Journal of Internet Services and Applications, 2026, 17:1, doi: 10.5753/jisa.2026.5499

Vines: A CloudStack Platform for the Orchestration and Holistic Management of Virtualized Network Functions and Services

José Flauzino [**Federal University of Paraná** | ***jwvflauzino@inf.ufpr.br***]
Vinicius Fülber-Garcia [**Federal University of Paraná** | ***vinicius@inf.ufpr.br***]
Alexandre Huff [**Federal Technological University of Paraná** | ***alexandrehuff@utfpr.edu.br***]
Giovanni Venâncio [**Federal University of Paraná** | ***giovanni@inf.ufpr.br***]
Elias P. Duarte Jr. [**Federal University of Paraná** | ***elias@inf.ufpr.br***]

Department of Informatics, Federal University of Paraná, R. Evaristo F. Ferreira da Costa, 383-391 - Jardim das Américas, Curitiba - PR, 81530-090, Brazil.

Received: 31 January 2025 • **Accepted:** 19 November 2025 • **Published:** 24 February 2026

Abstract
Network Function Virtualization (NFV) is changing the way networks are built and maintained by replacing traditional middleboxes with software that runs on commodity hardware. This paradigm shift not only increases flexibility but can also reduce the cost of building and maintaining the network. The ETSI NFV-MANO reference model is currently widely adopted as the *de facto* NFV standard, with a large number of compliant platforms currently available. In this work, we propose Vines (*Vines Is an NFV-MANO Extensible Solution*), a CloudStack-based platform that supports NFV technology and is compliant with the ETSI NFV specifications. Vines includes an NFV Orchestrator (NFVO) that enables the creation of complex network services composed of multiple individual functions. Management & orchestration are holistic, in the sense that they provide a comprehensive set of functionalities for heterogeneous network functions and services. Moreover, nearly all NFV developments have been done on the OpenStack cloud platform. Vines opens up the possibility of using NFV solutions on Apache CloudStack, one of the most widely used cloud platforms worldwide. Experimental results are presented showing that Vines has the same performance level as the widely used OpenStack/Tacker.

Keywords: Network Functions Virtualization, Network Management, Network Services, Cloud Computing.

1 Introduction

Network Functions Virtualization (NFV) allows the replacement of hardware-based middleboxes by virtualized software that runs on off-the-shelf servers (Mijumbi *et al.*, 2015). NFV enables the implementation of Network Functions (NF) to be decoupled from physical network equipment. Thus, NFs such as firewalls, load balancers, intrusion detectors, among others, can be implemented in software that is executed as VNFs (Virtualized Network Functions). Furthermore, NFV also allows the composition of SFCs (Service Function Chains), which are complex network services consisting of multiple VNFs (Fulber-Garcia *et al.*, 2020). NFV has also been employed to run arbitrary computing services within the network (Venâncio *et al.*, 2022; Turchetti and Duarte, 2015; Turchetti and Duarte Jr, 2017). The NFV paradigm has the potential to increase flexibility and reduce both the capital (CAPEX) and operational (OPEX) costs to build and maintain the network (Martins *et al.*, 2014; Yousaf *et al.*, 2017; Venâncio *et al.*, 2019).

The European Telecommunications Standards Institute (ETSI) has proposed the NFV-MANO (NFV - MANagement and Orchestration) architecture as a standard for NFV technologies. The architecture consists of a set of specifications related to the deployment and lifecycle management of virtual network functions and services (ETSI, 2014). NFV-MANO (or simply MANO) defines several functional blocks. The Virtualized Infrastructure Manager (VIM) is in charge of managing and orchestrating the physical and virtual infrastructure resources. The VNF Manager (VNFM) is mainly responsible for managing the lifecycle of VNFs, which involves actions such as deploying, updating, and removing VNFs. The NFV Orchestrator (NFVO) is mainly in charge of managing the lifecycle of network services (Huff *et al.*, 2020). In addition, there is the VNF functional block; and each VNF has an associated EMS (Element Management System), which is responsible for traditional FCAPS (Fault, Configuration, Accounting, Performance, and Security) management functionality.

Solutions that implement the NFV-MANO architecture are usually based on cloud computing platforms. Cloud computing is a key NFV enabler that simplifies the orchestration and management of virtualized resources (which includes virtual compute, storage, and networking) and provides the means to automate management operations (Chiosi *et al.*, 2012). In particular, cloud computing platforms are used as the VIM in an NFV environment, while the VNFM and NFVO blocks are typically developed as modules adjacent to the cloud platform.

The MANO architecture has been widely adopted: multiple compliant platforms are currently available. An NFV platform is a system that implements the functional blocks of NFV-MANO, providing a higher-level abstraction layer for the development of NFV solutions, as well as their management and orchestration. To the best of our knowledge, all current NFV platforms are based on a single cloud platform: OpenStack (OpenStack, 2025). Examples include OSM (Open Source MANO) (ETSI, 2025), Open Baton (OpenBaton, 2025), in addition to Tacker (Tacker, 2025), which is an OpenStack project. Curiously, Apache CloudStack (ASF, 2025a), despite being one of the most adopted open source cloud platforms worldwide (Flexera, 2022), has been barely explored in the context of NFV, and in particular the NFV-MANO architecture.

In this work, we fill this gap by presenting Vines (Vines Is an NFV-MANO Extensible Solution) an NFV platform for Apache CloudStack. CloudStack is adopted by a very large number of relevant companies and organizations worldwide (ASF, 2025b). With Vines, NFV users/operators have a choice of cloud platforms to employ. Compared to OpenStack, CloudStack tends to be simpler to manage. An OpenStack environment requires the deployment of a myriad of individual components, such as the Tacker NFV project, the Nova project to provide compute instances, the Neutron network as a service project, and many others. On the other hand, CloudStack is based on pluggable services, which are executed integrated with the cloud.

Vines addresses limitations of NF management, compared to how it is done on other NFV platforms. One can fully comprehend this point only by grasping the subtle differences between NF and VNF. The core NFV element is the NF, which is itself decoupled from the way it is implemented, which could be either in hardware or software, including as a VNF. Consider as an example an IDS (Intrusion Detection System) implemented as a VNF. In order to fill the needs related to the management and orchestration of NFs running in virtualized environments, it is necessary to extend traditional management models (Mijumbi *et al.*, 2016). Thus, considering the IDS example, one problem is managing the lifecycle of the VNF that implements the IDS (deploying, removing, scaling in and out, *etc.*) while another problem comprises managing the IDS itself (reconfiguring the intrusion detection rules, reinitializing the detector, *etc.*). The major problem we see in other NFV platforms (that Vines solves) is that other platforms provide facilities for the management of the VNF lifecycle, but require manual management of individual NFs.

The management of an NF includes operations such as installing and configuring the network software on the virtual host (*i.e.*, virtual machine or container), on which the NF is running. Furthermore, there are other tasks such as starting, stopping, and restarting the NF. On the other hand, the VNF lifecycle management operations, as mentioned earlier, include deploying, updating, etc. Note that there are lifecycle operations that may involve one or more management operations related to the NF. For example, the complete deployment of a VNF instance may require the installation, configuration, and initialization of the NF to be executed. All current NFV platforms present limitations in this aspect. One common limitation is the provision of a reduced set of NF management features. Most NFV platforms only enable the automatic execution of a script during the deployment of a VNF. This script may contain instructions for the installation, configuration, and initialization of the NF. However, there is no support for any other NF management operations after the deployment, which must be done manually. One of the reasons for that limitation is the fact that current NFV platforms have poor support for VNF Execution Platforms (VNF-ExPs) (Garcia *et al.*, 2019b), such as ClickOS (Martins *et al.*, 2014), Click-on-OSv (da Cruz Marcuzzo *et al.*, 2017), COVEN (Garcia *et al.*, 2019a), among others. Note that while NFV platforms are systems that provide/support the whole NFV environment, VNF-ExPs can be seen as specialized virtual hosts for running NFs. Thus, while NFV platforms are capable of executing VNFs instantiated in this way, they are not able to interact with the VNF-ExPs interfaces that enable the management of the NFs themselves.

In this work, we present the design, implementation, and evaluation of Vines, an open-source NFV solution compliant with NFV-MANO and natively available for CloudStack. We propose an architecture that explores the NFV-MANO framework in-depth. In this sense, in addition to a VNFM and an NFVO (both featured by other NFV platforms), Vines also has a VNF-ExP called Leaf. Vines also includes a comprehensive and effective EMS, in addition to supporting VNF Packages (VNFPs). The EMS combined with the VNFM allows what we call "holistic VNF management", as the EMS makes it possible to execute all the required NF management operations transparently and independently of VNF-ExPs on which they are running (including, but not limited to, Leaf). In addition to the VNF lifecycle management operations that are also available in other NFV platforms (such as VNF deployment, updating, removal, autoscaling, and self-healing), Vines also supports NF management operations, such as those for installing, configuring, starting, and stopping the network software itself, without requiring the manual procedures of other NFV platforms.

Furthermore, Vines also includes an NFVO for the orchestration and management of SFCs. As mentioned above, an SFC consists of a chained composition of multiple VNFs. Traffic is steered through an SFC, being forwarded across the VNFs in a predetermined order. The Internet Engineering Task Force (IETF) proposed a reference architecture for SFCs. That architecture mainly comprises traffic classification and forwarding elements, allowing the orchestration and deployment of fully functional virtual services. The Vines NFVO is capable of composing and orchestrating SFCs over CloudStack's native virtual networks.

Vines was implemented and is publicly available[1]. Experimental results are presented, which involved the execution of multiple different VNF and SFC operations. Comparisons prove that Vines is competitive, presenting the same performance level as the widely adopted OpenStack/Tacker platform.

The main contributions of this work are the following:

- We present Vines, which is to the best of our knowledge the first CloudStack NFV platform fully compliant with the ETSI NFV-MANO specifications;
- Vines allows both VNF and NF management, through a comprehensive VNFM/EMS designed to enable holistic VNF management;
- Vines also includes an orchestrator that allows the composition of complex SFCs consisting of multiple VNFs;

[1] https://www.inf.ufpr.br/jwvflauzino/vines/

- The architecture, design, and implementation of the NFV platform are described, including functionalities and challenges;
- An empirical evaluation is presented, including comparisons with OpenStack Tacker.

The rest of this work is organized as follows. Section 2 presents an overview of related work. Section 3 details Vines, its architecture, and discusses design decisions. Section 4 describes the implementation, and Section 5 presents experimental results. Finally, Section 6 concludes the paper and presents future research directions.

2 Related Work

In this section, we present an overview of related work, including some of the most relevant open-source NFV platforms available. Tacker (Tacker, 2025) is an NFV module designed for the OpenStack cloud computing platform. OpenStack is thus the corresponding VIM of this NFV platform. In practice, Tacker runs with several other OpenStack modules (*e.g.*, Keystone, Neutron, Nova, etc.) as well as third-party modules, such as Open vSwitch, for instance. Tacker includes both the NFVO and VNFM functional blocks, and thus supports the VNF lifecycle management, including the deployment, updating, removal, monitoring, and scaling of VNFs, and also of virtual network services (implemented as SFCs). In terms of NF management, Tacker neither has an EMS nor supports third-party EMSes. As a result, only some primitive NF management tasks that can be executed through the cloud-init (Canonical, 2025a) application, which allows network operators to run shell scripts to configure an NF during its deployment. Note that in this section we do not consider NFV simulation and emulation tools (Son *et al.*, 2019; Tavares *et al.*, 2018).

Open Baton (OpenBaton, 2025) is an extensible NFV-MANO platform that also includes an implementation of an NFVO and a generic VNFM. Open Baton has mechanisms that allow on-demand implementation of adapters to support external VNFMs. Although OpenStack is the sole cloud platform supported as VIM, OpenBaton allows users to implement drivers to support other VIMs (no CloudStack driver is currently available). Open Baton supports the composition of network services (implemented as SFCs) and monitoring based on Service Level Agreements (SLAs), which includes scaling and recovery operations. In addition, Open Baton provides a generic EMS that runs within each VNF (acting as a Management Agent - MA) that allows the execution of lifecycle scripts (which must be included in the corresponding VNF Package) during VNF deployment. Open Baton also provides a Juju VNFM Adapter in order to allow users to deploy Juju Charms as VNFs.

Hosted by the ETSI NFV working group, Open Source MANO (OSM) (ETSI, 2025) delivers an NFV-MANO stack that implements three NFV-MANO functional blocks: NFVO, VNFM, and VIM. Thus, OSM includes functionality such as VNF lifecycle management, network service orchestration (SFCs), and infrastructure resource orchestration. This NFV platform employs OpenStack as the primary VIM, but it also supports other VIMs such as OpenVIM, Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform, VMware vCloud Director, and Kubernetes. OSM also provides templates for lifecycle operations that are performed internally to VNFs (called ‘Day 1” and “Day 2”). However, the execution of these templates depends on the availability of particular technologies, such as SSH (Secure Shell) and proxy charms (Canonical, 2025b).

Cloudify (Cloudify, 2025) is an open-source, multi-cloud orchestration platform, founded by the GigaSpaces company. One of the primary focuses of Cloudify is on the lifecycle automation of cloud-based services. Presenting multiple approaches to provide Environment as a Service (EaaS), Cloudify also supports NFV. In the context of NFV-MANO, Cloudify implements a generic VNFM and an NFVO for lifecycle service management. For the infrastructure resource orchestration, Cloudify relies on OpenStack.

T-NOVA (Kourtis *et al.*, 2017) provides a MANO stack to enable the management of NFV-based services, including support for multiple phases of the service lifecycle, namely resource discovery, service mapping, service deployment, and monitoring. The system was also developed as part of the EC FP7 T-NOVA project, and released as an open-source code. The MANO stack offered by T-NOVA was built around the TeNOR orchestrator, which was developed as part of the project. TeNOR includes modules such as NFVO, VNFM, and NFV repositories. T-NOVA adopts OpenStack as VIM and employs OpenDaylight to orchestrate the network infrastructure.

ONAP (Open Network Automation Platform) (ONAP, 2025) is a platform that provides features for the management, orchestration, and automation of network and edge computing services by leveraging SDN and NFV technologies. ONAP is an open-source project hosted by the Linux Foundation. Its functional architecture is organized into two main parts, called Design-time and Run-time. While the Design-time domain provides features related to onboarding services and resources into ONAP, the Run-time environment provides a framework for the instantiation and configuration of services and resources. As part of the Run-time set of components, ONAP originally had an EMS driver to support EMSes from different vendors. However, the EMS driver has been deprecated since the ONAP Guilin Release (ONAP, 2024).

Anuket (Anuket, 2025) is a Linux Foundation open-source project that was formed by joining the OPNFV (Open Platform for NFV) and Cloud iNfrastructure Telco Taskforce (CNTT). OPNFV, in turn, focuses on building an NFV ecosystem by integrating components from upstream projects related to cloud computing and SDN, such as OpenStack, Kubernetes, Ceph Storage, OpenDaylight, OVN, DPDK, and others. To also encompass VNFM and NFVO capabilities, OPNFV employs Tacker and ONAP.

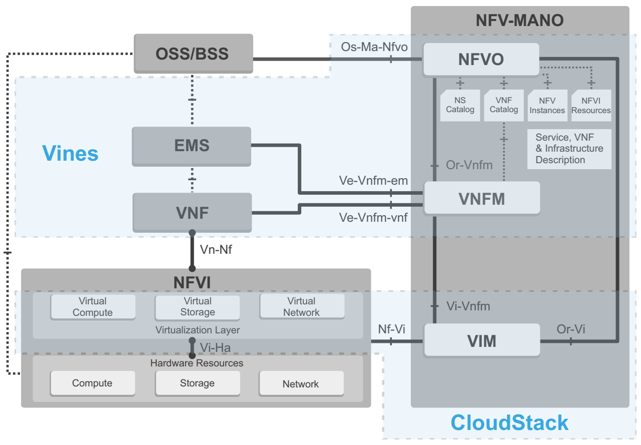
Table 1 summarizes the main features of the platforms mentioned in this section. It is important to note that all these platforms have several limitations regarding the management of the VNF component, especially the NF itself. Currently, Tacker, OSM, and ONAP do not provide a native EMS, and neither provide explicit support for third-party EMSes. Open Baton, in turn, offers a configurable EMS, but the inclusion of this EMS requires changes to the VNF execution platform, and may even require kernel code changes if the VNF execution platform is based on an unikernel operating system. Furthermore, it is evident that current NFV-MANO projects are highly dependent on OpenStack as the sole underlying VIM.

3 The CloudStack/Vines NFV Platform

This section presents the proposed Vines NFV Platform. Initially, Vines is presented in the context of the NFV-MANO reference architecture. Next, we delve into the Vines architecture, emphasizing its ability to facilitate holistic management of the VNF lifecycle. Finally, a description of the strategy to compose and manage multiple VNFs into SFCs concludes the section.

3.1 Vines and the NFV-MANO Reference Architecture

Figure 1. CloudStack/Vines within the NFV-MANO reference architecture.



The main purpose of Vines is to make NFV technology natively available in Apache CloudStack, being fully compliant with the ETSI NFV-MANO architecture. First of all, it is worth noting that CloudStack (without Vines) does support the manual instantiation and management of VMs that run applications that can be considered to be VNFs (ASF, 2025c). As that documents mentions that "instantiating a [VNF] appliance is actually simply provisioning a VM", the single difference is that "end-users need to create at least 3 networks (north, south, and management) and then attach the relevant networks to the VM."

Furthermore, it is also possible to build an external agent that works as an interface for components of the NFV-MANO architecture; for example, in (Bondan *et al*., 2019) a VNFM external to CloudStack manages network functions running on CloudStack. However, to the best of our knowledge, there is no other CloudStack solution that implements the full NFV-MANO reference architecture. In this way, Vines is the first complete, native CloudStack NFV-MANO solution.

Figure 1 presents CloudStack/Vines within the NFV-MANO reference architecture. The traditional CloudStack modules (the cloud platform itself) correspond to the VIM functional block of the NFV-MANO architecture, which manages the NFVI computational resources. Thus, Vines extends the capabilities of the CloudStack platform, by providing the functionalities of several NFV-MANO blocks, abstracting virtualization details, and providing a high-level interface for the management and orchestration of network functions and services. Vines itself consists of several MANO components, in particular the VNFM and the NFVO, but also features other functional blocks, such as the VNF and EMS blocks. In the next two subsections, the Vines approach to VNF management and SFC orchestration are described, respectively, together with a description of the Vines VNFM and NFVO, plus other components.

3.2 Vines: VNF Lifecycle Management

VNF instances are the elements that process data (network packets/traffic) and can be chained together to compose complex network services. VNF management is thus one of the key functionalities of an NFV platform. The VNFM (VNF Manager) Venâncio *et al*. (2021) is the NFV-MANO module primarily responsible for managing the lifecycle of VNFs, but to carry out its tasks in an effective manner other MANO modules are required, in particular the EMS (Element Management System).

The main challenge of VNF management is that VNFs are highly heterogeneous. First of all, the corresponding NFs implement a wide range of functionalities. Furthermore, each VNF (even those that have the same functionality) may have specific implementations that are also widely different from each other in several aspects. For instance, some can be instantiated through specialized VNF-ExPs while others are stand-alone. They can also have completely distinct communication interfaces. Moreover, the set of management operations available (or required) by each function can also vary.

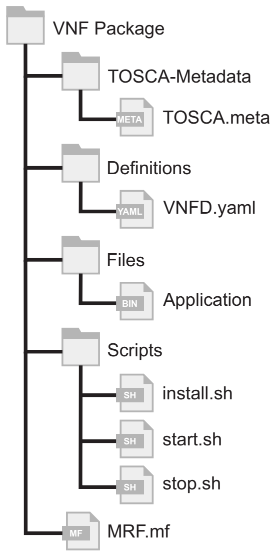
In order to deal with these issues, the NFV-MANO reference architecture defines the VNFP (VNF Package), which specifies for each VNF the management artifacts available. They can come as management scripts, software images that can be directly executed, or even the source code itself. Along with the Management Server, Vines maintains a VNF Catalog, composed of a set of VNFPs that contain the necessary artifacts for instantiating and managing a VNF. NFV-MANO uses TOSCA (Topology and Orchestration Specification for Cloud Applications) for the description of NFV elements and their interconnections (ETSI, 2014). Therefore, the standard VNFP model for Vines was defined based on the TOSCA YAML Cloud Service Archive (CSAR) (ETSI, 2018) specification. As shown in Figure 2, each VNFP contains a directory `TOSCA-Metadata` with the file `TOSCA.meta`, which includes definitions used as input to parse the rest of the contents of the CSAR file, such as subdirectories, VNFD (VNF Descriptor), management scripts, binary files, among others.

Also related to VNF management, the NFV-MANO reference architecture includes the EMS functional block. The EMS allows the VNFM to deal with all the heterogeneity. With an EMS, the VNFM is able to precisely determine how each VNF is managed. However, it is beyond the scope of

Table 1. Open-source NFV Platforms.

NFV Platform	NFVO	VNFM	Supported VIMs	Provides/Supports EMS	Provides/Supports VNF-ExP
OpenStack/Tacker	✓	✓	OpenStack.	-	-
OSM	✓	✓	OpenStack, OpenVIM, AWS, Microsoft Azure, Google Cloud Platform, VMware vCloud Director, and Kubernetes.	-	-
Open Baton	✓	✓	OpenStack.	✓	-
OPNFV/Anuket	✓	✓	OpenStack and Kubernetes.	-	-
ONAP	✓	✓	OpenStack and Kubernetes.	-	-
T-NOVA	✓	✓	OpenStack.	-	-
Cloudify	✓	✓	OpenStack.	-	-
Vines	✓	✓	CloudStack.	✓	✓

Figure 2. An example of the VNFP structure (CSAR) that Vines supports.



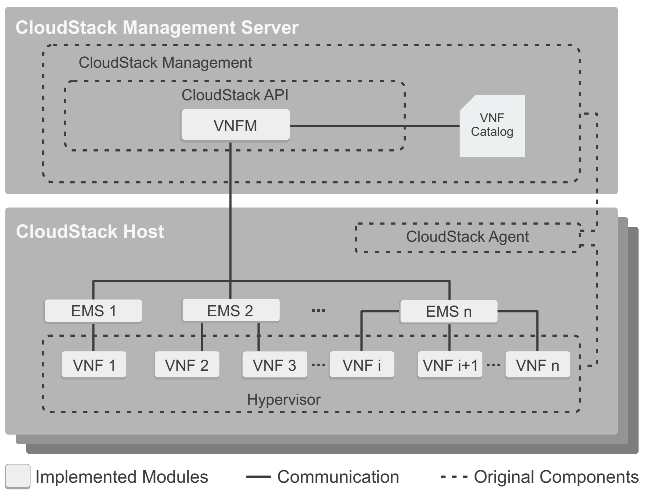
NFV-MANO to specify the internal structure of the EMS (as well as of the VNF module). A general EMS architecture has been recently proposed (Fulber-Garcia *et al*., 2023). But, in practice, it is left to each system to implement the model. Unfortunately, most NFV platforms either lack or have very primitive EMS implementations. Some VNF vendors also provide specific EMS solutions for their functions.

Most of the NFV platforms described earlier do not have a native EMS, including OpenStack/Tacker, OSM, and several others. Open Baton, on the other hand, does have a simple EMS that is executed within each VNF instance. That EMS simply mediates the communication between the VNF and the VNFM. However, this approach has several disadvantages: *(i)* either the VNF source code or the corresponding VNF-ExP code has to be changed to include the EMS. In the case of VNF-ExPs that are based on a unikernel/minimalist OS (like OSv or Click OS, for example) it is necessary to change the kernel itself; *(ii)* the communication between the VNFM and each specific VNF (*i.e.*, the EMS inside it) requires some specific communication protocol and its implementation, for example AMQP (Advance Message Queueing Protocol) and RabbitMQ, respectively, in the case of Open Baton.

On the other hand, the Vines EMS is external to the VNF. Our proposed architecture extends the NFV-MANO specification by defining the internal structure of the EMS to address FCAPS functionality, providing holistic VNF management. Thus, while the communication between the VNFM and the Vines EMS occurs in a standardized way, the communication between the EMS and the VNF can be done using different protocols and technologies. The Vines architecture, as well as the internal structure of the EMS and VNF blocks, are described below.

Figure 3 presents an overview of the Vines VNF management architecture. The figure shows a simplified CloudStack scenario, composed of a single management node and multiple compute nodes. The relationship between the VNFM and the other NFV-MANO blocks that make up the architecture is also shown. Vines is designed so that its core is part of the main CloudStack application, known as CloudStack Management, which operates on a type of node called Management Server. As a result, users (network operators) access the VNFM of Vines through the CloudStack API (Application Programming Interface). The figure also shows that compute nodes run a CloudStack Agent (which in some cases is the virtual machine hypervisor itself) and the interface between Management and Agent.

Figure 3. The Vines VNF management architecture.



Vines implements a generic VNFM that has the basic functionalities for managing the lifecycle of VNFs. Note that all NFV platforms described in Section 2 have similar VNFMs. One of the main contributions of Vines is actually the fact that it presents enhanced management capabilities, which heavily rely on a comprehensive EMS. The Vines VNFM uses the interface between two native applications: CloudStack Management and CloudStack Agent. Actually, this interface can be thought of as between VNFM and VIM, since it is through the CloudStack Agent that the Vines VNFM can request the hypervisor to manage the NFVI resources. Examples of such requests are those to instantiate, scale, or remove a VM that runs an NF. However, Vines heavily relies on the EMS component to be able to achieve full VNF management. The EMS component is described next. Each VNF has a corresponding EMS, while an EMS can be responsible for one or more VNFs.

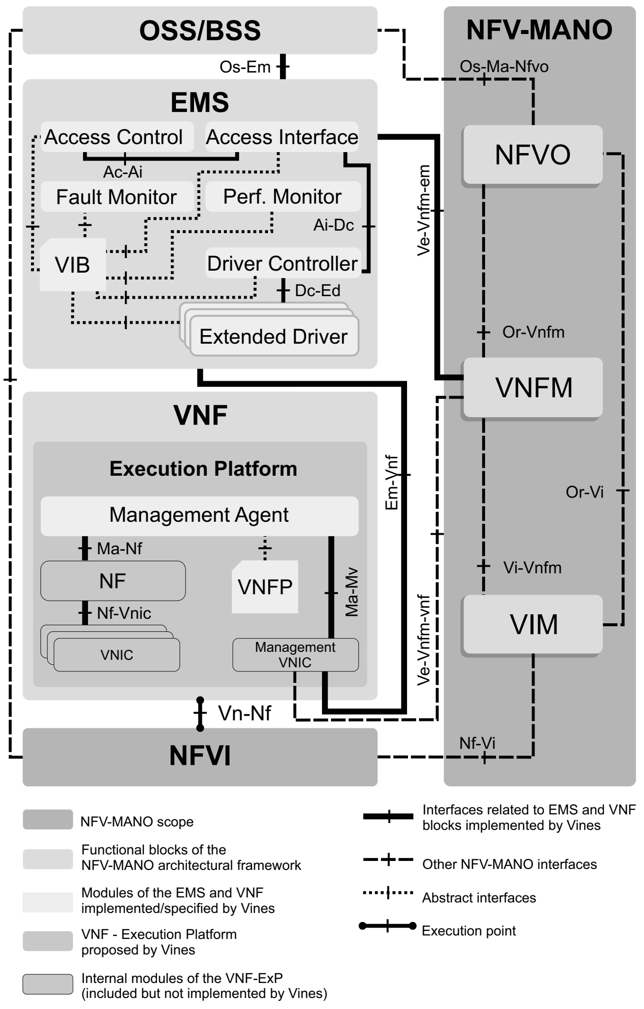
The proposed EMS architecture is shown in Figure 4 with the corresponding VNF and the remaining NFV-MANO functional blocks. The EMS was designed to be flexible enough so that it is capable of handling heterogeneous VNFs properly. The EMS communicates with different functional blocks of the NFV-MANO architecture. The communication with the VNFM is through a standard interface specified as the NFV-MANO reference point *Ve-Vnfm-em*. Despite the fact that the NFV-MANO documents mention that the EMS communicates with both VNF and OSS/BSS (Operations Support System / Business Support System), those interfaces are not specified in the ETSI documents. To fill this gap, Vines defines these reference points and corresponding interfaces.

We define the reference point between OSS/BSS and EMS which is called *Os-Em*, while *Em-Vnf* represents the reference point between EMS and VNF. The interfaces for these reference points are adapted from those previously specified by NFV-MANO for the *Ve-Vnfm-em* and *Ve-Vnfm-vnf* reference points (ETSI, 2020). Those interfaces encompass the entire set of operations available to the related blocks. The *Os-Em* reference point includes all the interfaces of the *Ve-Vnfm-em* reference point. The EMS receives requests from both the VNFM and the OSS/BSS. Since the OSS/BSS is an external entity, their requests must go through an access control process. That is not the case for VNFM requests. The *Em-Vnf* reference point, on the other hand, adopts the operations of the *Ve-Vnfm-vnf* reference point. The only exception is the Indicator VNF interface since the EMS does not subscribe to the VNF to receive notifications. Actually, the Vines EMS *has* a notification service, which actively monitors VNFs to obtain the required data and thus provide the required information.

Figure 4 shows the internal architecture of the EMS and VNF blocks and their interactions with other blocks of the NFV-MANO reference model. Next, we describe the EMS and then the VNF block in detail. The proposed internal modules of the EMS functional block are the following:

- **Access Interface:** exposes an access interface to the set of EMS operations, bringing together all the interfaces specified by the NFV-MANO reference point *Ve-Vnfm-em*, as well as the *Os-Em* (specified in this work);
- **Access Control:** authenticates the entities requesting VNF lifecycle operations and verify that they have the proper permissions to perform each operation;
- **VNF Information Base (VIB):** a database to store the information provided by the VNFM regarding the VNFs for which the EMS is responsible. The VIB can maintain both VNF configuration and monitoring information. Information in this database is used by the EMS to implement FCAPS management functionalities;
- **Fault Monitor:** monitors the state of the VNFs that are registered in the VIB and employs a predefined fault monitoring policy. Based on those predefined policies, this module sends failure notifications to the VNFM;
- **Performance Monitor:** collects data related to VNF resource usage. The VNFs must be registered in the VIB and have a predefined performance monitoring policy. This module notifies the VNFM in case the policy is violated;
- **Driver Controller:** forwards requests received by the Access Interface module to an Extended Driver (defined below) through which it communicates with the VNF, properly handling the VNF-ExP interface;
- **Extended Driver:** acts as a communication broker between EMS and VNF. It can perform functionalities such as parameter conversion, reassemble URIs, and execute different call methods (*e.g.*, HTTP, SSH, and Socket), among others. There must be an Extended Driver for each specific type of VNF-ExP interface.

Figure 4. The Vines EMS and VNF execution platform: architecture.



In the context of the architecture shown in Figure 4, the main blocks are the OSS/BSS, VNFM, EMS, and VNF. All communications between these blocks are performed through the interfaces defined for the *Ve-Vnfm-em* reference point (specified in the NFV-MANO architecture), and by the *Os-Em* and *Em-Vnf* reference points, which were defined earlier in this work. Through the set of interfaces provided by the Access Interface module, the OSS/BSS and VNFM blocks can request the EMS (respectively, through the *Os-Em* and *Ve-Vnfm-em* reference points) to execute each of the operations for which it is responsible. Security is also a concern for NFV (Krishnan *et al.*, 2020). Requests received by the Access Interface module are authenticated by the Access Control module and forwarded to the Driver Controller. The VNF-ExP specified in the request is used by the Driver Controller to forward the request to the corresponding Extended Driver. The Extended Driver is designed to handle heterogenous VNF-ExPs in a transparent way, dealing with each type of VNF-ExP. Therefore, it is one of the components that allow Vines to provide holistic VNF management.

The OSS/BSS and VNFM functional blocks can also use the operations provided by the Access Interface module to subscribe to the corresponding EMS to receive notifications of certain events from a specific VNF. However, in the case of OSS/BSS, the permission for this type of subscription is defined by each VNF and is specified as a monitoring policy that is evaluated by the EMS Access Control module. Two EMS modules – Performance Monitor and Fault Monitor – use the interfaces defined by the *Em-Vnf* reference point to collect data on resource usage and check that the VNF is fault-free, respectively. These two modules notify the VNFM and/or the OSS/BSS based on the current subscriptions. Therefore, they are the ones that enable both automatic scaling and automatic recovery of VNFs, which are key components for the dynamic allocation of resources for VNFs and SFCs (St-Onge *et al.*, 2023). However, note that both modules only perform notifications upon the occurrence of certain VNF conditions, they do not make any decisions.

Finally, an EMS stores data about each VNF instance in the VIB module. Thus, in Vines, each EMS has its own VIB, which can be accessed by all other internal modules. Multiple types of data are stored in the VIB, such as unique identifiers for each VNF, data about resource usage, fault monitoring policies, the IP address of the management network interface of each VNF, and the required VNF-ExP, among others.

Our proposed EMS enables the management not only of VNFs instantiated on VNF-ExPs but also those that run directly on a regular VM. For example, an EMS can manage a VNF via the SSH protocol (available on most VMs), or via the interface provided by a VNF-ExP (such as a RESTful API). Nevertheless, we argue that using a specialized environment such as a VNF-ExP can lead to more fine-grained, efficient, and secure management.

Actually, Vines provides a VNF-ExP - which is presented in Figure 4, and is internal to the VNF block. The architecture defines the minimum requirements to enable the instantiation and management of NFs in a standardized, simplified, and flexible manner. The internal modules of the VNF-ExP are described below:

- **Management Agent:** the module that provides an interface to several operations related to the management of the lifecycle of each VNF;
- **Network Function:** the network function software to be executed. This may consist of one or more VNF Components (VNFCs);
- **VNFP:** a directory containing descriptors, scripts, and other items related to the VNF (please refer to Figure 2).

The sequence diagram in Figure 5 summarizes the execution of VNF management operations. The diagram illustrates how Vines interacts with the VNF-ExP when managing the NF running on a VM. The network operator requests the execution of an operation to the Vines VNFM (*e.g.*, install, configure, start, or stop the network function). Next, the VNFM selects the corresponding EMS, and sends the operation to be executed. The EMS checks for permissions and forwards the request to the VNF-ExP, which executes and returns the outputs which eventually reach the network operator.

3.3 Vines: SFC Orchestration

Similar to the VNF Catalog, Vines also maintains along with the Management Server a repository called Network Service Catalog. This repository consists of descriptors such as the NSD (Network Service Descriptor) and the VNFFGD (VNF Forward Graph Descriptor), which specify the VNF chain that comprises each network service.

Vines also includes an NFVO to perform NFV orchestration, including the composition of network services. Like its VNFM, Vines' NFVO is designed to be part of the main CloudStack application (CloudStack Management). Therefore, its functionality can also be accessed by network operators through the native CloudStack interface.

The NFVO has interfaces to other NFV-MANO functional blocks that are fundamental to its orchestration activities. For instance, the NFVO has direct access to NFV repositories and the VNFM. The interface with the VIM (CloudStack itself) is established through the CloudStack Management and CloudStack Agent applications, allowing the NFVO to access the virtualized resources. Figure 6 presents an overview of the Vines service orchestration architecture, that extends Figure 3 used to illustrate the VNF management architecture. Figure 6 shows the relationship between the NFVO and the other Vines components.

CloudStack deploys client VMs in the so-called Guest networks. Each Guest network is isolated and the communication of a VM with other Guest networks or networks outside the cloud is routed through a Virtual Router (VR) that acts as the gateway. In CloudStack, a VR is implemented as a system VM, that is a VM specifically prepared to act as a virtual element in the cloud network (even having an internal agent). Vines instantiates a VNF in a Guest network with a system VM acting as a VR. Figure 7 illustrates instances of VNFs connected in a typical CloudStack virtual network environment.

Because they operate as local networks, when VNFs belonging to the same Guest network communicate with each other, the network's VR acts as a virtual switch (*i.e.*, a layer-2 device). In other words, any such communications do not

Figure 5. Execution of VNF management operations.

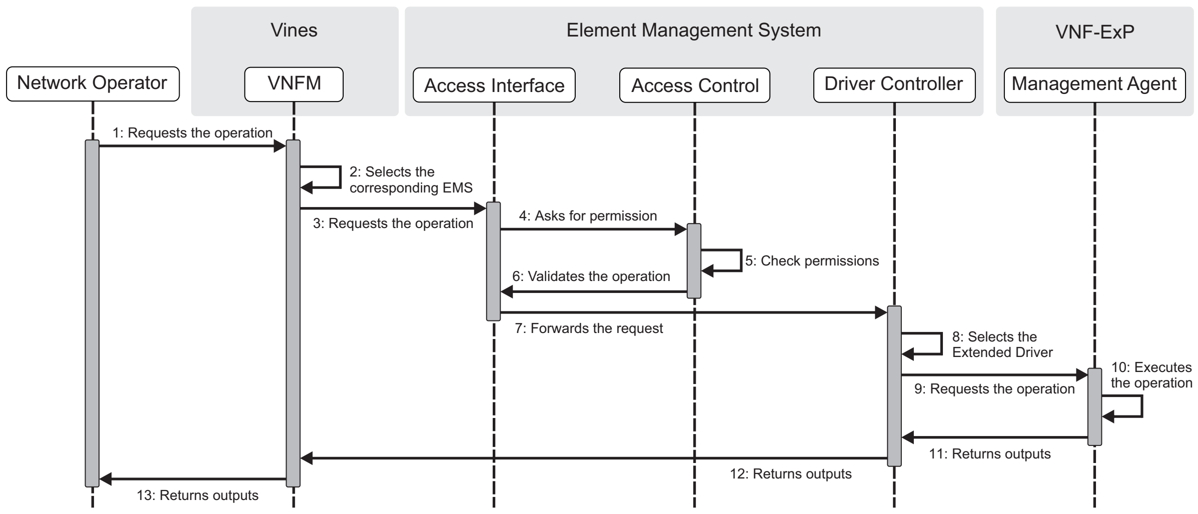


Figure 6. The CloudStack/Vines NFVO Architecture.

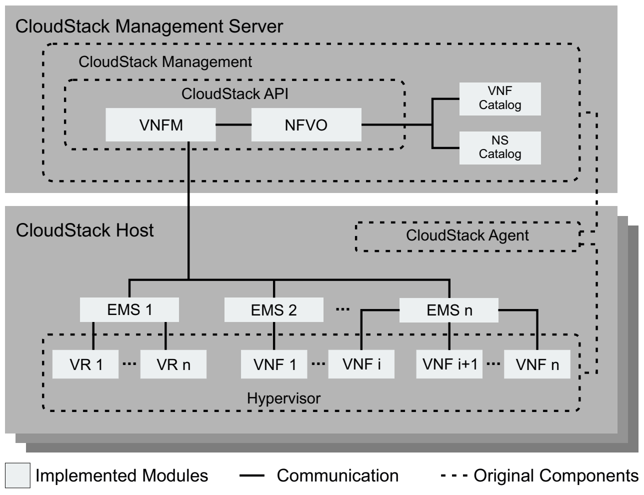
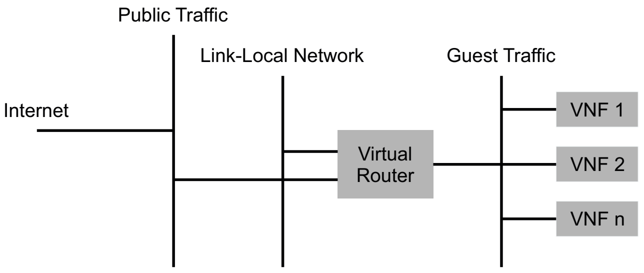


Figure 7. VNFs connected in a typical CloudStack virtual network environment.



involve the VR's tables (*i.e.*, layer-3 routes are not employed in this case).

Each VNIC (Virtual Network Interface Card) of a VNF instance has a private IP address in the respective Guest network to which is connected. To enable intra-network communication between VNFs, an IP address from the CloudStack public network is associated in the VR with the private IP address of each VNF. This association is made internally to the VR, regardless of the VNF's VNIC configuration. In this way, the VR allows a VNF to access other networks via SNAT (Source Network Address Translation). Such a network can be another Guest network or a network that is outside the cloud (*e.g.*, in the Internet).

Since CloudStack's native virtual networks are not based on SDN (Software Defined Network) technology, there is no SDN controller, and no compatibility with protocols such as OpenFlow [2]. Therefore, the main challenge in enabling the composition and management of virtualized network services on top of CloudStack's native networks is to devise an approach to properly orchestrate the network elements (*i.e.*, operate the control plane).

Vines adopts a strategy that allows greater centralization of the control plane of CloudStack's virtual networks. According to that strategy, the NFVO leverages the full management capabilities of the VNFM. The key point of this strategy is to treat each CloudStack VR as a VNF internal to the system – as opposed to a user VNF. Thus, CloudStack's system VM template is treated as a VNF-ExP by Vines. In this way, in addition to making orchestration easier, CloudStack's own network resources are treated in the same way as any VNF, also having an EMS responsible for their management. Figure 6 demonstrates that each EMS can be responsible for one or more VNFs and also for one or more VRs. Thus the Vines NFVO inherits all the characteristics of the Vines VNFs management architecture to support network orchestration. This approach gives the NFVO sufficient management capabilities to perform the necessary configurations of the network elements when composing virtualized network services.

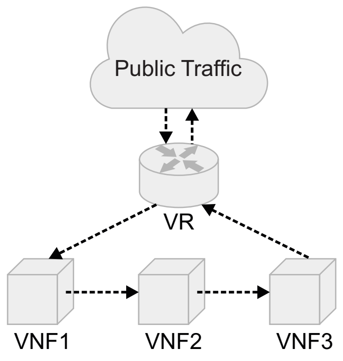
Vines creates a network service through the composition of multiple VNFs chained together in an SFC through which network traffic flows in a specific order. Given the features of CloudStack native virtual networks (mentioned earlier), the Vines NFVO composes SFCs configuring both the VNFs themselves and the VR that acts as a gateway in the Guest network to which the VNFs of the SFC are connected.

Vines forwards traffic to an SFC by routing the packets to an IP address of the CloudStack public cloud network. Each SFC must have an IP address of the CloudStack public traffic network. That address may even be a public Internet address, which is used by the SFC clients to access the network service.

[2]There are CloudStack plugins that make it possible to build SDN-based networks by interacting with external controllers, but that control plane is no longer CloudStack's.

Consider that each SFC consists of n VNFs (in Figure 8, $n = 3$). VNF 1 is the first VNF of the chain, while VNF 3 is the last. Specifically, when composing an SFC, the Vines NFVO assigns a public IP address to the last VNF and associates this address with the SFC. It then configures the VR so that traffic headed to the SFC's public IP (*i.e.*, directed to the last VNF) is routed to the first VNF in the SFC.

Figure 8. Traffic flows across a Vines SFC.



Furthermore, the NFVO also configures the first $n - 1$ VNFs of the SFC so that traffic passes through all the VNFs before reaching the last one. Since it is the "real" destination, when traffic arrives at the last VNF of the SFC, that VNF processes each packet and returns it to the corresponding VR.

Depending on the functionalities performed by the VNFs that comprise an SFC, there are cases where the flow does not reach the last VNF. For example, at a certain point in an SFC, there may be a firewall VNF that rejects or blocks passing traffic, preventing the flow from moving forward and reaching the last VNF. Situations like this are completely normal and, although they create behavior that diverges from what was expected at first (*i.e.*, that traffic would reach the last VNF), they do not affect the operation of SFCs in CloudStack/Vines.

4 System Implementation

This section presents the CloudStack/Vines implementation. All the elements required to provide VNF lifecycle management were implemented, including the support for the composition and management of SFC-based network services. CloudStack/Vines is publicly available as an open source software[3]. The core of Vines (VNFM and NFVO) was developed within CloudStack's source code. The generic EMS and the Leaf VNF-ExP were also implemented and are key elements of the holistic management architecture of Vines (presented in Section 3.2). Both are available as VM templates on the platform. All Vines functionalities are accessed through the CloudStack API.

CloudStack is composed of a core, which provides the basic functionalities for cloud orchestration, while other functionalities are provided through pluggable services, also called plugins, which are made available in a unified way. It is important to note that CloudStack plugins are natively part of the platform and can be activated or deactivated on demand, thus they cannot be added or removed in each deployment. Each new CloudStack release may bring enhancements to existing plugins or make new plugins available.

We implemented the Vines VNFM and NFVO as a single CloudStack NFV plugin that follows the CloudStack platform's pluggable services architecture. Thus, all the operations provided by the VNFM and NFVO of Vines are accessible by the users (network operators) through CloudStack's API, which is a REST (Representational State Transfer) interface (Fielding, 2000). As a result, all the functionalities included in the platform through the implementation of the NFV environment provided by Vines can be seen as new functionalities of CloudStack itself. We argue that CloudStack/Vines is significantly easier to use when compared to other NFV platforms, in particular because it is not necessary to handle multiple interdependent projects in a non-trivial way.

The following subsections describe the implementations of several key Vines functionalities. The first subsection is on the implementation of the NFV repository, next, the VNF lifecycle management, and finally, virtualized network service orchestration.

4.1 Implementation of the NFV Repository

The NFV-MANO specification defines several NFV repositories, such as the NFV Instances Repository, the VNF Catalog, and the NS Catalog. Vines provides several functions that allow the management of these repositories. Following the NFV-MANO specification, the NFVO of Vines is primarily responsible for the NFV repositories, but the VNFM also interacts with the NFVO to gain access or modify their contents.

In Vines, each VNF is represented by a VNFP (to recall, 'P' stands for Package) that holds all the VNF artifacts (as presented in Subsection 3.2). The set of all VNFPs of a Vines deployment comprises its VNF Catalog. It is critical to enable VNFPs to be easily uploaded to the platform and managed properly. In this regard, Vines provides a set of operations for managing the VNF Catalog. Table 2 presents these operations. Note that they cover the complete CRUD (*Create, Read, Update, and Delete*) functionality.

Table 2. VNF Catalog management operations.

Operation	Description
createVnfp	Retrieves a VNFP from a URL and adds it to the VNF Catalog
listVnfps	Lists information from one or all VNFPs in the VNF Catalog
updateVnfp	Updates the metadata or files of a VNFP contained in the VNF Catalog
deleteVnfp	Removes a given VNFP from the VNF Catalog

Each VNFP contained in the VNF Catalog of Vines can be used as a template for the creation of VNF instances through requests to the VNFM (functionalities that are described later in Subsection 4.2). Another relevant point is that the implemented solution supports the inclusion of VNFPs consisting of a compressed file in zip format or obtained from a Git repository. In any case, a VNFP must comply with the CSAR

[3]The Vines project page is available at: `https://www.inf.ufpr.br/jwvflauzino/vines`

specification. These Vines features aim at facilitating VNFP onboarding.

A critical element for the composition and orchestration of virtualized network services is the NS Catalog. Table 3 lists the operations provided by Vines that are related to the management of the NS Catalog. Through these operations, it is possible to create, list, update, and remove descriptors related to network services (*i.e.*, VNFFGD and NSD).

Table 3. NS Catalog management operations.

Operation	Description
createVnffgd	Retrieves a VNFFGD from a URL and adds it to the NS Catalog
listVnffgds	Lists information from one or all VNFFGDs in the NS Catalog
updateVnffgd	Updates information a given VNFFGD in the NS Catalog
deleteVnffgd	Removes a given VNFFGD from the NS Catalog
createNsd	Retrieves an NSD from a URL and adds it to the NS Catalog
listNsds	Lists the details of one or all NSDs in the NS Catalog
updateNsd	Updates a given NSD in the NS Catalog
deleteNsd	Removes a given NSD from the NS Catalog

The NS Catalog descriptors are also standardized according to the TOSCA specifications and the NFV-MANO reference architecture. Thus, each NSD of the NS Catalog uses an existing VNFFGD to define the sequence in which VNFs will be chained together when composing a service, among other tasks. In Vines, an NSD is treated as a template that can be used to create instances of SFCs (*i.e.*, the executions of the network services themselves).

Vines also maintains an NFV Instances Repository. This repository stores information about VNF instances and SFC instances. That information is automatically updated as the respective instances are executed and management operations are executed on them. This is described in more detail below in Subsections 4.2 and 4.3.

4.2 VNF Lifecycle Management Implementation

The Vines VNFM is designed to provide complete lifecycle management of heterogeneous VNFs. Therefore, Vines provides all the traditional VNF management operations, as shown in Table 4.

Table 4. VNF lifecycle management operations.

Operation	Description
deployVnf	Instantiates a VNF
listVnfs	Lists information about one or all instantiated VNFs
scaleVnf	Scales vertically a VNF instance
recoverVnf	Recovers a crashed VNF instance
updateVnf	Updates a VNF instance
destroyVnf	Removes a VNF instance

Vines provides these high-level functionalities by abstracting the various operations that constitute them, bringing transparency to user execution. Most of the basic VNF lifecycle management operations involve the communication between the VNFM and the VIM, which executes the corresponding lower-level operations. However, some operations require an interaction between the VNFM and the EMS responsible for each managed VNF. Examples include operations such as *deployVnf*, *scaleVnf*, and *recoverVnf*, which involve lower-level tasks such as initializing the NF within the VNF-ExP. In the case of the *deployVnf* operation, low-level tasks such as retrieving the VNFP from the VNF Catalog, installing the artifacts, and configuring the NF are also required. Note that those operations are directly related to the VNF instances, and they are used to handle the VNFs in the VNF Instance Repository.

In order to provide the full management of VNF instances, Vines also provides direct access to NF management operations. This means that through the Vines interfaces, the network operator is able to control the network function running on the VNF-ExP, whether it is a legacy NF or a newly developed NF. Table 5 describes the operations related to this type of management. The execution of all those operations also involves the communication between the VNFM and the EMS of each VNF.

Table 5. NF management operations.

Operation	Description
startNf	Initializes the network function within the VNF instance
stopNf	Stops the network function within the VNF instance
getNfStatus	Gets the current status of the network function

The Vines EMS, presented in detail in Subsection 3.2, is implemented separately from the platform core. All EMS modules were developed using the Python programming language. The communication with the VNFM is based on HTTP. The *Access Interface* module which is responsible for that communication, was implemented based on the Flask[4] framework. The EMS *VIB* consists of a set of files that are handled by other EMS modules. The *Extended Drivers* are Python files that implement operations related to the *Em-Vnf* reference point, which provides a standardized interface. Each *Extended Driver* is imported by the EMS as a Python module (a library) and is responsible for adapting the interface to the particular VNF-ExP being used.

Currently, Vines EMS instantiation is manually done by the network operator responsible for preparing and/or managing the NFV platform. Each new EMS instance created needs to be recognized by the Vines VNFM. For this purpose, the VNFM provides a simplified operation to register an EMS instance on the platform, which enables the VNFM-EMS communications. As described in Subsection 3.2, the VNFM of Vines can subscribe to each known EMS to receive monitoring notifications from VNFs. Sending those notifications is done by each EMS to the VNFM via an operation that is

[4]https://flask.palletsprojects.com

also provided by Vines. Thus, when sending a notification to the VNFM, each EMS must authenticate itself and inform the unique identifier of the VNFM subscription (this identifier is generated during the subscription process). The EMS-related operations are shown in Table 6.

Table 6. EMS-related operations.

Operation	Description
registerEms	Registers a new EMS instance
notifyVnfm	Notifies the VNFM of a VNF monitoring event

The VNF-ExP framework (presented in Subsection 3.2) defines a high-level architecture, which was implemented as the Cloudstack/Vines VNF-ExP: Vines-Leaf. Similar to the EMS Access Interface module, the Vines-Leaf Management Agent was implemented in Python and Flask. To execute and manage an NF, Vines-Leaf uses VNFP artifacts, such as management scripts (prepared by the NF developer) and the NF code (*e.g.*, binary files), among others. This allows the Management Agent to perform activities such as source code compilation, software installation, application launch, among others.

Finally, Extended Drivers are provided for the following VNF-ExPs: Vines-Leaf, Click-On-OSv, and COVEN.

4.3 Implementation of Network Service Orchestration

The Vines NFVO communicates with the VIM (*i.e.*, CloudStack) and the VNFM through their respective Java interfaces. In addition, an Extended Driver has been prepared to extend EMS functionalities to support VR-related operations. The communication between the EMS and the VR is performed through SSH commands, avoiding the need to add other components to mediate VR communication. This approach is feasible due to the strict set of commands required (route configurations and firewall rules) and the fact that each CloudStack Management Server already has by default an SSH key that allows administrative access to the VRs.

The set of all service orchestration functionalities that can be performed by the NFVO is made available through the operations described in Table 7. As with the VNFM, these operations are accessed by users via CloudStack's native API.

Table 7. Orchestration operations for virtualized network services.

Operation	Description
createSfc	Triggers all actions required to compose an SFC
startSfc	Initializes the NF of all VNFs belonging to an SFC
stopSfc	Stops the NF of all VNFs belonging to an SFC
listSfcs	Lists the information of one or all SFC instances
updateSfc	Changes the operational and data parameters of a SFC instance
destroySfc	Deletes an SFC and releases the computational and network resources

The implementation of operation *listSfcs* is relatively trivial, basically involving read operations on the CloudStack database. However, all the other operations referring to SFCs include more complex procedures that effectively require the NFVO capabilities. The *createSfc* operation can be considered the most important of those provided by the Vines NFVO, as it chains together individual instances of VNFs in a predefined order to compose a given SFC. This operation implements the SFC composition strategy described in Subsection 3.3, performing route deviation configuration in the VR and adjusting each VNF to forward traffic to the next hop along the chain. Such settings are made based on the order of VNFs and traffic classification rules defined in the VNFFGD that is pointed to by the NSD used at the time of SFC composition.

The VNFFGD can contain one or more traffic classification rules. Each rule defines a protocol and its respective port. Thus, it is possible, for example, to specify that only TCP traffic on port 80 should flow through a specific SFC (which, in this case, corresponds to typical HTTP traffic). The execution of the *createSfc* operation causes the NFVO to allow the flow of all packets that match the traffic classification rules specified in the VNFFGD. The NFVO also configures the VR to apply the appropriate traffic redirection to the public IP reserved for that specific SFC. Finally, the NFVO configures each VNF of the SFC to forward traffic as required and registers the new SFC instance in the NFV Instance Repository.

Each SFC is composed based on the definitions of the corresponding NSD (including its respective VNFFGD), which can be seen as a template for the composition of the SFC. However, after the SFC has been instantiated, the Vines NFVO allows network operators to apply modifications. This functionality is provided by the *updateSfc* operation. Through this operation it is possible, for example, to modify traffic classification rules, change VNF ordering, change the number of VNFs (*i.e.*, add or remove VNFs) in the SFC, and edit the metadata of the SFC registry in the NFV Instance Repository.

In the same way that the VNFM enables the initialization and shutdown of the NF of each VNF instance, the NFVO also enables the execution of these operations at the service level. These operations are performed by *startSfc* and *stopSfc* operations, which perform the initialization and shutdown of the NF of each VNF belonging to a given SFC, respectively. Note that when the *stopSfc* operation is executed, the SFC remains instantiated, but none of the VNFs belonging to it process any packets, since all NFs are stopped. Thus, when *startSfc* operation is executed, all VNF instances will resume processing network traffic.

Finally, the *destroySfc* operation can behave in two different ways, defined by its parameters. In the first mode, only the chaining is cleared when destroying an SFC. In this case, all the VNFs that constitute the SFC remain instantiated, but the traffic classification rules are cleared and the SFC instance records are removed from the NFV Instance Repository. In the second mode of operation, in addition to clearing the traffic classification rules and deleting the SFC records, the *destroySfc* operation also removes each VNF that is part of the SFC, freeing up all allocated compute and network resources.

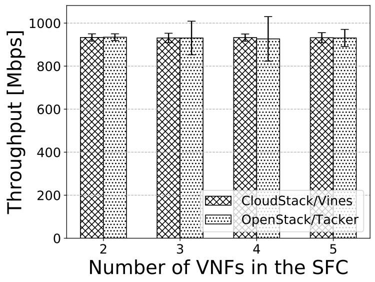
5 Evaluation

A set of experiments was conducted to evaluate CloudStack/Vines, including comparisons (when possible) with the widely adopted Openstack/Tacker solution. Experiments were executed on two different environments. One environment consisted of three machines, one for running the controller, and two of them dedicated for running SFC's. Each of those two machines was based on an Intel(R) Core(TM) i7-12700 processor @ 2.5GHz with 8 physical cores, and 16GB RAM running Linux Ubuntu 24.04 LTS, and connected on a 10 Gbps Ethernet LAN. The other environment consisted of a single machine based on an Intel(R) Core(TM) i7-6700 @ 3.40GHz processor with 4 cores, and 8 GB RAM, running Ubuntu 18.04 LTS connected on a 1 Gbps Ethernet LAN. That host was responsible for running the NFV platform as well as the SFC's. In both cases, a client generates the workload through Shell and Python scripts. The workload consisted of network traffic or management operation requests, depending on the experiment.

5.1 Performance Evaluation of Virtualized Network Services

In this subsection, we present results for the performance evaluation of network services instantiated on top of both CloudStack/Vines and Openstack/Tacker. First, we assess the throughput of the SFCs in scenarios in which the number of chained VNFs varied from 2 to 5 running locally on a single machine (the second environment specified above). All SFCs processed TCP traffic. Figure 9 shows the results obtained for the different scenarios.

Figure 9. SFC Throughput: CloudStack/Vines vs. Openstack/Tacker.



Across all scenarios and both platforms, the average throughput was consistently around 930 Mbps. Specifically, for CloudStack/Vines, the average throughput was of 933 Mbps, 930 Mbps, 932 Mbps, and 932 Mbps for SFCs with 2, 3, 4, and 5 VNFs, respectively. In comparison, Openstack/Tacker showed average values of 934 Mbps, 930 Mbps, 926 Mbps, and 931 Mbps. These results indicate that the throughput of SFCs is very similar on both platforms. However, in many scenarios, Openstack/Tacker SFCs presented greater throughput variation, suggesting less stability of its network services when handling the workload generated during the experiment.

In the next experiment, we evaluated the latency of SFCs in the same scenarios: SFC sizes changing from 2 to 5. The type of traffic generated was ICMP (Internet Control Message Protocol). The procedure adopted to compute the latency was as follows. We measured the time it took for an ICMP Echo Request (Type 8) to move through the whole SFC (all VNFs) and the ICMP Echo Reply (Type 0) to arrive from the last VNF directly to the monitor client. The results were very similar for the two NFV platforms. For this reason, Figure 10 presents the latency distribution for all scenarios in **microseconds** (μs), instead of milliseconds (ms), as usual.

Figure 10. SFC latency distribution in microseconds.
(a) Latency of CloudStack/Vines. **(b)** Latency of Openstack/Tacker.

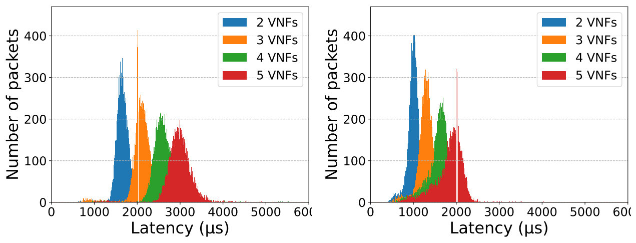
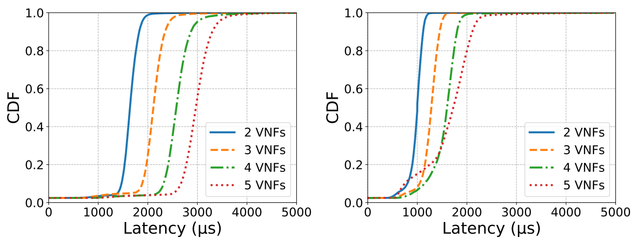


Figure 11. CDF of the latency of the SFCs.
(a) CDF of latency in CloudStack/Vines. **(b)** CDF of latency in Openstack/Tacker.



As shown in Figure 10(a), the latency distribution for network services on CloudStack/Vines was approximately 1600μs, 2100μs, 2600μs, and 3000μs for SFCs with 2, 3, 4, and 5 VNFs, respectively. In contrast, Figure 10(b) shows that Openstack/Tacker showed lower latencies for SFCs with 2 and 3 VNFs, of around 1000μs and 1250μs, respectively. However, for SFCs with 4 and 5 VNFs, Openstack/Tacker showed a significant latency dispersion.

Figure 11 presents the Cumulative Distribution Function (CDF) for the latency results that were presented in Figure 10(a). This function complements the analysis by showing the probability of network traffic latency reaching certain levels in each scenario. As shown in Figure 11(a), there is a 95% probability that the latency of CloudStack/Vines SFCs will be less than or equal to 1890μs, 2430μs, 2990μs, and 3430μs for SFCs with 2, 3, 4, and 5 VNFs, respectively. In contrast, Figure 11(b) shows that for Openstack/Tacker SFCs with the same configurations, there is a 95% probability that the latency will not exceed 1150μs, 1500μs, 1860μs, and 2190μs, respectively. Note that the time unit is microseconds, thus the difference can be considered very small.

Overall, CloudStack/Vines and Openstack/Tacker network services showed similar performance in all experiments. The difference of the peak latency between the SFCs of the two platforms was about 0.0009 seconds in the worst case (scenario with 5 VNFs, Figure 10), while the largest difference in average throughput was about 5 Mbps (with 4 VNFs). These results confirm that Vines meets the expected performance levels.

We also investigated the influence of running SFCs on top of multiple dedicated nodes instead of on a single-machine-based deployment. Figure 12 presents a comparison of the throughput observed for a single-node and a multi-node deployment, considering scenarios in which SFCs consist of 2, 4, 6, 8, and 10 chained VNFs.

Figure 12. CloudStack/Vines SFC Throughput: Single-node vs. Multi-node.

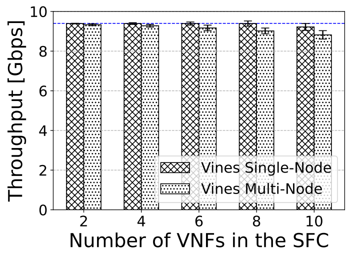
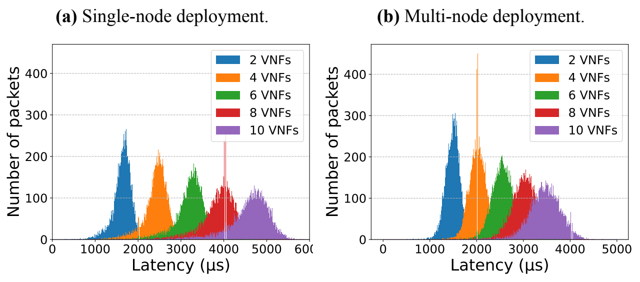


Figure 12 shows that the throughput remains consistently high in both deployments across different SFC lengths. For the single-node deployment, the throughput is nearly constant, staying close to the maximum achievable value (around 9.40 Gbps, see the blue dashed line) regardless of the number of VNFs in the chain. In contrast, the multi-node deployment exhibits a slight performance degradation as the chain length increases. For shorter chains, consisting of 2 and 4 VNFs, the throughput is comparable to the single-node case, reaching an average of 9.33 Gbps and 9.28 Gbps, respectively. However, as the number of VNFs increases (e.g., for the chains with 8 or 10 VNFs), the throughput reduces in average by approximately 3.94% and 4.34%, respectively. This reduction reflects the additional overhead introduced by inter-node communication plus the larger number of VNFs that packets have to traverse. However, despite the decrease in average throughput in scenarios with longer SFCs, the confidence intervals indicate that there is no statistically significant reduction. It also suggests that CloudStack/Vines can scale across multiple nodes with only a slight impact on throughput.

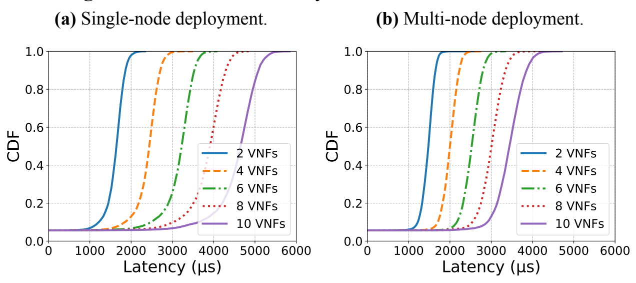
Next, we describe results obtained for the latency in the same scenarios, also using SFCs with 2, 4, 6, 8, and 10 VNFs. The latency distribution observed for both the single-node and multi-node environments is presented in Figure 13.

Figure 13. SFC latency distribution of Cloudstack/Vines in microseconds.



These results are not intuitive as one might expect that by instantiating VNFs across multiple nodes, rather than on a single one, the inter-node communication would introduce additional overhead and thus increase latency. However, Figure 13 shows that results were the opposite. This observation is confirmed by Figure 14 – note that the curves are further to the left in Figure 14(b). A discussion of why the multi-node deployment presented a lower latency (approximately 0.001 seconds in the worst case for 10 VNFs) must be based on the

Figure 14. CDF of the latency of SFCs in Cloudstack/Vines.

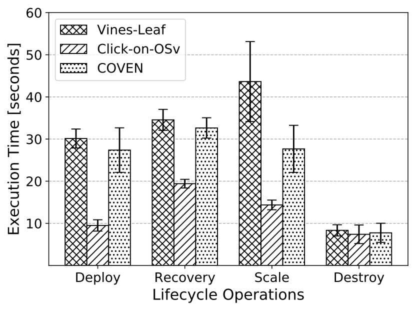


fact that there is an overhead for instantiating, scheduling, and managing multiple Virtual Machines (VMs) on a single physical machine. In particular, in the experiments each VNF runs on a different VM. Thus if there are for example 10 VNFs in a given chain, this involves sending packets locally across the 10 VMs, which requires a substantial amount of context switching. On the other hand, in case the packet is transmitted to another host, it is simply forwarded to the local network interface.

5.2 Performance Evaluation of VNF Management Operations

The last experiment was executed to evaluate Vines' VNF management capabilities. This subsection presents results for the management of multiple VNF-ExPs. As mentioned in Section 2, Openstack/Tacker does not provide native facilities to support the specific characteristics of different VNF-ExPs. Therefore, only results of CloudStack/Vines are presented. In each scenario, a VNF running a packet forwarder was instantiated using a different VNF-ExP. The following three VNF-ExPs were employed: Vines-Leaf, Click-on-OSv, and COVEN. For each VNF-ExP, four operations – deploy, recover, scale, and remove – were performed 30 times each. The results, presented in Figure 15, show the average execution time for each operation, with a 95% confidence interval.

Figure 15. Average execution time of management operations.



CloudStack/Vines successfully completed all operations in under 60 seconds, with most runs averaging less than 40 seconds. However, the execution times varied depending on the VNF-ExP used. Among the three VNF-ExPs considered, Click-On-OSv consistently presented the lowest execution times for all operations. This result was expected, as Click-On-OSv is based on a unikernel operating system (OSv), which uses a minimal set of libraries and operates within a single address space for both user and kernel applications. COVEN and Vines-Leaf, on the other hand, were instantiated on Ubuntu Cloud. COVEN benefits from the parallel execution of many internal sub-operations, resulting in a faster performance compared to Vines-Leaf – in particular for the deployment, recovery, and scaling operations, as shown in Figure 15. Conversely, Vines-Leaf, while slightly slower, offers more flexibility by being able to run any NF, regardless of its implementation, at a similar cost as that of Coven.

5.3 A Comparison of Vines VNF Lifecycle with OpenStack-Based NFV Solutions

In this subsection we describe the main contributions of Vines, in comparison with Openstack-based NFV systems we are aware of. The goal is to show in which aspects Vines facilitates SFC/VNF lifecycle management compared to alternatives.

1. Vines features a native VNF-ExP, and moreover can run any other VNF-ExP. To the best of our knowledge, none of the Openstack-based NFV platforms offer that support. They do not even have a native VNF-ExP, nor a comprehensive EMS. Open Baton does have a simple EMS that is executed *within* each VNF instance. Moreover, that EMS simply mediates the communication between the VNF and the VNFM. And this approach has several other drawbacks: *(i)* either the VNF source code or the corresponding VNF-ExP code has to be changed to include the EMS. In the case of VNF-ExPs that are based on a unikernel/minimalist OS (like OSv or Click OS, for example) it is necessary to change the kernel itself.
2. Vines supports different protocols for VNF communication along the lifecycle management. Frequently, in other NFV platforms, the communication between the VNFM and each specific VNF requires a particular communication protocol. For example, Open Baton requires AMQP (Advance Message Queuing Protocol) and RabbitMQ.
3. Vines facilitates VNF/SFC lifecycle management, including deployment (instantiation), scaling, recovery, among others. For example, Tacker has a generic component called “Management Drivers”, basically featuring the *cloud-init* tool which only supports initial VNF configurations, but depends on an external EMS for additional operations. Furthermore, the initial VNF configuration is not performed directly by the VNF Manager, as this task is delegated to the VIM via *cloud-init*. The VnfLcmController component [5] only considers the basic create/update/delete lifecycle operations and delegates other operations for the VIM to execute. Frequently, the operations are very basic. For instance, the Tacker update operation just updates the VM that runs the VNF, instead of updating the VNF itself. Vines, on the other hand, is capable of updating the VNF by modifying the network function code (which can be used to update it to a new version, for example).
4. Vines presents a crucial advantage as it simplifies NF (besides VNF) management. This issue is often overlooked in the literature. The very essence of NFV is to enable the implementation of Network Functions (NFs) in software with virtualization technology. However, each NF could also be implemented in hardware or using other software strategies. The NF is the “soul” of the NFV. The EMS is the component that allows NF management Fulber-Garcia *et al.* (2023). As mentioned above, Tacker neither has an EMS nor supports third-party EMSes. As a result, only some primitive NF management tasks that can be executed through the cloud-init (Canonical, 2025a) application, which allows network operators to run shell scripts to configure an NF during its deployment.
5. An advantage that Vines inherits from CloudStack: everything is a CloudStack plugin. On the other hand, to execute Openstack Tacker, one needs to execute multiple independent systems (or projects) that evolve independently. Very often, a new version of one of those systems will prevent the NFV system to execute.

[5] https://docs.openstack.org/tacker/latest/user/architecture.html

6 Conclusion

This work proposed Vines, which is, to the best of our knowledge, the first Apache CloudStack NFV platform. Although the importance of the NFV paradigm cannot be overestimated, and Apache CloudStack is one of the most adopted open source cloud platforms worldwide, it is surprising that CloudStack has barely been explored in the context of NFV technology. Vines is compliant with the ETSI NFV-MANO specifications, and supports the composition and management of virtualized network services (implemented as SFCs) on top of CloudStack’s native virtual networks.

The main contributions of Vines in comparison with the Openstack-based NFV platforms are as follows. Vines relies on a native full-fledged EMS, which is the basis for holistic and fine-grained management of heterogeneous VNFs, also simplifying management of the NF itself. Vines features a native VNF-ExP and supports other VNF-ExPs, furthermore, it also supports different protocols for VNF communication along the lifecycle management. Finally, Vines is a CloudStack plugin, which facilitates system use in comparison to the Openstack platforms, which often require a complex combination of multiple independent systems. An empirical evaluation shows that Vines achieves SFC throughput and latency that closely align with the performance observed in OpenStack/Tacker. The results also confirm the effectiveness of Vines in managing the lifecycle of VNFs instantiated on different VNF-ExPs.

One of our main targets as future work is to integrate Vines to the Apache CloudStack standard distribution. In order to do that, it is necessary to add unit tests to each and every module, which although not being a hard task is labor- and time-intensive. Running Vines on large-scale deployments as well as geographically distributed multi-site scenarios is also one of our main targets. We believe several challenges will emerge from the experience of running large scale SFC’s on those kinds of scenarios. Issues such as setting up and managing SFCs across multiple domains, clouds, and orchestrators are among those, and intelligent approaches (Fulber-Garcia *et al.*, 2024) should be integrated as native Vines features. Future work also includes adding new features to CloudStack/Vines, such as support for dynamic traffic reclassification and also for VNF chaining based on NSH (Network Service Header).

From a more general perspective, we highlight that although CloudStack Vines is designed with interoperability in mind, enabling holistic communication between the NFV-MANO components (VNFM and NFVO) and the VNFs through their execution platforms, several interoperability challenges remain open within the broader NFV reference architecture. For example, the literature provides few insights into how to enable communication between the NFV-MANO and the Element Management Systems through the VNFM (Ve-Vnfm-em interface). Although recent works have presented a comprehensive description of the EMS architecture Fulber-Garcia *et al.* (2023); ETSI (2020) defining its primary operations and attributes, there are still no concrete specifications, neither from standardization organizations (ETSI/IETF) nor from recent research papers, on how to actually implement the Ve-Vnfm-em interface within VNFM platforms. Furthermore, the interoperability challenge extends to the heterogeneity of OSS/BSS solutions when implementing the Os-Ma-nfvo interface. Even though the ETSI provides interface definitions, data models, and technical guidelines for this interface ETSI (2023, 2022), many OSS/BSS solutions are not yet fully prepared to operate with the NFV paradigm, requiring additional efforts from NFV-MANO developers to intermediate and translate legacy protocols into the NFV context in order to fulfill system requests. These aspects indicate important NFV research directions and highlight critical points of attention that will enable the NFV-MANO model and the CloudStack Vines platform to better adapt to real-world needs and support deployment in production networks.

Declarations

Authors' Contributions

José Flauzino contributed to the conception of this study, developed the methodology, performed the experiments, and wrote the manuscript. Vinicius Fülber-Garcia, Alexandre Huff, Giovanni Venâncio, and Elias P. Duarte Jr. contributed to the conception of this study, developed the methodology, and reviewed the manuscript. Elias P. Duarte Jr. also supervised the work and wrote the manuscript. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests related to the research, authorship, and/or publication of this article.

Acknowledgements

The authors would like to thank Amanda Viescinski for her assistance in preparing the infrastructure for conducting part of the experiments.

Funding

This work has been partially supported by the Coordination for the Improvement of Higher Education Personnel (CAPES) - Program of Academic Excellence (PROEX), Funding Code 001; and the Brazilian National Council for Scientific and Technological Development (CNPq) - grant 305108/2025-5.

Availability of data and materials

The software developed in the current study is available at the project page: https://www.inf.ufpr.br/jwvflauzino/vines.

References

Anuket (2025). Anuket. `https://anuket.io/`. Accessed Oct., 2025.

ASF (2025a). Apache CloudStack. Available at:`http://cloudstack.apache.org`. Accessed Oct., 2025.

ASF (2025b). Apache CloudStack Users. Available at:`http://cloudstack.apache.org/users.html`. Accessed Oct., 2025.

ASF (2025c). VNF Appliance Integration. Available at:`https://cwiki.apache.org/confluence/display/CLOUDSTACK/VNF+Appliance+Integration`. Accessed Oct., 2025.

Bondan, L., Franco, M. F., Marcuzzo, L., Venancio, G., Santos, R. L., Pfitscher, R. J., Scheid, E. J., Stiller, B., De Turck, F., Duarte, E. P., *et al.* (2019). FENDE: Marketplace-based distribution, execution, and life cycle management of VNFs. *Communications Magazine*, 57(1):13–19. DOI: 10.1109/mcom.2018.1800507.

Canonical (2025a). Cloud-init: The standard for customising cloud instances. Available at:`https://cloud-init.io/`. Accessed Oct., 2025.

Canonical (2025b). Deploy Osm using Charmhub - The Open Operator Collection. Available at:`https://charmhub.io/osm`.Accessed Oct., 2025.

Chiosi, M., Clarke, D., Willis, P., Reid, A., Feger, J., Bugenhagen, M., Khan, W., Fargano, M., Cui, C., Deng, H., *et al.* (2012). Network functions virtualisation: An introduction, benefits, enablers, challenges & call for action. Technical report, European Telecommunications Standards Institute. Available at:`https://portal.etsi.org/nfv/nfv_white_paper.pdf`.

Cloudify (2025). Cloudify devops automation & orchestration platform, multi cloud. Available at:`https://cloudify.co/`. Accessed Oct., 2025.

da Cruz Marcuzzo, L., Garcia, V. F., Cunha, V., Corujo, D., Barraca, J. P., Aguiar, R. L., Schaeffer-Filho, A. E., Granville, L. Z., and dos Santos, C. R. (2017). Click-on-osv: A platform for running click-based middleboxes. In *The 15th IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 885–886. IFIP/IEEE. DOI: 10.23919/inm.2017.7987396.

ETSI (2014). Network functions virtualisation (nfv): Management and orchestration. Technical report, European Telecommunications Standards Institute. Available at:`https://www.etsi.org/technologies/nfv`.

ETSI (2018). Network functions virtualisation (nfv) release 2; protocols and data models; vnf package specification. Technical report, European Telecommunications Standards Institute. Available at:`https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/006/02.08.01_60/gs_NFV-SOL006v020801p.pdf`.

ETSI (2020). Network functions virtualisation (nfv) release 3; management and orchestration; ve-vnfm reference point - interface and information model specification. Technical report, European Telecommunications Standards Institute. Available at:`https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/010/03.03.01_60/gs_NFV-IFA010v030301p.pdf`.

ETSI (2022). Network functions virtualisation (nfv) release 4; protocols and data models; restful protocols specification for the os-ma-nfvo reference point. Technical report, European Telecommunications Standards Institute. Available at:`https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/009/04.04.01_60/gs_NFV-SOL009v040401p.pdf`.

ETSI (2023). Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Os-Ma-nfvo reference point - Interface and Information Model Specification. Technical report, European Telecommunications Standards Institute. Available at:`https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/048/04.04.01_60/gs_NFV-IFA048v040401p.pdf`.

ETSI (2025). Open Source MANO. Avaialble at:`https://osm.etsi.org`. Accessed Oct., 2025.

Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine - EUA. Available at:`https://roy.gbiv.com/pubs/dissertation/fielding_dissertation.pdf`.

Flexera (2022). State of the cloud report from flexera. Technical report, Flexera: IT Management Software, Optimization & Solutions. Available at:`https://info.flexera.com/CM-REPORT-State-of-the-Cloud?lead_source=Organic%20Search`.

Fulber-Garcia, V., Duarte Jr, E. P., Huff, A., and dos Santos, C. R. (2020). Network service topology: Formalization, taxonomy and the custom specification model. *Computer Networks*, 178:107337. DOI: 10.1016/j.comnet.2020.107337.

Fulber-Garcia, V., Flauzino, J., Dos Santos, C. R., and Duarte, E. P. (2023). An etsi-compliant architecture for the element management system: The key for holistic nfv management. In *The 19th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE. DOI: 10.23919/cnsm59352.2023.10327882.

Fulber-Garcia, V., Flauzino, J., Venâncio, G., Huff, A., and Junior, E. P. D. (2024). Breaking the limits: Bio-inspired sfc deployment across multiple domains, clouds and orchestrators. In *2024 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–6. IEEE. DOI: 10.1109/nfv-sdn61811.2024.10807470.

Garcia, V. F., Marcuzzo, L. C., Souza, G. V., Bondan, L., Nobre, J. C., Schaeffer-Filho, A. E., Santos, C. R. P. d., Granville, L. Z., and Duarte, E. P. (2019a). An NSH-enabled architecture for Virtualized Network Function platforms. In *The 33rd International Conference on Advanced Information Networking and Applications (AINA)*, pages 376–387. Springer. DOI: 10.1007/978-3-030-15032-7$_3$2.

Garcia, V. F., Marcuzzo, L. d. C., Huff, A., Bondan, L., Nobre, J. C., Schaeffer-Filho, A., dos Santos, C. R., Granville, L. Z., and Duarte, E. P. (2019b). On the design of a flexible architecture for virtualized network function platforms. In *The 62th IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE. DOI: 10.1109/globecom38437.2019.9013111.

Huff, A., Venâncio, G., Garcia, V. F., and Duarte, E. P. (2020). Building multi-domain service function chains based on multiple nfv orchestrators. In *The 6th IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 19–24. IEEE. DOI: 10.1109/nfv-sdn50289.2020.9289849.

Kourtis, M.-A., McGrath, M. J., Gardikis, *et al.* (2017). T-nova: An open-source mano stack for nfv infrastructures. *IEEE Transactions on Network and Service Management*, 14(3):586–602. DOI: 10.1109/tnsm.2017.2733620.

Krishnan, P., Duttagupta, S., and Achuthan, K. (2020). Sdn/nfv security framework for fog-to-things computing infrastructure. *Software: Practice and Experience*, 50(5):757–800. DOI: 10.1002/spe.2761.

Martins, J. *et al.* (2014). Clickos and the art of network function virtualization. In *The 11th Symposium on Networked Systems Design and Implementation (NSDI)*, pages 459–473. USENIX. Available at:`https://www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-martins.pdf`.

Mijumbi, R., Serrat, J., Gorricho, J.-L., Bouten, N., De Turck, F., and Boutaba, R. (2015). Network Function Virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262. DOI: 10.1109/comst.2015.2477041.

Mijumbi, R., Serrat, J., Gorricho, J.-L., Latré, S., Charalambides, M., and Lopez, D. (2016). Management and orchestration challenges in network functions virtualization. *IEEE Communications Magazine*, 54(1):98–105. DOI: 10.1109/mcom.2016.7378433.

ONAP (2024). ONAP Documentation - VF-C Architecture. Available at:`https://docs.onap.org/projects/onap-vfc-nfvo-lcm/en/kohn/platform/architecture.html#vf-c-internal-component`. Accessed Oct., 2025.

ONAP (2025). Open Network Automation Platform. Available at:`https://www.onap.org/`. Accessed Oct., 2025.

OpenBaton (2025). Open Baton: an open source reference implementation of the ETSI Network Function Virtualization MANO specification. Available at:`https://openbaton.github.io`. Accessed Oct., 2025.

OpenStack (2025). OpenStack - Main Page. Available at:`https://wiki.openstack.org/wiki/Main_Page`. Accessed Oct., 2025.

Son, J., He, T., and Buyya, R. (2019). Cloudsimsdn-nfv: Modeling and simulation of network function virtualization and service function chaining in edge computing environments. *Software: Practice and Experience*, 49(12):1748–1764. DOI: 10.1002/spe.2755.

St-Onge, C., Kara, N., and Edstrom, C. (2023). Nfvlearn:

A multi-resource, long short-term memory-based virtual network function resource usage prediction architecture. *Software: Practice and Experience*, 53(3):555–578. DOI: 10.1002/spe.3160.

Tacker (2025). Tacker - OpenStack NFV Orchestration. Available at:`https://wiki.openstack.org/wiki/Tacker`. Accessed Oct., 2025.

Tavares, T. N., da Cruz Marcuzzo, L., Garcia, V. F., de Souza, G. V., Franco, M. F., Bondan, L., De Turck, F., Granville, L. Z., Junior, E. P. D., dos Santos, C. R. P., *et al.* (2018). Niep: Nfv infrastructure emulation platform. In *The 32nd International Conference on Advanced Information Networking and Applications (AINA)*, pages 173–180. IEEE. DOI: 10.1109/aina.2018.00037.

Turchetti, R. C. and Duarte, E. P. (2015). Implementation of failure detector based on network function virtualization. In *2015 IEEE International Conference on Dependable Systems and Networks Workshops*, pages 19–25. IEEE. DOI: 10.1109/dsn-w.2015.30.

Turchetti, R. C. and Duarte Jr, E. P. (2017). Nfv-fd: Implementation of a failure detector using network virtualization technology. *International Journal of Network Management*, 27(6):e1988. DOI: 10.1002/nem.1988.

Venâncio, G., Garcia, V. F., da Cruz Marcuzzo, L., Tavares, T. N., Franco, M. F., Bondan, L., Schaeffer-Filho, A. E., Paula dos Santos, C. R., Granville, L. Z., and P. Duarte Jr, E. (2021). Beyond vnfm: Filling the gaps of the etsi vnf manager to fully support vnf life cycle operations. *International Journal of Network Management*, 31(5):e2068. DOI: 10.1002/nem.2068.

Venâncio, G., Turchetti, R. C., and Duarte, E. P. (2019). Nfv-rbcast: Enabling the network to offer reliable and ordered broadcast services. In *2019 9th Latin-American Symposium on Dependable Computing (LADC)*, pages 1–10. IEEE. DOI: 10.1109/LADC48089.2019.8995681.

Venâncio, G., Turchetti, R. C., and Duarte Jr, E. P. (2022). Nfv-coin: unleashing the power of in-network computing with virtualization technologies. *Journal of Internet Services and Applications*, 13(1):46–53. DOI: 10.5753/jisa.2022.2342.

Yousaf, F. Z., Bredel, M., Schaller, S., and Schneider, F. (2017). NFV and SDN—Key technology enablers for 5G networks. *IEEE Journal on Selected Areas in Communications*, 35(11):2468–2478. DOI: 10.1109/jsac.2017.2760418.