期待社区在这方面会持续改进和提高。

- 需要 VDI

OpenStack 天生就具有 VDI 的潜质，其分布式的部署架构可以选择非持久化和持久化存储，支持 VNC 等方式，已经具备了一个 VDI 的雏形。通过集成一些支持前端设备透传的协议，增强用户管理和对高分辨率显示的支持，OpenStack 可以比较容易地改造为一个价廉物美的 VDI 平台。

- 适合教学科研的云平台

国内许多高校都开办了云计算的课程，无论老师还是学生都需要深入地了解云平台本身的技术原理。因此，此类云平台需要在基本云平台上特殊开发一些功能，如快速恢复功能。网络设计上也需要不但能访问虚拟机数据网络，还可以非常容易地切换到管理网络，让同学们可以从管理员角度观察和使用云平台。另外，云平台还可以承担一定的教学和科研任务，需要可以在不同任务之间快速切换，进行环境清理和资源调度回收。

- 是否可以自主研发

当然可以，因为 OpenStack 本身是开源的。但如果搭建企业生产环境，或者 50 台以上规模，或者包括多种网络或异构存储，也可以找专业公司寻求一定的技术支持。

8.2 企业云环境规划

无论是搭建企业级公有云，还是私有云，都是一项艰巨的、复杂的系统工程，主要涉及下列环节。

- (1) 理解现有和未来一年或三年企业业务需求和预期。
- (2) 基础设施的清单和扩充规划。
- (3) 云环境规划和部署设计。
- (4) 实施。
- (5) 后期开发和维护。

8.2.1 理解企业业务需求和预期

无论是公有云，还是私有云，无论是大规模，还是中小规模，云的设计和规划都必须符合企业业务需求，这里有一系列问题。

- (1) 云平台支持什么样的应用场景？

云存储，如金山云存储、腾讯云存储，基于 OpenStack 最好的选择是 Swift。如果只是云存储，基本上，对于 OpenStack 的核心服务，有所相关的只有 Keystone。

而如果是云主机，那么基本上 OpenStack 的核心服务都会涉及。如果是企业私有云，

则进一步分析如下。

- 是搭建开发测试云？
- POC (Prove-of-Concept) ？
- 企业内部非生产环境？
- 企业内部生产环境？
- 对企业内部和外部提供服务？
- VDI (桌面虚拟化) ？
- 大数据？
- 对性能的要求？
- 对可靠性的要求？
- 对安全性的要求？

企业里的应用经过许多年积累，非常庞杂，典型的应用是 Web 数据库应用。但对于数据库，又有如 MySQL、SQLite 等，也包括非结构化数据库，如 MongoDB 等。商业的有如 Oracle、DB2、SQL Server 等。每种商业数据库，无论内存、磁盘空间规划、集群模式，又是一个非常广阔的内容。其他还包括 Windows 平台上开发的大量应用软件，而且没有统一的模式，如进程、配置、网络、存储等元素。还有许多针对行业的应用，则无法枚举。

由于应用和需求的多样性，需要对资源池进行划分，不同的资源池满足不同的应用需求。在 OpenStack 里，可以有多种计算资源池，如 KVM 资源池、LXC/Docker 资源池、Baremetal 资源池、Hyper-V 资源池等。

(2) 云环境是通过跨越多个地理区域的数据中心来提供的吗？

如果要跨越多个地理区域，问题更加复杂，我们需要考虑以下问题。

- 部署多套 OpenStack 环境。
- multi-region(多区域)。这也是一种典型的用户场景。基本上，我们需要考虑 OpenStack 的核心服务，如 Nova、Neutron、Cinder、Dashboard 等项目。
- 更进一步，我们需要考虑哪些共享服务如何管理，包括 Keystone、Glance、Ceilometer 等。
 - 是否需要集成企业级如 LDAP、AD 等认证？
 - 是否需要共享的镜像服务？
 - 是否需要集中的统计和计费信息？
- 此外，还有消息中间件和数据库等基础服务。

- 是维护一个统一的消息机制和数据库，还是各个数据中心有独立的消息机制和数据库？

(3) 是否需要提供差异化服务？

如果需要提供差异化的服务，并且保证 SLA，则问题如下。

是否需要提供热迁移？

是否需要提供基于物理机和 Hypervisor 的高可用性？

如何保证网络 QoS？

(4) 云环境支持的业务和用户规模如何？

(5) 网络如何规划和搭建？

OpenStack 提供了两种网络组件可用。

- nova-network。

nova-network 提供了三种网络管理器，分别是扁平网络、扁平 DHCP 网络和 VLAN 网络，然而这些网络更适合于规模不大的私有云，而非公有云或大型私有云。其优点是非常容易支持，而且比较稳定和成熟。VLAN 是使用最广泛的网络拓扑，对企业现有网络改动最小，也最成熟和安全。对于非常小规模或单一的应用场景，则扁平网络是最佳选择。

- Neutron。

Neutron 除了上述三种网络，还提供了如 L3 Agent，容易支持 SDN 等软件定义的网络，提供了非常丰富的功能。然而，Neutron 的实施非常复杂，而且代码还有不少问题，对技术难度要求极高，如果不是 100 台计算节点以上规模，或必须提供 VPC，建议使用最简单和最稳定的 VLAN 和扁平网络，毕竟，稳定可靠和安全是企业最希望的。

8.2.2 云平台规划

对企业现在和未来预期的业务需求进行梳理之后，我们需要了解现有的数据中心基础设施和云环境的差距，来制定硬件采购计划和实施规划。

对云实施来说，关键信息如下。

- (1) 现有多少物理主机满足云主机需求？包括是否支持虚拟化，是否有足够的 CPU、内存、网卡等配置，是否可以根据配置差异进行分组。
- (2) 是否有外部存储设备？如果需要使用 Cinder 组件，那么社区是否已有比较成熟的驱动？对于软件定义的存储功能，是否提供了足够支持？
- (3) 如果需要对象存储来搭建云存储，那么是否有足够的存储服务器？是否有至少万兆网络来支持？
- (4) 如果希望用 Ceph 来提供块存储服务，或者用 GlusterFS 等来提供共享存储，那么是否有足够的网络带宽（比如万兆网卡和万兆交换机）来支持？
- (5) 是否有如 Power 等非 X86 硬件也希望能加入云基础设施？
- (6) 网络带宽是否足够？是否支持或打算购买 SDN 交换机？
- (7) 是否已经购买过其他虚拟化软件？
- (8) 旧有系统是否需要迁移进云环境？
- (9) 应用对物理设施的需求如何？如电源、网络等。

1. 常用应用类型的特性分析

表 8-1 对一些常用的应用类型从计算、网络、存储方面进行了比较。

表 8-1 常用应用类型特性分析

应用类型	计算吞吐 (CPU)	内存容量	网络带宽	存储 IOPS	存储容量	存储带宽
Web 应用	中/高	中	低	中	低	低
Hadoop	低/中/高	中	中	低	中/高	高
大数据	中/低	低	高	中	高	中
VDI	中	高	中	高	中	中
CDN	低	中	高	中	高	高
Memcache	高	高	中	高	低	中
Database	高	高	中	高	中	中
Cold Storage	低	低	低	低	高	低

通常情况下，客户都会新采购一些服务器专门用于 OpenStack 部署，但需要考虑：这些服务器硬件配置是否合适？是否可优化用于云平台？或如何最大限度地使用盘活企业旧有服务器资产？当面对这些情况时，需要从用户用例入手。

- 具体业务场景：数据中心是用来部署基于 Web 的生产环境应用的，还是开发&测试云的，或是用来替换内部 IT 环境？不同情况下的选择和方案都是不一样的。
- 通常都希望在配置上既要考虑到性能，又要考虑到费用；然而在实践中，只能在两者中间取一个平衡点，而无法使两者都是最优。
- 在硬件和系统配置中，考虑因素包括 CPU、内存、网络、存储。在实际过程中，非常容易过度投资于某个方面，而忽视另外的方面。
- 各个硬件厂商并没有彻底解决配置和性价比最优的问题。尽管有宣称专门优化用于“虚拟化”或“关键应用”的服务器，但这些特性都很难量化或比较。
- 尽管基于 VMware 的虚拟化/云有非常好的、实战测试过的硬件推荐配置，但基于 OpenStack 的此类最佳配置推荐目前还没有，许多厂商也只是推广自己的硬件和软件而已。

影响硬件配置选择的因素非常之多，然而为了简化，我们主要考虑两个参数。

- 你的云平台上需要运行的虚拟机总数。
- 每个虚拟机平均的资源（CPU、内存）。

下列配置也会影响硬件配置选择和计算，但更加复杂。

- 网络因素：网卡和交换机。
- 存储因素：临时存储和块存储。
- 不同用户场景和应用类型

此外还有如下一些约束。

- 不要引起内存的过载：当物理内存不够时，操作系统都会使用虚拟内存，而虚拟内存会引起虚拟机性能严重下降。
- 如果希望提高性能，对每个物理的核，不要计划超过 2 个 vCPU。
- 如果追求费用最低，对每个物理的核，不要计划超过 6 个 vCPU。
- 对每种不同厂商、型号的服务器配置，为了更多虚拟机和物理主机的操作系统运行正常，内存的使用率不应高于 60%。

如果觉得这些问题太复杂，不妨咨询一些专业的规划设计团队。

2. OpenStack 部署规划

云环境的规划和部署, 需要进行 OpenStack 技术与抽象的部署模型的映射, 然后对应到具体的物理基础设施中去。典型的工作如图 8-1 所示。图中包括三部分。

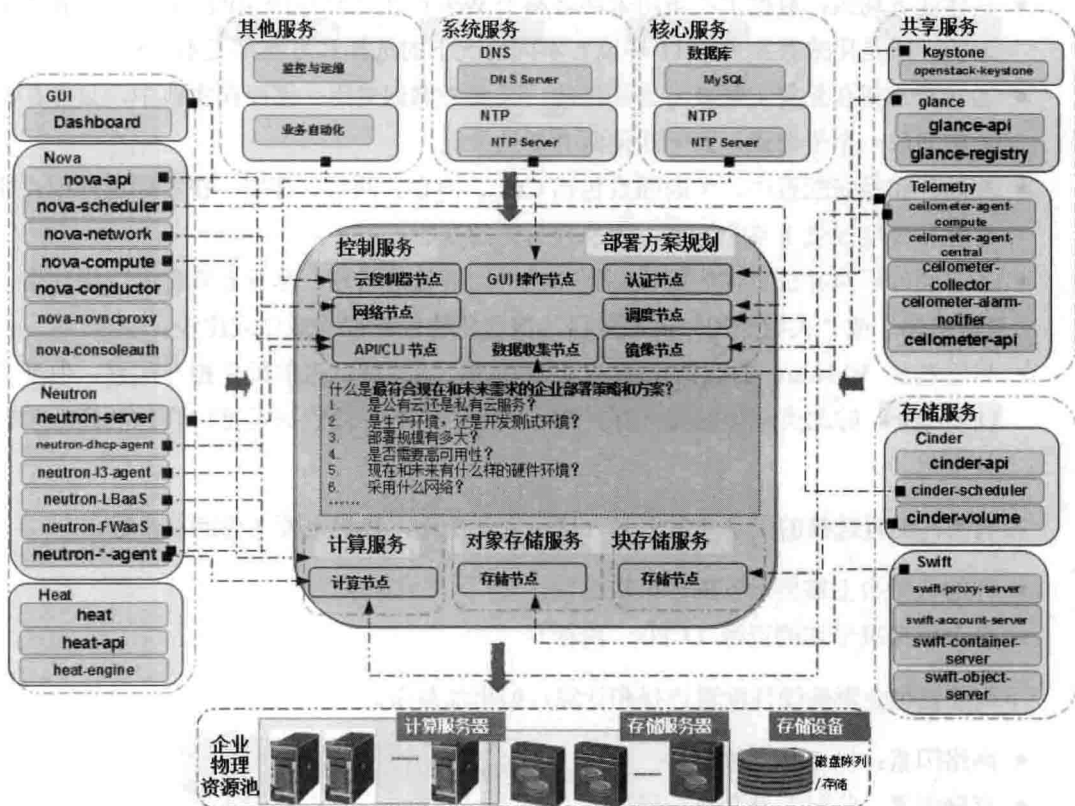


图 8-1 规划 OpenStack 到实际部署场景

(1) 图 8-1 中左、上、右部区域为 OpenStack 核心项目和组件, 包括如下部分。

- Dashboard、Nova、Neutron。
- Heat。
- 共享服务, 包括 Keystone、Glance、Telemetry。
- 存储服务, 包括 Cinder、Swift。
- 其他还有: DNS、NTP 服务; 数据库、消息中间件; 其他服务, 如监控与运维、云平台业务自动化等由企业自己部署或开发的功能与服务等。

(2) 图 8-1 的中间部分为抽象的逻辑节点, 包括如下部分。

- 涉及控制服务的抽象节点。
 - 云控制器节点，该节点将包括全部控制服务，但特定服务将运行于其他类型控制服务节点、计算节点、对象存储节点与块存储节点。当用户希望只部署单控制器时，则所有控制服务都部署于此节点，因此可以说，控制服务可以完全分散到多台物理服务器上，也可以全部部署于一台服务器。这也是 OpenStack 的强大和魅力之一。
 - 网络节点，提供了 OpenStack 如 DHCP、L3 路由与转发、LBaaS、FWaaS 等服务。
 - API/CLI 节点，是用户调用云 API 接口编写自己业务逻辑的访问点，也是系统管理员进行命令行操作的节点。它通常和 GUI 节点合二为一。
 - GUI 操作节点，部署了 Dashboard，或企业自己开发的云管理界面的节点。另外，如果希望用户也可以通过 VNC 等方式访问云主机，则需要部署相应的服务。
 - 认证节点，部署了 OpenStack 的 Keystone 服务的节点。通常，企业有自己的用户和权限认证系统，也需要和 Keystone 集成在一起。
 - 调度节点，OpenStack 有两种调度，分别为：nova-scheduler，用于合适的节点来启动和运行云主机；cinder-scheduler，选择合适的存储节点来申请卷。
 - 镜像节点，部署了 OpenStack Glance 服务的节点。
 - 数据收集节点，部署了 OpenStack Ceilometer 主服务的节点。
- 涉及计算服务的抽象节点。
 - 计算节点，即运行特定 Hypervisor，提供云主机服务的节点，但不同的计算节点资源池可以运行相同的或者不同的虚拟化管理程序。
- 涉及对象存储服务的抽象节点。
 - 对象存储服务的存储节点。对象存储服务的部署方式根据存储规模，差异很大，包括 All-in-One 部署方式、Proxy 节点+存储节点方式，以及各个服务完全单独部署等方式。
- 涉及块存储服务的抽象节点。
 - 块存储服务的存储节点，即部署了 OpenStack cinder-volume 服务的节点。

对于绝大多数中小型企业来说，控制服务 All-in-One 是最理想的控制服务部署方式。然而，有如下例外的情况。

- 用户部署的是 SDN 网络，需要多个负载均衡的网络节点来处理云平台的网络流量。
- 用户需要大量的云主机镜像模板，来支持其企业各种应用。此时，镜像服务需要单独的、提供高可用性的多个物理节点。

- 在大型云平台，统计和计量计费信息是非常巨大的，而且用户希望以 OpenStack 的 Ceilometer 为基础，使用企业自己的统计、数据收集工具，则数据收集节点需要由多台节点组成。
- 在计算节点超过千台规模的情况下，各个控制服务都会产生相当的压力，用户希望特定的服务节点可以提供主备高可用模式，或当容量进一步增长时，控制服务节点可以线性扩展，则相应的控制服务节点都会超过一台。

(3) 图 8-1 中最下方是数据中心物理设施的示意图。物理数据中心总是由物理主机、存储服务器、存储设备、各种网络设备组成。此外，构成数据中心的还包括电源、制冷、机架等。图 8-2 展示了一个大规模数据中心的机房。

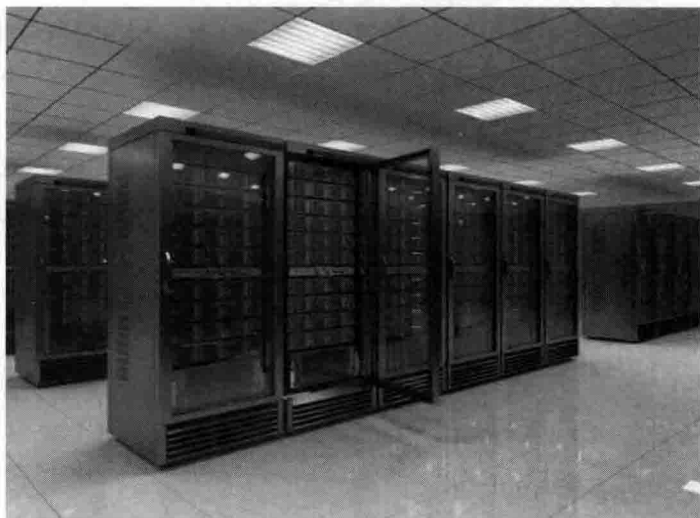


图 8-2 现代数据中心

8.3 区域和可用区

Zone 是一个资源管理的容器，可以映射到不同机房或者不同地区。每个 Zone 都可以拥有独立的 RabbitMQ-server、数据库、API Server，虚拟机可以在不同 Zone 之间进行调度。

许多客户数据中心，由于容量的限制、商业上的考量、价格的因素，或由于性能较低以及硬件的限制（如一个子网中可用的交换机数），无法容纳许多硬件，因此产生下列一些非常自然的划分。

- **Region:** 区域，物理上独立的数据中心。区域一定是物理上分布的。区域是用户可见的，用户需要选择离自己较近的区域，以达到较好的网络速度。
- **Availability Zone:** 可用区，即属于特定区域的数据中心。可用区不一定是物理上隔离的，也可以是纯粹的逻辑划分。可用区可以是用户可见的，但在一些企业里，也可以将其绑定到项目或租户，来实现一种对用户透明的调度。
- **Host Aggregate:** 主机聚合，进一步划分可用区为多个具有相同网络和存储的“组”，其通常只是管理员可见的。

以亚马逊为例，亚马逊 AWS 区域包括：

- 北美大区域；
- 亚太区域；
- 欧洲区域；
- 南美区域。

亚太区域包括：

- 新加坡区域，包括 2 个可用区；
- 东京区域，包括 3 个可用区；
- 悉尼区域，包括 2 个可用区。

如果是用户自己的数据中心，可参考图 8-3 来规划自己的区域、可用区。主机聚合通常是可有可无的。

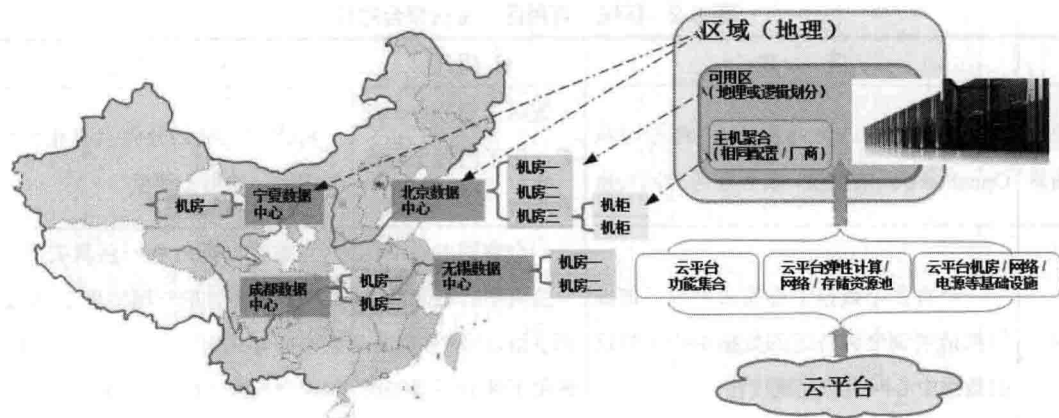


图 8-3 区域、可用区示例

区域、可用区：为了最大地提高 IaaS 服务的可靠性，每个区域包含多个可用区。可用区对应专属某个区域的分散的物理数据中心，通常通过多个可用区来提供一定形式上的物理隔离和冗余，例如，不同的电源供给、网络设施等。如果数据中心的一些机架有不同的电源，也可以将这些服务器划分为单独的可用区。可用区还可以用来划分不同类型的硬件，目前只有 nova-compute 服务拥有可用区的管理调度能力。

区域和可用区有下列一些约定。

- 一个区域包含多个可用区。
- 一个可用区只属于一个区域。

主机聚合：用于把主机池进一步划分成逻辑集群，共享某种资源（如网络 and 存储），或者有相同的特性，如可信赖的硬件资源和条件，来做同一可用区的高可用性、负载均衡和虚拟机分布策略，或无宕机的升级等，例如，集群，或者是同类 Hypervisor 主机的“计算资源池”或“组”。

最主要的应用场景是，将主机按照 CPU 和内存资源的配置划分为不同的主机集群，然后让不同的虚拟机类型调度到不同的主机集群，这样，高密度计算、低密度开发和测试可以共存于云平台，高密度不会受困于资源短缺，而低使用率应用也不会浪费资源。

区域、可用区和主机聚合的对比如表 8-2 所示。在 OpenStack 部署实践中，区域一级通常为单独而完整的 OpenStack 部署；而可用区和主机聚合则进一步切分单个 OpenStack 部署。

表 8-2 区域、可用区、主机聚合对比

	区 域	可 用 区	主 机 聚 合
使用场景	单独的 API 访问点，区域之间从 OpenStack 云的角度，没有任何协同关系	逻辑上划分一个部署，用于物理上隔离或冗余	将物理主机划分为具有相同特性的组，并用于调度
例子	一个有多个数据中心的云平台，把虚拟机请求调度到特定的数据中心（最近的数据中心网络访问速度快）	一个数据中心中，一些机架有不同的电源供给，或网络连接，或用于区分不同等级的硬件	将虚拟机调度到一些具有某些共同硬件特性的一组主机上，如数据中心有几组主机，他们的 CPU 和内存超配不同，则可将不同虚拟机类型调度到不同的一组主机去

续表

	区 域	可 用 区	主 机 聚 合
部署	<ul style="list-style-type: none"> 每个区域有不同的 API 访问点; 每个区域部署了一套完整的 Nova 	在 nova.conf 中配置	在 nova.conf 中配置
共享服务	Keystone	<ul style="list-style-type: none"> Keystone 全部 Nova 服务 	<ul style="list-style-type: none"> Keystone 全部 Nova 服务
可见性	用户可见, 如果要将虚拟机运行于不同数据中心, 需要明确选择	API 用户可见, 对计算节点资源池进行划分	管理员可见, 常用于 nova-scheduler

OpenStack 是高度可配置的, 以满足不同的计算、网络 and 存储选项的不同需求。图 8-4 展示了对于大型或者特大型部署规模, 可以对各个项目做出的各种组合。

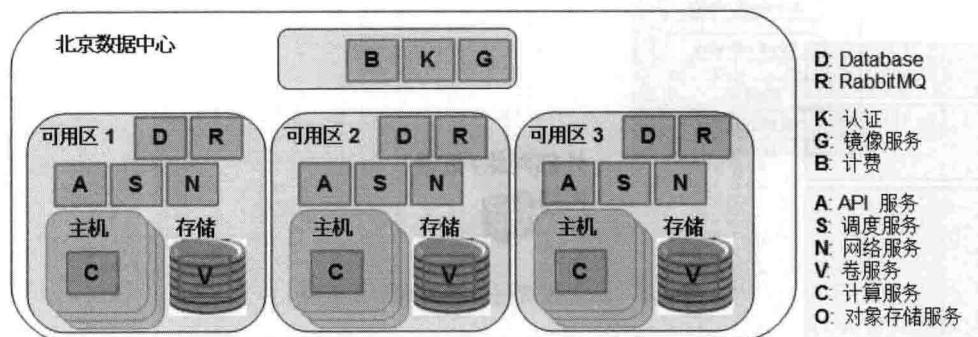


图 8-4 跨数据中心部署模型

8.4 典型部署拓扑

8.4.1 基于传统网络的基本部署架构

图 8-5 是一个基于传统网络的部署图。这种部署架构有下列特点。

- 控制节点部署全部控制服务, 包括认证服务 (Keystone)、镜像服务 (Glance)、控制面板 (Dashboard) 和计算服务的管理部分进程。该节点还包括了所有的 API 服务、MySQL 数据库和消息中间件。控制节点只有一个。

- 计算节点运行计算服务的 Hypervisor 部分，用来创建、运行和管理租户虚拟机。该节点也运行 KVM。计算节点也配置和运行租户网络、安全组等。计算节点可以运行多个。

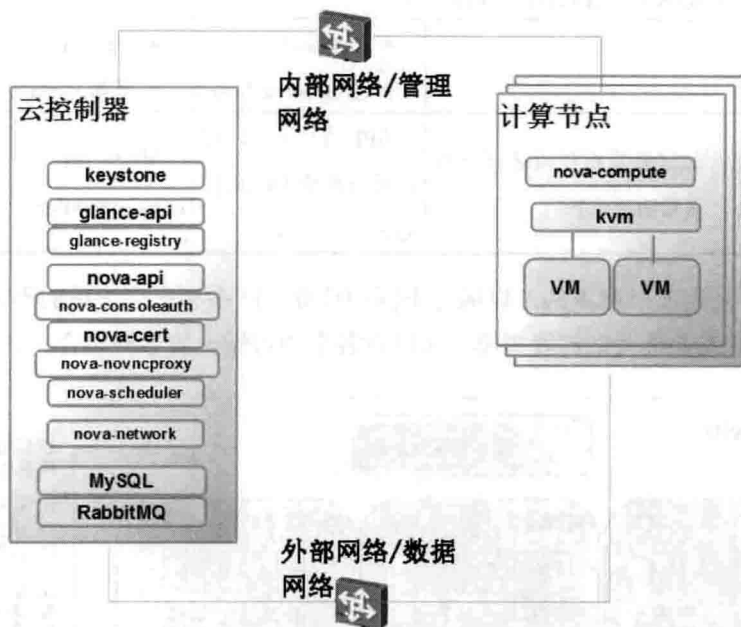


图 8-5 基于传统网络的基本部署架构

这是最基本的 OpenStack 部署拓扑之一，控制节点只使用一台物理机，没有专门的网络节点、存储节点。管理网络和数据网络分离，满足最基本的企业安全需要。

网络由于使用的是 nova-network，因此可以使用扁平网络、扁平 DHCP 网络和 VLAN 网络。由于只建立了两个网络，每台物理机最多需要两个网卡，降低了对硬件的要求。

由于没有部署存储服务（Cinder），因此，无法给虚拟机分配持久化存储作为卷挂载。虚拟机只能使用非持久化存储（本地硬盘），这样当虚拟机被终止之后，虚拟机内部数据将丢失。

另外，无论控制服务，还是计算服务，都没有提供高可用性，因此，在数据中心或主机意外断电、网络故障或磁盘损坏的情况下，会造成部分服务或功能不可用。

该部署适合于搭建开发/测试云环境，或运行非关键性的企业生产系统业务。

8.4.2 基于 OpenStack Neutron 的部署架构

图 8-6 是基于 OpenStack Neutron 的部署架构。这种部署架构有下列特点。

- 控制节点部署绝大部分控制服务，包括认证服务（Keystone）、镜像服务（Glance）、控制面板（Dashboard）和计算服务的部分管理进程。该节点还包括了所有的 API 服务、MySQL 数据库和消息中间件。控制节点只有一个。
- 网络节点运行网络插件 agent 和几个三层 agent，来配置租户网络，并提供相应的服务，包括路由、NAT、DHCP 等。它也用来处理外部网络与租户虚拟机的网络通信。
- 计算节点运行计算服务的 Hypervisor 部分，用来创建、运行和管理租户虚拟机。该节点运行 KVM。计算节点也运行网络插件 agent，配置和运行租户网络、安全组等。计算节点可以运行多个。

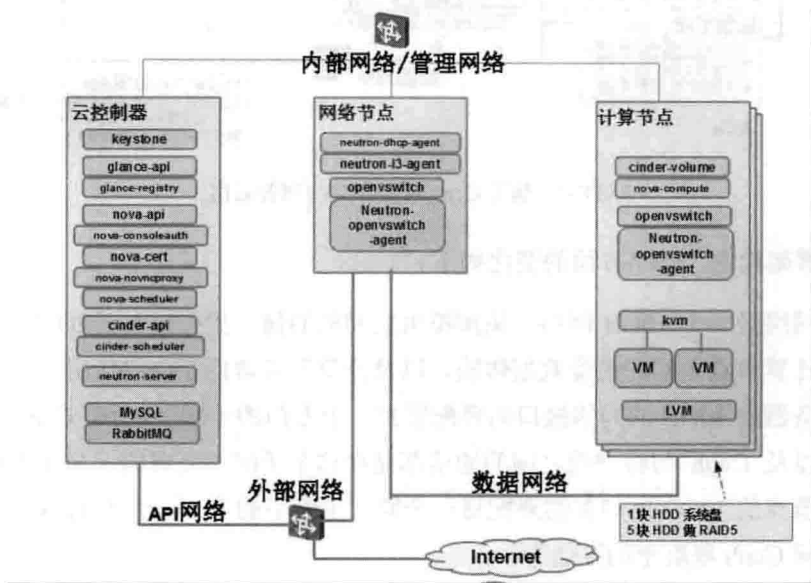


图 8-6 基于 Neutron 的部署架构

在这种部署架构中，引入了专门的网络节点，该节点将专门用来处理 SDN 的控制，以及网络三层交换等功能。

8.4.3 基于 Ceph 统一存储的部署架构

图 8-7 展示了一种基于 Ceph 统一存储的部署架构。Ceph 的引入，对部署至少带来下

列影响。

- Ceph 作为统一存储方案，既可以作为镜像服务（Glance）的存储后端，也可以作为块存储服务（Cinder）的后端，对两种服务的安装配置都带来了变化。
- Ceph 给网络带来很大压力，通常选择至少 10GB 网络，这样与网络内只支持/需要万兆网络的访问分离，同时因为镜像/存储需要同时访问 Ceph，对网络拓扑带来变化。

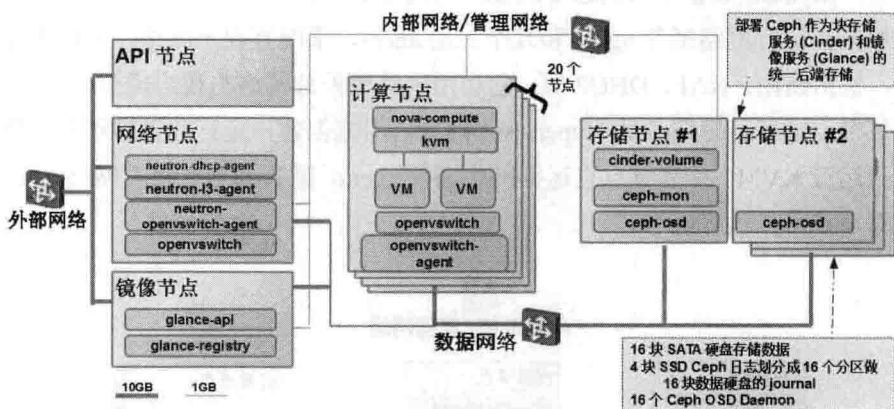


图 8-7 基于 Ceph 统一存储的部署架构

在本部署架构中，网络方面的变化如下。

- 数据网络是一个 10GB 网络，供虚拟机之间的通信、虚拟机与云外的通信、镜像节点与计算节点之间的镜像数据传输，以及计算节点访问存储卷使用。存储集群中所有服务器的 10GB 的网络接口需要配置到一个专门的子网下，所有 Ceph 服务之间的通信以及 Ceph 与客户端之间的通信都是在这个子网上完成的，与此相应的是每个计算节点的 10GB 网口都需要配置一个同一子网下的 IP 地址，只有这样计算节点才能访问 Ceph 集群里的存储卷。
- 外部网络：API 节点由于带宽需求并不高，可以选择 1GB 或 10GB 网络连接到控制节点；另外，镜像节点和网络节点由于需要较高的外部带宽，因此各有一个 10GB 的网口连接到控制节点和网络节点。

8.4.4 中型企业私有云部署架构

图 8-8 展示了一种针对中型企业的部署架构。这种部署架构有下列特点。

- 网络选取的是 VLAN 方式，VLAN 方式很好地实现了各个项目之间的隔离，保证了安全性。VLAN 包括：
 - 提供公共服务的网段；
 - 虚拟机内部通信的网络；
 - 预留 VLAN ID 段。
- 最左侧为控制节点，两个控制节点为 Active-Standby，任意一台出现问题之后，另外一台可以快速作为对方的备份节点。
- 左下角为自动部署的服务器，可以为物理机和虚拟机提供快速部署服务。
- 左边靠上的两台服务器，利用 GlusterFS 形成冗余，并可互为灾备节点，分别提供镜像和卷服务。
- 右上服务器代表一个机房中的服务器形成一个可用区（Zone），为特定项目提供开发测试虚拟机服务。
- 右中的服务器连接到 OpenStack 控制服务形成一个可用区（Zone），为整个公司提供内部 IT 服务。
- 右下的服务器是 Swift 服务器，作为企业网盘的后端。

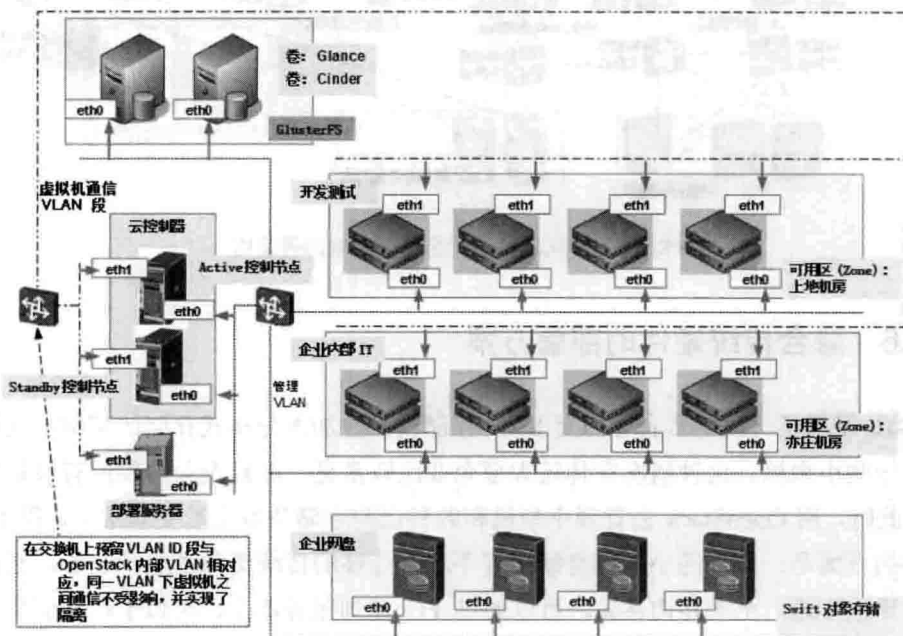


图 8-8 中型企业私有云部署架构

8.4.5 中型企业差异化资源池、多种存储池的部署

图 8-9 展示了一种大规模部署架构，其中包括了各种资源池，还有接入层和汇聚层交换网络。OpenStack 的部署是非常灵活和弹性的，可以根据各种需要进行调整 and 变化。

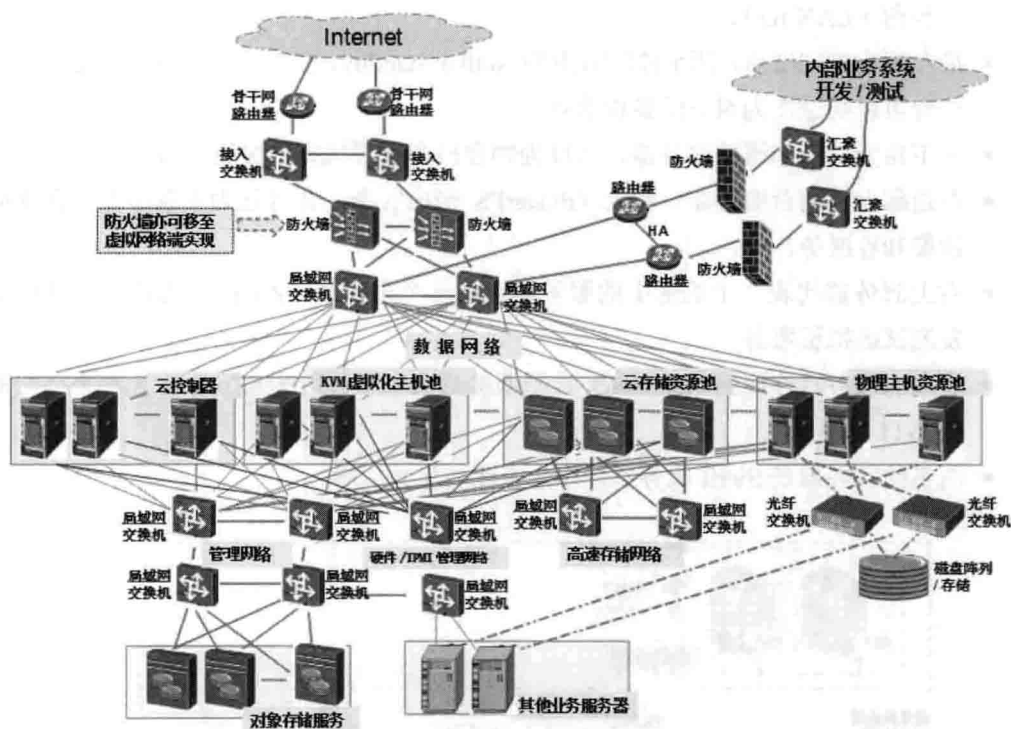


图 8-9 差异化资源池、多种存储池部署架构

8.4.6 融合传统硬件的部署方案

图 8-10 展示了一种整合企业旧有 SAN 存储，同时加入分布式存储的架构。这种企业同时还有一些小型机。这种情况在传统大型企业比较常见，而且企业内部旧有设备数量不在少数。此时，用 OpenStack 去管理小型机和各种已购存储基本上难度非常大，得不偿失。一个变通的方案是：原来的小型机继续运行不适合迁移的传统资源密集型业务，通过网络与新的云平台互通。至于异构存储，可以通过 FC 挂到服务器上，类似于存储服务器的本地磁盘。但这样会损失性能，并不是所有客户都接受。

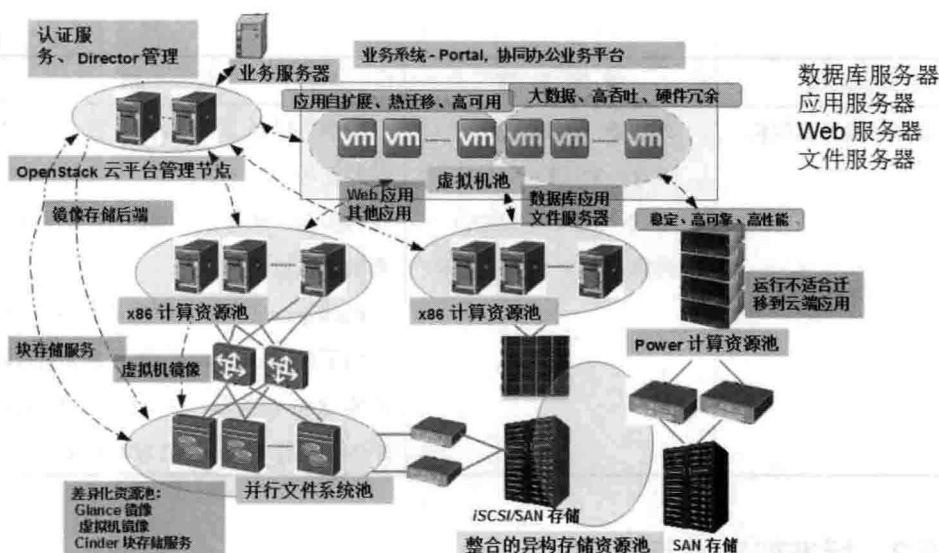


图 8-10 融合企业旧有存储整合部署架构

8.5 云平台硬件选择

8.5.1 试验环境推荐配置

表 8-3 是一种针对小规模实验环境的硬件配置信息。这些配置基本都适合实验环境或者演示环境。

表 8-3 实验环境推荐硬件配置

节点类型	推荐配置	说明
云控制器，运行 OpenStack 控制服务组件，包括网络、卷管理、调度器、镜像服务和 API	处理器：64bit X86 内存：12GB RAM 磁盘：30GB (SATA、SAS 或者 SSD) 卷存储：2TB 的两个磁盘 (SATA) 网络：一个 1Gbps 网卡	两个网卡是推荐配置，但一个网卡也可以运行控制服务。一个四核服务器，12GB 内存作为控制节点更好

续表

节点类型	推荐配置	说明
计算节点（运行虚拟机）	处理器：64bit X86 内存：32GB RAM 磁盘：30GB（SATA） 网络：两个 1Gbps 网卡	最小可以到 2GB 内存作为一个初步的测试验证环境，可以创建一个 m1.small 的虚拟机实例，或者三个 m1.tiny 但没有交换内存的虚拟机实例。 Rackspace 使用 96GB 内存作为计算节点。 为了运行 nova-compute，跑 Hypervisors，需要 X86 服务器，并支持 AMD 处理器的 SVM 扩展，或 Intel 处理器的 VT 扩展

8.5.2 标准部署推荐配置

如果用于生产环境，则硬件配置可参考表 8-4 中列举的信息来购买和配置。

表 8-4 标准部署环境推荐配置

节点类型	配置
控制节点	Model: Dell R620 CPU: 2 x Intel XeonCPU E5-2620 0 @2.00GHz Memory: 32GB Disk: 2 x 300GB 10000 RPM SAS Disks Network: 2 x 10G network ports
计算节点	Model: Dell R620 CPU: 2x IntelXeonCPU E5-2650 0 @2.00GHz Memory: 128GB Disk: 2 x 600GB 10000RPM SAS Disks Network: 4 x 10G network ports （Forfuture proofingexpansion）
存储节点	Model: Dell R720xd CPU: 2 x Intel XeonCPU E5-2620 0 @2.00GHz Memory: 64GB Disk: 2 x 500GB 7200RPM SAS Disks + 24 x600GB 10000 RPM SAS Disks Raid Controller: PERC H710P Integrated RAID Controller, 1GB NV Cache Network: 2 x 10G network ports

续表

节点类型	配置
网络节点	Model: Dell R620 CPU: 1 x Intel Xeon CPU E5-2620 0 @2.00GHz Memory: 32GB Disk: 2 x 300GB 10000 RPM SAS Disks Network: 5 x 10G network ports

8.6 控制节点的设计

8.6.1 硬件抉择

表 8-5 列举了一些在购买配置硬件时需要考虑的因素。充分的考虑可以让您的环境更符合需要。

表 8-5 硬件考虑因素

考虑点	细 化
同时有多少虚拟机运行	需要准确估计数据库的负载。这种压力发生在假如由许多虚拟机需要同时报告状态、被调度,或批量新虚拟机开始运行的时候
同时有多少计算节点运行	需要确保消息队列可以成功处理到来的请求
有多少用户需要访问 API	如果许多用户需要发起请求,确保 CPU 负载足够负担
有多少用户需要访问 Dashboard 界面	Dashboard 界面发起的请求甚至多过 API 请求,如果用户主要是通过 Dashboard 来访问,则需要额外的 CPU 处理能力
同时有多少 nova-api 服务运行	推荐每个服务对应一个核
单个虚拟机会运行多长时间	虽然启动和删除虚拟机实例主要是由计算节点来完成的,但也需要控制节点,包括 API 调用、调度等
认证系统是否由云外系统提供(如 LDAP 或 ActiveDirectory)	确保云控制节点和外部认证系统的网络连接稳定,而且控制节点有足够的 CPU 能力来处理请求

8.6.2 服务分离式部署

控制服务并不总是需要部署于同一个节点，也并不一定在物理机。表 8-6 列举了一些其他的部署方式。

表 8-6 服务分离式部署

类 型	说 明
将 glance-*服务与 swift-proxy 放在同一服务器	这种部署可以利用对象存储 proxy 服务器空闲的 I/O 能力，镜像传输在同一物理服务器上，同时与对象存储后端有非常好的网络连接
将数据库运行于专门的服务器	这种部署使用一个专门的数据库服务器服务于其他所有服务。可以搭建数据库的 master-slave，提供故障转移、隔离数据库本身的更新等
将服务运行于虚拟机	这种部署将控制服务运行于多个 KVM 虚拟机里。每个服务使用一个虚拟机，这将帮助快速部署、快速扩展，根据每个服务的负载来调整资源以提高性能（通常在部署时，无法估算实际的负载会有多高） 绝大部分服务都是可以被虚拟化的，如果性能下降，只需要很简单地运行多个服务实例即可。然而，一些服务必须运行于物理环境中，包括 nova-compute、swift-proxy 和 swift-object 服务
使用外部的负载均衡器	如果企业里已有外部负载均衡器，可以用它们来在多个物理服务器上运行多个 nova-api、swift-proxy 服务

8.6.3 数据库

OpenStack 服务使用 SQL 数据库来保存和获取状态信息。MySQL 是最流行的 OpenStack 部署环境使用的数据库。

然而，数据库也会导致错误。因此，推荐搭建数据库的集群来做容错和线性扩展。

8.6.4 消息中间件

消息总线集群是许多 OpenStack 部署环境中的痛点。

RabbitMQ 有自身的集群支持，然而在大规模环境中还是出现不少问题。

QPID 是 Red Hat 和 CentOS 的消息中间件选择，但 QPID 本身并不支持集群，需要借

助如 Pacemaker 和 Corosync 来支持集群能力。

8.6.5 认证与授权

OpenStack Keystone 支持插件式的认证，这些可选的插件包括：

- SQL 数据库（如 MySQL 或 PostgreSQL）；
- PAM（Pluggable Authentication Module）；
- LDAP（如 OpenLDAP 或 Microsoft's Active Directory）。

许多部署选择了 SQL 数据库，然而对于许多已经拥有认证系统的企业来说，LDAP 是一个最好的选择，也易于集成。

8.6.6 网络

云控制节点需要处理多种不同的服务请求，例如，如果镜像服务部署在该节点，需要有足够的带宽来在可接受的速度下传输镜像文件；如果网络服务部署其中，则该节点变成所有虚拟机实例的网关，那么云控制器就必须能支持所有云与外网的网络传输。

在这些情况下，高速网卡都是必须的。

8.6.7 计算节点的设计

1. CPU 的选择

CPU 的类型是非常重要的，首先确保 CPU 支持虚拟化，如 Intel 芯片支持 VT-x，AMD 芯片支持 AMD-V。

核的数量也影响决定，当前大多数 CPU 都是 4 核以上，如果 Intel 支持超线程（Hyper-threading），则 4 核就变成 8 核。超线程是 Intel 专利技术，让多线程软件可在系统平台上并行处理多项任务，来提升 CPU 处理器执行资源的使用率。在实施时，可以考虑启动超线程，来提高多线程应用的性能。

2. Hypervisor 的选择

OpenStack 支持多种 Hypervisor，包括：

- KVM
- LXC
- QEMU
- VMware ESX/ESXi
- Xen
- Hyper-V
- Docker

当选择 Hypervisor 时，也许最主要的决定因素来源于当前使用什么、费用或者经验。其他还包括功能支持、文档、社区经验等。

KVM 是 OpenStack 社区使用最高的 Hypervisor。除了 KVM，其他部署还包括 Xen、LXC、VMware 和 Hyper-V。但是，这些 Hypervisor 都或多或少地缺乏一些功能实现，或者基于 OpenStack 的最佳实践或文档。

查询 <https://wiki.openstack.org/wiki/HypervisorSupportMatrix>，可以了解每种 Hypervisor 当前的社区支持程度。

<https://wiki.openstack.org/wiki/HypervisorSupportMatrix> 可以了解每种 Hypervisor 当前的社区支持程度。

如图 8-11 所示的一种逻辑架构，OpenStack 完全可以同时管理多种 Hypervisor 组成的计算资源池，这些不同 Hypervisor 组成的资源池以“可用区”的方式来调度。

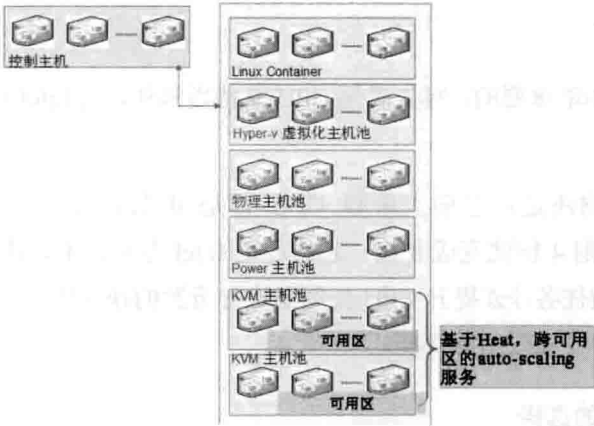


图 8-11 多控制节点、多虚拟化管理程序逻辑部署模型

OpenStack 社区默认的 Hypervisor 是 KVM，所以 KVM 是基于 OpenStack 使用最多、最广的 Hypervisor，然而企业情况总是非常复杂的。

- 基于 IBM 对 OpenStack 的大力支持，相关接口也已经逐步公开出来。如果用户已经购买了 IBM Power 小型机，并且希望也可以被云平台所管理，则可以通过部署 Power 驱动来让同一个控制服务节点同时管理 KVM 的 X86 计算节点和 Power 计算节点。目前，Power 驱动从社区中下架，IBM 实施部门则有内部可用的驱动，购买 Power 的客户，可以同 IBM 实施部门联系以取得 OpenStack 对 Power 节点的支持，或查询社区是否有新的可用的驱动发布出来。
- 若已经购买了 VMware ESXi/VC，OpenStack 可以通过 vCenter 来管理 ESXi 节点，相关的驱动正在开发中，主要取决于 VMware 开放了哪些接口。希望让同一套 OpenStack 管理 ESXi 节点的客户，需要查询社区是否有可用的驱动发布出来。
- 如果用户对性能有很高要求，但是对虚拟机的其他一些操作，比如调整资源大小操作（resize）、卷挂载、热迁移等没有要求，则可以部署 LXC、Docker 等。
- OpenStack 控制节点还可以管理物理主机，但相关功能还未完成。

3. 虚拟机的存储

非持久性存储，在这里指虚拟机创建时分配的存储空间，在虚拟机被销毁时，空间会被回收，而存储于这些存储空间的数据会全部丢失。

该空间有三个来源。

- 非共享的本地系统（独占）。
- 共享的本地系统。
- 共享的外部系统。

许多部署环境采用了计算节点和存储节点分离的方式，因为它们有着不同的需求。

- 计算节点通常需要更多的 CPU 和内存。
- 存储节点需要更多的块存储设备。

然而，如果物理机有限，如果需要跑更多的虚拟机，那么把计算和存储部署在一起也是一个理想的选择。

（1）共享的外部系统

在这种情况下，运行虚拟机使用的系统来自于其他服务器。这种部署带来以下一些好处。

- 如果计算节点失效，虚拟机实例通常比较容易恢复；
- 分离的存储系统一般比较容易管理和维护；
- 容易扩容和扩展；
- 共享的外部存储还可以用于其他目的。

缺点如下。

- 如果一些虚拟机产生很高的 I/O，会影响到其他不相关的虚拟机；
- 网络可能会降低性能。

(2) 共享的本地系统

在这种情况下，每个计算节点配置许多本地磁盘，但是通过分布式文件系统，让这些不同节点的空间组合成一个统一的空间。

这种部署的好处是，在需要更多存储时，很容易扩展，甚至扩展到其他外部存储。

缺点如下。

- 不易确定数据存储的真实位置；
- 恢复一个虚拟机实例比较复杂，依赖于多个节点；
- 网络会降低性能。

(3) 非共享的本地系统

在这种情况下，每个计算节点有足够的磁盘来存储本机的虚拟机实例。

这种部署的好处如下。

- 一个 I/O 压力很大的计算节点，不会影响其他计算节点；
- 直接的 I/O 访问可以提高性能。

缺点如下。

- 如果本计算节点失效，则运行于本机的虚拟机实例也丢失；
- 如果需要更多存储，这种方式不容易扩容；
- 虚拟机从一个节点迁移到另外节点变得复杂，也许依赖于一些未支持的功能，或更容易出错。

在计算节点之外的存储系统上提供共享系统，对于云平台来说，是个比较理想的选择，毕竟可靠性和可扩展性是云平台非常重要的因素。

至于需要使用计算节点本地共享系统，一个可能的场景是管理一个以前已经存在且对应用的配置无法控制的情况。使用计算节点本地文件系统，适用于 I/O 要求很高的应用，而不必担心其可靠性。

4. 热迁移

当主机需要维护或升级时，可以将虚拟机通过热迁移的方式移到另外一个计算节点上去，此时，被迁移的虚拟机仍然在正常工作，可以被访问到。另外，在使用率不高的下班时间，可以将虚拟机迁移到少数计算节点去，将空余出来的计算节点关机，达到省电的目的。

热迁移通常基于共享存储，KVM 也支持基于块的热迁移，但是对配置要求比较高，容易失败。

文件系统的选择

为了支持热迁移，通常需要分布式文件系统支持，可能的选择有：

- NFS
- GlusterFS
- MooseFS
- Lustre

以上的部署在生产环境都有先例。选择时，选取自己最熟悉的。否则 NFS 也是个选项，它的搭建非常容易。

5. CPU 和内存的超配

OpenStack 允许在计算节点上超配（Overcommit）CPU 和内存，在降低虚拟机性能的前提下，来增加单个计算节点上运行的虚拟机数量。

OpenStack 默认超配比例如下。

- CPU 超配比率：16:1。

对于一个物理的核，调度器会按照 16 个虚拟核来分配和调度。如果一个物理节点有 12 个核，意味着可以分配的核有 192 个。如果一个虚拟机需要 4 个虚拟的核，则总共可以提供 48 个虚拟机。

- 内存超配比率：1.5:1。

调度器可以持续往一个计算节点上分配虚拟机，只要这些虚拟机总共内存不超过物理机内存的 1.5 倍。例如，如果一个物理节点有 48GB 内存，则该节点上虚拟机总

共可以分配到 72GB 内存。

在实践中，如果虚拟机实际使用内存过多，引起物理机换页，即使用交换内存，则虚拟机性能将严重下降。

8.6.8 存储的选择

存储既和数据的可靠性相关，也和性能相关。在用户场景中，用户表达出不同的需要：一些用户需要快速地访问那些不经常变化的对象数据，或者给文件设置一个存活时间(Time to Live)；另外的用户则希望访问通过文件系统挂载过来的数据空间。

如表 8-7 所示，存储有许多种选择，而且有些需要的功能不止一种选择。在计划存储后端时，需要考虑以下一些问题。

- 是否需要块存储？
- 是否需要对象存储？
- 是否需要支持热迁移？
- 块存储驱动部署在计算节点，还是用外部存储？
- 需要多少存储容量？
- 在费用和性能之间，哪个最重要？
- 如何管理存储？
- 存储需要达到什么程度的冗余和分布？如果存储节点失效，多大程度上可以降低数据丢失的后果？

持久化可以通过基于分布式文件系统的存储来支持。

表 8-7 存储支持矩阵

	对 象	块	文件级（支持热迁移）
Swift	Y		
LVM		Y	
Ceph	Y	Y	测试中
GlusterFS	Y	Y	Y
NFS		Y	
ZFS		Y	
Sheepdog		测试中	

无论是 Linux, 还是 Windows, 用户应用通常都会使用到多个卷(磁盘): 在 Linux, 包括系统卷、交换卷、数据卷等, Windows 则通常标识为 C:、D:等。

卷分为逻辑的和物理的。

- 卷可以通过软件, 如 fdisk 或 Windows 系统工具, 分区或格式化成多个卷使用。
- 卷还可以通过 OpenStack 的 Cinder 服务, 来获取并挂载到虚拟机上。其效果相当于在物理机上增加了一块磁盘。

在一些特殊情况下, 通常无法使用/配置 Cinder, 虚拟机使用的是非持久化存储, 这意味着, 在虚拟机被终止之后, 这些“磁盘”空间会被释放和回收, 用户数据也将丢失。这种情况也可以满足一些业务场景。

只能使用非持久化存储的情况如下。

- 高可用情况。虚拟机在物理机断电, 网络/网卡故障, 主机磁盘出错时, 需要在另外的计算节点上恢复运行, 但面对的问题有: 或者虚拟机的卷由本地 Cinder 服务来提供; 或者无法从技术上解决虚拟机迁移后, 卷的重新挂载问题。
- 对于企业使用的存储设备, 社区还没有相应的 Cinder 驱动来供选择。

通常, 非持久存储默认来自于计算节点本机硬盘, 但也可以是其他方式。

- NFS 共享存储。
- 存储设备挂载到本机后生成的目录。
- 通过分布式/并行文件系统得到的共享存储。

然而, Cinder 提供的块存储, 可以让虚拟机根据需要得到非常灵活的存储能力, 卷可以独立存在、挂载到虚拟机, 或从虚拟机卸载。Cinder 管理的块存储来源如下。

- 本机 LVM。
- 存储设备, 并安装存储设备针对 Cinder 的驱动。
- 分布式/并行文件系统, 如 Ceph、GlusterFS、Sheepdog。

客户有共享存储需求的情况, EMC、NetApp、赛门铁克的存储方案都可以接入。基本上只要厂商提供的就可以用。例外的是, 有一个公司用 Swift 来支持其监控录像存储的应用, 这应该是特例, 因为目前很少有人这么用。对此需要进行评判, 特别是按照实际业务来进行测试, 看是否满足性能的要求。

共享存储主要是为了应对有迁移、虚拟机 HA、大数据类应用需求而储备，在这方面，文思海辉团队对 Ceph、GlusterFS、Sheepdog 和 Swift 都在进行测试，各种解决方案都有各自的优势。

1. 商用存储后端技术

(1) OpenStack 对象存储 (Swift)

OpenStack 正式支持的对象存储实现，是一个非常成熟的，已经在 Rackspace 生产环境中使用多年的技术。它是高度可扩展的，非常适合管理多到 petabytes 级别数据的存储技术。它深度集成到 OpenStack 中，包括认证服务 (OpenStack Keystone)，可以从 OpenStack Dashboard 上访问。它通过异步最终一致性复制，非常好地支持多个数据中心的部署。

因此，如果计划现在或以后支持跨越多个数据中心的存储集群，或者需要对计算和对象存储统一的用户账户管理，或者希望通过 OpenStack Dashboard 来管理对象存储，Swift 是一个最佳的选择。

OpenStack 对象存储解除了很多传统文件系统的约束，提供了一个高度可扩展的、高度可用的存储方案。这种类型存储基于一个认识：所有的硬件在任何层次、任何时间点和任何部件都可能会失效（而不是传统存储系统，千方百计要避免硬件故障或失效）。传统存储系统需要面对的问题，如一些故障引起 RAID 卡挂住或整个存储服务器无法访问，在对象存储里都得到了很好的解决。

因此，在设计对象存储集群时，首要考虑的问题是数据存放时间和可用性，即数据如何分布和放置，而不是硬件的可靠性。

在部署 Swift 集群时，需要考虑让服务器尽可能地分布在整个数据中心，跨越多个网络、电源、机架等。

Swift 对网络带宽要求很高，另外，增加 proxy 也意味着从部署上增加带宽。

(2) Ceph

Ceph 是一个可扩展的存储方案，在多个存储节点上复制数据。Ceph 最开始由 DreamHost 的一个创始人开发，现在还用于他们的生产环境。

Ceph 被设计用来同时为用户提供不同种类的存储接口，包括对象存储、块存储和文件接口，现在文件接口还不能用于生产环境。Ceph 提供了和 Swift 同样的对象存储 API，也可以用于 Cinder 块存储的后端和镜像服务 (Glance) 的存储后端。

Ceph 通过 Copy-on-Write, 支持自动精简配置, 这个特性在从卷启动时非常有用, 速度很快。它也支持基于 Keystone 的认证, 还可以让管理员精细地管理数据的分布、复制策略。

(3) Gluster

Gluster 是一个分布式的共享文件系统, 可以像配置硬件 RAID 一样, 配置 Gluster 来提供数据的可靠性和访问性能。并且, Gluster 也是可扩展、非常容易部署和使用的。目前已有许多企业部署 Gluster 用于生产环境, 数据从 GB 到 TB 级别。从版本 3.3 开始, 可以将对象存储和文件存储整合为统一的文件/对象存储系统, 成为 Gluster UFO。Gluster UFO 可以作为后端, 支持一个定制化后的 Swift。

使用 Gluster UFO 的好处在于, 如果需要同时支持一个分布式文件系统, 或者作为共享存储用于热迁移, 或者作为一个单独的文件系统提供给用户。

(4) LVM

LVM (Logical Volume Manager) 是一个基于 Linux 的, 在物理硬盘之上提供了一个抽象层, 给操作系统提供了逻辑卷。LVM 后端将块存储呈现为 LVM 的逻辑划分。其原理相当于将一个物理硬盘格式化成几个逻辑分区。

在需要运行块存储服务的主机上, 管理员必须首先创建一个 Volume Group, 指定专门为块存储服务使用。然后, 从 LVM 逻辑卷里创建需要的块 (逻辑磁盘) 给虚拟机使用。

LVM 不提供任何数据复制, 通常, 管理员会在这些通过 LVM 提供块存储服务的节点上配置 RAID, 来保护数据, 不会因为部分硬盘的失效而丢失数据。然而, LVM 无法在主机本身失效时得到保护。

(5) Ceph 和 Swift 的选择

Ceph 和 Swift 都提供对象存储服务, 而且在许多方面都有相似的特性。它们的特点比较如表 8-8 所示。

表 8-8 Ceph 与 Swift 对比

	Ceph	Swift
项目历史	项目开始于 2006 年	项目开始于 2008 年
实现语言	C++, 性能调优	Python, 灵活的中间件嵌套, 易于集成 WSGI 管道或不同的认证系统

续表

	Ceph	Swift
一致性	强一致性，在写入时检查一致性，完全写入后，才发送完成信息给调用客户端	最终一致性，即使硬件故障发生，数据被写入，后台进程保证数据最终一致性和高可用性
功能	块存储（基于其强一致性，目前很多应用场景用 Ceph 为 Cinder 块存储后端） 对象存储 文件存储（目前不可用于生产环境）	对象存储，是目前最好的对象存储选择。目前已经用于一些非常大型的生产公有云环境，并得到验证 可配置的跨数据中心数据复制、备份和容灾方案

因此，建议的选择如下。

- 如果只需要块存储服务，Ceph 是优先的选择。
- 如果只需要对象存储服务，Swift 是优先的选择。
- 如果同时需要块存储服务和对象存储服务，但不想管理两种不同的集群，则 RadosGW 可以适用于很多简单的用户场景。
 - 然而，RadosGW 的类 Swift API 没有提供全面的对象存储服务功能。
 - 部署 RadosGW 环境后，从 RadosGW 的块存储环境不能同时访问对象存储环境，因为两种服务有不同的使用模式，必须搭建不同的硬件环境。

如果是 I/O 高的业务则不推荐放在此类分布式存储上，这是网络存储的实现所决定的。如 AWS EBS、阿里云的 I/O 普遍都是 30~40Mbps，对于此类业务普遍无法支持得很好。对于非高 I/O 类型的重要业务，都是可以的。

2. 存储的性能

发现磁盘访问的 I/O 瓶颈一般比较困难。磁盘的 I/O 包括物理硬盘上的读/写操作，如果从磁盘上读取一个文件，处理器需要等待直到文件被读取出来（写操作也是一样的）。

当面对磁盘时，最要紧的是什么呢？访问时间，这个时间是计算机得到处理器的数据请求，然后从存储设备取得这些需要的数据所花费的时间。因为磁盘是磁性的，我们需要磁盘旋转到需要的磁盘扇区然后才能读取数据。

磁盘的延迟大约为 13ms，但是依赖于硬盘的质量和选装速度。内存的延迟约 83ns，区别有多大呢？如果内存是一个速度最高为每小时 1 190 英里（每英里为 1.6km）的 F-18 Hornet（超过 1.5 倍声速），磁盘访问的速度就是一个蜗牛，最高速度每小时只有 0.007 英里。

因此，将数据缓存在内存对于性能非常重要，内存和硬盘访问的延迟的差别是非常巨大的。

例如，让我们花费 1s 从 MySQL 数据库去取 10 000 行数据，然后在这些数据行上执行一些操作。此时，通过磁盘访问来取得那些需要的数据。在这个时候处理器在等待磁盘去获取数据，因而变得空闲。在这个例子中，磁盘访问需要 700ms，所以 I/O 等待是 70%。可以用 top 这个 Linux 上最常用的命令来查看 I/O 等待的比率。

如果 I/O 等待超过了 (1/CPU 核数)，那么 CPU 将要等待磁盘子系统很显著的一段时间。如果 I/O 等待持续在这个阈值，磁盘访问可能会严重减缓应用的执行。

对于随机磁盘访问，应用类型如果是数据库、邮件服务器、文件服务器，需要注意每秒有多少个读/写操作 (IOPS)。有以下四个因素会影响 IOPS。

- 多磁盘阵列：阵列里的磁盘越多，则 IOPS 越大。如果一个磁盘可以是 150 IOPS，两个磁盘就是 300 IOPS。
- 每个磁盘的 IOPS：如果一个磁盘可以处理的 IOPS 越大，那么则总共的 IOPS 也就越大。这很大程度上取决于磁盘的转速。
- RAID 的因素：许多应用使用的存储通常都是用了 RAID 配置，意味着用多块磁盘提供了可靠性和冗余。然而，一些 RAID 配置对于写操作有很大的负面影响，例如 RAID 6，每个写操作需要 6 次磁盘操作，对于 RAID 1 和 RAID 10，每个写操作只需要 2 次磁盘操作。磁盘操作数越少，IOPS 就越高。
- 读写的数量：如果数据处理中有大量的写操作，另外选择的 RAID 对于每次写请求又有许多步的操作，比如 (如 RAID 5 和 RAID 6)，那么 IOPS 也会变得非常低。

如何计算最大 IOPS？一个准确的方法是去判断离最大 I/O 吞吐有多近。可以通过得到理论的 IOPS，然后和真实的 IOPS 去比较。如果两个数字很近，则可能是有 I/O 问题。

可以用下面的公式来计算理论 IOPS：

$$\text{每秒 I/O 操作 (IOPS)} = (\text{磁盘数目} \times \text{一块磁盘每秒平均 I/O 操作}) / (\text{读操作}\% + (\text{Raid Factor} \times \text{写操作}\%))$$

这个公式中除了一个数据之外，其余数据都可以从硬盘规格中获得。我们需要估算出读/写所占的数据比率，而这个是和应用密切相关的。对于这个问题，可以使用工具如 sar，也可以安装 Scout Device 输入/输出插件，它可以报告读和写的吞吐。

在计算得到了理论上的 IOPS 之后，可以用 `sar` 显示出来的界面上的 `tps` 列去做比较。`tps` 列指出了每秒有多少数据传输到存储设备去，这是真实的 IOPS。如果 `tps` 已经接近理论 IOPS，说明可能已经接近极限。

尽管可以对磁盘性能调优，但是仍然无法赶上内存的速度。如果遇到磁盘 I/O 瓶颈，只是单纯地对硬件调优并不是最快的途径。硬件的改变通常意味着大量的测试、数据迁移，以及应用开发者和系统管理员的大量交流。如果遇到磁盘 I/O 瓶颈，通常都会调试服务/应用，让它使用内存来缓存许多数据，例如对于数据库，让服务器拥有尽可能多的内存，然后让 MySQL 尽可能多地缓存数据到内存中去。

在一个数据负载很重的服务器里，非常重要是对性能进行分析评测，判断如何通过改变应用来影响磁盘的性能。临时地查看 `top` 的输出并不能说明什么，这只是临时的内存使用高峰，可行的是分析 I/O 等待和两个月前有什么变化。

8.6.9 网络的选择

OpenStack 分别提供了 `nova-network` 和 `neutron` 组件，用于提供软件定义的网络功能。`nova-network` 提供了扁平网络和扁平 DHCP 网络，非常适合于小型、网络拓扑简单的应用场景，比如，开发测试云。而 `Neutron` 则由于逐渐成熟，在新的部署中使用，特别是希望能逐渐过渡到 SDN 网络的客户环境中。VLAN 网络则非常适合于简单的企业级应用场景，其对现有企业网络改动最小，而且技术非常成熟，实施难度极小。

1. 公有访问地址的选项

用户虚拟机有两种类型的 IP 地址：固定（Fixed）IP 和浮动（Floating）IP。固定 IP 地址在虚拟机启动时分配；而浮动 IP 地址是少量的，可以由用户分配，并可以在虚拟机运行过程中分配给其他虚拟机。两种类型的 IP 地址都可以是公有地址，也可以是私有地址，取决于使用场景。

固定 IP 地址在私有云里是私有的，而在公有云里是公有的。

浮动 IP 地址在以下一些场景下需要。

- 公网 IP 地址总是有限的，当需要从公网去访问私有云时，可将一些浮动 IP 地址分配给那些需要从公网访问的虚拟机。
- 为一些用户分配一些静态地址，当虚拟机被升级或移动之后，这些 IP 地址还可以被

重新分配。

2. IP 地址的规划

OpenStack 部署，取决于规模大小，可能会有许多子网。一个详细的 IP 地址计划可以很好地增进对网络划分和扩展方面的理解。表 8-9 列举了网络的类型和各种网络的目的。需要说明的是，控制服务也可以具有公有和私有地址。

表 8-9 网络的选择

网络类型	说 明	使用用例
管理网络	提供 OpenStack 组件间的内部通信，IP 地址通常只能在数据中心内部访问。用单独的网络确保系统管理和监控访问与虚拟机网络分离，避免来自于用户虚拟机的监听和攻击，确保云平台的安全性	
数据网络	在云环境中，供虚拟机之间相互通信，当虚拟机跨越物理主机时，由于物理机的网卡被设置为 trunk 模式，所以虚拟机相互之间的二层数据包通过物理交换机后直接可到，相互就像连接到同一个交换机上	
外部网络	在一些部署场景里，提供了虚拟机访问 Internet。任何 Internet 上的用户都可以到达这个网络上的 IP 地址	
API 网络	用来提供所有 OpenStack API 给租户。在这个网络上的 IP 地址应该是 Internet 上所有用户可达的。这个网络通常和外部网络是同一个，因为通常可以只分配外部网络的子网供其使用	可以访问 swift-proxy、nova-api、glance-api、Dashboard 等服务，可以是负载均衡器一边的地址，或一个指定节点的地址
存储网络	在很多部署环境中，搭建单独的二层网络，供存储设备与 cinder-volume 之间使用（这里指基于 IP 的协议）；也可以用于大型分布式/共享文件系统间	对于 Gluster 集群，集群节点内部组成一个单独网络；或者在对象存储部署中，Object/Account/Container 服务器间，以及与 proxy 组成的一个私有网络
硬件管理网络	在大型数据中心，通常建立单独的硬件管理网络，用于管理员远程配置 BIOS，远程启动或关闭硬件服务器，查询系统信息，而不用到机房的控制台操作	通常连接计算节点或存储节点的一个 1GB 的网口，组成一个网络。避免走公有网络

需要注意的是，虚拟机 IP 地址需要保留一个较大的、扁平的地址池。

另外，如果是小型的部署，分配的 IP 地址也许只是企业当前数据中心某网段的一部分，此时需要谨慎，避免因错误而引起企业其他系统的网络访问问题。

当前在很多企业里万兆交换机仍然是个稀缺品，主力仍然是千兆交换机，在某些企业里甚至还在用百兆交换机。

如果使用了 SDN 技术，在数据转发方面，可以基于一些经验从下面几个方面尝试进行一些优化。

- 将 OVS (Open V Switch) 流表查找转移到机柜接入交换机 (TOR)，节省服务器 CPU 资源，提升查找性能。
- 将 GRE tunnel 的封装、解封装卸载到机柜接入交换机，让网卡仍然可以对 TCP 报文进行分片加速，大大提升网络性能。
- 所有流表全用“proactive”预先配置，有效减少未知报文广播和流表学习所带来的性能问题。
- 将 L3 网关从一个集中的网络节点卸载 (offload) 到机柜接入交换机。
- 在 TOR 交换机上启用 ARP (地址解析协议) 代理，避免 ARP 广播。

通过这些措施可以将网络处理的部分工作从服务器移到 TOR 交换机，让服务器可以专注于计算，降低带宽损耗，降低 CPU 负载，一台物理服务器可以容纳更多虚拟机，来降低成本，提升用户体验。

同时，由于 GRE tunnel 的封装放在了 TOR 交换机上，TOR 交换机可以直接看到用户数据，可以对用户数据进行统计、限速等策略应用，解决了纯软件方式带来的网络可视化问题，有助于云服务提供商对网络进行诊断和管理。

对于最终用户来说，虽然他们看不到 SDN，但是也会享受到 SDN 带来的好处。在公有云平台的用户希望可以进行自助服务 (self-service)，自己创建虚拟机，自己创建虚拟网络，自己创建虚拟路由器，自己创建防火墙等。谁在底层支撑着这些动作的顺利执行？是 SDN 架构。没有 SDN，就没有这一切。SDN 的最高境界就是用户在享受着 SDN 带来的便利但却并没有意识到自己使用了 SDN。

3. 网络拓扑

OpenStack 提供了几种网络实现，如表 8-10 所示，每种都有优点和不足。

表 8-10 网络类型优缺点对比

类 型	优 点	不 足
Flat	<ul style="list-style-type: none"> • 非常简单 • 没有 DHCP 广播 	<ul style="list-style-type: none"> • 需要通过文件注入虚拟机 • 只限于一些 Linux • 配置比较困难, 不推荐
FlatDHCP	<ul style="list-style-type: none"> • 设置相对简单 • 标准组网方式 • 所有操作系统都支持 	需要 DHCP 广播域, 如果网段与其他非 OpenStack 环境用同一网段并网络相同, 可能带来潜在问题
VlanManager	每个租户都局限于自己的 VLAN	<ul style="list-style-type: none"> • 比较复杂 • 需要独立的 DHCP 广播域 • 多个 VLAN 需要配置到单个网口, 网口需要设置成 trunk 模式 • 标准的 VLAN 数目只有 4096 个 • 交换机必须支持 802.1Q VLAN tagging
FlatDHCP Multi-Host HA	<ul style="list-style-type: none"> • 网络故障被隔离时, 只会影响到故障主机所运行的虚拟机 • DHCP 流量只局限于单个主机 • 网络流量分布到各个计算节点 	<ul style="list-style-type: none"> • 搭建起来更加复杂 • 默认地, 计算节点需要公有 IP • 如果要支持热迁移, 需要非常谨慎地配置网络

8.6.10 计算硬件需求

硬件数量的多少取决于部署需求。当规划一个 OpenStack 环境时, 需要考虑以下因素。

- **CPU**: 取决于有多少虚拟机需要部署在云平台环境和每个虚拟机需要的 CPU 数量。
- **内存**: 取决于每个虚拟机需要的内存数量和单台物理机的内存资源。
- **存储**: 取决于每个虚拟机需要的本地存储设备、远程挂载到虚拟机的存储卷和对象存储的大小。
- **网络**: 取决于 OpenStack 部署架构、每个虚拟机的网络带宽和存储网络等。

在计算 OpenStack 需要的资源数时, 也需要考虑将来扩展时的场景。下面例子以一个典型的需求为例来计算 OpenStack 云平台需要的资源数: 100 个虚拟机, 平均为 2GHz 的亚马逊 EC2 计算单元, 最高为 16GHz 的亚马逊 EC2 计算单元。

1. 计算 CPU

可以使用下面的公式来计算每个虚拟机需要的 CPU 核数：

最大 GHz / (每核的 GHz 数目 × 1.3 超线程系数)

例如：

$$16\text{GHz} / (2.4 \times 1.3) = 5.12$$

因此，每个虚拟机需要分配至少 5 个 CPU 核。

可以用下面的公式来计算需要的总共 CPU 核数：

(虚拟机个数 × 每个虚拟机的 GHz 数) / 每个核的 GHz 数

例如：

$$(100 \text{ 个虚拟机} \times \text{每个虚拟机 } 2\text{GHz}) / \text{每个核 } 2.4 \text{ GHz} = 84$$

因此，100 个虚拟机所需要的 CPU 核数为 84。

取决于选择的 CPU，可以计算需要的 CPU 插座数目：

CPU 核的总数 / 每个插座支持的核数

例如，如果使用的是 Intel E5 2650-70 8 核 CPU：

$$84 / 8 = 11$$

因此，需要 11 个 CPU 插座。

下面来根据 CPU 计算需要的服务器数量：

需要的插座总数 / 每个服务器插座数

将需要的插座总数按最近的偶数取整，得到 12 个插座数：

$$12 / 2 = 6$$

因此，上述需要的虚拟机总共需要 6 台双插座的服务器。

现在就可以得到每个服务器运行的虚拟机个数了：

虚拟机个数 / 服务器的个数

例如：

$$100 / 6 = 16.6$$

因此，每台服务器需要部署 17 个虚拟机。

在做上述计算的时候，我们有以下一些假定。

- 没有 CPU 超配。
- 如果使用了超线程，不要乘以 2，而是乘以 1.3。
- 选择的 CPU 要支持部署需要的技术功能。

2. 计算内存

继续使用前面的例子，我们要确定使用多少内存来支持每个服务器可以运行 17 个虚拟机。让我们假定每个虚拟机平均需要 4GB 的内存，但是动态分配的虚拟机可以达到 12GB。计算时，如果假定所有虚拟机都使用 12GB 的内存，则每个服务器就需要 204GB 的可用内存。

另外还要确保服务器本身有足够的内存来运行内核和操作系统，对于每个虚拟机还需要一定的内存来运行虚拟机管理程序的容器（这里不是每个虚拟机本身分配的内存，而是操作系统需要一定的内存来“管理”虚拟机）。

服务器本身需要内存来执行它自己的操作，如调度进程、动态分配资源、处理网络请求。所以服务器本身至少也需要 16GB 或以上的内存。如果考虑到我们使用的服务器的内存条分为 4GB、8GB、16GB 和 32GB，我们需要在每个服务器中配置 256GB 的内存。对于一般为平均 2CPU 插槽的服务器主板，那么我们就需要 16~24 个内存条。如果要安装 256GB 内存到服务器里，需要 16 个 16GB 的内存条才能使一个服务器运行 17 个虚拟机，且每个虚拟机可以动态分配到 12GB 的内存，并保证了所有操作系统运行的需要。

3. 计算存储

当考虑存储时，有以下几个选项。

- 非持久化存储（虚拟机的本地存储空间）。
- 持久化存储（可以挂载到虚拟机的远程存储卷）。
- 对象存储（例如图像或其他存储对象）。

对于由计算节点服务器提供的本地存储，在我们前面所述的 100 个虚拟机的例子中，有下列假设。

- 每个虚拟机有 50GB 的本地存储，总共需要 5TB 的本地存储（100 个虚拟机×每个虚拟机 50GB）。
- 每个虚拟机 500GB 的卷存储空间，总共 50TB 的持久化存储。

回到上述例子，我们需要算出每台服务器需要配置多少存储空间来供 17 个虚拟机运行于单台服务器中。如果我们假定每个虚拟机使用到 50GB 的存储空间，我们需要至少安装 750GB 的本地存储。绝大多数的服务器一般都有 4~32 个 2.5 英寸的驱动器插槽，或者 2~12 个 3.5 英寸的插槽，再考虑到服务器的其他方面，如 2U 或者 4U，另外还需要考虑应用场景如何影响到存储。在这个例子中，一个 3TB 的硬盘就足可以供 17 个虚拟机所使用，且每个虚拟机有 50GB 的存储空间。如果磁盘访问速度不是个关键因素，可以考虑配置 2 或 3 个 3TB 硬盘，并配置成 RAID 1 或者 RAID 5 组成的冗余。如果速度很关键，那么也可以为每个虚拟机配置一个硬盘，在这种情况下，可以考虑 3U，有 24 个插槽的服务器。

另外不要忘记了服务器本身也需要本地存储空间，应订购可以满足需求的机型。在上述例子中，假定速度也很关键，那么不妨订购有 18 个硬盘驱动器的服务器，例如 2.5 英寸、15 000 RPM 146 GB SAS 盘驱动器。

（1）吞吐

至于吞吐，则取决于哪种存储，通常，可以这样计算吞吐：

磁盘驱动器的 IOPS × 每个服务器的驱动数 / 每个服务器的虚拟机个数

但是真实的 IOPS 取决于选择的驱动技术，如：

- 3.5 英寸盘慢而且廉价（每个盘有 100 IOPS，每 2 个组成镜像（mirrored））
- 100 IOPS × 2 驱动器 / 每个服务器 17 个虚拟机 = 12 读 IOPS，6 写 IOPS
- 2.5 英寸 15K（200 IOPS，4 个 600GB 驱动器，RAID 10）
- 200 IOPS × 4 驱动器 / 每个服务器 17 个虚拟机 = 48 读 IOPS，24 写 IOPS
- SSD（40K IOPS，8 个 300GB 驱动器，RAID 10）
- 40K × 8 驱动器 / 每个服务器 17 个虚拟机 = 19K 读 IOPS，9.5K 写 IOPS

很明显，SSD 可以提供最好的性能，但价钱方面的区别也很明显。因此可接受的结果取决于预算和性能/冗余的平衡。另外很重要的一个区别是，在云平台里，冗余的规则和传统环境里不一样。传统环境里是对单个服务器提供冗余，而云平台是通过整个平台的服务器来提高冗余，换句话说，对于组件冗余的重要性从单纯的单个操作系统方面移到服务器冗余来了。更为重要的是有冗余的电源，可热插拔的 CPU、内存，而不是存储的冗余。例

如，如果有 18 个磁盘，其中 17 个直接分配给每个虚拟机，其中一个磁盘坏了，则只需要更换一个新的，把数据复制过去，其余的虚拟机依然正常运行，而不受影响。

(2) 远程存储

IOPS 对于规划持久性存储也很关键。例如，考虑上述虚拟机总共需要 50TB 的远程卷存储空间。

- 12 个驱动，使用 3TB 3.5 英寸盘驱动并且是镜像式的 (mirrored)。
 - 36TB 或者 2U，每 U 有 18TB 存储空间。
 - 需要 3 个框架 (frames) (50TB / 每个服务器 18TB)。
 - 12 个插座×每个驱动器 100 IOPS = 1200 读 IOPS，600 写 IOPS。
 - 3 个框架×每个框架 1200 IOPS / 100 个虚拟机 = 36 读 IOPS，18 写 IOPS 每个虚拟机。
- 24 个驱动，每个 1TB 7200 RPM 的 2.5 英寸盘。
 - 24 个 1TB，或者使用 2U 的 12TB 盘。
 - 5 个框架 (50TB / 每个服务器 12TB)。
 - 24 个驱动 × 每个驱动器 100 IOPS = 2400 读 IOPS，1200 写 IOPS。
 - 5 个框架× 2400 IOPS / 100 个虚拟机 = 120 读 IOPS，60 写 IOPS 每个框架。

(3) 对象存储

当谈到对象存储，我们会发现需要的空间远比我们想象得要多。例如，假设我们需要 50GB 的对象存储空间，对象存储默认使用 3 倍需要的空间用于复制，意味着我们需要的是 150TB。然而，为了配置成两个区域，至少需要 5 倍需要的空间，现在变成了 250TB。计算到这里还没有结束，因为我们不想用完所有空间，所以前述所说的空间只能达到总容量的 75%，现在变成了总共 333TB，是原来计划的 6.66 倍。当然这个数量非常大，因此可以从 4 倍的空间容量开始，然后当现有空间快满时再购买新的硬件。这样，开始时需要 200TB 的空间。如果使用的是 3TB 3.5 英寸盘，则需要每个服务器配置 12 个驱动框架，提供 36TB 空间，总共 6 台存储服务器，有 216TB 的空间。当然也可以使用一个 36 框架，只需要 2 台服务器就够了，但这样的话，如果节点失效，复制的代价太大，容量也无法再扩展，因此并不推荐。

4. 计算网络

部署 OpenStack 中，最复杂的一步是设计 OpenStack 环境的网络。一个 OpenStack 环境

可能涉及多个网络，包括公共、私有和互联网。一个典型的环境可能包括多个租户网络、存储网络、多个租户私有网络等。这些网络很多都是 VLAN 的，而且必须提前规划来避免配置错误。对于网络，还是使用上述的例子，且有下列假定。

- 每个虚拟机 100Mbps。
- 高可用的架构。
- 存储网络对延迟不敏感。

为了达到上述目标，每个服务器需要两个 1GB 连接（ $2 \times 1000 \text{ Mbps} / 17 \text{ 个虚拟机} = 118 \text{ Mbps}$ ）。使用两个 1GB 网络连接也可以提高可靠性，另外也可以使用两个 10GB 的连接来提高吞吐并且降低延迟，这样每个虚拟机有约 1Gbps。但这时会有其他因素需要考虑。

扩展性和超配

比较讽刺的是，1GB 网络通常扩展性比 10GB 以太网要强，至少在 100GB 交换机大规模使用之前是这样的。可以将 1GB 连接汇聚在一个 48 口交换机，这样就有 $48 \times 1 \text{ GB}$ 下行链路和 $4 \times 10 \text{ GB}$ 的上行链路。如果在 10GB 交换机做同样的事情，那么就需要 $48 \times 10 \text{ GB}$ 的下行链路和 $4 \times 100 \text{ B}$ 的上行链路，产生了超配。许多 OpenStack 环境的问题，只要仔细地规划，就可以避免许多后续问题。规划不足，例如当需要在机架间移动节点的时候，问题就来了。避免这类问题最好的方法是规划成单独的，既有计算节点资源池，又有存储节点资源池，这样的池是一个不会产生超配的二层域空间。

8.6.11 软件部分检查清单

在计划一个具体部署时，除了考虑硬件和 OpenStack 本身，还有其他一些软件/系统需要考虑。表 8-11 列举了一些方面。

表 8-11 部署组件选择清单

检 查 项	推荐或说明
操作系统	控制节点：四种企业级，包括 Red Hat、CentOS、Ubuntu、Scientific Linux 计算节点：取决于选择的 Hypervisor
OpenStack 安装包 仓库	
Hypervisor	KVM: OpenStack 使用最广泛的 Hypervisor，功能完备，无 license 费用和任何限制 Xen、VMware、Hyper-V……

续表

检 查 项	推荐或说明
Database	MySQL: 在 OpenStack 环境里使用最多, 测试最多, 文档非常丰富
消息中间件 (Message Queue)	RabbitMQ: 容易使用, 经过大量生产环境的测试。另外, RabbitMQ 本身已经支持集群, 且配置简单
网络	OpenStack Neutron 不支持 multi-host, 会引起单点问题。 VLAN: 配置技术成熟、安全、物理层透明。也可以使用 VXLAN
镜像存储后端	文件或 GlusterFS 或 Swift 或 Ceph
认证服务后端	SQL: 简单
块存储后端	LVM/iSCSI: 许多存储后端插件依赖于特定的厂家或社区去支持。但是 LVM 非常健壮和稳定 GlusterFS
热迁移后端	GlusterFS: 非常容易搭建, 可扩展, 如果环境增长, 可以继续增加存储节点, 而不用购买昂贵的存储阵列
对象存储	Swift: 已在非常多的生产环境中部署, 非常成熟, 可以跨数据中心做数据的复制

8.6.12 与企业现有系统集成

搭建生产环境时, 如果企业里已经有大量相关应用, 或 OpenStack 需要部署于一个已经使用的数据中心机房, 有很大可能需要考虑企业现有基础设施带来的影响, 或需要考虑如何和现有系统进行集成。

(1) 认证

包括与企业里的 LDAP 或 Windows AD 等的集成, 包括开发或增强功能, 与企业现有认证系统连接。此时, OpenStack 将把用户的认证转给企业现有认证系统去执行, 根据返回结果决定是否可以使用 OpenStack 云平台的服务。

(2) 监控

OpenStack 环境中同样有大量的主机、进程、CPU、内存、磁盘、网络等信息需要监控。很多企业数据中心已经在使用一些监控工具, 因此, 需要整合云平台的监控。

(3) 告警

目前,云平台此项功能很弱,或近乎不可用,可以用第三方工具来实现。如果云平台本身使用第三方工具来完成此功能,邮件服务的配置仍然需要,包括接收者、邮件服务器地址等。

(4) DNS

云平台必须依赖于 DNS 服务才可以保证各项服务正常工作及相互之间的访问。DNS 服务器可以搭建于 OpenStack 控制节点,但也可以直接使用企业现有的 DNS 服务功能。

(5) NTP

云平台必须依赖于 NTP 服务才可以保证各项服务正常工作及相互之间的访问。NTP 服务器可以搭建于 OpenStack 控制节点,但也可以直接使用企业现有的 NTP 服务功能。

(6) 网段

在部署小规模生产环境,而用户已经有相当规模的数据中心时,给定的 IP 段(包括管理网络、数据网络、硬件管理网络等)有可能是现有网络的一部分,而且物理交换机相通。此时可能会发生以下情况。

- 云平台的 DHCP 服务器会干扰企业现有系统,或现有系统会错误地被分配到云平台的 IP 地址。
- IP 配置错误有可能引起与现有系统之间的 IP 冲突。

8.6.13 云环境扩展需求

OpenStack 云平台搭建完毕,用户往往发现在功能设计和使用上,依然与企业需求有一定的差距,因此需要进行一些针对具体企业需求的定制和开发。

(1) 云主机需求

- 云主机生命周期管理。
- 云主机状态详情查看。
- 镜像快照管理。
- 网络与访问安全管理。
- 云主机计费管理。

- 云主机监控报警管理。
- 快照、磁盘管理、迁移。
- 控制版本更新。

(2) 优化和新功能开发

- 云主机服务质量保证。
- 镜像快照存储处理优化。
- 调整云主机规格优化。
- 用户自助服务和运维管理平台。
- 云主机监控报警功能。
- 云主机实例存储配额功能。
- 更加强大、更加用户友好的管理界面。
- 更快的快照。
- 一键部署。
- 云平台中的数据分析。

(3) 云平台物理资源共享带来挑战

云平台物理资源共享到来如下挑战。

- 云主机性能指标变模糊，无法精确定义。
- 云主机间以及云主机与宿主机竞争资源，使云主机性能不稳定。
- 抢占资源经常出现，部分用户优先抢占资源造成其他用户性能差、网络慢。
- OpenStack 本身的资源调度策略是比较初级的，可能会从比较远、比较忙的节点调度资源，导致网络慢。
- 业务平台直接调用 OpenStack 的 RESTful 接口，调用多了之后遇到压力，响应速度慢。

为了提供性能指标明确、性能稳定可靠的云主机，可采用如下方案。

- 制定方案时需考虑：
 - 明确定义云主机性能指标。
 - 控制云主机资源占用，避免云主机间相互影响。
 - 预留一定物理资源给宿主机，避免影响宿主机正常运行。
 - 明确当前实现的服务质量保证（QoS）。
 - 需要的计算（CPU）、网络带宽等资源。

某公司对 OpenStack 进行了一些定制，具体如下。

- 将基于 L3 的网络模型重构为 L2 的网络，以获取更好的性能和更低的成本。
- 使用 tarball 格式的镜像，可以更加快速地建立新的实例。
- 给实例添加了一些 QoS 的参数，更适合关键业务。
- 建立实例的回收站，可以恢复误删的镜像。
- 对 Keystone 进行修改，与 360 账户系统集成。
- 改良操作面板。

8.7 生产环境 问题和对策

8.7.1 计算资源隔离和流量控制

在 Havanna 里，已经可以在主机类型里配置流量控制策略了。

如果使用的是早期版本，基于 Libvirt 做 CPU 资源隔离本身是有很有效的，CPU 的繁忙很多时候都是 I/O 等待所引起的。Libvirt 对 I/O 隔离支持得并不好，所以可以引入 cgroups 来实现 I/O 的控制，给每一台虚拟机做基于 IOPS 的限制。还可以给每个虚拟机加入网络连接数的限制，则 CPU 的负载就降下来了。这种策略也可以用于在业务上进行高端与低端客户的区分对待，提供差异化服务。

8.7.2 调度策略

OpenStack 本身是高度可定制化的，只需要配置系统自带的或者编写一个调度策略，忽略默认的 nova.conf 中的配置即可。

8.7.3 负载均衡

可以把 OpenStack 服务组件按照 Zone 进行拆分，将负载分摊到多个集群上。

8.7.4 OpenStack 的实施

1. 实施计划

在动手搭建 OpenStack 云平台之前，制订一个工作计划可以让整个部署变得有规划和顺利。典型的实施计划如表 8-12 所示。

表 8-12 实施计划示例

日 期	工 作 内 容
01/01/14	硬件设备检查
	制定实施规划
	IPMI 初始化，检查固件版本并升级
	存储初始化
	配置光纤交换机，进行 Zone 的划分
	以太网交换机升级，VLAN 划分，vtag 划分
01/02/14	OpenStack 控制节点安装部署
	OpenStack 计算节点安装部署
.....
01/05/14	分布式文件系统部署
01/06/14	OpenStack 测试
01/07/14	制作 Linux 和 Windows 镜像
	测试镜像
01/08/14	高可用测试
01/09/14	文档编写（系统信息文档、设计文档、实施文档、使用文档、培训文档）
01/12/14	系统交付、客户培训、文档交付
01/13/14	签字交付

2. 整体规划

在实际开始部署前，最好要形成类似下面内容的文档，对每一块的部署细节做出规划，并和客户交流，保证在可用的硬件环境下，满足尽可能多的客户需求。表 8-13 和表 8-14 所示为典型的主机分配示例和网络规划示例。

表 8-13 主机分配示例

云名称	硬件	虚拟化	存储	云管理
xx 高校教学云平台	N × 440 为控制节点 M × 220 为计算节点	CentOS 6.4 KVM	本地 LVM; GlusterFS 共享文件系统	OpenStack Havanna

表 8-14 网络分配示例

VLAN 号	网络类型	网段
4093	管理网络	192.168.2.0/255.255.255.0

3. 设置正确的硬件和网络环境

尽管 OpenStack 允许在一个单一的扁平网络上部署一切，但从安全的角度来看这并不安全。取决于所使用的虚拟机管理程序（Hypervisor）以及虚拟网络接口是否安全，在某些情况下还会导致从虚拟机可以监听管理网络的数据报文。因此建议至少使用两个网络：一个用来管理流量，一个用来进行虚拟机之间的通信。这意味着所有的云计算节点都需要两个网卡运行实例和网络管理。这些应该部署在不同的 IP 范围中。计算节点和实例的网络也需要支持 VLAN 标记，因为这是在“项目”之间隔绝流量所使用的机制。

4. 部署所有的 OpenStack 组件

标准部署应包括一个云平台控制器和一系列计算节点。控制器运行消息服务器、数据库和其他的组件来管理虚拟机。但是也可以把控制器分解为几个部分来改善性能，可以把 MySQL 放在不同的物理服务器中。对于安全而言，最关键的是确保每一部分都安装在安全的主机上。

云平台通常只有两部分需要暴露给外面的网络世界（即使只是企业私有云）：API 服务器/Web 控制台和网络管理者。这些服务器需要特别注意安全，甚至可以使用第三方网络接口来隔离后端管理用户连接产生的流量。

如果只是按照默认安装说明来部署，可能这些服务并不是那样安全。下面是一些推荐的修改。

- MySQL 服务器使用指定的用户账户，而不是根用户也是 MySQL 的管理账户。即使使用基于证书的认证，这个账户和密码将会分布在每一个云平台节点上，因为所有计算节点都需要访问数据库服务器。
- 在 MySQL 配置文件中，限制可访问的服务器，将 OpenStack 用户账户授权为唯一

图 8-13 展示了 4 个节点的 Load 视图，并在一个视图中进行聚合展示。不同颜色标识不同节点的负载情况。

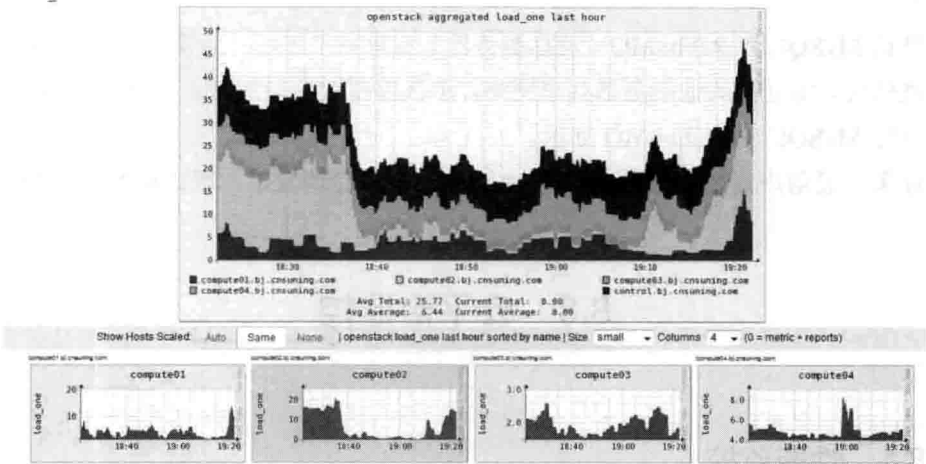


图 8-13 Ganglia 的聚合视图和节点视图

图 8-14 是 CPU 详细监控项目。

图 8-15 是一个二次开发的，可以用来监控节点运行的虚拟机的性能，本图展示的是监控一个虚拟机的视图。

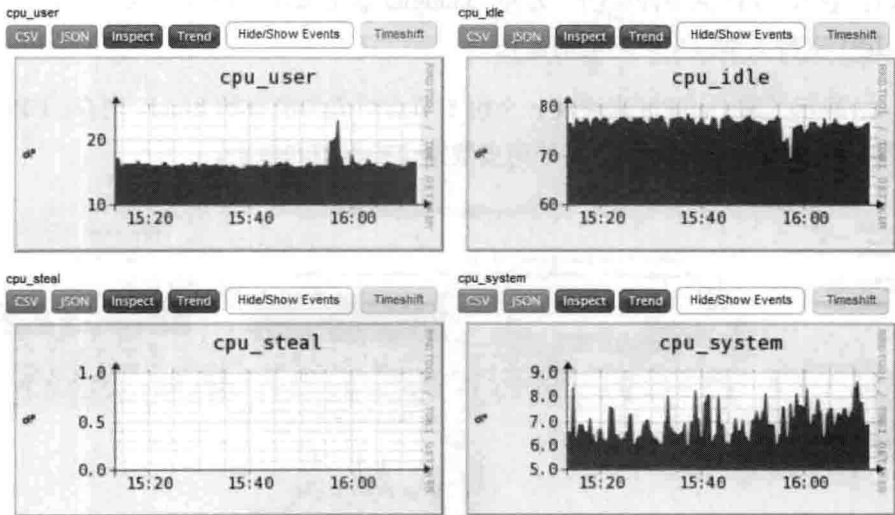


图 8-14 Ganglia 的 CPU 监控

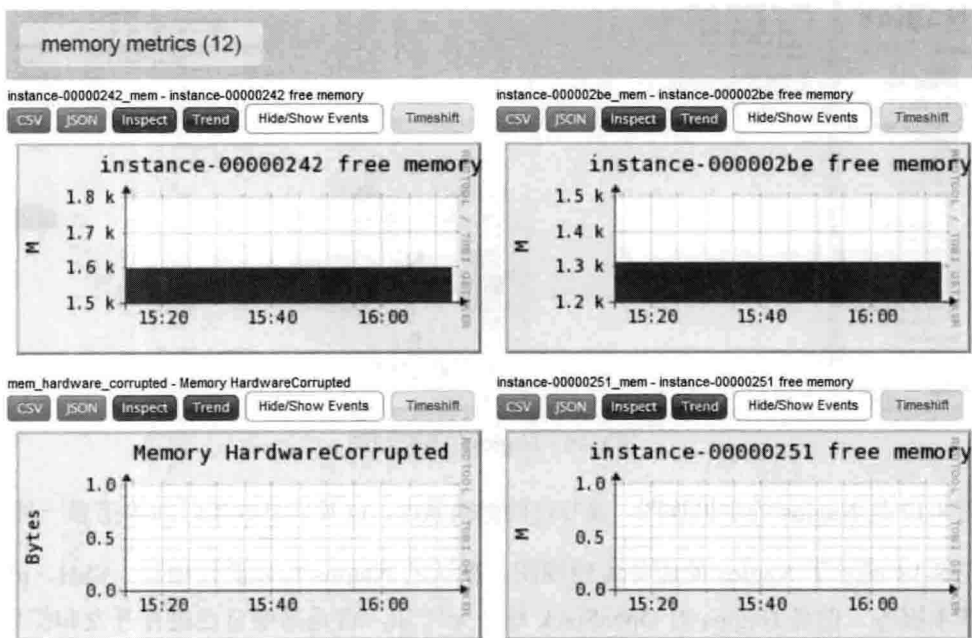


图 8-15 Ganglia 监控虚拟机内存

8.8.2 服务与资源监控

Nagios 是在 OpenStack 部署中广泛使用的，用于监控云平台各种服务的开源工具，另外，它的扩展性良好，很容易加入自行开发的监控项目，为用户提供了很大的便利性和灵活性。

图 8-16 展示了 Nagios 的连接视图，这里 5 个节点在同一个网络里。绿色代表该节点服务正常，而红色则说明该节点至少有一个以上的告警需要处理。

另外，Nagios 可以很容易地与邮件系统和短信平台集成，发送告警信息。

另外，本书展示的界面只是默认的，社区里有许多不同的展示界面，而且是免费的，可以根据自己的需要和偏好进行选择，提供不同的样式和数据展现形式。但数据本身则还是由 Nagios 服务提供。

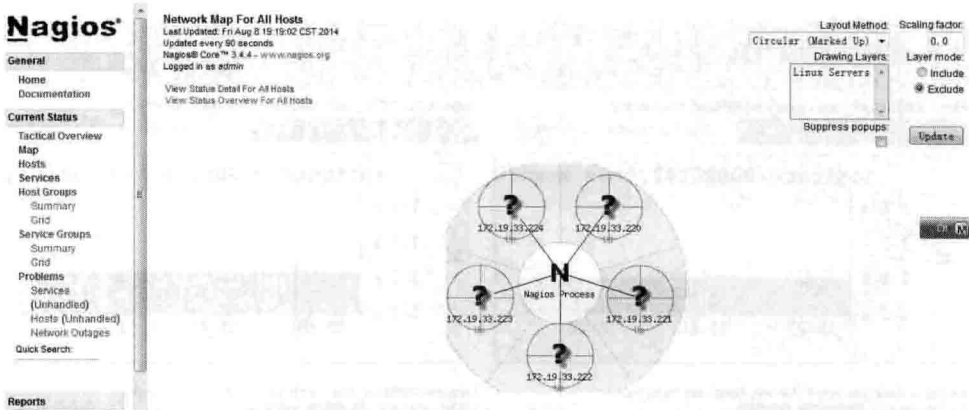


图 8-16 Nagios 的连接视图

图 8-17 是 Nagios 的主机视图，该节点的健康状况，以及信息收集时间等信息一目了然。

图 8-18 展示了 Nagios 的服务监控视图。默认的 Nagios 可以监控磁盘、SSH、ping 等系统基本服务，但是 Nagios 对 OpenStack 却一无所知，这是需要自己进行开发和扩展的，另外网上也有不少免费的监控脚本可以集成进来。

对于 OpenStack 服务的监控，最基本的可以从进程角度进行监控，如进程是否运行，响应速度如何等。更好的监控，可以结合不同服务的命令，通过查询来判断服务本身的质量如何。这需要初步了解各个服务的使用，以及用到租户、用户、密码等信息。但如果云平台修改密码，则这里也需要进行相应的修改。

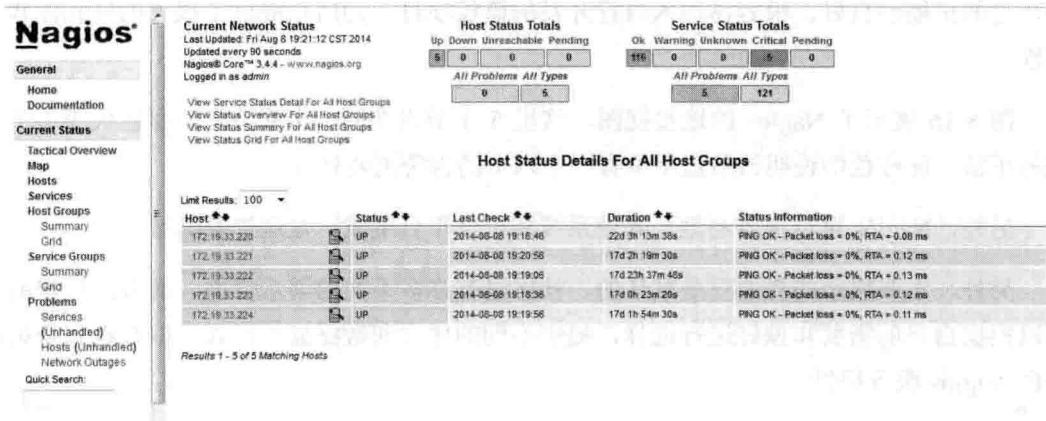


图 8-17 Nagios 的主机视图



图 8-18 Nagios 的服务监控视图

8.8.3 消息中间件监控

消息中间件在 OpenStack 云平台起着心脏的作用，运行不正常则整个云平台会运行不起来，因此监控消息中间件在 OpenStack 云平台至关重要。

图 8-19 展示了一款免费软件，可以用于消息中间件统计信息的展现和一些基本管理。

图 8-20 显示了消息中间件当前的各种连接，这些连接从不同节点和不同服务而来。



图 8-19 RabbitMQ 的 Overview 视图

RabbitMQ

Overview **Connections** Channels Exchanges Queues Admin

Connections

Filter:

Name	Protocol	Network				Overview		
		Client	From client	To client	Timeout	Channels	User name	State
172.19.33.220:32874	AMQP 0-8		3B/s (743.6MB total)	2B/s (11.2MB total)		1	admin	running
172.19.33.220:32919	AMQP 0-8		0B/s (737.6MB total)	0B/s (11.2MB total)		1	admin	running
172.19.33.220:32997	AMQP 0-8		0B/s (132.2MB total)	0B/s (12.1MB total)		1	admin	running
172.19.33.220:33015	AMQP 0-8		59B/s (737.6MB total)	5B/s (11.2MB total)		1	admin	running
172.19.33.220:33029	AMQP 0-8		0B/s (737.6MB total)	0B/s (10.9MB total)		1	admin	running
172.19.33.220:33309	AMQP 0-8		0B/s (7.6MB total)	0B/s (10.1MB total)		1	admin	running
172.19.33.220:35202	AMQP 0-8		0B/s (4.3MB total)	0B/s (10.0MB total)		1	admin	running
172.19.33.220:35207	AMQP 0-8		0B/s (736.6MB total)	0B/s (11.2MB total)		1	admin	running
172.19.33.220:36668	AMQP 0-8		0B/s (731.7MB total)	0B/s (9.4MB total)		1	admin	running

图 8-20 RabbitMQ 的连接视图

图 8-21 展示了消息中间件的通道视图。

RabbitMQ

Overview **Connections** **Channels** Exchanges Queues Admin

Channels

Filter:

Channel	User name	Mode (?)	Details				Message rates			
			Prefetch	Unacked	Unconfirmed	Status	publish	confirm	deliver / get	ack
172.19.33.220:32874 (1)	admin		0	0	0	Idle				
172.19.33.220:32919 (1)	admin		0	0	0	Idle				
172.19.33.220:32997 (1)	admin		0	0	0	Idle				
172.19.33.220:33015 (1)	admin		0	0	0	Idle				
172.19.33.220:33029 (1)	admin		0	0	0	Idle				
172.19.33.220:33309 (1)	admin		0	0	0	Idle			0.00/s	0.00/s
172.19.33.220:35202 (1)	admin		0	0	0	Idle				
172.19.33.220:35207 (1)	admin		0	0	0	Idle				
172.19.33.220:36668 (1)	admin		0	0	0	Idle			0.00/s	0.00/s
172.19.33.220:36673 (1)	admin		0	0	0	Idle				
172.19.33.220:36703 (1)	admin		0	0	0	Idle				
172.19.33.220:36704 (1)	admin		0	0	0	Idle				
172.19.33.220:41712 (1)	admin		0	0	0	Idle			0.00/s	0.00/s
172.19.33.220:42245 (1)	admin		0	0	0	Idle	0.00/s			

图 8-21 RabbitMQ 的通道视图

图 8-22 展示了消息中间件的交换（Exchange）视图。

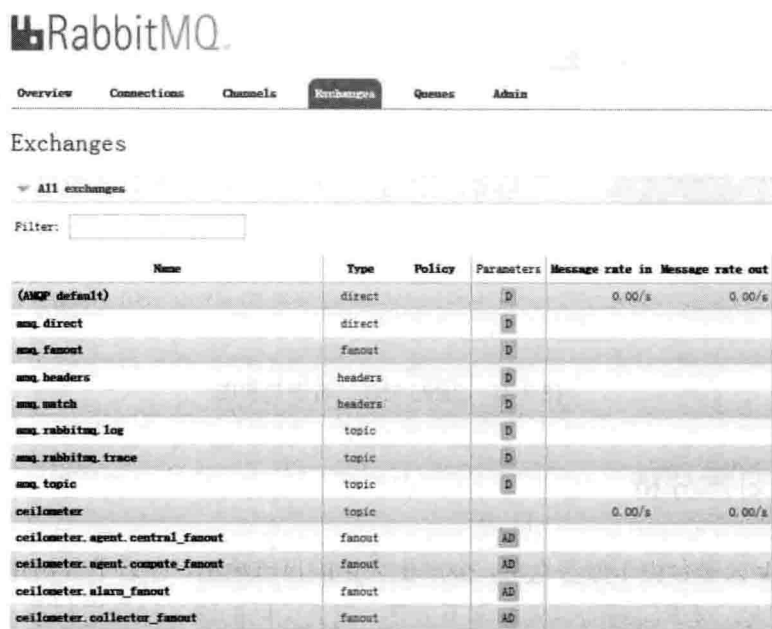


图 8-22 RabbitMQ 的交换视图

图 8-23 展示了消息中间件的队列视图，并包括了统计信息。

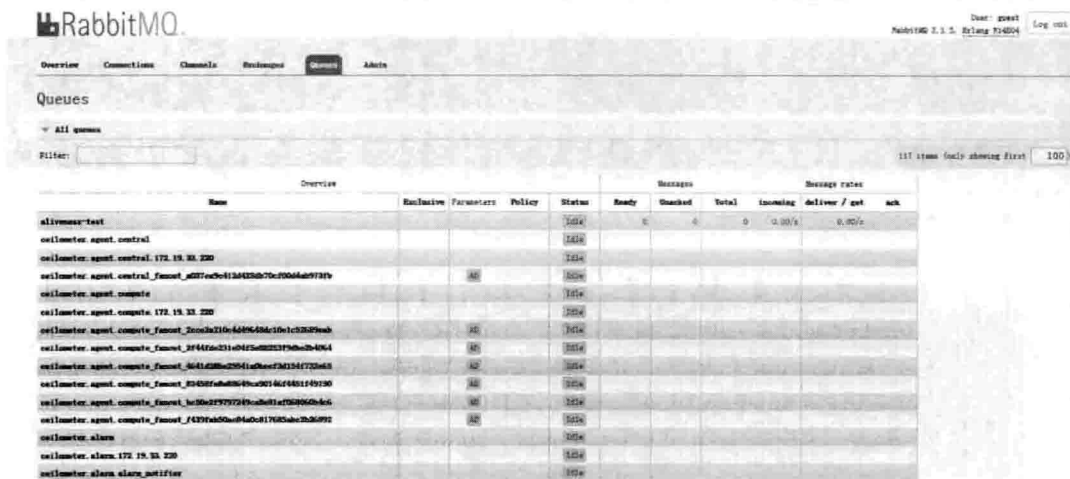


图 8-23 RabbitMQ 的队列视图

图 8-24 是管理员视图，可以修改密码、增加用户等。

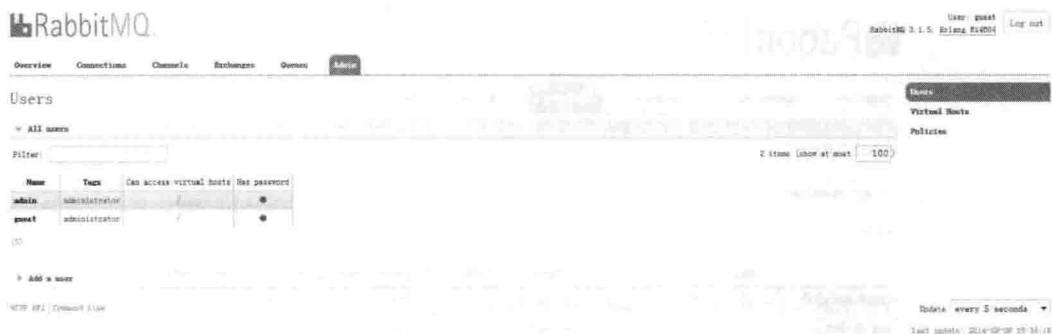


图 8-24 RabbitMQ 的管理员视图

8.8.4 日志分析

OpenStack 云平台由于服务众多，会产生大量的日志输出，在云平台初建，涉及 Debug 时，日志量巨大，对于问题的调试非常不方便，因为关联的服务往往运行于不同节点之上。

LogStash 在各个节点部署了代理程序，可以将日志输出汇聚到中心节点进行分析。中心节点本质上是搜索引擎和信息统计工具。

图 8-25 展示了 LogStash 的 Overview 视图，可以看到整个云平台日志产生的数量趋势。

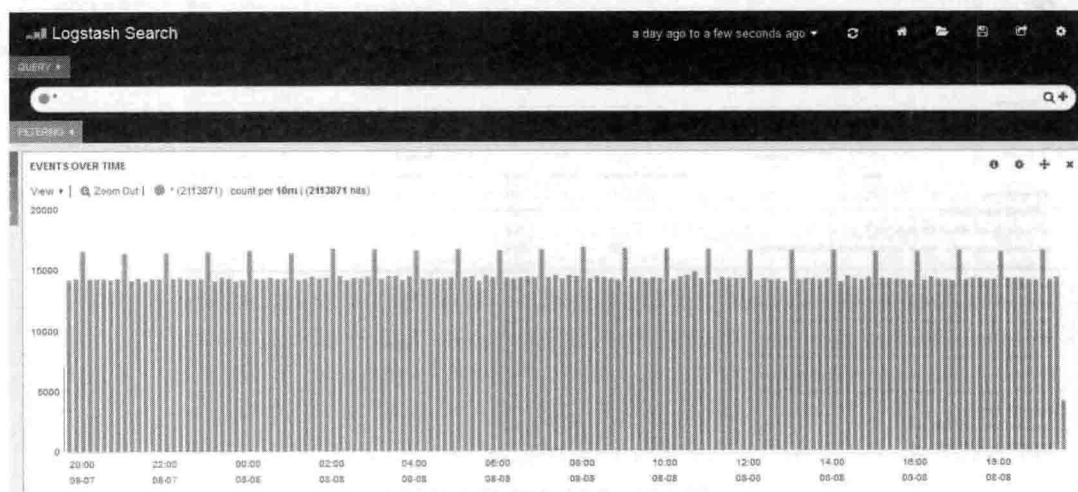


图 8-25 LogStash 的 Overview 视图

图 8-26 展示了 LogStash 收集的 OpenStack 云平台的日志信息汇总,并按时间顺序显示,并可以按不同的维度进行排序、关键词搜索。



图 8-26 LogStash 的消息视图

图 8-27 展示了过滤后的视图,这里我们只关心哪个节点在什么时候发生了什么事情。LogStash 可以很好地帮助我们实现这些。

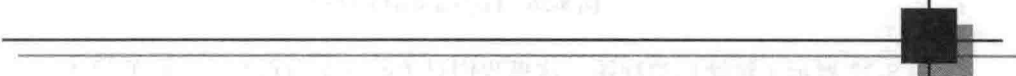


图 8-27 LogStash 的消息过滤视图

第 9 章



第三方工具搭建 OpenStack 运行环境



随着 OpenStack 的发展和演进,许多不同的部署工具相继开发出来。总体上,它们都是目前市场上领先的、也得到许多生产和实验环境验证的工具。简单的比较如表 9-1 所示。

表 9-1 免费部署工具对比

工 具	裸 机	自 动 化	编 排	注 释
Fuel (Mirantis)	Cobbler	Puppet	定制	<ul style="list-style-type: none"> • 典型的配置,定制化的 manifests; • 满足超过 80%用户需求的库; • GUI (FuelWeb)
Ubuntu Cloud (Canonical)	MaaS	Juju (bash, Python, Chef, Puppet)	Juju	<ul style="list-style-type: none"> • 初始的部署和更新; • 集成到 Landscape (Canonical 企业级系统管理平台); • GUI
Crowbar (Dell)	Crowbar	Chef	Crowbar	
Triple-O: OpenStack-on- OpenStack (HP)	Nova bare-metal	Heat	Heat	
RDO's PackStack (Red Hat)	目前假定 操作系统 已经安装	Puppet	可定制	由 Red Hat 提供的运行于 Red Hat & Fedora 系统上的可定制的 manifests 来配置 OpenStack
Chef for OpenStack (Opscode)	Razor	Chef	Knife OpenStack 插件和脚本	Opscode 试图合并来自于 Rackspace、AT&T、Dell、DreamHost、HP、HubSpot、Midokura、SUSE 的 OpenStack cookbooks
Alamo (Rackspace's Private Cloud)	CD 安装	Chef	OpenCenter	
IBM OpenStack Solution	Xcat, ISO	Chef	Shell 脚本	一键式安装;可组合的模块适应不同企业对于存储网络的不同需求,如千兆网、万兆网、存储设备、分布式存储、本地存储等

9.1 DevStack

DevStack 是由 Shell 脚本来控制搭建的完整的 OpenStack 开发环境，其网址为 <http://devstack.org/>。

目前 DevStack 支持 Ubuntu 12.04 (Precise)、Fedora 20 和 CentOS/RHEL 6.5。其中 Ubuntu 版本由于非常容易上手，并且文件较小，容易分享，得到了非常广泛的应用。

DevStack 提供了几种不同的安装选项，见表 9-2。

表 9-2 DevStack 部署选择

部署方式	说明
在虚拟机里运行 OpenStack	如果没有可用的硬件，可以在虚拟机里运行 OpenStack，虚拟机本身也在虚拟机里创建。因为使用的是 QEMU 仿真，虚拟机创建过程会比较慢
在真实的硬件里创建 All-in-One	在一个笔记本电脑或服务器里运行 OpenStack，体会真实的感觉
在真实的硬件里分布式部署 OpenStack 服务	搭建一个多节点的集群，使用 VLAN（当然可以是其他网络类型），来运行虚拟机和对虚拟机进行管理

DevStack 提供了搭建 OpenStack 环境的一步一步的文档，另外也有全部使用到的代码、配置文件、配置文件的例子和练习脚本等，是初始学习 OpenStack 非常好的教材。

然而需要提醒的是，一个教学用的 OpenStack 环境，并不能代替专业的针对生产系统的 OpenStack 环境，真实的生产环境不但要满足各种硬件、网络、存储的要求，对性能、数据的可靠性、安全性都有很多要求。

9.1.1 环境准备

如果使用的是物理硬件服务器，并没有特殊的要求，只需要支持 VT 的功能就可以，新的服务器一般都是支持的。对于比较老旧的机器，可以通过检测 CPU 来判断：

```
#egrep '(vmx|svm)' --color=always /proc/cpuinfo
```

如果有输出，则说明支持虚拟化技术；否则不支持。另外，使用 VirtualBox 一个常常犯的错误是本机的 BIOS 里 VT 没有打开。这只需要重启计算机，进入 BIOS，在 CPU 配置的部分打开即可，否则创建的虚拟机无法启动。

在这里我们使用 VirtualBox 创建了两个虚拟机来搭建一个多节点的 OpenStack 环境。其中一个节点将运行如 MySQL、RabbitMQ 以及其他一些控制服务，而另外一个节点将是计算节点。由于配置为 multi-host，所以一些 Nova 服务也在计算节点运行。这也是 OpenStack 设计的，即绝大多数服务是无状态的，而且可以运行多个进程，通过外部的负载均衡器或内部的消息总线来做负载均衡。

虚拟机只需要一块网卡即可。为了让两个虚拟机之间可以互通，在制作虚拟机时，选择了 Bridged Adapter，而不是默认的 NAT 方式，这样无论是这两个虚拟机之间，还是虚拟机与物理机之间网络都是相同的。另外，我们的虚拟机还可以访问外网，用来在安装过程中下载需要的软件包。

两个虚拟机是 Ubuntu 12.04 (Precise) 64 位的。在创建第一个虚拟机之后，用克隆就得到了两个同样的虚拟机。此时，我们需要确保两个虚拟机的主机名是不一样的。如一个虚拟机名字为 OpenStack，另外一个为 OpenStack-Compute。当然这个名字是任意的，只要满足主机名命名规范即可。

为了能够正确地安装所有的依赖，我们在这里安装了一个最小化的 Ubuntu 环境。干净的环境会减少后续许多麻烦。

9.1.2 安装

首先需要安装一些工具包供后续使用：

```
sudo apt-get install -y git
```

在本次安装中，使用的是 OpenStack 的 FlatDHCP 网络控制器，所以只需要一个网络。

在这里有如下网络规划。

- 控制节点和计算节点。

- IP: 9.181.89.155, 9.181.88.25。

- 网络掩码: 255.255.254.0。

- 网关: 9.181.88.1。

- 浮动 IP: 192.168.20.0/24。
 - 虚拟机使用网络 IP: 192.168.10.0/24。

如果只是一次性的尝试,则可以用系统 DHCP 分配的 IP 地址开始安装。如果环境需要反复使用,则需要将网络地址信息写到配置文件中。在 Ubuntu, 需要编辑/etc/network/interfaces 文件。对于 CentOS/RHEL 和 Fedora,则需要更新/etc/sysconfig/network-scripts/ifcfg-eth0 文件。

首先需要创建一个运行 OpenStack 的用户,这个用户不能是 root 用户,但要具有 root 的权限。在 Ubuntu 里,则属于一类具有 sudo 访问权限的用户。在本例中创建的用户名为“stack”,但也可以用任意其他合法名字替代。如果是安装多个节点,每个节点都要创建同样的用户。命令如下:

```
groupadd stack
useradd -g stack -s /bin/bash -d /opt/stack -m stack
```

在继续后续操作前,需要退出当前的登录,以 stack 用户重新登录进来,但由于此时并没有提供密码,系统不允许登录。所以最好此时就给新创建的用户提供密码。

上述操作需要在每个节点里执行。

用如下语句来赋予 sudo 特权给 stack 用户:

```
echo "stack ALL=(ALL) NOPASSWORD:ALL" >> /etc/sudoers
```

或者直接修改/etc/sudoers 文件,在文件最后加入上述语句。

接下来是设置 SSH。SSH 访问只是为了各个节点访问方便。除非用到了块迁移,OpenStack 本身目前并不适用 SSH 协议,而是通过 RESTful API 来访问各个服务。

默认安装的 Ubuntu 里,有 SSH 客户端使用的命令,但可能并没有 SSHD 相关的包安装。这里只需要执行下列语句即可:

```
sudo apt-get install openssh-server
```

安装后,SSHD 将直接运行。此时,可以用 ssh stack@IP 来访问另外一个节点。

可以使用 ssh-keygen 来创建可以分发的公私钥对。然后将生成的 id_rsa.pub 复制到需要 SSH 访问的目标主机,并复制 id_rsa.pub 里的内容到.ssh 目录里以 authorized_keys 命名的文件即可。

SSH 访问一般只需要从控制节点可以无密码访问计算节点即可。

现在开始下载最新版本的 DevStack。输入下列命令即开始下载：

```
git clone https://github.com/openstack-dev/devstack.git
```

下载后就在当前目录生成一个 devstack 的目录，里面包括了所有需要的脚本等。对于需要部署 OpenStack 的全部节点都需要完成这步。

接下来需要配置 local.conf，让 stack.sh 可以从配置文件里读取这些设置，而不是在命令行提示。该文件在 samples 目录里有一份，将该文件复制到当前目录进行编辑。

控制节点：

```
HOST_IP=9.181.89.155
ADMIN_PASSWORD=supersecret
MYSQL_PASSWORD=supersecret
RABBIT_PASSWORD=supersecret
SERVICE_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=devstacktestforbook
FLAT_INTERFACE=eth0
FIXED_RANGE=192.168.10.0/24
FIXED_NETWORK_SIZE=4096
FLOATING_RANGE=192.168.20.0/24
MULTI_HOST=1
```

计算节点：

```
HOST_IP=9.181.88.35
ADMIN_PASSWORD=supersecret
MYSQL_PASSWORD=supersecret
RABBIT_PASSWORD=supersecret
SERVICE_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=devstacktestforbook
FLAT_INTERFACE=eth0
FIXED_RANGE=192.168.10.0/24
FIXED_NETWORK_SIZE=4096
FLOATING_RANGE=192.168.20.0/24
MULTI_HOST=1
```

```
DATABASE_TYPE=mysql
SERVICE_HOST=9.181.89.155
MYSQL_HOST=9.181.89.155
RABBIT_HOST=9.181.89.155
GLANCE_HOSTPORT=9.181.89.155:9292
ENABLED_SERVICES=n-cpu,n-net,n-api,c-sch,c-api,c-vol
```

可以看出,控制节点与计算节点基本配置信息都一样,这些信息也是 OpenStack 配置文件必须根据每个环境需要修改而适应当前环境的。因为两个节点是独立配置的,因此,在配置计算节点服务时,我们还需要一些额外的信息,我们需要告诉这个计算节点,主控节点的 IP 地址、数据库、消息中间件、镜像服务的 IP 地址和端口等。在一个简单的环境里,这些服务中只有一个服务进程运行。除此之外,我们还需要告诉这个计算节点,哪些 Nova 和 Cinder 的服务可以运行。这些服务在一个 OpenStack 的部署环境里是可以运行多个服务实例的。

然后,就开始运行 `stack.sh`, 让它来配置我们的 OpenStack 环境,启动各种服务了。

首先在运行许多控制服务的主机里运行如下语句:

```
cd devstack; ./stack.sh
```

如果在准备 `local.conf` 文件时遗漏某些密码或者“service token”,会提示用户输入。其他信息都可以从运行环境中采集到。执行这一步会花费 20~40 分钟的时间,因为除了安装和配置,在这个过程中,它还要从网上下载许多依赖和需要的包,具体如下。

- OpenStack 运行所需要的系统软件,包含一些 Python 的组件、MYSQL、RabbitMq-server 等。
- 下载 OpenStack 组件,包括 Nova、Keystone、Glance、Horizon 等。
- 下载并安装 OpenStack 源码所依赖的 Python 库和框架。
- 安装 OpenStack 各组件。

当看到下列输出时,说明安装已经完成:

```
2014-02-21 21:11:24 + set +o xtrace
2014-02-21 21:11:24 stack.sh completed in 1307 seconds.
client command line is in exercise.sh
The default users are: admin and demo
The password: Passw0rd
This is your host ip: 9.181.89.155
```

按 Enter 键,即可看到 Shell 的提示符。

在控制节点执行完成之后,就可以通过 `http://9.181.89.135/` 来访问 Dashboard, 用户名为 admin, 密码为 supersecret, 看安装是否正常, 服务是否都能运行起来。

此时,从 Dashboard 界面里的 Hypervisors 里,可以看到已经有一个计算服务正在运行。单击各个链接,都可以访问,没有任何报错。

Dashboard 同时给项目用户提供了一个 portal, 前述步骤运行完毕, 也有一个默认的 demo 用户被创建。此时在同一个浏览器里, 新开一个 tab, 输入 `http://localhost/`, 然后输入 demo, 密码同样为 `supersecret`, 就可以看到项目的界面。

继续部署计算节点, 在计算节点里运行:

```
cd devstack; ./stack.sh
```

这一步同样会花费 20~40 分钟的时间, 因为除了安装和配置, 在这个过程中, 它还要从网上下载许多依赖和需要的包。

在这一步完成之后, 访问 Dashboard, 可以看到 Hypervisors 里已经多了几个虚拟机管理器, 每个代表了一个计算节点。

DevStack 安装后, 会创建两个用户: `admin` 和 `demo`, 并且同时创建两个租户 (项目) `admin` 和 `demo`。`admin` 是一个具有全部管理权限的用户, 同时属于租户 `admin` 和 `demo`, 而 `demo` 是一个普通用户且只属于租户 `demo`。

下面是 DevStack 提供的其他一些脚本。

- `unstack.sh`: 停止运行的 OpenStack 服务。如果提供了参数 `-all`, 则还会停止 MySQL 和 RabbitMQ 服务。
- `clean.sh`: 除了执行 `unstack.sh` 之外, 还会试图将环境恢复到初始状态, 包括清除创建的虚拟机镜像、`/etc` 目录下的配置文件, 清除数据库、消息、各个组件/服务的中间数据等。
- `rejoin-stack.sh`: 这个脚本在上次执行 `stack.sh` 的基础上, 试图将 OpenStack 恢复到正常运行状态。

在默认安装后, 下面是一些有用的目录信息。

- 可执行文件: `/usr/local/bin`。
- 源代码: `/opt/stack`。
- 日志: `/opt/stack/logs`。

`stack.sh` 会生成一个记录环境变量的 `.stackenv` 的文件, 记录了创建环境的一些信息如下:

```
# 2014-02-21-210858
BASE_SQL_CONN=mysql://root:supersecret@127.0.0.1
ENABLED_SERVICES=g-api,g-reg,key,n-api,n-crt,n-obj,n-cpu,n-net,n-cond,n-sch,
```

```
n-novnc,n-xvnc,n-cauth,c-sch,c-api,c-vol,h-eng,h-api,h-api-cfn,h-api-cw,hori
zon,rabbit,tempest,mysql
HOST_IP=9.181.89.155
LOGFILE=/opt/stack/logs/stack.sh.log.2014-02-21-204938
SERVICE_HOST=9.181.89.155
SERVICE_PROTOCOL=http
STACK_USER=stack
TLS_IP=
KEYSTONE_AUTH_PROTOCOL=http
OS_CACERT=
```

stack.sh 安装后, 还有一个用于生成本机配置的文件 openrc。如果此时要执行一些 OpenStack 命令, 而每个命令都需要带有一些配置信息, 如租户、用户、密码等。通过执行下列命令, 则这些配置信息就放到运行环境中, 不再需要对每个命令分别提供:

```
.openrc admin admin
```

openrc 会调用 stackenv 来获取环境信息。然后就可以用很简单的命令格式执行一些命令了。下面是一些 openrc 设置的信息:

```
# The introduction of Keystone to the OpenStack ecosystem has standardized the
# term **tenant** as the entity that owns resources. In some places references
# still exist to the original Nova term **project** for this use. Also,
# **tenant_name** is preferred to **tenant_id**.
export OS_TENANT_NAME=${OS_TENANT_NAME:-demo}

# In addition to the owning entity (tenant), nova stores the entity performing
# the action as the **user**.
export OS_USERNAME=${OS_USERNAME:-demo}

# With Keystone you pass the keystone password instead of an api key.
# Recent versions of novaclient use OS_PASSWORD instead of NOVA_API_KEYS
# or NOVA_PASSWORD.
export OS_PASSWORD=${ADMIN_PASSWORD:-secrete}
```

后面就可以执行一些 OpenStack 命令了:

```
# 列出创建的租户
keystone tenant-list
```

```
# 列出创建的用户
keystone user-list
```

```
# 增加用户和租户
NAME=bob
PASSWORD=BigSecrete
TENANT=$NAME
```

```
keystone tenant-create --name=$NAME
keystone user-create --name=$NAME --pass=$PASSWORD
keystone user-role-add --user-id=<bob-user-id> --tenant-id=<bob-tenant-id>
--role-id=<member-role-id>
# member-role-id comes from the existing member role created by stack.sh
# keystone role-list
```

一些可能用到的命令如下。

(1) 重置网桥。

下面的命令从网桥 br100 中删除一个接口 eth0.926。926 是一个 VLAN 号，创建于原来的网口 eth0。然后停止网桥并删除网桥。

同样可以用 `brctl addbr`、`brctl addif` 等命令来创建网桥和增加网桥上的接口。

```
sudo brctl delif br100 eth0.926
sudo ip link set dev br100 down
sudo brctl delbr br100
```

(2) 设置 MySQL 密码。

如果忘记设置 MySQL 管理员密码，则可以用下面的命令来设置：

```
mysqladmin -u root -pnova password 'supersecret'
```

有了上述环境之后，就可以尝试 OpenStack 的各种操作和功能，如开发修改源代码、启停服务等。

9.2 IBM OpenStack Solution for System X

无缝地配置、部署和使用各种异构的硬件设备，并使用开源技术来搭建企业级云平台是一件非常有挑战性的工作，合适的工具和组件可以帮助云平台更快捷搭建。下面是一些需要我们去解决的问题。

- 将硬件加电，联网，然后快速地安装和配置各种软件和应用。
- 可以远程安装操作系统，而不需要等待数据中心人员走到机房，找到正确机架上的机器，配置服务器。
- 当硬件发生故障时，能快速地恢复各种服务。
- 只需要一次就可以安装和配置各种操作系统和应用。

IBM OpenStack Solution for System X 是由 IBM 研发的, 基于 OpenStack 社区版和内部改进版本快速搭建企业级 IaaS 和部分 PaaS 的工具和一组软件。该项目从 OpenStack Folsom 版本开始, 先后支持 Grizzly、Havana 版本, 同时支持 CentOS 6.4 与 Red Hat 6.4 两种操作系统版本供用户选择。基于 6.5 的版本正在调试中。它不但很好地解决了各种部署拓扑的安装问题, 而且自带有许多 OpenStack 社区本身不具有的运维管理方面的工具。它可以帮助客户在几个小时内快速搭建企业私有云, 融合了领先的 OpenStack 技术, 是开源工具和许多硬件管理实施方面的最佳实践。

该工具在短短几个月中已经有许多家企业的实施案例, 行业涵盖保险、网游、制造业、电商、运营商等企业、高校、政府机关等。部署的节点从几个到几十个不等, 存储从本地存储, 到分布式存储、存储设备和对象存储等。其中高校的方案有效地解决了教学与科研的无缝切换, 一键式云平台恢复和集成的镜像制作平台与云平台对接, 得到客户的极大好评。

该项目参加了 2013 年 11 月在香港举办的全球 OpenStack 大会, 并作为演示项目。它还入围 2013 年 IBM 大中华区创新大赛, 在上百个竞赛项目中荣获最佳 12 项目之一。

9.2.1 OpenStack Solution 客户服务生命周期

“IBM OpenStack Solution”并不依赖于具体厂商的硬件, 能弹性而快速地适应不同厂商 X86 硬件, 具有完整的生命周期概念。因为云平台部署只是一个开始, 对于很多企业、学校、机关, 云平台搭建的目的是每天业务系统的使用、管理和维护。除了专业的技术和平台, 还需要有专业的支持和咨询, 这个完整的生命周期管理与支持如图 9-1 所示。

IBM OpenStack Solution 客户服务生命周期

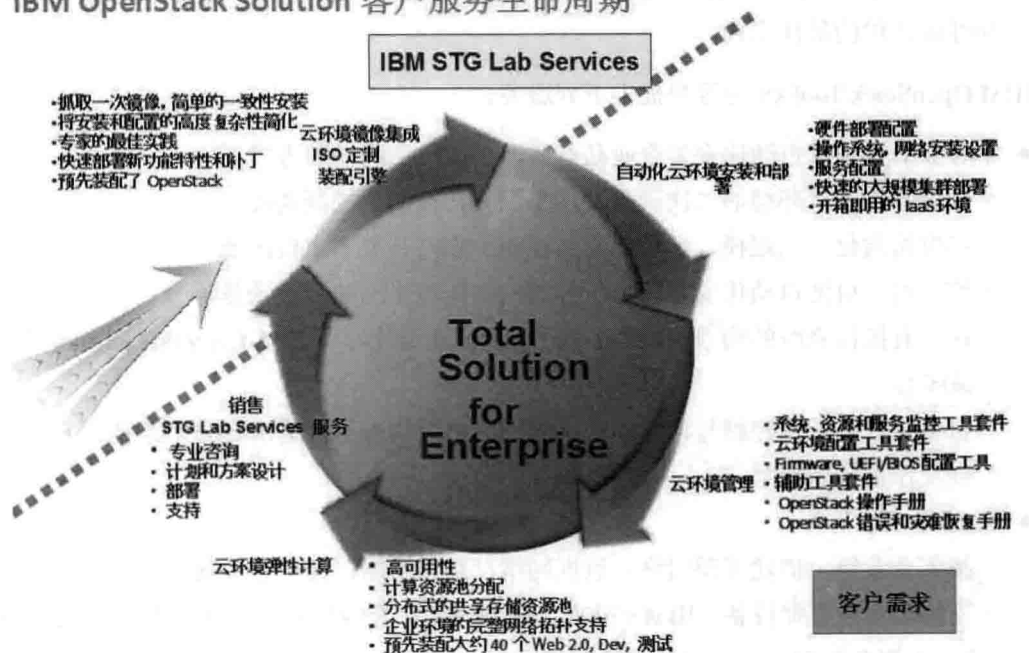


图 9-1 OpenStack 服务生命周期模型

9.2.2 OpenStack Solution 的功能特点与优势

IBM OpenStack Solution 具有如下一些优点。

- 开放。
 - 基于开源而领先的 OpenStack 技术。
 - 开放的 API, 包括各种插件、应用、工具、合作伙伴等组成的生态系统。
- 快速实现价值。
 - 在数小时内, 而不是传统的几周或几个月搭建完毕云平台。
 - 一键式全自动化、大规模的安装与配置。
 - 快速搭建运行用户业务系统。
- 降低维护代价。
 - 系统资源和服务的监控。
 - 控制服务和计算服务的高可用性。
 - 用户操作手册与错误调试手册。

- 系统管理最佳实践：集成了硬件管理、系统、Libvirt、KVM 等方面系统实施稳定性和性能调优的最佳实践。

IBM OpenStack Toolkit 为客户提供下列服务。

- 小时级的大规模自动化安装企业私有云，多种灵活的安装方式。
 - 部署网络主机环境的二次硬件发现与配置，灵活的控制功能。
 - 高度组装化、大规模、自动化安装控制节点与计算节点资源池。
 - 跨子网主机池自动化安装，用于控制网络跨越子网的企业场景。
 - 在已有操作系统的物理主机池安装 OpenStack 服务（只包括 CentOS/Red Hat 操作系统）。
 - 自动化安装配置控制与计算的高可用性，以及物理组件的绑定与多路径。
 - 纯操作系统的安装。
- 企业版功能。
 - 按照企业级，搭建管理网络、数据网络、存储网络、硬件管理网络等。
 - 支持企业级存储设备（IBM v7000、v3700 等），开源与企业版分布式/并行文件系统（GPFS & Gluster）；对象存储（Swift）。
 - 支持虚拟机创建时的镜像分发技术，避免镜像服务成为系统的一个瓶颈，可以容易地处理大规模虚拟机创建时的镜像分发风暴，保持系统的持续可用。
 - 热迁移，控制服务与虚拟机的高可用性。
 - 硬件层面的高可靠性，如网卡绑定、存储的多路径等，为应用提供一个非常可靠的运行环境。
 - 开源软件与 OpenStack 的深度集成，物理与虚拟资源的监控；多种监控供选择，包括 Ganglia、Nagios、Zabbix。
 - 自主研发运维管理工具，处理新节点加入/退出、存储扩容、密码管理、日志收集、主机管理与信息收集等常用业务场景。
 - 硬件管理工具的深度集成。
 - 多 Hypervisor 并存（KVM & PowerVC、Hyper-V）；多种网络组件的支持，包括 Linux Bridge、OpenvSwitch、扁平网络、VLAN 网络、VXLAN 等。
 - 安全，多种网络隔离，系统访问控制，企业级认证系统集成（LDAP）。
 - 轻量级的 VDI。
- 一键式的云平台恢复。在高校和企业级教学和研究环境中，用户往往需要一种可以快速恢复到以前某个状态的环境的能力，类似于虚拟机的快照。IBM OpenStack

Solution 可以快速地备份数据，并清理计算节点的残余数据，提供一个干净和确定的状态。通过一键式命令，将使云平台恢复到初始某个状态，包括其中已经注册的用户信息、网络拓扑信息、镜像信息等。

- 一键式的系统重置：在系统运行中，由于某些服务节点的断电、断网等原因，而且 OpenStack 是一个充分的分布式环境，在某些情况下，会引起系统关键服务的异常。IBM OpenStack Solution 提供一键式的服务重置，让系统快速恢复服务能力。
- 集成的镜像制作平台：虚拟机镜像制作对于云平台的使用意义重大，客户可以通过制作各种应用的虚拟机镜像来掌握和充分利用云平台，这对于 OpenStack 在企业的推广具有重要意义。IBM OpenStack Solution 提供高度与 OpenStack 兼容、最小化文件共享与传输、客户操作简单和简便的镜像制作平台，并与 OpenStack 云平台集成，保证客户对云平台使用的最大化，显著提升云平台对客户企业环境的存在和使用价值。

9.2.3 云平台内部的修补、优化与定制

OpenStack 是非常庞大而复杂的，集成了来自计算、存储和网络等各方面的技术并且还在不断演进。来自于 OpenStack 社区的 OpenStack 代码是一个初步能用的框架，在实际的使用中，存在许许多多的问题，特别是来自于系统和实际硬件环境的问题特别突出。未经实际验证的部署，只是一个“玩具”，只具有学习和研究的用途而已。

IBM OpenStack Solution 已经积累了许多来自客户现场第一手的部署、实施、运维经验和问题反馈，同时借助于内部来自各个领域技术专家的知识 and 经验，对云平台内部做了许多研究、调优和定制，保证推出的方案以稳定、可靠、性能良好、持续运行和专业的支持而得到客户的称赞。

这些优化包括：

- 跨厂商硬件兼容性：实际的客户环境是多厂商、多品牌的混合。产品先后在来自不同厂商、不同时期购买的硬件环境中部署和运行，取得了许多第一手经验，并在产品中进行了增强。
- 内核模块：OpenStack 的一些功能需要专门的内核模块，产品根据需要会加载相应的内核模块到内核，保证系统以可靠稳定的方式持续工作。
- Libvirt/KVM：系统专门对 Libvirt/KVM 进行了优化，保证客户得到可靠的功能和性能。
- 硬件发现：客户实际现场的硬件环境非常复杂，而产品集成了具有 10 余年 HPC 部

署经验的硬件发现工具，可以处理非常复杂的硬件发现场景，并有丰富的工具集可以进行操作。

- **主机名管理与 IP 地址分配：**产品的 IP 地址管理具有相当的智能化，降低了用户管理大型资源池的复杂性，同时又有足够的灵活性来进行定制。
- **资源池的管理/扩展：**产品充分考虑了在实际客户环境中，用户常常需要对资源池进行扩展、维护和替换的场景，因而提供了相应的设计和方案来应对各种针对计算、存储和网络资源的管理与维护需求。
- **操作系统：**现代操作系统是一个非常复杂的产品，可以满足用户各种企业级需求。IBM OpenStack Solution 虽然目前只支持 CentOS/Red Hat 操作系统，但针对云平台和虚拟化的特点，根据对操作系统多年实施的最佳实践，对底层的操作系统进行了相应的定制与调优，以发挥其最佳的功能与性能。
- **系统稳定性：**产品针对数据中心的断电、断网等各种意外和突发情况，在内部增加了相应的处理，可以非常容易地对故障进行隔离，并在故障恢复后，系统可以稳定而快速地恢复各种服务。
- **物理空间规划、挂载：**存储空间是企业级市场变化最多的方面。产品可以很容易地集成多种异构存储。在第一次部署开箱即用时刻，其存储规划依然可以满足一个中小型企业对镜像、虚拟机非持久化和持久化存储的需求。例如，系统初始部署后，用户可以完全放心地管理 50 个中等规模虚拟机镜像，或运行 100 个 small 主机类型的虚拟机，而无须担心存储空间崩溃的问题。
- **OpenStack 配置参数精选：**根据对 OpenStack 的深入了解，系统精选和调优了 Nova、Cinder、Glance、Neutron 的许多参数，而且对未公开的一些参数也进行了调优，保证系统即使在单控制节点部署情况下，也不会因为默认参数不合理而产生崩溃；另外，许多高级功能也需要参数的控制和调优，否则或者一些高级功能不支持，或者在并发或压力过高的时候会“卡”住。系统在这些方面做了精心的配置和调优，使所有发布功能可以放心而流畅地管理 200~500 节点的规模，一次性部署数百个虚拟机规模依然运行流畅。
- **MySQL 调优：**默认安装的 MySQL 是非常基本的。产品对 MySQL 做了许多调优，可以支持大数据量和并发请求。
- **消息中间件调优。**
- **社区 bug 同步与 patch：**产品在后台搭建了各种自动化引擎，包括持续集成引擎、自动化测试引擎，可以源源不断地打开/关闭与特定版本的代码同步，并持续集成自主开发的产品，然后自动装载并测试。另外，产品还不断邀请并不从事 OpenStack

研发的各种技术专家试用，以收集各种反馈。

- **安全与可靠性：**系统很好地处理了防火墙与端口屏蔽，在部署时强调物理与软件定义的安全并用的原则，并提供了多种选项来进行资源调度的隔离。
- **可靠性：**系统提供了物理层的许多可靠性措施，如 Raid、MAC 绑定、存储的多路径等，另外对断电、断网等各种场景，都提供了自动和人工恢复的手段和工具，可以在异常发生后最短时间恢复各种服务；另外，针对分布式系统的脑裂，从系统和人工途径都提供了多种手段用以检测和修复；对虚拟机提供了多种途径来自动和人工调度，如虚拟机迁移、Host Evacuate 等。
- **管理与运维：**IBM OpenStack Solution 内部开放了控制节点与计算节点的管理通道，物理机与虚拟机通道，并优化了 Windows 虚拟机的部署与管理；用户既可以通过 IP 网络访问虚拟机，也可以通过 VNC 来访问，另外对于 Windows 环境，还可以通过远程桌面来访问。多种技术的支持，提供了管理和使用虚拟机的多种方便的途径，带给客户许多便利。

9.2.4 模块化和自动化的云平台搭建技术

为了适应不同的企业环境和部署环境，IBM OpenStack Jumpstart Toolkit 采用了模块化的方式来搭建企业私有云平台。

- 一键式快速、大规模安装部署企业私有云，搭建基本的控制、计算、存储池（100 台物理机大约需要 2 小时）。
- 可选模块。
 - 共享文件系统（GPFS、GlusterFS）。
 - 存储设备管理。
 - 对象存储搭建。
 - HA 部署。
 - 大数据模块。
 - LDAP 搭建模块。
 - VDI 模块。
 - 镜像制作平台。

通过这些叠加式的组装，基本可以满足大部分企业私有云的搭建，满足基本的使用要求。

9.2.5 控制节点的设计

如图 9-2 所示, IBM OpenStack Solution 充分利用一个节点, 将控制服务和计算服务集成在一起, 同时提供了很多灵活性。同样, 许多试验或者生产系统部署里, 控制服务都运行于虚拟机里。这样带来许多好处, 如下。

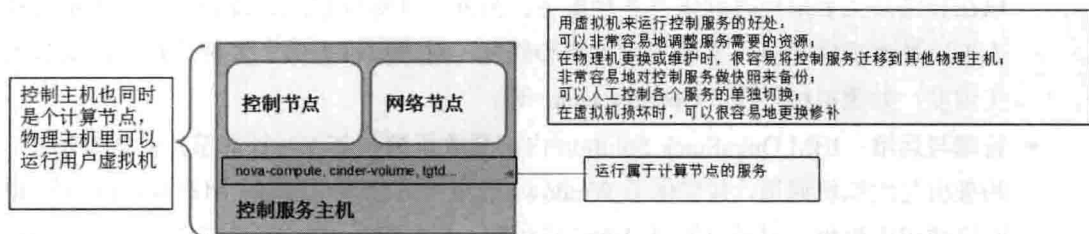


图 9-2 控制节点的设计

- 可以非常容易地调整服务需要的资源。
- 在物理机更换或维护时，可以很容易地将控制服务迁移到其他物理主机。
- 可以非常容易地对控制服务做快照来备份。
- 可以人工控制各个服务的单独切换。
- 在虚拟机损坏时，可以很容易地更换修补。

9.2.6 控制服务部署场景

IBM OpenStack Toolkit 的控制服务分开运行于两个虚拟机里，两个虚拟机完全是独立的。在这种设计中，如果客户部署的计算节点数比较少时，运行控制服务的主机其实还有很多空余的计算资源，则可以在这个主机里部署虚拟机，达到最大化利用资源、最大化客户投资的目的。

另外，在支持控制服务的 Active-Standby 时，又带来其他一些好处：两个控制服务节点可以独立切换，包括自动因为故障发生的切换，以及人工强制的切换。这种设计，可以同时支持单控制节点的部署，也可以支持“活跃-待机”的高可用性控制服务的部署。虽然增加了设计和实现的复杂度，但却带来很大的灵活性。

(1) 小规模部署场景

在小规模部署环境里，所有的控制服务虽然运行于两个虚拟机里，但是在同一物理主

机里。两个虚拟机分别同另外一个物理机里运行同样服务的虚拟机组成“活跃-待机”对，如图 9-17 所示。因为待机所在主机此时只需要很少资源，因此大部分资源可以运行计算服务。但同时对控制服务主机提供了保护。当故障（断电、断网、操作系统故障等）发生或人工强制切换时，服务将切换到运行待机服务的主机上。

两台物理主机都运行了计算节点的服务，因此都可以和普通计算节点一样运行虚拟机，最大化地利用了系统资源。图 9-3 左部展示了小规模部署场景的 HA 切换。

(2) 中等规模部署场景

在中等规模的云部署中，由于需要较多的资源用于运行控制服务，所以控制服务将分别运行于两个物理主机里。对待“待机”的虚拟机节点提供了故障切换的保护。如果这两台物理主机还有剩余的计算资源，同样用户可以在其上部署虚拟机。图 9-3 右部展示了在中等规模部署环境中服务的运行关系和切换原理。

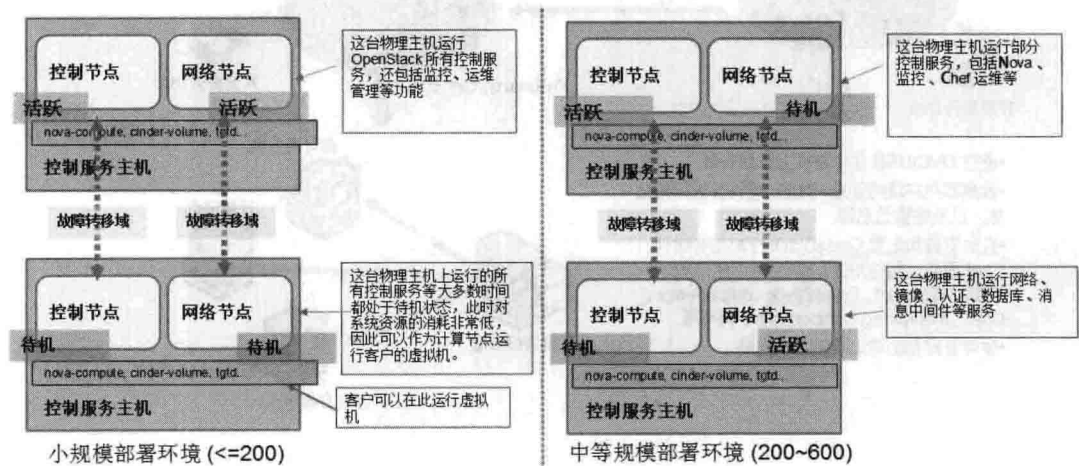


图 9-3 中小规模部署的 HA 切换原理

9.2.7 安装

IBM OpenStack Toolkit 的安装非常自动化、节省时间和可靠。全部过程中，需要人工操作的只有下面很少几步。

- 手工通过 DVD 或者 USB 插入安装媒介，启动安装。
- 在系统重启后，输入客户网络参数、DNS、NTP 服务器地址、邮件地址和服务地址。

址，另外确定是否需要控制节点的 HA。

- 在系统重启后，控制服务已经运行。此时将 HA 待机节点镜像和计算节点镜像导入系统。
- 重启其他主机。自动化安装将批量自动运行，无须人工干涉。

经过上述步骤，用户即得到一个包含控制、计算、网络和基本存储服务的基于 OpenStack 的多节点云平台。OpenStack 服务、监控、运维、DNS、NTP 等服务都已经在运行。用户马上就可以开始部署自己的业务系统。

整个安装过程如图 9-4 所示。

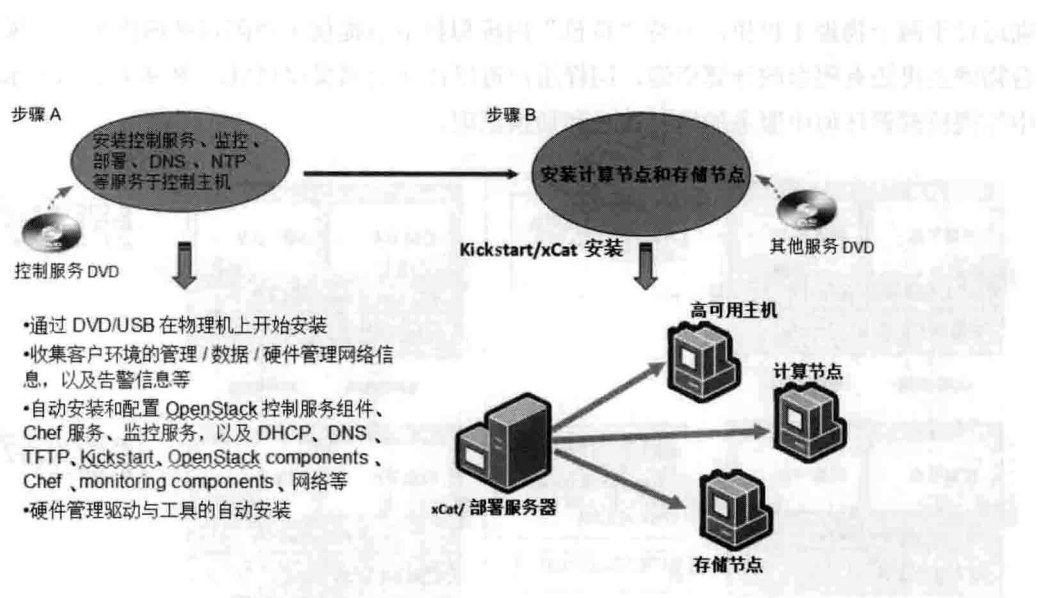


图 9-4 部署过程

下面是安装步骤中需要用户输入的一些步骤。

- (1) 首先用户需要接受 License Agreement，如图 9-5 所示。如果不接受，则安装会自动终止。

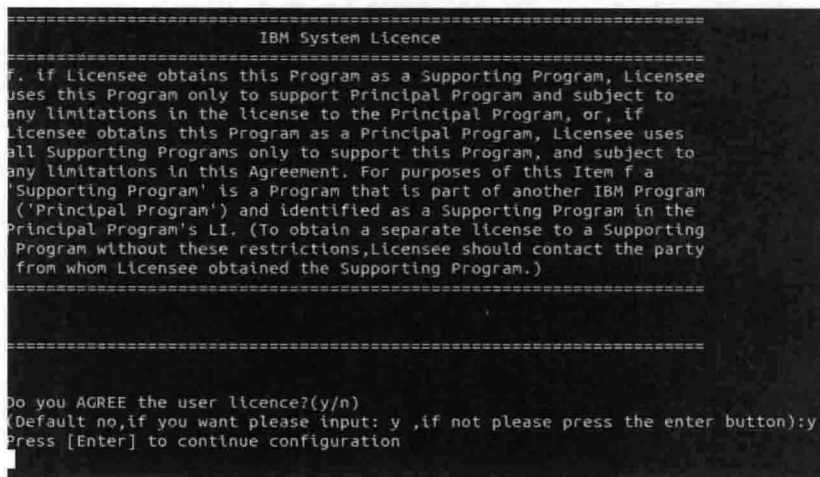


图 9-5 IBM System Licence

所有需要用户输入的信息都有默认值，若直接按 Enter 键，则系统自动使用默认值。

(2) 用户需要选择哪个网卡用于管理网，哪个网卡用于数据网（用于虚拟机运行的网络）：

```
You have ACCEPT the license, then the configure process will begin :
```

```
Please assign the admin network interface :(eth*)
```

```
Available network interface in your system:
```

```
eth0
```

```
eth1
```

```
(Default is eth0 ):
```

```
=====
admin_interface=eth0
=====
```

```
Please assign the data network interface :(eth*)
```

```
Available network interface in your system:
```

```
bradmin
```

```
brdata
```

```
(Default is eth1 ):
```

```
=====
data_interface=eth1
=====
```

(3) 输入管理网络信息:

```
Please assign the admin vlan :(0~4095)
```

```
(Default is 100 ):
```

```
=====
admin_vlan=100

```

```

=====
Please set the admin network start IP address to be used within this cloud : (format
xxx.xxx.xxx.xxx)
(Default is 10.101.0.102 ):
=====
admin_start=10.101.0.102
=====
Please set the admin netmask : (format xxx.xxx.xxx.xxx)
(Default is 255.255.255.0 ):
=====
admin_netmask=255.255.255.0
=====
Please set the admin router : (format xxx.xxx.xxx.xxx)
(Default is 10.101.0.1 ):
=====
admin_router=10.101.0.1

```

(4) 输入硬件管理网络信息，这里是 IBM 设备需要的 IMM 网络：

```

Please set the IMM network start IP address to be used within this cloud : (format
xxx.xxx.xxx.xxx
)
(Default is 10.20.0.2 ):
=====
imm_start=10.20.0.2
=====
Please set the IMM netmask : (format xxx.xxx.xxx.xxx)
(Default is 255.255.255.0 ):
=====
imm_netmask=255.255.255.0
=====
Please set the IMM router : (format xxx.xxx.xxx.xxx)
(Default is 0.0.0.0 ):
=====
imm_router=0.0.0.0
=====
Your imm_router ip address is 0.0.0.0

```

(5) 提供 NTP、DNS 服务。这些信息是可选的。如果用户不提供，则云平台内部会运行独立的服务：

```

Please set the external NTP server ip address: (format xxx.xxx.xxx.xxx)
(Default ip address is ):
=====
ntp_server=
=====
Please set the external DNS server1 ip address: (format xxx.xxx.xxx.xxx)

```



```
(Default ip address is ):
=====
dns_server1=
=====
Please set the external DNS server2 ip address:(format xxx.xxx.xxx.xxx)
(Default ip address is ):
=====
dns_server2=
```

(6) 确定主机的域名和主机命名规则。在最新版本里，可以让用户从集中规则里选择希望的主机命名规则：

```
Please set the domain name:(e.g ibm.com)
(Default domain: lbs):
Please do not set your domain begin with '.' input again:
=====
domain_name=lbs
```

(7) 提供告警用的邮件地址和邮件服务器地址，系统可以发送系统故障信息给管理员：

```
Please set the receive notification email address:(format xxx@xxx.xxx)
(Default is admin@lbs.com ):
=====
admin_email=admin@lbs.com
=====
Please set the email address to send the notication:(format xxx@xxx.xxx)
(Default is wjingxue@cn.ibm.com ):
=====
send_mail=wjingxue@cn.ibm.com
=====
Please set the mail_server address information:(e.g 1.2.3.4 or mail.ibm.com)
(Default is mail.lbs.com ):
=====
mail_server=mail.lbs.com
```

(8) 确定是否需要控制服务的高可用性。如果选择是，则系统自动指定下一个主机为 HA 的待机节点；用户也可以指定某台主机的 MAC 地址，系统会将待机服务部署到这个节点：

```
Note:If you want to install an HA environment, press y, if you only install a
standalone
environment, press enter.
-----
Do you want to set a HA HOST ?
(Default y,if you do not want please input: n ,if yes ,press the [Enter] button):
You will set the HA HOST
=====
ha_enabled=false
```

```

=====
Please input the MAC address for admin HA host:(format FF:FF:FF:FF:FF:FF)
(Default is ):
=====
admin_ha_host=

```

上述就是需要用户输入的信息。由于资源的原因，目前还没有提供 GUI 来让用户输入。但输入信息基本保证可以集成进企业环境，同时没有任何无关的信息需要用户提供。

9.3 Red Hat RDO

RDO 是用来部署 OpenStack 到 Red Hat Enterprise Linux、Fedora 和其他基于 CentOS、Scientific Linux 的社区。这里有文档、论坛等可以帮助用户和其他各个地方的人们一起，开始学习部署 OpenStack，同时还有社区支持的最新的 OpenStack 安装包和脚本供下载。

OpenStack 依赖于底层的操作系统和虚拟化管理程序。RDO 是一站式的、基于 Red Hat 平台的社区。

9.3.1 环境准备

我们使用的还是 VirtualBox，创建了两个虚拟机。根据 OpenStack 网络，控制节点需要三块网卡，一块管理网络使用。由于它可以和外部网络合用，所以两块网卡就可以准备这个环境。

我们有两个节点。节点一的 IP 是 192.168.1.105，为控制节点、网络节点、存储节点和计算节点；节点二的 IP 是 192.168.1.106，为一个计算节点。网关为 192.168.1.1，可以连接到外网。其中 eth0 为管理网口，访问外网是通过节点一的 L3Agent，通过网桥 br-ex 到 eth0 来实现的。我们需要把两台机器的 eth1 设置成 trunk 模式。

因为笔记本只有一个网口，需要增加一个 Host-Only 的网络适配器，然后通过 Host-Only Adapter 来连接到这个新的 VirtualBox Host-Only Network #2。此时系统多了一个网络设备，如下：

```

Ethernet adapter VirtualBox Host-Only Network #2:

Connection-specific DNS Suffix  . : 
Description . . . . . : VirtualBox Host-Only Ethernet Adapter #2

```

```
Physical Address. . . . . : 08-00-27-00-44-63
DHCP Enabled. . . . . : No
Autoconfiguration Enabled . . . . : Yes
Link-local IPv6 Address . . . . . : fe80::44d:b3e2:98f0:122d%59(Preferred)
IPv4 Address. . . . . : 192.168.23.1(Preferred)
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . :
DHCPv6 IAID . . . . . : 990380071
DHCPv6 Client DUID. . . . . : 00-01-00-01-19-07-36-B0-00-21-CC-D9-01-3A
DNS Servers . . . . . : fec0:0:0:ffff::1%1
                        fec0:0:0:ffff::2%1
                        fec0:0:0:ffff::3%1
NetBIOS over Tcpip. . . . . : Enabled
```

启动每个虚拟机的 Adapter 2，选择 Host Adapter。在 Adapter Type 里，我们选择 virtio，这种虚拟网卡驱动会得到最好的性能。

此时每个虚拟机都有两个网卡，分别连接到不同的虚拟网桥上。这样，各个节点虚拟机的每个网口单独组成一个网络，其中连接到 eth1 的网络在主机里面是相通的。而如果需要管理或连接外网，则要将网络流量转到 eth0 上去，eth0 连接到的网桥绑定到物理机的 eth0，eth0 连接到外网。

给两个节点设置静态地址。两个节点的网络设置基本一样，只是 MAC 地址和 IP 地址不同。

编辑/etc/sysconfig/network-scripts/ifcfg-eth0 的内容如下：

```
# cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
ONBOOT=yes
HWADDR=08:00:27:B2:5C:F0
TYPE=Ethernet
BOOTPROTO=none
IPADDR=192.168.1.105
NETMASK=255.255.255.0
/etc/sysconfig/network-scripts/ifcfg-eth1
```

eth1 不能设置 IP 地址，/etc/sysconfig/network-scripts/ifcfg-eth1 的内容如下：

```
# cat /etc/sysconfig/network-scripts/ifcfg-eth1
DEVICE=eth1
TYPE=Ethernet
ONBOOT=yes
```

另外，还需要关闭 NetworkManager，通常关闭 selinux，这个服务运行时会带来许多意想不到的问题和麻烦。

9.3.2 设置源

为了准备安装 OpenStack，首先需要安装 RDO 的源，并更新内核：

```
yum install -y  
http://rdo.fedorapeople.org/openstack-Havana/rdo-release-Havana.rpm
```

执行完后，查看系统/etc/yum.repo.d 目录，可以看到多出下列文件，我们后续安装需要的包都从下面这些源自动下载。

rdo-release.repo 文件：

```
[openstack-Havana]  
name=OpenStack Havana Repository  
baseurl=http://repos.fedorapeople.org/repos/openstack/openstack-Havana/epel-6/  
enabled=1  
skip_if_unavailable=0  
gpgcheck=1  
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-RDO-Havana  
priority=98
```

puppetlabs.repo 文件：

```
[puppetlabs-products]  
name=Puppet Labs Products El 6 - $basearch  
baseurl=http://yum.puppetlabs.com/el/6/products/$basearch  
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-puppetlabs  
enabled=1  
gpgcheck=1  
  
[puppetlabs-deps]  
name=Puppet Labs Dependencies El 6 - $basearch  
baseurl=http://yum.puppetlabs.com/el/6/dependencies/$basearch  
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-puppetlabs  
enabled=1  
gpgcheck=1  
  
[puppetlabs-devel]  
name=Puppet Labs Devel El 6 - $basearch  
baseurl=http://yum.puppetlabs.com/el/6/devel/$basearch  
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-puppetlabs  
enabled=0  
gpgcheck=1  
  
[puppetlabs-products-source]  
name=Puppet Labs Products El 6 - $basearch - Source
```

```
baseurl=http://yum.puppetlabs.com/el/6/products/SRPMS
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-puppetlabs
failovermethod=priority
enabled=0
gpgcheck=1
```

```
[puppetlabs-deps-source]
name=Puppet Labs Source Dependencies El 6 - $basearch - Source
baseurl=http://yum.puppetlabs.com/el/6/dependencies/SRPMS
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-puppetlabs
enabled=0
gpgcheck=1
```

```
[puppetlabs-devel-source]
name=Puppet Labs Devel El 6 - $basearch - Source
baseurl=http://yum.puppetlabs.com/el/6/devel/SRPMS
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-puppetlabs
enabled=0
gpgcheck=1
```

foreman.repo 文件：

```
[foreman]
name=Foreman stable
baseurl=http://yum.theforeman.org/releases/1.3/el6/$basearch
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-foreman
```

```
[foreman-source]
name=Foreman stable - source
baseurl=http://yum.theforeman.org/releases/1.3/el6/source
enabled=0
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-foreman
```

```
[foreman-plugins]
name=Foreman stable - plugins
baseurl=http://yum.theforeman.org/plugins/1.3/el6/$basearch
enabled=1
gpgcheck=0
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-foreman
```

```
[foreman-plugins-source]
name=Foreman stable - plugins source
baseurl=http://yum.theforeman.org/plugins/1.3/el6/source
enabled=0
```

```
gpgcheck=0
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-foreman
```

9.3.3 安装 PackStack

运行如下命令安装 PackStack:

```
yum install -y openstack-packstack
```

9.3.4 定制 PackStack 的 answer 文件

如果不需要特别定制, 可以使用如下命令直接安装:

```
packstack --allinone
```

如果需要定制, 则需要生成一个模板文件:

```
packstack --gen-answer-file=~/.deployment.txt
```

然后对里面的配置进行相应的修改。默认网络采用 nova-network 和 OpenvSwitch 的, 但也可以修改为 Neutron+OpenvSwitch, 租户网络是 VLAN 模式。单节点和多节点的安装基本一样, 只是需要增加计算节点。默认安装节点是控制节点, 但该节点同时也是一个计算节点。

(1) 公钥文件

指定 PackStack 可以使用的公钥文件, 如果未指定, 则 PackStack 在安装时会询问:

```
[general]
```

```
# Path to a Public key to install on servers. If a usable key has not
# been installed on the remote servers the user will be prompted for a
# password and this key will be installed so the password will not be
# required again
CONFIG_SSH_KEY=/root/.ssh/id_rsa.pub
```

(2) 组件选择开关

对 OpenStack 到 Havana 版本为止支持的每个组件, 包括 Dashboard 和 Client, 确定是否指定的节点上安装:

```
# Set to 'y' if you would like Packstack to install MySQL
CONFIG_MYSQL_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack Image
# Service (Glance)
CONFIG_GLANCE_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack Block
# Storage (Cinder)
CONFIG_CINDER_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack Compute
# (Nova)
CONFIG_NOVA_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack
# Networking (Neutron)
CONFIG_NEUTRON_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack
# Dashboard (Horizon)
CONFIG_HORIZON_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack Object
# Storage (Swift)
CONFIG_SWIFT_INSTALL=n

# Set to 'y' if you would like Packstack to install OpenStack
# Metering (Ceilometer)
CONFIG_CEILOMETER_INSTALL=y

# Set to 'y' if you would like Packstack to install OpenStack
# Orchestration (Heat)
CONFIG_HEAT_INSTALL=y

# Set to 'y' if you would like Packstack to install the OpenStack
# Client packages. An admin "rc" file will also be installed
CONFIG_CLIENT_INSTALL=y
```

(3) NTP 与 Nagios

决定是否使用外部 NTP 服务, 是否安装 Nagios, 以及哪些节点不需要再安装 OpenStack 组件和服务:

```
# Comma separated list of NTP servers. Leave plain if Packstack
# should not install ntpd on instances.
```

```
CONFIG_NTP_SERVERS=

# Set to 'y' if you would like Packstack to install Nagios to monitor
# OpenStack hosts
CONFIG_NAGIOS_INSTALL=y

# Comma separated list of servers to be excluded from installation in
# case you are running Packstack the second time with the same answer
# file and don't want Packstack to touch these servers. Leave plain if
# you don't need to exclude any server.
EXCLUDE_SERVERS=
```

(4) MySQL

配置 MySQL 的管理信息:

```
# The IP address of the server on which to install MySQL
CONFIG_MYSQL_HOST=192.168.1.105

# Username for the MySQL admin user
CONFIG_MYSQL_USER=root

# Password for the MySQL admin user
CONFIG_MYSQL_PW=123456
```

(5) QPID

配置 QPID 的管理信息:

```
# The IP address of the server on which to install the QPID service
CONFIG_QPID_HOST=192.168.1.105

# Enable SSL for the QPID service
CONFIG_QPID_ENABLE_SSL=n

# Enable Authentication for the QPID service
CONFIG_QPID_ENABLE_AUTH=n

# The password for the NSS certificate database of the QPID service
CONFIG_QPID_NSS_CERTDB_PW=Passw0rd

# The port in which the QPID service listens to SSL connections
CONFIG_QPID_SSL_PORT=5671

# The filename of the certificate that the QPID service is going to
# use
CONFIG_QPID_SSL_CERT_FILE=/etc/pki/tls/certs/qpid_selfcert.pem
```



```
# The filename of the private key that the QPID service is going to
# use
CONFIG_QPID_SSL_KEY_FILE=/etc/pki/tls/private/qpid_selfkey.pem

# Auto Generates self signed SSL certificate and key
CONFIG_QPID_SSL_SELF_SIGNED=y

# User for qpid authentication
CONFIG_QPID_AUTH_USER=qpid

# Password for user authentication
CONFIG_QPID_AUTH_PASSWORD=Passw0rd
```

注意 Red Hat 默认使用的是 QPID，而不是 OpenStack 社区默认支持的 RabbitMQ。两种消息中间件都支持 AMQP，用户需要根据自己熟悉的技术来选取。

(6) Keystone

配置 Keystone 相关的信息：

```
# The IP address of the server on which to install Keystone
CONFIG_KEYSTONE_HOST=192.168.1.105

# The password to use for the Keystone to access DB
CONFIG_KEYSTONE_DB_PW=Passw0rd

# The token to use for the Keystone service api
CONFIG_KEYSTONE_ADMIN_TOKEN=a865f7dd7bc9481f92577791d9600faa

# The password to use for the Keystone admin user
CONFIG_KEYSTONE_ADMIN_PW=Passw0rd

# The password to use for the Keystone demo user
CONFIG_KEYSTONE_DEMO_PW=demo

# Keystone token format. Use either UUID or PKI
CONFIG_KEYSTONE_TOKEN_FORMAT=PKI
```

这里配置了几个密码，分别用于不同的管理目的。另外，还指定了 admin 和 demo 用户使用的密码。这些密码在登录 Dashboard 时使用。

(7) Glance

配置 Glance 需要的信息：

```
# The IP address of the server on which to install Glance
CONFIG_GLANCE_HOST=192.168.1.105

# The password to use for the Glance to access DB
CONFIG_GLANCE_DB_PW=Passw0rd

# The password to use for the Glance to authenticate with Keystone
CONFIG_GLANCE_KS_PW=Passw0rd
```

这里并没有要求配置 Glance 在哪里存放用户镜像。默认为存放于本节点的文件系统里。

(8) Cinder

配置 Cinder 服务需要的信息:

```
# The IP address of the server on which to install Cinder
CONFIG_CINDER_HOST=192.168.1.105

# The password to use for the Cinder to access DB
CONFIG_CINDER_DB_PW=Passw0rd

# The password to use for the Cinder to authenticate with Keystone
CONFIG_CINDER_KS_PW=Passw0rd

# The Cinder backend to use, valid options are: lvm, gluster, nfs
CONFIG_CINDER_BACKEND=lvm

# Create Cinder's volumes group. This should only be done for testing
# on a proof-of-concept installation of Cinder. This will create a
# file-backed volume group and is not suitable for production usage.
CONFIG_CINDER_VOLUMES_CREATE=y

# Cinder's volumes group size. Note that actual volume size will be
# extended with 3% more space for VG metadata.
CONFIG_CINDER_VOLUMES_SIZE=20G

# A single or comma separated list of gluster volume shares to mount,
# eg: ip-address:/vol-name, domain:/vol-name
CONFIG_CINDER_GLUSTER_MOUNTS=

# A single or comma separated list of NFS exports to mount, eg: ip-
# address:/export-name
CONFIG_CINDER_NFS_MOUNTS=
```

在这里还可以指定块存储服务的后端，支持的存储后端有 LVM、GlusterFS 和 NFS。搭建 GlusterFS 或配置 NFS 本身需要单独的过程来支持。如果要使用 GlusterFS 或 NFS，则

这里假定它们都已经配置好并已经运行，在这里有配置项需要提供相关的信息来与 OpenStack 集成。

另外，这里可以创建一个基于文件的卷组，可以指定大小，只是用来验证而已，并不能支持真正的生产系统使用。

(9) Nova

Nova 服务本身包括许多独立的服务进程，因此，这里的配置项格外多些。

- Nova 各个服务所在的节点地址，包括 nova-api、nova-cert、nova-novncproxy、nova-conductor、nova-scheduler、nova-compute 以及 nova-network 等。
- 对于 nova-scheduler，还可以指定超配（overcommitting）的值，包括 CPU 和内存。
- Nova 访问数据库的密码和与 Keystone 交互的密码。
- nova-network 的参数，包括用于各种网络的物理网卡、固定和浮动 IP 范围、VLAN 信息等。如果指定使用 Neutron，则 nova-network 不会被运行。

具体配置如下：

```
# The IP address of the server on which to install the Nova API
# service
CONFIG_NOVA_API_HOST=192.168.1.105

# The IP address of the server on which to install the Nova Cert
# service
CONFIG_NOVA_CERT_HOST=192.168.1.105

# The IP address of the server on which to install the Nova VNC proxy
CONFIG_NOVA_VNCPROXY_HOST=192.168.1.105

# A comma separated list of IP addresses on which to install the Nova
# Compute services
CONFIG_NOVA_COMPUTE_HOSTS=192.168.1.105

# The IP address of the server on which to install the Nova Conductor
# service
CONFIG_NOVA_CONDUCTOR_HOST=192.168.1.105

# The password to use for the Nova to access DB
CONFIG_NOVA_DB_PW=Passw0rd

# The password to use for the Nova to authenticate with Keystone
CONFIG_NOVA_KS_PW=Passw0rd
```

```
# The IP address of the server on which to install the Nova Scheduler
# service
CONFIG_NOVA_SCHED_HOST=192.168.1.105

# The overcommitment ratio for virtual to physical CPUs. Set to 1.0
# to disable CPU overcommitment
CONFIG_NOVA_SCHED_CPU_ALLOC_RATIO=4.0

# The overcommitment ratio for virtual to physical RAM. Set to 1.0 to
# disable RAM overcommitment
CONFIG_NOVA_SCHED_RAM_ALLOC_RATIO=1.5

# Private interface for Flat DHCP on the Nova compute servers
CONFIG_NOVA_COMPUTE_PRIVIF=eth1

# The list of IP addresses of the server on which to install the Nova
# Network service
CONFIG_NOVA_NETWORK_HOSTS=192.168.1.105

# Nova network manager
CONFIG_NOVA_NETWORK_MANAGER=nova.network.manager.FlatDHCPManager

# Public interface on the Nova network server
CONFIG_NOVA_NETWORK_PUBIF=eth0

# Private interface for network manager on the Nova network server
CONFIG_NOVA_NETWORK_PRIVIF=eth1

# IP Range for network manager
CONFIG_NOVA_NETWORK_FIXEDRANGE=192.168.10.0/22

# IP Range for Floating IP's
CONFIG_NOVA_NETWORK_FLOATRANGE=192.168.20.0/22

# Name of the default floating pool to which the specified floating
# ranges are added to
CONFIG_NOVA_NETWORK_DEFAULTFLOATINGPOOL=nova

# Automatically assign a floating IP to new instances
CONFIG_NOVA_NETWORK_AUTOASSIGNFLOATINGIP=n

# First VLAN for private networks
CONFIG_NOVA_NETWORK_VLAN_START=100

# Number of networks to support
```

```
CONFIG_NOVA_NETWORK_NUMBER=1
```

```
# Number of addresses in each private subnet
```

```
CONFIG_NOVA_NETWORK_SIZE=255
```

(10) Neutron

Neutron 需要更多和更复杂的信息，具体如下。

- neutron-server 服务运行的节点地址，访问数据库密码和与 Keystone 交互的密码。
- l3-agent 运行的节点地址以及绑定的网桥。
- l3-dhcp 服务运行的节点地址。
- LbaaS 服务运行的节点地址。
- meta data 服务运行的节点地址以及密码。
- Neutron 二层交换使用的插件（默认是 OpenvSwitch）、网络类型、VLAN、VXLAN、GRE 等需要的配置信息等。

具体配置如下：

```
# The IP addresses of the server on which to install the Neutron
```

```
# server
```

```
CONFIG_NEUTRON_SERVER_HOST=192.168.1.105
```

```
# The password to use for Neutron to authenticate with Keystone
```

```
CONFIG_NEUTRON_KS_PW=Passw0rd
```

```
# The password to use for Neutron to access DB
```

```
CONFIG_NEUTRON_DB_PW=Passw0rd
```

```
# A comma separated list of IP addresses on which to install Neutron
```

```
# L3 agent
```

```
CONFIG_NEUTRON_L3_HOSTS=192.168.1.105
```

```
# The name of the bridge that the Neutron L3 agent will use for
```

```
# external traffic, or 'provider' if using provider networks
```

```
CONFIG_NEUTRON_L3_EXT_BRIDGE=br-ex
```

```
# A comma separated list of IP addresses on which to install Neutron
```

```
# DHCP agent
```

```
CONFIG_NEUTRON_DHCP_HOSTS=192.168.1.105
```

```
# A comma separated list of IP addresses on which to install Neutron
```

```
# LBaaS agent
```

```
CONFIG_NEUTRON_LBAAS_HOSTS=192.168.1.105
```

```
# The name of the L2 plugin to be used with Neutron
CONFIG_NEUTRON_L2_PLUGIN=openvswitch

# A comma separated list of IP addresses on which to install Neutron
# metadata agent
CONFIG_NEUTRON_METADATA_HOSTS=192.168.1.105

# A comma separated list of IP addresses on which to install Neutron
# metadata agent
CONFIG_NEUTRON_METADATA_PW=Passw0rd

# The type of network to allocate for tenant networks (eg. vlan,
# local, gre)
CONFIG_NEUTRON_LB_TENANT_NETWORK_TYPE=local

# A comma separated list of VLAN ranges for the Neutron linuxbridge
# plugin (eg. physnet1:1:4094,physnet2,physnet3:3000:3999)
CONFIG_NEUTRON_LB_VLAN_RANGES=

# A comma separated list of interface mappings for the Neutron
# linuxbridge plugin (eg. physnet1:br-eth1,physnet2:br-eth2,physnet3
# :br-eth3)
CONFIG_NEUTRON_LB_INTERFACE_MAPPINGS=

# Type of network to allocate for tenant networks (eg. vlan, local,
# gre)
CONFIG_NEUTRON_OVS_TENANT_NETWORK_TYPE=local

# A comma separated list of VLAN ranges for the Neutron openvswitch
# plugin (eg. physnet1:1:4094,physnet2,physnet3:3000:3999)
CONFIG_NEUTRON_OVS_VLAN_RANGES=
# A comma separated list of bridge mappings for the Neutron
# openvswitch plugin (eg. physnet1:br-eth1,physnet2:br-eth2,physnet3
# :br-eth3)
CONFIG_NEUTRON_OVS_BRIDGE_MAPPINGS=

# A comma separated list of colon-separated OVS bridge:interface
# pairs. The interface will be added to the associated bridge.
CONFIG_NEUTRON_OVS_BRIDGE_IFACES=

# A comma separated list of tunnel ranges for the Neutron openvswitch
# plugin (eg. 1:1000)
CONFIG_NEUTRON_OVS_TUNNEL_RANGES=

# The interface for the OVS tunnel. Packstack will override the IP
```

```
# address used for GRE tunnels on this hypervisor to the IP found on
# the specified interface. (eg. eth1)
CONFIG_NEUTRON_OVS_TUNNEL_IF=
```

(11) 客户端访问和 Dashboar

配置客户端访问和 Dashboar 需要的信息:

```
# The IP address of the server on which to install the OpenStack
# client packages. An admin "rc" file will also be installed
CONFIG_OSCLIENT_HOST=192.168.1.105

# The IP address of the server on which to install Horizon
CONFIG_HORIZON_HOST=192.168.1.105

# To set up Horizon communication over https set this to "y"
CONFIG_HORIZON_SSL=n

# PEM encoded certificate to be used for ssl on the https server,
# leave blank if one should be generated, this certificate should not
# require a passphrase
CONFIG_SSL_CERT=

# Keyfile corresponding to the certificate if one was entered
CONFIG_SSL_KEY=
```

(12) Swift

配置 Swift 需要的信息:

```
# The IP address on which to install the Swift proxy service
# (currently only single proxy is supported)
CONFIG_SWIFT_PROXY_HOSTS=192.168.1.105

# The password to use for the Swift to authenticate with Keystone
CONFIG_SWIFT_KS_PW=Passw0rd

# A comma separated list of IP addresses on which to install the
# Swift Storage services, each entry should take the format
# <ipaddress>[/dev], for example 127.0.0.1/vdb will install /dev/vdb
# on 127.0.0.1 as a swift storage device(packstack does not create the
# filesystem, you must do this first), if /dev is omitted Packstack
# will create a loopback device for a test setup
CONFIG_SWIFT_STORAGE_HOSTS=192.168.1.105

# Number of swift storage zones, this number MUST be no bigger than
# the number of storage devices configured
```

```
CONFIG_SWIFT_STORAGE_ZONES=1

# Number of swift storage replicas, this number MUST be no bigger
# than the number of storage zones configured
CONFIG_SWIFT_STORAGE_REPLICAS=1

# FileSystem type for storage nodes
CONFIG_SWIFT_STORAGE_FSTYPE=ext4

# Shared secret for Swift
CONFIG_SWIFT_HASH=a22562fa882f4d21

# Size of the swift loopback file storage device
CONFIG_SWIFT_STORAGE_SIZE=2G
```

Swift 是一个非常大的内容，其部署结构有许多变化，对网络压力有一定的要求。因此，可以在一个小型的环境进行尝试，生产环境则需要单独的存储网络，根据业务需求来设计部署模型。

(13) Heat

配置 Heat 需要的信息：

```
# The IP address of the server on which to install Heat service
CONFIG_HEAT_HOST=192.168.1.105

# The password used by Heat user to authenticate against MySQL
CONFIG_HEAT_DB_PW=Passw0rd

# The password to use for the Heat to authenticate with Keystone
CONFIG_HEAT_KS_PW=Passw0rd

# Set to 'y' if you would like Packstack to install Heat CloudWatch
# API
CONFIG_HEAT_CLOUDWATCH_INSTALL=y

# Set to 'y' if you would like Packstack to install Heat
# CloudFormation API
CONFIG_HEAT_CFN_INSTALL=n

# The IP address of the server on which to install Heat CloudWatch
# API service
CONFIG_HEAT_CLOUDWATCH_HOST=192.168.1.105

# The IP address of the server on which to install Heat
# CloudFormation API service
```



```
CONFIG_HEAT_CFN_HOST=192.168.1.105
```

(14) Ceilometer

配置 Ceilometer 需要的信息:

```
# The IP address of the server on which to install Ceilometer
```

```
CONFIG_CEILOMETER_HOST=192.168.1.105
```

```
# Secret key for signing metering messages.
```

```
CONFIG_CEILOMETER_SECRET=80e23e2b82224f49
```

```
# The password to use for Ceilometer to authenticate with Keystone
```

```
CONFIG_CEILOMETER_KS_PW=Passw0rd
```

(15) Nagios

配置 Nagios 需要的信息:

```
# The IP address of the server on which to install the Nagios server
```

```
CONFIG_NAGIOS_HOST=192.168.1.105
```

```
# The password of the nagiosadmin user on the Nagios server
```

```
CONFIG_NAGIOS_PW=Passw0rd
```

(16) Tempest

配置 Tempest 需要的信息:

```
# Whether to configure tempest for testing
```

```
CONFIG_PROVISION_TEMPEST=n
```

```
# The uri of the tempest git repository to use
```

```
CONFIG_PROVISION_TEMPEST_REPO_URI=https://github.com/openstack/tempest.git
```

```
# The revision of the tempest git repository to use
```

```
CONFIG_PROVISION_TEMPEST_REPO_REVISION=master
```

(17) 其他配置

其他配置如下:

```
# Whether to provision for demo usage and testing
```

```
CONFIG_PROVISION_DEMO=n
```

```
# The CIDR network address for the floating IP subnet
```

```
CONFIG_PROVISION_DEMO_FLOATRANGE=172.24.4.224/28
```

```
# Whether to configure the ovs external bridge in an all-in-one
```

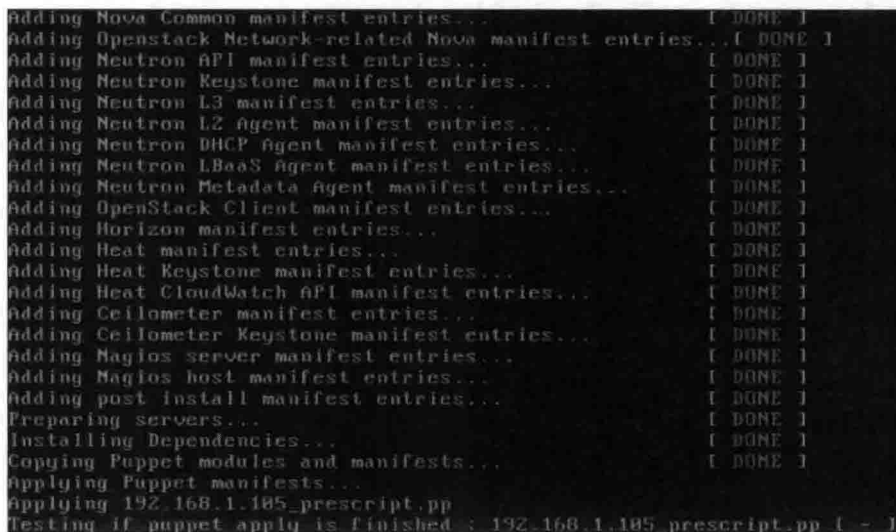
```
# deployment
CONFIG_PROVISION_ALL_IN_ONE_OVS_BRIDGE=n
```

9.3.5 安装

在编辑完响应文件之后，就可以开始根据这个文件来进行安装：

```
packstack -answer-file=~/.deployment.txt
```

RDO 安装过程如图 9-6 所示。



```
Adding Nova Common manifest entries... [ DONE ]
Adding Openstack Network-related Nova manifest entries... [ DONE ]
Adding Neutron API manifest entries... [ DONE ]
Adding Neutron Keystone manifest entries... [ DONE ]
Adding Neutron L3 manifest entries... [ DONE ]
Adding Neutron L2 Agent manifest entries... [ DONE ]
Adding Neutron DHCP Agent manifest entries... [ DONE ]
Adding Neutron LBaaS Agent manifest entries... [ DONE ]
Adding Neutron Metadata Agent manifest entries... [ DONE ]
Adding OpenStack Client manifest entries... [ DONE ]
Adding Horizon manifest entries... [ DONE ]
Adding Heat manifest entries... [ DONE ]
Adding Heat Keystone manifest entries... [ DONE ]
Adding Heat CloudWatch API manifest entries... [ DONE ]
Adding Ceilometer manifest entries... [ DONE ]
Adding Ceilometer Keystone manifest entries... [ DONE ]
Adding Magios server manifest entries... [ DONE ]
Adding Magios host manifest entries... [ DONE ]
Adding post install manifest entries... [ DONE ]
Preparing servers... [ DONE ]
Installing Dependencies... [ DONE ]
Copying Puppet modules and manifests... [ DONE ]
Applying Puppet manifests...
Applying 192.168.1.105_prescript.pp
Testing if puppet apply is finished : 192.168.1.105_prescript.pp [ - ]
```

图 9-6 RDO 安装过程

packstack 在安装过程中，会在系统的/var/tmp/packstack 目录下安装时间戳（在下面例子里是 20140223-084101-TJeBIb），生成一个目录，里面存放需要的各种文件：

```
[root@RDO-OpenStack-Compute 20140223-084101-TJeBIb]# ll
total 12
drwx-----. 2 root root 4096 Feb 23 08:43 manifests
-rw-----. 1 root root 6531 Feb 23 08:43 openstack-setup.log
[root@RDO-OpenStack-Compute 20140223-084101-TJeBIb]# cd manifests/
[root@RDO-OpenStack-Compute manifests]# ll
total 108
-rw-----. 1 root root 1440 Feb 23 08:41 192.168.1.105_api_nova.pp
-rw-----. 1 root root 1828 Feb 23 08:41 192.168.1.105_ceilometer.pp
-rw-----. 1 root root 1239 Feb 23 08:41 192.168.1.105_cinder.pp
-rw-----. 1 root root 1159 Feb 23 08:41 192.168.1.105_glance.pp
-rw-----. 1 root root 519 Feb 23 08:41 192.168.1.105_heatcw.pp
```

```

-rw-----, 1 root root 811 Feb 23 08:41 192.168.1.105_heat.pp
-rw-----, 1 root root 1131 Feb 23 08:41 192.168.1.105_horizon.pp
-rw-----, 1 root root 3212 Feb 23 08:41 192.168.1.105_keystone.pp
-rw-----, 1 root root 14807 Feb 23 08:43 192.168.1.105_keystone.pp.log
-rw-----, 1 root root 1867 Feb 23 08:41 192.168.1.105_mysql.pp
-rw-----, 1 root root 5308 Feb 23 08:43 192.168.1.105_mysql.pp.log
-rw-----, 1 root root 1441 Feb 23 08:41 192.168.1.105_nagios_nrpe.pp
-rw-----, 1 root root 4515 Feb 23 08:41 192.168.1.105_nagios.pp
-rw-----, 1 root root 2851 Feb 23 08:41 192.168.1.105_neutron.pp
-rw-----, 1 root root 5290 Feb 23 08:41 192.168.1.105_nova.pp
-rw-----, 1 root root 1060 Feb 23 08:41 192.168.1.105_osclient.pp
-rw-----, 1 root root 231 Feb 23 08:41 192.168.1.105_postscript.pp
-rw-----, 1 root root 567 Feb 23 08:41 192.168.1.105_prescript.pp
-rw-----, 1 root root 634 Feb 23 08:43 192.168.1.105_prescript.pp.log
-rw-----, 1 root root 874 Feb 23 08:41 192.168.1.105_qpid.pp
-rw-----, 1 root root 288 Feb 23 08:43 192.168.1.105_qpid.pp.log
[root@RDO-OpenStack-Compute manifests]#

```

如果打开一个文件，如 `192.168.1.105_glance.pp` 文件，会看到下列内容，这些都是我们配置每个服务时在配置文件里(`/etc/xxx/xxx.conf`)必须修改的信息：

```
Exec { timeout => 300 }
```

```

class {"glance::api":
    auth_host => "192.168.1.105",
    keystone_tenant => "services",
    keystone_user => "glance",
    keystone_password => "Passw0rd",
    pipeline => 'keystone',
    sql_connection => "mysql://glance:Passw0rd@192.168.1.105/glance"
}

```

```
class { 'glance::backend::file': }
```

```

class {"glance::registry":
    auth_host => "192.168.1.105",
    keystone_tenant => "services",
    keystone_user => "glance",
    keystone_password => "Passw0rd",
    sql_connection => "mysql://glance:Passw0rd@192.168.1.105/glance"
}

```

```

class { 'glance::notify::qpid':
    qpid_password => 'guest',
    qpid_username => 'guest',
    qpid_hostname => '192.168.1.105',
}

```

```

    qpid_port => '5672',
    qpid_protocol => 'tcp'
}
# Create firewall rules to allow only the hosts that need to connect
# to glance

$hosts = [ '192.168.1.105' ]

define add_allow_host {
    $source = $title ? {
        'ALL' => '0.0.0.0/0',
        default => $title,
    }
    firewall { "001 glance incoming ${title}":
        proto => 'tcp',
        dport => ['9292'],
        action => 'accept',
        source => $source,
    }
}

add_allow_host {$hosts:}

```

在同一目录下，packstack 生成日志文件，如 openstack-setup.log，如果有问题发生，可以在这里查看一些错误原因：

```

2014-02-23 08:41:01::INFO::shell::78::root:: [localhost] Executing script:
rm -rf /var/tmp/packstack/20140223-084101-TJEBIB/manifests/*pp
2014-02-23 08:41:01::INFO::shell::78::root:: [192.168.1.105] Executing script:
mkdir -p ~/.ssh
chmod 500 ~/.ssh
grep 'ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAqisHCnILxumm+AmlxUBoHzHbWgtxCNUrr8NxHBigZ2nTfqKK
9w1KHSXpXU+u3AcHp7yUkvzi6r7FnD+3tHkRx720GgRrACxD3M/BdyY9eMvoJf9vT3AmBjaWosF/
g2MZweZc+NXMJalH8YgTLJ4H0lWluC4GWdIrLyZmVYP9vYW+m/7spMqh9rMUVDzpCis6RVg/r5tv
HeHzd6B+CNa9HoVj6NBzrOPBW9S2gI5pKiOjgUpOECs/WdWaApp9odc+CkKz/d5MDSZSjaLYjsJy
fVQYm4Zg94ZDLxo6q053IinZI2syN6dzoWg8HXi8IiThfZEHxTQDhY7sDGzMThqCTQ==
root@RDO-OpenStack-Compute' ~/.ssh/authorized_keys > /dev/null 2>&1 || echo
ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAqisHCnILxumm+AmlxUBoHzHbWgtxCNUrr8NxHBigZ2nTfqKK
9w1KHSXpXU+u3AcHp7yUkvzi6r7FnD+3tHkRx720GgRrACxD3M/BdyY9eMvoJf9vT3AmBjaWosF/
g2MZweZc+NXMJalH8YgTLJ4H0lWluC4GWdIrLyZmVYP9vYW+m/7spMqh9rMUVDzpCis6RVg/r5tv
HeHzd6B+CNa9HoVj6NBzrOPBW9S2gI5pKiOjgUpOECs/WdWaApp9odc+CkKz/d5MDSZSjaLYjsJy
fVQYm4Zg94ZDLxo6q053IinZI2syN6dzoWg8HXi8IiThfZEHxTQDhY7sDGzMThqCTQ==
root@RDO-OpenStack-Compute >> ~/.ssh/authorized_keys
chmod 400 ~/.ssh/authorized_keys

```

```

restorecon -r ~/.ssh
2014-02-23 08:41:01::INFO::shell::78::root:: [192.168.1.105] Executing script:
cat /etc/redhat-release
2014-02-23 08:41:01::INFO::shell::78::root:: [192.168.1.105] Executing script:
mkdir -p /var/tmp/packstack
mkdir --mode 0700 /var/tmp/packstack/0212b047b8c847089fb01969ba2394ef
mkdir --mode 0700 /var/tmp/packstack/0212b047b8c847089fb01969ba2394ef/modules
mkdir --mode 0700
/var/tmp/packstack/0212b047b8c847089fb01969ba2394ef/resources
2014-02-23 08:41:02::INFO::shell::78::root:: [192.168.1.105] Executing script:
rpm -q --whatprovides lvm2 || yum install -y lvm2
2014-02-23 08:41:02::INFO::shell::78::root:: [192.168.1.105] Executing script:
vgdisplay cinder-volumes

```

安装和运行过程中如果出现问题，可以到 <http://openstack.redhat.com/forum/> 寻找答案。

如下为一个最近发现的问题和解决办法：http://openstack.redhat.com/forum/discussion/910/havanna-issues-with-multi-node#Item_14。RDO 发现在多节点部署有问题，如在计算节点创建的虚拟机都会失败。解决办法是在安装完成后在控制节点添加一条 iptables 的规则：

```
iptables -I INPUT -s 192.168.1.105/24 -p tcp -dport 9696 -j ACCEPT
```

并保存规则：

```
service iptables save.
```

默认安装后，将会创建一个文件，作为存储的后端，但这个只能做一些实验而已。下面步骤创建一个 LVM 的卷组 cinder-volumes，可以提高性能：

```

umount /dev/sda2
pvcreate /dev/sda2
vgcreate cinder-volumes /dev/sda2

```

修改/etc/fstab，去掉 cinder-volume 的开机挂载，不然可能会导致系统无法重启：

```
sed -i '/cinder-volume/s/^/#/' /etc/fstab
```

由于目前 RDO 的问题，没有创建 br-ex，需要我们手工来创建：

```

# cat /etc/sysconfig/network-scripts/ifcfg-br-ex
DEVICE=br-ex
IPADDR=192.168.1.20
GATEWAY=192.168.1.1
ONBOOT=yes

```

修改/etc/sysconfig/network-scripts/ifcfg-eth0，注意，这里一定要加上 eth0 的 MAC 地址，不然会出错：

```
# cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
ONBOOT=yes
HWADDR=08:E0:01:D8:43:BE
```

然后运行下面命令：

```
ovs-vsctl add-port br-ex eth0; service network restart
```

当运行 `ovs-vsctl add-port` 的时候，网络会中断，所以两条命令需要一起执行：

Dashboard 可以通过 `http://192.168.1.105/dashboard` 登录，用户名为 `admin`，密码在 `/root` 目录里的 `keystonerc_admin` 文件里：

```
# cat /root/keystonerc_admin
export OS_USERNAME=admin
export OS_TENANT_NAME=admin
export OS_PASSWORD=Passw0rd
export OS_AUTH_URL=http://192.168.1.105:35357/v2.0/
```

9.3.6 增加计算节点

对于使用者来说，在初始部署后，增加新的节点是必须做的事情。任何云平台，都必须具备可扩展能力。

首先，需要编辑在第一次运行 `packstack` 时生成的响应文件（answer file），这个文件可以在运行了 `packstack` 命令的目录中找到，例如 `packstack-answer-$date-$time.txt`：

```
vi $youranswerfile
```

修改网卡名字，把 `CONFIG_NOVA_COMPUTE_PRIVIF` 和 `CONFIG_NOVA_NETWORK_PRIVIF` 从其他名字修改为合适的承载计算服务的网卡，例如 `eth1`。

修改 IP 地址，将 `CONFIG_NOVA_COMPUTE_HOSTS` 赋予的前一节点的 IP 地址修改为新节点需要的 IP 地址，检查确认 `CONFIG_NOVA_NETWORK_HOSTS` 指向控制节点的 IP 地址。

如果不使用 `Neutron`，而是使用 `flat_dhcp nova network`，需要将 `CONFIG_NETWORK_HOST` 修改为 `CONFIG_NOVA_NETWORK_HOSTS`，并提供相应的 IP 地址。然后重新运行 `packstack`，并以修改后的响应文件为输入：

```
sudo packstack --answer-file=$youranswerfile
```

`packstack` 将会询问每个节点的 root 用户密码。

9.4 Mirantis Fuel

在 2003 年，安装部署一直是困扰 OpenStack 初学者的一大难题。要搭建一个基本能用的 OpenStack 环境非常麻烦，需要自己装机，自己配置安装源，根据文档输入各种命令，也没有成型的文档，这一切变得非常艰难。笔者听过有人经过好几个月还处于安装阶段。安装的困难和艰苦吓退了很多想尝试的人。

如图 9-7 所示，OpenStack CAMPS Mirantis 是一家非常活跃的杰出的 OpenStack 服务集成商，在社区贡献排名前 5 名中是唯一一家依靠软件和服务的公司（其他分别是 Red Hat、HP、IBM、Rackspace）。另外，Mirantis 公司负责和推动了 OpenStack 中几个新的而且使用价值非常高的项目，如 Savanna 用于大数据集成与处理，Murano 用于 Windows 和 Linux 的服务管理与配置。

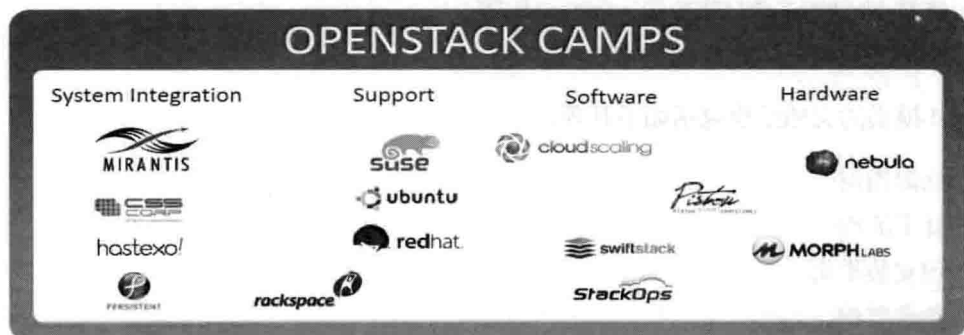


图 9-7 OpenStack CAMPS

9.4.1 Fuel 简介

Mirantis Fuel 是 Mirantis 开发的一款用于 OpenStack 安装和管理的免费软件，具有良好的 GUI 交互体验，并支持多个 OpenStack 版本及插件，版本更新节奏非常快，平均每两个月就能提供一个相对稳定的社区版。

Mirantis 提供了 ISO 和 IMG 两种安装媒介，并提供了在 VirtualBox 里安装的 VBscript 脚本。另外还有非常详细的文档，从原理、最佳实践到步骤都有详细的说明，沿袭硅谷人

做事完整、用户优先的传统。总体上，Mirantis Fuel 是所有免费版本中最强大和用户体验最友好的版本。

Mirantis Fuel 是个管理和部署 OpenStack 环境的端到端“一键部署”工具，功能包括 PXE 方式的系统部署、DHCP 服务、OpenStack 环境管理与编排和 Puppet 配置管理等功能。另外还包括 OpenStack 关键服务健康检查、日志实时查看等。用图形化界面，并且关注使用时的具体问题，是 Mirantis Fuel 最大的优点和亮点。从 Fuel 可以看出 Mirantis 在企业级部署实施方面的经验有多强。Fuel 的主要特点如下。

- 节点的自动发现和预校验，这是非常实用的一个功能，在部署前就可以知道配置正确与否。
- 配置简单、快速，操作都可以在界面输入或拖曳来完成。
- 支持多种操作系统和发行版，支持 HA 部署，这个是非常有用的功能。
- 对外提供 API 对环境进行管理和配置，例如动态添加计算/存储节点。
- 自带健康检查工具，这个也是实施时的一个利器。
- 支持 Neutron，如 GRE 和 namespace 等。
- 子网能配置具体使用哪个物理网卡等。

Fuel 提供的文档至少包括如下几项。

- 安装指南。
- 补丁流程。
- 预安装指南。
- 参考架构。
- 用户使用指南。
- Release nodes。

目前 Mirantis Fuel 已经发布了 4.0 版本和 4.1 版本，都是基于 Havana 版本的，新发布的 5.0 则是支持 IceHouse 版本的。另外其向导式的 Wizard，涵盖了企业级部署时对多种技术的选取，具有非常强的学习和实用价值，而不是把这大量的工作留给用户去决定。

9.4.2 Mirantis 支持的架构

假如具备足够的 OpenStack 知识，并且对各种组件和网络非常熟悉，Mirantis Fuel 是可以足够搭建一个可用的生产环境的。Mirantis Fuel 提供了控制节点的高可靠性，而且从技

术角度，几乎无法挑出其不足。另外，还可以和 Ceph 集成，与 Swift 交互，而无论 Ceph、Swift 等这些复杂技术，都可以通过 Fuel 这个控制界面配出来。

图 9-8 展示了一个多节点部署模型，其中，控制节点必须至少是三个，而且扩展也必须是奇数个的。这是因为其中使用到的 Galera 组件用于提供 MySQL 的 Active-Active 工作方式，而 Galera 的仲裁必须是奇数个节点。回想 IBM 的 GPFS，其中的仲裁也限制了必须是奇数个节点。原理是相似的。

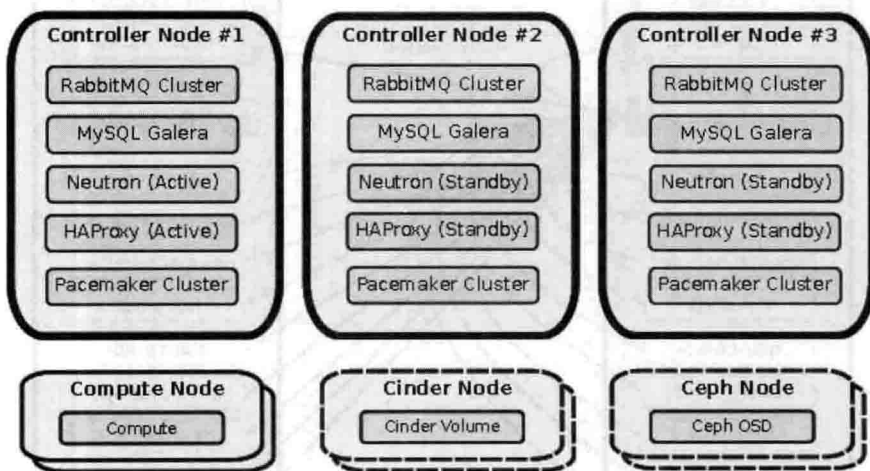


图 9-8 多节点的 HA 部署模式

如图 9-9 所示，这里使用了 HAProxy、Pacemaker、Corosync 来完成 HA 集群的管理。

OpenStack 服务可以分为：

- 无状态服务，这些服务理论上是可以同时运行多个活跃的服务实例的，HAProxy 可以将请求根据配置，调度到任何一个服务进程来处理。无状态服务本身并不保留状态和中间数据，而是这些信息都保存在数据库中。因此可以想象到的一个问题是：性能，因为每次处理，都需要访问数据库，进行查询和检索。无状态服务根据调度方式又可以分为两类。
 - 由 HAProxy 来驱动，包括所有*-API 服务、neutron-server 等。
 - 由 RabbitMQ 来驱动，这些服务监听 RabbitMQ 的消息，一个服务进程从队列中取走一个消息，则同类的其他服务进程就不会再处理。工作原理类似于生产者-消费者模型。
- 有状态服务，包括 Neutron 的所有代理服务（Agent），它们都只能在 Active-Standby

模式下工作。

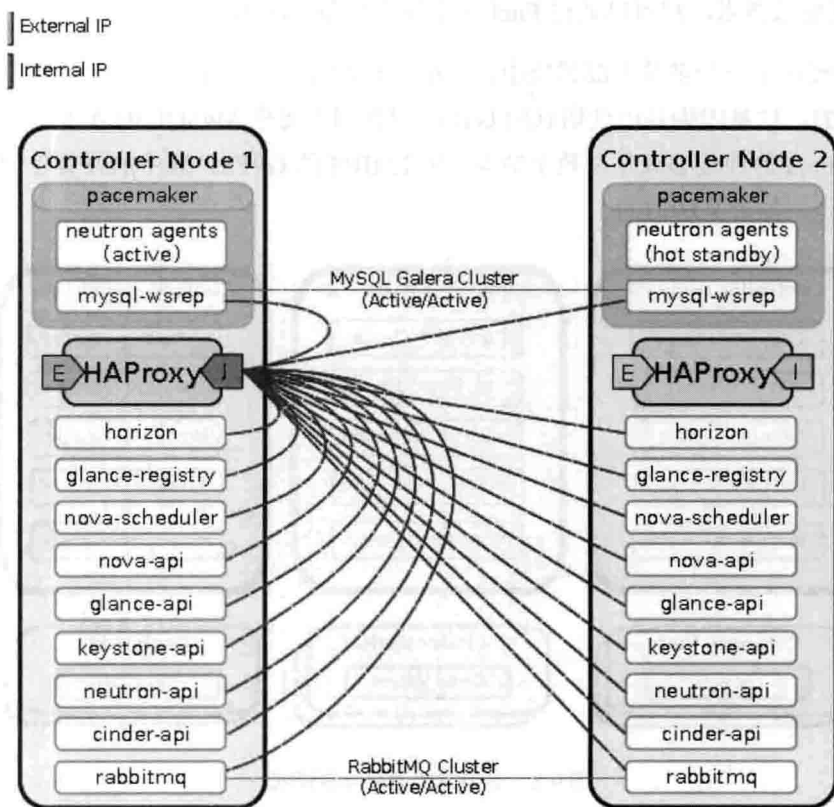


图 9-9 多节点部署模式的 HA 控制节点原理

图 9-10 和图 9-11 分别展示了使用 Neutron 和 Openvswitch，网络模型分别为 VLAN 和 GRE 的网络模型。

图 9-12 展示了一个部署环境的物理节点和网络关系。这种物理和网络，不但对于学习实验有很强的真实感，而且确实可用于规划真实的生产环境。

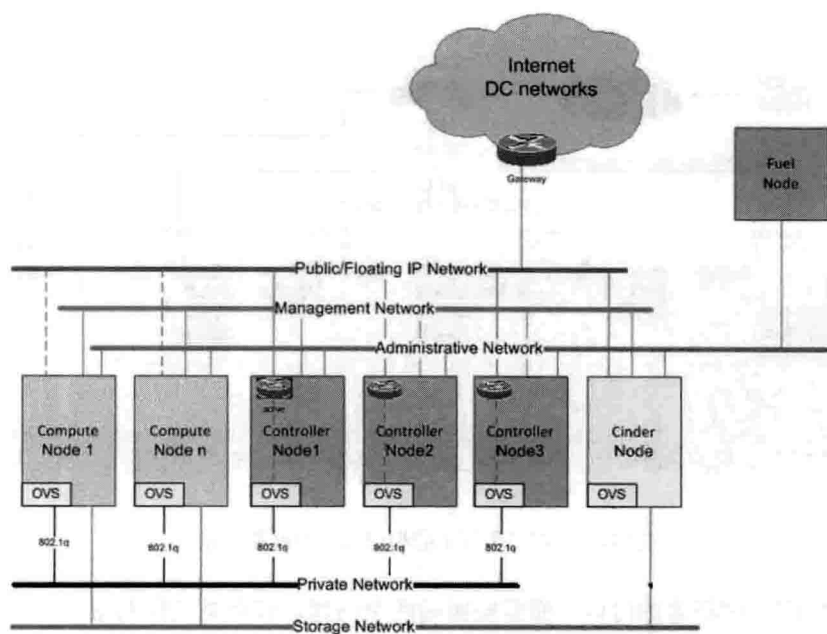


图 9-10 使用了 VLAN 和 OVS 的 Neutron 网络

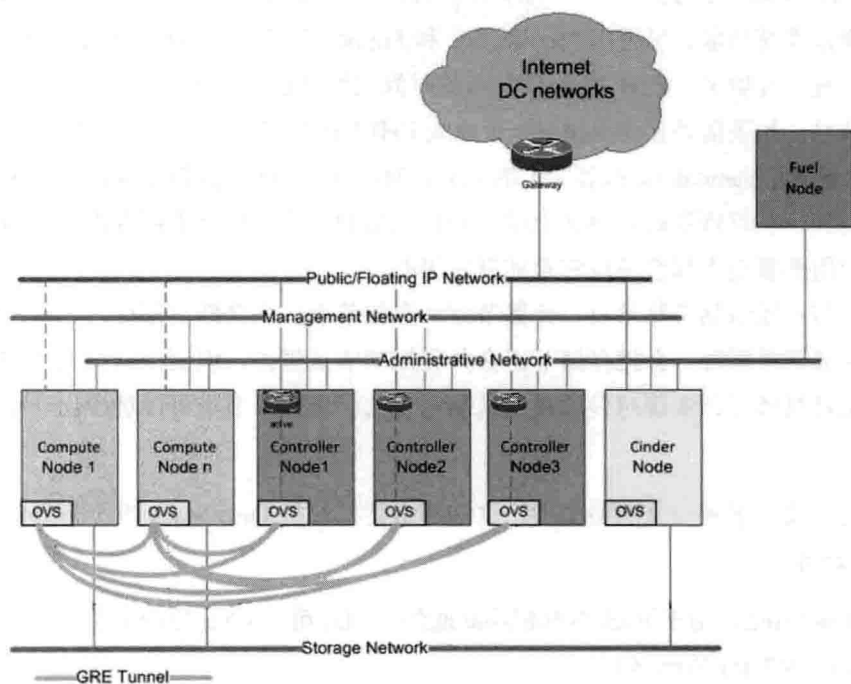


图 9-11 使用 GRE 和 OVS 的 Neutron 网络

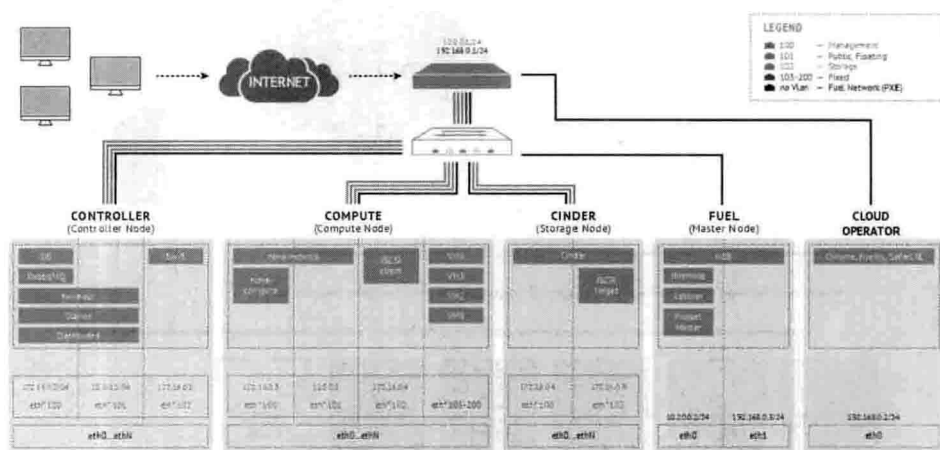


图 9-12 部署环境中的物理节点和网络关系

在研究任何一种部署的时候，都需要明确两个问题，节点类型和网络关系。在 Mirantis Fuel 里，节点包括以下几种。

- **Master Node**。这其实是一个部署控制器，由开源软件 Cobbler 提供的 PXE 方式安装操作系统功能，另外由 Mcollective 和 Puppet 分别提供编排（Orchestration）服务和配置管理服务。Fuel ISO 包发布的时候已经打包了 CentOS 6.4 和 Ubuntu 12.04 安装包，如果需要使用 Red Hat 企业版 RHEL 6.4，要自己手动上传；目前可以支持 OpenStack Standalone 或者 HA 的安装。Master Node 里面包括操作系统镜像、硬件发现和一些管理逻辑，再加上简单而且实用的界面，功能不需要很多，但经常研究和部署的人都会发现它是非常实用的。
- 其他节点还包括控制节点、计算节点和存储节点，在文档里都称为 **Slave Node**。在 HA 部署模型时，会把存储节点合并到控制节点里去，但至少需要三个控制节点，这也是具体设计实现时候的投票机制需要的，奇数个节点可以选举出一个有效的领导者。

Fuel 也定义了下列一些网络类型，这些网络类型都是 OpenStack 典型部署需要的，除了 Fuel Network。

- **Fuel network**：用于 Fuel 内部模块间通信，同时用于 PXE 启动部署（推荐是一个没有 VLAN TAG 的网络）。
- **Public network**：用于虚拟机访问外网、Internet、办公网络（使用的是 VLAN 101）。

- Floating network: 用于从外部网络访问虚拟机（在这里与 Public network 共享同一个二层接口，即 VLAN 101）。
- Management network: 用于 OpenStack 服务之间的通信（VLAN 102）。
- Storage network: 用于存储相关的网络流量（VLAN 103）。
- Fixed network: 一个或者多个网络用于虚拟机之间通信（VLAN 104）。

9.4.3 Fuel 安装

首先需要定义如下 4 个网络。

- Net1
 - Network name: VirtualBox host-only Ethernet Adapter#1
 - Purpose: Fuel administrator network
 - IP block: 10.20.0.0/24
 - Linux device: eth0
- Net2
 - Network name: VirtualBox host-only Ethernet Adapter#2
 - Purpose: public/ floating network
 - IP block: 172.16.0.0/24
 - Linux device: eth1
- Net3
 - Network name: VirtualBox host-only Ethernet Adapter#3
 - Purpose: management/ internal network
 - IP block: 192.168.10.0/24
 - Linux device: eth2
- Net4
 - Network name: VirtualBox host-only Ethernet Adapter#4
 - Purpose: Storage network
 - IP block: 192.168.20.0/24
 - Linux device: eth3

其次是创建虚拟机，也可以是相应的物理节点。虚拟机可以是 VirtualBox 的，也可以是 VMware 的，注意将混杂模式打开：

- VM1
 - Name: Fuel_4.0_Master
 - vCPU:1
 - Memory :1536MB
 - Disk:30GB
 - Networks: net1
- VM2
 - Name : Fuel_4.0_controller
 - vCPU:1
 - Memory :1536MB
 - Disk:30GB
 - Networks:net1,net2,net3
- VM3
 - Name: Fuel_4.0_compute1
 - vCPU:2
 - Memory :1536MB
 - Disk:30GB
 - Networks:net1,net2,net3,net4
- VM4
 - Name: Fuel_4.0_storage
 - vCPU:2
 - Memory :768MB
 - Disk:30GB
 - Networks:net1,net2,net3,net4

创建网络 net1、net2、net3、net4，注意不能启用 DHCP，否则会干扰 Fuel 的 DHCP 服务。

安装 Fuel 的 Master Node，虚拟机名字为 fuel_4.0_master。网卡需要选用 net1，也就是 VirtualBox 的“VirtualBox host-only Ethernet Adapter#1”网络。

虚拟机选择“网络启动”，这样就可以用 Fuel 的 PXE 来安装控制了。

在挂载 Mirantis Fuel 4.0 ISO 后，就开始了安装 Master Node 的过程。这个过程和平时

安装一个 CentOS ISO 的过程完全一样。从 ISO 启动后,就看到如图 9-13 所示的安装界面。安装界面只有一个选项,那就是安装 Fuel。在键盘按下“Enter”之后,安装就开始了,如图 9-14 所示。



图 9-13 Mirantis Fuel ISO 开始安装

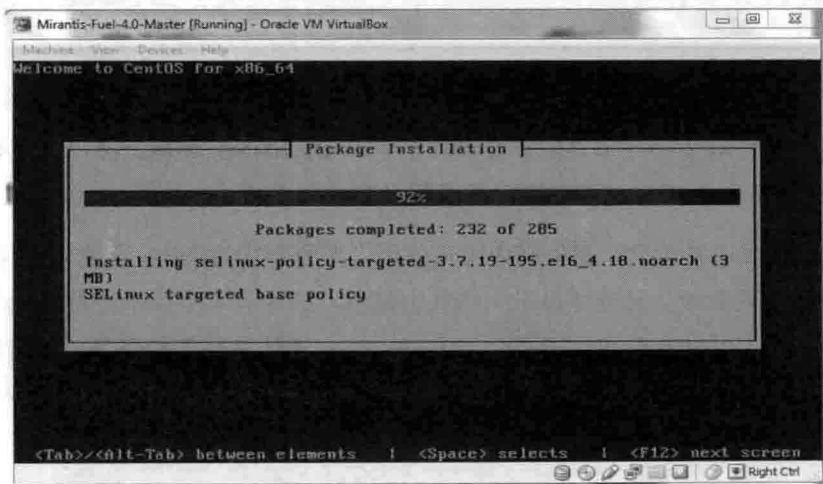


图 9-14 Mirantis Fuel ISO 安装软件包

这个安装界面我们并不陌生，一系列包需要被安装，但实际上只有 285 个包，远远少于标准 ISO 的包的数量，显示这是个经过剪裁的 ISO。

后面还有 Post-Install 阶段。回顾前面讲述过的 ISO 定制和安装过程，在这里可以加入许多自己的逻辑，用于安装各种应用的包，并对系统进行一定的自动化配置。

在安装完成后，就显示如图 9-15 所示的界面，并有相应的登录信息、用户名和密码。

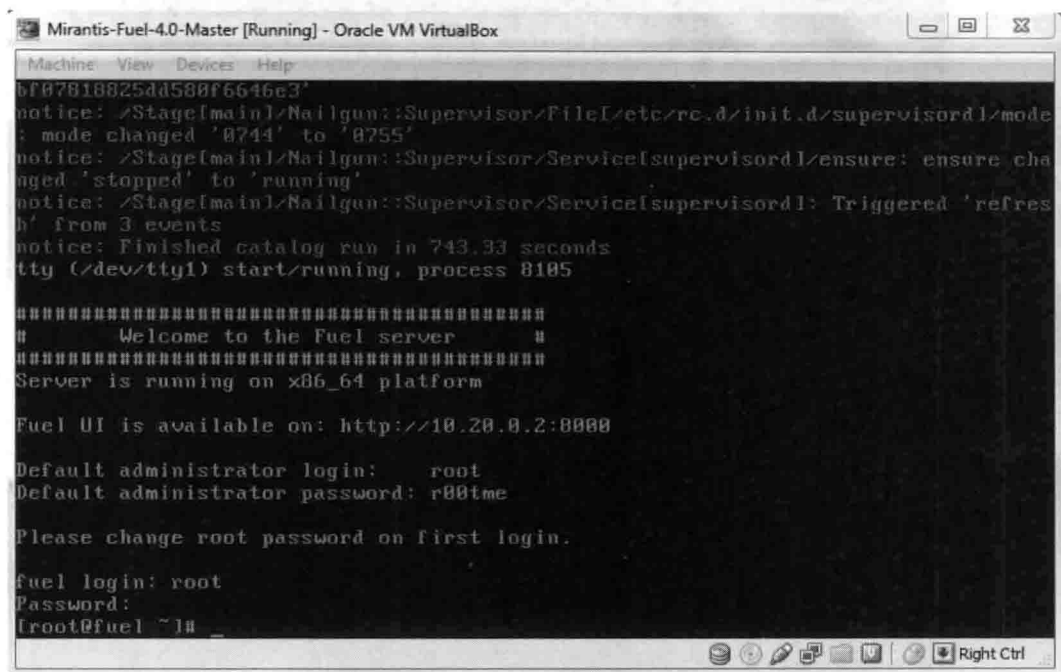


图 9-15 Mirantis Fuel Master 节点安装完毕

在最后，看到图 9-16（该界面已经添加了一个 OpenStack 环境定义，并且有 8 个节点被发现），表示 Fuel 的安装已经完成，可以登录 <http://10.20.0.2:8000/>。

这个界面已经支持中文，便于国内用户使用。界面上非常简洁，只有“环境”、“版本”和“支持”几个菜单项。另外还显示一些状态信息，比如有多少个节点被发现等。在这个管理界面上，我们可以查看和管理某个 OpenStack 部署环境的各种信息。首先是发现的节点，会显示在这个列表里，并列几个角色，如控制节点、计算节点、存储节点，并且可以给各个节点分配所属的角色。

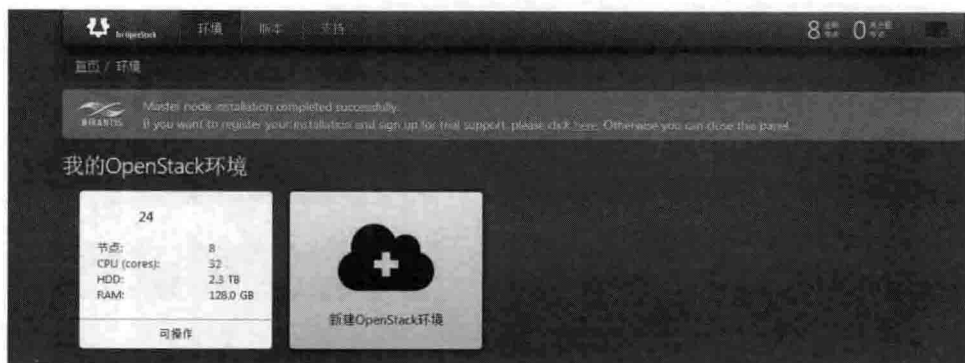


图 9-16 Mirantis Fuel 部署管理界面

如果 Web 页面不能正常访问，可能是本机的防火墙引起的，请先禁用防火墙再试。如果使用了浏览器 http 代理，请关闭代理直接访问。

查看“版本”菜单，界面中显示包中支持并且可以安装的版本，如图 9-17 所示。

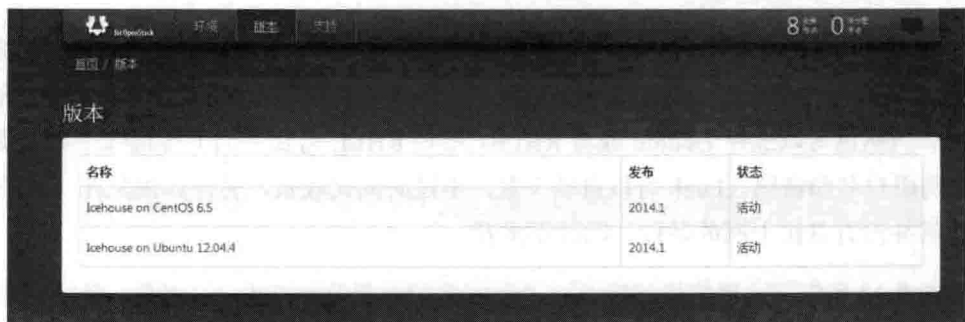


图 9-17 Mirantis Fuel 支持的版本

如图 9-18 所示，Fuel 在 console 里还有一个命令行工具，可以修改一些具体的网络参数，这些都是一个真正环境必须面对和了解的配置参数，对 OpenStack 环境的运行起着非常重要的作用。

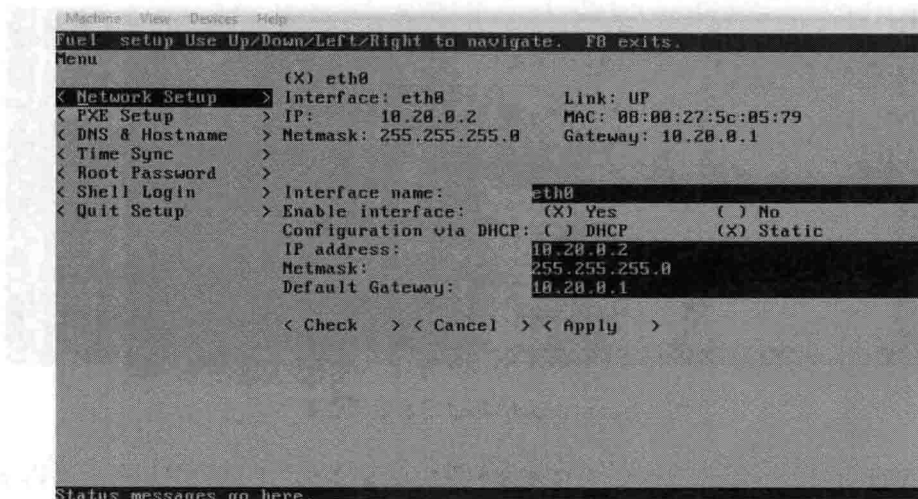


图 9-18 Mirantis Fuel 部署配置界面

接下来就开始安装 OpenStack 环境了。

定义一个 OpenStack 部署环境从命名开始，如图 9-19 所示。同样的环境可以创建多个，可见 Fuel 可以同时管理多个 OpenStack 环境。可供选择的操作系统有三种，这里默认选择 CentOS，当然也可以选择 Ubuntu 或者 RHLE，不过 RHLE 需要手动上传镜像或者提供 Red Hat 官网用户名和密码。Fuel 可以自动下载，不过时间比较长，另外如果是在真正的企业里，通常并没有直接上网的条件，因此不推荐。

如图 9-20 所示，这里选择部署 OpenStack 多节点非 HA 模式。这也是一种贴近生产环境的部署模式。

由于我们是在虚拟机中运行虚拟机，这里选择 Hypervisor 的类型为 QEMU，如图 9-21 所示，因为我们使用的是虚拟机。这些虚拟机可以用 VirtualBox 搭建，也可以是 VMware 的虚拟机。另外，OpenStack 也可以和 VMware vCenter 集成，读者如果有 vCenter 环境，可以尝试。

在图 9-22 中，可以选择 OpenStack 的网络部署模式，我们选目前比较成熟的 Neutron VLAN。在这里可以看到，Mirantis Fuel 同时支持 Neutron GRE 和 nova-network。这也是非常强大的功能，nova-network 非常成熟，但缺乏 SDN 的支持，随着 Neutron 的成熟，逐渐不再是人们选择的焦点，而 GRE 被视为以后支持 SDN 的较好选择而受到广泛关注，但目前生产环境还是不推荐，主要是受到 L3 性能和不支持 HA 的制约。



图 9-19 Mirantis Fuel 环境定义之名称和版本界面

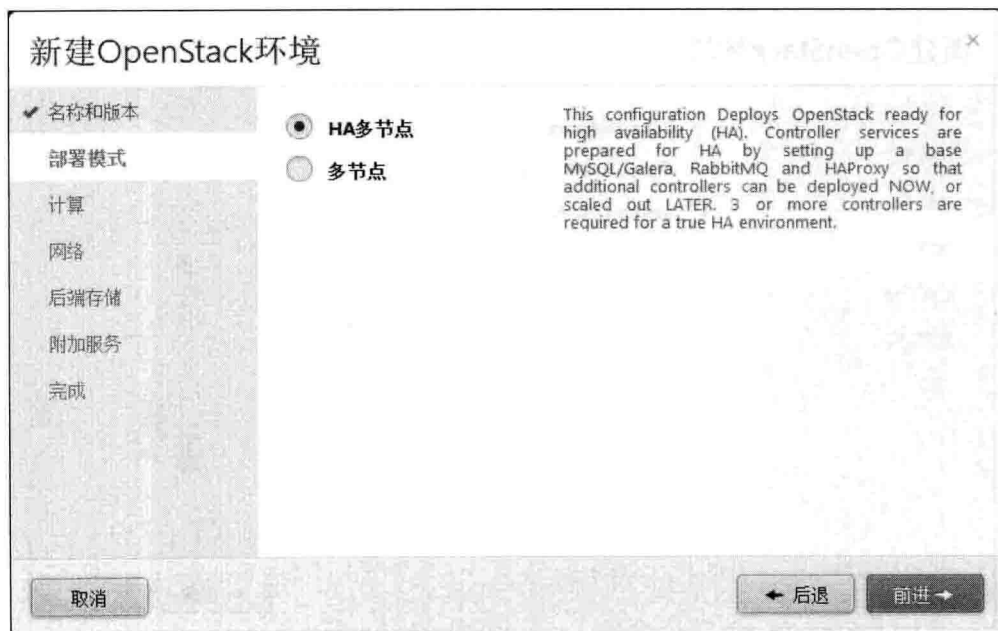


图 9-20 Mirantis Fuel 环境定义之部署模式界面



图 9-21 Mirantis Fuel 环境定义之计算界面



图 9-22 Mirantis Fuel 环境定义之网络界面

图 9-23 是选择存储后端，在如图 9-23 所示的界面里，包含了足够多的信息，而且每

种存储后端又是非常复杂的，当前流行的分布式文件系统或对象存储具体如下。

- Cinder
 - 默认的情况下，使用 LVM 卷作为存储，这也是 OpenStack 本身安装的默认方式，要求在安装时创建一个 Volume Group，并且配置到 Cinder 的配置文件里。
 - 另外，也可以选择 Ceph，最流行的 OpenStack 块存储后端。注意，Ceph 也可以提供对象存储和文件访问接口，只是目前文件访问接口通常认为不可用于生产系统。
- Glance
 - 在 HA 部署模式中，默认使用的是 Swift 对象存储。
 - 在简单多节点环境中，默认使用的是控制节点本地存储。
 - 另外还可以选择 Ceph。



图 9-23 Mirantis Fuel 环境定义之存储界面

在这里涉及 OpenStack 环境里最复杂也是最头疼的存储问题。我们可以用表 9-3 来说得更清楚一些。

表 9-3 Mirantis Fuel 创建 OpenStack 环境之存储类型

部署模式	存储类型	虚拟机非持久化存储	Cinder 块存储	Glance 镜像存储
简单多节点	本地文件系统	Yes		Yes
	LVM		Yes	
	Swift			
	Ceph		Yes	Yes
HA 模式	本地文件系统	Yes		
	Swift			Yes
	Ceph		Yes	Yes
	LVM		Yes	

因为 Ceph、Swift 都是非常复杂的存储方案，下面来看一下选择之后环境里最终有什么。

- 如果不部署 Ceph，则在简单多节点环境下，我们会使用到本地文件系统和 LVM；在 HA 部署模式下，我们会用到本地文件系统、LVM 和 Swift。
- 如果不部署 Swift，则在简单多节点环境下，我们会使用到本地文件系统、LVM 和 Ceph；在 HA 模式下，我们会使用到本地文件系统、Ceph。因为 Ceph 也提供块存储的功能，所以 LVM 可以不支持。

接下来，有一些非常好的项目，如图 9-24 所示，Mirantis Fuel 也提供了安装和部署，包括 Savanna(OpenStack 大数据方案)、Murano(在 OpenStack 里管理配置 Windows 和 Linux 里的企业级服务)，以及 Ceilometer。

最后到达完成界面，完成环境创建。

接下来，可以到准备好的节点，选择 PXE 启动，在 Mirantis Fuel 界面上可以发现那些节点。OpenStack 环境中涉及几类不同角色的节点，可以在图 9-25 的界面进行管理。Mirantis Fuel 定义了如下几类节点角色。

- **Controller:** 被分配此角色的节点将被推送 OpenStack 控制服务，并作为 OpenStack 服务的管理节点。如果选择了 HA，则至少要准备三个节点来部署。
- **Compute:** 部署了 OpenStack 计算服务的节点，可以有多个。
- **Storage – Cinder LVM:** 顾名思义，这些节点作为存储节点，并且使用的是 LVM。
- **Storage – Ceph OSD:** 这些节点作为存储节点，但存储后端是 Ceph 的 OSD。
- **Telemetry – MongoDB:** 这类节点是 Ceilometer 收集数据所用。



图 9-24 Mirantis Fuel 环境定义之服务

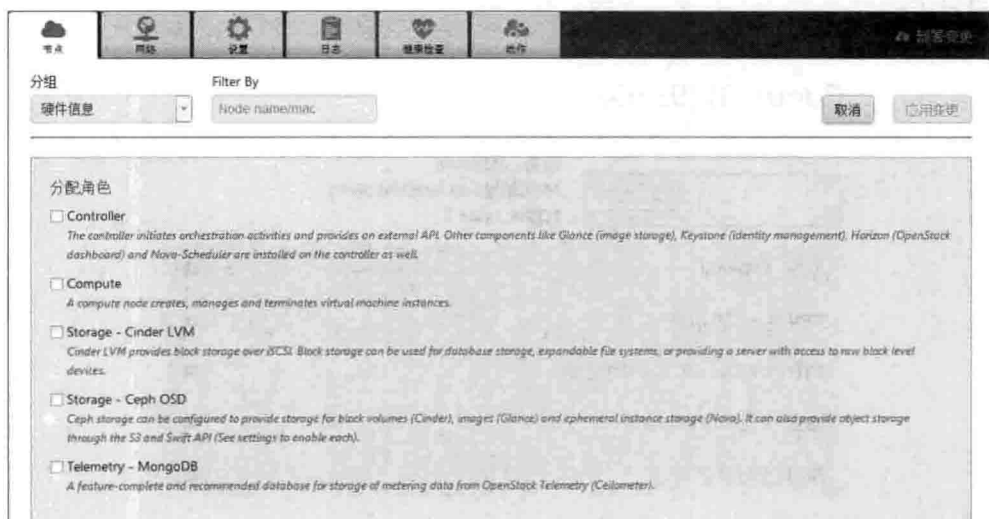


图 9-25 Mirantis Fuel 环境定义之节点角色分配

有些节点类型是可以同时赋予同一物理节点的。Mirantis Fuel 界面会控制这些，确保节点真实部署的服务不会出现冲突，比如，存储服务与控制服务一般不能部署到同一个物理节点上。

图 9-26 展示了分配角色的节点列表。

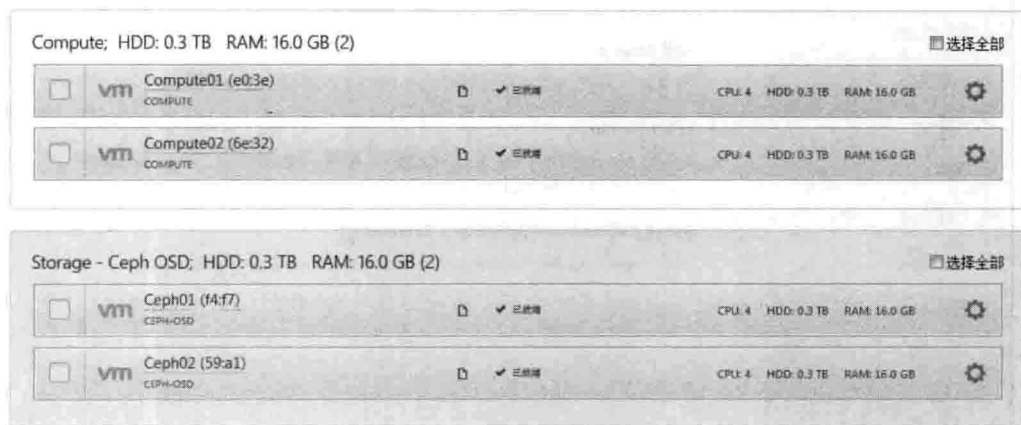


图 9-26 Mirantis Fuel 环境定义之分配了角色的节点

图 9-27 可以查看一个节点的信息，而图 9-28 用 UI 的方式展现了节点各个网口承担的网络角色（分配）。修改节点的物理网卡和 OpenStack 逻辑网络的映射关系，这里只需要拖曳就可以了。

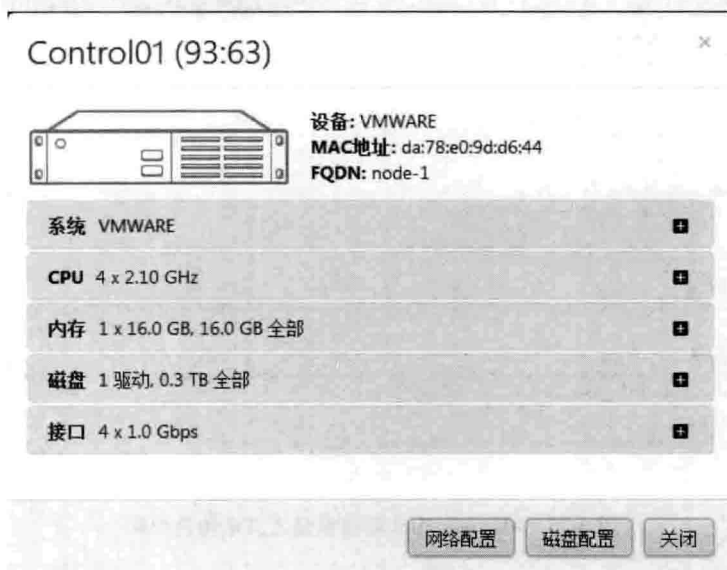


图 9-27 Mirantis Fuel 环境定义之节点信息



图 9-28 Mirantis Fuel 环境定义之节点网络分配

图 9-29 根据各个节点的角色，展现存储空间的分配。

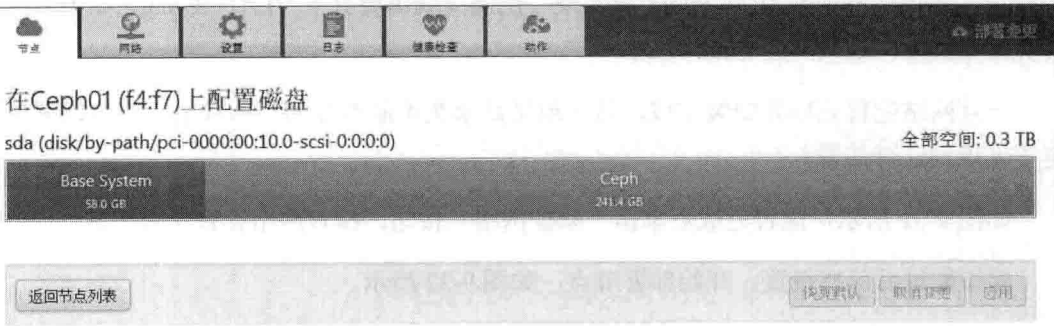


图 9-29 Mirantis Fuel 环境定义之节点存储分配

图 9-30 可以对整个环境的网络进行规划和配置。如果选择的是 VLAN，和选择 nova-network 的 FlatDHCP 输入信息稍微有所不同，在于 Neutron L2 配置需要输入 VLAN 的范围。另外，管理、存储网络都需要输入相应的 VLAN ID。如果用到 L3 的配置，需要配置浮动 IP。但是如果选择了 nova-network，在 FlatDHCP 模式下，则只需要输入每个网络的 VLAN ID，并且是否使用 VLAN 是可选的。使用 nova-network，而网络是 VLAN 时，则没有 L3 的内容。

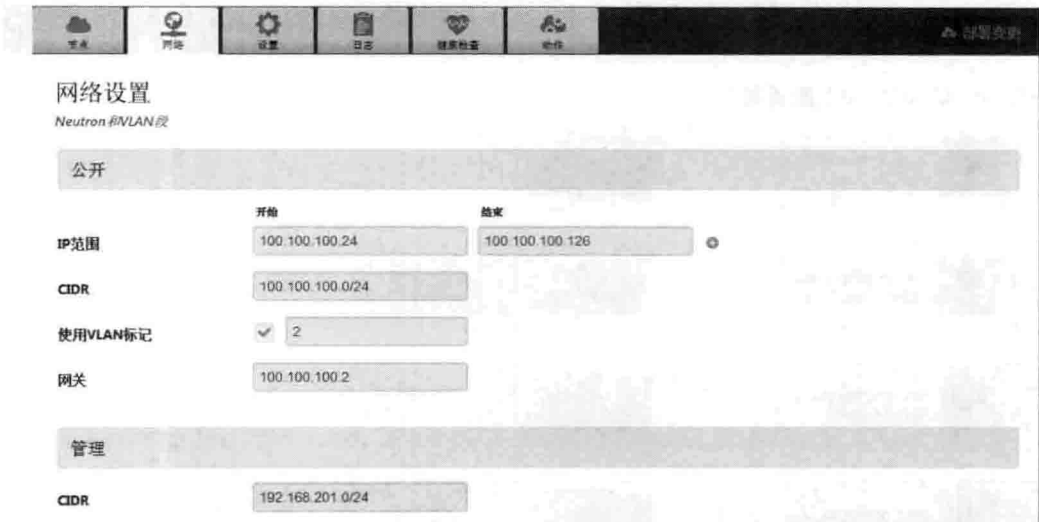


图 9-30 Mirantis Fuel 环境定义之环境网络规划

关于使用到的网络，Public IP 用于物理机器和外界通信，浮动 IP 用于动态分配给 OpenStack 虚拟机实例实现和外界通信。这里地址块不能重叠。由于 private、management 和 storage 共用同一网卡且 IP 块不同，要实现二层隔离就需要打上 VLAN 的 tag，如果是接在真实的交换机，必须启用 trunk 模式。

一旦网络配置完毕并安装完成，这个地址是永久不能改变的，所以生产环境部署前一定要先规划好再部署。

如图 9-31 所示，配置完成后单击“验证网络”按钮，检查网络设置是否正确。

验证通过后保持设置，开始部署节点，如图 9-32 所示。

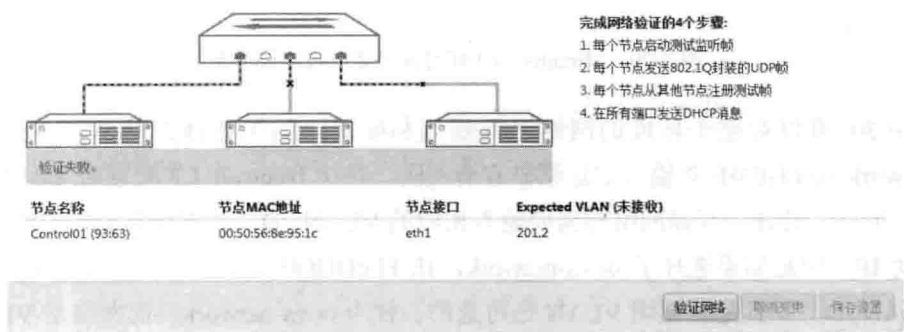


图 9-31 Mirantis Fuel 环境定义之网络验证

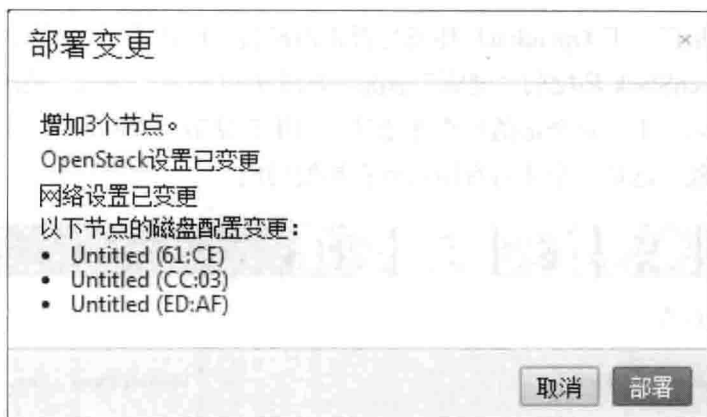


图 9-32 Mirantis Fuel 创建 OpenStack 环境之开始部署服务

此时可以发现几个虚拟机自动重启并开始安装 OS。

当需要查看安装部署日志的时候，可以到日志标签视图查看安装的日志，选取“其他节点”，再选对应节点的 Puppet log 看 log 的输出，如图 9-33 所示。

最后，一切顺利的话，大概二十几分钟就完成安装了，不过具体时间取决于机器性能，这时候点击 <http://172.16.0.2> 或者 <http://10.20.0.4> 都可以访问 OpenStack 的 Dashboard。区别在于 172.16.0.2 是公网 IP 地址，登录 Dashboard 后可以直接使用 VNC 来访问 instance，而 10.20.0.4 网段则不能。

至此，OpenStack 的环境部署完成。



图 9-33 Mirantis Fuel 创建 OpenStack 环境之部署日志输出

最后，就是验证一下 OpenStack 环境是否正确部署。Fuel 有一个非常好的功能，就是可以快速检测 OpenStack 环境的“健康”情况，如图 9-34 所示。进入“健康检查”选项卡，可以一键安全检测。不一定全部的检查都通过。如果存储节点没有安装，则 create volume 相关的服务会失败。这是一个非常有用而且有趣的功能。

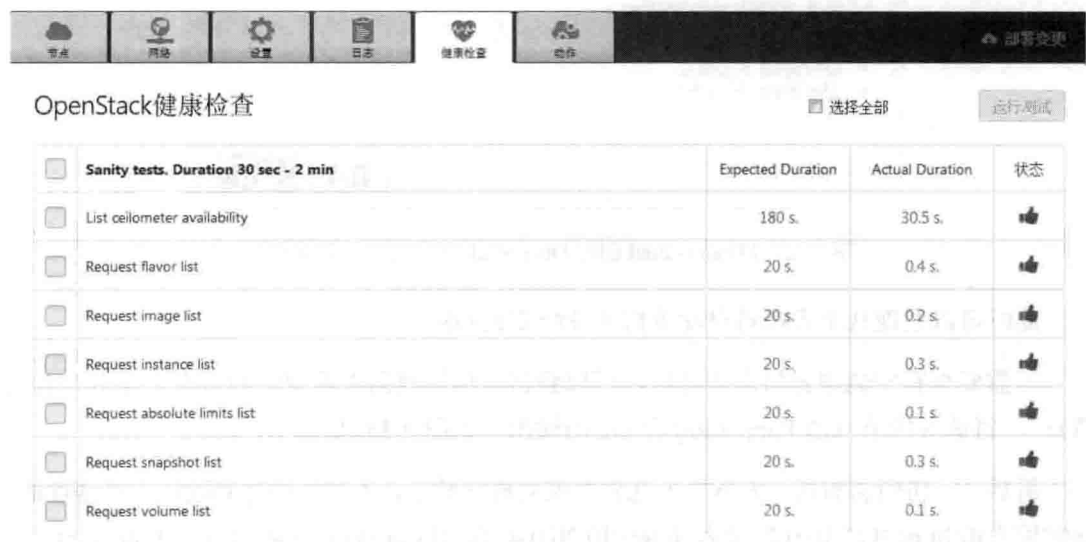


图 9-34 Mirantis Fuel 创建 OpenStack 环境之健康检查

9.5 Dell Crowbar

Crowbar (<http://sourceforge.net/projects/crowbar/>) 最开始由 Dell CloudEdge Solutions 开发，作为一个 OpenStack 的部署工具，后来开源并提交到 GitHub，是一个从裸机开始部署配置 OpenStack 的工具，目前支持部署 OpenStack 和 Hadoop 集群。从 GitHub 上可以看出，在 Havanna 发布 4 个多月之后，Crowbar 稳定版本依然是 Grizzly，最新更新时间为 2013 年 11 月 8 日，显然其社区非常不活跃。在这里介绍，是希望给读者一个扩展的思路，展现另外一种 OpenStack 的部署方式。

Crowbar 社区维基地址为 <https://github.com/crowbar/crowbar/wiki>。

Crowbar 的功能如下。

- 服务器发现。

- 固件升级。
- 通过网络启动, PXE 方式安装操作系统。
- 通过 Chef 来部署应用。

9.5.1 Crowbar 安装 OpenStack

在图 9-35 的界面, 可以配置及更新 BIOS 与 BMC 的一些信息。在 Bulk Edit 界面可以对 BIOS 和 RAID 进行设置。

通过 ISO 文件给管理节点安装操作系统很容易, 完全不需要人工干预。在管理节点上安装 Crowbar 的时候, 需要根据具体的情况对网络进行修改, 修改就集中在一个文件中, 相对容易。给集群 (相当于 OpenStack 的可用区) 添加新的节点也比较容易。

在 OpenStack 的安装方面, 需要为每个组件创建 proposal, 还需要匹配节点和角色之间的关系。



图 9-35 Dell Crowbar 管理界面

(1) 安装基本操作系统

通过 PXE 可以直接安装操作系统, 安装的操作系统是 Ubuntu。操作系统安装完毕后, 一些需要的基本服务已经配置完成, 如 Ganglia、Nagios, 另外还包括配置网络基础核心服务 (NTP、DNS、DHCP)。PXE 在探测到节点的时候, 会给节点分配一个 IP 池中的 IP。

(2) 安装 OpenStack 组件

通过 Chef 可进行组件安装, 在 Website 上, 通过为服务创建 proposal 来配置一个服务, 通过关联一个节点和一个服务角色来部署服务。在 Crowbar 中, 所有的组件都是单独部署的, 然后 Crowbar 自动地检测和集成相关的组件。而且在部署 OpenStack 的时候, 组件的部署是有顺序的: MySQL→Keystone→Nova dashboard→Glance→Nova。

9.5.2 界面

(1) Dashboard 界面

Dashboard 界面如图 9-36 所示, 可以单击节点名称来查看节点详细信息, 右上角组的功能是为了方便节点管理, 并进行逻辑上的划分。

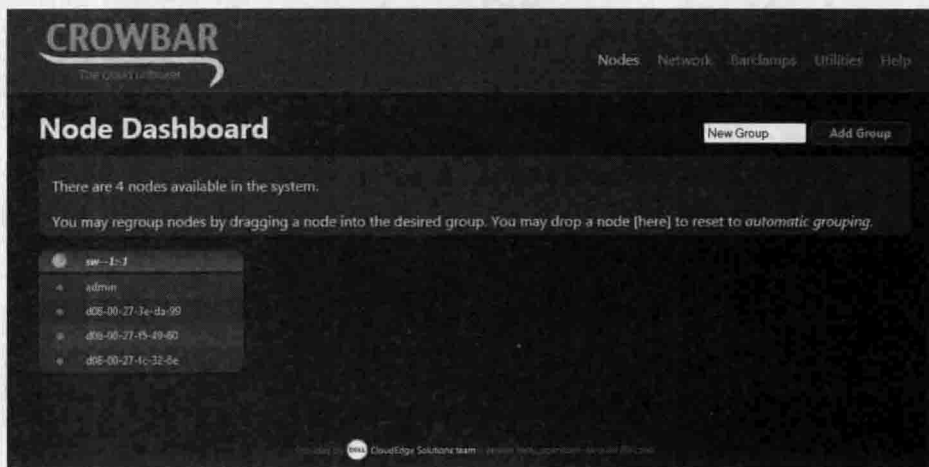


图 9-36 Dell Crowbar Dashboard 界面

(2) Bulk Edit 界面

Bulk Edit 界面如图 9-37 所示, 在安装节点的操作系统的时候, 通过这个界面可以设置

BIOS 和 RAID 信息。

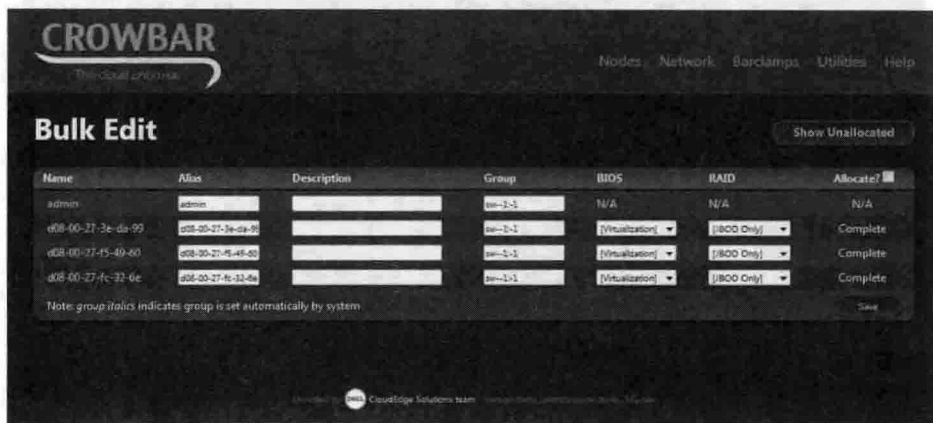


图 9-37 Bulk Edit 界面

(3) VLAN 和 Switch 界面

如图 9-38 所示, 在这个界面可以配置虚拟机交换机和 VLAN。但 OpenStack 本身可以管理虚拟机交换机, 所以实际上这个界面只是用来查看, 而不能修改什么。这个界面还是沿袭了传统交换机管理的思想, 因为对于传统交换机, 每个端口都是需要单独配置的。

如图 9-39 所示的界面可以对主机与网络的对应关系进行规划和指定。

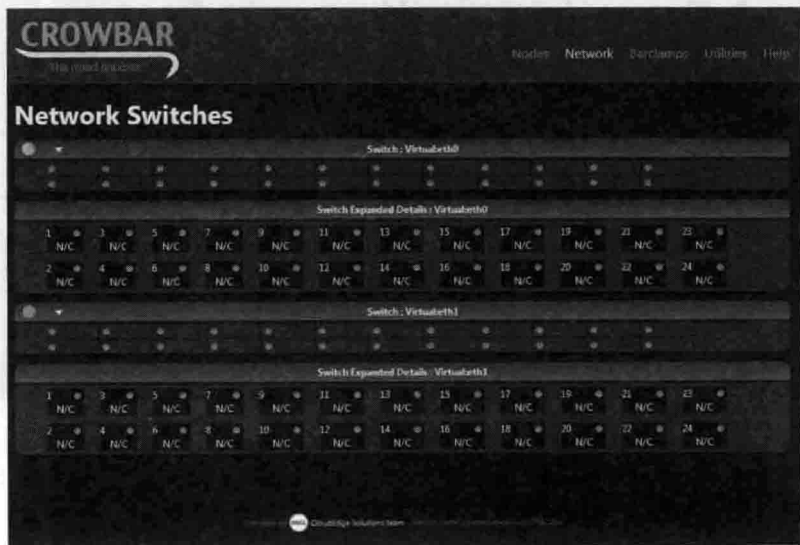


图 9-38 VLAN 和交换机配置界面



图 9-39 主机与网络的规划

(4) Barclamp 列表界面

如图 9-40 所示的界面列出了所有可用的 Barclamp 的列表，可以通过新建 proposal 来配置和启动服务。每个 Barclamp 对应了 OpenStack 环境使用到的一个组件，包括系统和服务的。

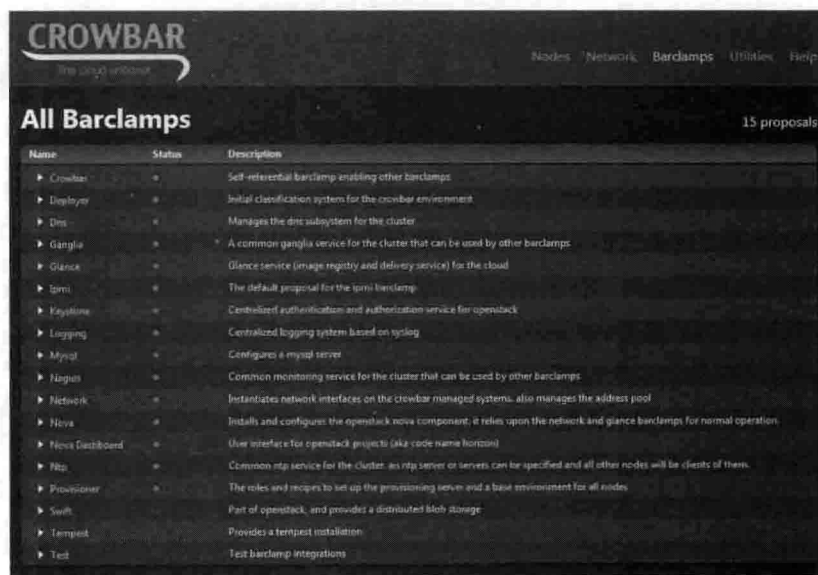


图 9-40 OpenStack 环境组件的管理

(5) OpenStack 服务的 Barclamp 列表

如图 9-41 所示是所有已经内置的 OpenStack 的组件，由于是 E 版本的，所以没有 Cinder

和 Quantum, OpenStack 的组件的启动是有顺序的。

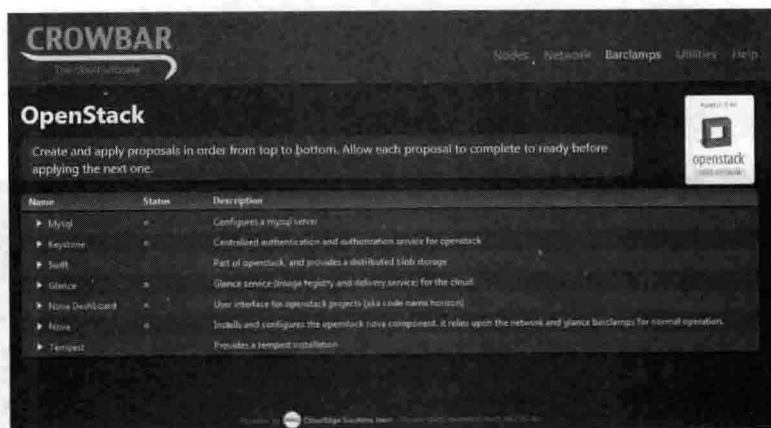


图 9-41 OpenStack 组件的管理

(6) Proposal 编辑页面

图 9-42 是 Keystone 的 proposal, 可以看到 Keystone 的 proposal 需要 MySQL 的实例, 所以必须先创建 MySQL 的 proposal。界面下半部分左侧是节点列表, 右侧是服务的角色列表, 可以通过拖曳的方式匹配节点和角色, 来实现服务的部署。

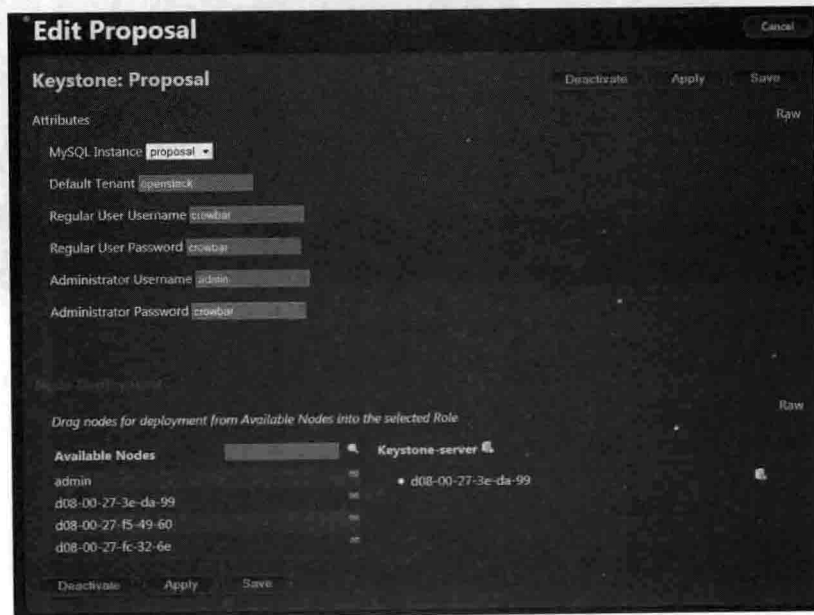


图 9-42 proposal 的编辑

(7) Barclamp 的导入/导出功能

图 9-43 可以以导入的方式新建 Barclamp。单击右上角的 All 按钮，可以显示所有的 Barclamp。

如图 9-44 所示的界面，可以导出 Log 和 Chef 配置。



图 9-43 Barclamp 导入

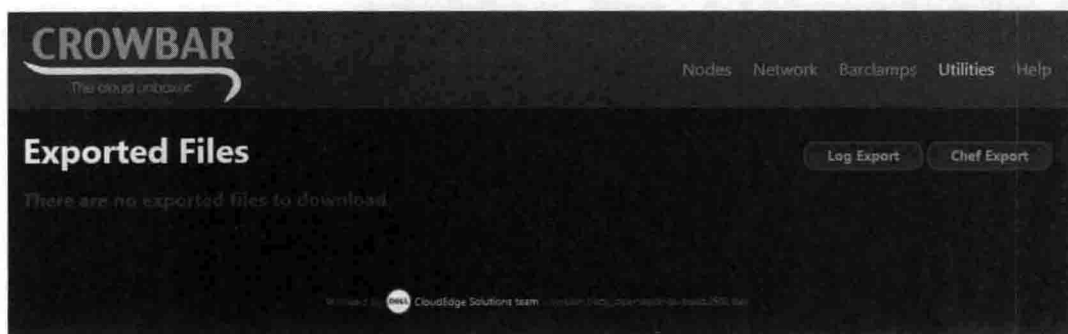
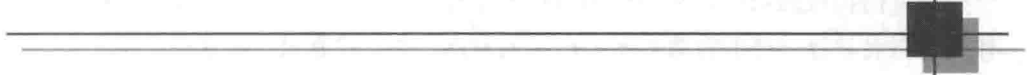


图 9-44 Barclamp 导出

第 10 章



九州云 Animbus 融合架构 一体机解决方案



在国内 OpenStack 领域，九州云（99cloud）是最早的专业提供 OpenStack 服务的公司之一，其主导推动的 TryStack 社区为广大 OpenStack 初学者提供了很好的 OpenStack 学习平台，在国内有一定的知名度和影响力。九州云也是国内在社区代码的贡献度比较靠前的公司之一，它基于 OpenStack 研发的产品有某些独树一帜的地方，可以作为很好的学习研究对象。本章内容只是抛砖引玉，通过解构一家创新公司在基于 OpenStack 为核心的“计算存储融合架构”上的探索成果，来了解更多可以在 OpenStack 上进行创新和变革的可能性。

10.1 产品背景

随着 OpenStack 的生态环境越来越成熟，依托 OpenStack 发展的商业公司很多，商业模式也多种多样。OpenStack 代表了未来数据中心云架构的一个事实标准，它的存在打破了一些原来由传统商业厂商把持产品造成的壁垒，对传统的数据中心的形态进行了重塑，这种变革将会导致一些旧有数据中心产品的衰落，也会发展出一些新的产品形态。总的来说，这种重塑朝着两个方向延伸：向上突破和向下突破，大部分商业公司把重点放在了前者，而九州云的产品试图在后者上有所成果。

- **向上突破：**即依托 OpenStack 做更上层的优化整合，这是最为普遍的商业模式，它试图在面向用户的业务交互模式上进行改造，借此打造不同的云计算产品。以 OpenStack 打造公有云是最显而易见的商业模式，因为 OpenStack 一开始就以 AWS 为模仿对象，因此诸如 HPCloud、Rackspace，甚至是国内的一些小型公有云提供商，以 OpenStack 为基础，再自定义了前端用户界面、计量统计、订单管理和计费流程，就可以对外提供公有云服务。另外一方面，以 OpenStack 为基础打造私有云也是一条向上突破的商业模式，在这一领域主要追赶模仿 VMware，试图通过 KVM+OpenStack 模式替换（或者整合）既有的 ESXi+vCloud 组合模式，同时提供比 VMware 产品更为丰富的 API 接口和二次定制化开发可能性来吸引客户，这一领域主要有专注 OpenStack 发行版的国际巨头 Red Hat、Mirantis，也包括一些如 Piston 等的积极跟进的 OpenStack 专业服务提供商。
- **向下突破：**即依托 OpenStack 对更底层硬件进行重塑，通过更为深入的数据中心基础底层架构改造，以此为客户带来价值。向下突破的主要驱动力，来自于传统数据中心管理者对底层基础架构变革的渴望，互联网公司如 Google、Facebook 对数据中

心的改造,让数据中心管理者看到基础架构在管理简化、横向扩展和成本控制上的可能性。“计算存储融合架构”是这种数据中心底层变革的手段之一,通过虚拟化技术和分布式文件存储的配合,在通用 X86 服务器群集上搭建计算资源池和存储资源池,节点之间通过万兆网络连接,整个群集通过技术手段保证可用性,任何单一节点的故障,不会影响上层计算资源和存储资源的使用。群集内的资源调度具有一定的自我调节能力,实现整体的资源负载均衡。通过分布式的设计,群集可以实现线性的横向扩展能力,结合自动化安装,让群集横向扩容(Scale-Out)变得非常容易。

如图 10-1 所示,传统和现代的数据中心,以及应用方式发生巨大的从概念层次的变革。云应用将会以“牲口(Cattle)”方式扩展。由此衍生出了“计算存储融合架构”,以及“计算存储分离架构”两种区别。而“计算存储融合架构”这一领域已经吸引越来越多的关注,新兴公司 Nutanix 通过虚拟化技术和分布式存储技术,在商业领域取得了巨大的成功。VMware 等传统虚拟化厂商也积极开发出 vSAN 积极应对(将本地服务器磁盘聚合到共享存储集群中),同时有信号显示 VMware 有意愿将自身业务拓展至一体机硬件领域。在本章内容中,我们则会介绍九州云如何依托 OpenStack 以及其他开源产品,尝试打造基于开源技术的“计算存储融合架构”。

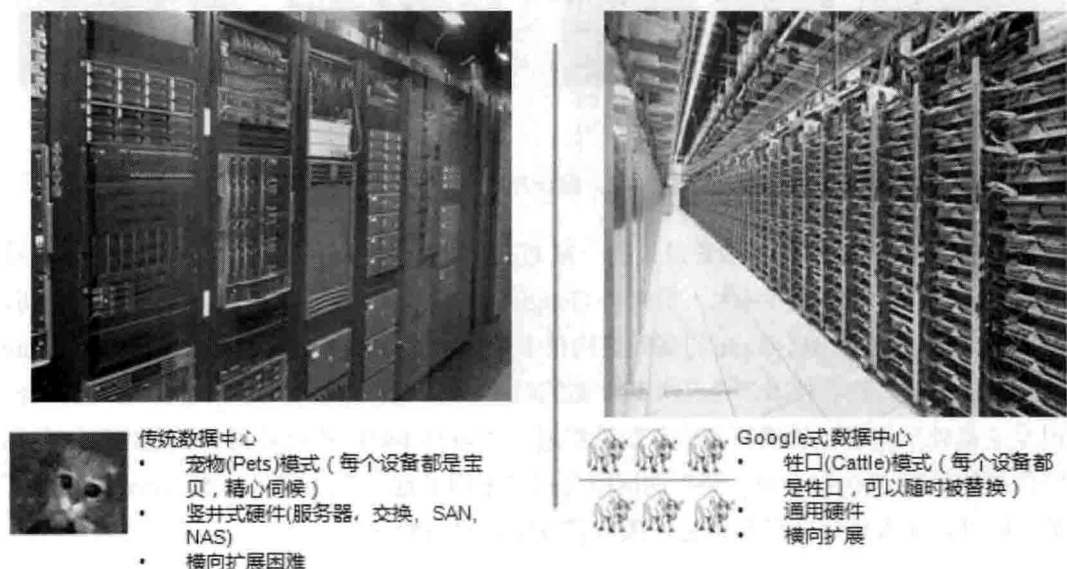


图 10-1 传统与云应用数据中心和应用方式的区别

回顾“融合架构”的发展，也经历了不同阶段。最初“融合架构”技术来自一些传统硬件进行组合，以一体机机柜的形态出现在市场上，这些融合架构在形态上简化了设计，但深层次的技术整合不够，很多产品“新瓶装旧酒”，只在不同产品模块整合上做了一定的优化，未带来实质性的突破和变革。Nutanix 做了非常大的改进，九州云则试图通过 OpenStack 的开源框架做进一步的突破，如图 10-2 所示。

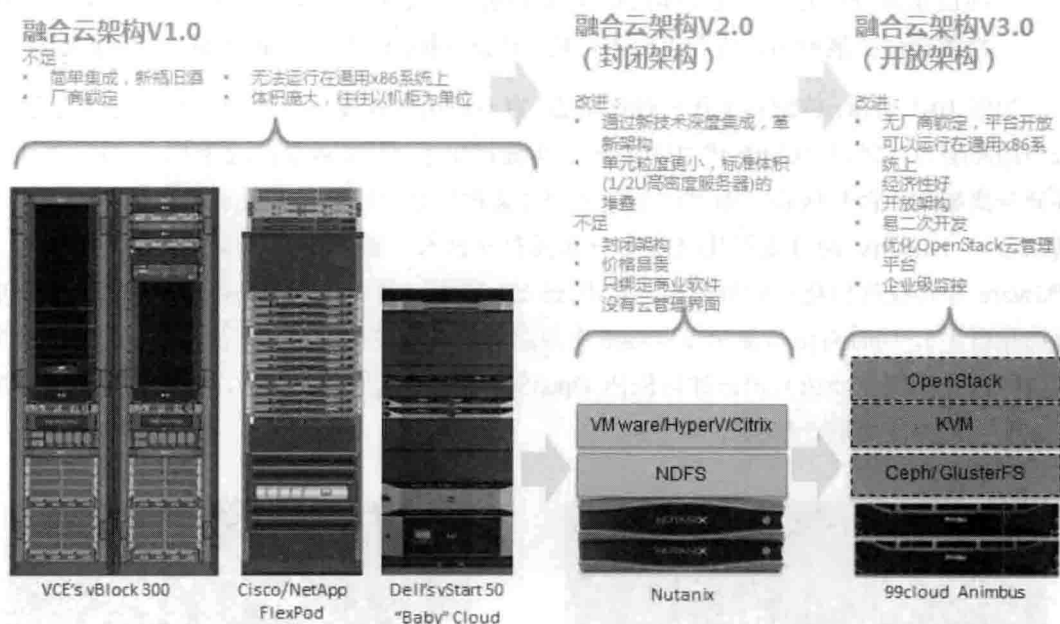


图 10-2 融合云架构演进

Nutanix 是过去十年硅谷增长最快的一家 IT 基础设施初创公司，Nutanix 拥有深厚的技术基因，这家硅谷新贵很多技术人员来自 Google、Facebook 等互联网公司，它在创始之初，提出的口号便是“把类似 Google 的基础架构技术带给企业 Bringing Google-like Infrastructure to Enterprises”，其核心技术便是计算和存储的融合技术，管理简单，结合商业虚拟化平台，实现企业基础架构资源的极简管理和线性扩展，短短几年时间内就获得了长足发展，包括一些传统的世界 500 强企业、新兴的网络公司都采用了这一产品。然而 Nutanix 也存在一定的局限性，主要体现在厂商锁定、成本等方面，罗列如下。

- 封闭架构，厂商锁定（必须用 Nutanix 的硬件）。
- 价格昂贵。
- 只绑定商业虚拟化软件如 VMware/Xenserver 等。

- 只有虚拟化管理界面，没有云管理界面。

OpenStack 的出现，也更重要的是以 OpenStack 为核心的其他开源生态系统(Ecosystem)里诸如分布式存储技术、OpenStack 存储插件的日渐成熟，使得“计算存储融合架构”得以再往前跨越成为可能。九州云选择了这一方向持续努力，试图依托 OpenStack，对融合架构做进一步的改进，产品在企业自建私有云、托管私有云等应用场景使用，取得了积极的市场反馈。

10.2 九州云计算存储云一体机

九州云这一款产品被称为“计算存储云一体机”，设计目标是“基于 OpenStack 和相关开源技术，打造适合企业级使用的类似 Google 的智能计算/存储融合架构云平台，简化数据中心管理，降低数据中心成本支出”，出于这样的目标，其产品在设计之初，便遵循以下几大原则。

- **通用原则：**产品专注在软件层面的实现，对底层服务器和网络设备无特殊要求。可以运行在通用的 X86 服务器上，无厂商锁定和限制。可以运行在通用的万兆交换机网络中，对硬件交换设备无特殊要求，通过软件控制整体网络行为。
- **无中心原则：**系统任何节点失效都不影响整体使用，主要体现在如下两点。
 - 任何节点都可以成为（或接管）控制器节点。
 - 任何节点都可以成为“存储+计算”融合节点。
- **群氓智能：**整体平台体现高度的智能调度和运维自动化，包括如下三点。
 - 任何节点出现问题，虚拟机都可以自动修复错误，保证虚拟机运行。
 - 新增节点自动安装，加入即成为资源池的一部分，接受系统调度。
 - 支持自动资源调度，根据策略实现优化负载，实现整体资源使用均衡。
- **线性横向扩展：**随着设备的堆叠和扩容，计算资源池和存储资源池能够实现横向线性（或近似线性的）扩展，如图 10-3 所示。
- **业务可用性：**内置整体云平台管理界面，无须额外采购，即可拥有用户交互界面，实现开机即可用。

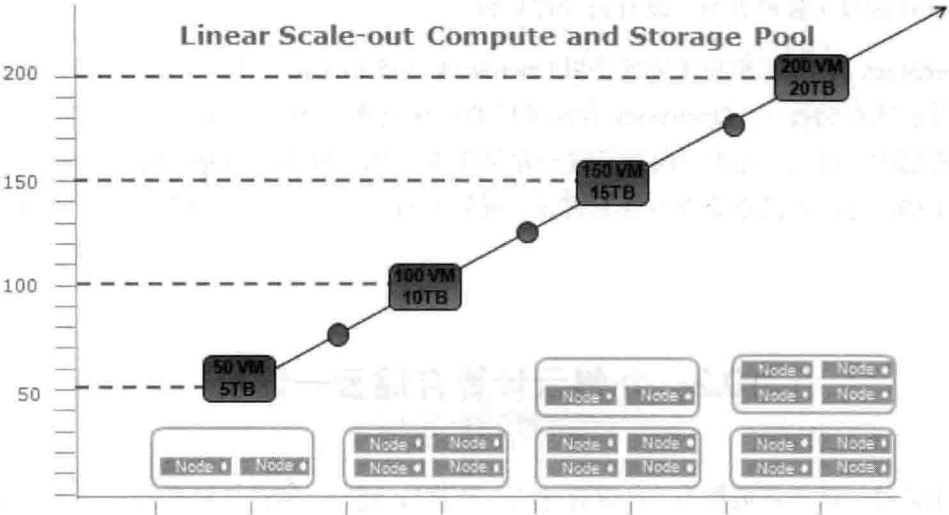


图 10-3 “存储融合架构”的线性横向扩展

虽然在形态上九州云的 Animbus 一体机和 Nutanix 类似，但深入研究下去，抛开简单的商业软件和开源实现上的差异外，还有许多的不同点，如表 10-1 所示。

表 10-1 Nutanix 与九州云一体机对比

	Nutanix 一体机	九州云 Animbus 一体机
虚拟化层	VMware/Xenserver/HyperV（成本高）	KVM（成本低）
分布式存储	NDFS	GlusterFS 或者 Ceph
云平台管理	无	OpenStack
业务前端界面	无	Animbus 私有云 / 公有云界面
产品形态	软/硬一体（硬件锁定）	纯粹软件方案（无硬件锁定）
对服务器是否有特殊要求	有，必须运行在 Nutanix 服务器上	无，可以运行在通用 X86 服务器上

首先九州云 Animbus 云一体机把重点放在了软件解决方案的实现上，而力求底层服务器的通用性。其次以 OpenStack 为核心，通过一系列开源软件的整合，力求在成本上更加有优势。同时，在 OpenStack 本身架构做了稳定性和扩展性上的改良，也对上层和底层都做了一些深入的优化，如图 10-4 所示。



图 10-4 九州云“计算存储融合”的优化

这里我们重点需要介绍的是九州云对底层的优化，即前文所述的“向下突破”。由于 OpenStack 的设计初衷并没有对底层基础架构有过多关注，其设计目标也并没有把兼容融合架构作为主要方向，因此一开始九州云就根据未来产品形态做了深入探讨和研究，定义了以下设计目标。

- 计算管理 (KVM) 和存储管理 (GlusterFS/Ceph) 能和上层 OpenStack 更紧密集成。
- 计算管理 (KVM) 和存储管理 (GlusterFS/Ceph) 在一体机上能相互隔离，协同工作。
- 计算管理 (KVM) 和存储管理 (GlusterFS/Ceph) 在一体机上性能达到最优。
- 计算管理 (KVM) 和存储管理 (GlusterFS/Ceph) 在整个资源池中能够被智能调度。

针对以上设计目标，九州云对底层主要实现了以下技术改进。

- **融合架构下计算管理 (KVM) 和分布式存储管理 (GlusterFS/Ceph) 资源隔离：**计算和存储之间由于在同一物理硬件上，因此对管理节点本身的 CPU/Memory 资源的使用需要做相应的限制和隔离，以免资源冲突造成整体平台性能的下降。隔离方式

可以选择 cgroup、KVM 虚拟机和 Docker。经过对比和测试，最后选择了将分布式存储封装在虚拟机里面的方式，实现更稳定的存储资源和计算资源隔离。

- **融合架构下的高可用：**存储级别的高可用通过分布式存储中设置多副本方式实现；计算资源即虚拟机的高可用，是在 KVM 基础上，结合 Nova Scheduler 的优化，实现多节点的虚拟机在故障发生时在宿主机节点之前的切换；OpenStack 控制节点做了更多的改进，每个节点上都预安装了控制节点，后端数据库放在分布式存储上面，保证了高可用，同时在某一控制节点发生故障时，可以随时启用任意节点上的控制器增加新的冗余节点，如图 10-5 所示。

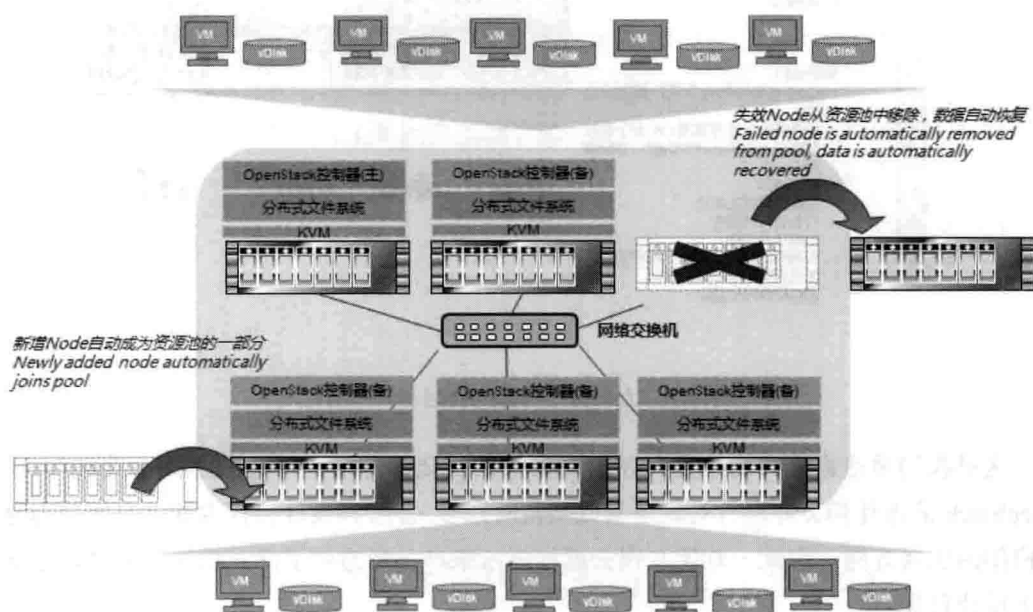


图 10-5 九州云高可用架构

- **分布式存储监控：**一体机在硬件上实现了计算存储融合，因此在监控上也需要进行更深入的集成。九州云实现了一体机的监控界面，即能够监控计算资源池状态，也能获取存储资源池信息，如图 10-6 所示。



图 10-6 九州云分布式存储监控

- **存储读写性能优化:** 对底层分布式存储做了一些优化, 包括虚拟化层和底层分布式文件系统的通信优化, SSD 的集成方式优化和不同大小文件的读写优化。
- **融合架构组件部署自动化:** 实现了自动化部署, 支持快速扩容。
- **通用 X86 平台最佳实践建议:** 根据测试和实际使用结果, 在底层 X86 服务器硬盘搭配、阵列卡选择和网卡配置等方面给出了最佳配置建议。

除了底层的优化之外, 九州云也对中层 OpenStack 本身和上层面向用户的交互方式做了优化。其中对中层 OpenStack 本身的优化目标, 主要集中在基于最新 OpenStack 版本提供一个更适合企业使用的稳定版本, 改进包括如下几点。

- **Nova Scheduler 优化:** 在“计算存储融合一体”的架构中, 如何选择合适的节点进行负载分摊变得更加重要, 九州云对 OpenStack 调度上做了一定的优化, 使得资源分配更加合理。
- **VM 高可用:** 实现了虚拟机的高可用, 能够在物理宿主机出现故障时进行切换。
- **自动化安装:** 一键式根据拓扑的快速安装。
- **OpenStack 服务的高可用:** 实现了 Nova、Glance、Cinder、Keystone、Neutron 等服务的高可用。
- **补丁修复:** 修复了 OpenStack 版本中的一些缺陷。

对 OpenStack 上层的优化目标, 是让 OpenStack 提供的资源池更好地服务于业务需要, 主要的改进包括如下几点。

- **界面改进：**云一体解决方案内置了一个经过优化的云管理界面，这一界面可以和 Horizon 界面共存，用户可以根据需求，在原生 OpenStack 界面或是九州云经过优化后的 Animbus 云平台界面做一个选择，如图 10-7 所示。

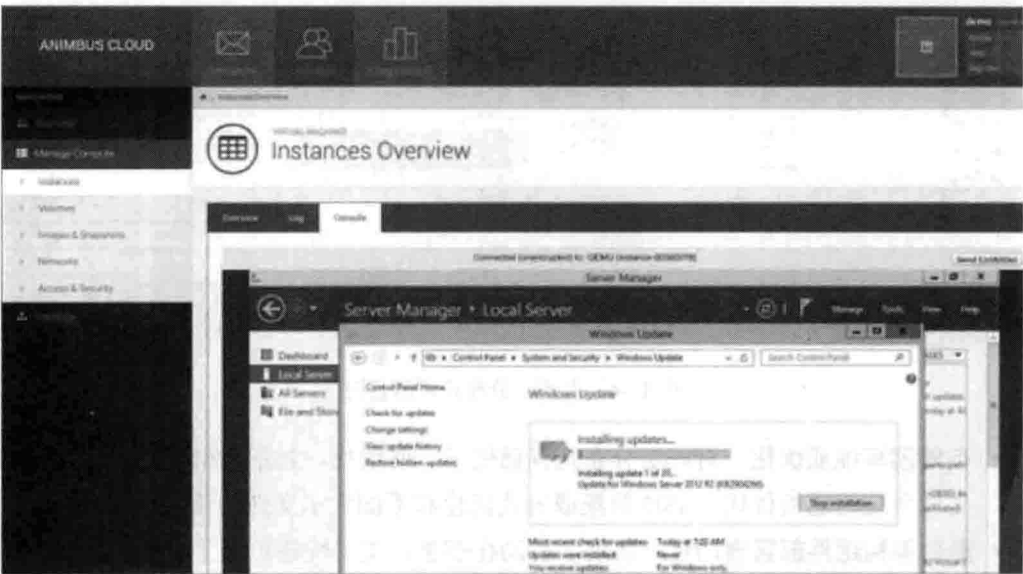


图 10-7 九州云界面

- **流程集成：**定制化的界面包含了流程模块，可以实现资源申请的审批，如图 10-8、图 10-9 所示。



图 10-8 九州云流程管理



图 10-9 九州云审批管理

- **计量数据报表统计：**通过整合 Ceilometer 数据，对数据进行视图展示，比如加入了如“成本中心”等业务字段，可以实现按照成本中心进行统计。
- **监控集成：**OpenStack 原生平台本身监控能力上相对薄弱，一般部署的时候都会借助第三方监控平台诸如 Nagios、Zabbix 进行监控。这样的不足使管理员需要通过登录不同的平台才能看到监控信息。九州云的云一体机加强了 OpenStack 本身的监控能力，集成了最基本的监控数据。
- **AD 集成：**支持通过 AD 进行用户验证。

总体来说，九州云基于 OpenStack 做了不少大胆的尝试，在一些具体细节上做了很多的改进，让 OpenStack 在更多企业的落地更加简单容易。他们向下突破方面做了相当多的工作，以 OpenStack 为核心，挖掘基础架构的改造可能性，推动数据中心底层形态上的变革，这些探索代表了未来数据中心演变的一个重要方向，传统数据中心能否借助“计算存储融合架构”，朝着架构简单、管理高效和线性横向扩展的架构往前推进一步，我们将拭目以待。



view
VIEW.COM.CN

博文视点 · IT 出版旗舰品牌

博文视点诚邀精锐作者加盟

十载耕耘奠定专业地位

以书为证彰显卓越品质

《C++Primer (中文版) (第5版)》、《淘宝技术这十年》、《代码大全》、《Windows内核情景分析》、《加密与解密》、《编程之美》、《VC++深入详解》、《SEO实战密码》、《PPT演义》……

“圣经”级图书光耀夺目,被无数读者朋友奉为案头手册传世经典。

潘爱民、毛德操、张亚勤、张宏江、咎辉Zac、李刚、曹江华……

“明星”级作者济济一堂,他们的名字熠熠生辉,与IT业的蓬勃发展紧密相连。

十年的开拓、探索和励精图治,成就博古通今、文圆质方、视角独特、点石成金的计算机图书的风向标杆:博文视点。

“凤翱翔于千仞兮,非梧不栖”,博文视点欢迎更多才华横溢、锐意创新的作者朋友加盟,与大师并列于IT专业出版之巅。

英雄帖

江湖风云起,代有才人出。

IT界群雄并起,逐鹿中原。

博文视点诚邀天下技术英豪加入,

指点江山,激扬文字

传播信息技术,分享IT心得

专业的作者服务

博文视点自成立以来一直专注于IT专业技术图书的出版,拥有丰富的与技术图书作者合作的经验,并参照IT技术图书的特点,打造了一支高效运转、富有服务意识的编辑出版团队。我们始终坚持:

善待作者——我们会把出版流程整理得清晰简明,为作者提供优厚的稿酬服务,解除作者的顾虑,安心写作,展现出最好的作品。

尊重作者——我们尊重每一位作者的技术实力和生活习惯,并会参照作者实际的工作、生活节奏,量身制定写作计划,确保合作顺利进行。

提升作者——我们打造精品图书,更要打造知名作者。博文视点致力于通过图书提升作者的个人品牌和技术影响力,为作者的事业开拓带来更多的机会。



联系我们

博文视点官网: <http://www.broadview.com.cn>

CSDN官方博客: <http://blog.csdn.net/broadview2006/>

投稿电话: 010-51260888 88254368

投稿邮箱: jsj@phei.com.cn



@博文视点Broadview



微信公众账号 博文视点Broadview



内容简介

本书结合作者亲身经历的各类OpenStack的咨询、规划和实施经验,以循序渐进的方式,从理论和工程角度,讲述了如何将OpenStack(本质上只是一堆相关的进程和服务)变成企业可运营的、托管企业各类生产环境的云平台的方方面面,让OpenStack真正变成我们身边默默无闻但又实实在在的环境的一分子。本书分为10章,分别介绍了OpenStack与云,OpenStack社区,OpenStack与AWS、VMware、虚拟化管理工具,虚拟机管理程序与典型应用,OpenStack架构与组件,OpenStack部分组件安装示例,系统定制技术,OpenStack部署,第三方工具搭建OpenStack运行环境,九州云Animbus融合架构一体机解决方案等内容。

本书面向广大OpenStack工程人员、技术专家、IT管理人员、云计算架构师等。对于初学者也有很强的参考意义。

作者简介

张小斌,拥有5年丰富的计算机软件设计、开发和管理经验,分别于西安交通大学和中科院计算所完成本科和硕士研究生学业,现在是苏宁北京研发中心云计算研发部负责人,主要著作有《黑客分析与防范技术》和《计算机网络安全工具》(国内最早的网络安全书籍)。

他曾在朗讯贝尔实验室和硅谷Terawave等公司工作多年;在HP担任解决方案架构师;在赛门铁克任主任工程师,研发存储备份软件,曾参与公司全球“Cutting Edge”技术大会并做技术报告;在北电网络、Websense、TrustGo分别担任技术经理、研发经理和研发总监职位,曾负责邮件安全、移动安全、移动互联网搜索引擎等的研发管理工作;在VMware和IBM的云计算部门负责云计算产品的架构设计和解决方案等工作。

他现在专注于云计算的研究探索,即关注社区,更注重OpenStack在企业的实践和落地。在企业级计算、网络和存储的整体解决方案、高可用性、安全以及应用管理自动化和弹性等领域都有所涉猎。作为国内早期的OpenStacker,于IBM工作期间研发基于OpenStack的企业私有云与数据中心解决方案,产品在一年中先后完成了政府机构、运营商、高校、超大型保险公司、著名网游等公司的10多起部署案例,并在公司内部进行了JVDI、SED等多个合作项目。所领导项目获得大中国区创新大赛Top12,获得多个美国专利,并作为OpenStack香港峰会的展示项目。另外,在苏宁工作期间,负责OpenStack在苏宁落地的研发、设计、规划、项目管理、迁移等,并带领团队搭建大型数据中心基础架构云等。在OpenStack巴黎峰会也有技术方案录取而获邀成为发言者。



博文视点Broadview



@博文视点Broadview

上架建议:云计算

ISBN 978-7-121-24690-6



定价: 69.00元



策划编辑: 张春雨 付 睿
责任编辑: 付 睿
封面设计: 李 玲