A Design of MANO System for Cloud Native Infrastructure

2021 International Conference on Information and Communication Technology Convergence (ICTC) | 978-1-6654-2383-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICTC52510.2021.9620858

Jangwon Lee
Dept. of Information and Telecommunication Engineering
Soongsil University
Seoul, Korea
jangwon.lee@dcn.ssu.ac.kr

Younghan Kim
School of Electronic Enginnering
Soongsil University
Seoul, Korea
younghak@ssu.ac.kr

***Abstract*— Virtual Network Functions (VNFs) usually have been running as software on Virtual Machine (VMs) in cloud infrastructure. However, as container technologies are growing in the cloud area, containerization of VNFs is required, and updated NFV reference architecture is under development in European Telecommunications Standards Institute (ETSI) and hybrid NFV system which support both of VMs and containers are required. In this paper, we solve two problems: Kubernetes, where only a few features are used to provide container based VNFs, and administrator challenges due to different resource models used in each infrastructure. With the proposed architecture and new resource models that fully utilize Kubernetes, administrators can obtain a single hybrid MANO system and management speed with more readable and friendly descriptors. For these improved solutions, we propose through Kubernetes, Operator Framework, and KubeVirt.**

Keywords—NFV, Cloud Native, Kubernetes, KubeVirt

I. INTRODUCTION

Network Function Virtualization (NFV)[1] is a method of virtualizing network services such as routers, firewalls, and load balancers that have previously run on proprietary hardware. European Telecommunications Standards Institute (ETSI) continues to implement standards for NFV to define an architecture, interface, and descriptor of NFV and Management and Orchestration (MANO). As a result, network functions (NFs) were decoupled from hardware to software. These Virtual Network Functions (VNF) have been running on Virtual Machine (VM) so it has become possible to operate a variety of network functions in the cloud infrastructure[2]. As a result, various MANO open source projects have been launched and implemented with the cloud infrastructure field such as OpenStack.

With advent and development of container technology, the cloud infrastructure began to change. Since the era of container technology, explosive advances have been made in a variety of methods like microservice architecture, DevOps, Agile to increase the efficiency of development and management[3]. Linux Foundation created Cloud Native Computing Foundation (CNCF) and announced several open source projects under development in Cloud Native. Kubernetes[4] is one of them and container technologies are growing in cloud area, discussions have taken place to accommodate hybrid system in NFV.

ETSI defined the elements and interfaces needed for container-based VNFs, taking into account the differences from the existing VM environment. ETSI reported GR NFV-IFA 029 that what is needed to enhance of the NFV architecture towards Cloud-native and PaaS. They defined the Container Infrastructure Management System (CISM) that can be a Kubernetes Master and the role and position of it with 6 options which corresponds to NFV MANO[5]. Accordingly, various open-source MANO projects also develop to support them. But, current open-source projects have been carried out by creating VNFs with each cloud resource rather than in hybrid form. As a result, the container infrastructure has simply become a Virtualization Infrastructure Manager (VIM) that supports container resources, no different from the role of traditional VIMs.

In this paper, we want to solve two problems with a proposed architecture. First is Extending container infrastructure, Kubernetes as MANO system unlike other Open Source MANO System. Kubernetes will initiate and manage VM and container-based VNF so administrators can deploy to any Kubernetes even if providers are different. Second is TOSCA based descriptor commonization. Currently administrators have to know each resource languages of VIM such as HEAT template when VIM is OpenStack even if they request with TOSCA based one. If administrators can read and write resource models of VIM by TOSCA based, it will improve readability and speed. In order to solve these problems, we describe and implement our proposal using open source container platform Kubernetes and its addons opensource project Operator Framework and KubeVirt[6] to extend Kubernetes to support VNFs.

II. RELATED WORK

A. OpenSource MANO Projects

ETSI defines the basic NFV architecture with entities: NFV infrastructure (NFVI), VNFs, and MANO[7][8]. A variety of MANO open source projects are under way based on ETSI. There are Open Network Automation Platform (ONAP)[9] that developed based on the telecom infrastructure, a representative private cloud OpenStack MANO project, Tacker[10], and Open Source MANO project (OSM)[11], which are being implemented in ETSI. These projects have been based on VM environments since several years ago, and are now accepted in each projects as defined on Cloud Native in ETSI. ONAP has previously used Kubernetes to configure its infrastructure, and it is now possible to deploy and manage container-based VNFs through the Helm package as a method for Cloud Native. OpenStack's Tacker is being discussed in several ways by converting existing TOSCA-based descriptors into resources of Kubernetes.

B. Operator Framework

Operator Framework[12][13] is an open source project that allows Kubernetes to be easily and quickly expanded and managed using a component called Operator. The Operator Framework allows administrators to define new resources with CRD form and controllers that monitor and manage them in the same way that each resource controller in Kubernetes manages. It is widely used for application deployments where settings or objects can change continuously, and a typical example is Prometheus Operator. The problem of manually changing and restarting Prometheus when monitoring and alarm has to be change is solved by providing CRDs which related monitoring and alarm so Prometheus automatically re-enable the changes. Operator Framework consists of the following components:

- Operator SDK: used by developers to implement Operators and can be implemented without knowledge of the Kubernetes API.
- Operator Lifecycle Management (OLM): manager who manages the operator. It installs, updates, and manages the lifecycle of the operator.

III. SYSTEM ARCHITECTURE AND PROCEDURE

A. Architecture

This section describes the MANO structure and procedures for the Kubernetes-based Cloud Native proposed by this paper.

Fig. 1 describes the proposed structure mapping into ETSI NFV reference architecture. The ETSI NFV MANO block can be replaced by the VNF Operator proposed in this paper and consists of pods in Kubernetes. VIM should be able to manage NFVI, which corresponds to Kubernetes' Master Node. In addition, NFVI can be correspond as a work node because it must have the resources to create a VNF. VNF and EM of ETSI NFV can consist of a set of pods in Kubernetes.

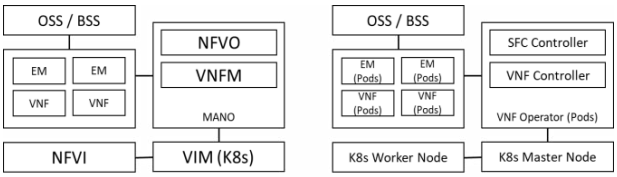

Fig. 1. Kubernetes Mapping into NFV Reference Architecture

Fig. 2 describes proposed VNF management and orchestration architecture. As shown in Fig. 1, it consists of the Kubernetes Master node and the Worker node, which consists of:

- VNF Operator: The MANO System proposed in this paper consists of VNF Controller, which manages Life Cycle Management (LCM) of VNF like VNFM, and SFC Controller, which manages Service Function Chaining (SFC), which is part of NFVO. It is implemented using Operator Framework.
- VNF API: API component that manages the VNF Operator and allows it to process inputs from the extended Kubectl binary or REST API.
- virt Controller: Controller component of KubeVirt, an open source project for VM management at Kubernetes. It is also implemented using Operator Framework and is responsible for the LCM of VM-based pods.
- virt API: API component for KubeVirt. It processes a similar role to the VNF API.
- Pod: smallest deployable computing unit that can be created and managed by Kubernetes. In the architecture, VM-based and container-based pods can be deployed.
- Pod (virt-handler): distributed for each Worker Node to manage VM-based pods.
- VNF: can be composed of a Pod or multiple Pods defines a Pod as a VNFC. Also, VNF can be composed regardless of the VM and container.

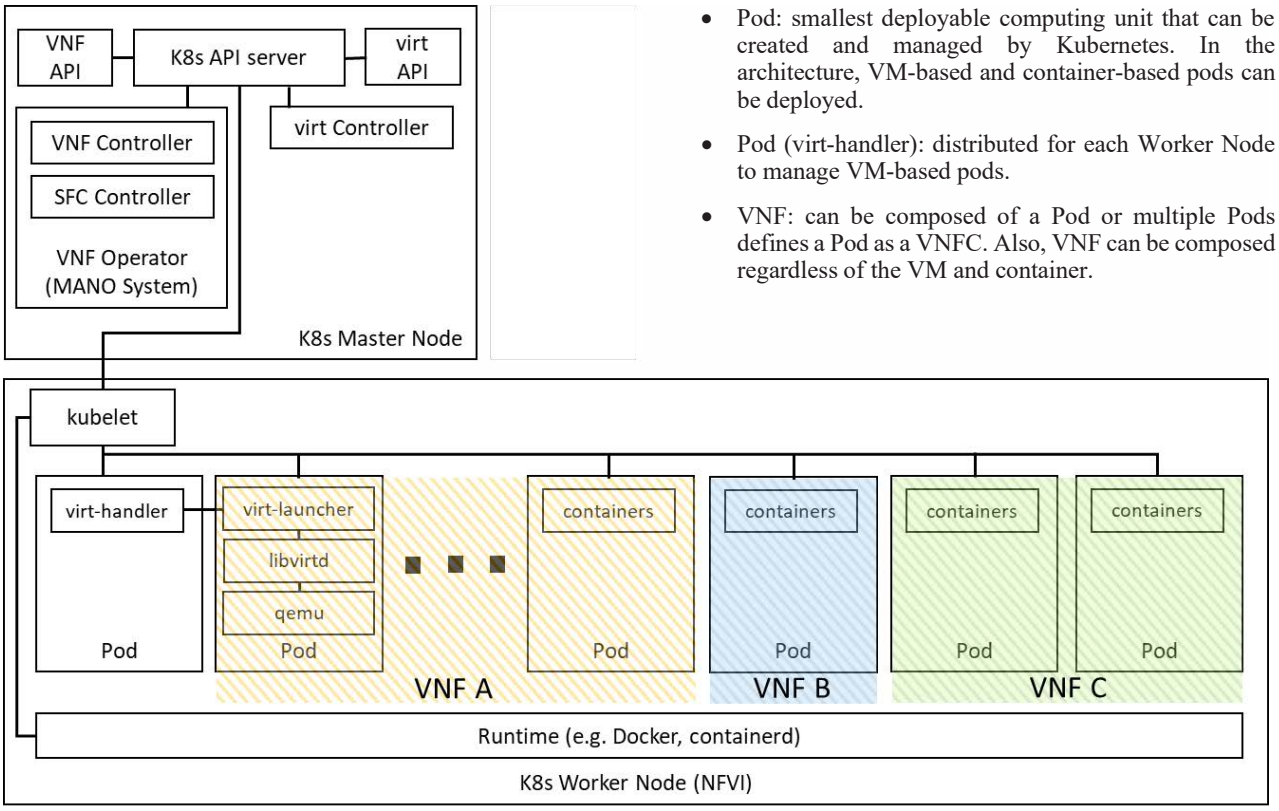

Fig. 2. Proposed VNF Management and Orchestration Architecture

B. Procedure

Fig. 3 describes VNF deployment procedure using the proposed architecture. The administrator can request via API or CLI command or deploy CRDs of VNF to K8s API server manually. The requested VNF CRD is known by the VNF Controller of the VNF Operator being monitored and initiates the procedure for VNF initiation. The VNF Controller can convert to appropriate Kubernetes resource models and request the K8s API Server to create a container-based VNF. When the VNF Controller requires a VM-based pod, it creates resources using KubeVIrt's CRD, Virtual MachineInstance (VMI). Virt Controllers are equally aware of resource requests during monitoring and are able to provide VM-based pods through actions with Kubelet and virt-handler[14].

C. Custom Resource Definition of VNF

Fig. 4 describes an example of VNF Custom Resource Definition (CRD) that has 1 VDU, 1 CP defined by ETSI. Defined CRD can get parts of templates that defined by ETSI NFV-SOL at the bottom of spec field. This method means that NFV managers can configure and update VNFs based on the grammar defined in ETSI. This can make the resource model more readable, easier to manipulate, and solve problems that have been dependent on existing infrastructure resource models.

Tenants used in cloud infrastructures can be configured through Kubernetes namespace. VNF Operator converts and distributes VNF CRD to Kubernetes resources such as Pod, Service, and HPA, and associates and manages VNF resources. Considerations for other 3rd parties apply only to annotation field of the transformed Deployment and Pod resources. If an administrator changes VNF resource, the VNF Operator detects it and automatically updates the associated resources and redirects them as necessary.

The VNF Controller of VNF Operator already has a framework of related structures to handle VNF CRD, which can accommodate templates defined in ETSI NFV-SOL 001[15]. In addition, special templates from providers that can be configured in the way defined in ETSI can be supported by deployments in package formats described in the following paragraphs.

```
apiVersion: dcn.com.my.domain/v1
kind: VNF
metadata:
  name: vnf-sample
  namespace: default
spec:
  node_templates:
    VDU1:
      type: tosca.nodes.nfv.Vdu.Compute
      properties:
        name: VDU1
        description: VDU1 compute node
        vdu_profile:
          min_number_of_instances: 1
          max_number_of_instances: 1
        sw_image_data:
          name: Software of VDU1
          version: '0.4.0'

    capabilities:
      virtual_compute:
        properties:
          virtual_memory:
            virtual_mem_size: 512 MB
          virtual_cpu:
            num_virtual_cpu: 1

    CP1:
      type: tosca.nodes.nfv.VduCp
      properties:
        layer_protocols: [ ipv4 ]
```

Fig. 4. Example of VNF Custom Resource Definition

D. VNF Package

VNF Package specification is defined in ETSI NFV-SOL 004[16]. For package configuration, TOSCA Cloud Service Archive (CSAR) is defined and Fig. 5 describe its structure:

- TOSCA-Metadata: directory that holds the TOSCA.meta file containing the entry information for the CSAR package.
- Definitions: directory that contains TOSCA template files defined by ETSI for VNF initiation.
- Files: directory that contains additional files for initiating or managing VNFs, such as licenses and images.
- Scripts: directory where the scripts will proceed after VNF Initiation.

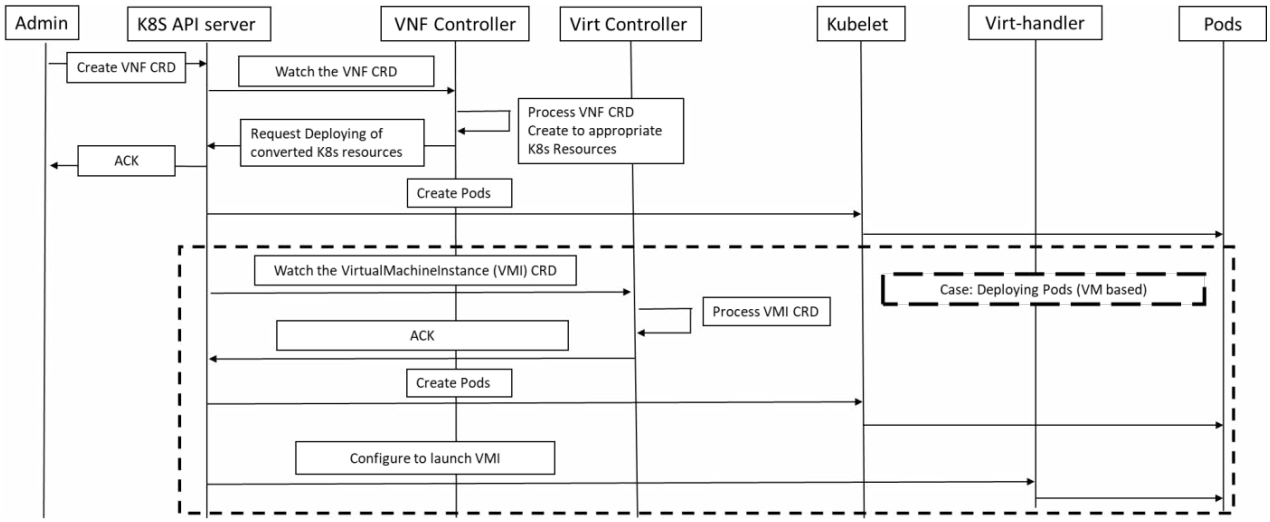

Fig. 3. VNF Deployment procedure

```
!------TOSCA-Metadata
        !------TOSCA.meta

!------Definitions
        !----- MRF.yaml
        !----- OtherTemplates (e.g. type definitions)

!------Files
        !----- ChangeLog.txt
        !----- MRF.cert
        !----- image(s)
        !----- other artifacts
        !------Tests
        !----- file(s)
        !------Licenses
        !----- file(s)
!------Scripts
        !----- install.sh
```

Fig. 5. TOSCA CSAR Example

The system proposed in this paper uses Kustomize[17], which can customize the settings of Kubernetes resources for package configuration for VNF. Each directory in Fig. 5 can exist under each of the bases in Fig. 6. VNF can be distributed by modifying the folder or by partially modifying using patch features of Kustomize if it is necessary due to differences in the requirements of each provider, etc. of the provider.

```
# Adds namespace to all resources.
namespace: default

# Value of this field is prepended to the
# names of all resources, e.g. a deployment named
# "wordpress" becomes "alices-wordpress".
# Note that it should also match with the prefix (
# field above.
namePrefix: vnf-operator-

# Labels to add to all resources and selectors.
#commonLabels:
#  someName: someValue

bases:
- ../TOSCA-Metadata
- ../Definitions
- ../Files
- ../Scripts
```

Fig. 6. Example of kustomization.yaml for VNF package

IV. Conclustion and Fucture work

In this paper, we propose an architecture to provide Cloud Native using only the container infrastructure, Kubernetes. It also allows administrators to leverage TOSCA-based templates defined in ETSI NFV-SOL, reducing dependencies on infrastructure resource models and improving readability and commonality.

Currently, the functions performed on VNFM are primarily implemented, but later the functions performed on NFVO need to be implemented and more diverse states and procedures need to be defined and implemented to meet ETSI standards. It is expected that if these problems are improved, it can be a project corresponding to existing open-source MANO projects.

Acknowledgment

This research was supported by the MSIT (Ministry of Science and ICT). Korea, under the ITRC (Information Technology Research Center) support program (IITP-2021-2017-0-01633) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation.

References

[1] ETSI, NFV whitepaper. at the “SDN and OpenFlow World Congress”, Oct. 2012. https://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[2] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee, “Network Function Virtualization: Challenges and Opportunities for Innovations”, IEEE Communication Magzine, vol. 53, pp. 90-97, February 2015.

[3] Armin Balalaie, Abbas Heydarnoori, “Microservices Architecture Enables DevOps; Migration to a Cloud-Native Architecture”, IEEE Software, vol. 33, pp. 42-52, March 2016.

[4] Kubernetes, accessed: 2021-08-14, [Online]. Available: https://kubernetes.io/

[5] ETSI, “Network Function Virtualization (NFV) Release 3; Architecture; Report on the Enhancements of the NFV architecture towards “Cloud-native” and “PaaS””, ETSI GR NFV-IFA 029 (v. 3.3.1), November. 2019.

[6] Ilias Mavridis, Helen Karatza, “Orchestrated sandboxed containers, unikernels, and virtual machines for isoloation-enhanced multitenant workloads and serverless computing in cloud”, Concurrency and Computation practice and Experience, May. 2021.

[7] ETSI, “Network Function Virtualization (NFV); Architectural Framework”, GS NFV 002 (v. 1.2.1), December. 2014.

[8] ETSI, “Network Function Virtualization (NFV); Management and Orchestration”, GS NFV-MAN 001 (v. 1.1.1), December. 2014.

[9] ONAP, accessed: 2021-08-14. [Online]. Available: https://docs.onap.org/en/latest/guides/overview/overview.html.

[10] Tacker: OpenStack NFV Orchestration, accessed: 2021-08-14. [Online]. Available: https://wiki.OpenStack.org/wiki/Tacker.

[11] Open Source MANO (OSM), accessed: 2021-08-14. [Online].Available: http://www.osm.etsi.org.

[12] Operator Framework, accessed: 2021-08-14. [Online]. Available: https://sdk.operatorframework.io/docs/overview/.

[13] Kubernetes, Operator pattern, accessed: 2021-08-14. [Online]. Available: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/.

[14] KubeVirt, accessed: 2021-08-14. [Online]. Available: https://kubevirt.io/user-guide/architecture/.

[15] ETSI, “Network Function Virtualization (NFV) Release 3; Protocols and Data Models; NFV descriptors based on TOSCA specification”, ETSI GS NFV-SOL 001 (v. 3.3.1), September. 2020.

[16] ETSI, “Network Function Virtualization (NFV) Release 2; Protocols and Data Models; VNF Package specification”, ETSI GS NFV-SOL 004 (v. 2.5.1), September. 2018.

[17] Kubernetes, Declarative Management of Kubernetes Objects Using Kustomize, accessed: 2021-08-14. [Online]. Available: https://kubernetes.io/docs/tasks/manage-kubernetes-objects/kustomization/