Toward True Cloud Native NFV MANO

David Breitgand *, Vadim Eisenberg *, Nir Naaman *, Nir Rozenbaum *, Avi Weit *
* IBM Research – Haifa, Israel
email: {davidbr,vadime, naaman, nirro, weit}@il.ibm.com

***Abstract*—The telecommunication industry is making a major shift towards cloud native Network Functions Virtualization. However, being cloud native implies more than just replacing Virtual Machines with containers. The cloud native approach changes the way virtual network functions are being composed, deployed, and configured as network services and how these services are being orchestrated and managed at run time. Traditional Network Management and Orchestration (MANO) frameworks are implemented via specialized engines that have been conceived prior to the cloud native era. Thus, they inherit many traits, which impede pivoting to cloud native. In this paper we focus on some fundamental challenges and propose a novel cloud native MANO architecture that exploits advantages of Kubernetes (K8s) a de facto industry standard for container orchestration.**

***Index Terms*—NFV, MANO, Kubernetes, cloud native**

I. Introduction

Paraphrasing the Cloud-Native Computing Foundation (CNCF) definition [1], cloud native computing is a set of technologies that help developing and operating scalable and portable applications as containerized micro-services that are dynamically managed and orchestrated by a platform to utilize the advantages of the cloud computing model. Spurred by advancement of 5G, the telecommunication industry is going through a cloud native transformation [2]. However, applying the cloud native approach to the telecommunication workloads is challenging and the industry has to go a long way before the potential of cloud native is fully realized for the telecommunication workloads. The challenges stem from the fact that, on the one hand, many of these workloads are legacy ones and even when containerized do not readily render themselves to the cloud native methodology. On the other hand, leading cloud native platforms, such as K8s lack certain features that are required for executing and managing telecommunication workloads and these gaps should be filled before platforms like K8s become the platforms of choice for the telco industry. Discussing all aspects of applying the cloud native methodology to telecommunication is beyond the scope of this paper. Rather, we focus on four essential aspects of the problem that have received less attention so far.

First, traditional network services orchestration frameworks predate the cloud native era [3]. While recently major MANO frameworks claim that they have become cloud native, in reality this amounts to containerizing the MANO engines and executing them on top of container orchestrators, such as K8s. The workflow orientation, centralization of the management logic, and limited support for highly dynamic workloads still permeate the MANO frameworks. To address these issues, [3] proposed making K8s itself a MANO framework. Rather than deploying K8s as "dumb" Network Functions Virtualization Infrastructure (NFVI), it is utilized at the levels of Virtual Network Function Manager (VNFM) and Network Function Virtualization Orchestrator (NFVO). Yet, to the best of our knowledge, no in-depth study of making K8s itself a cloud native MANO framework suitable to manage telecommunication workloads has been presented so far. This is a broad topic and we consider this work to be a first necessary step in this direction.

While MANO is a generic concept that can be implemented in a multitude of ways, the workflow oriented frameworks, such as OSM [4] or ONAP [5], are divorced from the cloud native paradigm that is being used by the workloads that they need to manage. The workflow orientation holds true even for experimental MANO implementations that rely on K8s native workflow operators, such as Argo [6], as was reported in [7–9]. In [3], K8s itself has been proposed as a potential next generation cloud native MANO. However, we argue that, as part of switching to K8s as a basis for MANO, the traditional workflow oriented mindset of MANO needs to be transformed to K8s native controller-based orchestration approach [10] to exploit best practices of K8s, such as Operators [11].

Second, cloud-native prompts for decomposition of traditional VM-based Virtual Network Functions (VNF) into smaller container based micro-services with well defined functionality that can be developed, scaled, moved, managed and orchestrated in a much more flexible way than legacy VNFs. Furthermore, this decomposition opens up opportunities for more Network Equipment Providers (NEP) to enter the market which promotes much needed innovation and competition in a traditionally vertically integrated industry. However, these new opportunities also bring about a problem of how to compose these diverse Cloud Native (or Container) Network Functions (CNFs) into larger services in a methodological way suitable to the telecommunication requirements. In particular, a traditional telecommunication taxonomy differentiates between Virtual Deployment Units (VDU), Virtual Network Function Components (VNFC) that might comprise several VDUs, Virtual Network Functions (VNF) that might be constructed of a few VNFCs, VNF Packages, and Network Services that are built of VNF packages [12]. In [13], these concepts are mapped to K8s pods (VDU), Replica Set and Deployments (VNFC), Services (VNFs), and Helm charts (VNF packages). However, there is

978-1-6654-2434-9/21/$31.00 ©2021 IEEE

no natural concept for Network Service in K8s. We refer to this as a *composition problem*[1].

Third, network services comprising multiple CNFs might have complex inter-dependencies between them. These inter-dependencies often lead to the need of propagating dynamically bound properties from one CNF to another[2]. In the telecommunication domain, there exist multiple examples when dynamic dependency management cannot be achieved via standard methods like those of K8s that require every micro-service to be accessible via predefined known ports and via predefined local URLs. As an example consider non-HTTP protocols such as Session Initiation Protocol (SIP) being part of a typical telecommunication workload, such as VoIP. We refer to this issue as a *dynamic dependency management problem*.

Fourth, it is required that MANO would support multiple NFVIs. We argue that K8s is well suited for multi-NFVI scenarios because of being extensible via the mechanism of Custom Resource Definitions (CRD), which allows to present non-K8s resources to K8s based controllers and therefore make them part of K8s ecosystem. We propose to use the K8s Operator pattern to develop Virtual Infrastructure Managers (VIM) for non-K8s NFVI[3].

Our specific contributions are as follows.

- We propose K8s and its ecosystem as a cloud native alternative to the existing MANO orchestration frameworks. We claim that this is a more elegant, portable, and streamlined compared to deploying separate orchestration engines on top of K8s;
- We address four aspects related to this proposition and, broader, to the cloud native shift in the telecommunication industry: K8s native orchestration, composition, dynamic dependencies management, and multi-NFVI management;
- We offer an open source prototypical implementation that we deem useful for the community for experimentation and research of applying cloud native principles to telecommunication workloads.

The rest of this paper is organized as follows. In Section II we highlight the background and related work. In Section III we formulate the problems in greater detail. This is followed by proposing a solution in Section IV that details our proposed architecture and walks the reader through our open source implementation available in [14, 15]. We provide illustrative examples to accrue intuition and understanding of the issues involved and why they are not trivial to address. Section V offers concluding remarks and outlines future directions.

II. Background and Related Work

[16] provides introduction into Network Function Virtualization (NFV) and explains its benefits, enablers and challenges. ETSI MANO [17] is a standard for NFV Orchestration and Management (NFV MANO). [18] describes the challenges involved in NFV MANO. OSM [4] and ONAP [5] are popular NFV MANO frameworks.

[3] proposed using K8s [19] as VNF orchestrator, however the authors did not expand on the issues involved in using K8s as NFV MANO framework.

Since the focus of K8s originally was on enabling orchestrating container-based applications, its core lacks support for composition of these applications. The core K8s provides neither first-class-citizen support for declaring which application K8s constructs (pods, configmaps, etc.) belong to, nor supports using the applications as building blocks in composition of other, higher-level applications.

To address grouping of basic K8s constructs into higher level applications, *K8s Application CRD and controller* [20] were introduced. However, this construct does not provide means to compose the applications into higher-level applications. It also does not provide means to dynamically manage dependencies between the applications.

In K8s ecosystem, composing K8s constructs into higher level applications can be achieved by Helm [21], a popular package managing tool for K8s. Helm has a concept of charts and sub-charts. A chart represents a packaged application that can be deployed to K8s. Using Helm sub-charts feature, applications represented by charts can be composed into complex applications. However, composition of applications by Helm lacks dynamic dependency management. Helm handles only static (deployment-time dependencies) dependencies between K8s components, using text templating.

Another framework that handles composition and dependency management on K8s is Crossplane [22]. Crossplane allows creating applications that claim various composite infrastructure resources while handling dependency management between the resources and the applications. The focus of Crossplane, however is on infrastructure management for applications. A limitation of Crossplane is that the composite resources are cluster-scoped.

Yet another framework that handles composition and dependency management, not only for K8s but also for other infrastructures, is Juju [23]. However, Juju is not K8s-native and does not apply the Operator pattern of K8s.

III. Challenges

In this section we describe the problems of dynamic dependencies management and composition in greater details. We consider a Network Service (NS) that is composed of multiple CNFs. The CNFs have a set of inter-dependencies between them. We consider the case where CNF1 depends on a set of properties of CNF2. While the set of dependent properties is known to the NS developer, the actual values of these properties are not known until CNF2 is started. *Dynamic Dependency Management* addresses the problem of automatically resolving such dependencies at run time. The NS developer should only have to define the dependencies. The orchestrator is then responsible for resolving these dependencies by addressing

[1]The composition problem transcends Network Services composition, but in this paper, we only consider aspects related to telecommunication workloads.

[2]The inter-dependencies that we discuss here are runtime dependencies among distributed components that should not be confused with build/linkage/-dynamic loading dependencies declaration w.r.t. libraries and software packages within a single software component

[3]Note that while K8s removes the need for VIM for containerized workloads, VIM is still useful when it comes to managing non-K8s resources.

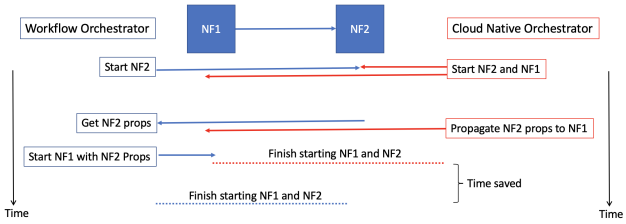

Fig. 1: Property Propagation Comparison

two aspects: control over the start-up sequence of the CNFs, and dynamically propagating the set of dependent properties. Control over the start-up sequence means that CNF1 should wait until CNF2 is started and the values of the dependent properties are known. Note that this does not mean that the two components cannot be started at the same time, only that CNF1 should wait until the values of all dependent properties of CNF2 are propagated to it. Dynamic property propagation means that once the value of a dependent property is set in CNF2 it should be propagated and become available to CNF1.

Fig. 1 presents a comparison between workflow based orchestration and a cloud native orchestration with dynamic property propagation. It should be noted that in the cloud native case a controller continues to constantly watch the dependencies even after both CNFs have been started. Upon any change the controller will propagate the properties to the components that depend on them.

The problem of *composition* is how to enable NS developers to reuse repeating building blocks when composing a NS. Typically a NS comprises multiple VNFs (or CNFs) and each VNF comprises multiple VNF Components (VNFC). In many cases there is a subset of VNFs or VNFCs that creates a repeated pattern. Such patterns can be reused as building blocks to make the design of NSs and VNFs easier. The developer should be able to create a composition of multiple basic building blocks (e.g., VNFCs) and then extend this with compositions that also includes other compositions. Once such compositions are created the orchestrator should be able to orchestrate them as a single unit, e.g., scale a composition in/out or perform health checks and healing operations at a composition level. Using such compositions greatly simplifies NS development and ensures more efficient and consistent lifecycle management. K8s, however, does not support such reusable composition and only supports orchestration at the level of resources (K8s native or CRs).

Another challenge deals with supporting multi-NFVI scenarios. MANO achieves support for multi-NFVI use cases by introducing VIMs that serve as adaptors between the standard APIs (e.g., as defined by ETSI) and the platform ones of different NFVIs. In our proposition, all cloud native resources are managed directly, so no adaptor is required. However, for non-K8s NFVIs either a K8s managed VIM should be provided or delegation to existing MANO frameworks should be exploited or some combination thereof. Due to the lack of space we only briefly discuss one possible implementation that follows the first approach.

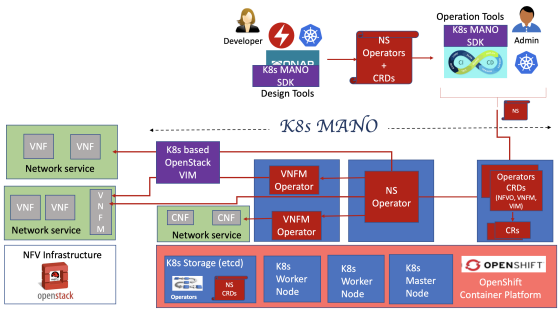

Fig. 2: Architecture Overview

IV. Proposed Solution

In this section we present a prototypical implementation of a K8s native NFV MANO. We first describe the architecture along with elaboration on some implementation details. Next we present an example of a Network Service that is designed and managed using the concepts and tools we described earlier.

A. Architecture and Implementation

Fig. 2 presents a high level view of the architecture. Our cloud-native NFV MANO extends the K8s style of performing operations on applications to orchestrating CNFs using the *operator pattern* [11] according to a control-loop [10].

Following the operator pattern, CNFs are managed by *K8s operators* [11], ideally provided by CNF vendors on an operator marketplace like [24]. The operators provide a definition of a K8s Custom Resource (CR), an instance of Custom Resource Definition (CRD), that represents the desired and the actual state of the CNF. The operator watches the CR and tries to reconcile the desired state with the actual state. The NS lifecycle is also managed by an operator, with a CR that represents the components of the NS and the dependencies between them. Therefore, orchestrating a NS means running multiple operators (one per CNF and one for NSs), working continuously in control loops and updating CRs of one another.

Our proposed K8s native NFV MANO is designed in a cloud native fashion as a hierarchy of K8s operators, with an additional SDK that facilitates development of NSs along with their orchestrating operators.

The NS developer creates a NS descriptor which is represented as a CRD. In addition, the NS includes a set of VNFM operators, one per VNF or CNF that the NS comprises. The set of NS operators and CRDs is then deployed by an Admin to a K8s cluster. To simplify and standardize NS development we implemented a NS operator that is acting as a generic Network Function Virtualization Orchestrator (G-NFVO). When a NS CR is created from the CRD representing the NS descriptor, the G-NFVO processes it and instantiates the NS. During NS instantiation the CRs for the VNFM operators are created and these operators instantiate the CNFs included in the NS. All other NFV MANO lifecycle operations (e.g., scale or update) are performed in the same K8s native fashion by modifying the NS CR. The G-NFVO operator watches all NS CRs and executes any operations that are required to bring the NS's

NFV MANO Function	Implementation in K8s + Operators
Onboarding	Operator Lifecycle Manager
Instantiate	kubectl create
Scale	kubectl scale + horizontal pod autoscaling
Update configuration	kubectl apply/edit
Upgrade	kubectl apply/edit + rolling update
Fault management	reconciliation by the operators + liveness probes
Terminate	kubectl delete + owner references + finalizers

TABLE I: Implementation of MANO operations by K8s and Operators

current state to the desired state specified in the NS CR. In Table-I we describe how MANO operations are implemented using K8s operators and CRs.

In addition to NS lifecycle management, the G-NFVO operator implements dynamic property propagation to manage dynamic dependencies. The NS CR contains the list of components and includes mapping of properties between different components. Each component represents a CNF CR and has its own specification. The NS CR also contains a list of properties in the NS level that can be propagated to and from the components that the NS consists of. As every K8s operator, G-NFVO operator works in continuous periodic reconciliation cycles. In each cycle the whole state of the NS is reconciled to address any changes in any dependent resource. As part of this reconciliation cycle the G-NFVO goes over the list of the components specified in the NS CR and dynamically propagates values of properties between components.

As Fig. 2 shows, to support non-K8s NFVI, we offer VIM operators to orchestrate these NFVIs. The design follows the same pattern as before. The desired and the observed state of a non-K8s resource is represented using a corresponding CR while VNFM and NFVO can manipulate the desired state to trigger the VIM operator to reconcile the observed state of the resource with the desired state via North Bound Interface (NBI) of the managed NFVI (in this case OpenStack).

B. Network Service Example

To illustrate the architecture and highlight our implementation details we use an example of a Voice Over IP (VoIP) NS. The VoIP NS comprises three CNFs – *ippbx*, which is used to connect telephone extensions to the public switched telephone, *voipGateway*, which is used to load balance traffic across a dynamic pool of registered ippbx servers and *sipp*, which serves as a traffic generator for the VoIP SIP protocol. These three components exhibit chain-dependencies of properties where sipp depends on voipGateway which depends on ippbx. Examples of these dependencies include properties such as the K8s namespace which the NS CR will be deployed to, and the IP addresses that will be assigned to the pods running the CNFs. The values of these properties are not known during the development of the NS since they are set only when the CNFs are actually instantiated. To resolve the dependencies between the different CNFs and to ensure that each CNF will have the correct values of the properties it requires, dynamic property propagation is used.

Listing 1: VoIP Network Service Custom Resource

```
apiVersion: gnfo.ibm.com/v2alpha1
kind: NetworkService
metadata:
  name: voip-network-service
spec:
  components:
    ippbx:
      template:
        apiVersion: ippbx.example.com/v1alpha1
        kind: Ippbx
        metadata:
          name: "[% meta.name %]-ippbx"
          namespace: "[% meta.namespace %]"
    voipGateway:
      template:
        apiVersion: voipgw.example.com/v1alpha1
        kind: VoipGW
        metadata:
          name: "[% meta.name %]-gateway"
          namespace: "[% meta.namespace %]"
        spec:
          dispatchList: "[% ippbx.status.dataAddress %]"
          port: 5060
    sipp:
      template:
        apiVersion: sipp.example.com/v1alpha1
        kind: Sipp
        metadata:
          name: "[% meta.name %]-sipp"
          namespace: "[% meta.namespace %]"
        spec:
          targetSipAddress: "[% voipGateway.status.externalAddress %]"
```

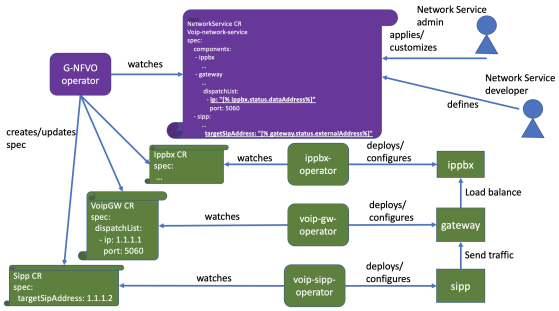

Fig. 3: VoIP Example - Operators and CRs

In Listing-1, we show the CR of the VoIP NS. Once this VoIP CR is deployed to K8s, the G-NFVO gets an event notification and starts taking actions in order to bring the system to the desired state as detailed in the CR. At the first stage, the G-NFVO creates all the sub-resources as they are specified in the components field. In our example it creates three additional CRs – Ippbx, VoipGW and Sipp. These resources have their own K8s operators which act as VNFMs that handle each CNF's lifecycle. Once the CNF's properties have values assigned, the G-NFVO propagates the values according to the specification and makes them available to the CNF that depends on them. A high level overview of the VoIP sample operators hierarchy is shown in Fig. 3, including the VoIP CNFs VNFM operators along with our G-NFVO.

In addition to the G-NFVO operator, we also supply an SDK that facilitates development of new K8s operators for composition of NSs, by implementing project scaffolding.

Once the NS developer declares a NS CR, and wants to reuse it in another NS, the developer provides it as an input to the SDK along with a name of the new component. The SDK automatically generates a new CRD with the provided name and a new operator that can translate any instance of the CRD to a NS. From that point it is possible to use the newly created CRD as one of the components in another NS descriptor. Once the top-level NS CR is created, G-NFVO operator handles the creation of all participating CRs including the composite CRs that represent other NSs as described in Fig. 4.

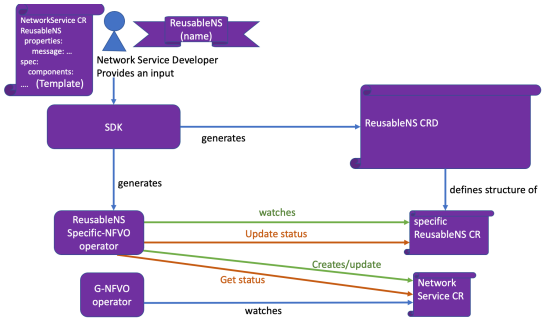

Fig. 4: SDK Generation for Composition

The auto-generated operator is responsible for two tasks:(a) to watch for any change in the composite CR and to create/update the CRs of the component NSs (which in turn are handled by the G-NFVO operator) and (b) to get status updates from the CRs of the components of the NS, and to propagate status updates to the composite CR. The auto-generated operator can also be customized so it is possible to add custom logic on top of the translation.

V. Conclusions and Future Work

We presented a novel, truly K8s based approach to NFV MANO. Further studies are required to explore the performance and scale of the proposed approach, also in integration with K8s federation. It should be noted that presently comparative quantitative study is impeded by the lack of widely accepted benchmarks. Also, additional use cases and NFVIs should be explored. We defer studying these aspects for future work.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871533 (5GZORRO).

References

[1] Cloud-Native Computing Foundation (CNCF), "CNCF Cloud Native Definition v1.0," 2018.

[2] B. Sayadi and et. al., "From webscale to telco, the cloud native journey," 07 2018.

[3] ——, "Cloud-Native and Verticals' Services," 08 2019.

[4] ETSI, "Open Source Mano (OSM)," 2021. [Online]. Available: https://osm.etsi.org/

[5] Linux Foundation Projects, "Open network Automation Platform," 2021. [Online]. Available: https://www.onap.org/

[6] "Argo Project: Open Source Kubernetes Native Workflows, Events, CI and CD," 2021. [Online]. Available: https://argoproj.github.io/

[7] F. Alvarez and et. al., "An edge-to-cloud virtualized multimedia service platform for 5g networks," *IEEE Trans. Broadcast.*, vol. 65, no. 2, pp. 369–380, 2019.

[8] S. Rizou and et. al., "Programmable Edge-to-Cloud Virtualization for 5G Media Industry: The 5G-MEDIA Approach," in *Artificial Intelligence Applications and Innovations. AIAI 2020 IFIP WG 12.5 International Workshops - MHDW 2020 and 5G-PINE 2020, Neos Marmaras, Greece, June 5-7, 2020, Proceedings*, ser. IFIP Advances in Information and Communication Technology, I. Maglogiannis, L. Iliadis, and E. Pimenidis, Eds., vol. 585. Springer, 2020, pp. 95–104.

[9] K. Konstantoudakis and et. al., "Serverless streaming for emerging media: Towards 5g network-driven cost optimization," *CoRR*, vol. abs/2102.04910, 2021. [Online]. Available: https://arxiv.org/abs/2102.04910

[10] Kubernetes, "Kubernetes Controllers," 2021. [Online]. Available: https://kubernetes.io/docs/concepts/architecture/controller/

[11] ——, "Kubernetes Operator Pattern," 2021. [Online]. Available: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/

[12] ETSI, "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV – ETSI GS NFV 003 V1.3.1 (2018-01)," 2018.

[13] ——, "Network Functions Virtualisation (NFV) Release 3; Architecture; Report on the Enhancements of the NFV architecture towards "Cloud-native" and "PaaS" – ETSI GR NFV-IFA 029 V3.3.1 (2019-11)," Nov 2019. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/NFV-IFA/001_099/029/03.03.01_60/gr_NFV-IFA029v030301p.pdf

[14] 2021. [Online]. Available: https://github.com/IBM/GNFOrchestrator

[15] 2021. [Online]. Available: https://github.com/IBM/CompositionSDK

[16] N. W. Paper, "Network functions virtualisation: An introduction, benefits, enablers, challenges call for action. issue 1," Oct. 2012.

[17] ETSI, "Network Functions Virtualisation (NFV); Management and Orchestration – ETSI GS NFV-MAN 001 V1.1.1 (2014-12)," 2014. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/nfv-man/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf

[18] F. Paganelli, F. Paradiso, M. Gherardelli, and G. Galletti, "Network service description model for vnf orchestration leveraging intent-based sdn interfaces," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, 2017, pp. 1–5.

[19] Kubernetes community, "kubernetes.io," 2021. [Online]. Available: https://kubernetes.io

[20] The Kubernetes authors, "Application CRD and Controller," 2021. [Online]. Available: https://github.com/kubernetes-sigs/application

[21] Helm Community, "HELM: The Package Manager for Kubernetes," 2021. [Online]. Available: https://helm.sh/

[22] Crossplane authors, "Crossplane," 2021. [Online]. Available: https://crossplane.io

[23] Juju authors, "Juju," 2021. [Online]. Available: https://jaas.ai/how-it-works

[24] OperatorHub.io community, "OperatorHUb.io," 2021. [Online]. Available: https://operatorhub.io