

深入理解OPNFV

利用OPNFV加速NFV转型

Amar Kapadia

Nicholas Chase



Pure Play Open Cloud

深入理解 OPNFV

利用OPNFV加速NFV转型

**Amar Kapadia
Nicholas Chase**



深入理解 OPNFV

Copyright © 2017 Mirantis, Inc.

ISBN: 978-1975678449

All rights reserved. No part of the book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Mirantis, Inc. and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Mirantis, Inc. has endeavored to provide trademark information about all the companies and products mentioned in this book by the appropriate use of appropriate marks. However, Mirantis, Inc. cannot guarantee the accuracy of this information.

First publication: August 2017

Published by Mirantis Inc.
525 Almanor Avenue
Sunnyvale, CA 94085, U.S.A.
www.mirantis.com

英文原版由Mirantis Inc.出版2017。
此翻译版得到OPNFV和Mirantis Inc.的许可

英文版致谢

作者

Amar Kapadia

Nicholas Chase

审阅

Bryan Sullivan

Brandon Wick

Christopher Price

Fatih Degirmenci

Frank Brockners

Gregory Elkinbard

Heather Kirksey

Kathy Caciato

Marc Cohn

Morgan Richomme

Raymond Paik

Serg Melikyan

Steve Vandris

Tapio Tallgren

Yongsheng Gong

受访

Christopher Price

Dusty Robbins

Madhu Kashyap

Prakash Ramachandran

Susan James

封面设计

David Stoltenberg

插图

Amar Kapadia

编辑&校对

Nicholas Chase

中文版致谢

翻译

章宇军（封面、封底、前言）
张军（第一章、第二章）
冯晓伟（第三章、第八章）
尚小冬（第四章、第九章）
董文娟（第五章）
吴之惠（第六章）
杨阳（第七章）
钟军（第十章、第十一章）

编辑&校对

章宇军

审阅

迟刚
高亮
张军
章宇军
周楠

本书中文版翻译由OPNFV社区协作完成

关于作者

Amar Kapadia是一名为**Mirantis**提供咨询服务的**NFVI**专家，，当前他在自己的初创公司**Aarna**网络担任**NFV**战略编排师（戏称）。在此之前，他曾在**Mirantis**担任产品市场高级总监。在**Mirantis**之前，他是**EVault**长期存储服务的高级总监，该公司提供基于**OpenStack Swift**的公有云存储服务。他在**Emulex**、**Philips**和**HP**积累了20多年的数据存储、服务器和网络方面的技术经验。**Amar**当前主要关注**NFV**、**unikernels**、容器、**serverless**调度、**OPNFV**、**ONAP**和**CORD**等领域。他拥有加州大学伯克利分校的电子工程硕士学位。他的博客地址是
<http://aarnanetworks.com/blog>，你也可以关注他的**twitter**账号 @akapadia_usa。

我由衷感谢**Mirantis**的**Boris Renski**和**Intel**的**Adarsh Sogal**资助本书。特别感谢**Linux**基金会的**Brandon Wick**，没有他的鼓励和社区推广，就不会有这本书。最后还想感谢我的家人对我深夜及周末撰写本书的宽容。

Nichlos Chase是**Mirantis**技术与市场内容负责人。在**Mirantis**之前，他在**IBM DeveloperWorks**担任大师级作者，在一家交互开发公司担任首席技术官，曾经做过**Oracle Instructor**，软件开发者，社交书签网站**NoTooMi.com**的创始人，数十本科技书籍的作者或共同作者，原克利夫兰**Freenet**的系统管理员，某个采用网页浏览器更新数据库的早期应用的推动者。他跟他的夫人**Sarah Jane**一起居住在一个农场，有27只鸡、12只鹅、7只鸭子、2只狗和宠物、3条鱼，2只长尾鹦鹉和刺猬**Hazel**。他拥有**Case Western Reserve**大学和**John Carroll**大学的物理学和教育学学士学位，当前致力于云计算、物联网、机器学习和人工智能。他的博客地址是
<http://www.nicholaschase.com>，你也可以关注他的**Twitter**账号 @nickchase

我想感谢我的夫人兼项目经理**Sarah Jane**, 她帮助我把工作组得井井有条, 令我有可能完成各项任务 (包括此书)。我也要感谢**OPNFV**项目坚信此书背后的理念、提供各方面的技术支持和理解这个疯狂的日程。我最需要感谢的是**Amar Kapadia**的加入。没有你的努力工作也就没有此书。

目录

| | |
|------------------|----|
| 内容范围 | 10 |
| 目标人群 | 12 |
| 注册 | 12 |
| | |
| 1. NFV是什么？ | 15 |
| NFV业务驱动力 | 15 |
| NFV是什么？ | 16 |
| NFV收益 | 19 |
| NFV用例 | 21 |
| NFV在ETST进展 | 22 |
| NFV需求 | 24 |
| 下一步计划 | 27 |
| | |
| 2. NFV转型：CLI去哪了？ | 29 |
| NFV驱动技术变革 | 30 |
| 对组织的影响 | 38 |
| 开始旅程 | 44 |
| | |
| 3. 这是OPNFV！ | 47 |
| OPNFV驱动因素 | 47 |
| 什么是OPNFV？ | 48 |
| 开源系统集成 | 50 |
| 开源的益处 | 51 |
| 多途径通往NFV | 52 |

| | |
|---|------------|
| 再一次，为什么关心？ <i>{Create-Compose-Deploy-Test}.Iterate</i> | 53 |
| 4. OPNFV中的上游项目 | 55 |
| NFV管理和编排(MANO) | 56 |
| 虚拟化的基础设施管理器 (VIM) | 60 |
| 软件定义网络 (SDN) 控制器 | 64 |
| NFVI计算 | 67 |
| NFVI存储 | 70 |
| NFVI虚拟交换 | 70 |
| NFVI硬件 | 72 |
| 范围之外 | 72 |
| 5. OPNFV上游贡献 | 75 |
| 项目详细说明 | 76 |
| 项目分析 | 81 |
| 成功案例： <i>OPNFV Doctor</i> | 82 |
| 6. OPNFV持续集成 | 89 |
| <i>OPNFV RelEng</i> : 发布工程 | 89 |
| <i>OPNFV Pharos</i> ：社区实验室基础设施 | 93 |
| <i>OPNFV Octopus</i> ：持续集成项目 | 95 |
| 7. OPNFV自动化软件部署 | 99 |
| 什么是 <i>OPNFV</i> 场景？ | 99 |
| 安装工具 | 103 |
| 其他软件部署项目 | 107 |
| 8. OPNFV持续测试 | 109 |
| 测试用例分类 | 110 |
| 测试项目 | 112 |
| 看板 | 115 |
| Plugfests | 117 |
| 9. 为OPNFV构建VNF | 121 |

| | |
|-------------------------------|------------|
| <i>构建VNF</i> | 121 |
| <i>VNF上线运行</i> | 123 |
| <i>OPNFV中的Clearwater vIMS</i> | 125 |
| 10. 利用OPNFV驱动业务 | 131 |
| <i>OPNFV和最终用户</i> | 131 |
| <i>OPNFV和技术供应商</i> | 138 |
| <i>OPNFV和个人</i> | 138 |
| <i>参与</i> | 139 |
| 11. 其他资源 | 140 |

前言

网络功能虚拟化（**Network functions virtualization**），即**NFV**，是一个划时代的变革，将会彻底改变网络的构建和运营。这一趋势将对电信运营商和技术供应商产生深刻影响。事实上，如果没有**NFV**，我们很难想象5G的愿景能够得到实现。这一变革在非电信领域同样影响深远：公司如何将分支机构连接到中央数据中心，智慧城市如何提供全市范围内的连接，以及生产商、公用事业和汽车制造商如何构建物联网，这一切都会被**NFV**改变。

开源已经悄然改进企业构建**IT**系统的方式。现在，同样的事情将在**NFV**中发生。**OPNFV**这一开源项目致力于对下一代网络进行全栈集成测试来加速**NFV**。世界各地公司的工程师、网络运维人员和商业领袖们都在好奇什么是**OPNFV**以及如何加以利用。此书将带给你**OPNFV**项目的全面概览，无论你是在电信运营商还是技术供应商工作，都可以由此获得充分的自信，为你的技术团队在如何使用或参与**OPNFV**上提供方向性的指引。

内容范围

第一章，**NFV是什么？**，介绍了网络功能虚拟化，**NFV**收益、用例、**ETSI**的角色和**NFV**需求。

第二章 *NFV转型：CLI去哪了？*, 涵盖了模型驱动的架构、**DevOps**、云原生软件、**NFV**转型对组织的影响和如何开始**NFV**的旅程。

第三章，这是*OPNFV !*, 阐述创建**OPNFV**项目的驱动因素和**OPNFV**的概览，开源系统集成的概念、**OPNFV**的收益和其他可选方案。

第四章，*OPNFV上游项目*, 提供了**NFVI**、**VIM**和**MANO**项目在**OPNFV**中集成的详细介绍。

第五章，*OPNFV上游贡献*, 简要介绍了所有**OPNFV**“特性”项目对上游项目的需求和代码贡献。同时对其中一个成功项目**OPNFV Doctor**进行了深入剖析。

第六章，*OPNFV持续集成*, 涵盖了**OPNFV**的基础设施项目，详解全自动的集成、构建、发布和测试的**CI流水线**。

第七章，*OPNFV自动化软件部署*, 讨论了场景的概念和四个主要的安装器以及其他部署相关的项目。

第八章，*OPNFV持续测试*, 讲述**OPNFV**各个测试项目，测试仪表盘和**plugfest**

第九章，为*OPNFV*构建*VNF*, 对*VNF*软件架构和*VNF*上线的概述，辅以一个具体的例子，在**OPNFV**上承载**Clearwater**项目的试验

第十章，利用*OPNFV*驱动业务, 涵盖所有的关于“这对我是来说什么？”的重要话题 – 换句话说，**OPNFV**如何帮助最终用户、技术供应商和个人？本章进一步提供了如何参与**OPNFV**的具体方法。

第十一章，其他资源，以上十章内容的相关链接。

目标人群

此书面向希望深入了解**OPNFV**以及如何通过**OPNFV**加速**NFV**部署的工程师、网络运维和商业领袖。通过此书，你将获取更多的信心来引导公司的**NFV**转型方面的工作，给技术团队指明方向。此书也能够帮助技术供应商来理解如何充分利用**OPNFV**框架，帮助个人在成长中的**NFV**生态系统里拓宽职业发展路径。此书假设你具备一定的技术基础，但对各个主题没有任何实践知识的要求。

读者反馈

我们期待你的反馈！此外，尽管我们尽力去保证内容的精确性，但是错误在所难免。
请将反馈与勘误发送到opnfvbook@mirantis.com

注册

欢迎在**Mirantis**网站注册来接收**NFV**相关的博客、网络研讨、白皮书和其他内容！
在<https://content.mirantis.com/Understanding-OPNFV-eBook-Landing-Page.html>注册将得到本书的英文电子版拷贝。

序言

自2014年9月成立以来，**OPNFV**项目已经走过了一段不凡的旅程。看着我们的社区拥抱开源，完成多次发布，在上游社区和自身活动中日趋成熟，我对我们产业、我们的能力以及我们对网络变革的承诺信心满满。

与**NFV**的最终用户一起工作，使我们对现实世界中**NFV**部署的挑战有了格外深刻的认识，伴之而来的是向软件定义未来演化的诸多机会。通过这些进行中的协作，**OPNFV**通过下一代网络栈的集成测试推进了开源**NFV**在工业界的发展和演化。能为**NFV**最终用户、技术供应商和个人创造真正的商业价值是一个巨大的激动人心的挑战。

参与**OPNFV**的大门对所有人敞开，无论你是一个公司的雇员还是仅对网络变革充满热情而已。阅读此书将会帮你在开源**NFV**，**OPNFV**项目和社区方面打下坚实的基础，串联起你在其中的定位和收益等点点滴滴。奔向**NFV**的旅途不仅仅涉及各种技术，同样也是对人和过程的一种文化转变。我热忱地欢迎你加入这个生机勃勃、敏捷高效且持续改进中的社区。

我们希望你们能加入这一旅途，与我们一起奔向前方。

Heather Kirksey
Director, OPNFV
April 2017

1

NFV是什么

NFV业务驱动力

电信运营商，接入运营商（MSO，比如有线和卫星服务提供商）和网络服务商正面临以下几方面的竞争压力：



OTT/ Web 2.0



ARPU 面临压力



更高的灵活度

OTT(Over-the-top)和Web服务正在扩张，需要提供差异化服务，而不只是管道

由于收购成本增加，用户流失，每用户平均收入面

发展现有服务和发布快速增长的新服务的压力越来越大

拥有大量分支机架或**IOT**部署的企业也面临着类似的挑战。如果现在电信运营商或企业再从头开始建设基础设施网络，那么他们将倾向于使用与**Google**或**Facebook**基础设施类似的软件定义资源。这就是网络功能虚拟化的前提。

NFV是什么？

让我们从专有硬件开始。

从使用数以百计的线缆连接到单个设备，我们已经走了很长的一段路，但即使通信服务首次被计算机化，这通常也是在使用了一系列专用硬件情况下达成的，例如交换机，路由器，防火墙，负载均衡器，移动网络节点和策略平台。通信技术伴随着硬件改进一同进步，但这个过程非常缓慢，因为需要时间来进行开发和实现新设备，替换或淘汰旧设备。通讯公司和互联网服务提供商是这样，自建**IT**基础设施的大型企业也是这样。

今天，由于移动网络和云计算的出现，在很大程度上提高了消费者和企业对网络的要求，而这导致了不可预知的（“随时随地”）流量模式，和提供新服务的需求，比如在便携式设备上提供语音和视频。更重要的是，终端设备和传输技术的持续改进将不断强化这些需求。

需求的灵活性导致了软件定义网络 (**SDN**) 的发展。使用**SDN**，管理员能够根据需要轻松地配置、部署和控制网络、子网和其他网络架构，而不必手动配置专有硬件，而且这种方式对于通用硬件可以重复进行。**SDN**也使得“基础设施即代码”成为可能，配置信息和**DevOps**脚本可以像其他应用程序一样，进行监控和版本控制。

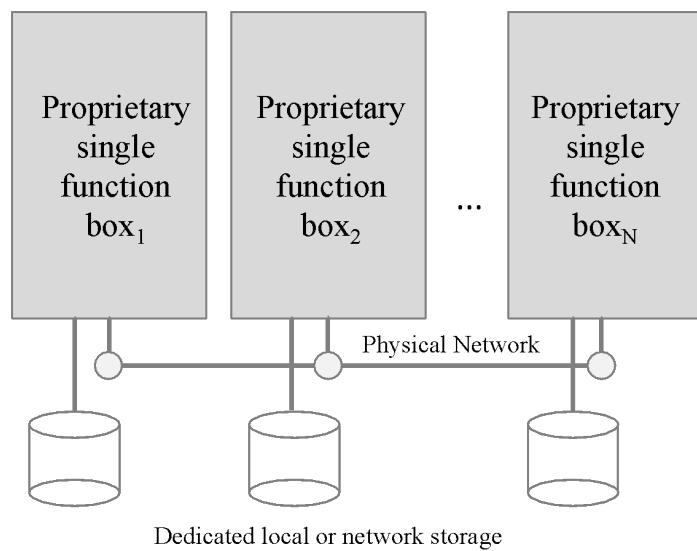
当然，这些专有硬件自有存在的价值。

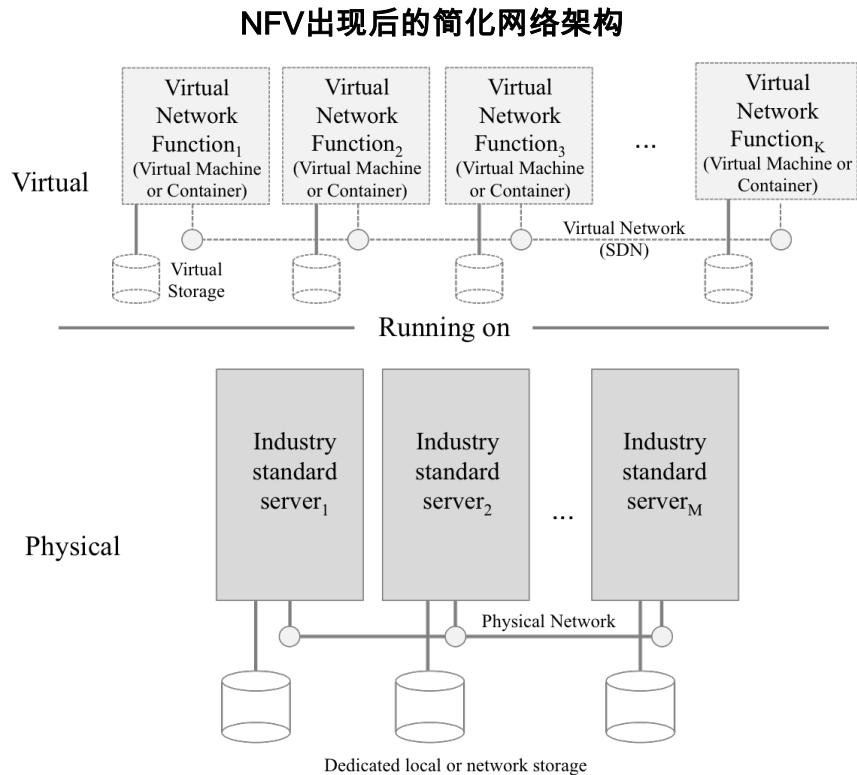
摆脱专有硬件并不像部署**SDN**那么简单；它们的存在有其特定的原因，通常和性能问

题或专用功能相关。但随着半导体性能的提升以及传统计算机硬件在复杂数据包处理能力上的增强，使得对这些专用的网络功能进行虚拟化和整合成为可能。

网络功能虚拟化（**NFV**）在这种情况下应运而生。**NFV**可以在数据中心的计算节点上执行复杂的网络功能。在计算节点上运行的网络功能称为虚拟网络功能（**VNF**）。因此，多个**VNF**可以像网络一样运行，**NFV**有机制来确定它们如何链接在一起，以提供对网络中的流量进行控制。

NFV出现前的简化网络架构





虽然大多数人都认为**NFV**面向电信应用，但**NFV**的实际用途要广泛得多，从基于应用或流量类型的基于角色的访问控制（**RBAC**）到在网络边际（此处通常需要）管理内容的内容分发网络（**CDN**），以及电信相关的更明显的应用，如分组核心网（**EPC**）和IP多媒体系统（**IMS**）。

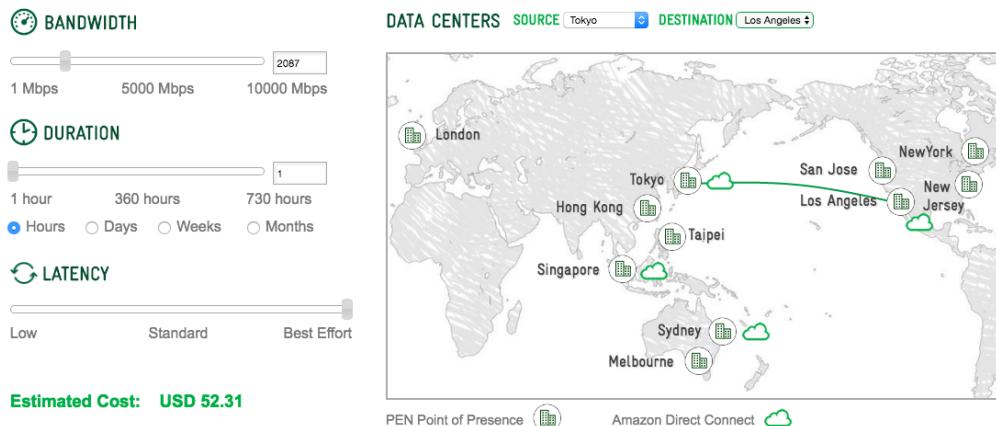
NFV收益

NFV是基于“Google infrastructure for everyone else”这一趋势构建起来的，大型公司试图复制web巨头的这一最佳实践，以增加收入和客户满意度，同时降低运营和资本成本。这就是为什么电信运营商和企业都对NFV有强烈兴趣，它具有如下好处：

增加收入

新服务可以更快地推出（因为我们编写和尝试代码，而不是设计ASIC或新的硬件系统），现有服务也可以更快地得到部署（一样的原因，软件部署而不是硬件购买）。例如，Telstra的PEN能够将WAN产品部署时间从三周减少到几秒钟，这消除了采购订单和工作时间，同时将WAN链路的客户承诺时间从一年缩短到一小时。

Telstra's PEN 产品能力 PEN Pricing Calculator



提升客户满意度

通过灵活的基础设施，实时计算每个服务基础设施资源利用率，并动态配置所需的资源，这样就再也不会出现某个服务没有资源可用的情况。（当然，基础设施总量仍然存在限制）。例如，移动终端用户不再会被降低速率或服务等级。如果运营商可以提供丰富的服务目录，同时这些服务可以快速的自助部署，客户可以先试用后购买的方式。

式尝试这些服务，这样客户的满意度也将得到提高。

减少运营支出 (Opex)

NFV消除了大量手动任务。基础设施，网络功能和服务都可以自动化部署，甚至提供自助服务。这将取消如现场服务，计划会议，IT工单，架构讨论等一系列工作。对于非电信用户，云技术已经能够将运营团队规模缩小**4倍**，让个人能专注于其他更高价值的任务。

硬件的标准化也降低了运营成本。您的团队现在只需管理标准化的几十种资源，而不是上千种各不相同的资源。减少**opex**的好处是缩短盈亏平衡时间。这是因为，除了仅虚拟化各个功能之外，**NFV**还允许以自动化的方式快速部署由多功能组成的复杂服务。通过立即部署服务，减少客户请求的时间和费用，进而转化为利润，大幅缩短运营商的盈亏平衡时间。

减少资本支出 (Capex)

NFV极大提高了硬件利用率。为应对业务峰值而配置的专用硬件资源浪费问题不再会出现。相反，您可以通过点击按钮来部署服务，并让这些服务根据资源利用率自动扩展或收缩。另一个非电信行业的例子，游戏IT公司**G-Core**通过切换到私有云，将硬件利用率提高一倍。

使用行业标准服务器和开源软件可进一步降低**capex**。有多个供应商大量生产这些标准服务器，将使价格降低到有吸引力程度。同样，开源软件通常也有多个供应商，竞争将降低价格。这将在减少或消除供应商锁定并降低价格方面获得双赢。

此外，运营商可以通过不同的采购模式来减少**capex**。在**NFV**之前，传统模式是向网络设备制造商（**NEM**）发布**RFP**，并从其中一个采购全部的解决方案。**NFV**模式下，运营商现在可以为不同的堆叠组件选择不同的最佳供应商。事实上，在有些领域，运营商也可以通过使用**100%**的开源软件而完全跳过供应商。（最后两个选项不是单纯的理念，我们将在下一章中探讨不同采购模式的利和弊。）

TIA Network的“虚拟化革命：**NFV**生产力 – 未来网络[纪录片](#)，第6部分”指出，基于**NFV**的架构的**opex**和**capex**整体支出可能会降低高达**70%**。

为新目标释放资源

如果运营商每个资源都忙于保持和运行当前的服务，将没有足够的人力资源来应对新的、即将到来的方向，如5G和IoT。运营商降低opex的另一个作用是释放组织资源，并投入到这些重要的新方向，从而有助于整体提高竞争力。或者换个说法，除非完成底层的自动化，否则不会有充分的时间聚焦在OSS / BSS层面上，而这才是真正提高竞争力并产生收入的层面。

总体拥有成本(TCO)分析案例

英特尔和PA咨询集团为vCPE创造了一个全面的TCO分析工具（见下文）。在与英国电信合作的一次典型研究中，工具假设客户的物理网络功能被部署在运营商的云中。在该研究中，工具显示，运营商可以将总成本降低32%至39%。该图包含了硬件、软件、数据中心、员工和通信成本等所有成本。本次TCO分析包含了五年期内涉及到的防火墙、路由器、CGNAT, SBC, VPN和WAN优化等一系列功能。这些结果具有代表性，如果假设条件不同，这些结果或许会发生明显改变。此外，如前所述，成本仅是NFV的众多收益之一。

**研究发现
vCPE可以
降低32%-
39%的
TCO**

NFV 用例

由于NFV普及的发起组织主要来自于电信运营商，因此大多数原始用例与电信领域相关也就不足为奇。正如我们所讨论的那样，NFV用例涵盖了更广泛的行业。为避免面面俱到，下面将介绍三个最常见的用例：

vCPE (虚拟客户端设备)

vCPE通过虚拟化一系列设备（例如防火墙，路由器，VPN, NAT, DHCP, IPS / IDS, PBX, 编码器，WAN加速等）来将业务或消费者连接到互联网或将分支机构连接到总部。通过虚拟化这些功能，运营商和企业可以通过消除现场施工和漫长的手动

过程来快速部署服务，从而增加收入并降低成本。**vCPE**用例同时还预示了分布式计算需求，集中式云可以由边缘计算来补充。

vEPC (虚拟分组核心网)

从**2G**到**4G/LTE**演进的过程中，在**5G**前昔，无论是流量还是使用数据业务的用户数量的绝对值都在持续增长。**vEPC**使移动虚拟运营商 (**MVNO**) 和移动虚拟集成商 (**MVNE**) 能够使用虚拟而不是物理基础设施来承载语音和数据服务。同时提供多个服务的先决条件是支持“网络切片”或网络多租户，**vEPC**也提供了这种能力。总而言之，**vEPC**可以在减少**opex**和**capex**的同时加快交付，并支持按需扩展。

vIMS (虚拟IP多媒体系统)

OTT的竞争正在促使传统电信、有线和卫星供应商提供基于**IP**的语音，视频和消息服务。虚拟化系统所提供的敏捷性和可扩展性，使**IMS**在经济上可以有效地与初创公司竞争。

甚至在短期内，这个用例列表也不可能得到完善。当前已经有数不清的用例，并且新的用例还在不断涌现。其中最明显的就是**5G**，需要提供**50倍**的更高速率，**10倍**的更低时延，物物通讯，车联网，智慧城市，电子健康，物联网以及边缘计算和网络切片，很难想象电信服务商和企业能通过使用物理网络功能而获得成功。

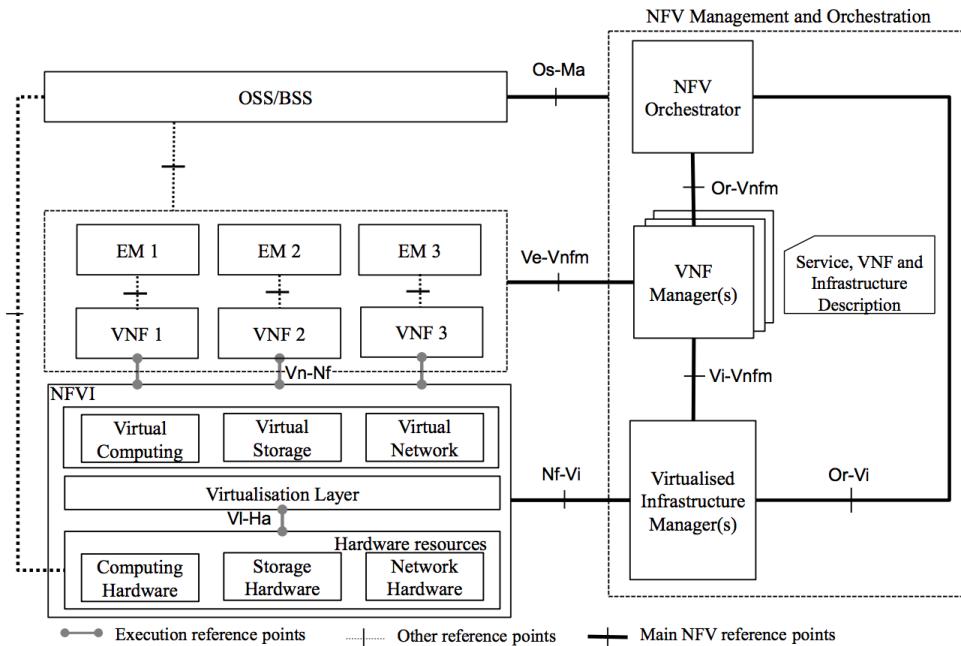
NFV在ETSI进展

ETSI在**2012**年开始**NFV**的标准化进程，其目标是定义**NFV**是什么，以及行业将要为此准备什么。**ETSI**的[NFV行业标准化工作组 \(ISG\)](#)已经从七家增加到三百多家企业，并发表了**40**多篇文章详细介绍应如何实施和测试**NFV**。**ETSI**在标准领域不断推动**NFV**的贡献工作是非常宝贵的。让我们快速浏览一下**ISG**创建的三个与**NFV**相关的重要文件。

NFV 架构框架文档

所有的**ISG**工作组都需要参考本[文档](#)进行工作，因而尤为重要。

ETSI NFV架构(源自ETSI)



实际的NFV架构框架由几部分组成：

- 网络功能虚拟化基础设施 (**NFVI**) 是实际发布这些服务的基础设施，包括：
 - 物理硬件，如服务器，存储，网络和
 - 虚拟基础设施，如虚拟计算（虚机或容器化虚拟操作系统），虚拟存储和虚拟网络（软件定义的网络）。
- 运行在**NFVI**之上的一个或多个虚拟网络功能 (**VNF**)，这些**VNF**提供实际网络功能的软件版本。在大多数情况下，已经可以取代传统基于硬件实现的网络功能，如交换机、防火墙和负载均衡器。软件功能应当与硬件无关，但通常由于性能原因而需要硬件优化。
- 当然，**NFVI**和**VNF**都必须进行管理，这由**NFV**管理和编排器 (**MANO**) 负责。由于**MANO**根据服务的需要部署软硬件资源，并负责软硬件资源的生命周期管理，因而显得非常复杂。**MANO**通常不是一个孤立系统，还需要与现有的外部**OSS**和**BSS**进行交互。

- 最后，**NFV MANO**需要使用元数据，这些元数据可以描述服务，**VNF**和对基础设施的要求。

架构框架非常重要，需要确保架构的不同部分的多个参与者开展工作，因此有必要对各个组件进行抽象和虚拟化，以确保它们之间不相互耦合。一旦解耦，彼此间的接口就可以确定和标准化。然而，即使被标准化了，各组件之间的互操作也是一个持续而复杂的任务。

NFV主要概念术语文档

[本文档](#)是其他文档中所使用术语的集合。**ISG**希望其他团体不仅能使用这些术语，也能贡献其它额外的术语，以简化与**NFV**相关问题的对话。

NFV原型框架文档

如果没有真实的实现来证明这些方案是可行的，那这些理论将毫无意义。为此，**ISG**还为那些希望根据其研究进行原型验证（**PoC**）的人制定了[规范](#)。这些规范包括提案模板，以及对参与方的最低需求。例如，**PoC**必须至少包括两个供应商和至少一个已经是**NFV ISG**成员的网络运营商/服务提供商。

这些**PoC**的目的是为了实现全功能**NFV**生态系统而提供反馈，让人们了解必须克服的一个或多个互操作性或技术挑战。

NFV需求

虽然很容易确定**NFV**的积极方面，但**NFV**要想取得成功，有几个需求必须解决。下面讨论一些更重要的“电信级”需求。

安全

安全在当今世界至关重要，每天都会发现新的威胁，其中网络间谍是一个普遍问题。

也就是说，理解支撑**NFV**的虚拟化基础设施所引入的额外安全问题就显得非常重要。

NFV ISG安全专家组注意到这一情况，并提出了针对这些虚拟化环境的**10个安全问题**：

- **拓扑验证和强化**：拓扑验证和强化确保数据流只能到达其预定的目标。在物理网络中，这更直接，数据流可以在管理上进行隔离。而在虚拟环境中，涉及更多内容 – 数据流可以到达任何地方，无意间形成的回路被利用，这些可能性理论上都存在。
- **支撑基础设施的管理可用性**：我们在这里不是在谈论组织中的上级，而是讨论在系统崩溃后对系统进行管理的能力。尽管使用单独的物理网络更好，但考虑到成本因素，通常通过独立端口提供连接；请务必确保对此端口访问进行保护，至少也需要使用**VPN**技术。
- **安全启动**：主机能够在启动**VNF**之前验证其是否是正版，**VNF**在执行敏感操作之前也清楚主机未发生泄露，这些非常重要。
- **安全崩溃**：当**VNF**崩溃时，系统必须确保销毁对其的所有引用，例如打开的连接或持久化存储到文件系统的数据。
- **性能隔离**：在公有云中，这通常被称为“坏邻居”问题；两个进程运行在同一个CPU核上或共享其他资源会互相干扰对方的性能。为了解决该问题，您可以指定进程必须拥有各自的预留资源，但这又会导致资源使用率的挑战。
- **用户/租户认证，授权和计费**：**NFV**增加了用户必须识别的组件层数，这导致了计费的复杂性。使用令牌而不是直接身份传递可以减轻安全隐患，但总体而言，**NFV**确实比传统的硬件环境具有更大的攻击面。
- **认证的时间服务**：这里指的是一个潜在的攻击事件，而不是实际攻击；一些代码依赖时序以正常工作，因此它通过查询多个源以获得“真实”时间。然而，**VNF**只能与一个虚拟机管理器进行交互，根本不可能获得多个“投票”。（请注意，这实际上是一个云计算问题，而不是**NFV**特有的问题。）
- **克隆镜像中的私钥**：在理想情况下，克隆的镜像不会包含用于访问的私钥或密码，这些信息仅在启动时被注入。不幸的是，这不是理想情况，这不一定是个挑战的技术问题，而是一个运维问题。
- **用于虚拟化测试和监控功能的后门**：另一个操作问题涉及官方或非官方启用的用于测试或调试的接口，操作员通过这些接口能够测试或调试生产环境的组件。如果您必须保留这些接口，请确保它们受到保护以免受攻击。
- **多管理员隔离**：减轻多管理员隔离问题是一个仍然需要研究的领域。当**VNF**的管理员试图提出比整系统的管理员更安全的访问要求时会产生一系列技术问题

。（这种情况在“合法侦听”时会出现）问题是，一般而言，很难保证这些资源对系统管理员是安全的。

[NFV安全专家组](#)已经向**NFV ISG**提出了这些问题，同时与其他标准机构合作以试图减轻这些问题。

性能

NFV基础设施旨在模拟一个由软件组成的硬件环境，但这并不意味着您只需将软件放入云中并放任不管。大部分**NFV**组件都要求系统提供极高的性能，但**VNF**并不总是知道它们的需求能否从环境中获得。

这意味着若要在性能方面利用最佳实践，完全取决于操作员如何配置和部署这些软件。事实上，根据**NFV ISG**，使用**NFV**和正确使用**NFV**之间的速率差异可以高达1000%。不用说，**NFV**减少资本支出的承诺可能会很快被削弱，如果你不得不购买多个性能不佳的标准服务器来替代一个专有硬件。总之，对于这个行业，我们需要缩小**NFVI**与物理网络功能性价比之间的差距。

互操作性

NFV是一个多厂商参与的生态系统，因此这些厂商的解决方案可以协同工作就显得特别重要。为此，定义整个系统各部分之间的互操作性规范至关重要，如编排器，虚拟化基础设施管理器（**VIM**），虚拟网络功能管理器（**VNFM**），**VNF**和网络功能虚拟化基础设施（**NFVI**）。此外，与传统OSS / BSS系统的互操作性也同样重要。

在这种情况下，通过开源和鼓励PoC都可以帮助达成这一目标，多个运营商也尝试使用多个组件来组合系统。

易于操作

亚马逊拥有一大批云专家来运营其**AWS**。如果**NFV**部署也有类似要求，这将极大增加运营成本并减缓其部署。因此，**NFV**若要成功，关键是要操作简单。从“day-1”初始

部署到“**day-2**”部署后运维，例如更改配置，添加容量，添加功能，修改参考架构，更新和升级，都需要极大简化，这样才能大规模部署**VNF**。而且，这些操作在执行时不能中断现有的**VNF**业务。此外，需要提供易使用的日志监控，计数器和告警的集群汇总功能。

NFV相关需求

NFV有许多独特的、与企业业务不同的需求。我们来看一下最重要的几个：

- **服务保证**：可以这样说，**VNF**的可用性要求高于互联网或移动云应用。毕竟，**OnStar**客户为应对事故，基于**LTE**网络请求紧急服务，不能容忍网络有任何可用性的损失！那么我们如何检测出故障根因并有效地解决它呢？这是服务保证的关键，我们希望能够追踪基础设施真实故障的根本原因，并以最优的方式解决。随着时间的推移，希望在故障发生之前能主动预测潜在的失败，而不是被动发生。终极目标寄希望于使用人工智能和机器学习来预测和应对这些问题。
- **服务功能链**：这是为服务提供商将端到端之间多个**VNF**的数据流链接起来提供服务的概念。这些服务可以是认证、安全和转码等的任何服务。更有意思的是，需要基于策略动态插入**VNF**。因此，根据所使用的设备，签约计划，以及位置，从同一网站请求视频的两个用户可能会插入不同的**VNF**来提供服务。

下一步计划

在电信和企业领域，**NFV**正处于增长势头，这正在引导新的服务模式和用例，2016年8月**Heavy Reading**进行的一项[研究](#)证明了这一点，98%的受访服务商表示**NFV**是他们战略的一部分。一般来说，**NFV**只不过是一个网络密集型的负载（而不是计算密集型）。随着**IOT**，物物通信，智能边缘设备（可穿戴，无人机，增强现实等）的兴起，**NFV**也将在企业中得到广泛应用。可以肯定的是，每个人都希望他们的应用和服务具有更高的质量，更有活力，更有弹性并得到更好地优化。

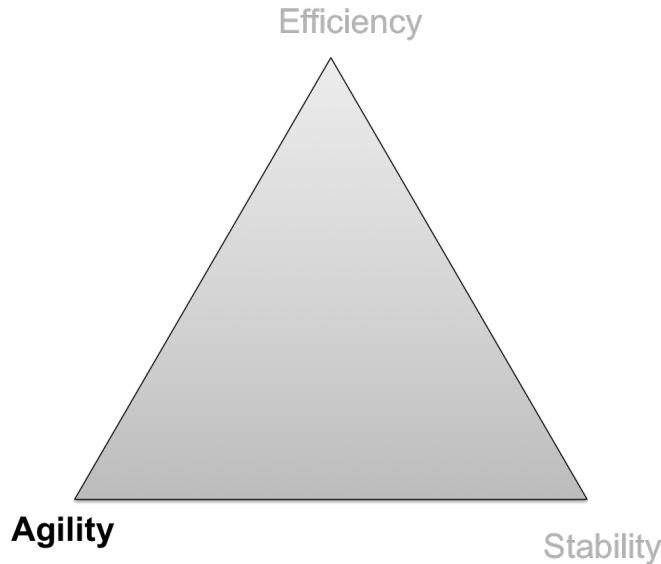
除**ETSI**之外，许多的标准和开源组织也开始了**NFV**相关的工作。其中**Linux Foundation**及旗下的**OPNFV**社区在这方面处于领导地位。在第3章回到**OPNFV**之前，我们在下一章先看看**NFV**转型对一个组织的意义。

2

NFV转型：CLI去哪了？

我们已经确定网络、软件开发敏捷性和相比竞争对手的创新能力将变得越来越重要，而**NFV**正好可以实现这种敏捷性。然而，若要在**NFV**方面取得成功，用户需要进行更广泛的转型，而不仅仅是技术转型。事实上，**NFV**影响到许多非技术方面，如流程，组织结构和技能集合的获取。除此之外，**NFV**还对商业模式和采购方式等次要因素产生影响。我们先来看看**NFV**要求的三大技术变化，然后再看看对组织产生的影响。

现在敏捷已经成为关键差别



NFV驱动技术变革

要取得成功，运营商需要接受三项广泛的技术变革：

- 模型驱动架构
- DevOps
- 云原生VNF

尽管大多数这些变革才初露端倪，但随着它们日趋成熟，将变得越来越重要：

模型驱动架构

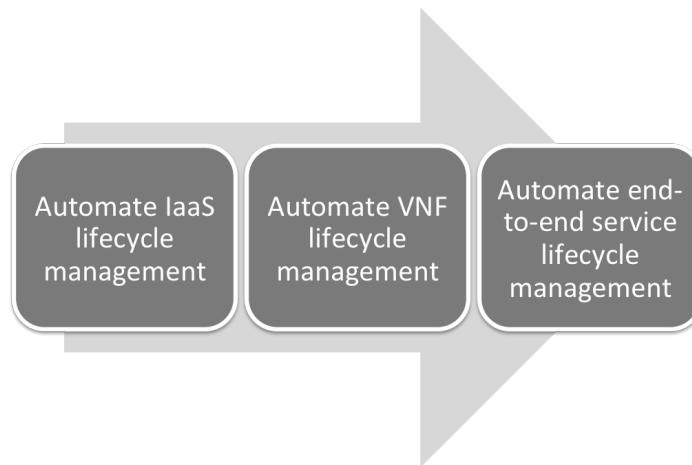
在物理网络功能上部署一项新服务花费30–60天并不罕见。必须购买设备，通常使用设备商特有的语法，工具和专业服务才能进行配置和部署。然后把这些物理设备连接起来组成服务。

用行业标准服务器替换这些固定功能硬件，可以快速配置基础设施，但仍没有解决快
30

速部署服务的问题。即使借助虚拟化，**VNF**也必须安装和配置，然后连接起来组成完整的服务。提供服务通常要求这些功能必须按一定的顺序进行部署。最后，为每个**VNF**提供监控的基础设施必须到位。一旦服务启动并运行，“**Day-2**”工作就会涉及到更新、升级、扩容、缩容、自愈及配置更改等系列活动（参见下面**NFV**软件生命周期全自动化图）。如果手动执行可能需要几周或几个月的时间。

模型驱动的架构应运而生。模型驱动的模板采用了一个便于人阅读的文件，其描述了整个网络服务，每个**VNF**的配置，拓扑和连接，各种依赖和触发行动的策略。这些模板还可以描述管理，监控和能力**API**。当达到整个网络服务描述符（**NSD**）即代码时，不需要手动步骤或工单来部署服务，即使在不同的底层**NFVI**、**VIM**或**SDN**环境，也都可以每次以完全相同的方式创建，组合和部署服务。此外，模型驱动模板也可以被版本控制，以便更好地跟踪和遵守。

模型驱动架构助力NFV



如果模型驱动模板是如此有用，为什么仅用于网络服务描述？实际上，这些模板也用于描述较低级别的组件，例如**VNF**描述符，软件应用程序和基础设施资源，以及更高级别的项目，如产品（具有商业属性的服务集合）和套餐（市场配置的产品组合）。虽然较低级别的模板关注于资源，策略，拓扑，配置和依赖性上，较高级别的模板关注流程，订单，定价和业务规则，模型驱动模板都能适用。总而言之，模型驱动架构

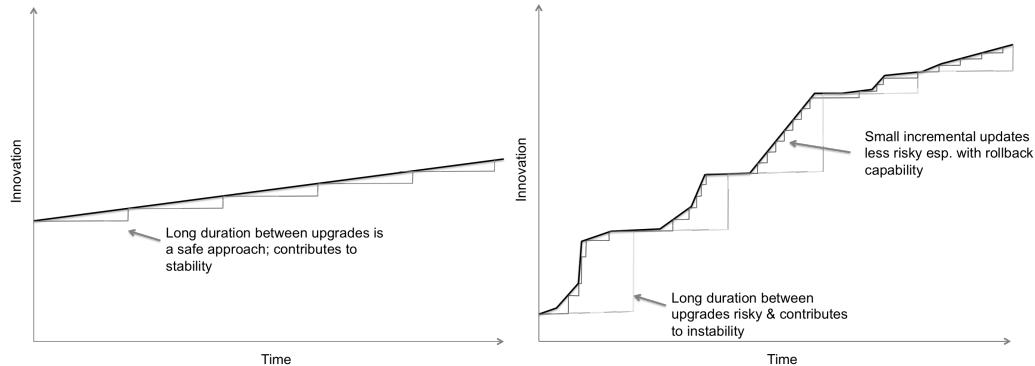
大大减少了运营商为客户交付新产品和服务所需的时间，并且能够快速上线第三方解决方案。

DevOps

确保NFV转型成功的第二项技术变革领域是**DevOps**。

NFV业务栈（VNF, NFVI和MANO）的每一层都以比以往在物理网络功能时期所见到的创新速度要更快，而且这一步伐不会减慢。为利用这种创新速度，组织必须从根本上改变技术组件的方法。

技术创新需要新方法



左图显示，设备商在一定的基础上以可预测的，相对较慢的速度进行创新。在这种情况下，用户每6–18个月才升级一次版本，在此期间进行日常维护操作。

右图显示，来自于供应商和开源社区的快速、连续和不可预测的创新速度，同样安全和稳定的方法不再适用。6–18个月升级周期将对整个服务或某类服务造成巨大的风险

-

此外，在不必要的**6–18**个月窗口期中，就像开发者无法得到宝贵的用户反馈一样，使创新远离最终用户。相反，定期小的增量更改就显得很有价值。

这是一种什么样的环境呢？以**Netflix**为例，尽管**Netflix**不是严格的**NFV**架构，但它的确体现了这种开发模式；**Netflix**每天都会对其生产环境进行数以千次的小更改。使用“**forklift upgrades**”的旧方法，这种速度不仅闻所未闻，而且无疑是一种自杀行为。

这种进行持续增量更改的过程源自一种称为**DevOps**的实践。最近，“**DevOps**”这个术语被稍微过度使用，导致了对其实际意义的一些争议（即使在维基百科进行了定义）。有些人将其视为开发和运营团队紧密合作的一种心态。有些人将其视为一系列活动。实际上也许两者兼有。

前者认为，**DevOps**使两个团队不同目标达成一致。开发团队¹的目标是提高软件交付速度。显然，开发人员热衷于创造变更。另一方面，运维团队关注于网络可用性，平均故障间隔时间（**MTBF**），平均修复时间（**MTTR**）和其他运维指标。所以，运维团队未必会试图阻止变更，但他们的确有义务实施变更，并确保不会发生中断。

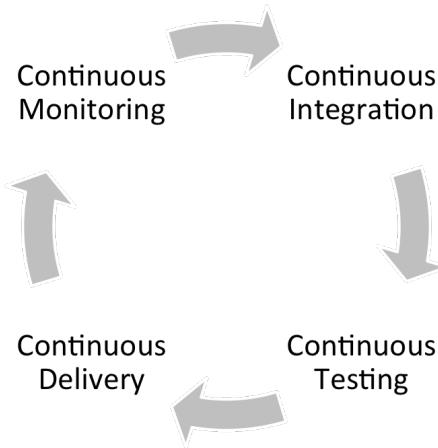
DevOps是通过打破这两个团队之间的壁垒鼓励团队合作的一种方式，并帮助他们在不危及稳定性情况下拥抱变更。

我们以第二种视角“**activities**”来看**DevOps**，它涉及以下的步骤：

持续集成，持续测试，持续发布和持续监控。

DevOps涉及步骤

¹ 在**NFV**用例中，**NFV**软件栈和**VNF**的开发通常不是在内部能完成的。正是由于这个原因，**NFV DevOps**中的“**Dev**”可以指低耦合的外部供应商和开源社区。这和**Web**巨人绝大部分开发都在内部完成的方式不同。



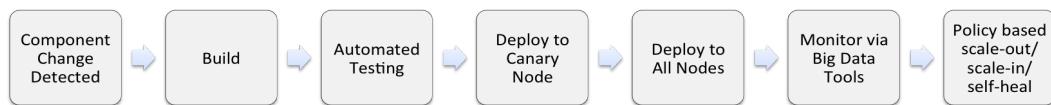
面向内部软件开发的传统**DevOps**与**NFV**的**DevOps**不同，因为**NFV**的软件组件既来自供应商，也来自于开源社区。在这个意义上来说，**NFV**用户不拥有**VNF**的开发－集成－测试的周期，而是作为组成第三方**VNF**的集成商。另一个不同在于，传统**DevOps**假设是相同的服务，而**NFV DevOps**的服务是多种多样的。由于这些原因，**NFV**的**DevOps**与传统**DevOps**有着明显的不同。我们来深入了解一下：

- **持续集成 (CI)**：出发点是持续构建和代码集成。当在内部开发时，字面上的意思是开发人员的每个提交都会导致一次新的集成。然而，当使用第三方和开源软件时，这通常意味着每当外部软件版本发布时，才可以启动集成工作。在有这么多组件（虚拟机管理器，**VIM**，**MANO**，**SDN**控制器，**VNF**）的情况下，如果不每天而是每周集成，您可以想象未来每次会有多大的变化。
- **持续测试**：仅仅进行软件集成是不够的。即使软件每一部分都经过测试，整体软件栈也必须在开发或测试环境中成功完成集成和测试。显然，所有这些测试都必须是自动化的。进行这么高频率的测试，手工测试是行不通的。
- **持续交付 (CD)**：一旦软件通过所有自动化测试，就会部署到生产环境。理想情况下，新的软件组件与旧版本共存，新组件仅用于一部分受影响的服务。这项技术称为金丝雀（**canary**）或蓝绿部署。如果新组件在最初有限的生产部署中可以胜任，那么它将扩展到其余的生产环境，并取消旧组件。如果不能胜任，可以简单地回滚变更。这样，通过小的增量变化促进了系统稳定。**CD**将故障一次限定在一个组件，不会发生整个服务或多个服务失败的情况，但这却是大版本升级典型问题。
- **持续监控**：最后一个难题是对**NFV**服务进行持续监控。因为任何**NFV**软件组件

的生命周期管理决策必须根据策略自动进行，因此需要使用大数据系统对每个日志，指标和告警进行自动处理。人类根本无法监控所有这些单个组件。这就是为什么你不可能看到通过**CLI**访问某个**VNF**的原因，在**NFV**中这个概念已经没有意义！

为了成功实现**DevOps**，组织需要自动化工具。**CI**（包括持续测试）和**CD**常用于自动化的流水线。我们将在第6章中详细介绍**OPNFV CI**流水线。一般来说，**CI / CD**流水线及持续监测，可以概括如下：

典型的CI/CD流水线和进行中的监控



再次，如果这个话题看起来离我们如此遥远，也无需担心，我们并不指望几年就可以建立起整个**NFV DevOps**。

云原生软件

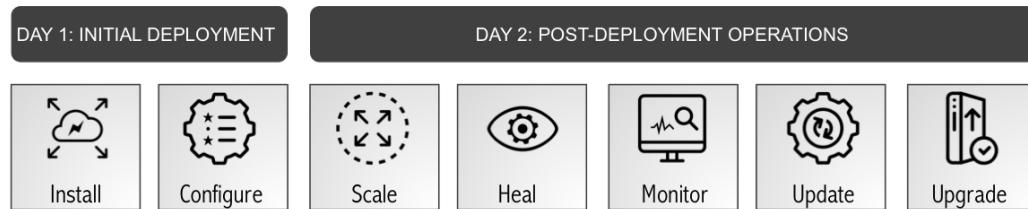
确保**NFV**转型成功的第三项技术变革领域是云原生软件。整个**NFV**栈需要随着时间的推移迁移到云原生软件架构。类似于**DevOps**，云原生**NFV**与消费者及企业使用的云原生应用类似，但不一样。

简单地说，云原生**NFV**软件架构有如下要求：

- 能够自动化扩容（到多个节点）、缩容和自愈。
- 能够容忍经常出现故障的硬件环境，甚至运行得很好。
- 支持**NFV DevOps**（见上文），以对**NFV**软件栈的每个组件的整个生命周期达到全自动化管理。

简而言之，下面的每一步都必须是软件定义的，零手动步骤：

全自动的NFV软件生命周期的管理



这三个要求需要围绕一套核心原则要求重新构建软件应用。下面的列表并不完整；现代云原生软件基于一个称为“十二要素应用”的原则，但是当您认为**NFV**就一定是“云原生”时，结果未必这样。

您看到的十二因素应用意味着无状态 – 处理和状态管理分离，使得系统的任何部分与任何其他部分没有差别。然而，网络应用都是关于状态管理的，所以正常的云原生假设需要为**NFV**²做一些调整：

- **明确定义API的微服务：**VNF需要分割成松耦合的小部件或微服务，这些微服务通过明确定义的API进行通讯。代码和数据模型需要拆分。这种方法为运营商和VNF供应商都带来好处。VNF能够以非常细的粒度来进行弹性，降低故障的影响（通过将涉及范围限制为一个微服务），并使用DevOps机制来管理VNF的生命周期，运营商将受益匪浅。供应商也将受益于更快地推出新功能并取消无用功能。正如我们所讨论的，VNF与传统应用不同，它们具有执行包处理的数据平面和提供管理、信令等的控制平面。因此，微服务必须小心使用。数据平面不适合使用，数据通路的多个微服务可能会由于多跳而降低效率，因为每一跳都增加了交换和数据包处理开销。
- **硬件解耦：**大部分DevOps章节的收益只有在工作负载与底层硬件完全解耦并且可以通过API请求资源时才能获得。这当然是通过虚拟化实现的。解耦并不意味着将VNF与硬件脱离，因为总会有一些VNF需要服务器提供特定功能，例如数据面加速。随着时间的推移，诸如平台即服务（PaaS）和“serverless apps”（对底层服务器，操作系统和相关基础设施进行完全抽象，代码只在响应事件时执行）等新范例将进一步推动VNF和硬件解耦。

² 我们在大多数情况下讨论VNF，这些准则对于NFV软件栈的其它组成部分同样适用
36

传统应用依赖底层基础设施硬件提供可用性，基础设施必须保证应用所需要的多少个9的可用性。在云原生应用中，需要假定硬件任何时间都可能发生故障，这个责任转移到了VNF。当前的模式通常被称为“**Pets**”（即每个节点必须得到关注），与其对照的是一种被称为“**Cattle**”的新模式（即实例被作为一个集合来处理，而其中某一个实例可能随时启随时停）。

例 1： VNF 和‘Pets’ 基础设施之间的旧合约

- CPU 品牌, 类型, 频率, 数目
- 内存容量
- NVMe 闪存类型, 容量
- 硬盘 品牌, 类型, 速率, 容量和数量
- 网卡 品牌, 性能和数量
- 电源和风扇规格和数量

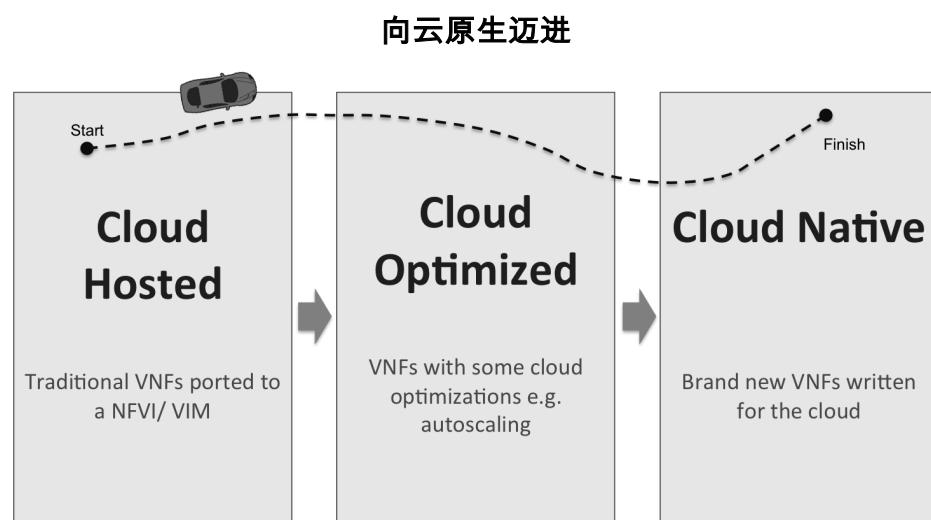
例 2： VNF 和‘Cattle’ 基础设施之间的新合约

- vCPU 数量
- 内存容量
- HDD 或 SDD 块存储容量
- 具体性能属性的 IO
- 数据面加速或其它特别要求

- **状态隔离**：在传统的云原生应用中，应用的状态隔离在数据库服务（SQL 或 NoSQL）中。所有其他功能都是无状态服务。而网络本质上全是分布式状态管理，所以状态隔离仅应用于VNF的微服务上下文中。每个微服务应将事务处理逻辑与其状态分离。这可以促进更快的软件开发速度，并有能力把持久性数据库或存储层标准化。数据库或存储层可以跨越从内存，闪存，块存储到对象存储的整个性能范围。
- **横向扩展与纵向扩展**：云原生应用程序的扩容应该是通过增加实例数量而不是给某指定实例添加更多资源来实现。这对于实现硬件无关性和响应资源利用率的性能扩缩容至关重要。横向扩展的必然结果是自愈，如果现有实例变得不可用时，系统将创建新的微服务实例。
- **抗脆弱性**：这可能是最重要但最被忽视的因素。一个抗脆弱的系统好于一个稳定的系统。这是一种提升稳定性来应对基础设施和其他故障为压力的系统。自

互联网问世以来，网络服务被设计为抗脆弱服务，但这对**VNF**来说却是全新的。事实上，**Netflix**开创了一个全新的“混沌工程”领域，在这个领域里，他们通过对实例、网络、可用域和整个区进行故障测试，以确保其抗脆性。并在产品开发的所有阶段都会进行。

实际上，要让一个组织的所有工作负载转化为云原生需要花费很长一段时间。**Netflix**花了7年的时间才从传统的应用完全转移到云原生！对于大多数用户来说，在此期间，使用云托管的**VNF**(传统的**VNF**虚拟化)或云优化的**VNF**(部分云原生的**VNF**)也能实现**NFV**的很多好处。事实上，大多数**VNF**供应商正在采取这种方法，从仅虚拟化其物理网络功能并进行一些基本的性能优化来开始。



对组织的影响

以上三个变化 – 模型驱动的架构，**DevOps**和云原生**VNF**，影响了组织的许多方面，因而需要进行转型。尽管关于这个转型可能是什么样子的完整讨论超出了本书的范围，但是，我们将提出一些问题将帮助您确定您的组织是否为**NFV**做好了准备。对每个问题进行1到5打分 (1 = 未准备好， 5 = 完全准备就绪)。对于每部分的得分意思如下：

| 分数 | 准备程度 |
|-------|---------|
| 20-25 | 完全准备好 |
| 15-20 | 大部分准备好 |
| 10-15 | 部分准备好 |
| 5-10 | 需要重大转型 |
| 0-5 | 还没有开始转型 |

对组织结构的影响

电信运营商的技术团队通常按物理网络功能进行组织，更不用说，IT、网络和OSS/BSS团队是相互独立的。对于NFV而言，组织结构可能需要围绕公共基础设施池和工作负荷进行重组。并且所有上述功能的技能也需要汇集在一起。

此外，云原生思维可能会迫使更大的变化。例如，在Twitter，一个团队负责一个微服务，团队不超过七人。每个团队都是跨职能的，拥有完整的开发，测试，以及在产品环境部署和监控他们的微服务的角色。这些团队通常像一个初创公司一样设置。他们负责自己的技术决策，优先考虑自己的积压任务，并对团队负责。当然会存在跨团队的合作和协调。这样做解放了平台团队，其只负责实际的硬件操作和提供自动化工具。故障是不可避免的，当有故障的时候，会有一个跨团队非问责性的复盘。

组织结构准备问卷：

| | |
|----|-------------------------------------|
| 问题 | 准备程度: 1-5 (1 = 未准备好, 5 = 完全准备就绪) |
|----|-------------------------------------|

| | |
|---|--|
| 有没有为新技术跨职能部门组建团队的先例？ | |
| 您能想象每个NFV服务或组件（如NFVI, VIM, MANO）都拥有开发，测试，发布和产品化的完全自主的团队吗？ | |
| 您能想象一个分散式的决策方法吗？每个团队都能在合理的范围内做出自己的技术决策。 | |
| 您能想象NFV服务团队和平台团队之间的沟通是否完全自主的吗？ | |
| 是否有机制来讨论影响多个团队的需求，并协调这些需求？ | |

对过程的影响

在2016年奥斯丁举行的OpenStack峰会期间，Jonathan Bryce谈到了一个客户耗时44天部署一个应用程序。一旦公司实施了OpenStack，他们就可以把时间缩短到42天，尽管同样令人失望。他们保存了所有的检查点和手动签字！一旦解决了文化和流程问题，其耗时将缩短至2小时。这证明：要成功，组织需要“软件定义的人”。

流程包含了例如检查点和签字之类常规的流程，也包含例如资产管理，安全性，服务保证，审计等复杂流程。由于实例的启停波动，这些在虚拟世界中很难实现。通常任何改变都面临许多官僚作风，以促进稳定和降低风险。对于NFV，将需要对所有流程进行全面审查，那些不能真正有助于稳定性的流程将被基于信任和责任感的文化所取代。任何不能带来真正价值的电子表格、会议、单据或移交都必须消除。目标是将围绕物理网络功能的过程转移到基于云的方式。

流程准备问卷：

| 问题 | 准备程度: 1-5 (1 = 未准备好, 5 = 完全准备就绪) |
|----|-------------------------------------|
| | |

| | |
|--|--|
| 您能想象一个实习生在工作的第一天就往生产环境推送服务更新吗？（另一个不完美的类比，但是在互联网世界中， Etsy 要求他们的实习生在工作的第一天将变更推送到现场） | |
| 您能想象一个工程师在没有得到任何许可的情况下，对用户群的1%的用户试验一个新的特性吗？(Twitter 允许。) | |
| 您使用敏捷方法论吗？这种方法论真的很敏捷吗？还是它是一个 <u>瀑布和敏捷</u> 的混合体？ | |
| 您的财务团队是否可以批准不分配给某个特定成本中心而是用于聚合 NFV 云的硬件采购吗？ | |
| 您的合规，资产和安全团队能否处理动态的虚拟基础设施？ | |

对技术的影响

技术的影响也许是最清楚的。组织必须能够吸收这些新技术并有效地使用。

技术准备问卷：

| 问题 | 准备程度: 1-5 (1 = 未准备好, 5 = 完全准备就绪) |
|---------------------------------------|-------------------------------------|
| 您是否有任何 NFV 软件需要云托管，云优化或云原生的计划？ | |
| 您的组织能否接受 <u>混沌工程</u> 的原理？ | |
| 您的组织是否准备好搭建 DevOps 流水线和自动测试？ | |
| 您的运维团队能否适应模型驱动架构？ | |

| | |
|---|--|
| 您能想象一个无法访问单个 VNF 或实例的监视框架，而只能在聚合云上进行管理吗？ | |
|---|--|

对技能集合的影响

NFV中的技术栈比物理功能需要更高等级的技能。管理**NFV**部署的团队需要同时具备功能领域和非功能领域的技能，在功能领域，需要了解如**NFVI**, **VIM**, **SDN**控制器, **MANO**等新组件，在非功能领域，需要了解基础设施和**VNF**的生命周期，监控，故障排除等。这些技能必须有组织地灌输，因为您不可能在外部能雇用到足够多的人才。即使有，也可能更昂贵。

技能集合准备问卷：

| 问题 | 准备程度: 1-5 (1 = 未准备好, 5 = 完全准备就绪) |
|---|-------------------------------------|
| 员工是否允许任何时间和/或访问实验室资源来自学新事物？ | |
| 您可以想象举行一个公司级的 Plugfest / Hackfest 来促进围绕诸如工具开发，互操作测试， VNF 上线和运营最佳实践等活动的紧密协作？ | |
| 是否有正式的培训计划？您公司的技术人员参加有关的展览或峰会有多容易？ | |
| 是否有内部工作坊或知识转移计划？团队会彼此分享学习吗？ | |
| 是否有项目的复盘？你能想象在这些复盘过程中上级管理层不会进行问责吗？ | |

尽管组织结构，流程，技术和技能集获取是受影响的主要方面，此外还有两个次要方面也需要转变：商业模式和采购。与主要方面不同，我们将跳过次要影响方面的问卷。

次要影响：商业模式

目前电信运营商业务模式已针对大众市场进行了优化。随着物联网和网络切片等新趋势发展，商业模式必须针对特定行业和商业案例进行量身定制。一个方案不可能适用于所有商业模式。如果客户的交易更适合不同的持续时间，例如每天，每月甚至是每小时，以及不同的指标，如使用虚拟现实访问的城市数量，或者捕获的Pokemon数量等，那么商业模式将必须做出相应调整。

次要影响：采购

在NFV转型中，您的组织如何与供应商打交道，需要重新评估。涌现了各种各样的新模型。



有很多衍生变化，但广泛地说，模型有：

- **100%自己动手 (DIY)**：在这种模式下，用户购买商用硬件并仅使用开源软件。这导致人员配置更重要的是保留内部软件工程，集成和运营。需要大量投资来建设架构，工程，测试，**DevOps**，集成，互操作测试，故障修复团队和运维支撑团队。此外，运营团队需要进行设计，部署，监控，生命周期管理，中断/修复支持等工作。除了投资和有些复杂之外，用户还需要分摊这些投资。

最终问题是，100%的DIY做法能提供差异化吗，或者只是让人分心？

- **来自不同供应商的软件**：在此模式中，用户购买商用硬件并从各种供应商（闭源或开源）采购软件。用户将充当系统集成商，并自己负责运维。这个负担小于100%的DIY的案例，但仍然很大。除了所有的运维任务，**DevOps**，集成，测试，互操作测试，**bug**修复都是用户的责任。用户不得不善于在多供应商环境中查找问题，光确认是哪个供应商的问题可能都要花费数月时间。
- **外部系统集成商**：在这种模式中，用户将从一个供应商那里购买整体解决方案，但保留云运维管理。虽然与前两种模式相比，压力较小，但运维NFV云仍然是用户的责任。
- **托管服务**：在最后这个模型中，用户将从一个供应商处购买解决方案，然后要求他们（或另一个供应商）进行管理。从效率的角度来看，这是责任最低的选择，但可能更昂贵。

显然，这些选项没有正确的答案。不同用户也许有不同的答案。选择错误的模型实际上是增加而不是消减TCO。

开始旅程

尽管NFV转型仍处于早期阶段，但已经有一些来自用户的最佳实践（如KPN和德国电信在2016年SDN世界[大会](#)上的讨论）

1. **明确表达目标**：关注正确的目标非常重要，并通过内部沟通获得员工最大的支持：**A)** 专注于为更高价值的功能释放资源，而不是削减成本。如果员工觉得NFV威胁他们的工作，那么就不太可能得到他们的支持。**B)** 专注于NFV的收益，例如降低成本，增加服务，增加客户群，增加客户满意度，而不是将NFV视为蚕食现有业务。**C)** 除短期目标外，沟通长远愿景。**D)** 解释NFV不是可选活动。如果不进行NFV转型，将面临OTT竞争对手严重影响业务的风险。
2. **有组织地建立技能**：正如之前讨论过的那样，您不可能雇用到足够多的专家，所以你需要强烈关注于通过集中目前分散在不同团队的资源来建立内部技能。当然，这些团队应根据需要来补充外部技能，同时

认识到供应商不可能包揽一切。

3. **灵活而不是大爆炸**：等规范全部完成再进行完整的实现，对于**4G**及之前的规范并不罕见，但对于**NFV**，你需要采取灵活的哲学，而不是等每个细节都被确定。需要尝试一些小事情，并随着时间纠正和演进。
4. **规划好路径**：尽管模型驱动架构，**DevOps**，云原生**VNF**都很“酷”，但并不是每个用户都必须立即转入这些新技术中。事实上，有些路径是非常有效的：

| | 阶段 #1 | 阶段 #2 | 阶段 #3 |
|-------|---|---|---|
| 路径 #1 | 仅计算虚拟化 | 计算虚拟化 + SDN 控制器 | NFVI + SDN 控制器 + VIM + MANO |
| 路径 #2 | 计算虚拟化 + SDN 控制器 | NFVI + SDN 控制器 + VIM + MANO | |
| 路径 #3 | NFVI + SDN 控制器 + VIM + MANO | | |

上表不涉及**VNF**；即使使用**VNF**，用户也可以使用云托管，云优化或云原生**VNF**。类似地，用户可以把**DevOps**仅用于开发环境，也可以应用于所有阶段(**dev / test / staging / production**)。所以有很多路径达到目的。

5. **发现用例**：构建**20–50**个节点的**NFVI / MANO POC**，然后等待**VNF**工作负载，这的确太诱人。但是这种方法导致了孤岛云，因此行不通。真正可行的是找到一个真实负载的生产环境 – 理想情况下，没有关键任务 – 然后围绕它创建**NFVI / MANO**基础设施。从一个小的用例开始，这可能比较完美，这样你会有一些实时的收益！举个非电信例子，比如大众汽车，他们为西班牙的客户搭建了一个**OpenStack**云用于部署选配汽车配置的应用。

6. **寻求主管支持:** 对于NFV这类关键转型，自上而下的支持是至关重要的。没有主管的支持，NFV这样的措施很快就会失败。诸如KPN网络创新主管André Beijen以及德国电信副总裁Axel Clauberg等高管都表示支持NFV战略。
7. **专职团队：**组建一个专职新团队而不是给现有团队增加任务，似乎更容易成功。新团队也可以拥抱新的文化，流程和工具，而不是重用已有的。例如，KPN成立了一个新的虚拟化项目团队，通过志愿而不是分配来招募关键的有天赋的人才。
8. **知识传递：**初始团队需要像催化剂一样，将其新获得的技能，知识，思维，文化和过程转移到组织的其他部门。回到大众的例子，他们积极开展工作坊和技术日等活动，在整个公司内传播知识。

除了上述最佳实践外，作为一个开源社区，OPNFV也可以协助您进行NFV之旅。在下一章中，我们将介绍一下OPNFV。

3

这是OPNFV！

OPNFV不同于传统的开源项目，不能从传统开源项目的视点来理解，否则就会像观众看见超人在天空飞翔，却困惑于是鸟还是飞机。在介绍什么是**OPNFV**之前，让我们先来看看创建又一个开源项目的驱动因素。

OPNFV驱动因素

OPNFV的基本假设是，开源软件和协作技术是创建部分软件栈（如果不是整个软件栈）的有吸引力的方法（见本文第1章**ETSI**的定义）。由于不缺乏开源网络项目，这本应该是直截了当的，但事实却并非如此，当我们把这些开源项目应用到一个特定的用例比如**NFV**，就会面临**4**个非常有趣的挑战：

- **影响项目**：当涉及未来规划，大的开源项目通常会有两个特征：第一，没有中心产品管理或者类似机制去驱动具体功能，基本的决策都是以一种非常分散的方式制定。第二，这些开源项目通常面向多种用例，所以将开源项目引向一个具体的用例（如**NFV**）是一个挑战。
- **组合栈**：**NFV**需要集成多个开源项目的软件，没有哪个项目有义务保证所有这些软件是集成的和互操作的。因为这些上游项目从网络规模，网络和**IT**三个不同的角度来解决问题，基于他们来构建不同的组合栈是特别困难的。困难之处就在于将这三个观点融合在一起，并从三个方面进行最佳实践。

- **测试栈**：最后多个项目组合成的软件栈是必须经过测试的。虽然开源项目自己也会做一些测试，但是这些测试基本都限制在自己的领域，没有哪个项目会去测试这些新组合的软件栈，也没有任何一个社区去测试面向NFV的特性。
- **获取终端用户反馈**：除非一个开源项目针对特定用例，否则就很难获得用户的深度参与，而这对于项目的成功又是至关重要的。

综上所述，就是OPNFV产生的原因。

什么是OPNFV？

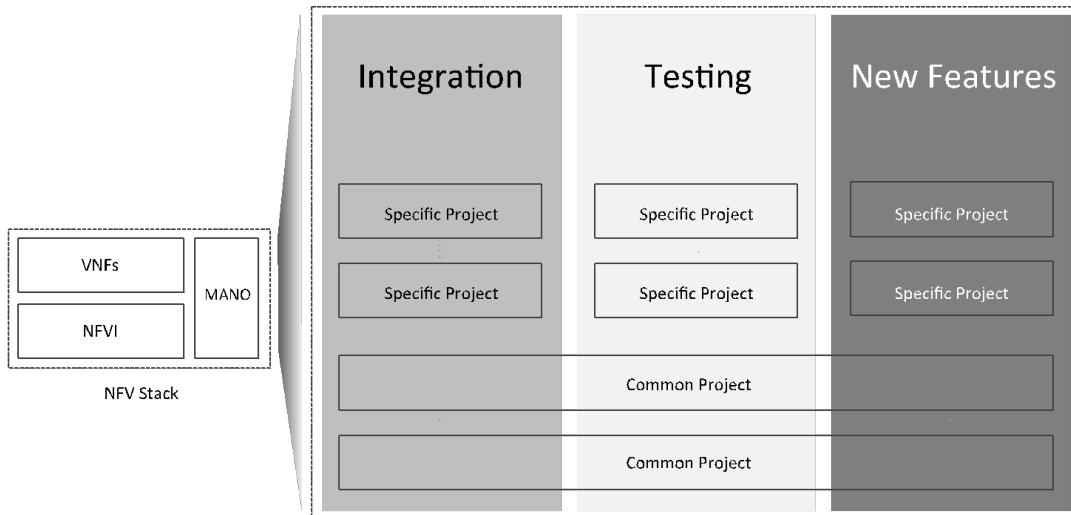
OPNFV是Open Platform for Networks Functions Virtualization的缩写，于2014年9月份成立，由Linux基金会主持。最初，OPNFV的宗旨是开发一个构建NFV功能的经过集成和测试的开源平台。根据OPNFV[网站](#)上的描述，当前OPNFV的使命是：

NFV开放平台（OPNFV） 助力跨各种开源生态系统的**NFV**组件的开发和演进。通过系统级的集成，部署和测试，OPNFV创建一个**NFV**的参考平台，来加速企业和服务提供商网络的转型。

OPNFV构建端到端的软件栈来支持经过功能和特性验证的**NFV**，建立敏捷参考方法（需求，文档和传播；持续集成，持续测试和持续交付），并为测试和验证**NFVI**及**MANO**产品和解决方案提供一整套的流程和支持工具。

OPNFV通过围绕以下三大支柱组织项目，有意识地将最终用户作为主要贡献者，来直接影响四个驱动因素。

OPNFV Project Pillars

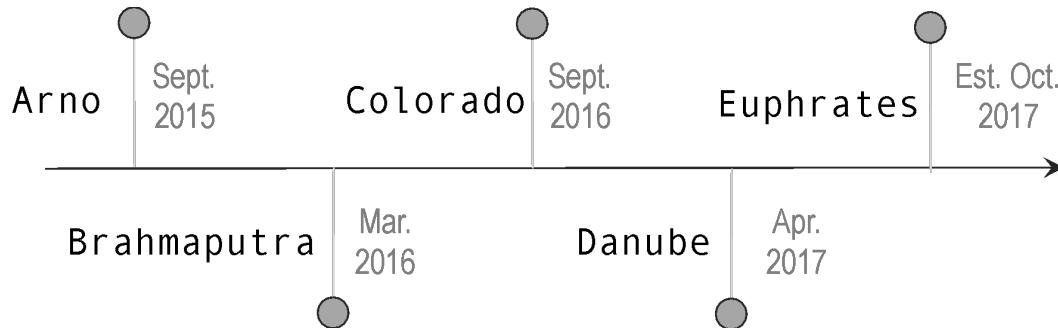


- **集成：**OPNFV通过集成各个开源项目来满足特定的**NFV**需求。
- **测试：**OPNFV通过各种**NFV**特定参数来测试整个软件栈。
- **新特性：**对于上游开源项目来说，OPNFV可以被视作作为**NFV**需求的载体。通过积极参与上游项目并独家提供**NFV**需求，OPNFV引导这些开源项目开发满足**NFV**需求的新特性。

OPNFV的技术工作是通过项目形式运作的。有一些项目跨越所有这三个支柱，比如CI/CD，安全和文档，还有一些项目只涉及到某一支柱。

当2017年4月份开始写这本书的时候，OPNFV有51个公司参与，项目多于45个，340多个贡献者，16个测试实验室和20000多次代码提交（数据来源：[OPNFV社区dashboard](#)）。为了确保有足够的最终用户参与，OPNFV的[EUAG](#)（最终用户专家组）组有30多个成员。迄今为止，OPNFV已经组织了3次Plugfests，两次世界范围的峰会。平均而言，OPNFV每六个月发布一次主要版本，版本号以河流命名。最新的版本多瑙河（Danube，即第4版）的亮点包括压力测试的引入，MANO的集成，数据面加速技术的支持以及CI和测试框架的显著改进。

OPNFV Release Timeline



开源系统集成

大多数人都以为，开源项目就是一群程序员自发的组织起来开发一段代码，而开源代码不需要专注于创建一个软件。**OPNFV**是一个集成项目，其工作包含测试，CI/CD，文档等。

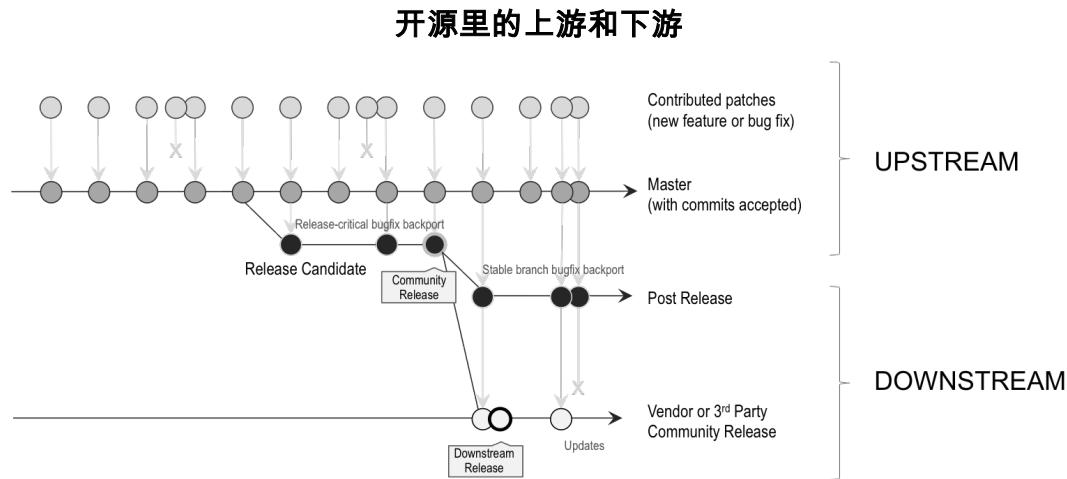
OPNFV是系统集成，以开源社区的方式来努力

OPNFV不做什么？

如上所述，**OPNFV**集成了各种上游开源项目，对其进行测试，进而影响其未来规划。但是，**OPNFV**不会重建一个增强的软件项目或者增加新的软件功能，所有这些增强都会在上游项目里完成。更进一步说，虽然**OPNFV**会做一些扩展测试，但是不会构建产品，换句话说，它不为特定应用规定任何特定的软件组合，这些工作是留给用户和厂商来做的。基于上述原因，你不必像传统的开源项目一样下载，安装和运行**OPNFV**。出于同样的原因，如果你是一个供应商，虽然你可能有一个通过**OPNFV**集成或测试的软件组件的商业发行版，或者某个集成产品包含了若干个来自**OPNFV**软件栈的组件，但是你不太可能拥有一个所谓的“**OPNFV**发行版”。

OPNFV是“中游”项目

软件开发借鉴河流的形态，通过“上游”，“下游”等术语来进行类比（非常有趣的是OPNFV的版本发布也是以河流来命名的），上游代码库是被下游代码库复刻的代码库。对开源来说，上游是所有贡献者工作的主库，下游是为了加入某些特殊功能或者用于缺陷修复的分支。



OPNFV是一个独特的项目，在上游和下游都有工作。集成和测试是下游活动，而新功能开发则是上游活动。基于这一点，OPNFV也可以说是“中游”项目。

开源的益处

既然OPNFV也是一个开源项目，我们来简要回顾一下开源项目都有哪些好处。

开源是对开放标准的补充

直到最近，开放标准仍是获得互操作性的手段。开放标准是最终用户增加可能的供应商以避免供应商锁定和促进创新的一种方式。然而，正如斯特林·佩林在“[2016年阅读报告](#)”中强调的那样，标准无法跟上当前创新的步伐，标准也往往无法确保真正的互操

作性。而且由于长期得不到客户反馈，标准有时候是徒劳无功的。我们绝对不是主张标准消亡论；我们所说的是，某些类型的标准可以被开源软件所取代。在这个意义上，开源软件是开放标准的补充。举个例子来说，就是OPNFV与标准机构如ETSI NFV ISG, IETF, MEF, TM论坛等的合作。

虽然开源本身不是一种商业效益，但它可以为最终用户带来的下列主要收益：

- **互操作性**：开放源代码保证了比开放标准更强的互操作性，因为不仅API是开放的，同时参考实现也是开放的，并且由多家公司，包括竞争对手共同创建。
- **创新速度**：开源项目的创新速度通常超过专有产品和标准。
- **更快的解决问题**：在多供应商系统中，故障排除成为一个复杂的多方努力的问题。而使用源代码，各方包括最终用户，都可以访问源代码和有能力修复错误。
- **影响未来规划**：在开源项目中用户的影响力高于专有产品。事实上，如果用户需求不被满足，他们可以参与社区贡献并直接实现他们想要的功能。
- **降低成本**：开源项目不是免费的。然而开源可以促使多个公司分担研发负担，带动更大的竞争力，从而节省成本。
- **透明度**：在当前安全隐患提高的时代，开源透明度为防范恶意功能提供保护。

多途径通往NFV

以下可以和OPNFV合作：

- **CORD**：Central Office Re-architected as a Datacenter (**CORD**) 是Linux基金会组织的另一个开源项目，作为新的多边项目的一部分正被纳入到OPNFV中。**CORD**定义了由硬件（行业标准服务器，白盒交换机和接入设备），以及**ONOS SDN控制器**, **OpenStack VIM**和**XOS VNFM**组成的整个堆栈。**CORD**的重点是开发，而不是集成和测试，这使得和OPNFV的合作非常吸引人。**CORD**通过使用相同的核心技术，不同的VNF和接入技术来实现三种不同的规格（企业，固定和移动）。
- **公有云**：对于NFV用例来说使用公有云通常是不切实际的，然而对于部分用例，如**vIMS**，公有云可能也是一种选择。此外，**NFV**的部分组件如**MANO**软件也可以托管在公有云中。

以下可以替代**OPNFV**：

- 专有虚拟化解决方案：诸如[vSphere](#) **ESXi**, 专有**MANO**或**SDN**控制器都是**OPNFV**的潜在替代品。

再一次，为什么关心？

正如第二章描述的那样，**NFV**转型更多的是关于流程，组织结构和技能获取，而不是技术。除了技术之外的在这些领域的深入渗透是**OPNFV**的独有特性。也正如这本书的其余章节将要描述的那样，模型驱动架构，**DevOps**和云原生软件根植于**OPNFV**的**DNA**之中。参与**OPNFV**将使你不至于误入歧途，浪费数年时间，同时节约几千万甚至上亿美元的成本。

从纯技术的角度讲，您可能仍然在想，如果**OPNFV**不是一个可以使用的即用型单一发行版的标准开源软件项目，为什么还要花费时间在它上面呢？是这样的，如果你正在通过使用开源项目来部署电信或企业网络级的**NFV**，考虑**OPNFV**至关重要。**OPNFV**，简而言之：

- **促进集成和互操作性**：开源项目不可能做到集成在一起开箱即用。**OPNFV**保证不同项目集成在一起工作。集成工作并不是微不足道的。即使用户或供应商选择不使用**OPNFV**组件，这些集成工作也将大大削减内部研发工作；这些资源可以应用到更高价值的活动。
- **驱动要求**：**OPNFV**将关键的围绕**NFV**的功能和代码转化为相关的上游项目。

简而言之，**OPNFV**就像一个待烤的披萨，它不是完全煮熟的即食的，但它已经做了很多复杂的耗时的工作，减少你食用的时间，或在这种情况下，减少**NFV**产品化的时间！如果想了解更多关于**OPNFV**的好处，请参见第10章。

{Create-Compose-Deploy-Test}.Iterate

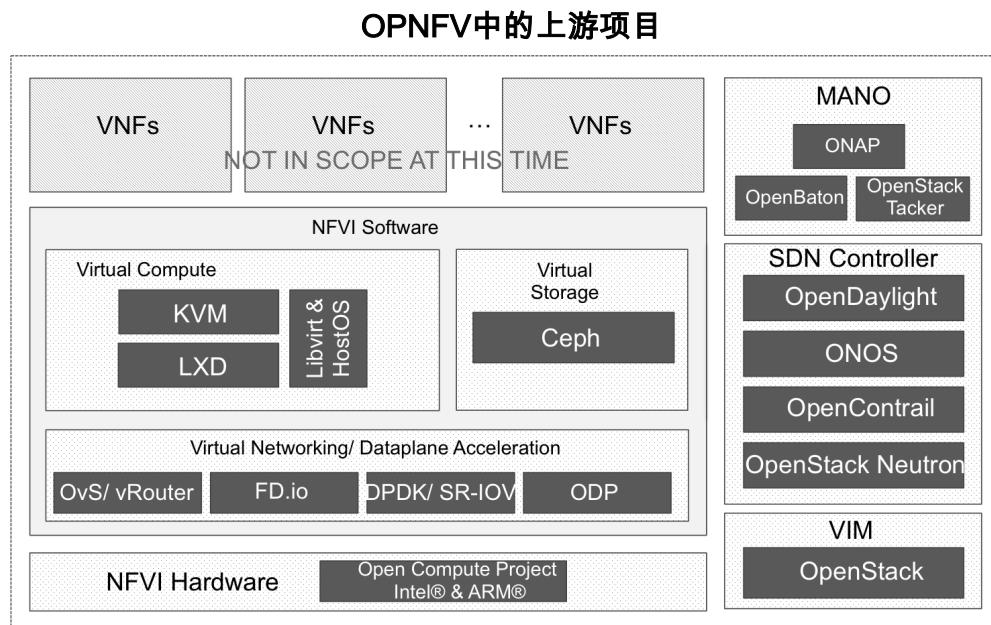
总结来说，**OPNFV**的目标是通过增加新的特性，集成各种上游开源项目来构建**NVF**软件栈，部署及测试他们。通过上游组件变更来触发整个循环的反复运行。

后续的章节会详细讲述这整个迭代的四个阶段：创建，组装，部署和测试。

4

OPNFV中的上游项目

如上一章所述，**OPNFV**是一个“中游”项目，许多项目源自于上游开源社区。**OPNFV**整合并测试这些上游不同项目的组合，使这些项目与**NFV**要求保持一致。当**OPNFV**社区确实需要修改代码以满足需求时，通常到相关的上游项目直接进行贡献。本章概述了**OPNFV**中使用的主要上游项目。



从上图可以看出，有些情况下，**OPNFV**为了相同的目的会集成多个软件项目。**OPNFV**通过这种方式，使最终用户能够有多种选择，而不是使用一种技术抛弃另一种技术。

在比较开源项目时，往往关注特性和功能。然而，社区本身及其构成、结构、管理、动力、资金和开发方法也同等重要，甚至加重要。在下面讨论中，您将看到这些因素在不同的项目中存在较大差异。

NFV管理和编排 (MANO)

当前与**OPNFV**集成的**MANO**项目有以下三个：**OPEN-O** (**ONAP**的一部分)，**OpenBaton**和**Tacker**。

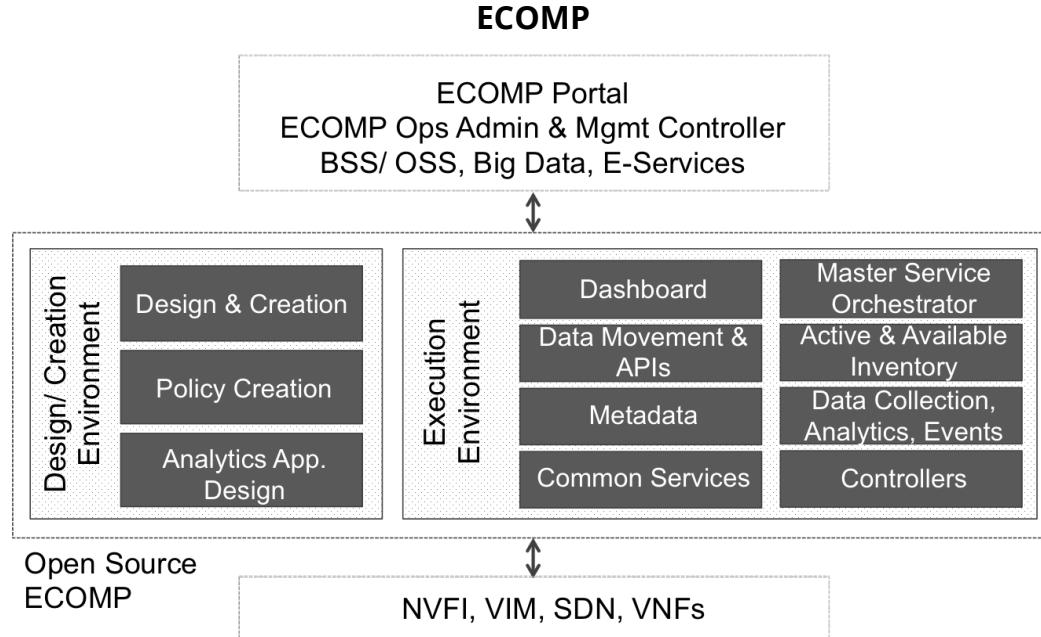
ONAP

开放网络自动化平台 ([ONAP](#)) 是一个Linux基金会下端到端服务编排项目。它是由**OPEN-O**和开源**ECOMP** (AT & T的**ECOMP**项目的开源版本) 两个项目合并而成的。AT&T和中国移动正在与其它成员一起推动**ONAP**的发展，其中白金会员包括**Amdocs, AT & T, 加拿大贝尔, 中国移动, 中国电信, 思科, 爱立信, Gigaspaces, 华为, IBM, 英特尔, Reliance Jio, 诺基亚, Orange, TechMahindra, VMWare**和中兴，还有很多银牌会员。

ONAP项目有两个方面比较特别的地方，首先，它最初就是由用户驱动的，而不是由供应商驱动的项目；其次，它通过使用微服务体系结构和敏捷开发方法，从一开始就体现了**Cloud Native**架构。作为敏捷实践的有力证明，它的代码超过800万行，却只用了300个开发人员1年半的时间就完成了开发！因为第一个融合版本还没有发布，所以下面我们分别了解一下两个项目。

ECOMP (Enhanced Control Orchestration Management and Policy, 增

强型的控制、编排、管理和策略) 更多的被地描述为一个**MANO ++**项目。 **ECOMP**使运行在云上的网络服务的设计和交付自动化成为可能。除了**SDN**任务的服务交付和自动化之外, **ECOMP**还可以自动执行许多服务健康状态检查, 性能管理和故障管理任务。

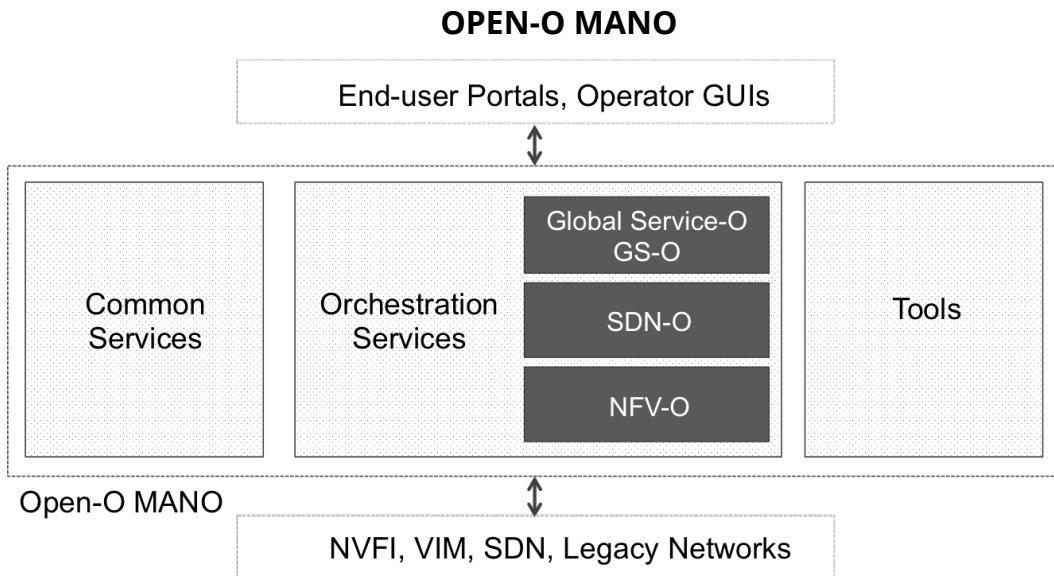


ECOMP在**ETSI MANO**架构之上增加了多项功能。它使用模型驱动架构的方法来处理整个服务交付生命周期。例如, 它为服务创建者提供了丰富的服务设计环境, 可以使用协作型目录驱动的自助服务设计工作室来构建服务。设计工作室还通过添加大量元数据来创建比**ETSI**架构更丰富的服务, **VNF**和基础架构描述。另外 **NFVO**和**VNFM**功能也得到加强, 可以更好地控制**NFVI**, **VIM**和**SDN**控制器, 以及更快地安装新的**VNF**类型。 **ECOMP**支持**YANG**, **TOSCA**, **OpenStack Heat**等建模语言。最后, 该项目包括**FCAPS** (故障, 配置, 计费, 性能和安全) 功能, 可以更好地控制闭环自动化 (在**ETSI**架构中, 该功能驻留在**EMS** – 网元管理系统中)。虽然**OpenStack**是支持的主要**VIM** (虚拟基础架构管理器), 但开源**ECOMP**也可以扩展到其他**VIM**。

[OPEN-O](#)是Linux基金会下开放的编排项目, 它结合了**NFV MANO**以及**SDN**与传统网

络上的互联服务编排。 **OPEN-O**采用模型驱动自动化方式，采用标准建模语言**YANG**（用于网络设备）和**TOSCA**（用于服务）。

OPEN-O架构上分为三个主要的编排功能：通用服务编排器（**GSO**），**SDN**编排器（**SDN-O**）和**NFV**编排器（**NFV-O**），以及一系列通用服务。**OPEN-O**建立在具有可扩展性的微服务架构之上，并支持多**VIM**、多**VNFM**、多**SDN**控制器和传统网络及网元管理系统的模块化设计。



在[Opera项目](#)中，OPNFV将OPEN-O项目与[Juju](#)及可用于VNFM管理的[Tacker](#)进行整合。

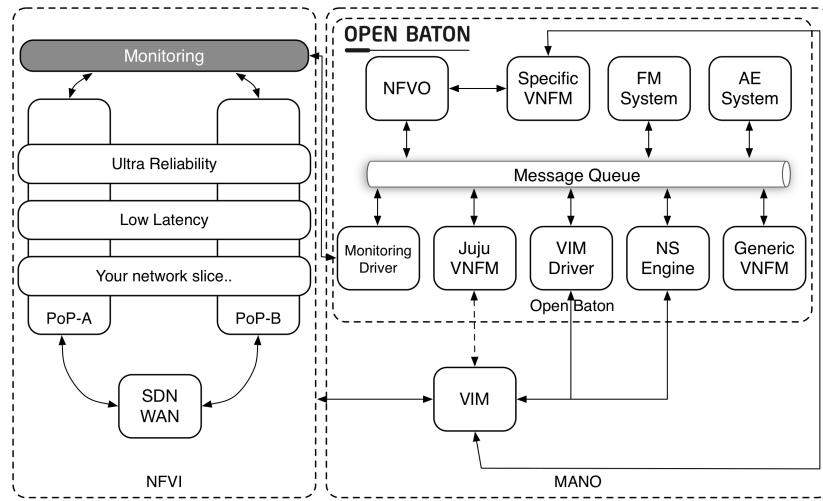
OpenBaton

[OpenBaton](#)是德国最大的研究小组[Fraunhofer Fokus](#)开发的MANO项目，其最初的设计就考虑了模块化、可扩展性和互操作性。

现在，在其第三个版本中，基于[OpenBaton](#)你可以只使用几个模块如**NFVO**和消息传递总线 – 或整套组件，如故障管理系统、自动弹性伸缩引擎、通用**VNF**管理器（**VNFM**）或[Juju VNFM](#)。它还包括对多**VIM**（最主要是[OpenStack](#)）的支持；对**Docker**的支持也已被证明是可行的。此外，[OpenBaton](#)支持网络切片，这对于5G很重要，在**VNF**中也有广泛的应用场景。

OPNFV已将OpenBaton集成到编排项目的整个平台中。

OpenBaton MANO

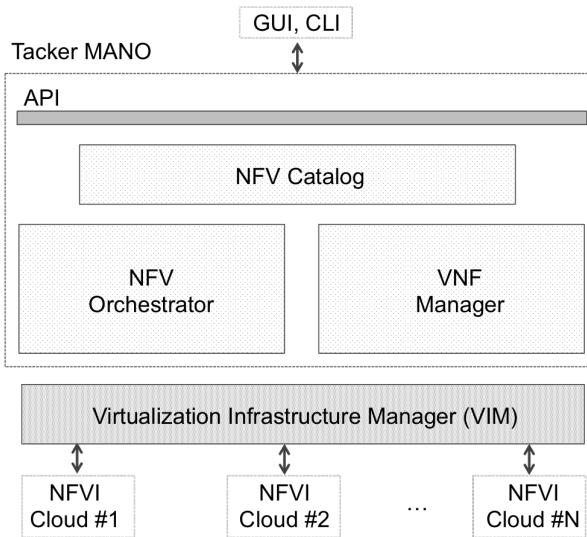


OpenStack Tacker

Tacker是一个官方的OpenStack项目，最开始仅作为一个通用的VNFM，但是现在已经扩大了其范围以提供NFVO，使其成为一个完整的MANO项目。

作为OpenStack的一部分，Tacker与其他OpenStack组件（如Nova, Horizon, Heat, Keystone等）进行了很好的集成。社区Tacker主要贡献者包括博科，Infinite, NEC, 99cloud, 华为, Imagea, 诺基亚和中国移动。因为可以仅为VNFM来使用，所以Tacker它也可以与其他开放的端到端服务编排项目（如OPEN-O和ONAP）共存。

Tacker MANO



除了这三个MANO项目之外，原来来自Telefónica的Open Source MANO ([OSM](#)) 现在在ETSI下，可以随时集成到OPNFV中。

虚拟化基础设施管理器 (VIM)

VIM负责创建和管理虚拟计算和虚拟存储等基础资源。VIM可以使用SDN控制器（见下一节）来创建虚拟网络。（严格来说，SDN控制器是VIM的一部分，但是为了清楚起见，我们把这些组件分开说明。）

OpenStack

当前，OPNFV使用[OpenStack](#)作为其主要VIM。在NFV的背景下：

OpenStack® is open source software for managing telecommunications infrastructure for NFV, 5G, IoT and business applications. Global telecoms including AT&T, China Mobile, Orange, NTT DOCOMO and Verizon deploy OpenStack as an integration engine with APIs to orchestrate bare metal, virtual machine and container resources on a single network. OpenStack is a global community of more than 70,000 individuals across 183 countries supported by

the OpenStack Foundation. Visit the Telecom and NFV [page](#) to learn more.
 (Source: OpenStack Foundation).

OpenStack是世界上最大的开源项目之一，从最初到现在已经有七年了，其第十五个版本Ocata是由来自285个组织的1,925个社区成员开发的。赞助商包括AT & T, Canonical, 华为, IBM, Intel, Rackspace, Red Hat和SUSE等白金会员以及大量的黄金会员及其它类型的赞助商。在最近的“Heavy Reading”[调查](#)中，全球86%的电信公司表示，OpenStack对于成功至关重要，而超过60%的用户也在使用或测试OpenStack for NFV。也许更有趣的是，超过21%的受访者计划将OpenStack作为OPNFV的一部分！

OpenStack本身是一个大型项目，下面有许多子[项目](#)。每个项目的详细描述不在本书的范围之内，下面只列出了核心项目。需要说明的是，目前有一些OpenStack项目虽然与NFV有关，但并未纳入OPNFV；最后一列显示该项目是否已和OPNFV集成。

OpenStack 核心项目

| 服务类型 | 项目 | 是否已和OPNFV集成？ |
|---------------------|----------|--------------|
| 计算 | Nova | 是 |
| 块存储 | Cinder | 是 |
| 网络(完整的SDN控制器或仅API层) | Neutron | 是 |
| 认证 | Keystone | 是 |
| 镜像服务 | Glance | 是 |
| 对象存储(API层和软件定义存储) | Swift | 可选 |

OpenStack包括许多可选项目，其中有一些与NFV相关。[OpenStack Project](#)

[Navigator](#) 列出了每个项目，并列出了每个项目年龄、成熟度、使用率等。年龄表示项目进行了多长时间的开发，使用率反映了通过双年度用户调查得出的生产部署的百分比。还有几个指标和成熟度相关，包括滚动升级支持、漏洞管理支持和文档。这些指标可以作为您进行项目评估的依据。一般来说，核心项目比可选服务更成熟，因此作为用户，您需要慎重选择。

和NFV相关的OpenStack可选服务

| 服务类型 | 项目 | 是否已和OPNFV集成？ |
|---------------------------|--------------------------------|--------------|
| 告警服务（基于测量的告警、通知） | Aodh | 是 |
| 编排 | Heat | 是 |
| 管理 | Congress | 是 |
| VNFM/ MANO | Tacker | 是 |
| 根因分析(和OPNFV doctor相关) | Vitrage | 是 |
| 资源预留即服务(和OPNFV Promise相关) | Blazar | 是 |
| 多区域跨Neutron网络自动化 | Tricircle | 是 |
| 集群服务 | Senlin | 是 |
| 仪表盘 | Horizon | 是 |
| 遥测 | Ceilometer (OPNFV Barometer使用) | 可选 |
| 工作流服务 | Mistral | 可选 |
| 监控即服务 | Monasca | 可选 |

| | | |
|------------------------------|------------|---|
| 时列数据库即服务 | Gnocchi | 否 |
| DNS服务 | Designate | 否 |
| 分布式备份、恢复和容灾即服务 | Freezer | 否 |
| 数据保护即服务 | Karbor | 否 |
| 分布式SDN控制器(可替代Neutron SDN控制器) | Dragonflow | 否 |

除可选服务外，OpenStack社区还有许多社区孵化项目，其中一些也与NFV有关。

和OPNFV相关的OpenStack社区项目

| 服务类型 | 项目 | 是否已和OPNFV集成？ |
|--|-----------------|--------------|
| 用于NFV网络服务的模型驱动、可扩展框架（可以替代Neutron API层）和OPNFV NetReady相关 | Gluon | 是 |
| 多区域部署的集中式服务，与OPNFV中multisite相关 | Kingbird | 是 |
| 基于YAML的TOSCA simple profile解析器，与OPNFV中parser相关 | TOSCA-Parser | 是 |
| 将TOSCA simple profile转换成Openstack中使用的HOT文件，与OPNFV中parser相关 | Heat-Translator | 是 |
| 综合网络服务编排 | Astara | 否 |

| | | |
|--------|-----|---|
| 基于组的策略 | GBP | 否 |
|--------|-----|---|

最新的**OpenStack**版本**Ocata**包含一些重要的**NFV**新功能，我们将在本书中陆续介绍这些主题。关键一点是**OpenStack**已将**NFV**视为一个重要的应用场景，并将**OPNFV**的需求包含在**OpenStack Ocata**发行版中，主要体现在**Nova**、**Vitrage**、**Congress**、**Neutron**、**TOSCA-Parser**和**Heat-Translator**等项目的增强上，同时**DragonFlow**和**Tricircle**也作为可选服务包含在**OpenStack**中。

OpenStack之外的VIM

除了**OpenStack**之外，**Danube**版本还试验性地支持基于[Kubernetes](#)（一个开放源码容器编排项目）的**VIM**。用官方的话来说：

Kubernetes is an open source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery. Kubernetes builds upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community.

Kubernetes使容器化的方式部署**NFV**成为可能。在**Danube**版本中使用的**Kubernetes**没有与任何**SDN**控制器进行集成，也不与任何其他以**NFV**为中心的项目集成。不过，我们预计与**Kubernetes**更深层次的集成将会在后续版本持续进行。**OPNFV OpenRetriever**项目就是一个旨在为**Euphrates**版本提供**Kubernetes**集成和测试的项目。

软件定义网络(SDN)控制器

SDN控制器主要是以软件的方式定义物理网络和虚拟网络的控制平面。在**SDN**之前，使用路由协议对交换机或路由器进行配置，而这些协议不允许对交换机或路由器进行较细粒度控制。**SDN**控制器通过北向接口与**MANO**和**VIM**连接，并通过南向接口与物理和虚拟交换机及路由器相连。当前虽然北向接口尚未标准化，但南向接口已经存在多种标准，如**OpenFlow**、**OpFlex**、**Netconf**、**P4**和**ovsdb**等

SDN控制器负责建立**overlay**网络， **overlay**网络是建立在为虚拟机提供连接的物理网络之上的，并可以通过路由器或网关与外部进行通信。

使用**SDN**控制器会带来如下好处：

- 整个控制平面是集中式的，这样易于管理，不过为了**HA**和扩展性原因依然可以采用分布式方式部署
- 控制策略是集中式的，所以冲突往往很容易解决
- 通过集中式控制平面能快速部署与网络有关的变更，可使用脚本和其他可编程方式进行管理
- 可以基于事件的方式对网元进行配置

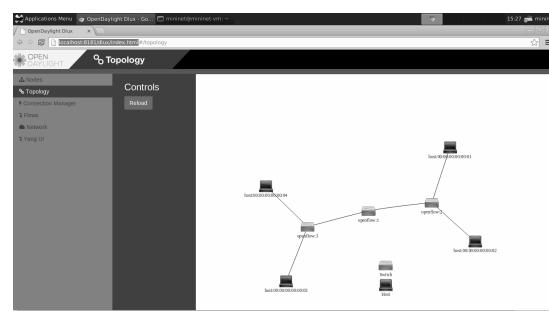
OPNFV集成了5种**SDN**控制器

OpenStack Neutron

OpenStack **Neutron** 既是**API**转换层又是**SDN**控制器。如果仅用作**API**层，则使用核心插件通过其对应的北向接口连接到第三方**SDN**控制器。如果用作**API**层和**SDN**控制器，**Neutron**通过驱动程序管理OVS（参见下面的**NFVI**虚拟交换机）和外部物理交换机。通常，在复杂环境中**Neutron**不用作**SDN**控制器，而是仅用作**API**转换层。另外，如果没有被**Neutron**的双重特性所迷惑，那么该项目实际上也有一些传统上被认为是**VNF**的服务插件，例如负载均衡器（**LBaaS**），防火墙即服务（**FWaaS**）及VPN即服务（**VPNaaS**）等。

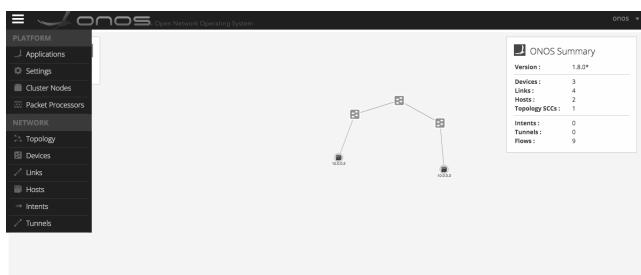
OpenDaylight

和OPNFV一样，**OpenDaylight**（ODL）也是**Linux**基金会下面的项目。它是



一个完整的模块化SDN控制器，可用于多种场景，如NFV，IoT和企业级应用。ODL为管理虚拟和物理交换机支持多种南向接口（OpenFlow，Netconf和其他协议）。对于OpenStack或其他协调层的北向接口，ODL使用YANG（标准建模语言）模型来描述网络，各种功能和最终状态。ODL社区规模庞大，博科，思科，爱立信，惠普，英特尔，红帽等都支持该计划。

ONOS

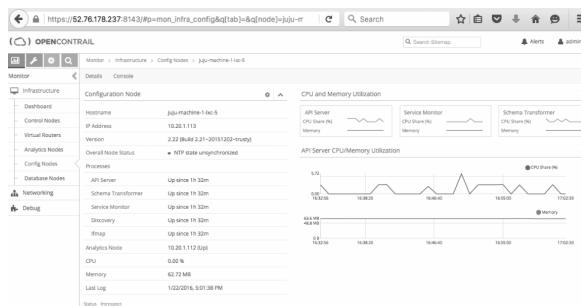


ONOS是一种模块化的SDN控制器，专为服务商提供电信级的可扩展性、高性能和高可用性而设计。ONOS的核心是分布式的，因此可以水平扩展。北向接口基于具有全局网络视图的框架，南向接口包括OpenFlow和Netconf，以便能够管理虚拟和物理交换机。ONOS由

开放网络实验室（ON.lab）管理，该组织由17个主要成员组成，其中包括AT & T，中国联通，Comcast，Google，NTT Communications，SK电讯和Verizon等运营商，同时也包含许多技术爱好者。

OpenContrail

[OpenContrail](#)是由[Juniper](#)开源的**SDN**控制器。它针对企业和服务提供商的云和**NFV**用例。与其他两个**SDN**控制器不同，[OpenContrail](#)使用**BGP**和**Netconf**作为主要的南向接口来管理物理交换机和路由器，而**XMPP**则用于管理计算节点中的虚拟路由器。鉴于选择**XMPP**而不是**OpenFlow**，[OpenContrail](#)不与**OVS**进行互操作，但在每个计算节点上安装**vRouter**。对于**overlay**，[OpenContrail](#)利用了通过**GRE**、**UDP**或**VXLAN**的**MPLS**协议。[OpenContrail](#)架构有三种类型的控制器，它们都可以水平扩展：控制节点集群，配置集群和分析集群。



OVN

| An Example | | |
|----------------------------------|-----------------------|-----------|
| Chassis (ovn-controller) | | |
| Name | Encap | IP |
| HV1 | Geneve | 10.0.0.10 |
| HV2 | Geneve | 10.0.0.11 |
| Bindings (ovn-controller) | | |
| Name | Chassis | |
| LP1 | HV1 | |
| Pipeline (ovn-northd) | | |
| Datapath | Match | Action |
| LS1 | eth.dst = AA | LP1 |
| LS1 | eth.dst = BB | LP2 |
| LS1 | eth.dst = <broadcast> | LP1,LP2 |

的问题。

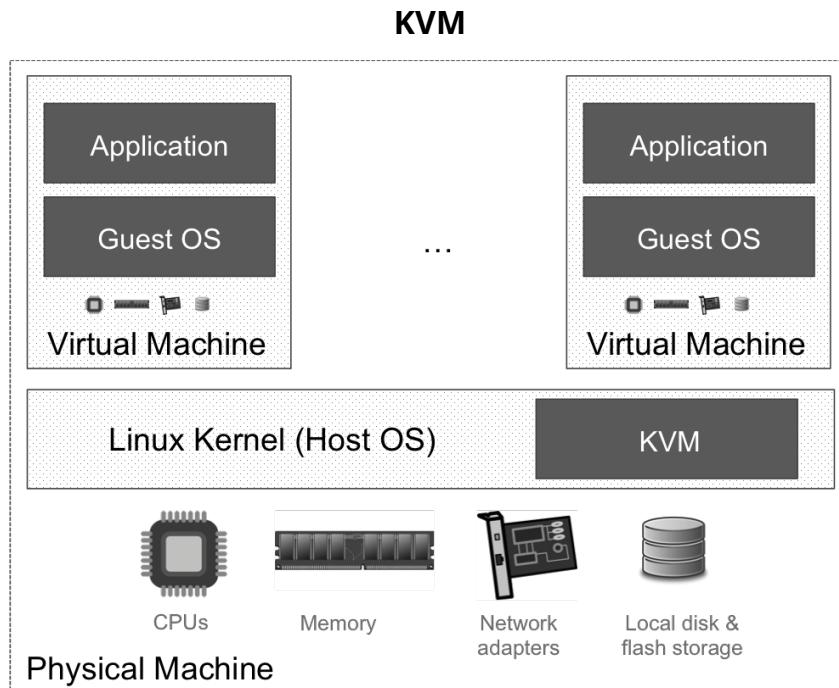
[OVN](#)即开放虚拟网络。它由开发[OVS](#)的原班人马开发。该项目旨在添加虚拟网络抽象，如虚拟**L2**和**L3 overlay**和安全组。[OVN](#)与上述项目不同，因为它仅局限于虚拟网络，而不是成为一个完整的**SDN**控制器。所以，它并不试图解决管理物理交换机的问题。通过这种方式，代码更轻量，更集中，仅解决小范围

NFVI 计算

此层提供运行虚拟机或容器的数据面。这些计算实例由[OpenStack Nova](#)调度，即创建和删除。

KVM

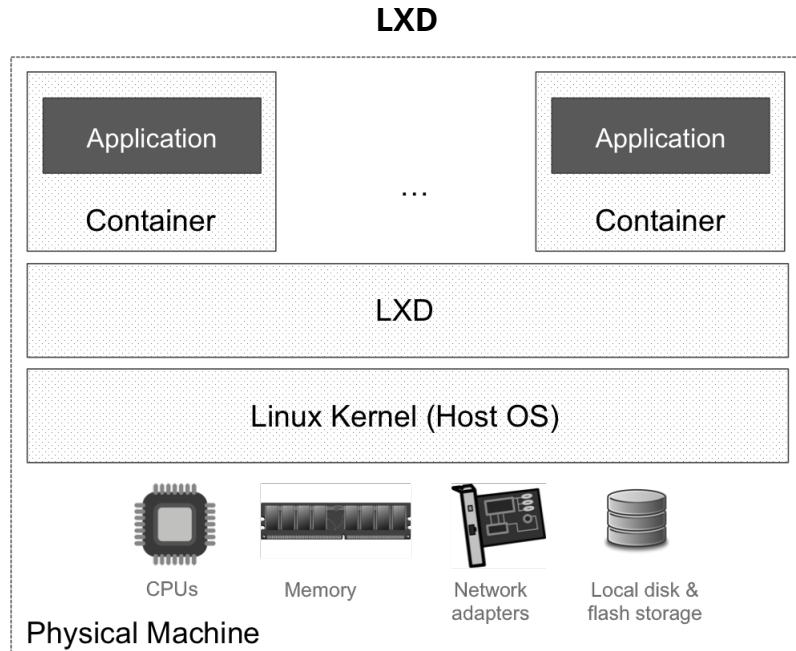
基于内核的虚拟机 (**KVM**) 是一个虚拟机管理程序，它是Linux内核的一部分，可以用它来创建虚拟机。



KVM提供的虚拟机，看起来就像给操作系统和相关的应用程序提供了一台物理机 – 它具有虚拟CPU，内存，网络端口，SSD或HDD存储。由于每个虚拟机都有自己的操作系统实例，因此隔离和安全特性几乎与单独的物理机一样好。**KVM**是一个成熟的，拥有10年历史的项目。

LXD

OPNFV还集成了**LXD**，一种来自**Canonical**的系统容器技术，在框架级别，**LXD**类似于**Docker**。它使用与**Docker**相同的底层隔离机制来提供容器的隔离。



容器虚拟化操作系统而不是整个机器。因此，容器比虚拟机轻量得多，但由于共享同一个内核而遭受隔离和安全问题。为此，许多使用者在虚拟机中运行容器。鉴于其体积小，容器在基于云原生的微服务的软件架构中很受欢迎。

libvirt/ HostOS

OPNFV支持Ubuntu, CentOS和SUSE作为运行管理程序的主机操作系统。[libvirt](#)是一个虚拟化API，用于管理在主机操作系统上运行的管理程序。它一端与[VIM](#)通讯，另一端与[KVM](#)或[LXD](#)进行通信。

NFVI存储

根据OpenStack用户调查，[Ceph](#)是OpenStack部署中最受欢迎的外部块存储软件。因此，OPNFV也集成Ceph。Ceph软件定义存储，通过添加存储服务器水平扩展存储。该软件由Red Hat开放，现在有红帽，中兴，Mirantis，SUSE，XSky，Digiware和Intel的贡献者。据官网说：

Ceph is a unified, distributed storage system designed for excellent performance, reliability and scalability. Ceph's foundation is the Reliable Autonomic Distributed Object Store (RADOS), which provides your applications with object, block, and file system storage in a single unified storage cluster – making Ceph flexible, highly reliable and easy for you to manage.

云原生架构下，持久性块存储的需求应该随数据库即服务和文件系统即服务而减少；由于VNF不是cloud native的，所以持久性块存储对于高可用性就显得很有用，其中主–主或主 – 备VNF可以达到相同的效果。

NFVI虚拟交换

虚拟交换机和路由器提供虚拟网络或[overlay](#)层来连接虚拟机或容器，这些都由SDN控制器管理。

OVS

[Open vSwitch \(OVS\)](#) 是Linux基金会下面的项目。它始于2009年，广泛应用于各行业。它在主机Linux操作系统中实现了一个虚拟交换机，为虚拟机之间提供虚拟网络。

OVS是一种多层虚拟交换机，可以分布在多台物理机器上，包括安全性、监控、QoS、VLAN、VXLAN、GRE、NetFlow、sFlow、NIC绑定和自动化控制功能。参考实现可以通过OpenFlow进行管理。OpenStack中的Neutron、ONOS和ODL都支持OVS。

vRouter

OpenContrail 带有自己的虚拟网络技术，在虚拟机管理程序中运行的**vRouter**。它类似于**OVS**，也可以执行转发，因此名称为**vRouter**。**OpenContrail**依赖于**XMPP**来管理**vRouter**而不是**OpenFlow**。

FD.io

Fast Data项目 ([FD.io](#)) 核心引擎是来自思科的矢量分组处理引擎 (**VPP**)，是**OVS**的高性能替代品。

VPP并行处理多个数据包，而不是一次一个。这扩展了整个数据包集合的查找和计算开销，从而提高了效率。**VPP**公开了一个高性能的低级**API**，对于需要使用更高级别机制与**VPP**通信的软件堆栈，另一种名为**Honeycomb**的**FD.io**技术将通过**Netconf / Restconf**公开的**YANG**模型转换为**VPP API**。支持**Netconf / YANG**的控制器，如**ODL**，可以“挂载”蜂窝管理代理与**VPP**进行通信。

独立测试[表明](#)，当转发到2000个IPv4地址时，**FD.io**吞吐量比使用**DPDK**的**OVS**要好5倍（参见下一节），转发到20,000个IPv4地址时，其吞吐量要高出约39倍。**FD.io**也是一个**Linux**基金会项目，**FD.io**的白金成员包括思科，爱立信和英特尔，还有12个额外的金牌和银牌会员。

数据面加速

鉴于性能的重要性，以包/秒、延迟或吞吐量来衡量，需要特殊技术来优化数据平面性能。这个是**NFV**特有的要求，并和计算密集型的企业场景不一样。**OPNFV**集成了三种主要的数据平面加速技术。具有**Nova's Placement API**的**OpenStack Ocata**版本将能够更好地控制与特定数据平面加速资源相关联的虚拟机的设置。

DPDK和其他性能/扩展特性

DPDK是另一个**Linux**基金会下的项目，以加速数据包处理，它是一组绕过内核并提供轮询机制而不是基于中断的库。在最近的一项研究中，使用**DPDK**的**OVS**在平均吞吐量提高了75%。

该技术可在多处理器上使用，被英特尔®推广，并作为旨在加速数据平面的**Enhanced Platform Awareness (EPA)**技术的更广泛组合的一部分。**EPA**除**DPDK**以外的主要技术是巨页、**NUMA**和**SR-IOV**。巨页通过减少页面查找来提高**VNF**的效率，**NUMA**确保工作负载使用处理器本地的内存，而**SR-IOV**可以使网络流量旁路管理程序，直接转到虚拟机。

ODP

由**Linaro Networking**集团及其13个成员公司支持的**OpenDataPlane (ODP)**项目旨在为网络数据平面加速创建一套标准的跨功能**API**。硬件厂商热衷于增加加速特性，如安全性、交换卸载等。但是，没有用户想要使用专有功能。**ODP**通过给应用程序提供标准**API**来弥合这种鸿沟，供应商可以采用其特定的硬件加速功能。

NFVI硬件

OPNFV根据英特尔®和**ARM**®架构的专有和开源硬件测试其软件。一个显着的开源硬件项目是开放式计算平台 (OCP)。**OCP**与传统的开源软件项目不同，因为它只关注硬件！**OCP**由**Facebook**发起，将其效率、成本和功耗创新带到社区。社区在开源中设计计算服务器、存储服务器、网络设备和整个机架。**OCP**内的电信工作组专门针对电信运营商的需求。该组拥有新的19“机架级系统设计，适用于计算、存储、网络和GPU插槽。

范围之外

当前，**VNF**开发、**VNF**集成或**VNF**功能测试不在讨论范围之内；然而需要说明的是，一些测试确实利用了开源的**VNF**，例如**vIMS Clearwater**项目。此外，**OPNFV**

Samplevnf项目旨在使用模拟真实VNF行为进行基准测试和性能优化。

在下一章中，我们将介绍**OPNFV**如何收集各种项目的需求，并引导实现这些功能。

5

OPNFV上游贡献

OPNFV有几个项目，收集了各种上游开源社区以**NFV**为核心的需求，并对自身项目产生了影响，甚至直接向上游社区和需要的地方贡献代码。这些贡献都旨在虚拟基础设施上重建与当前在物理基础设施上提供的相同级别的服务保证，服务质量（**QoS**）和可用性。

典型的上游贡献项目有如下可交付成果：

- **需求收集**：用例，架构，需求分析等
- **文档**：发布说明，安装指南，用户指南，平台概述，配置指导等
- **上游贡献**：根据项目收集的需求与上游社区合作：定义蓝图和贡献代码

以下是所有在上游贡献的项目（OPNFV Danube发行版本）和每个项目的简要说明。相关项目的细节可以在OPNFV的[维基](#)页面上找到。这些项目都归属于以下几类：

- 服务可靠性和可用性
- 与上游项目集成
- **NFVI/VIM/VNF**部署和生命周期管理
- 文档和安全

项目详细说明

与服务可靠性和可用性相关的项目如下：

| 项目 | 描述 |
|---------------------------------|--|
| 可用性 (OPNFV的高可用性) | 该项目为电信级NFV场景中的高可用性(HA)开发API和需求。涉及三个不同层面的 HA ：硬件 HA ，虚拟基础架构 HA 和服务 HA 。项目中比较成功的一个例子是，在 Colorado 版本定义了一系列差距，这些差距部分在 OpenStack M 版本得以成功解决，其中一部分则推迟到 Newton 版本实现。 |
| Barometer (软件快速路径服务质量指标) | 当前有许多开源数据收集监控工具。 Barometer 围绕NFV获取来自NFVI层的统计和事件，以便检测故障并增强服务水平协议 (SLA)。 Barometer 把监控数据传输到更高级别的故障管理系统。该项目使用 collectd 以及一些扩展插件，插件囊括了从IPMI, BIOS, OVS和DPDK到平台监控。收集的信息可以使用各种接口报告给更高级别的工具，包括标准电信接口 (如 SNMP)。 |
| Doctor (故障管理) | Doctor 项目开发故障管理和维护系统，以确保VNF的高可用性（请参阅下一节），以满足运营商在使用物理网络功能时的习惯。 |
| VES (VNF事件流) | 不同于 Barometer 专注于VNFI， VES 项目使用通用的模型和格式收集VNF事件流，以实现VNF健康和生命周期的管理。目前，终端用户必须使用不同的标准（例如，SNMP与3GPP或CSV与XML）处理来自VNF的各种不同格式的事件和统计信息，以实现服务保证。这会增加工程成本，并对引入新的VNF引入时延。 VES 旨在解决这个问题，由三个组件组成： agent 服务，将源测量数据映射到 VES 事件格式； collector ，用于接收和显示事件的收集器以及支持这两个组件的 VES 数据模式。 |

旨在解决各种上游项目与OPNFV集成的项目包括：

| 项目 | 描述 |
|--|---|
| Forwarding Graph (基于 OpenStack的 VNF转发图) | 该项目计划与 OpenStack network-sfc 和 Tacker 以及 ONF OpenFlow 一起工作，以演示这种技术组合可用于动态设置 VNF转发图 (VNFFG) ，也称为服务功能链接。它进一步表明，不同租户的流量可以流经不同的服务链（即一系列 VNF ）。 |
| Movie (Model oriented virtualized interface, 面向模型的虚拟化接口) | 尽管 OpenStack 有一个以围绕 IaaS 的 API ，但 MANO 仍然有很多细化的工作要做。 Movie 项目正在创建一个更抽象的 VIM 北向接口 (NBI)，作为现有 OpenStack API 的替代方案，可以简化 MANO 在资源访问，连接生成，流量识别，策略操作等领域的工作。 |
| NetReady (Network readiness, 网络准备就绪) | 鉴于 NFV 首先是以网络为中心的工作负载，因此 NetReady 项目在了解当前 OpenStack 网络模型和 API 的特性缺失方面发挥重要作用，因为它们与电信级需求有关。诸如 L3 之类的需求， WAN 连接和对传统网络的支持等等，对于特定的 NFV 场景，意味着当前的 OpenStack 网络架构需要演进。除了 OpenStack Neutron ， NetReady 还与新的 OpenStack Gluon （模型驱动，可扩展框架的 NFV 网络服务）项目合作。 |
| NFV-KVM | NFV-KVM 项目侧重于 NFV 中的 KVM 虚拟机管理和开发需求，并与上游社区协作实现这一集成。通过使用实时 KVM ，社区在小包性能方面表现出10倍的提升。 |
| ONOSFW (ONOS 框架) | ONOSFW 项目为 ONOS SDN 控制器开发需求。 |

| | |
|--------------------------------|--|
| OpenContrail 用于OPNFV的虚拟网络 | 该项目开发 OpenContrail SDN 控制器的要求。【作者注： OPNFV 也为 OpenDaylight (ODL) SDN 控制器提出需求 ，只是 OPNFV 中没有专门的项目。】 |
| Open vSwitch for NFV | 该项目开发需求，与上游社区合作，并创建功能和基准测试 ，以帮助 OVS 在 OPNFV 中集成。主要聚焦于性能，将 OVS 从 内核态移动到用户态并利用数据平面加速， <u>使得</u> 小数据包吞 吐量提高10倍。 |
| Opera (OPEN-O集成) | Opera 项目为支持 OPNFV OPEN-O MANO 开发新需求，重 点是 API 和数据模型。使用 Juju 作为 VNFM 。 |
| Orchestra (OpenBaton集成) | Orchestra 项目为支持 OpenBaton MANO 开发新需求，重 点是 API 和数据模型。使用 Juju 作为 VNFM 。 |
| SDN VPN (SDN 分布式路由和 VPN) | 该项目在 OPNFV 中开发了 OpenStack BGPVPN 的需求。该 项目实现了 L3 网络服务与广域网 (WAN) 的集成。 |
| SFC (服务功能 链) | SFC 项目开发需求，文档和基础设施，将上游 ODL SFC 实施 项目集成到 OPNFV 。 |

与**NFVI/VIM/VNF部署，生命周期管理**相关的项目：

| 项目 | 描述 |
|-----------------------------------|--|
| Copper (NFVI 部署策略) | 用户期望服务得到 VNF 部署的调度，然后将其调度到虚拟基础 设施。用户对地理，分区，安全性或特定 HA 要求的亲和性的 要求可能在翻译给基础设施部署时丢失。 Copper 通过与两个 上游项目， OpenStack Congress (策略即服务) 和 |

| | |
|---------------------------|--|
| | OpenDaylight Group-Based Policy (GBP) 合作来解决这个问题。最终的目标是确保策略是 VNF 软件包或元数据的一部分，并确保这些策略在不同的基础架构管理者的每个层面上都被执行。 |
| Domino (模板分发服务) | VNF 和底层基础架构的集中编排并不是一直可行。例如，运营商网络可能跨地域，或者运营商可能会进行兼并和收购，从而导致多种不同的编排工具。这些情况需要一个自顶向下的层，使用描述服务模型和策略的模板，并将该模板分成每个本地编排和控制器工具的特定模板。这就是 Domino 项目范围。 Domino 将策略转换为 TOSCA ，并在考虑依赖关系时使用发布/订阅系统分发相应的模板。项目的范围包括定义功能， API ，测试/集成和调试/跟踪。 |
| ENFV (边缘NFV) | 在 vCPE , vPE , IoT 或其他场景中，在集中式云和边缘计算节点之间拆分 NFVI 是有意义的。这也称为雾计算。以这种方式拆分 NFVI 还是具有独特的挑战性。该项目的目标是在隧道，处理许多小型“数据中心”，远程节点监控，数据平面优化等领域中明确边缘 NFV 的要求。该项目与其他 OPNFV 项目合作，影响上游 OpenStack , ODL 和 OVS 的项目。 |
| Escalator (平滑升级) | 使用 NFV ，升级 NFVI 和 VIM 变得比固定功能盒更复杂。 Escalator 项目定义了主版本或小版本变更的平滑升级的需求，要求 VNF 正常运行时间不能被不合理地降低。该项目识别升级中的一些重要参数，如升级或回滚的最长持续时间， VNF 中断的最长持续时间以及准备升级的机制。最近，该项目还创建了测试用例来测试升级。 |
| Models (模型驱动的 NFV) | 不同的 VNFM 使用不同的技术和数据模型。 Models 项目的目标是传播和促进与 VNFM 相关的信息和/或数据模型的融合。活动范围从创建场景测试到比较不同 VNFM 的 VNF 软件包，并为服务和 VNF 提供端到端的生命周期。该项目还充当 OPNFV 与其他 NFV 标准机构（如 BBF , ETSI , MEF , OASIS , ONF , |

| | |
|-------------------------------|---|
| | <p>TM论坛等) 之间的沟通桥梁。</p> |
| Multisite (多站点虚拟化基础设施) | <p>NFV场景不同于企业场景，要求服务处于跨地域的分布式的基础设施上。这为OpenStack核心服务（如Nova, Cinder, Neutron, Glance, Ceilometer和Keystone）在跨站点的网络管理，多站点映像复制，全局和每个站点配额管理等领域提出了新的需求。该项目记录了这些需求，影响了一些上游项目。Multisite与OpenStack Kingbird（用于多区域部署的集中式服务）和OpenStack Tricircle（跨多中心部署中的中间网络自动化）集成。在OpenStack Ocata版本中，Nova的cells v2可以更轻松地扩展并同时管理多个计算环境。</p> <p>Multisite基于99.9 %的可靠性的基础设施提供电信级(99.999%)的可靠性服务。通过服务跨越多可用区域甚至地理区域，使服务满足电信级要求。</p> |
| Promise (资源管理) | <p>在某些情况下，MANO层可能希望在一定时间内预留计算，存储和网络资源。在这个项目出现之前，OpenStack没有这种机制。OPNFV Promise项目通过在OpenStack中创建Blazar（预订即服务）项目来解决这一特性缺失。</p> |

与文档和安全相关的项目：

| 项目 | 描述 |
|----------------------|--|
| Documentation | 此项目的主要任务是为每个发布版本创建所需的文档。此外，该项目还为所有 OPNFV 项目的文档开发准则和工具，并维护文档库。 |
| Moon (安全管理) | Moon 项目与上游的 OpenStack Keystone 和 Congress 项目合作，以提升 VNF 之间的隔离，保护和互动。该项目通过识别 OpenStack 和 ODL 中的功能缺失，并在鉴权，日志，网络 |

| | |
|--|---|
| | <p>增强，存储增强等上游项目中提供特性功能。 Moon项目还确保OPNFV Copper项目能够满足安全策略。</p> <p>OPNFV还获得了核心基础设施计划(CII)的最佳实践徽章。CII是一个Linux基金会项目，用于资助和支持关键的开源项目。他们的目标是增强核心项目，以防止诸如Heartbleed之类的安全漏洞。 CII发布最佳实践徽章来展示开源项目对安全性的承诺。</p> |
|--|---|

以上项目全部由全球小团队开发。每个项目都有一个项目技术负责人 (**PTL**)，核心提交者和贡献者。团队常年工作，通过协作工具（参见第6章），每周会议和IRC进行沟通。为了形成浓烈的合作氛围，每年与**OPNFV**峰会同时举行一次设计峰会。每年也举行多次**hackfest**，团队成员可以面对面地进行沟通协作。这些通常与**plugfest**是同时举行的（参见第8章）。

项目分析

要了解这些项目做得如何，可以在 [OPNFV的统计网站](#)上查询到各种统计数据。

OPNFV 项目分析



成功案例：OPNFV Doctor项目和OpenStack Vitrage项目集成

[Doctor](#), 一个成功的在上游项目中开发电信级功能的**OPNFV**项目。该项目由**NTT DOCOMO**公司于**2014年11月**提出，旨在使虚拟化的故障管理达到和现在物理世界相同的水平。项目提出之时，物理网络中的故障管理很好理解和实现。监控系统会检测到故障，并自动进行故障迁移。然而，虚拟化和基于云托管的**VNF**的结合打破了这一机制。

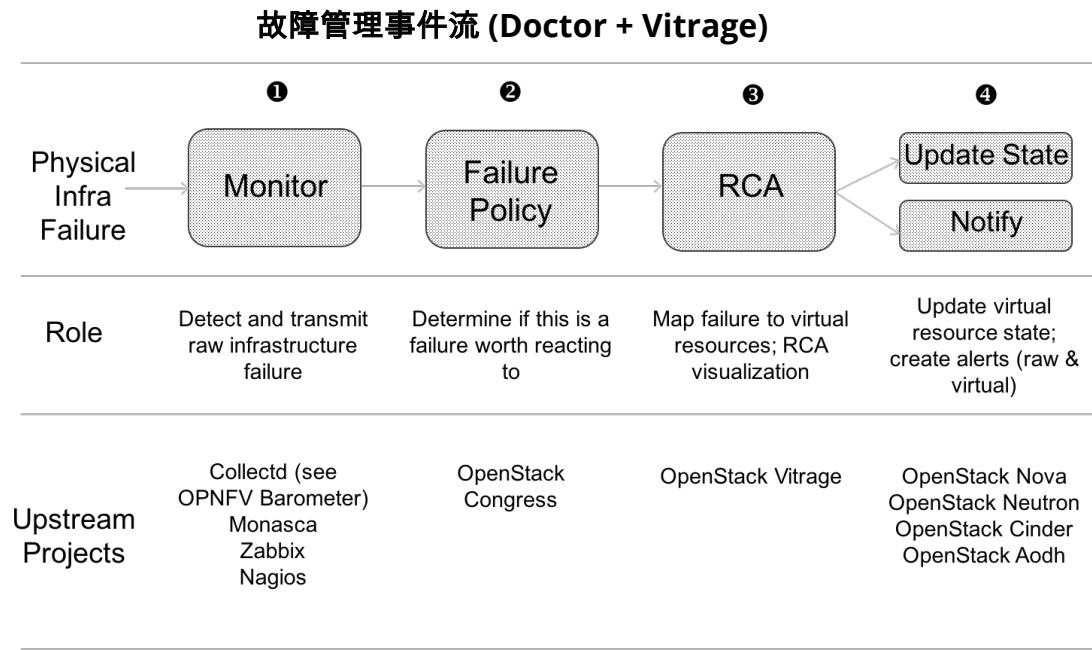
OpenStack当时的情况是通过软件轮询或者超时检测来发现虚拟机，存储或网络端口故障。在企业应用中这种延迟可以接受，但在电信级的**NFV**中是不可接受的。**Doctor**通过和众多上游**OpenStack**项目合作来解决这个问题，如**Nova**, **Neutron**, **Congress**和**Aodh**。

Doctor项目的目标很明确：一旦检测到物理基础架构故障，映射到受影响的虚拟资源，并在1秒钟内执行故障到换或任何其他操作。存在以下挑战：

- 定义哪些基础设施故障

- 映射到受影响的虚拟资源
- 合理地标记受影响资源的状态
- 向上级故障管理系统上报告警

Doctor也可以与OpenStack Vitrage (根因分析) 项目一起使用。OPNFV Doctor 和OpenStack Vitrage集成的解决方案如下：



监控

第一步当然是监控基础设施的故障。Doctor项目没有规定选哪种具体的监控组件，有许多不同的选择，包括collectd (参见OPNFV Barometer项目)。原始故障信息被发送到故障策略引擎处理。

故障策略引擎

原始故障意味着故障不是必须由MANO层处理。例如，如果有两个备用网口，一个失

败，运营商可以不将其视为失败。当然，网口必须被替换，但这是另外一个的话题。或者以内存利用率达到80%为例，某些运营商可能认为这是一个需要处理的故障，其他则不会。因此，需要一个从MANO层的角度来定义的策略引擎，确定原始故障是否确实是故障。**OpenStack Congress**是一个通过用户创建策略来实现的策略即服务的项目。**Doctor**提供**PushDriver**和**DoctorServiceDriver**的数据源驱动给**Congress**，使它能够评估来自上述**Monitor**的事件。

根因分析(RCA)-更新状态 - 通知

当策略引擎确定当前失败需要采取行动时，接下来有一系列的动作要发生。首先，需要确定受影响的虚拟机，存储和端口。接下来，必须适当地标记所有受影响的物理和虚拟资源（例如，错误，次优，**Down**状态等）。接下来，将告警发送到通知引擎，而且提供给用户的数据可视化要好。**OpenStack Vitrage**项目则覆盖上述所有操作。**Vitrage**从各种数据源提取数据，创建拓扑图，反映从物理资源到虚拟资源到应用程序的连接关系。接下来，用户只要提供一个模板，描述对事件做出什么反应。

一个简单的**Vitrage**的模板如下图所示([OpenStack巴塞罗那峰会演讲介绍](#))：

```

- scenario:
  condition: high_cpu_load_on_host and host_contains_instance
  actions:
    - action:
        action_type: raise_alarm
        action_target:
          target: instance
        properties:
          alarm_name: CPU performance degradation
          severity: warning
    - action:
        action_type: set_state
        action_target:
          target: instance
        properties:
          state: SUBOPTIMAL

- scenario:
  condition: high_cpu_load_on_host and host_contains_instances
  and alarm_on_instance
  actions:
    - action:
        action_type: add_causal_relationship
        action_target:

```

Raise alarm on
VM

Add causal
relationship

```

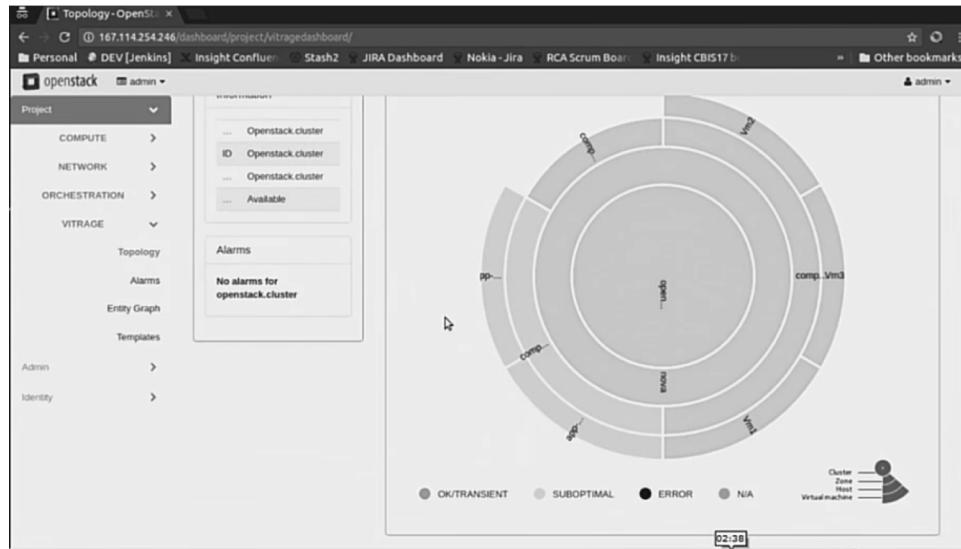
source: zabbix_alarm
target: instance_alarm

- scenario:
  condition: high_cpu_load_on_host
  actions:
    - action:
        action_type: set_state
        action_target:
          target: host
        properties:
          state: SUBOPTIMAL

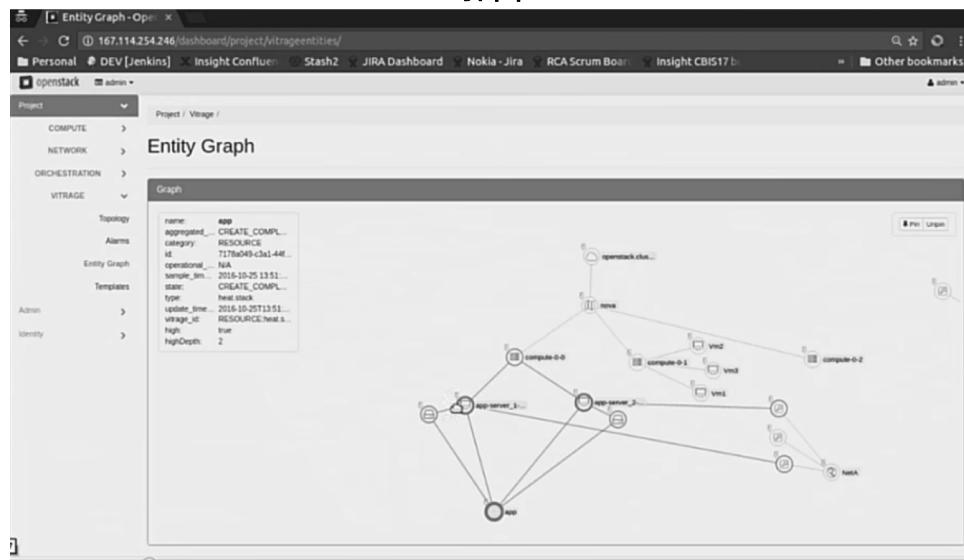
```

分受到影响，以及需要执行哪些模板。此步骤“推断”什么告警需要发送到管理层。**Vitrage**可以将故障映射到虚拟资源以及受影响的应用程序。同时，**Vitrage**还可以根据模板的要求适当地标记虚拟和物理资源的状态。最后，在**Vitrage**的**dashboard**上用户可以看到清晰友好的视图：

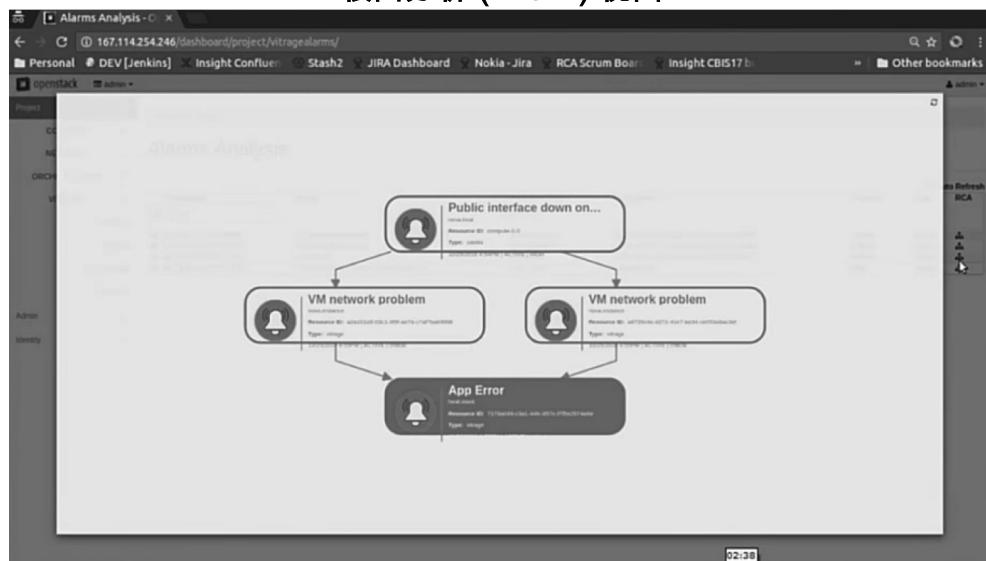
拓扑视图



扑图



根因分析 (RCA) 视图



告警视图

| TimeStamp | Name | Resource Type | Resource ID | Severity | Type | If Auto Refresh |
|---------------------|--------------------------------------|---------------|--------------------------------------|----------|---------|-----------------|
| 2016-10-25 04:55:00 | VM network | nova.instance | a2a101e8-03c1-49f4-ae7d-c7af7ba84898 | critical | vitrage | |
| 2016-10-25 04:55:00 | VM network problem | nova.instance | a07394e4-4272-41e7-bd34-ce053a8ac3ef | critical | vitrage | |
| 2016-10-25 04:55:00 | App Error | heat.stack | 7178ad49-43a1-44c0-857c-f795e2974a9d | critical | vitrage | |
| 2016-10-25 04:54:54 | Public interface down on compute-0-0 | nova.host | compute-0-0 | high | zabbix | |

OpenStack Nova, Neutron或Cinder项目将故障通知事件发送到其他组件，例如**OpenStack Aodh**项目（提供告警和通知），然后Aodh与MANO进行通信。

MANO层可采取适当的操作，例如触发故障倒换。在**OpenStack**巴塞罗那峰会期间，**vEPC**演示显示，网络故障发生后，响应故障的倒换在**500ms**内就完成了，正在进行的手机通话没有被中断。参见[视频](#)短片。

总之，这些在电信领域有着丰富经验的专家通过向**OPNFV**上游贡献，在**NFV**世界中重新创建相同的服务保证，**QoS**和可用性保证。在下一章，我们将介绍**OPNFV**的下一个支撑：持续集成。

6

OPNFV 持续集成

正如我们在第4章提及，**OPNFV**集成了一些上游项目。为了以自动化方式实现这一整合，**OPNFV**成立了三个基础设施项目³来支撑持续集成（CI）工作流：**RelEng**，**Pharos**和**Octopus**。这个三个项目对**OPNFV**项目的成功的至关重要。在这一章，我们将回顾这三个项目，然后详述它们是如何服务于集成与发布流程的。在第7章和第8章将讨论持续集成工作流是如何运用于部署与测试的。

OPNFV RelEng：发布工程

RelEng项目定义并支持使**OPNFV**获得成功所需的软件基础设施。它收集各个**OPNFV**项目的需求，安装配置所有的工具，软件自动化任务，脚本等，即自动集成，部署和测试所需的一切。同时，该项目提供使用指南，并为其他项目在使用软件基础设施上提供最佳实践支持。**RelEng**和**Linux**基金会基础设施团队提供地主要工具如下：

- 合作类：**JIRA/Confluence**
- 源代码管理和代码审查：**Git, Gerrit, Github**
- 持续集成/软件自动化：**Jenkins**
- 制品库：**Google云存储, Docker hub**

接下来进一步详细讨论每一个工具。

³ 开发项目和测试项目都需要使用这些基础设施项目
89

OPNFV JIRA

The screenshot shows the JIRA interface for the ARMBOARD project. The board is titled 'Backlog' and contains a single column labeled 'ARMBOARD-S1-E3'. The board shows 40 issues, all of which are 'Open'. The issues are listed with their titles and descriptions. Some issues have attachments and comments. The right side of the board has a summary of the issues, including counts for 'Danube 1.0', 'Colosse 3.0', 'Support for vMIG', 'Berkeley Infrastructure', and 'Upstream Amboard'. The bottom right corner of the board indicates 'Linked pages'.

合作-JIRA/Confluence

OPNFV使用Confluence [Wiki](#)进行社区协作，并使用[JIRA](#)进行错误和问题（史诗和用户故事）跟踪，并向贡献者分配问题。每个项目的[Wiki](#)页面，周例会（对外公开的），[JIRA](#)故事和下面描述的一些其他工具一起为提供了一种公开和透明的贡献方式。

源代码管理和代码审查

OPNFV使用了三种工具来进行源代码管理和代码审查：[Git](#), [Gerrit](#)和[Github](#)。

源代码管理-Git

OPNFV所有项目代码，脚本，模板文件和用于文档自动化的文档源码都保存在[Git](#)库。[Git](#)是由Linus Torvalds最初开发的分布式开放源版本控制系统，用于维护一个项目的主分支库（称为**trunk**）和管理所有的克隆分支（称为**forks**）。开发者可以从主分支（**trunk**）创建一个分支，在其分支上工作，然后将其变更提交到主分支。

OPNFV Git库示例

© 2016 Open Platform for NFV Project, Inc., a Linux Foundation Collaborative Project. All Rights Reserved.
Open Platform for NFV and OPNFV are trademarks of the Open Platform for NFV Project, Inc.
Linux is a registered trademark of Linus Torvalds.
Please see our terms of use, trademark policy, and privacy policy.

代码审查-Gerrit

在主分支提交变更需要一个批准的过程，通过**Gerrit**来管理这个过程。**Gerrit**是由**Google**开发的基于**web**的开源代码审查工具。贡献者使用**git push**或**git review**命令推送的所有更改，都会被多个评审员在**Gerrit**中查看和检查。评审人员还可以看到持续集成（**CI**）的构建结果和自动化验证测试运行的结果。评审人员提供**+2, +1, -1**或者**-2**的评分。**+2**表示明确的接受，同理**-2**表示明确的拒绝。**+1**或者**-1**的可以影响变更被接受，拒绝或者需要进行再次修改。

OPNFV Gerrit

源码控制管理镜像-Github

OPNFV使用[Github](#)镜像其Git库。

OPNFV Github镜像库示例

The screenshot shows the GitHub repository page for opnfv/yardstick. At the top, there are tabs for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Pulse, and Graphs. Below the tabs, it says "Mirror of the OPNFV Yardstick Project" with a link to https://wiki.opnfv.org/display/yardstick. The main area displays 963 commits, 3 branches, 6 releases, and 16 contributors. A list of commits is shown, all made by user kubi007, with details like commit message, file changes, and time of commit.

| Commit Message | File Changes | Time Ago |
|--|--------------|--------------|
| Merge "pylint fixes: remove redundant parens, fix comparison order" | api | 5 days ago |
| This Patch is used to add grafana config to opnfv dashboard for | dashboard | 7 days ago |
| Fix: yardstick-docker-build-push-master failure | docker | 12 days ago |
| Docs: Adding preliminary documentation for Network Service Benchmarking. | docs | 3 days ago |
| Adding sample yardstick.conf & pod files to help describe the topology | etc | a month ago |
| Fix installation dependency and authentication issue for fuel plugin | fuel-plugin | 2 months ago |
| Yardstick Plugin: add support for ssh login using key | plugin | 2 months ago |
| yardstick ping testcase yaml update | samples | 4 days ago |
| Merge " add scenario to solve apex ci fail" | tests | 4 days ago |
| Add support for Python 3 | third_party | a month ago |

软件自动化/持续集成

[Jenkins](#)是一款开源的软件自动化工具，通常用于自动化集成构建和测试。Jenkins被认为是强大的脚本的合并，是一个构建编译的实用工具，具有基于事件（例如Git库的变更）的触发活动的能力。OPNFV使用Jenkins来[执行](#)自动化持续集成，部署和测试。这些测试可能是简单的验证任务或某个更复杂的测试套（见第8章）。OPNFV将Jenkins任务也视为源代码；这些代码文件经过了开发，审查，测试和提交的。

Jenkins Job Builder (JJB) 通过自动化创建，测试，配置和维护OPNFV持续集成使用的Jenkins任务来实现这个过程。

OPNFV Jenkins

RelEng也提供其他的工具来收集、分析、搜索和可视化测试结果数据，例如 **Elasticsearch**、**Logstash**、**Kibana**和**Grafana**。

制品库-Google Cloud和Docker Hub

Git对于源文件是非常好的，但它不是存储大型制品（如详细文档，**Docker**映像，软件包，**ISO**文件等）的好地方。这些被存储在当前托管在**Google Cloud Storage**上的**制品库**中，或以**Docker**镜像的形式存储在**Docker hub**上。

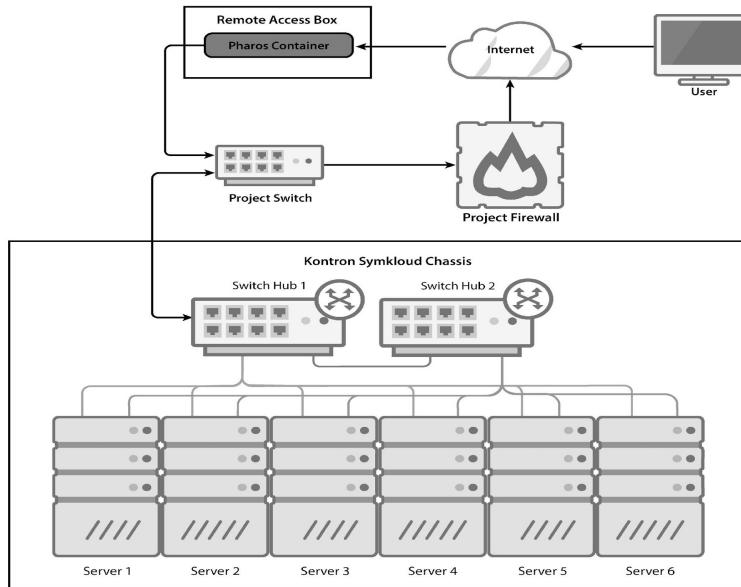
OPNFV Pharos：社区实验室基础设施

除软件基础设施外，**OPNFV**项目还需要相应的硬件基础设施项目进行开发，集成，部署和测试。这就是**Pharos**项目的目标。**Pharos**使用裸机服务器打造了多地域分布，多样化的社区实验室基础设施。目前，**Linux**基金会和北美，欧洲和亚洲的多个公司贡献了总共16个实验室。来自不同供应商，地理位置和**CPU**架构的多样性的实验室，是加强**OPNFV**软件的主要资产。

在**Pharos**上进行的苛刻测试令人印象深刻；对于任何特定版本，**Pharos**支持创建和销毁数以千计的**OpenStack + SDN**控制器集群，并针对其运行多个测试套件。这实

际上优于相应的上游社区进行的测试。毫无疑问，上游社区从OPNFV测试中获得巨大的价值。

CENGN Pharos 实验室⁴



Pharos项目涵盖了很多活动：

- **规格**：规定了每个环境或POD的最低要求。一个POD包括一台跳转服务器和用作控制节点，计算节点和存储节点的5台服务器。POD还需要一些网络，并规定了存储和交换的能力。一个实验室可以由一个或多个POD组成。实验室提供者需要发布关于实验室和POD的描述文档。规范性工作流继续为安全性，稳定性和生命周期管理制定额外的规范。
- **仪表板**：[Pharos仪表板](#)提供了所有实验室，资源和利用率的整体视图。

⁴ 此图得到了CENGN Pharos实验室许可。

OPNFV Pharos 看板

| Name | Slave Name | Booked by | Booked until | Purpose | Utilization | Status | Book | Info |
|---------------|--------------|--------------|--------------------------|-----------------|-------------|---------------|-----------------------|-----------------------|
| Huawei POD 12 | huawei-pod12 | jose.lausuch | April 17, 2017, midnight | SFC Integration | 14% | online / idle | <button>Book</button> | <button>Info</button> |
| Zte POD 1 | zte-pod1 | | | | 0% | online / idle | <button>Book</button> | <button>Info</button> |
| Huawei POD 7 | huawei-pod7 | | | | 0% | online / idle | <button>Book</button> | <button>Info</button> |
| Nokia POD 1 | nokia-pod1 | | | | 0% | online / idle | <button>Book</button> | <button>Info</button> |
| Arm POD 2 | arm-pod2 | | | | 0% | online / idle | <button>Book</button> | <button>Info</button> |
| Huawei POD 3 | huawei-pod3 | | | | 0% | online / idle | <button>Book</button> | <button>Info</button> |
| Huawei POD 5 | huawei-pod5 | | | | 0% | online / idle | <button>Book</button> | <button>Info</button> |
| Huawei POD 8 | huawei-pod8 | | | | 0% | online / idle | <button>Book</button> | <button>Info</button> |
| Zte POD 2 | zte-pod2 | | | | 0% | online / idle | <button>Book</button> | <button>Info</button> |

- 实验室即服务**: **Pharos**的关键目标是测试，而一些项目需要获得裸机资源才能开发。 **LaaS**以服务的形式向项目提供实验室资源。
- 有效验证器**: 一组检验实验室是否符合**Pharos**规范的工具。
- 虚拟机实验室**: 大部分实验室关注于裸机资源。但对于开发而言，基于虚机的实验室也应该被提供。

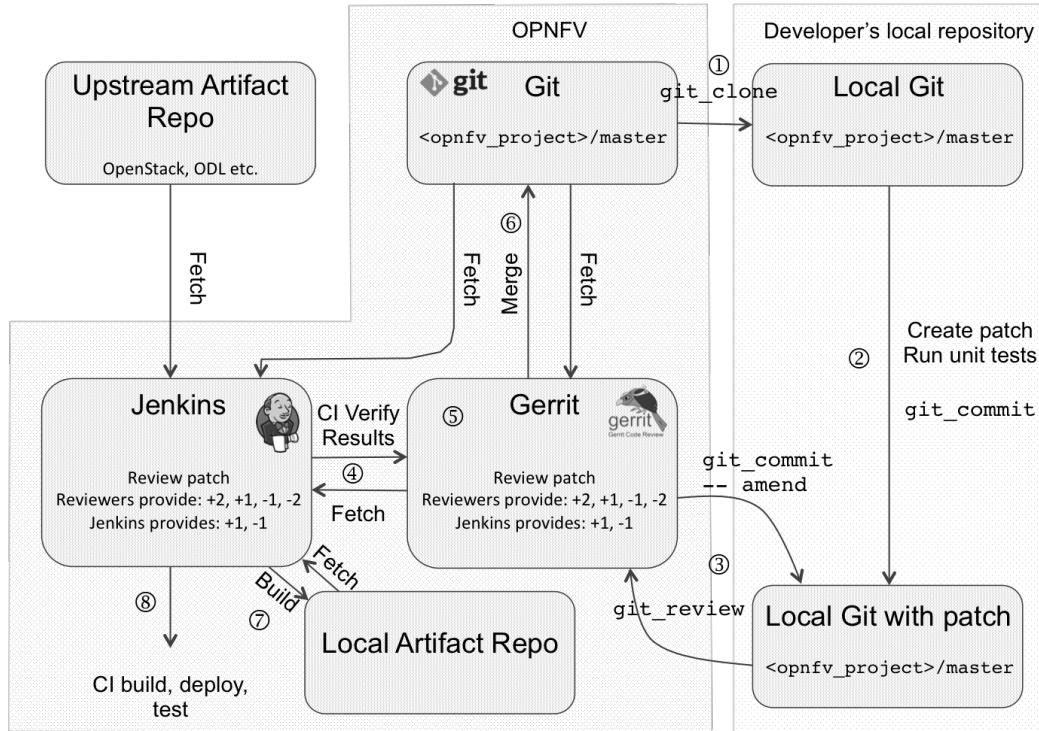
OPNFV Octopus: 持续集成项目

Octopus和**RelEng**项目创建了如下描述的**CI**工作流，它在各种**Pharos**实验室的POD上部署和测试了**OPNFV**软件。该工作流功能丰富，包括了构建，验证，合并，测试等工作。

CI 工作流

构建，测试和发布的工作流如下：

简化的CI工作流图



1. 贡献者克隆主分支到本地。
2. 贡献者在本地分支进行修改，新增代码或修改故障，完成本地的单元测试。
3. 贡献者提交补丁进行评审。
4. 通过**Jenkins CI**的验证任务来验证提交的补丁。
5. 评审人员可以检查该补丁和**CI**结果；该补丁可能被接受，拒绝或者打回再修改。
6. 如果该补丁被接受，**committer**中的一位将补丁合入到主分支，将触发**Jenkins**进行提交—合入的任务。
7. **Jenkins**也会创建本地制品库，例如用于测试工具的**Docker**容器。
8. 最后，**Jenkins**执行各种每日，每周或非周期性的测试任务。这些测试运行在使用从**Git**库，上游制品库（如**OpenStack**, **ODL**等）和本地制品库（如用

于测试工具的Docker容器) 构建的代码部署的OPNFV软件上。

那版本发布呢？

目前，OPNFV每六个月发布一个的正式版本。以这种正常节奏创造版本是开源项目中的常见做法。在接近发布日期时，重点从添加新功能转为故障修复。发布团队还会根据项目的成熟度和项目准备的充分度来决定是否包含或排除某些特定项目。

正如你所期望的那样，最终的目标是演变成一个持续的交付方式，可以以更高的频率进行发布，并且最终主分支在任何给定的时间点都是稳定的。

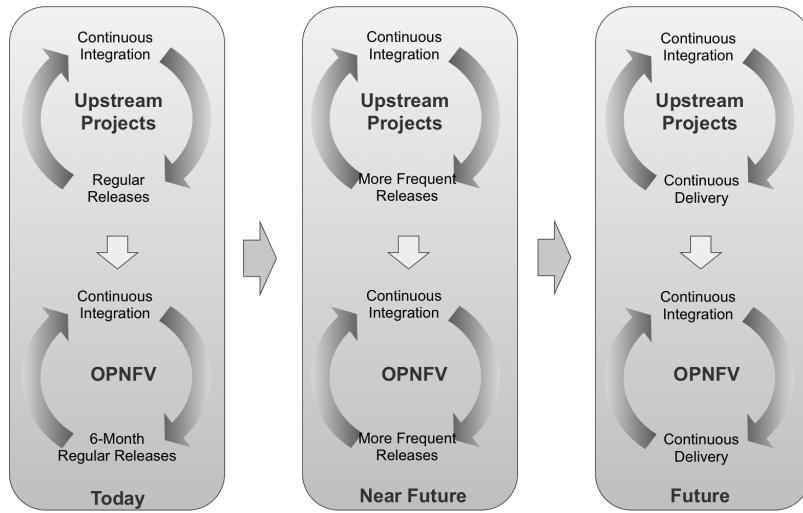
让CI / CD更加持续化

在OPNFV中使用DevOps和CI相当先进；但是，仍然有更多的事情要做。

一个重要的方面是安全性检查，其中每个提交将使用自动化工具检查安全违规。正在考虑使用工具对源代码文件的进行检查（例如Lint和漏洞扫描程序），二进制对象，可能包含密码，密钥或散列的字符串和许可证的缺失。

在消费上游项目时，通常的做法是使用常规的稳定版本。接下来，已经在一些确定的项目上，更为定期地甚至是每天去消费上游项目。在这样的过程里，OPNFV可以做两件事情，**i)** 通过健康检查为每个提交提供反馈，以及**ii)** 在最新的代码上进行每日和每周测试。OPNFV跨社区CI计划，或XCI，涵盖了最初针对OpenStack，ODL，FD.io和ONAP的这些新的流程。XCI的优势是更快地获得最新的创新和对上游社区的快速反馈。

CI/CD的现状与未来



这些改进是显而易见的，但是在幕后仍在进行不断的演变，以便在执行测试中更加清晰，提高资源的利用率，并帮助开发人员获得更多的洞察力和反馈。

总而言之，**OPNFV**集成了上游项目，用于运行不同的测试。为了实现这一点，**OPNFV**使用复杂的持续集成来确保获得最新的创新。到目前为止，我们还没有谈到实际测试的确切机制。在接下来的两章中，我们将看到如何部署上游项目，以及如何针对这些部署运行测试。

OPNFV自动化软件部署

如上一章所述，**OPNFV CI**的主要目标之一是持续的自动化测试。 测试过程分两步进行：

1. CI过程部署指定配置的一组上游项目
2. CI过程对部署的软件运行自动化测试

本章将介绍第一个步骤；第二个步骤将在第8章介绍。

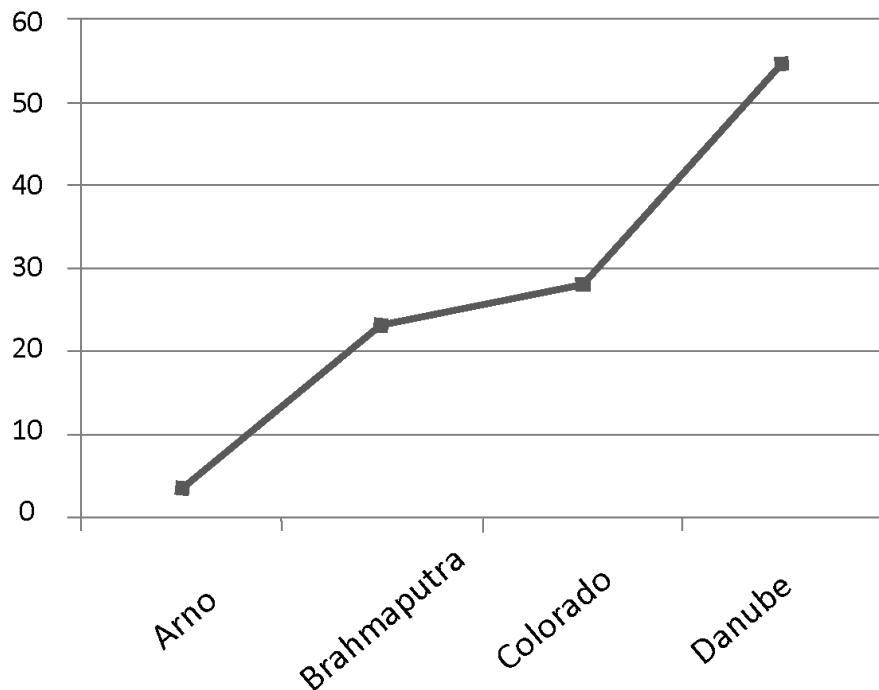
什么是**OPNFV**场景？

这里有一个明显的问题：如果**OPNFV**项目集成了几个层次上的多个软件项目，那么究竟被测试的是什么？举例来说，您无法同时测试ODL、ONOS和OpenContrail，也不能同时测试OVS和FD.io。此外，在配置中有许多变化，例如高可用性、DPDK、服务功能链（SFC）、BGPVPN等等，这里必须进行选择。为了解决这个难题，**OPNFV**提出了场景的概念。

场景就是部署一组组件及其配置

场景是一些组件和它们的配置的特定的组合。遍历所有可能的组合是不切实际的，因此，基于社区的关注点和终端用户的反馈（例如，来自终端用户咨询组 - EUAG），OPNFV社区部署和测试优先级最高的场景。对于**Danube**版本，共有55个场景。OPNFV的这种多功能性给用户提供了信心，使他们可以在多个预先测试的上游项目的组合中进行选择，以满足他们的需求。场景的数目随着时间一直在攀升，如下图所示。

各版本的场景数目



OPNFV为不同层次的NFV协议栈提供了选择，此外，OPNFV还提供了安装工具方面的选择，用来部署和配置这些软件。让我们更深入的了解一下：场景使用了下面的符号作为标识。因此，使用下面的关键字，例如：**os-odl_l2-fdio-ha,Fuel**，即表示使用Fuel安装的高可用模式的拥有ODL L2 SDN控制器和具备FD.io功能的OpenStack。

VIM - SDN 控制器 - 特性 - 模式 - 选项 , 安装工具

| | | | | |
|--|---|---|--------------------------|---|
| | | | | |
| 必选。 例如 : os (必选。 OpenStack) 等。 | 必选。 例如 : ovs 等。 nosdn、 ocl 、 odl、 onos 等。 | 必选。 例如 : nofeature 、 kvm、 必选。 ovs 等。 | 可选。 例如 : ha 、 noha | 可选。 例如 : 描述其 他选项 必选。 例如 : Fuel、 Apex, JOID 、 Compass 、 其他。 |

然而，并不是所有的55个Danube版本的[场景](#)都可以作为部署到生产环境的参考架构。其中有20个不具有高可用性（即noha），另外有3个是实验性的，因为他们使用了Kubernetes作为VIM或者使用Daisy作为安装工具。所以现在数字降到了32个，如下图所示。

| | Fuel | Apex | JOID | Compass |
|------------------------------|------|------|------|---------|
| os-nosdn-fdio-ha | | x | | |
| os-nosdn-kvm_ovs_dpdk_bar-ha | x | | | |
| os-nosdn-kvm_ovs_dpdk-ha | x | | | |
| os-nosdn-kvm-ha | x | x | | |

| | | | | |
|------------------------|---|---|---|---|
| os-nosdn-lxd-ha | | | x | |
| os-nosdn-nofeature-ha | x | x | x | x |
| os-nosdn-openo-ha | | | | x |
| os-nosdn-ovs-ha | | x | | |
| os-ocl-nofeature-ha | | | x | |
| os-odl_l2-bgpvpn-ha | x | | | |
| os-odl_l2-fdio-ha | | x | | |
| os-odl_l2-nofeature-ha | x | | x | x |
| os-odl_l2-sfc-ha | x | x | | |
| os-odl_l3-fdio_dvr-ha | | x | | |
| os-odl_l3-fdio-ha | | x | | |
| os-odl_l3-nofeature-ha | x | x | | x |
| os-odl_l3-ovs-ha | | x | | |
| os-odl-bgpvpn-ha | | x | | |
| os-onos-nofeature-ha | | x | x | x |
| os-onos-sfc-ha | | | x | |
| os-ovn-nofeature-ha | | x | | |

如您所见，如果我们忽略掉与特定安装工具相关的场景，则可以将用于创建生产环境的可选场景降到**21**种。

安装工具

基于提供多种选择的思想，**OPNFV**使用了四种安装工具。这些安装工具用来部署一个场景所需的所有软件，以及配置这些软件。尽管安装工具之间有很多差异，但是每个安装工具的基本机制都是类似的：安装工具首先在一个指定的“跳转服务器（**jumphost**）”上安装一个应用程序，该应用程序负责在指定的计算节点和控制节点上运行、下载、安装和配置所有必需的软件，以完成一个场景的完整部署。这也为在该场景上运行测试准备好了环境。安装工具的使用是通过**CI**流水线完全自动化的进行的。

坦率的说，如果安装工具的唯一工作是安装这些场景，则一个安装工具就够了，不需要使用这么多种安装工具。但是，这些安装工具还有除**CI**测试之外的生命周期。除了安装工作以外，它们还可用于**day-2**生命周期管理任务，例如部署后的配置更改，容量扩展，功能增强，更新和升级。一些安装工具还与监控软件集成。因此，**OPNFV**提供了多种安装工具以供选择。目前，**OPNFV**不会测试任何**day-2**的功能，所以，关于这些功能的讨论超出了本书的范围。让我们从初始部署（**day-1**）的角度，来深入的了解这四个安装工具。

Fuel

[Fuel](#)是由**Mirantis**公司推出的专用的**OpenStack**社区的安装工具项目。

Fuel界面示例

| CPU (Cores) | 47 (47) | RAM | 68.1 GB | HDD | 8.8 TB |
|------------------------------------|---------|------------------|---------|-----|--------|
| Total Nodes | 47 | Ready | | | 2 |
| Controller | 3 | Pending Addition | | | 45 |
| Compute | 32 | Offline | | | 45 |
| Cinder | 5 | | | | |
| Ceph OSD | 7 | | | | |
| Telemetry - MongoDB | 1 | | | | |
| Ironic | 1 | | | | |
| StackLight Infrastructure Alerting | 3 | | | | |
| Elasticsearch Kibana | 3 | | | | |

Fuel支持应用程序接口（API）、命令行接口（CLI）或者图形用户接口（GUI）。**Fuel**主节点启动后，使用PXE发现所有的计算、控制和存储节点。通过按功能组织的不同标签页，用户只需简单的点击的方式即可完成云的配置。**Fuel**有一个网络检查功能，可以在软件部署之前确保环境被正确的配置。部署完成后，还有一个部署后的健康检查。**Fuel**有一个可扩展的框架，通过使用插件，允许将新功能集成到**Fuel**中（例如，用于SDN控制器、MANO等的插件）。

Apex

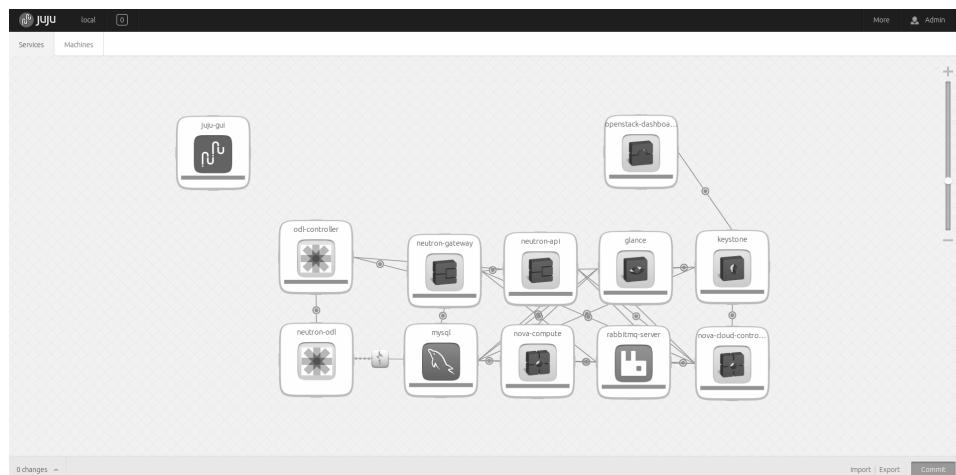
[Apex](#)是RDO基于OpenStack Triple-O项目的OpenStack安装工具。（RDO

是一个项目，它把最初为Ubuntu开发的OpenStack软件包移植到CentOS、Fedora和Red Hat Enterprise Linux。) Triple-O使用了一个有趣的理念，即运行在OpenStack之上的OpenStack。它首先安装一个轻量级的OpenStack，称之为“undercloud”。然后由undercloud使用Heat模板部署另一个实际工作负载使用的OpenStack，称之为“overcloud”。Apex提供了灵活性，但是代价是需要许多手动步骤。

JOID

[JOID](#)代表着Juju OPNFV基础架构部署程序（Juju OPNFV Infrastructure Deployer）。JOID使用了两个源于Canonical的开源项目：MAAS和Juju。两者都是通用技术，而不是专用技术。

JOID界面示例



MAAS（金属即服务）是部署裸机服务器的工具。MAAS集成了具有诸如部署OS、资产管理、IPAM（IP地址管理器）和PXE/IPMI等功能，并且提供了方便的图形用户接口（GUI）、命令行接口（CLI）和应用程序接口（API）。

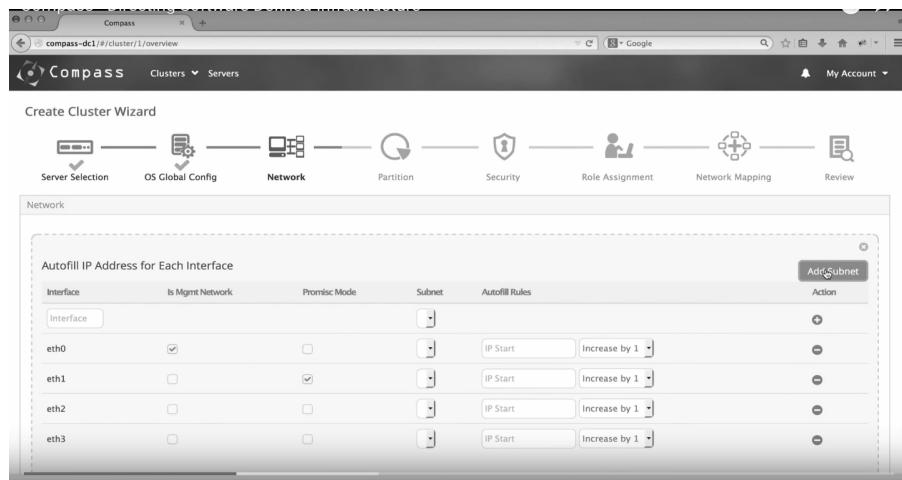
Juju是一种服务编排工具，用于在MAAS环境中部署OpenStack。如果您觉得

Juju听起来很熟悉，那是因为我们在第4章讨论过它，背景是作为多个**MANO**项目的**VNFM**。**Juju**使用称为**Charms**的描述符作为模板文件来定义服务。**Charms**支持诸如**Chef**、**Puppet**、**Ansible**等语言，并允许轻松的添加新功能，如**SDN**控制器或**MANO**。

Compass

Compass是来源于华为公司的另一个**OpenStack**社区的专用安装工具项目。

Compass界面示例



该工具可以通过图形用户接口（GUI）、应用程序接口（API）或者命令行接口（CLI）访问。**Compass**具有向导般的感觉，它系统地帮助用户跨越不同的部署阶段。部署是从发现开始，这一阶段可以使用各种机制，如**IPMI**、**SNMP**和英特尔®**RSA**。下一个步骤是部署操作系统 – **Centos**和**Ubuntu**它都支持。在多个配置步骤（网络、存储分区、安全性、各种角色的角色分配、以及网络映射）之后，使用**Chef**或**Ansible**部署各种**OpenStack**软件包。**Compass**还集成了日志和指标的基础设施监控看板。

其它软件部署项目

在上述四个安装工具之外，OPNFV还包括其他一些软件部署项目：

| 项目 | 描述 |
|-------------------------|---|
| Daisy (使用容器化的控制平面的安装工具) | OpenStack 部署的未来似乎是对 OpenStack 服务进行容器化并且对这些容器进行编排。这样做会显著提高 day-2 的体验。中兴通讯的开源 Daisy 项目使用了 OpenStack Kolla – 这是一个为 OpenStack 服务提供产品化的容器的项目，然后通过自动化部署、配置模板等工具进行构建。 Daisy4NFV 项目还在孵化阶段，并且承诺将简化大规模云的部署。 |
| 基于IPv6的OPNFV | 如项目的名称所示，该项目识别了在上游项目中实施完整的 IPv6 NFV 部署的差距，并且创建了 IPv6 场景。在 Danube 版本中，该项目可以使用 Fuel 、 JOID 和 Compass 。 |
| 部署模板翻译（Parser） | VNFM 层和 VNF 描述符提供了 NFVI 的需求（如 vCPU 、内存、存储、数据平面加速，等等），并以 TOSCA 等格式指定部署后的记录（如利用率、性能报告等）。但是，这些模板通常需要被翻译成其它格式，例如翻译成用于 VIM 层的 Heat 模板以便对其进行操作。该项目通过四个翻译项目来解决模板翻译问题： tosca2heat , yang2tosca , policy2tosca , tosca2kube . |
| ARMBand | ARMBand 的目标是确保OPNFV软件能够在 ARM 上运行。它复制了OPNFV的所有方面，如软件构建、 CI 、实验室配置，以及 ARM 架构的测试进程。目前，部分场景和测试项目的已通过 ARMBand 获得支持。 |

| | |
|--------------|-------------------------|
| 快速数据栈 (FDS) | FDS项目使用了APEX来部署FD.io场景。 |
|--------------|-------------------------|

目前，安装工具绑定到特定的POD。为了提高POD的利用率，以及为了诸如 **Multisite** (见第5章) 之类的项目，将安装工具与POD进行解耦变得非常重要，而且，使场景与POD的映射变得更加灵活和动态是OPNFV的未来发展方向。

在下一章，我们将看到如何在每一个场景上自动和持续的运行测试。

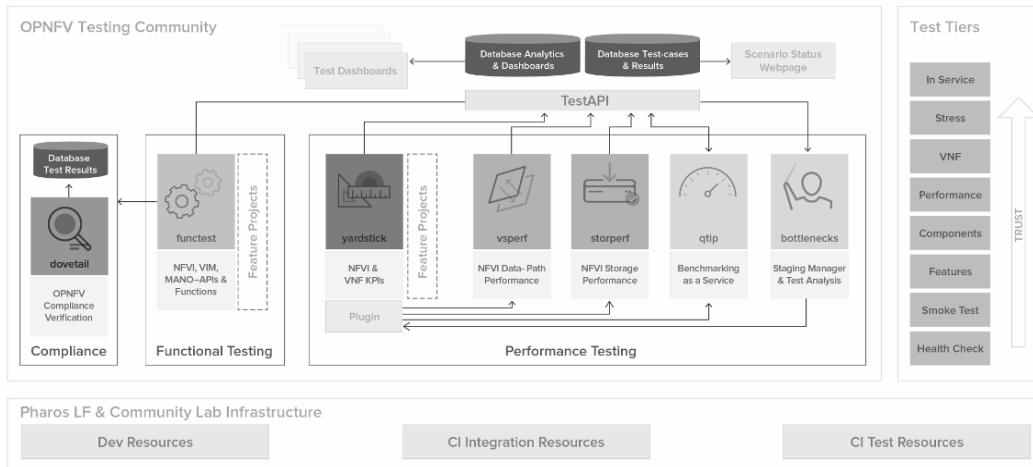
8

OPNFV持续测试

前面讨论的所有**OPNFV**的活动都是为了建立持续测试。重要的是重申对于整个软件栈端到端的测试是至关重要的，而且**OPNFV**是唯一在驱动这件事情的社区。通过这种测试，**OPNFV**能够发现单个组件测试无法发现的错误和安全问题。

OPNFV中的测试项目旨在使用多个测试用例来验证多个部署场景。下图显示了**OPNFV**的基本测试框架。本章的其余部分将对其进行详细介绍。

OPNFV测试组



一旦一个应用场景安装完成，我们就可以运行相对应的测试用例。这些测试在覆盖面，范围，层级，目的和测试频率方面都有所不同。这些测试用例和测试结果都被存储在数据库中，并通过多个可视化工具展示出来。正如我们常常说的，所有这些测试都是CI的一部分，并且以完全自动化的方式运行。

测试用例分类

OPNFV的测试用例可以从多个纬度进行分类：覆盖面，范围，层级和目的：

覆盖面

测试用例的覆盖面可以是端到端的，也可以是仅限于单个组件或子系统的。组件级测试涵盖NFV的各个项目，如OpenStack, NIC, ODL, VNF等。子系统级（也称为ETSI域）测试可以是整个VIM, NFVI的存储，由VNF（如vIMS）组成的完整服务等。端到端测试用例则将整个场景的所有组件作为一个内聚单元。

范围

每个测试用例的范围划分成三大类：功能，性能和合规性。目前为止，OPNFV中还没
110

有足够的稳定性和安全性测试，所以这些都归属于功能测试。随着对这些领域强调的越来越多，渐渐的就会归属到它们应在的类别中。涵盖OAM（操作维护和管理 – 包括生命周期管理），寿命（环境能够稳定多长时间），规模和破坏性测试的测试用例目前也不在OPNFV范围内。

层级

OPNFV的测试用例分多个测试层，其复杂性逐级增加而测试频率则可能逐级降低。

| 层级 | 例子 | 频率 | 信任度 |
|----------|---|--------|-----|
| 其他 | Bottleneck & QTIP项目（见下文） | 非周期性 | 高 |
| VNF | vIMS (Clearwater)测试 | 每周 | |
| 性能 | 完整性能测试 | 每周 | |
| 组件 | 完整的OpenStack测试，特定的性能测试 | 每天 | |
| 特性 | 特定项目的测试，如Doctor, Promise, security scan, HA, IPv6, SFC等 | 每天 | |
| 冒烟 | 基本冒烟，完整性，vPing测试 | 每天 | |
| 验证（健康检查） | 门限测试 | 代码提交测试 | 低 |



对于测试用例和场景，OPNFV引入了信任指标。简单来说，信任意味着随着每个后续层级的测试，对被测场景的信心增加了。具有较高信任度的测试用例其运行频率也较

低，这是因为它们通常是运行时间较长的测试，并且硬件资源不足也是社区面临的问题之一。通过这种方式，信任指标可帮助用户评估场景成熟度及帮助基础架构团队优化CI资源。

目的

一个测试用例可以有一个或者多个目的：

- **代码提交门限**：验证或健康检查测试是在一次代码提交时被触发，在gerrit上提供+1/-1。
- **版本发布门限**：这些测试用例是版本发布的时候必须通过的。如前所述，OPNFV每6个月发布一个大版本。每个大版本会伴有一些小版本，如Danube 1.0, 2.0 和 3.0，每个小版本都基于相同的稳定分支，只是会增加一些故障解决的合入。
- **只做信息参考**：一些性能测试不会阻碍版本发布。运行这些测试用例只为增加一些信息参考数据。
- **基准**：这些指标构成了基于性能指标的基准。
- **OPNFV认证测试**：这些测试针对一组验证标准进行测试，以便能够使用OPNFV商标。不要与法规合规测试相混淆。

另外一点：每个上游项目负责开发自己的测试，而不属于特定项目范围的全局测试则由测试项目开发。下面就让我们来深入了解这些测试项目。

测试项目

OPNFV测试项目根据测试范围来组织。最上层的是功能测试（**Functest**），其次是性能测试（**Yarstick**及一些子项目），然后是OPNFV合规测试（**Dovetail**）

Functest

Functest处理OPNFV中各种部署场景的验证和功能测试的工具和实际测试用例。目前的焦点是**VIM + NFVI**功能。其基本是使用各种上游项目的测试用例，如**OpenStack**

的**Tempest**（功能）和**Rally (Scale)**，ODL的**Robot**框架和ONOS的**Teston**框架。在**Functest**中重复这些上游测试有四个原因：**1)** 集成在一起测试以确保端到端的互操作性；**2)** 为许多OPNFV上游项目增加功能测试（这些测试也会对上游项目有贡献）；**3)** 增加了一些端到端的测试用例，这些测试用例可能是最重要的；**4)** 整合开源**VNFs**（如**Clearwater vIMS**和**MANO**项目），以在底层**VIM + NFVI**上产生真实的测试条件。

Yardstick

[**Yardstick**](#)是一个完全的性能测试项目，其基于**ETSI**参考测试用例集。在调查了大量的**NFV**工作负载后，**Yardstick**项目将整体需求分解为一组性能指标集，以量化**NFVI**的计算，网络和存储性能。该项目开发了自动化测试框架并为每组性能指标集开发了测试用例，并且通过并行运行，故障注入，测试多种拓扑等方式，我们可以用**Yardstick**组合出非常复杂和综合的用例。

Yardstick通过插件架构来实现可扩展性。通过**APIs**与其性能测试子项目进行通信。这些子项目有**VSPERF (vSwitch)**，**Cperf (SDN控制器)**，**Storperf (存储)**，**Qtip (基准测试即服务)**和**Bottlenecks (瓶颈检测)**。

为了评估已上线**VNF**的性能，**Yardstick**还提供了流量生成工具以及网络服务基准测试模块。

VSPERF

如果从吞吐量、抖动，每秒数据包和处理延迟来衡量，虚拟交换机（如**OVS**）是影响**VNF**性能的一个主要方面。因此，[**VSPERF**](#)项目测量**vSwitch**及相关的虚拟和物理网络端口的性能。它目前主要评估有部署**DPDK**及没有部署**DPDK**情况下**OVS**的性能，但该测试项目不依赖于特定的**vSwitch**实现和流量生成器。其性能测量路径如：

- 端口 → 虚拟交换机 → 端口
- 端口 → 虚拟交换机 → **VNF** → 虚拟交换机 → 端口
- 端口 → 虚拟交换机 → **VNF** → 虚拟交换机 → **VNF** → 虚拟交换机 → 端口

基于多个行业规范和用例，不同的测试用例测量点也不同，如转发速率，坏邻居（**Noisy Neighbour**）的影响，数据面和控制面耦合，CPU和内存利用率等。

VSPERF把vSwitch当成是物理交换机来测试，使用外部流量生成器，运行中谨慎地确保测试的准确性，一致性，稳定性和可重复性。**VSPERF**可以独立启动或通过**Yardstick**启动。

Cperf

[**Cperf**](#)测试SDN控制器的性能。它吸收上游项目性能团队的专业知识，如OpenDaylight性能小组。该项目运行众多性能测试，如：

- 只有一个控制器节点的网络可扩展性（例如，最多的交换机，端口，链路等）
- 集群可扩展性（例如，最多的控制器）
- 只有一个控制器集群的网络可扩展性（例如，最多交换机，端口，链路等）
- 流量性能（例如，每秒最大流量，数据包延迟等）
- API性能（例如北向，南向API延迟等）
- 数据存储的性能（例如，每秒最大读速率，每秒最大写速率等）

Storperf

[**Storperf**](#)测试外部块存储的性能。该项目的目标是提供基于**SNIA**（存储网络行业协会）性能测试规范的报告。该项目测量不同块大小和队列深度（未处理的I / O数）的延迟，吞吐量和IOPS（每秒IO次数）。

下面讲具体步骤，在跳板机上部署一个有storperf测试API的Docker容器。自动测试使用这些API来生成卷和虚拟机，连接它们，运行各种存储测试并收集结果。**Storperf**适用于HDD和SSD存储。使用Docker容器作为测试工具进行测试的方法在测试项目中是很常见的。最后，**Storperf**也可以独立启动或通过**Yardstick**启动。

QTIP

正如**MIPS**或**TPC-C**试图通过一个单一的数字来测量基础设施的性能，**QTIP**也尝试对**NVFI**的计算（一部分的存储和网络也在路标中）性能进行相同的测量。**QTIP**是一个**Yardstick**插件，它从五个不同类别的测试中收集指标：整数，浮点数，内存，深度包检测和加密速度。这些数字被不断压缩以产生单个**QTIP**指标。基准是**2500**，数字越大越好！在这个意义上，**QTIP**的目标之一就是让**Yardstick**的结果更直观易懂。

Bottlenecks

Bottlenecks项目试图在开发阶段而不是生产环境中发现系统性能限制。**Bottlenecks**与**Yardstick**进行了集成。与**Qtip**的创建新的基准的目标相反，**Bottlenecks**的目标是使用各种现有的基准和指标来衡量网络，存储，计算，中间件和应用程序的性能是否满足用户的要求。整个过程被用户设置的“实验配置文件”驱动。**Bottlenecks**基于这些配置文件驱动测试，并全自动的设置基础架构，创建工作负载，运行测试和收集结果。从这些测试中收集的数据往往相当大，因此该项目还实现了用于分析和可视化的工具。这些测试结果有助于识别不符合要求的测试指标，从而使用户能够对硬件，软件或者协议等进行选择决策，这个应该是评估被测系统是否满足SLA的要求。

Dovetail

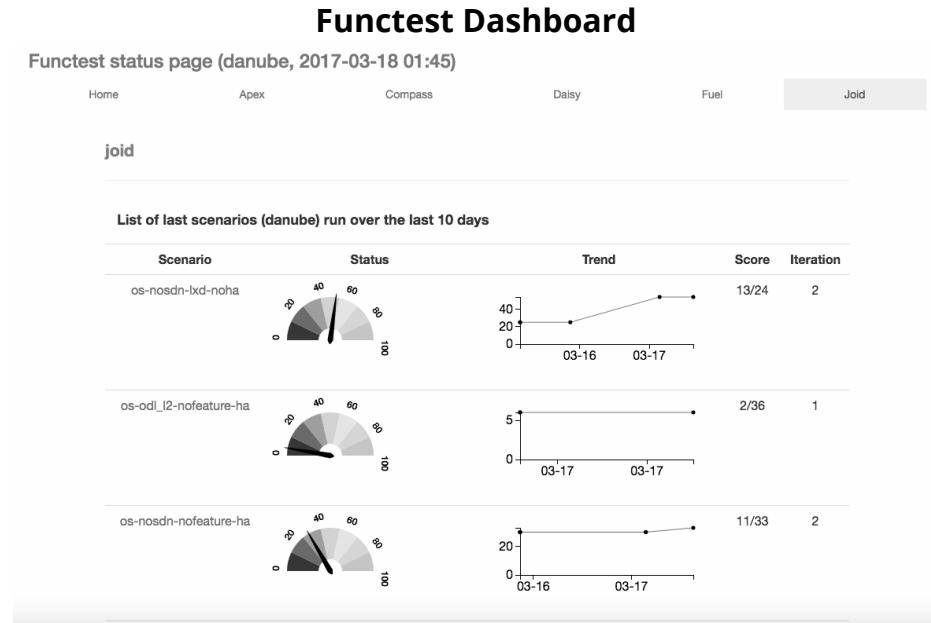
正如第3章描述的，使用开源软件的主要好处之一是避免供应商锁定。但说易行难。用户如何验证供应商的发布与原始项目一致，并且未被（无意中）更改？这对**OPNFV**来说更难，因为它不仅要处理一种单一发布，相反，有许多场景而且其中任何一个都可以被认为是发布的基础。Dovetail通过提供合规性测试来解决这个问题。通过运行**Functest**和**Yardstick**的这些测试，**Dovetail**构成了商业产品的合规性验证项目（**CVP**）的基础。**CVP**的目标是帮助构建基于**OPNFV**的基础设施和运行在该基础设施上的应用程序的市场，降低最终用户的采用风险，降低测试成本并增强互操作性。

看板

OPNFV实现了多种**Dashboards**分析和呈现测试结果。可以分为如下几类：场景状态报告，复杂结果现实（**Grafana**）或者API交互（**Swagger**）。在**Danube**版本中

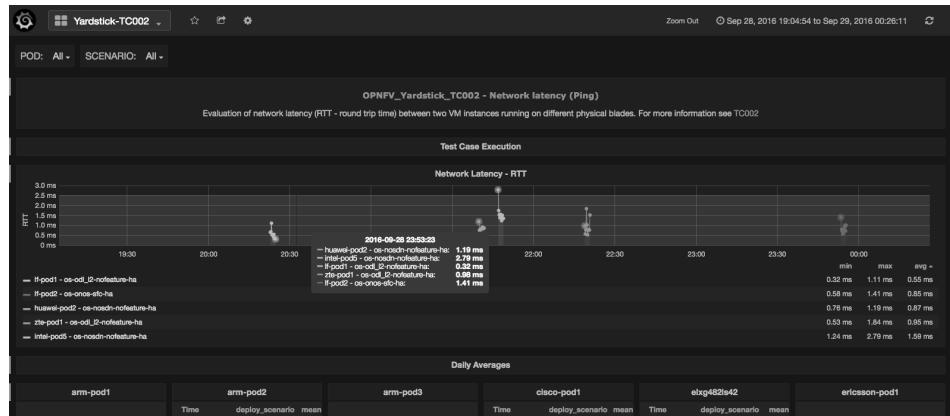
, 他们可以通过统一的[入口](#)访问。

Functest, Yardstick, Storperf和**VSPERF**测试项目都提供了场景报告[结果](#)。对于**Functest**, 还有关于**vIMS VNF**的报告。例子如下：



Yardstick结果可以通过[Grafana看板](#) (访问用户名和密码 : **opnfv / opnfv**) 进行可视化。任何测试用例都可以通过场景和POD来查看。由于数据来自时间序列数据库**InfluxDB**, 用户还可以指定时间段。

Yardstick Dashboard



Swagger[页面](#) (访问授权：`opnfv / api @ opnfv`) 使用用户能够直接与TestAPI进行交互⁵。例如，如果您尝试`GET / api / v1 / projects / {project_name} / cases`并输入`promise`，您将获得Promise的所有测试用例的列表。

Swagger Dashboard

The screenshot shows the Swagger API documentation interface. At the top, it displays the URL `http://testresults.opnfv.org/test/swagger/spec.json`. The main area is titled 'spec : Test Api Spec' and lists various API operations categorized by HTTP method and endpoint. Each entry includes a brief description of the operation.

| Method | Endpoint | Description |
|--------|---|---|
| GET | /versions | list all supported versions |
| GET | /api/v1/pods | list all pods |
| POST | /api/v1/pods | create a pod |
| GET | /api/v1/pods/{pod_name} | get a single pod by pod_name |
| GET | /api/v1/projects | list all projects |
| POST | /api/v1/projects | create a project |
| DELETE | /api/v1/projects/{project_name} | delete a project by project_name |
| GET | /api/v1/projects/{project_name} | get a single project by project_name |
| PUT | /api/v1/projects/{project_name} | update a single project by project_name |
| GET | /api/v1/projects/{project_name}/cases | list all testcases of a project by project_name |
| POST | /api/v1/projects/{project_name}/cases | create a testcase of a project by project_name |
| DELETE | /api/v1/projects/{project_name}/cases/{case_name} | delete a testcase by project_name and case_name |

⁵ 附注，测试项目不强制使用API框架，但大多数都实现了。

Plugfests

OPNFV conducts plugfests after each release. Vendors and community members get together and collaborate on interop testing and different test projects. The plugfest serves two purposes. First, it accelerates projects significantly by having a burst of intense activity with the ability to work on issues and solve problems with other contributors face-to-face rather than remotely. Second, vendors with proprietary products that cannot be included in the OPNFV CI pipeline can test their wares across different scenarios and test cases. These plugfests are generally held concurrently with hackfests, as mentioned in Chapter 5.

每个版本发布之后OPNFV都会举办一次**Plugfest**峰会。供应商和社区成员聚集在一起进行互操作测试和加强不同的测试项目之间的合作。**Plugfest**的举办两个目的。首先，它通过把贡献者召集在一起面对面（而非远程）处理事务、解决问题来加速项目的发展。第二，不能被OPNFV CI测试到的供应商专有产品，可以在**Plugfest**期间通过不同场景和用例进行测试。如第5章所述，**Plugfest**通常是与**hackfests**同时进行的。

除了已经提到的**RelEng**, **Pharos**和**Octopus**之外，还有许多非常重要的幕后基础设施建设工作如开发测试，故障排除，创建和管理数据库，为测试项目提供**Docker**容器化框架，构建工具来通过各种测试提升场景信任度，以及最终分析具体场景和**pod**对总体测试结果的贡献。

Finally, the OPNFV community recognizes the value of running functional and performance tests using real VNF workloads. In addition to project Clearwater mentioned above, a new project called Samplevnf has been created to do performance testing, benchmarking and characterization using five open source VNFs. These are “open source approximations” of carrier grade VNFs and not production grade; nevertheless, they are adequate for testing purposes. They are:

最后，OPNFV社区意识到使用真实VNF工作负载运行功能和性能测试的价值。除了上述提到的**Clearwater**项目之外，还创建了**Samplevnf**项目，使用五个开源VNF进行性能测试，定标和表征。这些**VNFs**是“开源近似”的电信级的，而不是产品级的；然而，就测试来说是足够的。这些**VNFs**是：

- CG-NAT (电信级网络地址转换器) VNF

- 防火墙 (**vFW**) VNF
- 边缘路由提供商 (**vPE**) VNF
- 接入控制列表 (**vACL**) VNF
- 下一代基础设施核心 (**NGIC**) 即 **VEPC-SAE-GW** VNF

本章完成了各种**OPNFV**测试项目的讨论。在下一章中，我们将讨论**OPNFV**如何实现VNF测试的。

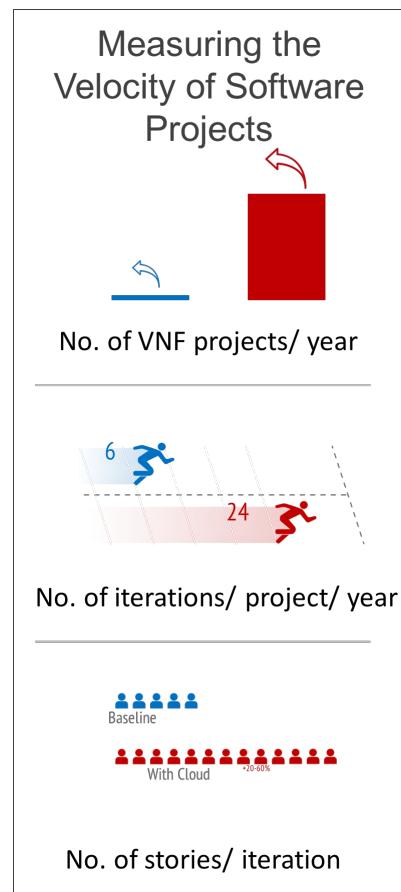
为OPNFV构建VNF

整个OPNFV栈最终用目的只有一个：通过一系列虚拟网络功能(**VNF**)，来构建网络服务。本章将着重介绍两个主题：如何构建**VNF**以及如何使构建的**VNF**上线运行。我们的分析将以已经在OPNFV上运行的vIMS VNF **Clearwater**作为示例。

构建VNF

我们在第2章中讨论了三种**VNF**架构：云托管，云优化和云原生。作为**VNF**创建者或买家，您首先需要考虑的是选择合适的架构。

云托管是在不进行任何优化的情况下仅将物理网络功能转换为**VNF**的最简单方式。云托管应用单一臃肿，而且通常是有状态的，这些**VNF**通常需要一个庞大团队去维护，而这个团队本身可能使用敏捷开发方法，也可能不使用；为了提高可用性，这些应用需要依赖于底层基础设施，即便如此通常也无法弹性伸缩。而在有些情况



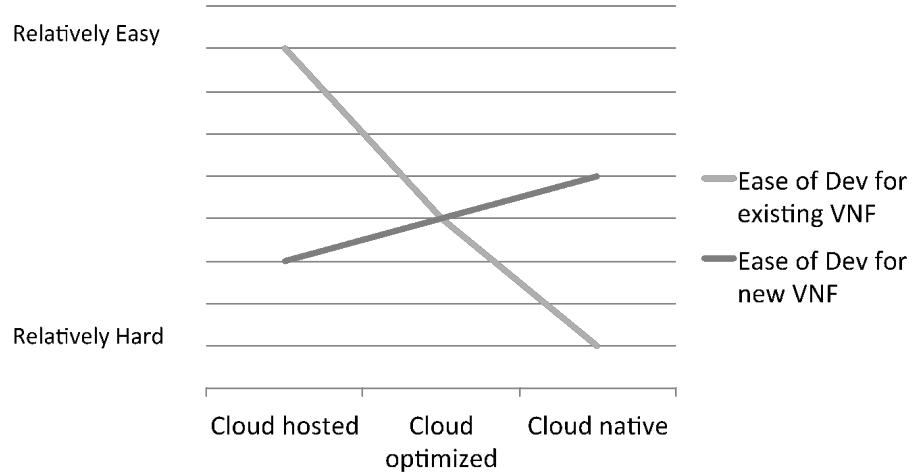
下，这样的**VNF**甚至可能需要手动进行配置。

于是有些开发人员开始重构基于云托管的**VNF**，使其更加适合运行在云上，称其为“云优化”。达成这一目的的非破坏性方法是将大型的应用进行分解，并将其转换为可通过**REST API**访问的服务；然后将有状态的**VNF**转换为专用服务，这样该应用的其余部分将成为无状态的。所有这些更改可以在软件开发方面获得更快的速度，同时有能力执行以云为中心的操作，如弹性扩容，弹性缩容和自愈。

将现有**VNF**转化为完全云原生的应用可能会过于繁重，如果可能，所有新的**VNF**应该专门以云原生的方式编写（我们已经在第2章中介绍了云原生应用程序模式）。通过使用云原生架构，开发人员和用户可以在创新方面获得更快的速度，并在**VNF**编排和生命周期管理中具有高度的灵活性。在**Mirantis**和**Intel**进行的一项企业终端用户研究中，云计算平台的年平均增长率从6增加到24（增长4倍），用户故事/迭代数增加了20–60%。尽管企业云原生应用程序与云原生**VNF**不同，但这些优势通常也适用于**NFV**。

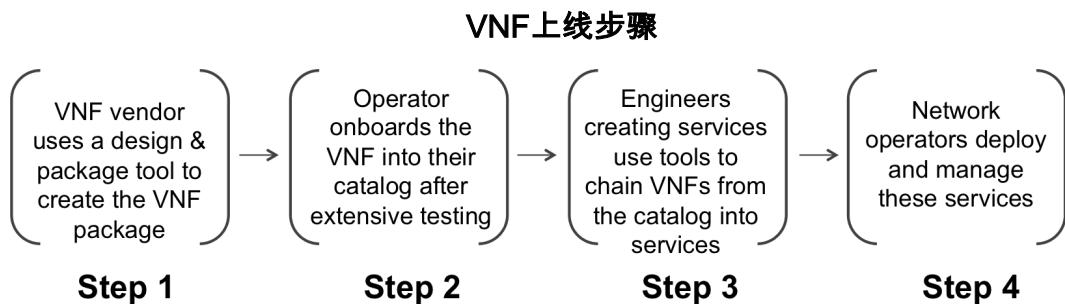
最终，总结下来，对现有**VNF**架构的选择没有对错之分，而对新的**VNF**应该被设计为云原生方式。下图显示了**VNF**应用架构权衡。

VNF架构选择的权衡



VNF上线运行

将VNF集成到OPNFV场景中时需要考虑的下一个主题是VNF如何上线运行。VNF本身对于上线无能为力，这要借助于MANO：MANO使用相关的元数据和描述信息来管理这些VNF；包含VNF描述（VNFD）信息的VNF软件包描述了VNF的要求、如何配置VNF以及如何管理其生命周期。有了这些信息，我们下面可以通过四个步骤来看下VNF上线过程。



这些步骤的详细描述超出了本书的范围，我们下面重点介绍VNF软件包。

要使VNF成功上线运行，需要在VNF软件包中指定以下类型的属性。需要说明的是，下面的清单并没有包含所有需要的信息，而只是一个示例。VNF软件包可能包括：

- 基本的信息如：
 - 价格
 - SLA (服务等级协议)
 - 授权模式
 - 供应商
 - 版本
- VNF软件包 (**tar**或者**CSAR**等格式)
- VNF配置信息
- NFVI 的要求，如：
 - vCPU数
 - 内存
 - 存储
 - 数据面加速信息
 - CPU架构
 - 亲和性/反亲和性要求
- VNF生命期管理信息：
 - 开始/停止
 - 弹性伸缩
 - 自愈
 - 更新
 - 升级
 - 终止

当前，如何给**VNF**打包和以及如何描述**VNF**还缺乏行业标准。每个**MANO**供应商或者说**MANO+NFV**供应商都有自己的私有格式。届时，为了使**VNF**上线，需要针对**VIM**添加额外的事项，并获得一个庞大到无法管理的开发和互操作矩阵。用户需要花费几个月的手动工作，才能将**VNF**部署到特定的**MANO + VIM**中，这是因为格式必须进行适配，然后才能进行测试。对于这个过程，用户和**VNF**提供商都认为不够好，不知道支持哪些模型，也不知道哪些组件要优先测试。

VNF管理器 (VNFM) 可能会使情况进一步复杂化。对于简单的**VNF**，也许通用**VNFM**可能就够用，但对于更复杂的**VNF**如**VoLTE (voice over LTE)**，可能需要

定制化（解读：私有）的**VNF**，由**VNF**供应商提供。毫无疑问，这种情况下，本已复杂的互操作矩阵将变得更加复杂。

除了手工工作和浪费时间外，由于缺乏标准还会暴露其他一系列问题。例如，**VNF**无法确定提供的资源是否符合其要求，另外在安全、隔离、弹性、自愈和其他生命周期管理阶段也可能和设计存在差距。

OPNFV认识到标准化**VNF**上线流程的重要性，**MANO**工作组以及**Models**项目（见第5章）正在尽力规范**OPNFV**的**VNF**上线流程。这些项目涉及多个问题，包括**VNF**软件包的开发、**VNF**软件包导入、**VNF**验证/测试（基本测试和在线测试）、**VNF**导入目录、服务蓝图创建和**VNFD**模型等。对于建模主要考虑三种语言：**UML**、**TOSCA-NFV simple profile**和**YANG**：

- **UML**: 统一建模语言（**UML**）由对象管理组（**OMG**）标准化，适用于各种用例。**ETSI**正在使用**UML**来规范其**VNFD**定义。**UML**可以被认为是以应用为中心的高级语言。
- **TOSCA-NFV simple profile**: **TOSCA**是以云为中心的建模语言。**TOSCA**蓝图描述了节点模板及其连通性的有向图；其次，工作流明确一系列动作如何产生，这些动作在考虑各种依赖关系时可能会变得复杂；最后，**TOSCA**还允许定义一定的策略，基于事件来触发工作流。**TOSCA-NFV simple profile**规范涵盖了使用**TOSCA**语言定义**NFV**相关的数据模型。
- **YANG**: **YANG**是IETF组织制定标准化的建模语言。与**TOSCA**或**UML**不同，**YANG**是一种以网络为中心的建模语言。**YANG**为网络元素的状态数据和配置进行建模，并描述了一个节点的树以及它们之间的关系。

上述三种建模方法**OPNFV**都适用，在某些情况下，可以混合使用多种建模语言来解决**VNF**上线问题。因为建模对**OPNFV**来说很重要，所以**OPNFV**需要与**ETSI**、**TMForum**、**OASIS**、**ON.Lab**等外部组织和项目进行大量合作。

OPNFV中的Clearwater vIMS

[Clearwater](#)是由[Metaswitch](#)开源的虚拟IP多媒体系统（vIMS）软件，是一个VNF项目。它是一个具有多个互连虚拟实例的复杂云原生应用。

对于OPNFV测试，[TOSCA](#)用作VNFD建模语言。[TOSCA](#)蓝图首先描述每个节点及其彼此间的连接。代码片段如下所示：

用于Homestead HSS镜像的VNF描述

```

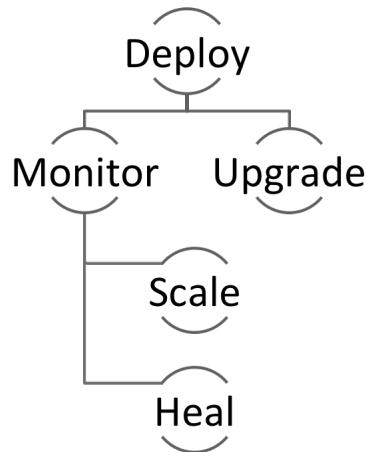
homestead_host:
    type: clearwater.nodes.MonitoredServer
    capabilities:
        scalable:
            properties:
                min_instances: 1
    relationships:
        - target: base_security_group
          type: cloudify.openstack.server_connected_to_security_group
        - target: homestead_security_group
          type: cloudify.openstack.server_connected_to_security_group

homestead:
    type: clearwater.nodes.homestead
    properties:
        private_domain: clearwater.local
        release: { get_input: release }
    relationships:
        - type: cloudify.relationships.contained_in
          target: homestead_host
        - type: app_connected_to_bind
          target: bind

```

接下来，使用[TOSCA](#)蓝图定义了一些工作流，这些工作流涵盖[Clearwater](#)的全生命周期管理。最后，蓝图描述了基于事件触发工作流的策略。

Clearwater TOSCA 工作流



下面的**TOSCA**代码片段展示了基于阈值的弹性扩容策略，当工作流被触发时，**Sprout SIP**路由器实例数将从初始1扩展到最大5。

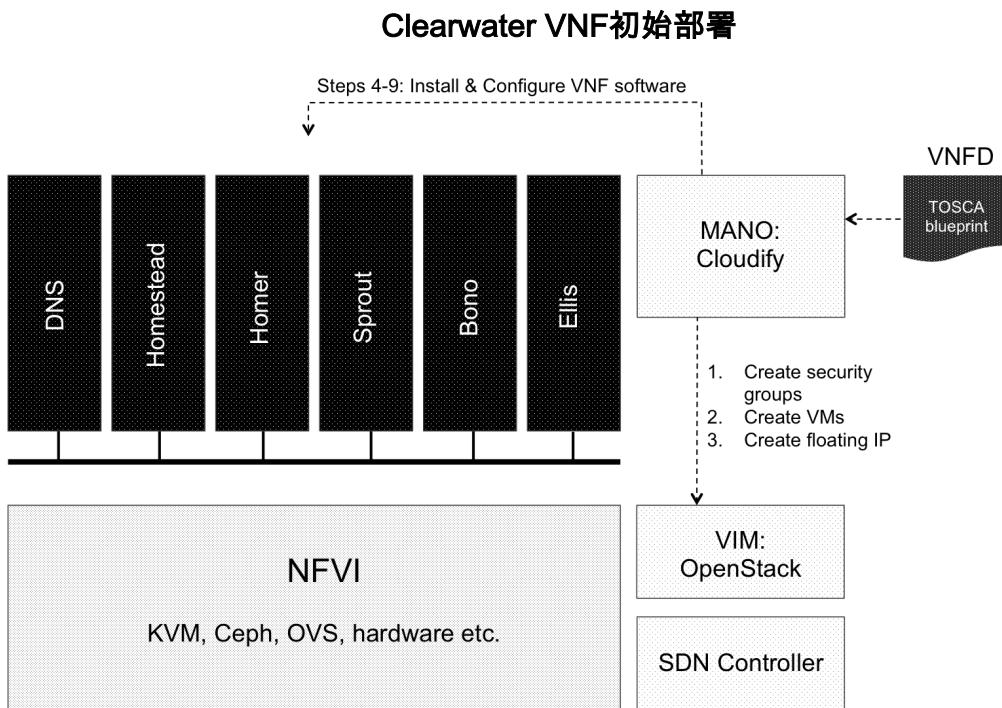
用于Sprout SIP的TOSCA 扩容政策和工作流

```

policies:
  up_scale_policy:
    type: cloudify.policies.types.threshold
    properties:
      service: cpu.total.user
      threshold: 25
      stability_time: 60
    triggers:
      scale_trigger:
        type: cloudify.policies.triggers.execute_workflow
        parameters:
          workflow: scale
          workflow_parameters:
            scalable_entity_name: sprout
  
```

```
delta: 1
scale_compute: true
max_instances: 5
```

蓝图完成后，编排器需要对TOSCA蓝图进行解释并根据蓝图执行动作，OPNFV使用[Cloudify](#)来测试Clearwater。[Cloudify](#)是[Gigaspaces](#)的MANO产品，有商业和开源的两种版本。[Cloudify](#)对上述蓝图中描述的每个工作流进行编排，具体来说，部署VNF的工作流如下所示：



完整的步骤可以通过**Functest**以自动化方式执行，要求如下：

步骤1：部署**VIM**, **SDN控制器**和**NFVI**

步骤2：部署**MANO**软件（可以是**Heat**、**Open-0**或**Cloudify**）。为了测试，可以使用完整的**MANO**栈（**NFVO + VNFM**）或仅使用**VNFM**。

步骤3：测试**VNF**，对于**Clearwater**项目，**Functest**运行了超过100种缺省信令，测试涵盖大多数**vIMS**测试用例（通话，注册，重定向，忙碌等）。

目前为止，我们虽然讨论了一个特定的**VNF**，但是这种方法同样可以应用于其他**VNF**（开源或专有）。由于**VNF**上线的复杂性，使用**OPNFV**作为构建**VNF**的标准方式将为行业带来了巨大的价值。没有任何供应商或用户有资源或时间能对完整的互操作矩阵执行测试，但作为一个社区，这种可能性是存在的。

从这一点上，为了能说明开源的力量，值得多啰嗦几句。最初**Clearwater**项目的测试工作由**Orange**的实习生完成。这项工作后来变得相当受欢迎，被众多厂商采用，并影响了**OPNFV MANO**工作组，甚至说服了一些运营商使用**OPNFV**作为构建**VNF**的工具。

到此为止，我们讨论了如何构建基于不同应用架构的**VNF**并上线运行，并结合一个具体的例子演示了如何使用**OPNFV**上线**Clearwater vIMS VNF**进行测试。在下一章中，我们将讨论如何从**OPNFV**项目中受益以及如何参与到**OPNFV**项目中。

10

利用OPNFV驱动业务

OPNV项目给NFV最终用户、技术供应商以及个人提供真正的商业收益。

OPNFV和最终用户

NFV为最终用户提供了有形和无形的收益。有形的收益包括那些直接影响商业指标的因素，而无形的包括那些加速整个NFV转型之旅，但难以衡量的好处。

有形收益

通过参与OPNFV项目，电信运营商和企业可以获取到可衡量的收益：

- **加快新服务上市**：网络部署经常遭受如技术评估，原型项目，设备商选择，现场实验和生产部署的无限期延迟的损害。通过加入OPNFV，最终用户可以和社区其他用户合作，从而完成以上大多数步骤。由于供应商的产品从社区的测试和合作中收益，余下的工作仅仅是简单选择一个基于OPNFV技术栈的供应商提供产品就行，对最终用户而言，这将直接导致新产品的上市时间缩短。

- **VNF上线平台**：由于底层的**NFVI**, **SDN**控制器和**VIM**平台在供应商间不一致，所以没有简单的方法来比较**NFV**的基准。通过使用**OPNFV**场景，最终用户可以在一致的环境中进行标准化，对不同的**VNF**进行测试，表征和基准测试。而且，用户可以利用**Yardstick**和其它**OPNFV**性能测试项目来内部验证平台性能。这导致上市时间的缩短和改进的技术栈，从而为最终用户带来更高的项目投资回报率(**ROI**)。
- **一流的组件**：正如我们在第3章讨论的，开源软件主要的优点之一就是能够挑选和选择符合特定用例的一流组件。**OPNFV**专门在**NFV**堆栈的不同层次上提供多种选项，从而能够让用户进行选择。通过增加堆栈的价值并为其它活动释放宝贵的开发者资源，从而提供更大的项目投资回报率。
- **没有供应商锁定**：开源软件另一个主要优点就是可以消除供应商锁定。只要市场足够大，一定有多个供应商；这可以减低成本，用户也可以切换供应商。我们不可能最大限度减少切换供应商的工作量，但这要比在切换专利技术时要小得多。这样可以导致总体拥有成本 (**TCO**)显著的降低。
- **解决问题的产品**：**OPNFV**社区中的用户不需要通过一系列守护者（如客户经理，系统工程师，产品经理，系统架构师，现场CTO等）与专有供应商的工程团队进行沟通，而是可以和供应商的工程师一起并肩工作，确保他们关心的功能包括在产品中。用户甚至也可以自己贡献功能。这样可以为最终用户提高项目投资回报率。

无形收益

将**OPNFV**仅仅视为技术是短视的。通过参与**OPNFV**社区，电信运营商可以深入了解**NFV**转型的各个方面，加快他们转型的旅程。

明确来说，让我们回顾第2章讨论的**NFV**变革的四大支柱，并评估**OPNFV**如何帮助每一个。

对组织架构影响

OPNFV中超过45个项目团队都是由**PTL**（项目技术主管）、提交者和贡献者组成的小

组。这些团队优先考虑他们的积压任务，开发需求，根据需要贡献上游代码，开发他们自己的测试，确保给定版本项目准备就绪。技术指导委员会（TSC），基础设施和测试项目执行OPNFV各项目的协调和公共活动。

用户可以通过参与OPNFV来了解如何组织他们的团队成功进行NFV转型，并对OPNFV之后的组织结构进行建模。我们再来看看第2章的组织架构准备问题调查问卷，看看OPNFV如何帮助：

| 问题 | 准备就绪：1–5 (1=未准备好， 5=完全准备就绪) | 为什么？ |
|---|---------------------------------------|--|
| 有没有跨职能团队组建新技术团队的例子？ | 4 | 大多数团队拥有来自用户和技术提供商的广泛代表。 |
| 您可以想象拥有开发，测试，发布以及生产的每个NFV服务或组件（如NFVI，VIM，MANO）的完全自主的团队？ | 5 | 团队拥有各个方面。（生产阶段不适用于OPNFV。） |
| 您可以想象一个分散的决策方法，每个团队在理性之中可以做出自己的技术决策？ | 5 | 这一原则完全纳入OPNFV。每个团队在理性之内铺平了道路。 |
| 你能想象NFV服务团队和平台团队之间的沟通是完全自动化？ | 4 | OSS / BSS和MANO，MANO和NFVI / VIM之间的接口完全通过API指定。不需要人为的交互。 |
| 是否有机制来讨论影响多个团 | 5 | 通过技术指导委员会，基础设 |

| | | |
|---------------|--|--------------------|
| 队的需求，并协调这些要求？ | | 施和测试团队，有很多机制来协调项目。 |
|---------------|--|--------------------|

对过程影响

虽然OPNFV不管理生产环境NFV云，但开发，测试和发布流程比传统电信运营商更接近Web 2.0公司。OPNFV项目之间的决策和协调由轻量级流程控制。例如，创建新项目的建议与创建Wiki页面一样简单，并在初步审核两周后将项目呈现给技术社区。如果没有异议，建议提交至TSC等待批准。

在更具体的层次上，OPNFV采用的CI/CD流程给传统电信公司提供了一个很好的机会来学习和了解DevOps方法，此方法在某种程度上被认为不适用于电信社区。

用户可以通过参与OPNFV来学习如何修改NFV成功转型的过程。我们再来看一下第2章的过程准备情况调查问卷，看看OPNFV如何帮助：

| 问题 | 准备就绪：1-5 (1=未准备好， 5=完全准备就绪) | 为什么？ |
|--|---------------------------------------|-----------------------------------|
| 你能想象一个实习生在工作的第一天提交一个变更到生产网络吗？(Etsy要求他们的实习生这样做) | 4 | OPNFV还没有CD流水线，但实习生可以在第一天提交补丁用于CI。 |
| 你能想象一个工程师在1%的用户基础上的生产网络实验新功能，而不需要任何许可？(Twitter允许这样。) | N/A | 不适用。 |
| 您是否使用敏捷方法？方法 | 5 | OPNFV方法是完全敏捷的，由 |

| | | |
|---|-----|-----------------|
| 是真正的敏捷还是敏捷瀑布（即瀑布和敏捷的混合体）？ | | JIRA的史诗和用户故事驱动。 |
| 您的财务团队可以处理未分配到特定成本中心的硬件采购，但是对于总体 NFV 云计算采购呢？ | N/A | 不适用。 |
| 您的合规性，资产清单和安全团队能否处理动态虚拟基础架构？ | N/A | 不适用。 |

对技术影响

OPNFV项目的主要目标是为用户提供他们可以使用并为其目的量身打造的开源技术。

除了上述“有形福利”部分的好处之外，通过参与，用户还可以通过成功的**NFV**转型所需的技术获得宝贵的经验。

我们再来看一下第2章的技术准备问题调查问卷，看看**OPNFV**如何帮助：

| 问题 | 准备就绪：1-5 (1=未准备好， 5=完全准备就绪) | 为什么？ |
|--------------------------|-----------------------------------|---|
| 您是否准备好云原生应用程序和微服务？ | 5 | OPNFV 使用非常适合云原生应用程序的 OpenStack 作为VIM。 |
| 你的组织是否接受 <u>混沌工程</u> 原理？ | 4 | 虽然 OPNFV 没有正式的混沌工程项目，但技术社区文化面临着这些相同的原则。 |

| | | |
|--|---|------------------------------------|
| 您的组织是否准备好设置 DevOps 管道和自动测试？ | 5 | DevOps CI / CD方法论是OPNFV的基石。 |
| 您的运营团队能否适应模型驱动架构？ | 5 | OPNFV集成了都是模型驱动的开源MANO项目。 |
| 你可以想象一个监控框架，您无法访问单个 VNF 或实例，而是在聚合云上完成管理的？ | 5 | 参见第5章，多个项目协助管理云的这一方面。 |

对技能集获取的影响

到目前为止，用户获得NFV转型相关技能的最便宜和最可扩展的方法是参与**OPNFV**。仅仅通过各种社区沟通方式，使用工具，与社区成员互动，您的团队就将获得与**VIM**，**SDN**控制器，**MANO**，**NFVI**，服务保证，性能基准等相关的特定技能，以及通用的云原生技能，如模型驱动架构，微服务和**DevOps**。在大学或企业培训机构中，根本没有足够的正式课程来填补使用传统技术的技能获取差距。

我们再来看一下第2章的技能集准备问题调查问卷，看看**OPNFV**如何帮助：

| 问题 | 准备就绪：1-5 (1=未准备好， 5=完全准备就绪) | 为什么？ |
|-------------------------------|-----------------------------------|--|
| 员工是否允许任何时间和/或访问实验室资源来自己学习新事物？ | N/A | OPNFV Pharos Labs 允许成员和其他人用 CI / CD 基础架构进行集成，并运行 VNF 的测试部署。 |
| 你能想象一个公司范围的“黑客周”（一周的黑客松）吗？ | 5 | OPNFV 定期进行为期一周的黑客松和兼容性测试活动。 |

| | | |
|--|-----|--------------------------------------|
| 是否有正式的培训计划？您公司的技术人员参加相关的贸易展览或峰会有多容易？ | N/A | 不适用。 |
| 是否有内部研讨会或知识传递计划？团队会彼此分享学习吗？ | 4 | OPNFV举办峰会和活动来传递知识，有很多学习资源（主要是社区驱动的）。 |
| 是否有项目的回顾？你能想象一环境，上层管理层在这些项目回顾过程中不会指责吗？ | 4 | 每个项目都进行自己的评估。例如，TSC不会为错过期限进行任何责任。 |

提案请求OPNFV问题示例

将OPNFV相关问题纳入您的提案请求（RFP）还为时尚早。这里有一些示例问题；我们相信您的采购团队可以提出更多的建议。

| 问题 |
|---|
| 您的产品是否使用OPNFV项目的任何上游组件？如果是，有哪些？ |
| 您的产品是否包含任何OPNFV上游贡献项目？如果是，哪些？如果没有，为什么呢？ |
| 您的产品是否包含任何OPNFV测试和集成项目？如果是，哪些？如果没有，为什么呢？ |
| 您的产品是否利用任何特定的OPNFV场景？ |
| 您是否使用OPNFV中使用的其中一个安装部署工具？如果是，哪一个？如果没有，请描述您用的安装工具。 |

| |
|---|
| 您的产品是否针对 OPNFV 组件进行测试？请说明。 |
| 你是否测试所有的 OPNFV 测试套件？如果是，请提供结果。如果没有，为什么呢？ |
| 在功能或服务方面，您在 OPNFV 集成堆栈上创造了什么价值？ |
| 您的内部流程是否使用持续集成和测试？ |
| 您的堆栈是否通过了 OPNFV 合规性验证计划（尚未提供，但即将推出）？ |

OPNFV和技术提供商

OPNFV除了对用户有用外，还对技术提供商有益，例如：

- 硅晶体厂商 (**CPU, NIC**等.)
- 服务器厂商
- 交换器&裸机交换器厂商
- 存储器厂商
- **VNF, MANO, SDN** 控制器, **VIM** 供应商
- 操作系统和虚拟化供应商

基于**OPNFV**集成堆栈来创建产品的供应商可以通过借助**OPNFV**社区的努力来缩短其开发进度并减少其工作量。或者，供应商可以选择将其专有产品与**OPNFV**集成。在这些情况下，**OPNFV**在其常规测试中不包括专有产品，但是无法阻止技术提供商，通过建立私有类似**Pharos**样的实验室和**CI**管道运行特定场景下的测试用例。这些测试和场景可以被增强以适应专有产品。测试产品的另一个论坛是**OPNFV plugfest**，在那儿所有的技术和商业产品都被欢迎。

在上述两种情况下，供应商项目的投资回报率显著地提高。

OPNFV和个人

OPNFV的每一项活动也对个人开放。参与**OPNFV**是一种以完全免费的方式构建你下一个工作所需技能的机制。**NFV**的技能难以寻觅，只有随着市场增长和诸如**5G**等新

使用场景的出现，需求才会增长。即使您不是开发人员，您也可以帮助编写文档，优先考虑痛点，优化用例等等。

参与

用户，技术提供商和个人有多种方式参与OPNFV。你不必是OPNFV的会员，但很多人决定加入并支持开源的NFV。[参与](#)的具体步骤仅需要一个免费的Linux Foundation帐户。一切就这么简单！

以下是一些具体的参与方式，以及相关的好处：

| 活动 | 适合 | | | 收益 |
|------------------|------|-------|----|------------------|
| | 最终用户 | 技术提供商 | 个人 | |
| 加入已有的项目 | x | x | x | 学习并且使堆栈更适合您的使用场景 |
| 创建新项目 | x | x | x | 使堆栈更适合您的使用场景 |
| 参加活动/黑客松 | x | x | x | 学习，合作，交流 |
| 出席活动 | x | x | x | 与他人分享学习，招募贡献者 |
| 使用CI流水线持续消费OPNFV | x | x | x | 学习并提供反馈以改善堆栈 |
| 贡献Pharos实验室 | x | x | | 提高OPNFV测试能力 |

| | | | | |
|------------------------------|---|---|--|---------------------------------|
| 加入最终用户咨询组 | x | | | 提供反馈和要求，以确保 NFV 痛点得到解决 |
| 介绍客户成功案例 | x | | | 帮助其他用户参与 |
| 参加 Plugfest 兼容测试活动 | | x | | 确保您的开放或专有产品与 OPNFV 的互操作性 |
| 建立一个“私有的”类 Pharos 实验室 | x | x | | 确保您的专有代码或产品与 OPNFV 的互操作性 |
| 发布基准和测试结果 | | x | | 在 OPNFV 上下文中推广您的技术 |

总之，**OPNFV**为最终用户，技术提供商和个人带来了实实在在的好处。

下一代网络正在建设中，它正在用开源工具来构建。**OPNFV**正在加速测试，集成和合作，以便利用开源的**NFV**端到端。

下一代网络的基础正在形成；并且**OPNFV**正在加速测试，集成和合作，以便利用开源的**NFV**端到端。参与很容易，为什么等待？马上参与吧！

11

其它资源

通用的

| | |
|------------|--|
| OPNFV网站 | opnfv.org |
| OPNFV 维基网页 | wiki.opnfv.org |
| OPNFV 问答 | ask.opnfv.org |
| OPNFV软件和文档 | opnfv.org/software/downloads |

第1章节资源

| | |
|-----------------------|--|
| Telstra PEN | telstraglobal.com/products/connectivity/pen |
| 虚拟化革命：网络功能虚拟化释 放视频 | youtu.be/-c5P2CvBTRg |

| | |
|--|--|
| 英特尔vCPE总成本计算器 | builders.intel.com/docs/vE-CPE-for-communication-service-providers.pdf |
| ETSI NFV ISG 欧洲电信标准协会网络功能虚拟化因特网业务组 | etsi.org/technologies-clusters/technologies/nfv |
| ETSI NFV architectural framework 欧洲电信标准协会网络功能虚拟化结构框架 | etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf |
| ETSI NFV terminology 欧洲电信标准协会网络功能虚拟化术语 | etsi.org/deliver/etsi_gs/NFV/001_099/003/01.02.01_60/gs_NFV003v010201p.pdf |
| ETSI NFV POC framework 欧洲电信标准协会网络功能虚拟化POC框架 | etsi.org/deliver/etsi_gs/NFV-PER/001_099/002/01.01.02_60/gs_NFV-PER002v010102p.pdf |
| ETSI NFV Security Problem Statement 欧洲电信标准协会网络功能虚拟化安全问题申明 | etsi.org/deliver/etsi_gs/NFV-SEC/001_099/001/01.01.01_60/gs_NFV-SEC001v010101p.pdf |
| CSP和OpenStack的深入研究 | openstack.org/assets/pdf-downloads/OpenStack-survey-results-public-presentation.pdf |
| 欧洲电信标准协会网络服务头（服务链） | datatracker.ietf.org/doc/draft-ietf-sfc-nsh/ |

第2章节资源

| | |
|------------------|--|
| 12因素应用 | 12factor.net/config |
| 关于waterscrumfall | waterscrumfall.org/ |
| 混沌工程 | principlesofchaos.org/ |
| SDx中心运营商采用试错法 | sdxcentral.com/articles/news/sdn-world-congress-carriers-strive-agile-nfv-becomes-real/2016/10/ |

第3章节资源

| | |
|--------------------------|--|
| OPNFV 仪表盘 | opnfv.biterg.io |
| OPNFV最终用户咨询组 | opnfv.org/end-users/end-user-advisory-group |
| Heavy Reading发布的开放新时代的通信 | www.fujitsu.com/us/Images/Communications-in-the-New-Era-of-Open.pdf |
| CORD | opencord.org |
| VMware vSphere ESXi | vmware.com/products/vsphere.html |

第4章节资源

| | |
|--------|--|
| ONAP项目 | onap.org |
|--------|--|

| | |
|----------------------------|--|
| OPEN-O项目 | open-o.org |
| OPNFV Opera 项目 | wiki.opnfv.org/display/PROJ/OPNFV-OPEN-O |
| Juju | jujucharms.com |
| OpenBaton | openbaton.github.io |
| OPNFV Orchestra 项目 | wiki.opnfv.org/display/PROJ/Orchestra |
| OpenStack Tacker 项目 | docs.openstack.org/developer/tacker |
| 开源 MANO | osm.etsi.org |
| OpenStack开源项目 | openstack.org |
| OpenStack 电信和网络功能虚拟化 | openstack.org/telecoms-and-nfv |
| OpenStack项目导航 | openstack.org/software/project-navigator |
| Kubernetes项目 | kubernetes.io |
| OpenStack Neutron网络组件 | wiki.openstack.org/wiki/Neutron |
| OpenDaylight项目 | opendaylight.org |
| ONOS项目 | onosproject.org |
| OpenContrail项目 | opencontrail.org |

| | |
|----------------|--|
| OVN | benpfaff.org/~blp/dist-docs/ovn-architecture.7.html |
| KVM虚拟机 | linux-kvm.org |
| LXD容器管理 | ubuntu.com/cloud/lxd |
| libvirt虚拟API项目 | libvirt.org |
| Ceph存储项目 | ceph.com |
| OVS虚拟交换项目 | openvswitch.org |
| FD.io通用数据面项目 | fd.io |
| FD.io性能研究 | lightreading.com/nfv/nfv-tests-and-trials/validating-ciscos-nfv-infrastructure-pt-1/d/d-id/718684?page_number=8 |
| DPDK项目 | dpdk.org |
| DPDK性能研究 | software.intel.com/en-us/articles/using-open-vswitch-with-dpdk-for-inter-vm-nfv-applications |
| ODP项目 | opendataplane.org |
| 开放计算项目 | opencompute.org |

第5章节资源

| | |
|---------------------|--|
| 诺基亚: OpenStack和实时应用 | openstack.org/videos/barcelona-2016/nokia-openstack-with-real-time-applications |
|---------------------|--|

| | |
|--|--|
| OVS DPDK显示10倍性能增益的讨论 | wiki.opnfv.org/display/ovsnfv/Project+Proposal |
| 维基百科雾计算页面 | en.wikipedia.org/wiki/Fog_computing |
| OPNFV CII最佳实践徽章 | bestpractices.coreinfrastructure.org/projects/164 |
| OPNFV Doctor项目 | wiki.opnfv.org/display/doctor/Doctor+Home |
| 基于OPNFV Doctor项目框架，以OpenStack Congress和Virtage项目为基础的故障管理视频 | openstack.org/videos/barcelona-2016/fault-management-with-openstack-congress-and-vitrage-based-on-opnfv-doctor-framework |
| OpenStack主题演讲： OpenStack和OPNFV – 保持您的手机通话连接 | youtube.com/watch?v=Dvh8q5m9Ahk&t=9s |

第6章节资源

| | |
|-------------------|--|
| OPNFV JIRA工具 | jira.opnfv.org |
| OPNFV项目会议 | wiki.opnfv.org/display/meetings |
| OPNFV git库 | git.opnfv.org |
| OPNFV gerrit评审库 | gerrit.opnfv.org |
| OPNFV gitbhub 镜像库 | github.com/opnfv |

| | |
|---------------------------|--|
| OPNFV Jenkins CI任务运行状态显示页 | build.opnfv.org |
| OPNFV制品库 | artifacts.opnfv.org |
| OPNFV Docker hub 库 | hub.docker.com/search/?q=opnfv |
| OPNFV Pharos项目 | opnfv.org/community/projects/pharos |
| OPNFV Pharos 仪表盘 | labs.opnfv.org |

第7章节资源

| | |
|---------------------|--|
| OPNFV Danube版本场景 | wiki.opnfv.org/display/SWREL/Danube+Scenario+Status |
| OPNFV Fuel部署工具项目 | wiki.opnfv.org/display/fuel/Fuel+Opnfv |
| OPNFV Apex部署工具项目 | wiki.opnfv.org/display/apex/Apex |
| OPNFV JOID部署工具项目 | wiki.opnfv.org/display/joid/JOID+Home |
| OPNFV Compass部署工具项目 | wiki.opnfv.org/display/compass4nfv/Compass4nfv |

第8章节资源

| | |
|-----------|--|
| OPNFV测试概述 | wiki.opnfv.org/display/testing/TestPerf |
|-----------|--|

| | |
|-----------------------------------|--|
| Functest 功能测试项目 | wiki.opnfv.org/display/functest/Opnfv+Functional+Testing |
| Yardstick 性能测试项目 | wiki.opnfv.org/display/yardstick/Yardstick |
| VSPERF 网络性能测试项目 | wiki.opnfv.org/display/vsperf/Vsperf+Home |
| Cperf 项目 | wiki.opnfv.org/display/meetings/CPerf |
| VSPERF 存储测试项目 | wiki.opnfv.org/display/storperf/Storperf |
| QTIP 基准测试项目 | wiki.opnfv.org/display/qtip |
| Bottlenecks 系统瓶颈测试项目 | wiki.opnfv.org/display/bottlenecks/Bottlenecks |
| Dovetail 合规验证项目 | wiki.opnfv.org/display/dovetail/Dovetail+Home |
| Danube 版本发布测试仪表盘 | testresults.opnfv.org/reporting/danube.html |
| Swagger API 仪表盘 | testresults.opnfv.org/test/swagger/spec.html |
| Grafana 项目仪表盘 | testresults.opnfv.org/grafana |
| Functest 项目仪表盘 | testresults.opnfv.org/reporting/functest-danube.html |
| OPNFV 12月16号 Plugfest 测试报告 | opnfv.org/wp-content/uploads/2017/02/OPNFV_Plugfest_II_Report_FINAL.pdf |

第9章节资源

| | |
|--------------------|--|
| 对象管理组 | omg.org |
| 欧洲典型标准协会VNFD规范 | etsi.org/deliver/etsi_gs/NFV-IFA/001_099/011/02.01.01_60/gs_NFV-IFA011v020101p.pdf |
| TOSCA NFV简介 | docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html |
| YANG数据建模语言 | tools.ietf.org/html/rfc6020 |
| Clearwater项目 | projectclearwater.org |
| Clearwater TOSCA蓝图 | github.com/Orange-OpenSource/opnfv-cloudify-clearwater |
| Cloudify云管理平台 | getcloudify.org |

第10章节资源

| | |
|-------------------|--|
| 参与OPNFV | opnfv.org/community/get-involved |
| OPNFV 教程视频 (如何参与) | youtube.com/playlist?list=PLiM029E-zvrs15SWfjfcJ19FWCsTT49f0 |