MEF Standard

MEF 120

Lean NFV Overview and Framework

May 2022

Disclaimer

The information in this publication is freely available for reproduction and use by any recipient and is believed to be accurate as of its publication date. Such information is subject to change without notice and MEF Forum (MEF) is not responsible for any errors. MEF does not assume responsibility to update or correct any information in this publication. No representation or warranty, expressed or implied, is made by MEF concerning the completeness, accuracy, or applicability of any information contained herein and no liability of any kind shall be assumed by MEF as a result of reliance upon such information.

The information contained herein is intended to be used without modification by the recipient or user of this document. MEF is not responsible or liable for any modifications to this document made by any other party.

The receipt or any use of this document or its contents does not in any way create, by implication or otherwise:

a) any express or implied license or right to or under any patent, copyright, trademark or trade secret rights held or claimed by any MEF member which are or may be associated with the ideas, techniques, concepts or expressions contained herein; nor

b) any warranty or representation that any MEF members will announce any product(s) and/or service(s) related thereto, or if such announcements are made, that such announced product(s) and/or service(s) embody any or all of the ideas, technologies, or concepts contained herein; nor

c) any form of relationship between any MEF member and the recipient or user of this document.

Implementation or use of specific MEF standards, specifications, or recommendations will be voluntary, and no Member shall be obliged to implement them by virtue of participation in MEF Forum. MEF is a non-profit international organization to enable the development and worldwide adoption of agile, assured and orchestrated network services. MEF does not, expressly or otherwise, endorse or promote any specific products or services.

Table of Contents

List of Figures

List of Tables

1 List of Contributing Members

The following members of the MEF participated in the development of this document and have requested to be included in this list.

- Amartus
- Lumen Technologies
- Nefeli Networks
- Spectrum Enterprise
- Verizon

2 Abstract

Lean NFV represents an approach to implementing Network Functions Virtualization in the context of MEF LSO. NFV solutions have become fundamental in moving network functions to the Edge (where by Edge we mean a compute stack not in the cloud core, up to and including the premises edge). Lean NFV offers a uniform approach to integrating different components involved in NFV using MEF LSO APIs.

Lean NFV takes an approach of reducing the difficult task of integration that usually accompanies disaggregation, relying on a Key-Value Store (KVS)-like function as the universal point of integration and using a plug-in approach to the computing infrastructure. Lean NFV reduces reliance on monolithic controllers and other NFV elements so that independently developed components work together, whether they run in customer premises equipment (uCPEs), the network edge (public or private), data centers, or even the cloud core.

The purpose of this document is to define Lean NFV in the context of the MEF 55.1 LSO Framework [3]. In this document, Lean NFV effects the functions of Infrastructure Control and Management. In this context it serves as a vehicle to instantiate two LSO APIs: Presto and Adagio. This document introduces the concept, provides a framework for the details (internal APIs and schema), and establishes principles that guide its design. The goal is to enable network operators to offer more customized services to their customers and to transform their businesses to adopt a software-first mindset.

3 Terminology and Abbreviations

This section defines the terms used in this document. In many cases, the normative definitions to terms are found in other documents. In these cases, the third column is used to provide the reference that is controlling, in other MEF or external documents.

Term	Definition	Reference
API	Application Programming Interface	This document
Application Programming Interface	A computing interface that defines interactions between multiple software intermediaries	MEF 55.1 [3]
CNF	Containerized Network Function	This document
Containerized Network Function	A VNF or VNF component designed to be deployed and managed on Container Infrastructure Service (CIS) instances	ETSI NFV Release 4 [6]
Infrastructure	The computing elements and communication fabric in an implementation of Lean NFV	This document
Kubernetes	An open-source system for automating deployment, scaling, and management of containerized applications	Linux Foundation [5]
KVS	Key-Value Store	This document
Key-Value Store	A well-known term in the industry for a data structure designed for storing, retrieving, and managing associative arrays, where data objects are stored, indexed by keys. In the context of LSO, an implementation of Infrastructure Control and Management (ICM).	This document
Lean NFV	An open architecture approach to NFV based on a single point of integration and a common interface to that point of integration for all NFV components	This document
LRS	Lean NFV REST Server	This document
Lean NFV REST Server	A REST server that provides CRUD (Create, Read, Update, Delete) and watch access to the KVS, and performs validation on all write operations as well as standard hooks for authentication, authorization and audit	This document
MANO	Management and Orchestration	This document
Management and Orchestration	Framework for the management and orchestration of all resources in a virtualized data center	[10]
Network Function	A network-related operation, which includes VNFs and CNFs but in this document not physical network functions (PNFs)	This document
NF	Network Function	This document

Term	Definition	Reference
Network Function Virtualization	A network architecture concept that uses the technologies of IT virtualization to virtualize entire classes of network node functions into building blocks that may connect, or chain together, to create communication services	ETSI NFV White Paper [2]
NFV	Network Function Virtualization	This document
NFVI	NFV Infrastructure	This document
NFV Infrastructure	The physical resources and accompanying software where computing, storage, and networking resources with which VNFs are deployed	SDxCentral NFV Definitions [13]
NFVO	NFV Orchestrator	This document
NFV Orchestrator	A component of MANO that orchestrates compute, storage, and network resources for VNFs and virtualized network services	SDxCentral NFV Definitions [12]
SD-WAN	An acronym for software-defined networking (SDN) in a wide area network (WAN)	MEF 70.1 [4]
VIM	Virtual Infrastructure Manager	This document
Virtual Infrastructure Manager	Management component responsible for controlling and managing the NFV infrastructure (NFVI) compute, storage, and network resources, usually within one operator's infrastructure domain	SDxCentral NFV Definitions [7]
VNF	Virtualized Network Function	This document
Virtualized Network Function	A software implementation of a network function that can be deployed on a Network Function Virtualization (NFV) Infrastructure	ETSI NFV White Paper [2]
VNFM	VNF Manager	This document
VNF Manager	A component of MANO that provides lifecycle management for VNFs	SDxCentral NFV Definitions [11]

Table 1 – Terminology and Abbreviations

4 Compliance Levels

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 (RFC 2119 [8], RFC 8174 [9]) when, and only when, they appear in all capitals, as shown here. All key words must be in bold text.

Items that are **REQUIRED** (contain the words **MUST** or **MUST NOT**) are labeled as **[Rx]** for required. Items that are **RECOMMENDED** (contain the words **SHOULD** or **SHOULD NOT**)

are labeled as **[Dx]** for desirable. Items that are **OPTIONAL** (contain the words **MAY** or **OPTIONAL**) are labeled as **[Ox]** for optional**.**

5 Introduction

The advantages of virtualization, long valued for computing, are now being applied to networking, generally under the rubric of Network Functions Virtualization, or NFV. These benefits include greater vendor choice from unbundled solutions, greater flexibility with independent software for control and management including centralized control governed by policy, the price-performance advantages of commercial off-the-shelf (COTS) hardware platforms, and greater network visibility. Since the concept was introduced in 2012 [2], NFV solutions have become fundamental in moving network functions to the edge. The corresponding Virtualized Network Functions (VNFs) or Containerized Network Functions (CNFs) now span across all major networking functionalities, including (or especially) security. At the same time, adoption has been hampered by a number of integration and flexibility challenges:

- Current NFV solutions are complex to deploy, because functions such as the Virtual Infrastructure Manager (VIM) are often implemented in closed and monolithic packages closely coupled to the physical infrastructure rather than its abstraction, preventing an NFV-related task from being treated as just another workload.
- Components within a monolithic controller cannot be replaced, or chosen, by the customer.
- The interfaces within a monolithic controller are specific to the solution and subject to proprietary development and innovation schedules.

NFV solutions are complex to automate because coordination is done through a number of pairwise APIs that, even if standard, are specific to the components they connect and their function. Specific examples include:

- functions such as chaining, configuration, scaling, and failover implemented by NFV managers and associated VNFs

- the components of the VNF manager(s).

Rather than a single monolithic controller (MANO), Lean NFV offers a collection of "micro-controllers", each tackling one aspect of NFV management and orchestration (such as configuration, scaling, or monitoring).

Lean NFV is an approach to NFV based on a common interface to a single point of integration for all NFV components. The single point of integration operates like a Key-Value Store (KVS), and a Lean NFV REST Server (LRS) abstracts its operation and communicates with all the NFV components using the common interface.

Lean NFV is targeted at operators (including service providers, such as SaaS companies, communication service providers, digital service providers, private cloud operators, and public cloud providers) looking to build new services utilizing NFV technology on their choice of bare-metal, hypervisor-based, or container-based realizations; system integrators looking to provide professional services in support of these implementations; and designers of VNFs looking to enable deployment of the new services. By implementing the standard Lean NFV API, VNF and micro-controller designers can develop products ready for widespread inclusion in dynamic service chains without modification of such common tasks as placement, launching, performance and health monitoring, and licensing that are not specific to the functions of the VNF or micro-

controller. (Tasks specific to these functions, such as producing a set of rules for a firewall or a set of paths for a router, still require specific knowledge on the part of a software module or human operator.)

Note that Lean NFV applies equally to both VNFs and CNFs so throughout this document any reference to VNFs implicitly includes CNFs.

- Section 6 in this document describes the Lean NFV architecture, relates it to ETSI NFV, and describes the objectives of this standard.
- Section 7 articulates the design principles.
- Section 8 positions Lean NFV in the MEF LSO architecture.
- Section 9 describes the business drivers.
- Section 10 provides some relevant use cases.
- Section 11 lists some useful references.

6 Lean NFV Architecture

Lean NFV's open architecture is designed to manage a multi-vendor ecosystem for NFV. As outlined in the MEF Lean NFV white paper [1], the core observation underlying the Lean NFV approach is that the basic NFV components – the NFV Management and Orchestration (NFV MANO), which includes the Virtualized Infrastructure Manager (VIM), the VNF Manager (VNFM), and the NFV Orchestrator (NFVO), the NFV Infrastructure (NFVI), and the Virtualized Network Functions (VNFs) – are well understood as individual components, but that current NFV designs have failed to integrate them in a simple and consistent manner and have treated them too monolithically. To enable greater flexibility and the choice of best-of-breed elements, Lean NFV deconstructs the larger control components into smaller ones called micro-controllers and introduces a new component to the NFV architecture: a KVS-like function that is wrapped by a Lean NFV REST Server (LRS). The KVS, accessible through the LRS, serves as the universal point of coordination between all other components, specifically the VNFs and micro-controllers. It stores status, configuration, and operational parameters that the VNFs and micro-controllers need to effect control, management, and monitoring. VNFs and micro-controllers can read data from or write data to the KVS using a standard API and without knowing or having to directly communicate with other entities that might use the data. (The ETSI NFV architecture [6] defined pairwise APIs that were specific to the function of the components that communicated with each other, making it hard to deploy and innovate.) The LRS API provides a clean single interface with which to exchange information between any pair of components.

6.1 Key components

The key components of the Lean NFV architecture are:

- The Lean NFV REST Server (LRS), which is the main point of integration, assuring authentication, authorization, and accounting (AAA) that abstracts the KVS-like function of the data store.
- The KVS-like function, which acts as a data store for state and configuration information needed by all components. No implementation is mandated or implied in this document. Other data stores are possible but a KVS-like storage offers the advantages of lightweight code, efficient change management, and easy implementation for high availability and disaster recovery.
- A virtual switch that is deployed for data-plane connectivity.
- The management micro-controllers that are modular in nature and together perform the NFV Orchestrator (NFVO), VNF Manager (VNFM), and Virtual Infrastructure Manager (VIM) functions.
- The VNFs and CNFs and any necessary corresponding Element Management Systems (EMS), all of which can come from any source.
- The common API by which the LRS, micro-controllers, and NFs communicate. Inherent in this API standard are the data models for conveying the data to and from the LRS.

The LRS stores data objects used by the VNFs and the micro-controllers. There are three main categories of such data objects:

1. Those that are understood by the micro-controllers but not the VNFs, such as where to place a VNF or what it takes to scale VNFs or create new instances, in other words objects not related to the specific nature and function of a VNF. This feature fosters the development of micro-controllers that perform functions on VNFs regardless to which specific VNFs they apply.

2. Those that are understood by the VNFs but not the micro-controllers, such as firewall rules or other details specific to the nature and function of a VNF. This feature fosters the development of VNFs that do not need to be customized for their operating environment.

3. Those that are understood by both the VNFs and the micro-controllers, such as where licenses are found and certain high-level configuration states. This feature enables VNFs to deal with some generic tasks in a consistent manner.

Note that the KVS might not be the best place to directly store fast-changing or high-volume telemetry data. It would be better suited for a time-series database, of which there are many examples. The KVS might store a pointer to that database to include information on where to find the database, which metrics are relevant, and possibly credentials for accessing the database.

In the figure below, a simple KVS is shown along with a subset of its features. Note that the data shown in the Values column is not uniform in type or extent. This feature makes the KVS suitable for storing state, configuration information, and other widely varying types of data that might be associated with a wide variety of VNFs and used by a wide variety of micro-controllers, as mentioned in the bulleted list above. VNFs and micro-controllers can write data to and read data from the KVS without communicating directly with each other and without knowing how each other might be implemented.

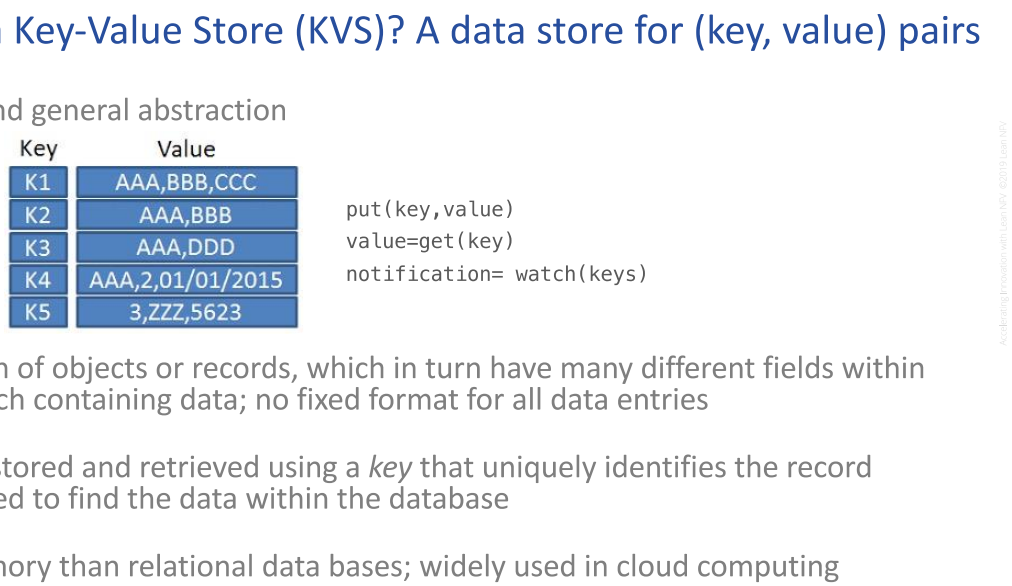

Figure 1 – Key-Value Store

The overall Lean NFV architecture may be viewed in Figure 2 – Lean NFV Sample Architecture. The infrastructure piece shown in the figure incorporates the compute, storage, and networking

infrastructure on which the Lean NFV components run. There is no special interface to the infrastructure for Lean NFV so the thick arrow between virtual switch and the infrastructure represents the usual data paths between compute, storage, and networking elements as designed for the performance required.

Introduction of a new, independently created VNF involves describing it (abstractly) in the KVS, where the appropriate micro-controllers can both discover it and use it as desired. Once licensed, placed, and properly configured, it becomes available for inclusion in service chains. Similarly, introduction of a new, independently created micro-controller requires no effort of the part of the VNFs; the micro-controller discovers the VNFs it can operate on and they respond as they would to any micro-controller, through the abstraction of the KVS. Lean NFV does not change the need to properly instantiate VNFs or micro-controllers; it only simplifies their interaction.

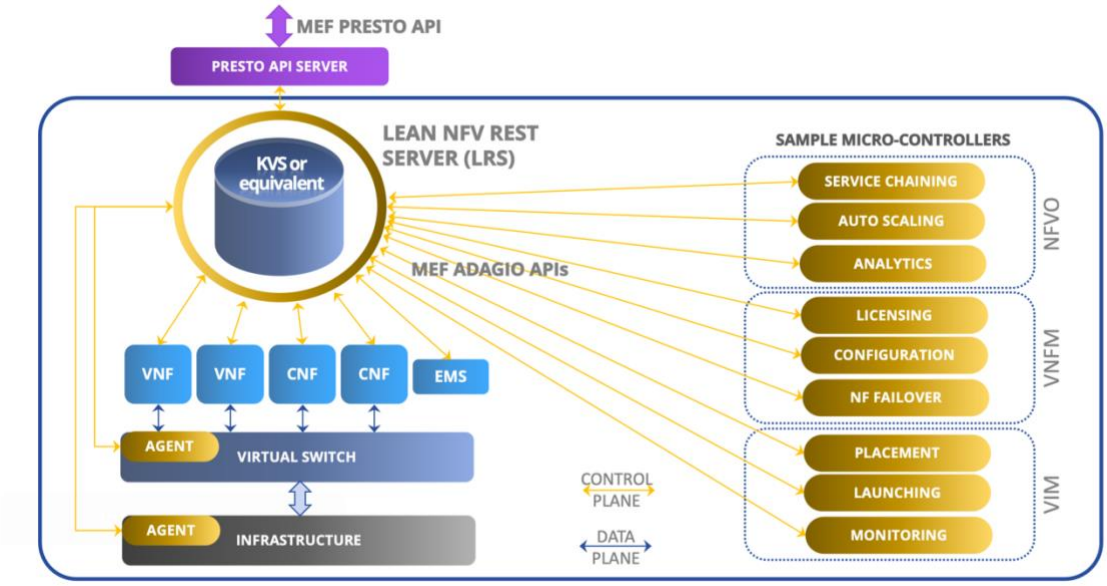

Figure 2 – Lean NFV Sample Architecture

6.2 Lean NFV in relation to ETSI NFV

NFV was launched in ETSI in 2012 [2] and Lean NFV derives the principles of NFV from the ETSI effort. The architectures differ in that ETSI NFV compartmentalized the components into functional groupings and defined a number of bilateral APIs between them, resulting in considerable effort to introduce independent components, many of which have to be custom designed for their deployment environment. Lean NFV relies on a single API for all components, resulting in their simplified design and reduced customization. This makes it particularly easy to dynamically create arbitrary service chains by instantiating library NFs as needed. The functional groupings of VIM, VNFM, and NFVO are shown in the above figure only to help those familiar with ETSI NFV relate Lean NFV to it. The micro-controllers shown therein represent independent entities all sharing the same API to the LRS, so the functional groupings carry no particular meaning and can be disregarded as elements of the Lean NFV architecture.

6.3 The scope of this Standard

This document (Lean NFV Overview and Framework) introduces the concepts and terminology of Lean NFV and positions Lean NFV in the LSO architecture, specifically showing that the northbound API from the LRS is an instance of MEF LSO Presto and the southbound API from the LRS is one or more instances of MEF LSO Adagio.

This document introduces the notion that VNFs can use the LRS APIs to facilitate their runtime management, meaning common runtime functions like configuration, monitoring, and scaling but it does not describe how they do this. Ideally, these functions would be implemented by a general NFV manager. However, in many of today's VNFs these functions are supplied by vendor-specific Element Management Systems (EMSs). As shown in Section 7 it is important that any easily deployable yet future-proof architecture support both modes of management. Accordingly, Lean NFV is designed to allow developers of new VNFs to build new streamlined ones (streamlined because they outsource common runtime tasks to general managers) while also allowing developers of existing VNFs to incrementally transition to using general NFV management. This allows operators to pick and choose between the above modes of management on a per-VNF basis and to gracefully integrate Lean NFV into existing NFV infrastructures.

6.4 Outside the scope this Standard

This Standard does not include the schema of the Lean NFV APIs (LSO Adagio and Presto), sample functionality for the microcontrollers, sample code for a Lean NFV network function, the specific data types stored in the KVS, or the specific data models by which the data are represented. (This standard introduces only the framework and architecture of Lean NFV and the general categories of data stored in the KVS.)

Beyond the scope of this Standard is the minimal set of changes to VNFs needed to simplify basic runtime management tasks such as configuration, licensing, and basic monitoring (which in turn can trigger actions such as scaling, healing, and chaining), as well as additions to enable more advanced management capabilities, such as diagnostics, analytics, and state management.

Also not included in this standard are supporting tools that ease the development of VNFs that are natively adapted to Lean NFV (meaning they are built to use Adagio to communicate with the LRS), such as:

- A tool that accepts, as input, descriptions in either source code form or in an interface definition-language (IDL) such as Protobuf and generates code to generate and register a variety of data representations and to automatically encode data using these representations.
- A tool that accepts, as input, definitions of high-level data objects and generates an implementation using the low-level APIs that **publish** all publicly accessible members and **subscribe** to types corresponding to methods. This provides functionality akin to an object-relational mapping (ORM) tool.
- Debugging tools that reconstruct causal history for the interactions between the NFV Manager and individual VNFs. Among other things this can be used to debug cases in

which translating from declarative configurations to imperative commands used by current VNFs results in errors.

7 Design Principles

Lean NFV embraces two important goals: making it easier to deploy NFV at any given point in time while allowing for rapid and ongoing innovation in the future. This requires that Lean NFV start with a set of minimal changes to traditional approaches to NFV while laying the foundation for deeper and ongoing innovation. Lean NFV does so through three design principles:

1. Open architecture: An open architecture, in which independently developed components interact using a common, standard API, enables the industry to innovate with different approaches to issues such as configuration and isolation (and micro-controllers in general).

2. Clean separation between VNFs and all other components: Such a clean separation enables vendors to write VNFs without knowing in advance about the inner workings of the NFV manager, the VIM on which it will be deployed, the other VNFs with which the VNF coexists, or the NFV service chain within which it operates. With this level of separation, these other components can be changed without having to rewrite VNFs. Ideally a network function will be instantiated many times in its life over different service chains of limited lifespan.

3. Minimal constraints on VNF implementations and micro-controllers: Lean NFV supports flexibility in all NFV components by not stipulating how VNFs and micro-controllers are implemented. This can be done by specifying only how VNFs and micro-controllers integrate into the overall system, not whether they are realized in VMs or containers, or whether they are monolithic or highly disaggregated, or which functions are handled by EMSs and which by the general NFV manager.

Barriers to NFV deployment such as needing to adopt a VNF-specific VIM (or an NFV-specific virtualization orchestrator) and impediments to innovation (such as the lack of a common API among components) inhibit the virtualization of networks. Lean NFV is an attempt to avoid these challenges by focusing on a single universal mechanism for coordination (the LRS API) and having a clean separation of concerns between the various components. As a result, one can adopt Lean NFV with minimal changes to existing VNFs (only modified to use the KVS for their control plane), while allowing a spectrum of future implementations.

For instance, Lean NFV implementations could include VNFs running under an existing VIM (with an NFV manager providing the runtime functionality), in container-based implementations (using varying degrees of built-in management functionality), as highly disaggregated VNFs rather than large monolithic ones, or as stateless implementations where all VNF state is kept in the KVS and accessed via the LRS API. This flexibility would, in turn, bring additional benefits as it would allow streamlined NFV infrastructures, fully optimized for the specific context (e.g., Edge, customer premises, etc.)

The goal of Lean NFV is to define an open architecture where many of these futures can be realized through incremental evolution, starting with current VNFs and VIMs. Hence, this Standard focuses on a minimal set of changes that VNF writers can embrace that offer immediate benefits while simultaneously laying the foundation for many of the longer-term changes that will ultimately be necessary. The immediate benefit is the option to leverage a general NFV manager for common runtime management tasks.

8 Lean NFV in the LSO Architecture

A key objective in Lean NFV is to co-exist with other implementations of LSO utilizing different architectures by integrating through the appropriate LSO APIs and preserving their well-chosen abstractions. Lean NFV can be used for a range of use cases, such as for services offered at the enterprise edge, as an NFV platform, and for 5G-related services, primarily at the edge. Lean NFV supports orchestration and management functions via the Presto API (northbound of the LRS to allow a higher-level orchestrator to manage physical or virtual devices as abstractions via the LRS) and the Adagio API (as the common interface to the LRS from the network functions and micro-controllers); see Figure 3 – Lean NFV in LSO.

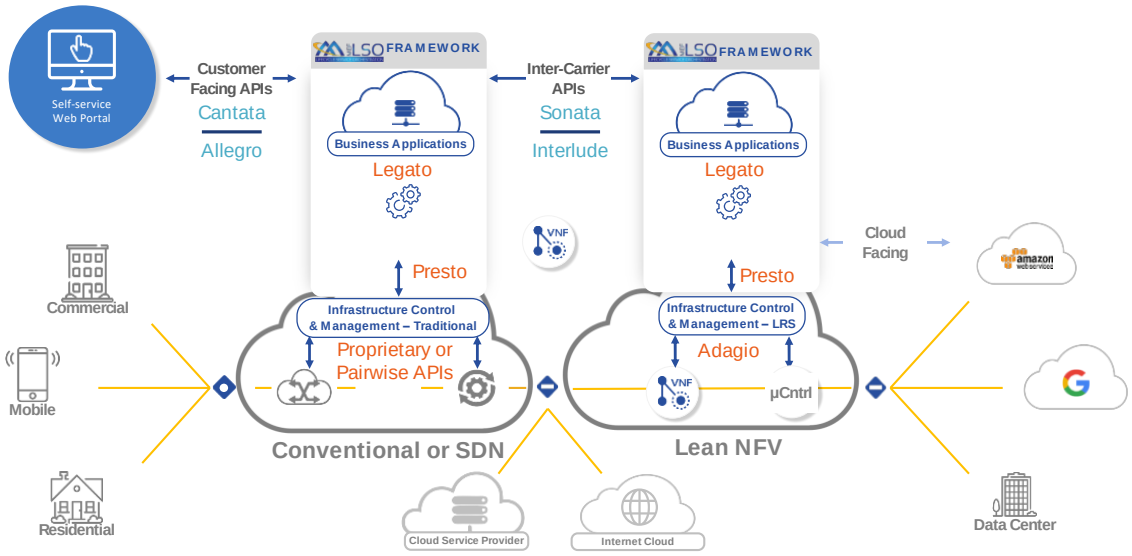

Figure 3 – Lean NFV in LSO

The most important points to note about Lean NFV in LSO are these:

- Having Lean NFV in one domain does not constrain any other domain.
- The LRS serves as an implementation of Infrastructure Control and Management (ICM).
- The Service Orchestration Function (SOF) interacts with the LRS using the LSO Presto API, as it does with other instances of ICM.
- The LRS interacts with the Lean NFV elements (see Figure 2 – Lean NFV Sample Architecture) using instances of the Adagio API. This contrasts with the traditional and SDN approaches on the left in which traditional controllers, SDN controllers, and network elements interact using a variety of APIs and protocols, including RIP, OSPF, BGP, MPLS LDP, STP, OpenFlow, SNMP, Netconf, and others. Significantly, these are usually pairwise APIs and protocols even if they are standard and thus not proprietary. Note that when necessary these existing interactions can be encapsulated and transported across a Lean NFV instance of ICM.

9 Business Drivers

The adoption of Lean NFV by the target user community is based on key business drivers of:

- Revenue expansion
- Enhanced customer experience
- Cost competitiveness

These are discussed in the following subsections

9.1 Revenue Expansion

Revenue expansion is generally achieved in two ways, through the timely addition of new services and the fast enhancement of existing services. Lean NFV supports revenue expansion in both ways. By making the NFs and micro-controllers independent of each other and enabling an open ecosystem, service providers can launch new services more quickly. While the enterprise edge use case is described in Section 10 (Use Cases), other services to which Lean NFV applies include fixed wireless and security services.

To enhance existing services, Lean NFV provides stand-alone management of the resources and services allocated to it. Lean NFV is intended to integrate easily through the MEF APIs with other orchestrators or an OSS/BSS (Operational Support System/Business Support System), without disrupting existing implementations using other technologies. This plug-in capability of functions implemented as micro-controllers therefore means that users can protect their existing investment in established services, while using Lean NFV to provide expanded capabilities.

Finally, Lean NFV provides a strong transition to emerging technologies and environments, such as cloud-native computing and containers instead of virtual machines because any component can be implemented or updated independently of the other components.

9.2 Enhanced Customer Experience

At the core of a successful service is an enhanced customer experience. Lean NFV enables this in two important ways. First, Lean NFV enhances the ease and speed of introducing virtual network functions and micro-controllers by an operator, which enables broader customer choice of network services and service features and promotes the creation of novel network functions that can be written once and deployed anywhere for greater service agility. Indeed, the adoption of Lean NFV could lead to a flourishing new open market for independently written network functions. For example, a service provider can give its customers a range of SD-WAN and security choices in an enterprise edge deployment, as they are all integrated quickly in the same way and are not dependent on other components. Upon customer request for a new service or feature, a new NF can be added in a short amount of time to the existing NFV implementation. Second, modular management (modular because there are no interdependencies between the NFs or the micro-controllers beyond their abstracted characteristics) makes automation much easier.

9.3 Cost Competitiveness

Lean NFV can improve the cost structure of an NFV implementation in three ways. First, by detaching the software from the hardware decisions (a basic feature of NFV), service providers can take advantage of a broader range of white-box hardware. Second, the simple abstractions in the KVS, and the KVS itself, can translate to less overhead, be it fewer cores, fewer or smaller servers, or simpler management. All other factors being equal, this reduction in infrastructure resources can translate to direct savings for operators and their customers. Finally, the standard abstractions and open framework translate to substantially lower customization services and support costs.

9.4 User Motivation

Ultimately user motivation can be summarized in the desire to accelerate the deployment of new capabilities (time to market) and reduce operating costs (total cost of ownership). These objectives are achieved by:

- Hosting new services
- Enabling an all-software networking (NFV) solution
- Seamlessly combining capabilities from different providers of network functions by creating service chains utilizing VNFs and CNFs from a variety of sources
- Leveraging standard models representing NFV components and open APIs
- Automating configuration, operation, and change management functionality
- Managing networking across their public cloud environments jointly with their on-premises installations.

The following benefits can be achieved with a Lean NFV implementation:

- Distributed: operate across multiple environments, including cloud native, private cloud, PoPs, and branch
- Flexible: use network functions from any source – VNFs, CNFs, open source, and custom
- Policy-driven: apply simple high-level service definitions across environments
- Elastic: via the modularity that provides optimal resource utilization
- Visible: gain data plane visibility to enable direct measurement and reporting of key metrics.

10 Use Cases

Lean NFV can be deployed beneficially wherever NFV makes sense, as these use cases show.

10.1 Use Case – Secure Access Service Edge (SASE)

An enterprise or a service provider adopts Secure Access Service Edge (SASE) to provide both SD-WAN-like flexibility and efficiency in the underlay connectivity and the best forms of enterprise security in an environment where the attack surface is very large. Security functions could be located in a variety of places, from customer premises and other remote sites to corporate data centers, metropolitan edge locations, and the Cloud. Orchestrating these functions in disparate locations, provisioning them, and chaining them across an expansive SASE domain suits the use of Lean NFV well. The VNFs and micro-controllers that access the LRS, and the LRS itself, are not bound to any particular location, especially the same location; they only need to be reachable via the usual means, which the infrastructure provides.

The figure below shows security functions in a SASE environment residing in the CPE, at the network edge, and in the public cloud. These can be orchestrated through an LRS at any location.

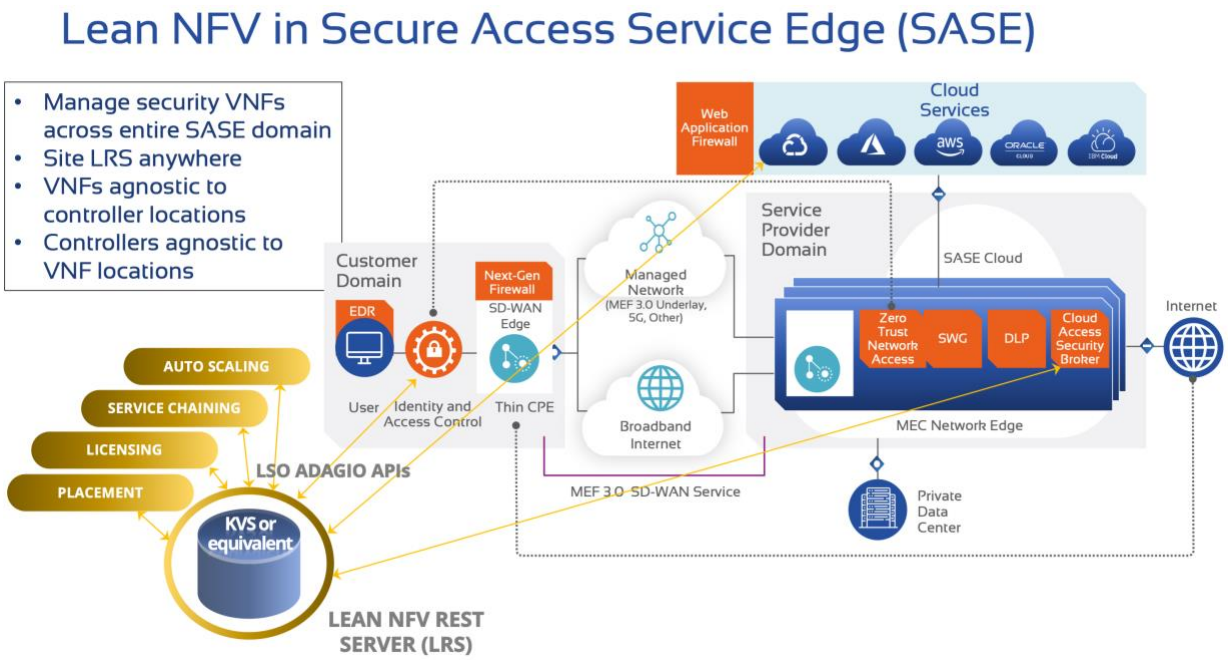

Figure 4 – SASE Use Case for Lean NFV

10.2 Use Case – Enterprise Edge

A service provider wants to enable a new service – some enterprise edge service, say – whereby enterprise customers can outsource networking functions to the service provider. The service provider will host their services in their DC, a public cloud, or white-box hardware on the customer premises (servers, uCPEs). With Lean NFV, the LRS – and typically the micro-controllers – run in the cloud or DC and the VNFs can be located in the cloud or DC or on the customer premises, to realize on-demand services such as SD-WAN, virtual firewalls (vFW) and other security

components. This arrangement permits the service provider to easily onboard VNFs and efficiently locate or move them around as needed in a distributed cloud computing environment. See the figure below for a graphical illustration of this scenario.

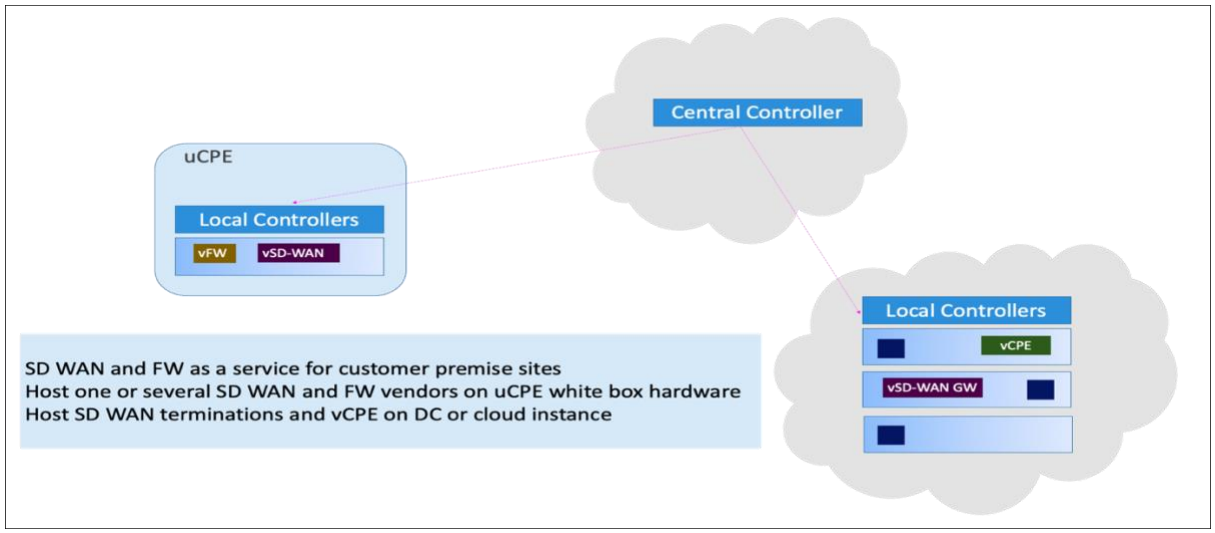

Figure 5 – Enterprise Edge Use Case

10.3 Use Case – NFV Platform

A SaaS provider wants to evolve their platform to work across multiple environments, such as public cloud and private data centers. The SaaS provider would also like to leverage commercial virtualized network functions such as firewalls, load balancers, routing, switching, and SD-WAN termination. Lean NFV can be deployed at the edge of the SaaS instance, whether that means hosted in the public cloud or in a data center, and can host all of the aforementioned functions. The SaaS provider can therefore drive consistent policy, make real-time changes, and obtain consistent visibility across environments. Lastly, an additional benefit is the ability to host customer-proprietary functionality on the platform as VNFs or CNFs, thereby offloading operational functionality such as high availability, monitoring, etc. The figure below illustrates this example.

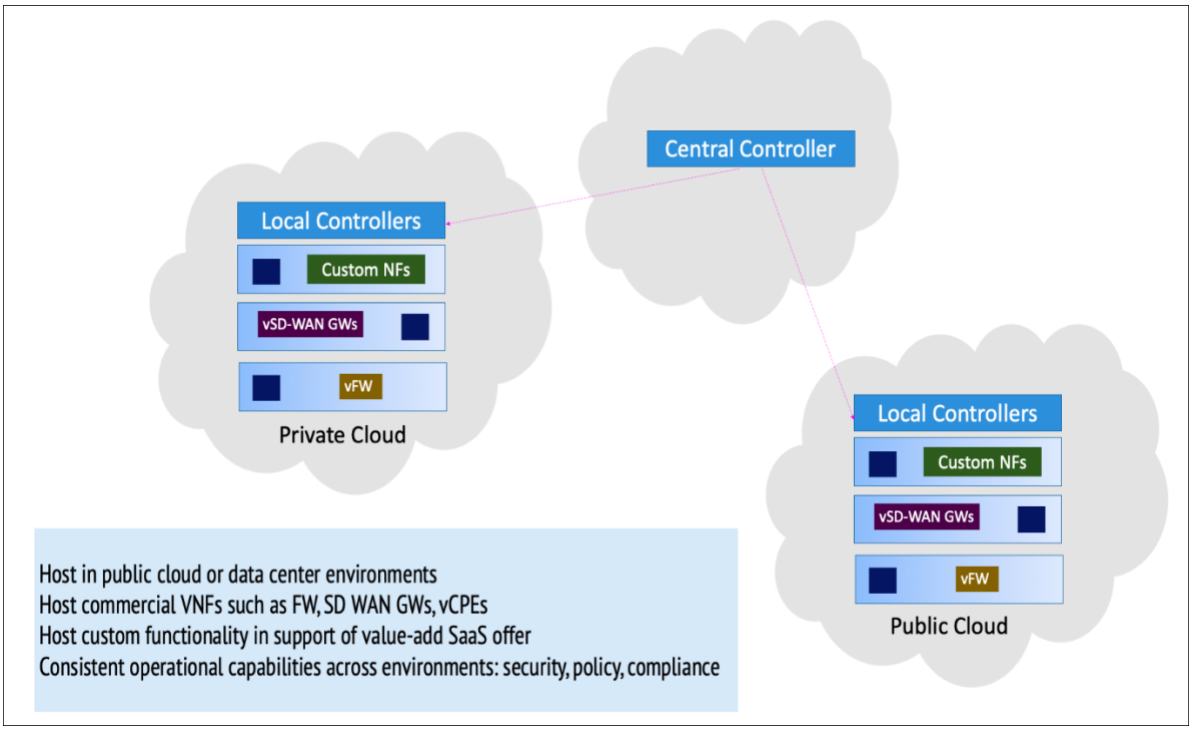

Figure 6 – NFV Platform Use Case

11 References

[1] MEF, *Simplifying NFV Operations in the WAN with LSO and Lean* NFV – White Paper, June 2020

[2] ETSI *Network Functions Virtualisation* – Introductory White Paper, October 2012

[3] MEF 55.1 Lifecycle Service Orchestration (LSO): *Reference Architecture and Framework*, February 2021

[4] MEF 70.1 *SD-WAN Service Attributes and Service Framework*, August 2020

[5] Linux Foundation: *Kubernetes API Overview*, API Version 1.22.0, August 8, 2021

[6] ETSI Specification: ETSI GS NFV-IFA 040 v4.1.1, Network Functions Virtualisation (NFV) Release 4; *Management and Orchestration; Requirements for service interfaces and object model for OS container management and orchestration specification*, November 2020

[7] SDxCentral NFV Definitions: *What is the Virtualized Infrastructure Manager (VIM)?* Definition, March 25, 2016

[8] Internet Engineering Task Force RFC 2119, *Key words for use in RFCs to Indicate Requirement Levels*, March 1997

[9] Internet Engineering Task Force RFC 8174, *Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words*, May 2017

[10] SDxCentral NFV Definitions: *What is NFV MANO?* Definition, October 9, 2014

[11] SDxCentral NFV Definitions: *What is a VNF Manager (VNFM)?* Definition, March 25, 2016

[12] SDxCentral NFV Definitions: *What is an NFV Orchestrator (NFVO)?* Definition, March 21, 2016

[13] SDxCentral NFV Definitions: *What is NFV Infrastructure (NFVI)?* Definition, April 27, 2016