Realizing Network Function Virtualization Management and Orchestration with Model Based Open Architecture

YuLing Chen
Dell Research
Dell Inc.
Santa Clara, United States
Yuling_C@DELL.com

Mark Lambe
Cobham Wireless
Dublin, Ireland

Mark.lambe@aeroflex.com

Yinghua Qin
Dell Software Group
Dell Inc.
Zhuhai, China
Yinghua.Qin@software.dell.com

Wenjing Chu
Dell Research
Dell Inc.
Santa Clara, United States
Wenjing_Chu@DELL.com

***Abstract*—The European Telecommunications Standards Institute (ETSI) has published a Group Specification document outlining Network Function Virtualization Management and Orchestration (NFV-MANO) systems, advocating an open environment for the purpose of integrating multiple vendor solutions into one NFV ecosystem. To align with this initiative, there is a need to transform the specification into real products with the necessary components and interfaces following the ETSI reference architecture. To support this effort, in this article we share our experience in the exploration of further refining and developing internal components of the ETSI NFV-MANO functional blocks with identified open integration interfaces. We focus on the detailed analysis of the life cycle management of the Virtual Network Functions (VNFs), which is the most fundamental and important functionality of NFV-MANO. Based on our analysis, we propose an abstract VNF Manager Integration Interface (VNFM Integration Interface) at the southbound of VNFM, to integrate with various clouds and systems to complete the life cycle management of the VNFs. In order to achieve advanced VNF life cycle management such as auto-scaling, a model-based monitoring solution is adopted for monitoring the VNFs across multiple layers of NFV architecture. To serve as a proof of concept, we automatically deploy and scale a sample VNF using the proposed software architecture and interfaces. With the prototyping experience using the VNF model as a plugin, to integrate with traditional virtualization and physical infrastructure hierarchy as a full topology model, we evaluate using a model-based monitoring system Dell Foglight™ for realizing NFV monitoring functionalities as part of NFV-MANO architecture framework.**

***Keywords*— Network Function Virtualization (NFV); NFV Management and Orchestration (NFV-MANO); cloud managed software; virtualized IP network; NFV monitoring**

NOMENCLATURE

EM	Element Management.
ETSI	European Telecommunications Standards Institute.
LCM	Life Cycle Management.
NFV	Network Function Virtualization.
NFVI	NFV Infrastructure.
NFV-MANO	NFV Management and Orchestration.
NFVO	NFV Orchestrator.
NS	Network Service.
NSD	Network Service Descriptor.
OSS/BSS	Operations Support System (OSS) and Business Support Systems (BSS) used by Telco service providers to manage and support various end-to-end telecommunication services.
PNF	Physical Network Function.
SLA	Service Level Agreement.
SNMP	Simple Network Management Protocol
TELCO	Telecommunication.
VNF	Virtual Network Function.
VNFC	Virtual Network Function Component.
VNFM	VNF Manager.
VIM	Virtualized Infrastructure Manager.
VNFD	VNF Descriptor.
VNFFG	VNF Forwarding Graph.

I. INTRODUCTION

In recent years, Network Function Virtualization (NFV) has been introduced into the Telecom industry to deliver reliable and efficient commercial services by deploying the service and networking components as software on standardized and programmable hardware systems, called Virtualized Network Functions (VNFs). NFV promises benefits in two areas: one is operational and capital expenditure (OpEx and CapEx)

978-3-901882-77-7 (C) 2015 IFIP

savings obtained from general purpose hardware; the other is increased automation, resulting in operations simplification, business agility, and faster time to market. NFV has been recognized as having the potential to revolutionize the whole telecommunication industry [1].

In the fall of 2012, leading Telecom service providers initiated a NFV standardization effort in the European Telecommunications Standards Institute (ETSI). Since then, standards efforts and proofs-of-concepts have made rapid progress. However, NFV is still in its early phase. Many challenges need to be addressed before considerable market adoption of NFV solutions is seen. The biggest challenge is in delivering carrier grade solutions, where operators need to deploy industry-standard tools to automate on-boarding, deployment and scaling of virtualized network functions. There has been considerable effort by researchers, standardization bodies, and commercial products on NFV Management and Orchestration (NFV-MANO) [2] [3] [4] [5]. From our research and studies on the available NFV-MANO tools, we observe that different tools expose different arrays of APIs to orchestrate limited set of VNFs on limited types of cloud management systems. However, the network service provider needs to deal with a much broader range of VNFs in various cloud environments. This requires a significant integration effort from the service provider. ETSI, as one of the major standardization bodies in this area, has published an architectural framework with multiple functional blocks and reference points to help with the integration of various vendor solutions into one ecosystem. Nevertheless, for the industry to generate products following these specifications, there is a need to transform the architectural framework into a pragmatic software architecture with definitions of key components to implement NFV-MANO functions. At the same time, well-defined interfaces are needed between the functional blocks and components in the architecture to help with integration in the system.

To support this effort, we propose an open software architecture with identified key components necessary for implementing the ETSI NFV-MANO architectural framework. In such a system the functional blocks defined in the ETSI NFV-MANO reference architecture are further developed into detailed components to realize the specified NFV-MANO features. Since the most fundamental functionality of NFV-MANO is to automatically deploy the VNFs into the cloud, as the first step we delve into a detailed analysis of the VNF life cycle management tasks and operations that need to be completed in the virtualized infrastructures. Based on our analysis, we have come up with a set of open and generic interfaces that abstracts the implementation details for integrating various types of clouds and systems to complete the life cycle management of the VNFs. Furthermore, we present a prototype to automatically deploy an example VNF in OpenStack managed cloud environments using the proposed APIs. With prototyping of VNF life cycle management use cases such as auto-scaling, we conclude that the Dell Foglight [10] monitoring system is a good fit for realizing NFV monitoring functionalities in performance monitoring, fault management, and rule engine with extensible, flexible, and pluggable model-based architecture.

In the next section, we provide the overview of the ETSI NFV-MANO architectural framework. Then we continue with the discussion on the refined NFV-MANO software architecture with open integration interfaces as well as results from the corresponding proof of concept. Finally, we summarize our evaluation on Foglight with prototype experience using this tool in NFV monitoring.

II. ETSI NFV-MANO Architectural Framework Overview

ETSI published the NFV-MANO reference architectural framework in the Network Functions Virtualization Management and Orchestration Group Specification [3]. The NFV-MANO architectural framework defines functional blocks that belong to NFV-MANO, other functional blocks that interact with NFV-MANO, and all reference points that enable communication and data exchange among the functional blocks.

The architectural framework specifies three functional blocks in the NFV-MANO domain: NFV Orchestrator (NFVO), VNF Manager(s) (VNFMs), and Virtualized Infrastructure Manager(s) (VIMs). These three functional blocks work together to achieve the functionality of NFV Management and Orchestration.

The NFVO is responsible for the on-boarding of a new Network Service (NS) composed of multiple VNFs, VNF forwarding graph, Virtual Links, and, as an option, Physical Network Functions (PNFs). The orchestrator also controls the life cycle of the Network Service including instantiation, scale-in/out or up/down, performance measurements, event correlation and termination. Further key operational functions are global resource management, validation and authorization of the NFVI resource requests, as well as policy management of the Network Service instances.

The VNFM focuses on the life cycle management of individual VNF instances. A VNF manager takes the responsibility of the management of a single VNF instance, or the management of multiple VNF instances of the same type. Common functions for all VNFs together with specific functions required by individual VNFs need to be supported in the VNF manager. VNFM also serves as an overall coordination and adaptation role for the configuration and event reporting between the VIM and the EM systems of traditional operator architectures.

The VIM is responsible for controlling and managing the NFVI compute, storage and network resources. At the same time, it collects performance measurements in the infrastructure and makes the data available for other functional blocks for monitoring purposes.

NFV-MANO architectural framework also identifies external functional blocks that share reference points with NFV-MANO. This includes Element Management (EM), Virtualized Network Function (VNF), Operation System Support (OSS) and Business System Support Functions (BSS), and NFV Infrastructure (NFVI).

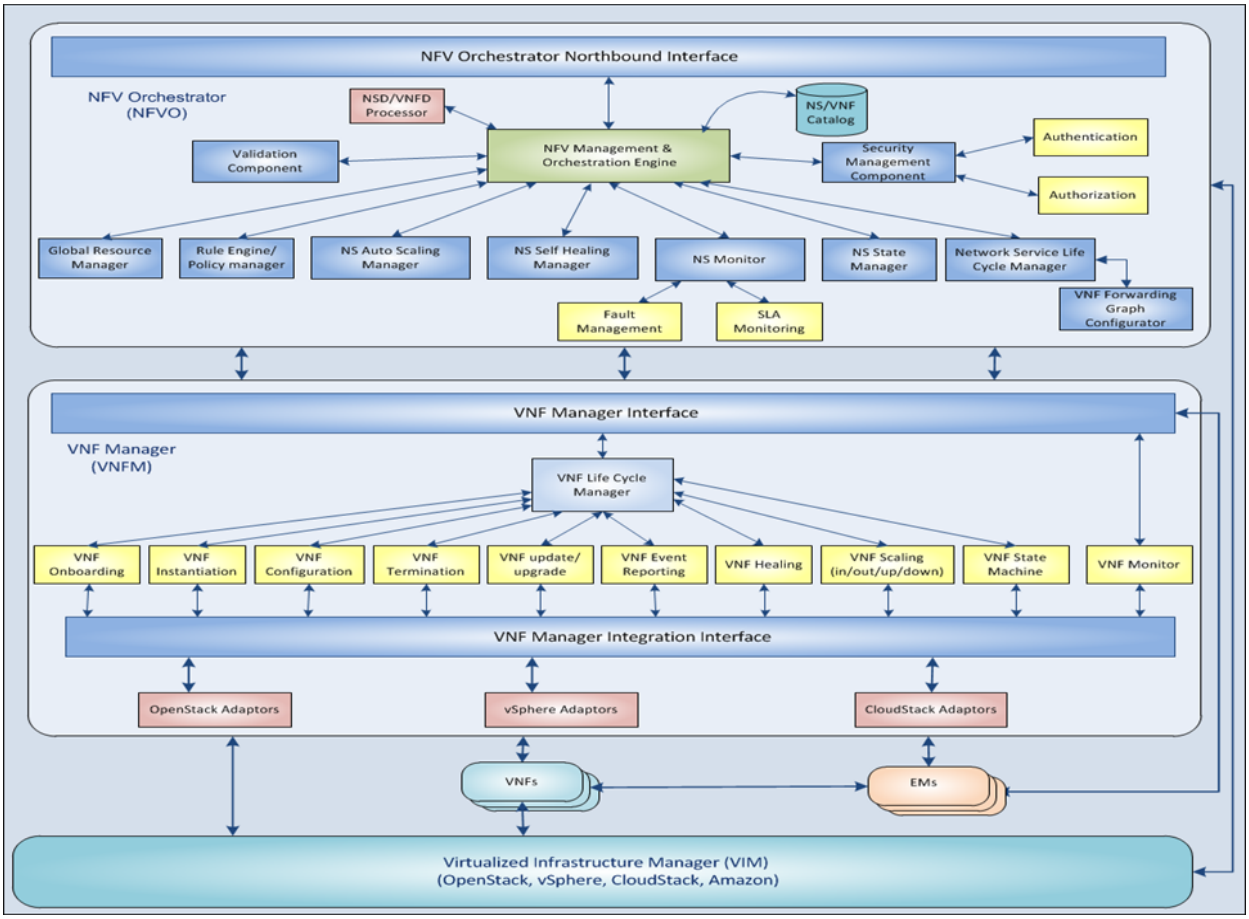

Fig. 1. Refined NFV-MANO system architecture with open integration interfaces

The reference points specified in the NFV-MANO reference architecture identifies the communication and data exchange between NFV-MANO functional blocks and their communication with external functional blocks. This includes Os-Nfvo for the interface between OSS/BSS and NFVO, VeEn-Vnfm between EM and VNFM, VeNf-Vnfm between VNFs and VNFM, Nf-Vi between NFVI and VIMs, Vnfm-Vi between VNFM and VIMs, Nfvo-Vnfm between NFVO and VNFM, and Nfvo-Vi between NFVO and VIM.

III. NFV-MANO Systems Software Architecture with Key Components and Interfaces

Based on ETSI's outline specification of the NFV-MANO architectural framework, we further refine and develop the functional blocks into software components, helping to implement and realize the functionality mentioned in ETSI NFV-MANO specifications. Fig. 1 illustrates the refined software architecture with key components defined in each functional block, as well as three distinct interfaces between these functional blocks.

As described in the diagram, the NFVO provides network service level management and orchestration. The central component of NFVO is the NFV management and orchestration engine. It receives the network service requests from the NFVO northbound interface. Each service request comes with a Network Service Descriptor (NSD), which contains references to one or multiple VNF Descriptors (VNFDs), to describe the metadata of the network service. The engine will first use the NSD Processor and VNFD Processor to parse and process the corresponding descriptors. Then the engine will call the validation component to validate the request. After validation, the engine will check if this is a network service on-boarding request. If so, it will register the network service and the related VNFs into the catalog. After that, it will call the security management component to authenticate and authorize the user to execute the operations. If successful, the engine will delegate this request to the corresponding components to execute the logic.

For network service life cycle management requests, NFVO calls the network service life cycle manager to process the requests. For individual VNF life cycle management, the network service life cycle manager will delegate the operations to VNFM. For VNF Forwarding Graph configuration in service chaining scenario, the network service life cycle manager will call VNF Forwarding Graph configurator to configure necessary virtual and physical network functions to construct a functional VNF forwarding graph.

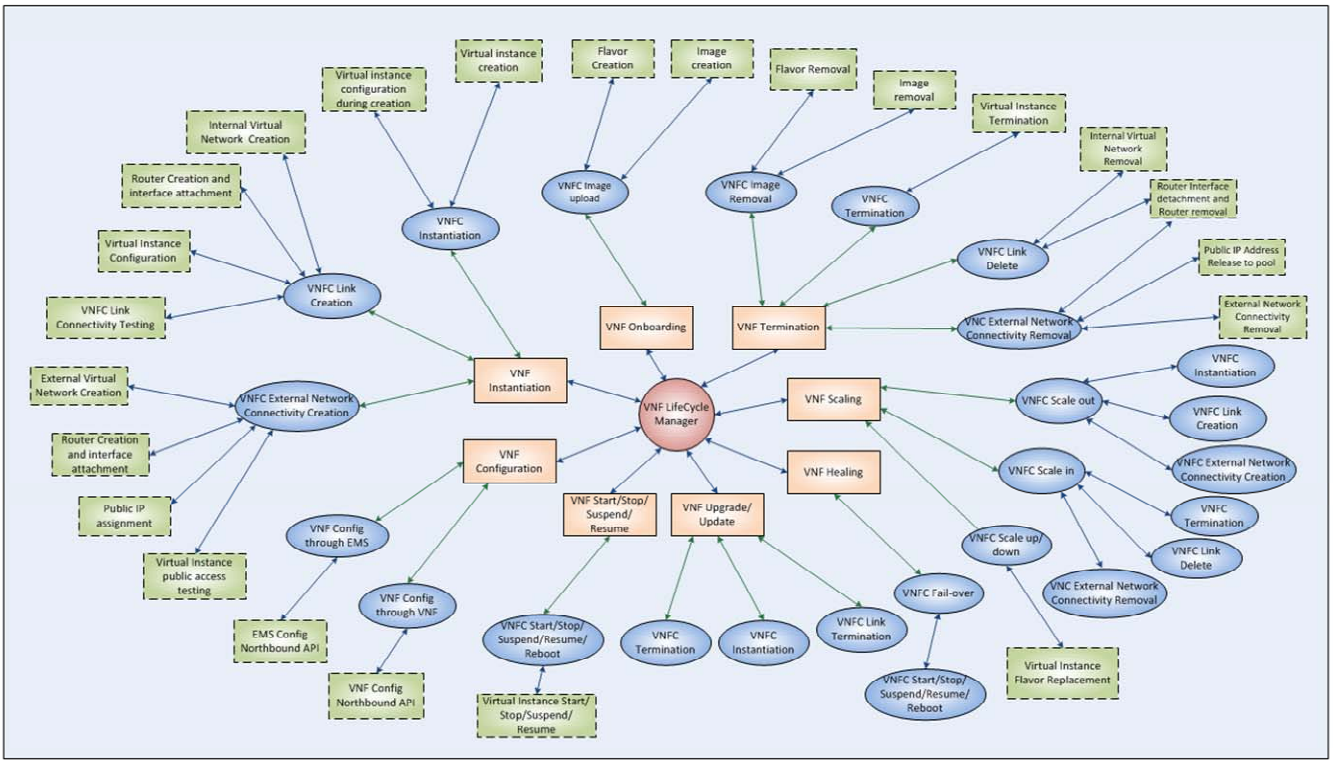

Fig. 2. VNF life cycle management analysis - tasks and operations

NFVO also contains an NS state manager and an NS monitor. Both of them communicate with the VNFM and VIM to obtain the state and other necessary information regarding each VNF and the underlying infrastructure to present a consolidated view of the network services state, fault, and SLA performance.

When the workload of the network service changes dynamically at runtime, NFVO could automatically scale the corresponding VNFs through the NS auto-scaling manager. When the fault happens in the VNFs, NFVO could automatically repair the VNF through NS self-healing manager. In both cases, NFVO leverages the monitoring results from the NS monitor and the rule engine/policy manager to trigger actions. The actions would instruct VNFM to either auto-scale or heal the VNFs using different approaches and based on different scenarios.

In the case when there are multiple clouds in different geographical locations, the global resource manager provides the capability for global resource planning and allocation. Various types of information and suggestions based on runtime monitoring and analytics could be provided to the user to assist with resource planning and allocation activities.

VNFM manages the life cycle of each individual VNF. The VNF life cycle manager, which is the central component in VNFM, receives VNF life cycle management requests from NFVO through the VNF Manager Interface. It delegates to the corresponding components including VNF on-boarding, instantiation, configuration, self-healing, auto-scaling, upgrade/update, and termination to complete the operations. VNF state machine maintains the state of each VNF. VNF monitor, which is optional in VNFM could monitor VNFs for fault and performance and make the status available for NS Monitor in NFVO to come up with the consolidated view of network service status.

The monitoring related modules including NS monitor, VNF monitor, and rule engine/policy manager could also be provided as functionalities from separate, self-contained NFV monitoring tools. In the latter case, NFVO will work together with the NFV monitoring tool to achieve the related functionalities.

In the system architecture for open integration as described in Fig. 1, there are three distinct sets of interfaces: NFVO Northbound Interface, VNF Manager Interface, and VNF Manager Integration Interface. When comparing the interfaces with the reference points in the ETSI NFV-MANO Group Specification, the NFVO Northbound Interface maps to Os-Nfvo between OSS/BSS and NFVO; the VNF Manager Interface maps to Nfvo-Vnfm between NFVO and VNFM; the VNFM Integration Interface covers VeEn-Vnfm, VeNf-Vnfm, and Vi-Vnfm. The VNFM Integration Interface groups these reference points together so as to complete VNF life cycle management by integrating with VIMs, VNFs, and EM systems. Some VNFs come with the Element Managers (EMs) and the VNFM needs to communicate with EMs for configuration of the VNFs.

One of the most basic and important requirements for a NFV-MANO system is to automatically deploy a network service into the cloud. This requires the automatic deployment of individual VNFs, which will go through the life cycle management of the VNFs. In the next section, we focus on the detailed analysis of VNF life cycle management so as to extract a set of generic and abstract VNFM Integration Interfaces to complete the VNF life cycle management.

IV. VNF Life Cycle Management tasks analysis and VNMF Integration Interface List

Fig. 2 illustrates the tasks generated from VNF life cycle management use cases. As described in the diagram, the VNF LCM tasks are arranged into three layers. The solid line squares represent the highest VNF level tasks corresponding to VNF LCM use cases. The ovals represent a set of self-contained and platform-independent tasks that need to be completed to realize the high level VNF LCM tasks. The dotted line squares are platform-dependent operations with the smallest granularity that need to be executed in the cloud, VNFs, and EMs for VNF life cycle management.

For example, the VNF instantiation use case can be broken down into VNFC instantiation, link creation, and external network connectivity creation tasks. These tasks are abstract, self-contained, and platform-independent. The VNFC instantiation can be further broken down into virtual instance creation and configuration during creation time; VNFC link creation can be decomposed into internal virtual network creation, router creation and interface attachment to the router, virtual instance configuration, and internal virtual network connectivity testing. These operations need to be completed in VIMs and, hence, are platform-dependent. In Fig. 2, the dotted line squares representing those platform-dependent tasks are based on the OpenStack context. These operations could be different in the context of other VIMs.

When implementing platform-dependent operations, we would need to use different APIs exposed from different cloud providers. For example, if the virtual infrastructure is an OpenStack managed cloud system, we will use the OpenStack RESTful API [6] to launch and configure a VM instance in the OpenStack cloud. If it is a vSphere cloud environment, we will use the vSphere SDK [7] to create and configure the VM instance in the vSphere cloud.

However, the oval tasks in Fig. 2 stay the same, irrespective of underlying virtual infrastructures. This set of tasks contains all that VNFM would need in order to complete the life cycle management of a VNF. If we have a set of APIs abstracting the functionality of these tasks while hiding the implementation details in various clouds and systems, we will be able to satisfy the need for VNFM to complete the VNF life cycle management.

Table I lists the interfaces that need to be included in the VNFM Integration Interface. Following RESTful API design, this set of interfaces identifies the resources that need to be exposed from the interface and applies Create, Read (Query), Update, and Delete (CRUD) operations on the resources. This APIs contains the interfaces for VNF life cycle management including VNF on-boarding, instantiation, configuration, scaling, healing, and termination. It also contains a Notify API to open the notification channel between VNFM and the external systems, which allows VNFM to update VNF state in the state machine based on the notification sent from the cloud adaptors.

Although monitoring is part of NFV MANO functionalities, in reality, a separate tool is usually used for NFV monitoring. In the next section, we will discuss Dell Foglight monitoring system as a tool for NFV monitoring, as well as the reasons for us to choose this tool for the proof of concepts of the proposed software architecture and the VNFM integration interface.

V. Using Dell Foglight Monitoring System for NFV Monitoring

In order to demonstrate VNF auto scaling as one of the advanced VNF life cycle management functionalities, we need to monitor the VNF and auto-scale it when a certain condition is satisfied. Lacking existing monitoring tools designed for visualized performance and fault management of VNFs, we would need to either develop a new tool from scratch or customize an existing tool. To reduce the development time, we chose the second option. While looking for an existing monitoring tool for our POC, we had the following design considerations in mind:

- A tool already has the virtualization and network monitoring capability.
- A tool can be extended easily to add NFV VNF monitoring capability.
- A tool has comprehensive rule engine to support a script-driven rule definition.
- A tool has the rule action extension which could be extended to trigger external actions.
- A framework to support customized user interface dashboard.

Resource	Operations	Description
VNFC Image	C, D	Create or delete the image associated with the VNFC in or from the VIM.
VNFC	C, R, D	Create, delete, or query the virtual instance associated with VNFC.
VNFC Link	C, D	Create, delete, query, or test the network connectivity between the two VNFCs.
VNFCPublicAccess	C, R, D	Create, delete, query, or test the external public access of the VNFC instance.
VNFCAdmin		With different parameters, start, stop, suspend, resume, or reboot the VNFC instance.
VNFCSaling		With different parameters, scales up, down, in, and out the VNFCs.
VNFCHealing		With different parameters, applies different approaches, such as fail-over , for VNFCHealing.
VNFConfig		With different parameters, connects to either EM or VNF control instance for VNF configuration.
Notify		Notify the subscriber with an object that contains the low level messages related to VNF lifecycle state change.

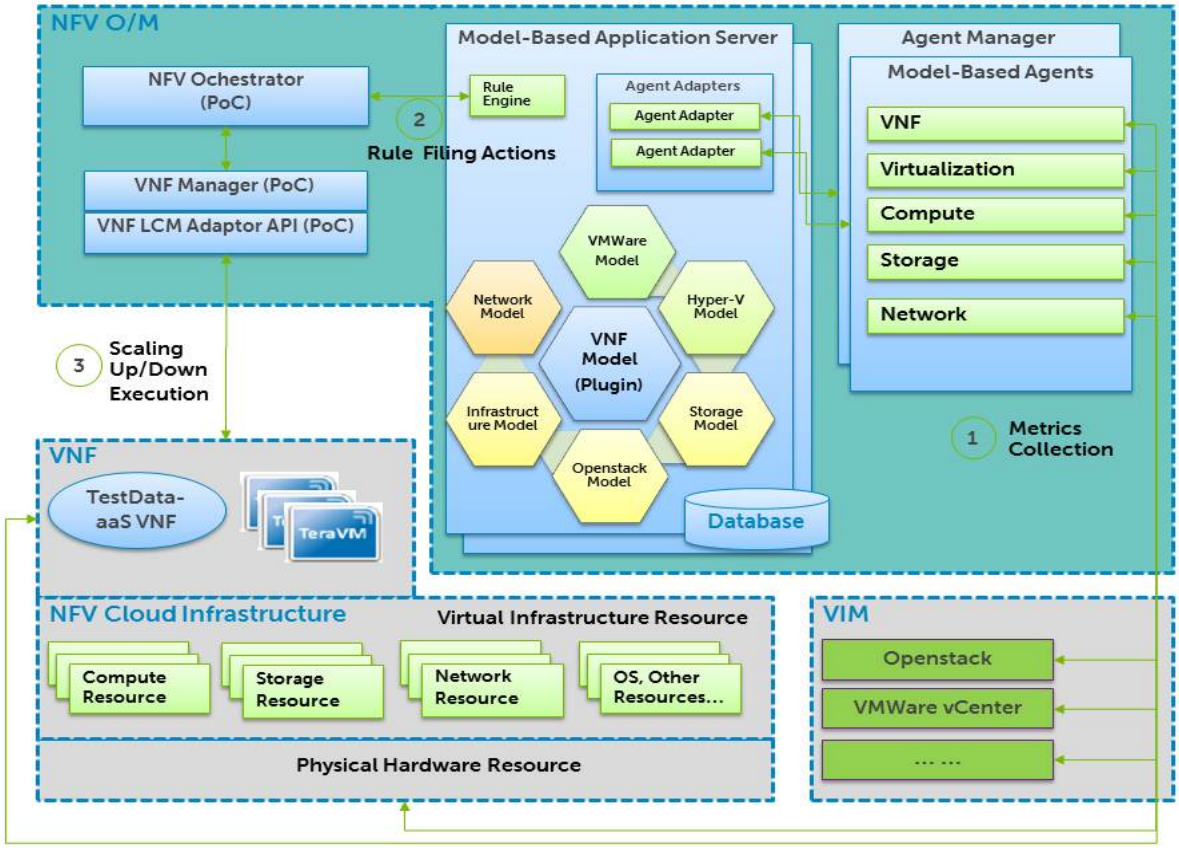

Fig. 3. Using Foglight in VNF auto scaling use case

With model-based system support, flexible extensions on both UI and agent, and a comprehensive rule engine script definition mechanism, the Dell Foglight Monitoring System has all the necessary elements for our advanced VNF lifecycle management functionality POC.

More specifically, Foglight is a model-based system with various out-of-box data models available in both physical and virtualized infrastructures. This includes data models to support VMware, OpenStack, servers, storage, and networking. For detailed information, please refer to Foglight eDoc [10]. In our POC, by adding a new data model for VNF Components, and integrating with the existing data models through object relationships to form a complete topology model across multiple layers, we presented a consolidated performance and fault management view in the NFV architecture.

Foglight has comprehensive rule engine support including multiple rule types, trigger types, and actions. Foglight supports both simple rules and multiple-severity rules, where simple rules do not generate alarms and multiple-severity rules generate alarms when the condition associated with any one of the severity levels is met. Foglight also invokes actions based on different trigger types including data-, time-, schedule-, and event-driven triggers. Foglight rule engine triggers different types of actions including SNMP traps, emails, commands, and scripts. In our POC, to auto-scale the VNF deployed in the cloud, we used simple rules with data-driven triggers to invoke remote command actions implemented using an external API. The choices we made were based on both rapid prototyping purposes as well as wanting to create a standalone component which could be used for NFV without binding to any particular monitoring system.

Foglight User Interface supports a broad range of customization using Web Component Framework (WCF). WCF framework supports drag and drop rapid web development as well as deep UI customization using the framework APIs. In the POC, we fully utilized Foglight WCM capabilities for the NFV monitor UI development with reduced time and efforts.

In summary, as described in Fig. 3, the NFV Monitor POC based on Foglight first periodically collects the corresponding metrics from the VNF deployed in the cloud. Then it checks the defined rule against the collected metrics values; when the specified condition is met, an action will be fired from the rule engine. Finally, the scale up/down action will be executed to automatically scale the VNF deployed in the cloud.

VI. Prototype and Result Validation

To serve as a proof of concept of the proposed software architecture and the VNFM integration interface, we created a prototype to automatically deploy and scale a sample VNF. Fig. 4 describes the architecture of the POC, which follows the ETSI NFV-MANO architectural framework.

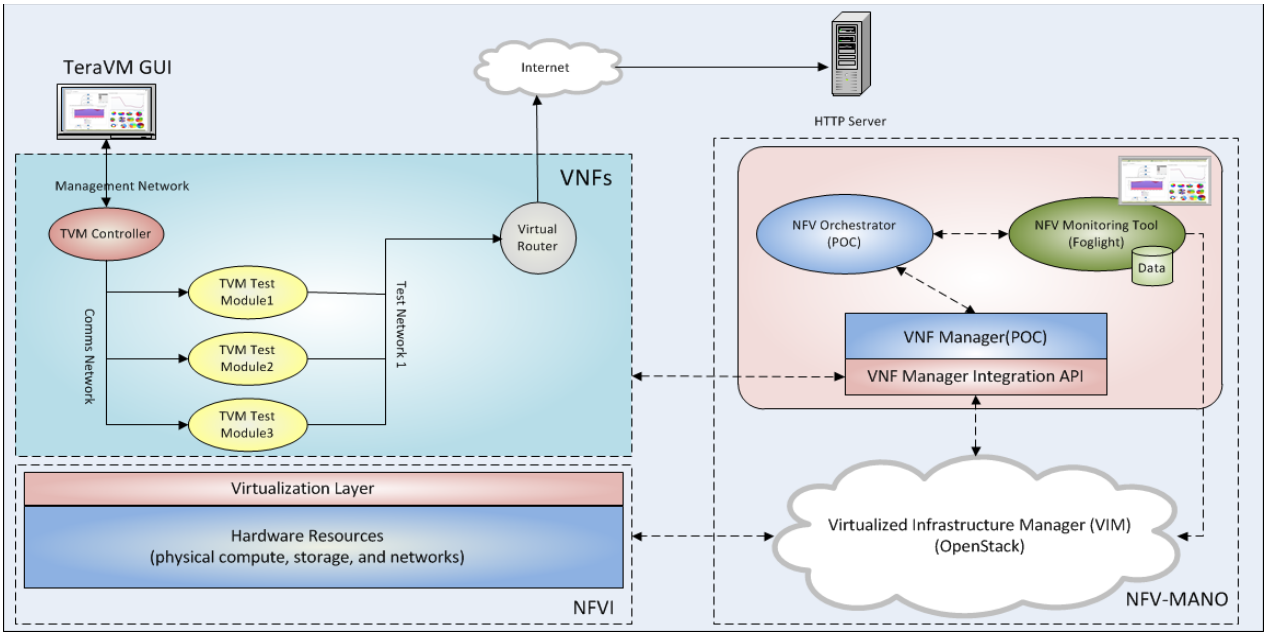

Fig. 4. POC to automatically deploy and scale TeraVM in OpenStack

The upper left of the diagram shows the sample VNF that we used in the prototype, which is called TeraVM [9]. This VNF was a fully virtualized IP application emulation, network traffic generation, and performance measurement tool. From the prototyping point of view, TeraVM presented typical and realistic VNF deployment scenarios. TeraVM contains two different types of VNFCs: TVM-Controller and TVM-TestModule. Various operations are needed during the life cycle management of the VNF. This includes image uploading, virtual instance creation in the cloud, VNFC link creation, public access creation for TeraVM GUI access, and controller instance configuration. TeraVM could also scale both vertically and horizontally to generate the test traffic for load testing of the network services. All these features help to construct a prototype to go through a complete life cycle management of a VNF.

The lower left of the diagram shows the hardware and virtualization layer providing the functionality of NFVI. In the prototype, we used four Dell 620 Blade servers, one Dell Force 10 switch s4810, and one Dell EqualLogic™ Blade Store PS-M4110 as the shared storage. On top of the hardware resources, we installed OpenStack with one controller node and three compute nodes to create a virtualization layer.

The lower right of the diagram shows OpenStack as the VIM that we used for managing the NFVI. We chose OpenStack because of its openness in architecture, design, implementation, source code, and interfaces for open integration.

On the upper right side, a prototype NFVO, VNFM, and NFV Monitor emulates NFVO and VNFM functionalities. Python and Ansible were chosen to implement the NFVO and VNFM POC including the VNFM Integration API and the OpenStack adaptors to achieve the functionalities of the VNFM. Ansible has gained popularity in IT automation because of its simplicity, ease-of-use, and agent-less machine management capabilities. In our prototype development, we found Ansible support of automation on OpenStack was complete enough to support VNF life cycle management functionalities. Compared to other automation tools, Ansible was easier to use and required shorter development cycles. On the NFVO, VNFM, and Integration API level, we chose Python because of its high-level programming capability and the full-fledged support of RESTful APIs. For the NFV monitor, we chose Dell Foglight for prototyping with the reasons that we have discussed in the previous section.

At VNF deployment time, the NFVO instructs VNFM to send requests to OpenStack and the TeraVM controller through a VNFM Integration API for life cycle management operations. The corresponding adaptors for VNFMI translate and convert the abstract requests into operations consumable by the OpenStack and the TeraVM controller. At runtime, Dell Foglight monitors the VNF's performance and health status. When Foglight detects the VNF is under heavy load, it instructs NFVO to auto scale the VNF through VNFM Integration Interface.

As a result of the prototype, we were able to automatically deploy TeraVMs with one TVM-Controller and three TVM-TestModules on an OpenStack managed cloud. The manual deployment of the TeraVM usually takes many hours up to a full day. In our POC, we were able to on-board the VNF, instantiate the VNFCs, configure the VNF, and start the VNF in under two minutes. We were also able to use Dell Foglight to monitor the performance of the VNF and auto-scale the VNF by adding a new VNFC instance when the VNF is under heavy load.

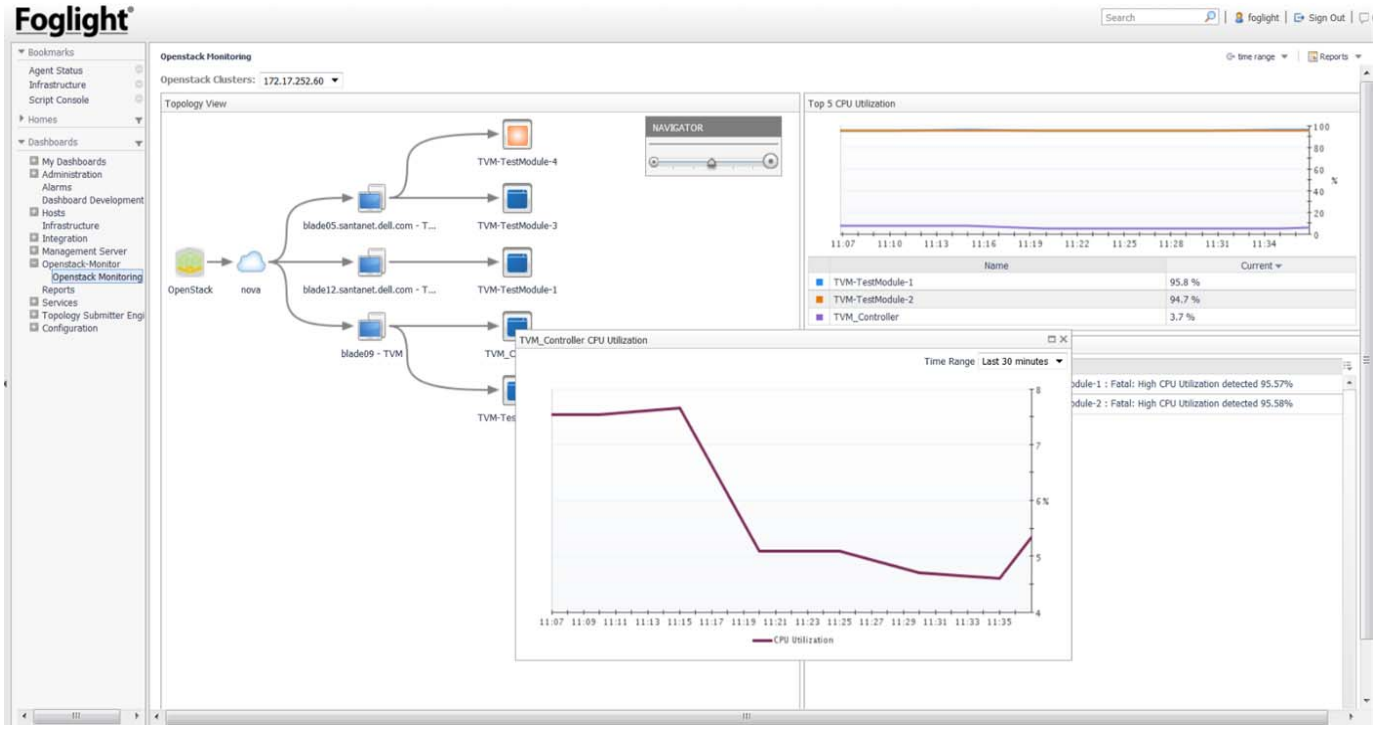

Fig. 5. VNF auto scaling using Dell Foglight with VNFM Integration Interface

The screenshot in Fig. 5 shows the Foglight OpenStack monitoring dashboard view of the VNF after being automatically deployed and scaled using the VNFM Integration Interface, triggered by the prototyped NFVO and VNFM working together with the Foglight rule engine. The highlighted TVM-TestModule-4 was the newly-created test module as a result of auto-scaling the VNF.

VII. Summary and Future Work

In this article, we refined and developed functional blocks in the ETSI reference architecture in order to help with product development, following ETSI published specifications. With our prototype experience, we proposed a VNFM Integration API at the VNFM southbound to integrate with multiple external systems to complete the life cycle management of the VNFs. Through prototyping, we also evaluated and concluded that Dell Foglight was a nice fit for realizing NFV monitoring functionalities.

In summary, we believe the architecture, design, interfaces, and monitoring approaches we have presented, together with the experiences gained from prototyping with various tools and systems, would be a good foundation for the further development of a full-fledged NFVO-MANO system.

Further POC work related to the proposed refined software architecture could be spent in interfaces other than the VNFM Integration Interface. This includes NFVO Northbound Interface and VNF Manager Interface. Further prototyping and exploration effort could also be spent in functionalities other than VNF Life Cycle Management. This includes NSD/VNFD parsing and processing, global resource management, and VNF Forwarding Graph configuration in service chaining scenarios.

More importantly, based on the fact that the robustness and reliability is critical in a carrier grade NFV system, future research and studies should focus on the performance of auto-scaling and self-healing of the Network Services composed of VNFs deployed in the cloud. With model-based architecture to support the monitoring of VNFs across different layers in the NFV architecture, further research and studies into methods and technologies to achieve the quick detection of potential fault and performance issues that have impact on the network services from every perspective of the architecture, and recover the network services from the potential problems in real-time, would be important.

Acknowledgment

We would like to thank Jai Menon and Gabby Silberman for their most support and encouragement on both the research as well as the paper. We are also immensely grateful to Cheryl Morris for her comments and editorial help in both the earlier and the final versions of the paper.

REFERENCES

[1] European Telecommunications Standards Institute (ETSI) White Paper, "Network Function Virtualization, An Introduction, Benefits, Enablers, Challenges & Call for Action", 2001. Available from http://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[2] Clayman S., Maini E., Galis A., Manzalini A., Mazzocca N. "The dynamic placement of virtual network functions", in IEEE NOMS, Krakow, Poland, 2014, pp 1-9.

[3] European Telecommunications Standards Institute (ETSI) Group Specification, "Network Function Virtualization Management and Orchestration Group Specification", 2014. Available from http://docbox.etsi.org/ISG/NFV/Open/Latest_Drafts/NFV-MAN001v061-%20management%20and%20orchestration.pdf.

[4] Caroline Chappell, "Decoding the Management and Orchestration Architecture for Network Functions Virtualization", 2014. Available from http://www.cyaninc.com/assets/docs/whitepapers/HR_Cyan_NFV_MANO_WP_2-18-14_-_FINAL.pdf.

[5] Overture Networks Inc., Research Triangle Park, NC 27560, USA, "Ensemble Service Orchestrator (ESO), Multi-Vendor NFV Service Management and Orchestration System", Available from http://www.overturenetworks.com/sites/default/files/data_sheet/Overture_EnsembleServiceOrchestration_DS.pdf.

[6] OpenStack Foundation, "OpenStack API Complete Reference", September 2014, by OpenStack. Available from http://developer.openstack.org/api-ref-guides/bk-api-ref.pdf

[7] VMWare Inc., Palo Alto, CA 94304, "vSphere Web Services SDK Programming Guide", Available from http://pubs.vmware.com/vsphere-51/topic/com.vmware.ICbase/PDF/wssdk_prog_guide_5_1.pdf.

[8] Lopez, D.R., "Network function virtualisation : Beyond carrier-grade clouds", OFC, San Francisco, CA, March 2014, pp 1-18.

[9] Aeroflex Inc., PlainView, NY 11803, USA, "TeraVM – A Fully Virtualized IP Test and Measurement Solution". Available from http://ats.aeroflex.com/virtualized-ip-test-solutions/teravm.

[10] Dell Inc., Round Rock, TX 78682, "Foglight eDoc". Available from http://edocs.quest.com/.