**Open Source MANO 技术调研报告**

# 引言

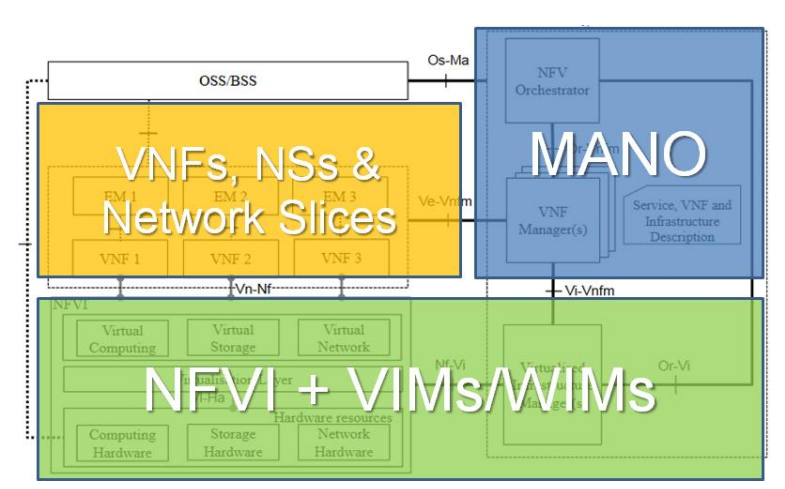
NFV即网络功能虚拟化（Network Function Virtualization），通过使用x86等通用性硬件以及虚拟化技术，来承载很多功能的软件处理。从而降低网络昂贵的设备成本。可以通过软硬件解耦及功能抽象，使网络设备功能不再依赖于专用硬件，资源可以充分灵活共享，实现新业务的快速开发和部署，并基于实际业务需求进行自动部署、弹性伸缩、故障隔离和自愈等。

NFV的强大之处在于，它可以使许多以前手动且缓慢的流程实现完全自动化。这些过程包括将特定于服务的功能部署到网络中，这是许多预期物联网应用程序的基础。与当前的手动流程相比，使用完全自动化将服务特定功能部署到NFV网络中最合适的位置可以既便宜又快速。因为这样的优势，NFV也是是5G服务预期爆炸式增长的关键技术组件。当然，NFV并不局限于移动访问，这种过程自动化将对传统网络也产生了类似的影响。

## NFV的组成

NFV架构大致分为三个部分，如图1所示：

1. NFVI & VIM： NFV基础设施(NFVI)托管虚拟机和容器，并通过虚拟链接(VL)将它们连接在一起。VIM即虚拟基础设施管理系统，是VM的管理工具。
2. VNFs, NSs, and Network Slice: 第二部分是VNF本身的集合，包括VNF互连组合成网络服务（NS）以及NS的组合和共享以形成网络切片。VNF是托管在NFVI上的特定VM和容器的互连组合。
3. Management and Orchestration: 第三部分是管理和编排系统，控制VNF、NS和网络切片的生命周期，控制和维护它们的配置，并监控它们在生命周期中的健康和性能。



1. NFV架构及模块划分

## MANO与OSM

NFV 自概念阶段开始，与开放平台之间的密切关系以及开放标准的发展演进就是其主要推动力之一，因而 NFV 与开源的发展密不可分。目前已有大量开源解决方案可以提供MANO模块的完整功能，常见的有效新兴技术有OPNFV(Open Platform NFV)以及OSM(OpenSource MANO)。

OPNFV是Linux基金会提供的NFV开源解决方案。OPNFV集成了大量可以帮助其实现NFV模型的开源项目，2015年底OPNFV决定扩展其功能范围，计划将整个MANO功能块的工具和软件(包括NFVO和VNFM)都集成到OPNFV软件中。OPNFV大而全的发展战略带来了更全面的功能也带来了更大的体量，较为臃肿的平台体系对于小型NFV项目来说并不十分划算。

OSM(Open Source MANO)于2016年2月在世界移动通信大会(Mobile World Congress)上发布，旨在提供完全符合ETSI架构的MANO功能。OSM已经收到来自Telefonica的OpenMANO(前面曾经讨论过，包括VIM和NFVO功能)、Canonical以及Rift.io的贡献，与该行业相关的其他大公司(如Intel和Mirantis)也都是OSM的贡献者。

相较于OPNFV，OSM提供了完整MANO功能的同时更加轻量易用。对于小型NFV项目有着更好的适用性。

OSM是NFV第三部分的解决方案，但也不止于此。**OSM旨在支持最广泛的NFVI、VIM、WIM以及尽可能广泛的VNF、NS和网络切片。除了涵盖尽可能广泛的NFVI和托管功能和服务之外，OSM还是一个编排和管理系统，用于管理托管功能的生命周期、配置和运行时方面。**

## 内容概览

本文将详细讲述OSM的功能和范畴。OSM接口的这些其他系统包括NFV的其他部分以及非虚拟化网络功能和现有的OSS/BSS系统。

第二节简述了OSM与Kubernetes 的关系及区别。

第三节描述了OSM的一般架构，包括它的内部结构方式和它与其他系统接口的方式。OSM对服务和功能采用分层方法，并将其反映在自己的体系结构以及与其他系统的接口中。OSM在更高的服务层为客户端提供服务管理接口（OSM北向接口）按需构建服务。OSM通过使用来自较低服务层的服务来实现这一点，特别是通过提供VIM和WIM的服务管理接口来自NFVI。除了这个服务平台视图，本章还描述了OSM使用的平台操作视图，包括与其他系统的交互，这些系统支持管理和监视OSM控制的服务层。

第四节介绍服务层概念，详细介绍了OSM北向接口的范围和功能，客户端服务层可以通过这些接口请求和管理OSM管理的服务层的网络服务、组件VNF和网络切片特性。然后介绍OSM对通过各种VIM和WIM接口请求和管理的基础设施服务的异构组合提供的服务进行抽象的方式。

第五节介绍平台操作层，详细说明了安全管理的基本内务处理功能的范围和功能，包括身份验证和授权以及数据记录和将数据导出到外部系统。

最后，第六节描述了OSM集成的几个关键组件。

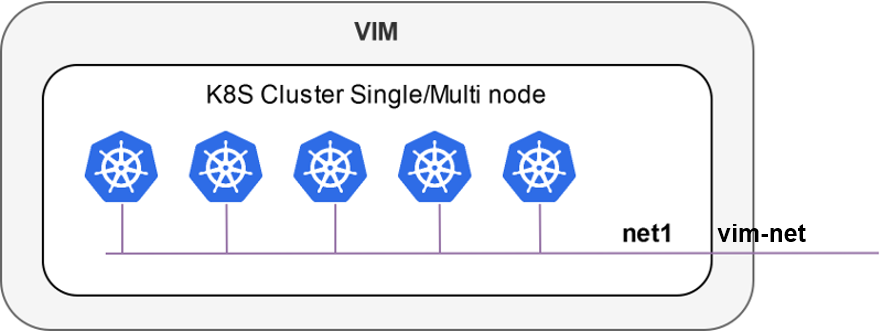
# OSM与Kubernetes

在谈论OSM之前，我们需要先知道K8S能做什么，为什么我们需要OSM。

Kubernetes用于管理容器化的工作负载和服务，使用声明式配置实现自动化运维。K8s为以容器运行的应用程序提供管理，满足扩展要求、转移故障应用、提供部署模式等需求，同时支持服务发现、负载均衡、自动部署、回滚以及自我修复等特性。K8S特性强大，但功能分散——由于 Kubernetes 是在容器级别运行，而非在硬件级别，它提供了 PaaS 产品共有的一些普遍适用的功能， 例如部署、扩展、负载均衡，允许用户集成他们的日志记录、监控和警报方案。 但是Kubernetes 不是单体式（monolithic）系统，那些默认解决方案都是分散的。Kubernetes 为构建开发运维平台提供了基础，而更进一步的集成化调用则由OSM以及类似功能的平台完成。

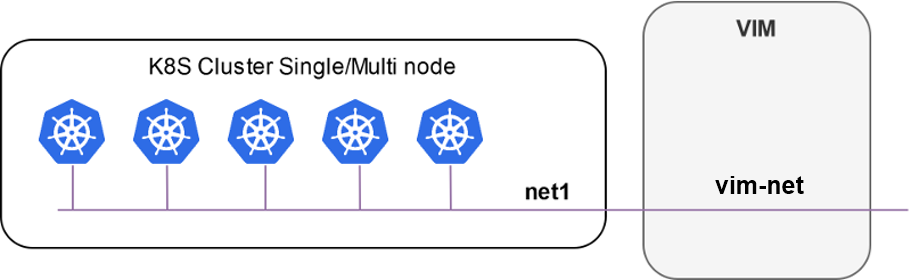
如上一章节所述，OSM 基于NFVI 和 VIM为托管的应用程序提供生命周期管理的能力。OSM将应用程序抽象为网络服务（NS），这种能力作为K8S功能的拓展与封装，使得应用部署、监控、自动化以及Day-2操作都可以得到一站式解决，在K8S的基础上避免了分散式配置。通俗来说，运维人员不必在K8S上分别去配置应用的各项功能，这些通过OSM就可以一站式解决。此外，OSM也提供了管理多集群的能力，不管是Openstack、公有云或是多个K8S集群都可以通过OSM进行统一管理。

OSM通过vim-net与VIM进行交互，在 OSM 中有两种与K8s 集群交互的模式。下图所示的交互模型中K8S集群被部署在诸如Openstack等虚拟基础设施中，此时OSM通过K8S依存的VIM层api与K8S进行交互。



1. K8s位于VIM内部

下图所示的交互模型用于K8S部署与裸机上的情况，此时没有vim层api可供使用，OSM通过创建一个逻辑VIM解决与K8S的交互问题，这样的设计使两种模式在逻辑上保持统一，更方便统一的多集群管理。



1. K8s位于VIM外部

经过上面的论述不难总结出OSM和Kubernetes的主要区别与联系：

* OSM位于Kubernetes上层，是Kubernetes的消费者；
* Kubernetes负责提供运行环境与功能，OSM负责功能的组织调用；
* OSM为Kubernetes的各种功能提供了一个统一组织调度的平台，一站式解决应用服务生命周期管理。

# OSM的功能与范围

ETSI OpenSourceMANO的目标是为电信服务开发社区驱动的生产质量端到端网络服务编排器（E2ENSO），能够建模和自动化真正的电信级服务，具有生产环境的所有内在复杂性。OSM提供了一种方法来加速NFV技术和标准的成熟，支持广泛的VNF供应商生态系统，并测试和验证编排器与其必须与之交互的其他组件的联合交互：商业NFV基础设施（NFVI+VIM）和网络功能（VNF、PNF或混合功能）

OSM的通过以下四个关键方面最大限度地减少集成工作：

### ETSI NFV信息模型——YANG models

**YANG**能够对网络功能（虚拟、物理或混合）、网络服务(NS)和网络切片(NSI)的整个生命周期进行建模和自动化，包括从初始部署（实例化、day-0和day-1）到他们的日常操作和监控（day-2）。OSM的IM完全与基础架构无关，因此可以使用相同的模型在各种VIM类型和传输技术中实例化给定的元素（例如VNF），从而使VNF模型的生态系统可以随时部署。

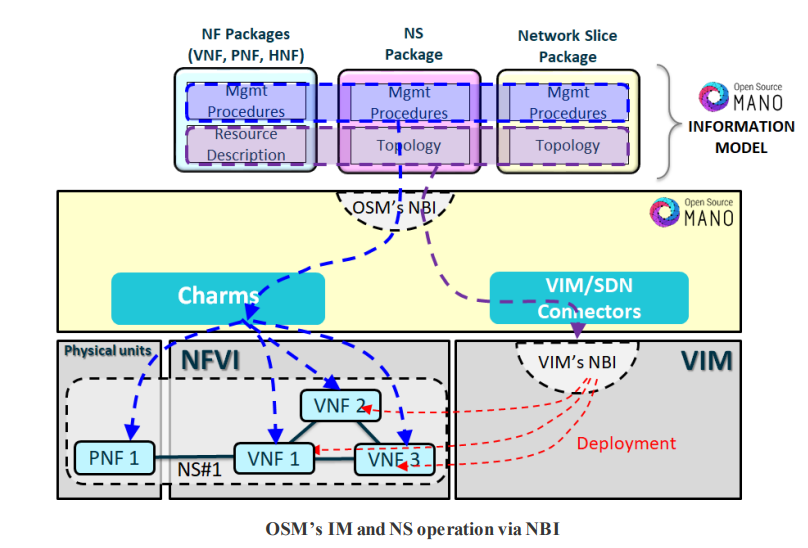
YANG数据建模语言在IETFRFC7950中定义。YANG语言用于对配置和状态数据进行建模。IETFRFC7950指出YANG表示的数据模型旨在由网络配置协议(NETCONF)操纵。自IETFRFC7950编写以来，基于HTTPREST的协议（例如IETFRFC8040(RESTCONF)）已被指定并采用用于基于YANG的模型。

YANG的命名没有使用驼峰命名法。相反，它使用kebab-case命名约定。因此信息模型中的属性（如defaultLocalizationLanguage）在YANG模型中将表示为defaultlocalization-language.YANG语言的第二点是它本质上是等级制的。因此，顶级容器代表其下所有子节点的父节点。因此，不需要在子节点中重复父节点的名称。父容器vnfd-element-group中的子节点vnfd-element-group-id将简单地重命名为id。某个属性的多个条目将表示为列表或叶列表。要识别此类列表中的条目，需要一个键。密钥通常被命名为“id”。因此，所有此类标识符在模型中都已重命名为“id”。具体消息结构参考[此文档](https://osm.etsi.org/wikipub/index.php/OSM_Information_Model)。

### 北向接口（NBI）

一般定义的SDN架构中，SDN的整体由下到上（由南到北）分为数据平面、控制平面和应用平面。其中，数据平面由交换机等网络通用硬件组成，各个网络设备之间通过不同规则形成的SDN数据通路连接；控制平面包含了逻辑上为中心的SDN控制器，它掌握着全局网络信息，负责各种转发规则的控制；应用平面包含着各种基于SDN的网络应用，用户无需关心底层细节就可以编程、部署新应用。控制平面与应用平面之间通过SDN北向接口进行通信，而NBI并非统一标准，它允许用户根据自身需求定制开发各种网络管理应用。

OSM基于NFVSOL005提供了一个统一的北向接口（NBI），使系统的全面运行以及在其控制下的网络服务和网络切片成为可能。OSM的NBI提供管理网络服务(NS)和网络切片实例(NSI)生命周期的服务，将所有必要的抽象作为服务提供，以允许通过以下方式完全控制、操作和监督NS/NSI生命周期客户端系统，避免暴露其组成元素的不必要细节。



1. OSM信息模型与北向接口

### 拓展的网络服务概念

与传统网络功能虚拟化中的网络服务不同，OSM对于网络服务的概念进行了拓展，不止包含传统电信网络中虚拟化的部分，而是更接近互联网中微服务的概念。OSM 中“网络服务”的扩展概念，使 NS 可以跨越已识别的不同域（虚拟、物理和传输），从而控制 NS 以无法区分的方式与 VNF、PNF 和 HNF 交互的整个生命周期 以及不同站点之间的按需传输连接。

### 网络切片管理

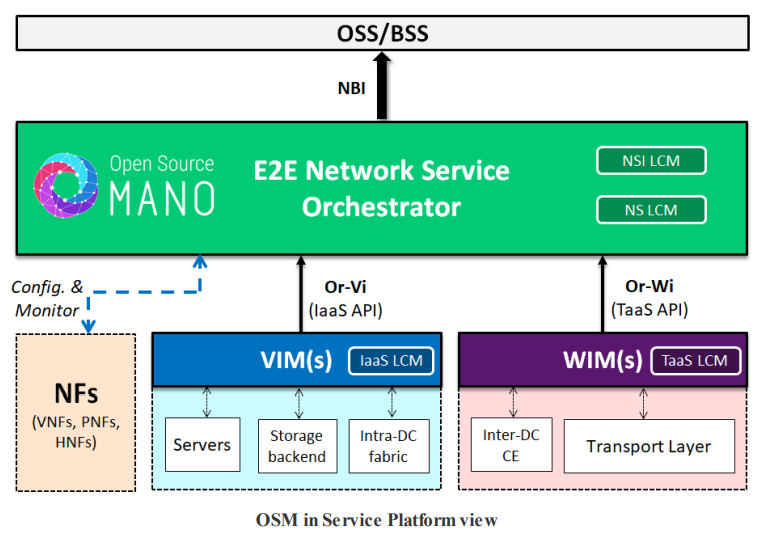
OSM 还可以管理网络切片的生命周期，如果需要还可以承担切片管理器的角色，并将其扩展为支持集成操作。

# OSM服务层面

OSM提供了NFV的主要能力之一：按需创建网络(“网络即服务”或NaaS)，以供服务提供商直接利用或潜在的向第三方商业化。OSM作为网络服务编排器(NSO)，网络服务平台的管理功能（详见服务平台视图和分层服务架构）旨在提供按需创建网络服务和返回服务的能力对象ID，通过后续调用OSM的北向API来控制网络服务的整个生命周期和操作，并以方便的方式监控其全局状态。

OSM能够按需提供两种类型的NaaS服务对象以支持NaaS功能：网络服务(NS)和网络切片实例(NSI)，后者由多个组件组成可以被视为单个实体的网络服务（两种类型的NaaS服务对象的特殊性将在下一节中描述）。

OSM作为服务平台的管理功能，消费来自其他服务平台的服务，并控制一些被管理的功能，以创建自己的复合更高级别的服务对象。因此，OSM使用由负责虚拟基础设施的平台和负责SW定义网络的平台（以获得所有必需的交互）提供的服务，并在组装后配置和监控组成网络功能（VNF、PNF、HNF），以便控制按需提供的整个NS/NSI的LCM。下图总结了OSM作为NFV服务平台架构的一部分的观点：



1. OSM服务层视角

## OSM提供网络服务 (NS)

网络服务(NS)是OSM中用于管理作为服务提供的网络的最小构建块，它将一组互连的网络功能（VNF、PNF和HNF）捆绑在一个服务对象中，这些功能可以跨越不同的底层技术（虚拟或物理）、位置（例如或多或少集中）和地理区域（例如作为大型跨国公司客户服务的一部分）。

为了有效地提供“按需服务”，OSM新创建的网络服务不是作为手工或临时过程的结果提供的，而是作为基于API调用的简单且公认的方法的结果（对OSM的NBI)和描述符遵循OSM的信息模型。这些描述应有助于创建由来自不同供应商的不同网络设备（VNF、PNF或HNF）组成的网络服务，以便这些设备（也称为网络功能或NF）可以由其提供商和服务提供商预先建模可以专注于网络服务本身的建模。

一旦网络服务完全建模，该模型就可以有效地作为模板工作（重复创建），该模板可以在NS创建时具体化（“参数化”）以合并该NS实例的特定属性，返回唯一的NS实例ID,有助于在后期驱动LCM操作。OSM还放置了所有必要的抽象，以允许客户端系统（通常是OSS/BSS平台）以规范化和可复制的方式完全控制、操作和监督NS生命周期。这个NS实例“处理程序”不需要公开其组成元素的不必要的细节，以尽量减少NF或NS拓扑中无意改变实际服务的潜在变化对最终服务的影响提供。

为了实现所需的灵活性和抽象级别，OSM增强了NS相对于ETSI NFV的概念，以合并物理和传输域，以实现可以扩展到虚拟域之外的真正E2E服务。因此，OSM提供了如下两点特性：

* 将虚拟网络功能(VNF)、物理网络功能(PNF)甚至由物理和虚拟元素（混合网络功能或HNF）组成的网络功能组合到一个NS中，更典型的是靠近接入网络的元素。
* 利用软件定义的网络平台的API，在分布式网络中部署此类NS，甚至按需创建站点间传输连接。

OSS 和 BSS 平台都被期望成为 OSM 按需创建的 NS 的消费者，有时甚至可以在需要时保持对 NS 的某些组成网络功能的控制（这对于重用遗留网络节点而不需要进行重大更改非常有用 在 OSS 中）。 出于这个原因，如果在 NS 模型中明确指定，OSM 还能够选择性地将 NS 的特定组成 NF 的控制委托给 OSS/BSS 平台，从而根据需要完全自由地支持遗留或混合场景。

## 网络服务（NS）的生命周期和操作

在接下来的章节中，我们将深入讨论和描述与端到端网络协调器中NS的生命周期和操作相关的各个阶段，以便在操作的上下文中更好地理解API功能(以及配套的IM建模语言)。

### （1）NS建模

OSM的信息模型提供了包含NS行为完整蓝图的机制，包括NS拓扑的完整描述、启用的生命周期操作、可用的NS原语及其自动化代码。由于网络服务根据定义由一个或多个异构类型和内部行为的网络功能（VNF、PNF或HNF）组成——并且可能来自不同的提供者——IM提供了一种让提供者描述内部行为的方法网络功能的拓扑结构、所需资源、过程和生命周期。这些信息捆绑在所谓的NF包中。

这种两层设计的模型可以防止NS包的设计者直接暴露于NF内部，并且可以专注于NS本身的组成，完全基于NF的外部属性和过程，使得NF与NS解耦合。以及在整个供应链中实现对NF及其配套NF包的一致且可复制的验证，以便NF供应商可以保证其元素始终以适当的方式使用和操作。因此，同一个NF包可以在多个NS中使用，而无需在NF级别进行额外的建模工作。

### （2）NS注入

一旦模型准备就绪，就可以将它们注入系统，以便稍后在称为入职的过程中将它们用作创建 NS 的模板。

OSM 的 NBI 提供 API 调用以支持对相应的 NS 和 NF 包的 CRUD（创建、读取、更新、删除）操作，以支持前面描述的两层建模方法（在网络的情况下可以变成三层） 切片），该 API 支持特定的 CRUD 操作，以将相应的 NS 和 NF 包（以及适用时的 NST）作为独立但相关的对象进行处理。 在这些操作中，特别是在入职步骤中，执行必要的检查以验证模型内和跨模型的一致性。

### （3）NS创建

一旦相应的 NS 和 NF 包在 OSM 中成功加入，就可以将它们用作实际 NS 创建的模板。 因此，OSM 提供 API 调用以支持与 NS 实例相关的 CRUD（创建、读取、更新、删除）操作。

在 NS 创建操作（也称为 NS 实例化）的情况下，OSM 将 NS 包和可选的一组附加部署约束（例如 NS 的特定 VNF 的目标部署位置）和参数作为输入 NS，正如 NS 包明确允许的那样。在创建 NS 的过程中，OSM 与南向的不同服务平台（VIM 和 WIM）和托管函数（NF）进行交互，创建 NS 实例的复合服务对象。

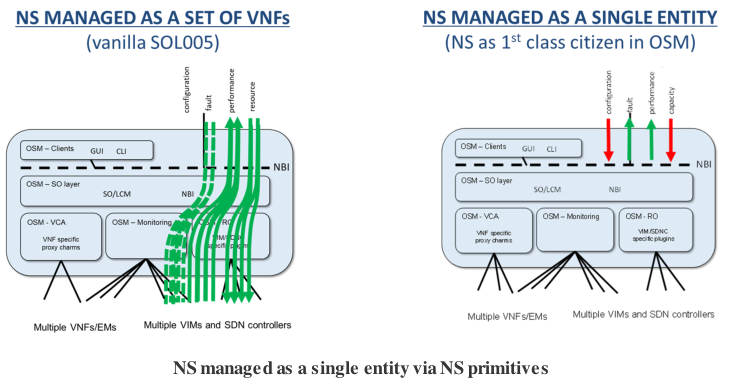
### （4）NS操作

成功创建 NS 后，NS 实例将成为进一步操作、生命周期和保证操作的唯一相关对象。

成功创建 NS 后，NS 实例将成为进一步操作、生命周期和保证操作的唯一相关对象。 NS/NSI 实例可能会受到不同类型的 API 驱动操作的影响，这些操作属于以下类别之一：

一般生命周期操作：必须有许多 API 调用允许触发可能适用于任何 NS 的众所周知的标准操作，例如缩放操作、暂停/恢复、按需监控请求、SW 升级等。

从 NS 原语派生的动作：除了可能适用于任何 NS/NSI 的操作之外，每个 NS/NSI 都可以有一组仅与 NS/NSI 提供的特定功能相关的操作，例如添加新用户、更改内部路由等。这些操作在相应的 NS 包中被枚举和编码（反过来，利用 NF 包中可用的原子操作）由 API 公开为给定 NS/NSI 中可用的主要操作。尽管客户端系统没有通过 API 直接请求，但必须注意，由于 NS 或 NF 包中定义的闭环策略，其他操作可以在 NS 内部触发。 通常，这些操作涉及监视 NS 或 NF 的某些参数，以及在达到给定阈值时触发上述操作之一（例如自动横向扩展）。



1. 网络服务(NS)管理

# OSM操作层

## 鉴权

OSM遵循GS NFV-SOL005 – V2.4.1标准，OSM提供的api只能被授权的消费者访问。因此OSM能够根据用户数据库对用户进行身份验证，并授权他们在项目上下文中执行一组操作。OSM 中的身份验证/授权方案可以是独立的，也可以与外部公共服务集成。 具体来说，在与其他组件进行集成部署的情况下，OSM 可以使用它们公开的协议/机制与身份验证/授权服务器进行交互。这种交互涵盖了从简单的用户身份验证到基于用户和项目的身份验证和授权，具体取决于外部系统提供的功能会有所不同。

必须注意的是，用户和项目管理的工作流程可能因情况而异。在独立部署的情况下，可以直接通过API创建和管理用户和项目；在集成案例中，用户和项目的管理可能存储在组织的用户数据库和系统中。

## 认证与授权管理

从实际运营的角度来看，OSM允许将OSM资源划分到不同的项目(也称为租户)中，为以下方面提供单独的空间:

* NF packages
* NS packages
* NST
* NS实例
* 网络切片
* 授权的VIM 帐户

一些通用资源（NF包、NS包和网络切片）可能会被声明为公开或可用于它们的列表。但是网络服务器和网络切片的实例只能属于一个项目。OSM使用基于RBAC的访问控制机制，允许在系统中定义用户角色，允许从整个操作集中获得特权或访问权限：

* NF 包的 CRUD 操作
* NS 包的 CRUD 操作
* NST 的 CRUD 操作
* 对网络服务实例的 CRUD 操作
* 对网络切片实例的 CRUD 操作
* 网络服务的高级 LCM（向外扩展/向内扩展）
* 网络服务或网络切片的Day-2操作
* 对 VIM 账户的 CRUD 操作
* 系统操作（用户、项目和角色管理）

系统的用户至少属于一个项目，根据项目的不同，每个用户可能具有不同的角色。 这就是通常所说的基于角色的访问控制（RBAC），基于每个项目和每个用户，即对于用户所属的每个项目，用户将拥有具有特定访问权限的角色。 必须注意的是，RBAC 机制通常涉及 admin/root 角色（具有所有权限）和 admin 用户的定义，在具有 admin/root 角色的所有项目中获得授权。

OSM 的客户端平台以项目为单位进行认证和授权，并根据在项目中的角色获得令牌。 该令牌标识后续操作的设置权限。客户端平台将使用该令牌进行API 调用。 根据令牌标识的授权权限，某些操作将被允许，而另一些操作将被禁止。

## 资源管理

网络服务和网络切片实例的创建需要 OSM 预先提供足够的信息来部署和操作它们。 其中一些输入（NF 包、NS 包和 NS 模板）的生命周期管理完全由 OSM 管理。 其余输入的生命周期由其他服务平台管理：

VIM 帐户——为了能够部署在由 VIM 管理的特定数据中心，OSM 需要知道并存储要使用的 VIM 帐户的访问信息和凭据。这些凭据由 VIM 管理员创建，并且应该授予 OSM 足够的权限来对风格、网络、图像、卷和实例执行 CRUD 操作。 这些 VIM 帐户需要先添加到 NSO，再添加到该数据中心的任何部署中。

VIM 帐户的管理网络——OSM 部署的每个 VNF 都需要可访问以进行配置和操作，这意味着至少有一个 VNF 接口必须连接到管理网络，其 IP 地址可从 OSM 访问。 这不仅需要在 VIM 中创建网络，还需要保证网络可以从将添加到 OSM 的 VIM 帐户使用，并且可以通过 OSM 的 IP 访问。

SDN控制器帐户和相关端口映射信息——OSM必须通过SDN控制器(SDN Assist)管理数据中心上的底层网络，OSM需要知道访问信息和凭据，以到达SDN控制器并执行与底层网络相关的CRUD操作。SDN控制器及其帐户的设置以及与数据中心基础设施的连接需要在供应阶段进行，通常由数据中心的管理员进行。然后，需要将SDN控制器访问信息和凭证添加到OSM中，并在由该VIM控制的数据中心中进行任何部署之前将其关联到特定的VIM。此外，OSM还需要知道计算节点接口之间的映射关系，例如元组(计算节点，物理接口)和SDN端口之间的映射关系，例如元组(交换机，端口)。这种映射将保证从OSM到VIMs和SDN控制器的操作的一致性。

PDU——在我们想要定义 NS 包或由 PNF 包组成的 NST 的情况下，OSM 会收到有关可用 PDU 的指示，即网络设备将成为这些 PNF包的构建块。 我们需要提供这些 PDU 的管理 IP 地址、类型、位置 (VIM) 以及与站点的物理连接（例如交换机和基础设施的端口）。

PNF 和外部网络——网络服务（或网络切片）需要连接到不属于网络服务本身的现有网络或 PNF 的情况下，我们需要向 OSM 提供适当的信息以将这些实体连接到我们的网络服务。 一种可能性是将 OSM 中的这些实体添加为 VIM 的外部网络，以便它们可以在实例化时映射到网络服务中的虚拟链接。 此外，如果 OSM 控制底层网络的创建，则有必要提供 PNF 或外部网络连接的数据中心交换机和端口。

## 安全机制

OSM的北向接口提供以下安全功能：

* + 来自客户端的认证和授权访问。
  + 每个项目隔离目录，但一些目录资源可以在项目之间共享。
  + 基于TLS的安全北向接口。

OSM还提供了阻止恶意访问的机制，恶意访问可能危及系统和系统存储的数据。例如，OSM的内部架构准备遵循商业发行版和安装程序中的常见安全实践，比如：

* 不将密码存储在文本文件中
* 不将默认用户和密码存储在内部数据库中
* 对输入到系统的帐户信息进行适当加密(例如，来自VIM帐户的密码)
* 不以root用户直接在系统主机中运行进程
* 只有当内部模块运行在不同的服务器上时，它们之间的通信才安全且经过身份验证。

# OSM集成

OSM并非一个孤立的平台，许多优秀的开源项目为OSM提供了各方面的功能特性。Canonical开源的juju框架是OSM关键功能——服务生命周期管理的执行者，通过juju可以完成各种环境中应用的操控。Operator是K8s集群中应用操作的解决方案，OSM通过使用Operator拓展利用定制资源管理应用及其组件。Charm则支持了非K8s环境中应用的生命周期管理工作。除以上几个关键项目外，OSM还集成了诸如LXD，Prometheus，Mongodb等优秀开源项目，共同构成完整的OSM平台，篇幅限制不再赘述。

## Juju

Juju是由Canonical开发的免费开源应用程序建模工具，旨在通过在公共和私有云服务以及裸机服务器和本地基于容器的部署上促进、部署、配置、扩展、集成和执行操作任务来减少软件的操作开销。

### 什么是juju

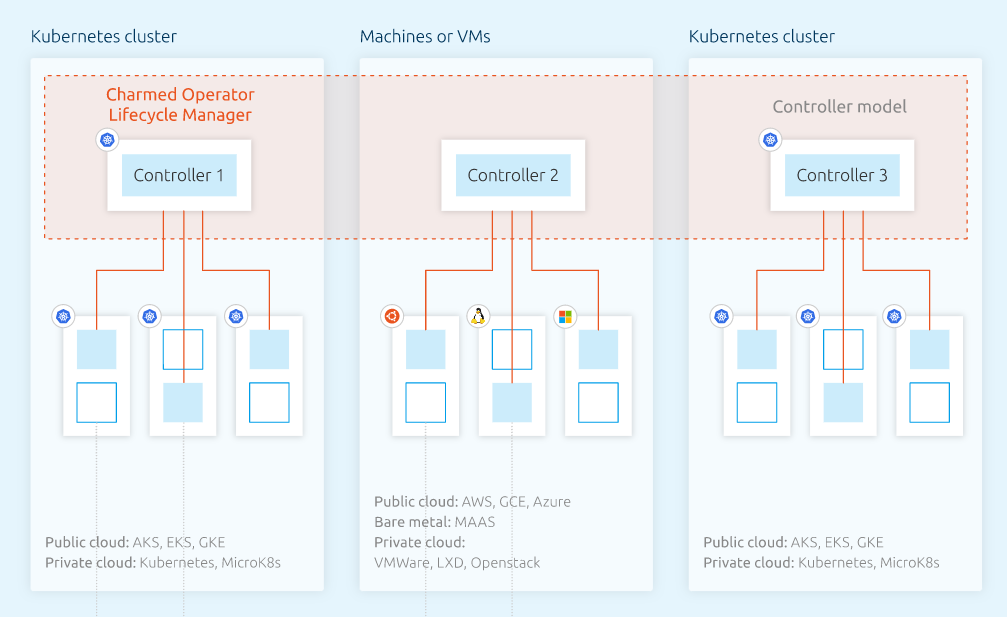
Juju 旨在提供一种建模语言，抽象出操作复杂软件拓扑的细节，以降低操作成本并提供灵活性。 Juju 模型是管理和操作一组软件应用程序的环境。 模型可以在各种公共云上运行。

Charmed Operator是Juju对Kubernetes运算符概念的扩展和概括。Kubernetes中的Operator是一个应用程序，其中包含应用安装和在Kubernetes集群上维护和升级它所需的所有操作知识。Charmed Operator（“Charm”）是对Operator的拓展，其中包含安装，维护和升级，将其与其他应用程序集成以及在Kubernetes集群、容器上所需的所有操作知识。

当应用程序运行环境变得复杂，比如跨虚拟机、Kubernetes、混合云和多云部署和管理应用程序时，运营人员很容易迷失在大量的 YAML配置文件、图表、操作手册、部署计划、执行脚本等琐碎文件中。Juju作为一个“驱动软件的软件”，可以帮助运维人员控制所有应用程序、基础架构和运行环境。Juju可以帮助用户节省许多的脚本管理时间，最小化部署成本，确保应用的冗余与弹性，将运维人员配置管理中解脱出来，实现真正的面向应用程序管理。

### Juju的功能

一个简单的例子，在混合云环境的 Kubernetes 和虚拟机上运行来自应用程序、数据库、监控组件等的工作负载。



1. Juju在混合云中使用

Juju 使用 Charms和 Charmed Operator Lifecycle Manager 来控制混合云中这些工作负载的部署、升级、集成、管理和操作。Charms 是封装了常见维护功能的小型应用程序，可将Day-0到Day-2的操作变成可重复且可靠的代码。这使得运维人员能够管理应用程序和场景，而不是专注于配置。

使用 Charms不仅可以部署或管理单个应用程序，还可以在“模型”中将它们相互关联，以处理扩展、管理和跨服务依赖性。 应用程序模型定义了哪些应用程序提供服务以及它们如何相互关联。模型、跨模型关系和模型驱动的操作使用户能够控制大规模、跨混合云、通过juju处理部署和操作。

Juju控制器是跨多个模型跟踪事件、状态和用户活动的服务。 控制器和模型类似于数据库服务器和服务器上可用的数据库。 每个模型都可以有不同的配置、操作软件集和具有不同级别访问权限的用户。 模型的示例包括 Web 应用程序、负载均衡器和“web 应用程序”模型中的数据库。 模型允许将部署隔离到逻辑解决方案中并单独管理。

## Kubernetes Operator

Operator 是 Kubernetes 的扩展软件， 它利用定制资源管理应用及其组件。Operator 模式 旨在记述（正在管理一个或一组服务的）运维人员的关键目标。 这些运维人员负责一些特定的应用和 Service，需要清楚地知道系统应该如何运行、如何部署以及出现问题时如何处理。在 Kubernetes 上运行的工作负载都喜欢通过自动化来处理重复的任务。 Operator 模式则是用于封装用户编写的Kubernetes 本身提供功能以外的任务自动化代码。Operator 模式概念允许你在不修改 Kubernetes 自身代码的情况下， 通过为一个或多个自定义资源关联控制器来扩展集群的能力。

## Charms

Charm是Kubernetes Operator概念的扩展和概括，也叫做Charmed operator. Charmed operator.是Kubernetes运算符概念的扩展和概括。

在Kubernetes中，Operator以容器形式存在，用于驱动应用程序的生命周期管理、配置、集成和日常操作。它处理特定于该应用程序的实例化、扩展、配置、优化、网络、服务网格、可观察性和Day-2操作。本着运营商应该“做好一件事”的原则，每个运营商驱动一个应用程序或服务。但是，它可以与其他运营商组合以提供复杂的应用程序或服务。由于Operator以可重用和可共享的形式打包专用知识，因此它们极大地简化了软件管理和操作。

在Juju中，Operator可以完成所有这些工作，但还支持更多的用途和更多的基础设施：使用Charm（由Juju的Charmed Operator Lifecycle Manager(OLM)协调）不仅可以部署应用程序，还可以将其连接到其他应用程序，并且还可以在Kubernetes集群，以及任何公有云或私有云上使用容器、虚拟机和裸机。

Charm按照运行环境可分为两类。

### （1）机器Charm

Juju一开始着重于在云优先世界中简化复杂应用程序和服务的部署。当时，其中许多应用程序在虚拟机或裸机服务器上运行。机器Charm可以部署到许多不同的底层计算/存储资源提供方上：

* 裸机（使用MAAS）
* 虚拟机（使用Openstack中的KVM、AWS中的EC2、VMware环境等）
* 容器（使用LXD集群）

机器Charm可以具有从属概念，当一个机器Charm不是从属的时候，它被称为主要的。基本区别在于，从属Charm部署在与其附属的主Charm相同的单元上，而主要Charm部署在新单元（即它自己的机器）上。创建从属Charms是为了支持开发可以与现有Charms一起部署的Charms，以使用原始应用程序Charm中可能未包含的特定功能来增强它们。从属Charm取决于与主要Charm的关系而创建，不会作为独立应用程序被部署。

一个例子：考虑部署一个扩展到n个副本的大型Web应用程序。在juju中，每个Charm实例都包含一个单元，而App是具有相同名称的给定Charm的所有单元的总和。当一个大型web应用扩展到n个副本时，n个单元将由Juju启动。Web应用程序的管理员可能希望从应用程序服务器收集日志。应用程序Charm的开发人员可能没有包含从与管理员特定环境兼容的服务转发日志的机制。通过使用诸如rsyslog-forwarder之类的从属Charm，管理员可以确保其Web应用程序的每个部署单元都自动配置为使用rsyslog转发日志。从属Charms的编写方式与其他Charms相同，只是在Charm的元数据中添加了一个额外的从属标志。

### （2）Kubernetes Charms

JujuOLM在Kubernetes上引入了对CharmedOperators的支持。Juju可以为部署在Kubernetes上的应用程序带来同样的好处，方法是将运算符与工作负载放在一起以在整个生命周期内对其进行管理。当使用Juju创建模型时，会创建相应的Kubernetes命名空间。当在Kubernetes上部署一个Charm时，它被部署为一个StatefulSet，它管理一组运行指定应用程序容器的Pod以及一个包含Charm代码的边车容器。

KubernetesCharms的分类基于“Charm操作的是什么”。一些Charms操作在容器上运行的工作负载；一些Charms操作Kubernetes资源，而另一些Charms操作juju资源。分别对应下文所述的sidecar、podspec和workload-lessCharm类别：

**Sidecar：**根据定义，sidecar模式旨在允许使用额外的功能和特性来增加工作负载。在这种情况下，该模式由一个小型辅助容器实现，该容器位于与您的应用程序相同的Pod中，它提供操作功能——这正是Juju在其他环境中的运作方式，也是Kubernetes社区中公认的模式。

通过利用这种模式，我们确保在每个工作负载单元旁边始终有一个操作员，无论应用程序如何扩展。操作员将始终可以直接访问共享内存、相同的网络名称空间以及根据需要操纵工作负载以保持其平稳运行的能力。这种方法简化了上游或第三方应用程序映像的操作，使管理员能够根据需要在运行时进行更改以适应他们的环境，而无需维护一组定制的容器映像。

对于Charm开发人员来说，这些好处是通过使用Pebble来管理工作负载来实现的。Pebble是一个轻量级的、API驱动的流程主管，专为与CharmedOperators一起使用而设计。Pebble使Charm开发人员能够定义他们希望工作负载如何运行，并提供一个API，使操作代码能够在整个生命周期中管理工作负载。

**PodspecCharms：**从jujuv.3.0开始，podspecCharms被弃用。

**Workload-less：**Charms是交换数据的进程。当用户希望在不必重写或重新配置Charms的情况下操作该数据；这种情况下可以实现一个“中间Charm”模式，该模式位于两者之间并像某种过滤器/代理一样操作关系数据。

# 现存的问题

### （1）有状态应用热迁移

目前为止尚未有支持Kubernetes集群中应用热迁移的技术，如何解决Kubernetes上有状态应用的热迁移是需要解决的主要问题之一。已知的技术路线有基于Kubernetes高可用机制以及基于CRIU提供pod级别应用热迁移。目前也在积极探索其他的解决方案。

### （2）智能编排算法

OSM和Kubernetes基于metris触发机制的编排算法在实际应用过程中不够灵活，支持智能编排算法的接入对于应用的持续可用性也起着至关重要的作用。开发一款智能编排算法以及使算法有效作用于应用编排也是目前需要解决的问题。