Auto-Scaling in NFV Using Tacker

Conference Paper · January 2017

CITATIONS
0

READS
2,084

3 authors, including:

Emanuel Coutinho
Universidade Federal do Ceará
114 PUBLICATIONS **313** CITATIONS
SEE PROFILE

Jose Souza
Universidade Federal do Ceará
160 PUBLICATIONS **830** CITATIONS
SEE PROFILE

Auto-Scaling in NFV Using Tacker

William Sales[14], Emanuel Coutinho[24], and José Neuman de Souza[134]

[1] Master and Doctorate in Computer Science (MDCC)
william@lia.ufc.br
[2] Virtual University Institute (UFC-VIRTUAL)
emanuel@virtual.ufc.br
[3] Department of Computer Science (DC)
neuman@ufc.br
[4] Federal University of Ceará (UFC) – Fortaleza – Ceará – Brazil

Abstract

Nowadays, computer networks are composed of hardware appliances with specific functions, such as firewalls, load balancers, web proxies and web servers. These appliances are called middleboxes, and generally their implementation is a difficult task because of its proprietary nature. In these appliances, Network Function are highly coupled to the underlying hardware, ensuring high performance and reliability Network Function Virtualization (NFV) emerges as a proposal to ease the provision of services, proposing a decoupling of the network functions from the underlying hardware, deploying them on commodity hardware (or off-the-shelf computing equipment), allowing Virtual Network Functions can be deployed on commodity hardware directly, or through a Virtual Machine. One of the greatest benefits obtained by NFV approach is scaling, which is the ability to dynamically extend or reduce resources allocated to the VNF as needed and at run-time. In this work, we are interested in the advantages and peculiarities of automatic scaling out/in, and for that we will use auto-scaling functionality available in Tacker.

1 Introduction

Currently, computer networks are composed of hardware appliances with specific functions, such as firewalls, load balancers, web proxies and web servers. Generally, the implementation of these appliances is a difficult task because of its proprietary nature and the lack of specialized professionals to integrate them and keep them. These appliances are also called middleboxes.

In these appliances, Network Function (NF) are highly coupled to the underlying hardware, ensuring high performance and reliability, at the cost of a low flexibility of network infrastructure. Thus, even a simple change of location of a middlebox in the network becomes a difficult and complex task.

Given this rigidity in adapting the network, the Network Function Virtualization (NFV) emerges as a proposal to ease the provision of services. In general, NFV proposes a decoupling of the network functions from the underlying hardware, deploying them on commodity hardware [1]. In addition, the Virtual Network Function (VNF) can be deployed on commodity hardware directly, or through a Virtual Machine (VM).

One of the greatest benefits obtained by NFV approach is Scaling, which is the ability to dynamically extend or reduce resources allocated to the VNF as needed and at run-time. Scaling may arise from the need of Scaling-out the VNF to handle added load or scaling-in to to release unused resources.

In this context, Auto-scaling is ability of automatic scaling of VNFs and it provides better resource utilization at a lower cost. Thus, it is possible quickly create new instances of a specific

[1] Commodity hardware are servers, storage and switches with standard specifications from the industry.

VNF that is overloaded, in order to split the workload. In the same way, it is possible to destroy instances that are idle. The lower cost is due to the use of general purpose servers, which are lower than the middleboxes.

In addition, there is a reduction in time to-market of new network services, since it is not necessary to design and manufacture specific hardware. In addition, there is also a reduction in Operation Expenditure (OPEX), since it is not necessary to have specialist professionals in the various existing middleboxes, resulting in the same operational procedures for all appliances. Also noteworthy is the economy resulting from the lower consumption of electricity, achieved through the migration of NFVs and the shutdown of unused hardware.

Among the solutions that implement the NFV-MANO (Management and Orchestration) specification, Tacker is the one that provides the most functionality contained in the specification, so it will be used in this work [11]. However, few papers present a case study using the auto-scaling feature.

Thus, the aim of this work is to try to use the auto-scaling functionality available in Tacker, in order to verify the viability of the solution.

2 NFV and Auto-Scaling

2.1 NFV

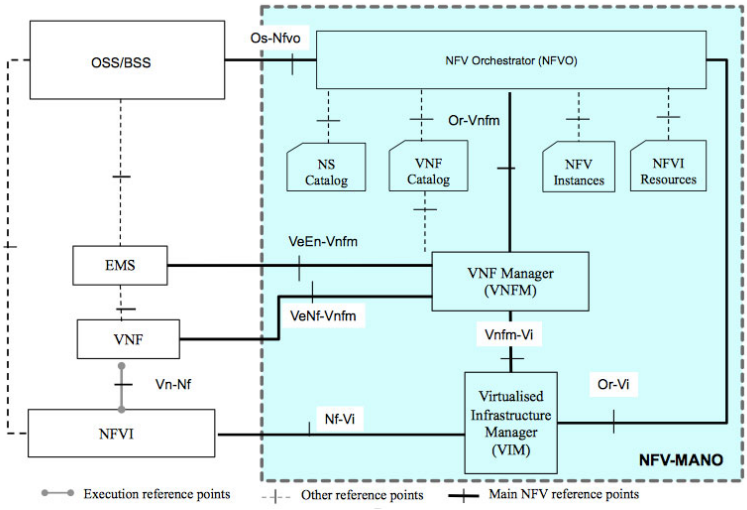

Figure 1: NFV MANO architecture [5]

With the introduction of VNFs, NFV imposes the need to add new features of management and orchestration to the model of operations, administration, maintenance and provisioning [10] of existing communications networks [7]. To meet this demand, NFV proposes the NFV-MANO framework (NFV Management and Orchestration), responsible for managing and orchestrating VNFs in order to provide a network service that meets the requirements of users. NFV-MANO is composed of the three function blocks: NFV Orchestrator (NFVO), VNF Manager (VNFM) and Virtualized Infrastructure Manager (VIM) [5]. Figure 1 shows functional blocks and contact points between them.

The NFVO has two main features: the Network Service Orchestration and Resources Orchestration. The first aims to coordinate groups of VNFs instances, resulting in network services that perform more complex functions. Moreover, it performs: manage the lifecycle of Network Services instances; instantiate and manage the VNFMs; maintenance of information about the relationship between Network Services instances and VNFs that compose it; configure the connections between VNFs; and dynamic management of configuration changes, such as resizing the capacity of the Network Service. In turn, the Resources Orchestration has the objective to orchestrate resource of NFV Infrastructure (NFVI) through multiple VIMs, as discussed later.

The VNFM, in turn, is responsible for managing the VNFs instances lifecycle, and can instantiate, update, query, resize (scaling out/in and up/down) and destroy VNFs. Additionally, it can perform automatic or assisted recovery of instances VNFs and collect information about performance and failures of VNFs. Each VNFM can manage one or more VNFs. The requirements for deploying and operating behavior of each VNF is represented by a template called VNF Descriptor (VNFD), which is stored in catalog of VNFs. The NFV-MANO architecture uses VNFD to create instances of VNFs and to manage the lifecycles of them. The resources of NFVI are assigned to a VNF based on the requirements stated in VNFD in addition to considering specific requirements, restrictions and policies which may overlap the VNFD.

VNFM uses the Virtualization Deployment Unit (VDU) as a abstraction to describe the deployment and operational behaviour of a subset of a VNF, or the entire VNF if it was not componentized in subset. Thus, it is possible define the hardware and software requirements necessary to deploy the VNF.

VIM is responsible for controlling and managing computing, storage and network resources, and it is usually located in a single infrastructure domain. Generally, VIM interacts with a variety of hypervisors and network controllers belonging to NFVI in order to achieve the functionality exposed by other functional blocks of NFV-MANO architecture. Others features include: orchestrate, optimally, the allocation, update, release and recovery of NFVI resources; manage the association (mapping) of virtualized resources to physical computing, storage and network resources; manage, through repositories, information about the capabilities and features of hardware (computing, storage and network) and software (hypervisors) resources of NFVI; manage the ability of virtualized resources and forwarding information related to NFVI resource capacity and usage reports; manage software images (add, delete, update, query, copy), as requested by other functional blocks of the NFV-MANO architecture, for example, the NFVO; collect and make available information on performance and failures of NFVI resources, either hardware (computing, storage and network), software (hypervisors) or virtualized resources (VMs), and provides analysis of the cause of NFVI performance issues; support the management of graphs NFV Routing (creation, query, update and delete).

2.2 Auto-Scaling

Scaling can be initiated in a number of ways, or by a manager who needs to increase the capacity of the VNF in advance due to a forecast of future demand growth or by a management system that needs to maintain the requirements of the Level Agreement Service (SLA) [9].

Scaling can be classified in scaling up/down and scaling out/in. Scaling up/down, also called Vertical Scaling, refers to the ability to add or remove allocated resources for existing computing nodes, such as memory, CPU capacity or storage. In turn, scaling out/in, or Horizontal Scaling, refers to the ability to scale by add/remove instances, such as VMs. Through scale out/in you can instantiate new VNFs quickly, providing a way to create and deliver new services quickly. In this work, we are interested in the advantages and peculiarities of scaling out/in.

Auto-scaling is a cloud computing service feature that automatically adds or removes compute resources depending upon actual usage. Thus, Auto-scaling can automatically increase the number of VNF instances during peak demand to maintain performance, and decrease capacity during idle periods to reduce costs. It helps you ensure that you have the correct number of VNF instances available to handle the load.

Also, in Cloud Computing, there is the concept of elasticity. According to the National Institute of Standards and Technology (NIST) definition, resources can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand [6]. Thus, elasticity is becoming a growing need due to the dynamic nature of different applications and diverse workloads. A detailed study about cloud elasticity were described in [3], such as: definitions, metrics, benchmarks, elastic strategies, challenges and trends in the construction of elastic solutions.

3 Literature Review

Several implementations of the NFV specification are available [4]. The following are the most important open source NFV projects available:

Open Platform for NFV (OPNFV) [8] is an open source project hosted by The Linux Foundation, that aims to help the evolution of NFV. The first version, released in June 2015 and named OPNFV Arno, initially focused on building NFV Infrastructure (NFVI) and Virtualized Infrastructure Management (VIM) by integrating projects such as OpenDaylight, ONOS, OpenStack, Ceph Storage, KVM, Open vSwitch, DPDK and Linux. The most recent releases, called Brahmaputra e Colorado, includes Management and Network Orchestration (MANO) components primarily for application composition and management. From Colorado release, OPNFV supports Tracker.

OpenMANO is a open source project released by Telefonica and aimed at implementing ETSIs NFV MANO framework. It is composed of OpenVIM (a lightweight, reference implementation of an NFV VIM), Openmano (an Manage and Orchestrator) and a graphical user interface (GUI). OpenMANO can also work with OpenStack as VIM in its architecture. OpenVIM is very similar to OpenStack, interacting with the compute nodes in the NFV Infrastructure and an OpenFlow controller to provide computing and networking capabilities and to deploy virtual machines.

Open Source MANO (OSM) is a open source project hosted by ETSI to develop an NFV Management and Orchestration (MANO) software stack aligned with ETSI NFV architecture, and focused on helping industry accelerate the implementation of network virtualization. OSM makes use of various projects, such as OpenMANO (acting as NFVM e VIM), RIFT.io (performing the function of GUI and NFVO), OpenStack (for VIM) and Ubuntu JuJu (for VNF Configuration and Abstraction). The only release available is called OSM Release ONE.

OPEN-O is an open source project hosted by the Linux Foundation that focused on enabling end-to-end service orchestration across network functions virtualization (NFV), software-defined networking (SDN), and legacy networks. The current release is called OPEN-O Release 1.0.

4 Tacker Architecture

Tacker is an OpenStack project for NFV Management and Orchestration to deploy and operate Virtual Network Functions (VNFs) and Network Services (NS) on an NFV Platform. It is based

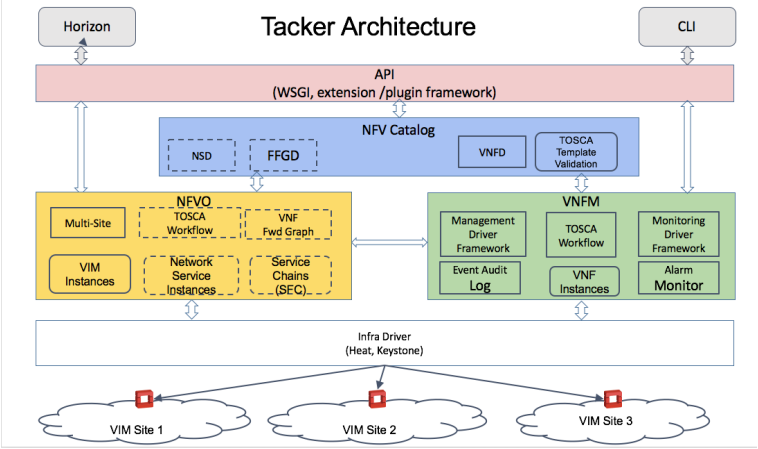

Figure 2: Tacker architecture [1]

on ETSI MANO Architectural Framework [11].

Figure 2 presents Tacker Architecture, which provides a specific API for lifecycle management of VNFs along with capabilities like VNF monitoring, auto-scaling and self-healing. The NFV Catalog stores the VNF Descriptor (VNFD), Network Service Descriptor (NSD), Forwarding Graph Descriptor (VNFFGD). Tacker uses Openstack as Virtualized Infrastructure Manager (VIM), specifically Nova, Neutron, Heat and Keystone services.

To define VNFD and NSD, Tacker uses Topology and Orchestration Specification for Cloud Applications (TOSCA), a OASIS standard, that is a declarative language to describe, in a standardized way, the application topology for a network or cloud environment that includes all its components. TOSCA specifications to describe processes that create or modify web services.

To provide a visual means to handle VNFs, the Tacker project has altered the Openstack Horizont service, so that interactions with VNFs are performed easily through dashboard. Figure 3 shown, on the left side, the menus added to Openstack, "VNF Management" and "NFV Orchestration".

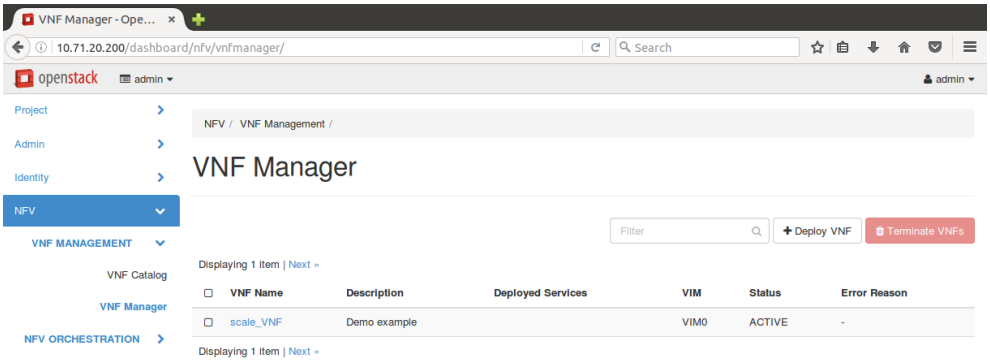

Figure 3: Openstack dashboard

5 Experiment Environment

The purpose of the experiment is to show the auto-scaling functionality of VNFs using Tacker. For this, VNFs will be deployed and subjected to a high processing load, so that Tacker automatically creates new instances to distribute the workload.

For the experiment, a scaling policy was defined to scale out the number of VNF instances when CPU or RAM usage reaches 70% of total availability from VM. The initial number of instances is 2, and can vary up to the maximum limit of 3 and minimum limit of 1. The variables CPU processing load and use of RAM Memory were easily retrieved by the Tacker Monitoring Framework in conjunction with the Ceilometer. The environment used to carry out the experiment consists of one Server HP ProLiant ML350p Gen8 (Intel Xeon E5-2620 Processor and 8 GB RAM Memory), a workstation with an i5 Intel processor (4 GB RAM Memory), interconnected by an 8-Port Gigabit Ethernet Mini-Switch, as shown in the Figure 4.

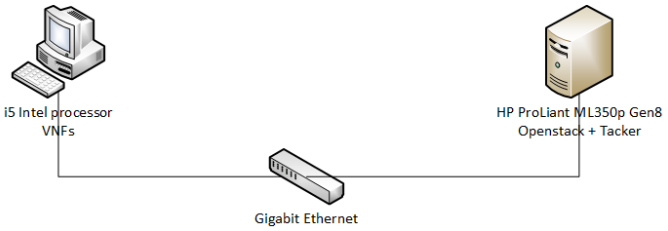

Figure 4: Experiment environment

The server hosts Openstack and Tacker and performs the functions of VIM, NFV Management and Orchestration, while the workstation serves as a Compute Node that hosts the VNFs. The installation of Openstack and Tracker on the server followed the steps described on the Tacker project website [2]. In the workstation the Openstack nova-compute service was installed, based on the instructions on the Openstack website [3].

Each NFV instance consists of only one VDU, in order to simplify the environment, and it has a Cirros operating system, that is a minimal Linux distribution that was designed for use as a test image on clouds such as OpenStack Compute. Each VDU is composed of 1 CPU, 1 HD of 1 GB and 512 MB of RAM Memory.

To perform the experiment, an HTTP server was enabled on the VNFs and the Load Balancing service of the Neutron Openstack (LBaaS) was used to stress the VNFs.

6 Experiments Results

As shown in Table 1, at the beginning of the experiment, the use of the CPU of Instance 1 was at 10% and Instance 2 was at 16%. After the stress tests, CPU usage of Instance 2 reached 70%, triggering alarm triggers for the Tacker Monitoring Framework, which, in turn, initiated the instantiation of a new VNF. Figure 5 shows the exact moment the NFV was performing the scaling out, represented by the PENDING_SCALE_OUT state.

[2]Tacker installation - https://wiki.openstack.org/wiki/Tacker/Installation

[3]Compute Node Installation - http://docs.openstack.org/juno/install-guide/install/apt/content/ch_nova.html

Table 1: CPU Usage in Instance 1 (I1), Instance 2 (I2) and Instance 3 (I3)

	#VNF	% CPU I1	% CPU I2	% CPU I3
Beginning of the experiment	2	10%	16%	-
After stress and just before Scale out	2	64%	70%	-
After Scale Out	3	46%	52%	44%

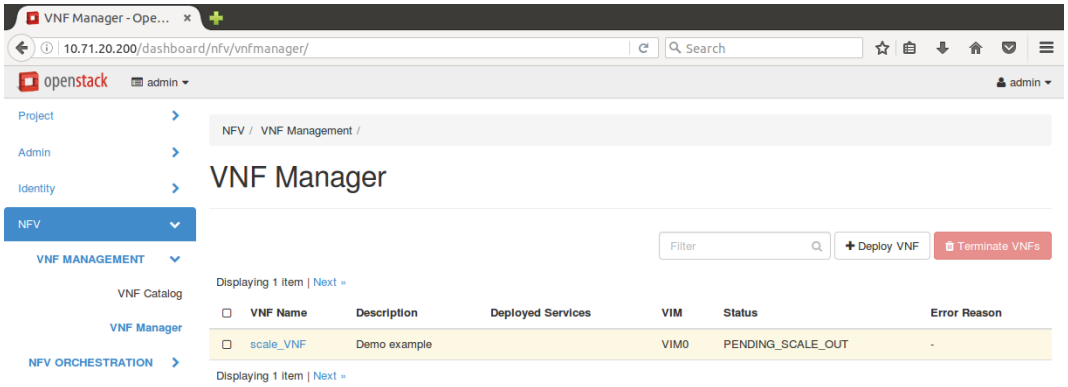

Figure 5: PENDING_SCALE_OUT status after triggering alarm

After the instantiation of the new VNF, the load can be distributed between the 3 instances, which will cause a decrease in the CPU usage of instance 1 and 2. Figure 6 shows the newly created instance, with IP address 192.168.120.3 and "Time since created" equal to zero minutes.

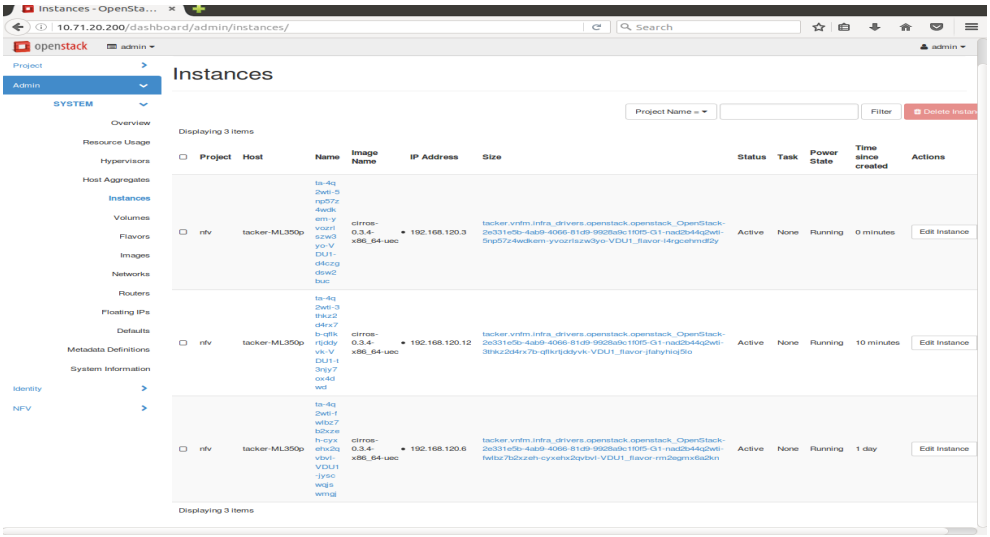

Figure 6: Three NFV instances

7 Conclusion

This work presented an experiment with Tacker, an OpenStack solution for NFV. In it, it was possible to check the running auto-scaling in order to achieve automatic scaling in/out. In this way, VNFs with HTTP proxy functions were deployed and subjected to a high processing load, so that Tacker automatically creates new instances to distribute the workload.

For the scaling in/out policy only the processing load variable was used. However, in future work other variables, such as the use of RAM, could be considered in the policy definition. Besides that, we intend to design larger experiments, with more VNFs, and collect metrics to improve performance. A strategy that can be used is to calculate how much the environment is elastic in a computational cloud environment, by means of specific elastic metrics [2].

References

[1] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba. nf.io: A file system abstraction for nfv orchestration. pages 135–141, Nov 2015.

[2] Emanuel F. Coutinho, Paulo A. L. Rego, Danielo G. Gomes, and Jos N. de Souza. Physics and microeconomics-based metrics for evaluating cloud computing elasticity. *Journal of Network and Computer Applications*, 63:159 – 172, 2016.

[3] Emanuel Ferreira Coutinho, Flávio Rubens de Carvalho Sousa, Paulo Antonio Leal Rego, Danielo Gonćalves Gomes, and José Neuman de Souza. Elasticity in cloud computing: a survey. *annals of telecommunications - annales des télécommunications*, 70(7-8):289–309, 2015.

[4] ETSI. Etsi industry specification group (isg) nfv, etsi gs nfv 001 v1.1.1: Network function virtualization. use cases. http://www.etsi.org/deliver/etsi_gs/nfv/001_099/001/01.01.01_60/gs_nfv001v010101p.pdf, 2013. Online; acessado em novembro-2016.

[5] ETSI. Etsi industry specification group (isg); network functions virtualisation (nfv); management and orchestration. http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf, 2014. Online; acessado em novembro-2016.

[6] Peter Mell and Tim Grance. The nist definition of cloud computing. *National Institute of Standards and Technology*, 53(6):50, 2009.

[7] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network function virtualization: State-of-the-art and research challenges. *Communications Surveys Tutorials, IEEE*, 18(1):236–262, Firstquarter 2016.

[8] OPNFV. Open platform for nfv (opnfv). https://www.opnfv.org/about, November 2016. accessed November, 2016).

[9] Shriram Rajagopalan, Dan Williams, Hani Jamjoom, and Andrew Warfield. Split/merge: System support for elastic execution in virtual middleboxes. nsdi'13, pages 227–240, Berkeley, CA, USA, 2013. USENIX Association.

[10] Brijendra Singh. *DATA COMMUNICATIONS AND COMPUTER NETWORKS Fourth Edition*. PHI Learning, 2014.

[11] Tacker. Tacker. http://docs.openstack.org/developer/tacker/, November 2016. accessed November, 2016).