논문 18-43-09-10 The Journal of Korean Institute of Communications and Information Sciences '18-09 Vol.43 No.09
https://doi.org/10.7840/kics.2018.43.9.1483

Tacker 기반 NFV 시스템의 워크플로우 정책 기반 VNF 모니터링 모델

응웬트리하이[◆], 유 명 식[°]

Workflow Policy-Based VNF Monitoring Model in Tacker-Based NFV System

Tri-Hai Nguyen[◆], Myungsik Yoo[°]

요 약

네트워크 기능 가상화(NFV)는 네트워크 인프라에 유연성 및 민첩성을 제공할 수 있는 새로운 네트워크 아키텍처 프레임 워크이다. 통신서비스 공급자 (CSP)는 NFV를 활용하여 기존 하드웨어 상에서 운용이 가능한 VNF (Virtual Network Function)이라는 소프트웨어 형태로 서비스 제공이 가능하다. 많은 장점에도 불구하고 VNF 모니터링에서의 확장성 문제는 반드시 다루어져야 하는 문제점이다. 현재 NFV 오픈소스 플랫폼에서 VNF Manager는 로컬 변수와 로컬 스레드를 사용하여 VNF 모니터링을 수행한다. 만약 시스템에서 많은 수의 VNF기 존재할 때 VNF Manger는 모니터링은 위하여 많은 스레드를 실행해야야 하여 확장성 문제가 발생한다. 이 문제를 해결하기 위하여 워크 플로우 정책기반 VNF 모니터링 메커니즘을 제안하고 실현 가능성 입증을 위해 OpenStack 클라우드 플랫폼의 오픈소스 NFV 오케스트레이션 및 관리 서비스인 Tacker에서 구현하였다. 제안된 모니터링 메커니즘으로 인하여 VNF Manager는 확장성 및 효율성 증대가 가능하다.

Key Words : NFV, Tacker, VNF Monitoring, Mistral, Workflow Service

ABSTRACT

Network Function Virtualization (NFV) is a new network architecture framework that can provide flexibility and agility in network infrastructure. By leveraging NFV, a communication service provider (CSP) can deploy network services as a software known as Virtual Network Function (VNF) running on commercial off-the-shelf hardware. Despite the multiple benefits that NFV provides, the scalability of VNF monitoring is one of the main challenges that must be addressed for the deployment of network services in the NFV. In the current NFV open-source platforms, VNF Manager uses a local variable and local threads to monitor the VNFs. If the system has many VNFs which need to be monitored, the VNF Manager server will have to run a lot of threads to monitor them. This causes the scalability problem in the VNF Manager server. To tackle this issue, a workflow policy-based VNF monitoring mechanism is proposed and implemented in Tacker to demonstrate the feasibility of the proposed mechanism. Thanks to the proposed monitoring system, the VNF Manager can be more scalable and effective.

※ This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2018-2017-0-01633) supervised by the IITP(Institute for Information & communications Technology Promotion).

◆ First Author : (ORCID:0000-0002-2132-2290)Department of ICMC Convergence Technology, Soongsil University, Seoul, Republic of Korea, nguyentrihai@soongsil.ac.kr, 학생회원
° Corresponding Author : (ORCID:0000-0002-5578-6931)School of Electronic Engineering, Soongsil University, Seoul, Republic of Korea, myoo@ssu.ac.kr, 정회원

논문번호 : 201809-273-D-RN, Received September 6, 2018; Revised September 17, 2018; Accepted September 17, 2018

Ⅰ. Introduction

To cope with the increasing network utilization driven by Internet-of-Things (IoT), and to satisfy the demand for new network services and performance guarantees, Network Function Virtualization (NFV)[1] was proposed to reduce cost and accelerate service deployment for the communication service providers (CSPs). Network Function Virtualization (NFV) is an emerging approach in which network functions are no longer executed by proprietary hardware appliances, but instead can run on generic-purpose servers as a software known as Virtual Network Functions (VNFs). As a result, it offers an opportunity to significantly increase the flexibility of infrastructure, simplify the resource management process, and decrease both capital and operational costs[2,3].

NFV architecture includes three major components: (i) Virtual Network Functions (VNFs) are software implementation versions of network functions; (ii) NFV Infrastructure (NFVI) contains the hardware and software components where VNFs are deployed; and (iii) NFV Management and Orchestration (MANO) includes NFV Orchestrator (NFVO), VNF Manager (VNFM), and Virtualized Infrastructure Manager (VIM) that orchestrate and manage the NFVI resources and lifecycle management of VNFs.

There are many open-source solutions for NFV[4]. The OpenStack cloud platform[5] and Tacker[6] projects are chosen among them. OpenStack is an open-source cloud computing platform that is identified as NFVI and VIM in the NFV architecture. Tacker is an OpenStack based NFV orchestration and VNF management service to deploy and operate Virtual Network Functions (VNFs) on an OpenStack based NFV Platform. It is based on ETSI MANO Architectural Framework and provides a functional stack to orchestrate VNFs end-to-end. Tacker is responsible for VNFM and NFVO in NFV architecture.

Despite the multiple benefits that NFV provides, some challenges must be considered and should be addressed for the deployment of network services. This research focuses on the challenges of scalable VNF monitoring in OpenStack Tacker-based NFV system. The Tacker server needs to monitor various status conditions of VNF entities that it deploys and manages. Currently, Tacker server uses a local variable and local threads to do the VNF monitoring. If the system has many VNFs which needs to be monitored, the Tacker server will have to run a lot of threads to monitor them. As a result, this causes scalability problem in Tacker server, and creates the huge impact on the application programming interface (API) function's performance. In this article, a workflow policy-based VNF monitoring model is proposed and implemented in Tacker. Recently, there is an OpenStack workflow service called Mistral[7], which is integrated as a part of Tacker system. Therefore, it is efficient when Tacker-based NFV system uses Mistral to monitor VNFs.

The remainder of this article is organized as follows. Section II shows the details of the proposed VNF monitoring mechanism. The experimental setup and results are shown in Section III. Finally, the conclusions are discussed in Section IV.

Ⅱ. A Workflow Policy-Based VNF Monitoring Model

In current model, Tacker hosts a local variable and local threads to monitor VNFs and directly update the status of VNFs to the database that stores VNF status information. Therefore, if many VNFs are deployed, Tacker needs many threads to do the monitoring which causes a huge impact on the performance of Tacker. Therefore, it brings significant benefits if we separate the monitoring of VNFs as a form of an independent task. As a result, it can improve the performance of API function and solve the scalability issue of VNF monitoring in Tacker VNFM.

Figure 1 illustrates the Mistral workflow policy-based VNF monitoring mechanism in Tacker. Mistral project aims to provide a mechanism to define tasks and workflows without writing code, manage and execute them in the cloud environment.

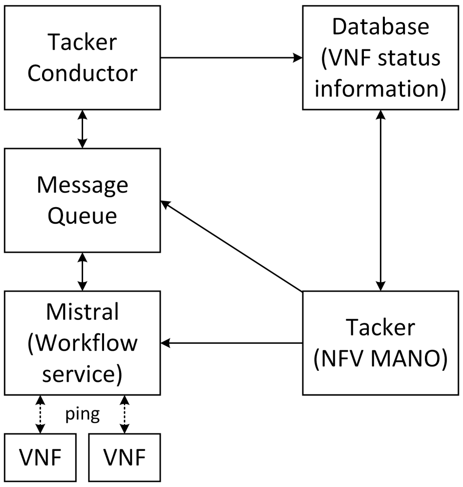


Fig. 1. A workflow policy-based VNF monitoring model in Tacker

In the proposed model, the Mistral workflow service handles the VNF monitoring task. Tacker will generate a VNF monitoring workflow and execute it via Workflow service if there is a VNF configured with monitor policy. The workflow and execution will be deleted once the monitored target VNF is removed. However, the workflow service cannot directly access to NFV database, hence, the Tacker Conductor is proposed to access database for the Mistral workflow service. When the workflow is removed, the Tacker will kill the workflow action via the message queue. The workflow service will use Remote Procedure Call (RPC) to communicate with Tacker Conductor server. To deal with the scalability of monitoring, multiple Tacker Conductors will be deployed.

Monitor policy is divided into two parts, that is, policy monitor and policy action. Policy monitor, that is, ping and HTTP-ping, is implemented as the VNF Policy Monitor in the workflow service. Policy action, for example, autoscaling, respawn, log, log-and-kill, will be called in Tacker Conductor. Each VNF with monitor policy will generate a monitor action ID with monitoring workflow stored as meta information of VNF instance, therefore, it can be easily managed.

There are three events that trigger this VNF monitoring mechanism, that is, VNF creation, VNF deletion, and VNF scaling. The VNF update does not trigger this VNF monitoring mechanism because

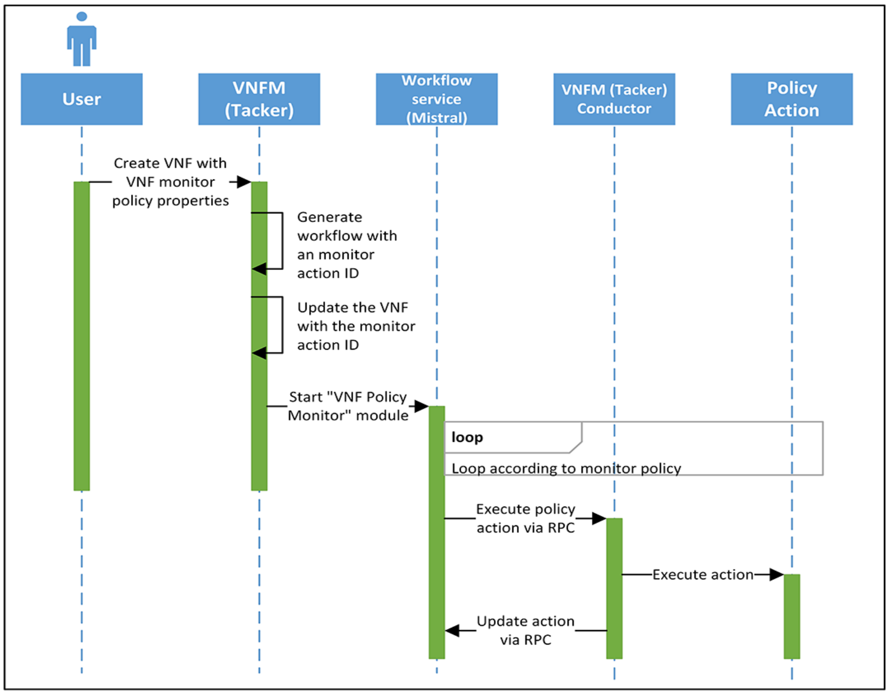

User
VNFM (Tacker)
Workflow service (Mistral)
VNFM (Tacker) Conductor
Policy Action
Create VNF with VNF monitor policy properties
Generate workflow with an monitor action ID
Update the VNF with the monitor action ID
Start "VNF Policy Monitor" module
loop
Loop according to monitor policy
Execute policy action via RPC
Execute action
Update action via RPC

Fig. 2. Sequence diagram for creating VNF in the proposed model.

it only configures the Virtual Deployment Units (VDUs) of VNF and does not affect the VNF health status. The following sequence diagrams show the procedure of creating VNF, deleting VNF and scaling VNF, respectively.

The sequence diagram for creating VNF is shown in Figure 2. After a user creates VNF with the VNF monitor policy, the Tacker generates monitoring workflow and corresponding monitor action ID, then it calls the VNF policy module in the Mistral workflow service via RPC channel. When policy action is needed, the VNF Policy Monitor module will call Tacker Conductor's execute-policy-action. Then, method execute-policy-action in Tacker Conductor will execute the predefined policy action such as respawn, log etc. If Tacker Conductor finds the action is obsolete, it will return bad-action update via RPC channel to the VNF Policy Monitor module, then the VNF Policy Monitor in Mistral workflow service will exit

Figure 3 illustrates the sequence diagram for deleting VNF. After the user sends the request to delete the VNF, the Tacker gets monitor action ID of the VNF and deletes VNF monitoring action workflow and its execution. Then, Tacker sends the kill action request to kill the VNF monitoring action in Mistral workflow service via RPC.

Figure 4 shows the sequence diagram for scaling VNF. When the VNF is scaling, the Tacker get the workflow information and monitor action ID of the

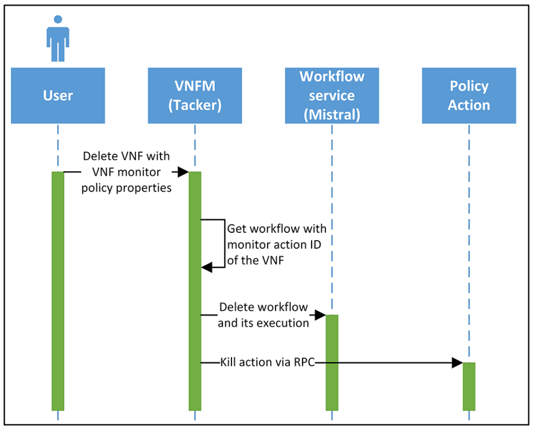

User
VNFM (Tacker)
Workflow service (Mistral)
Policy Action
Delete VNF with VNF monitor policy properties
Get workflow with monitor action ID of the VNF
Delete workflow and its execution
Kill action via RPC

Fig. 3. Sequence diagram for deleting VNF in the proposed model.

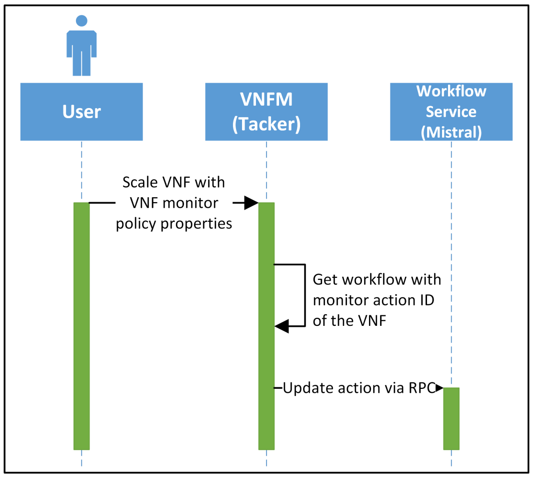

User
VNFM (Tacker)
Workflow Service (Mistral)
Scale VNF with VNF monitor policy properties
Get workflow with monitor action ID of the VNF
Update action via RPC

Fig. 4. Sequence diagram for scaling VNF in the proposed model.

VNF. Then, it will update action to Mistral workflow service via RPC. If the VNF scales out, the VNF monitoring task in policy action will be added for additional VNF instance. If the VNF scales in, the VNF monitoring task of removed VNF instance will be deleted.

Ⅲ. Experiment

Table 1 shows the implementation environment's specification. In this experiment, Tacker together with OpenStack is adopted to build the testbed environment via the Devstack tool[8] which is a series of extensible scripts used to quickly bring up a complete OpenStack environment.

The proposed model is implemented on the Tacker by modifying the "monitor.py" file in "vnfm" folder. The VNF monitoring workflow module is written in a new file called "vnf_monitor_action.py" contained in "vnfm/workflows" of Tacker source code. The "conductor" folder, which is conductor server mentioned before, is also added to the source code that conducts RPC communication channel for Mistral workflow service and Tacker Conductor. The implementation and configuration files used in this experiment can be found at [9].

Before the VNF is deployed, the VIM needs to be registered and the VNF Descriptor needs to be

Table 1. Specification of implementation environment

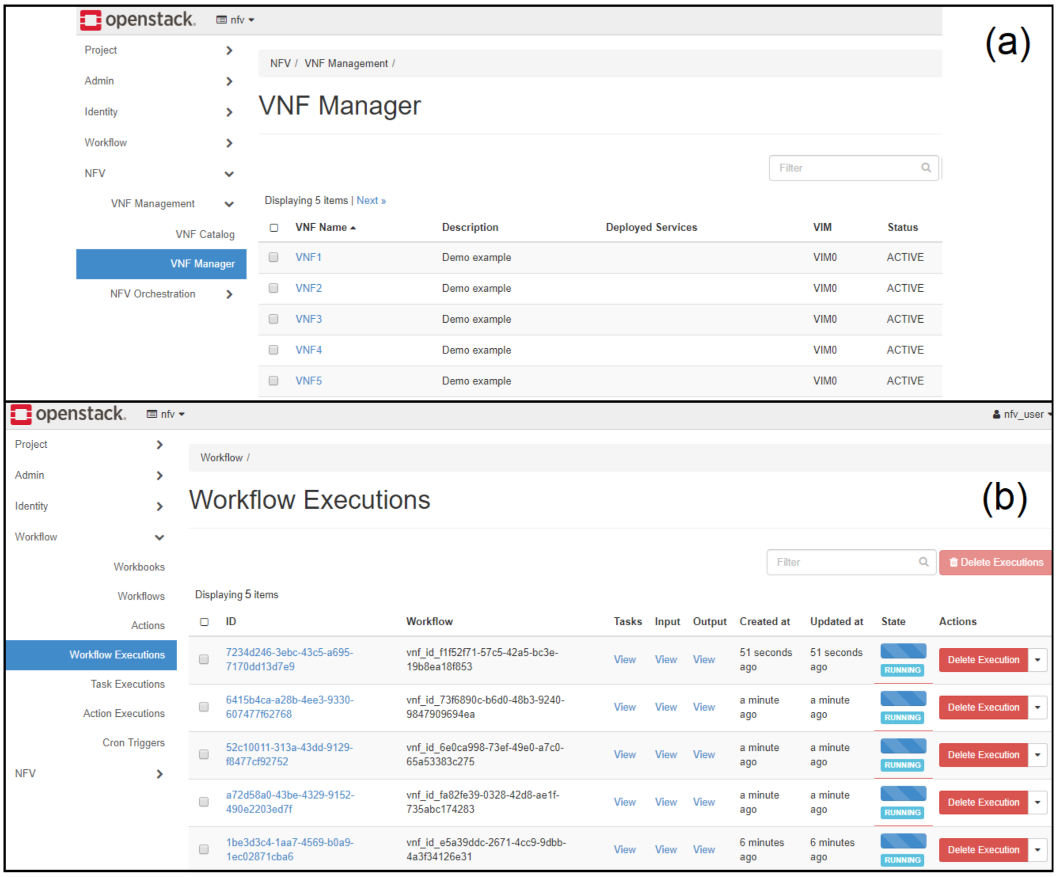
Entity	Details
Server hardware	Intel(R) Core(TM) i5-7600 CPU @ 3.50GHz (4 Cores), 16GB RAM, 120GB SSD
Operation System	Ubuntu 16.04 LTS, 64-bit
Devstack	Version: stable/pike
Tacker	Version: stable/pike
Mistral	Version: stable/pike

on-boarded. When the VNF is successfully deployed, the Mistral workflow service receives RPC request for the VNF monitoring. It validates and then starts the workflow task for VNF monitoring. The VNF monitoring action is called and the loop for monitoring is then executed.

When multiple VNFs are deployed as shown in Figure 5-a, the Mistral workflow service also creates multiple corresponding workflow executions as shown in Figure 5-b. If a VNF is deleted, a corresponding workflow execution is also removed.

Ⅳ. Conclusion

This article proposes and evaluates the workflow policy-based VNF monitoring mechanism for Tacker-based NFV system. The VNF monitoring task will be processed by an external workflow service, hence, the VNF Manager can be more scalable and effective. The proposed model is implemented in Tacker which is an open-source NFV MANO in the OpenStack cloud platform. The experiment results show that the proposed VNF monitoring model is feasible and effective.



VNF Manager

Displaying 5 items | Next »

VNF Name ▲	Description	Deployed Services	VIM	Status
VNF1	Demo example		VIM0	ACTIVE
VNF2	Demo example		VIM0	ACTIVE
VNF3	Demo example		VIM0	ACTIVE
VNF4	Demo example		VIM0	ACTIVE
VNF5	Demo example		VIM0	ACTIVE

Workflow Executions

Displaying 5 items

ID	Workflow	Tasks	Input	Output	Created at	Updated at	State	Actions
7234d246-3ebc-43c5-a695-7170dd13d7e9	vnf_id_f1f52f71-57c5-42a5-bc3e-19b8ea18f853	View	View	View	51 seconds ago	51 seconds ago	RUNNING	Delete Execution
6415b4ca-a28b-4ee3-9330-607477f62768	vnf_id_73f6890c-b6d0-48b3-9240-9847909694ea	View	View	View	a minute ago	a minute ago	RUNNING	Delete Execution
52c10011-313a-43dd-9129-f8477cf92752	vnf_id_6e0ca998-73ef-49e0-a7c0-65a53383c275	View	View	View	a minute ago	a minute ago	RUNNING	Delete Execution
a72d58a0-43be-4329-9152-490e2203ed7f	vnf_id_fa82fe39-0328-42d8-ae1f-735abc174283	View	View	View	a minute ago	a minute ago	RUNNING	Delete Execution
1be3d3c4-1aa7-4569-b0a9-1ec02871cba6	vnf_id_e5a39ddc-2671-4cc9-9dbb-4a3f34126e31	View	View	View	6 minutes ago	6 minutes ago	RUNNING	Delete Execution

Fig. 5. Multiple VNFs are monitored by corresponding workflow executions.

References

[1] ETSI NFV, *Network function virtualisation: An introduction, benefits, enablers, challenges & call for action*, Introductory White Paper, Issue 1, SDN & OpenFlow World Congr., Darmstadt, Germany, Oct. 2012.

[2] R. Mijumbi, et al., "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236-262, 2016.

[3] T. H. Nguyen, T. Nguyen, and M. Yoo, "Analysis of deployment approaches for virtual customer premises equipment," in *Proc. 32nd ICOIN 2018*, pp. 289-291, Chiang Mai, Thai Lan, Jan. 2018.

[4] C. Tipantuna and Y. Paul, "Network functions virtualization: An overview and open-source projects," *2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM). IEEE,* 2017.

[5] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "OpenStack: toward an open-source solution for cloud computing," *Int. J. Comput. Appl.*, vol. 55, no. 3, 2012.

[6] The OpenStack Foundation: OpenStack Tacker. [Online]. Available: https://docs.openstack.org/tacker/.

[7] The OpenStack Foundation: OpenStack Mistral. [Online]. Available: https://docs.openstack.org/mistral/.

[8] The OpenStack Foundation: DevStack. [Online]. Available: https://docs.openstack.org/devstack/.

[9] Tacker VNF Monitoring based on Mistral [On line]. Available: https://anda.ssu.ac.kr/tacker-mistral/.

응웬트리하이 (Tri-Hai Nguyen)

Tri-Hai Nguyen received his B.S. degree in computer science from the University of Information Technology, Ho Chi Minh City, Vietnam, in 2015 and M.Eng. degree in information and communication technology from Soongsil University, Republic of Korea, in 2017. He is currently pursuing the Ph.D. degree in information and communication technology at Soongsil University, Seoul, Republic of Korea. His research focuses on network function virtualization and cloud computing.

유 명 식 (Myungsik Yoo)

Myungsik Yoo received his B.S. and M.S. degrees in electrical engineering from Korea University, Seoul, Republic of Korea, in 1989 and 1991, and his Ph.D. in electrical engineering from State University of New York at Buffalo, New York, USA in 2000. He was a senior research engineer at Nokia Research Center, Burlington, Massachusetts. He is currently a professor in the school of electronic engineering, Soongsil University, Seoul, Republic of Korea. His research interests include visible light communications, sensor networks, Internet protocols, control, and management issues.