The Orchestration for Cloud-Native Network Function in Open Source MANO

Chengcheng Li[1], Jikun Tao[2], Chao Zhu[2], Jie Zeng[2], and Yasheng Zhang[1]

[1] The 54th Research Institute of CETC, Shijiazhuang, 050081, China
lengcangche@bupt.cn, zys163@163.com
[2] School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, 100081, China {3220221493,chao,zengjie}@bit.edu.cn

Abstract. The evolution from Physical Network Functions (PNFs) to Virtual Network Functions (VNFs) and Cloud-Native Network Functions (CNFs) signifies a dynamic progression within virtualization technologies, addressing constraints in flexibility, prolonged deployment, and scalability. CNFs encapsulate functionalities in lightweight, containerized units, prioritizing cloud-native principles of agility and resilience. Effective CNF management is crucial for modern network infrastructures, and Open Source Management and Orchestration (OSM) frameworks attempt to address these challenges. However, certain limitations persist, particularly in large-scale applications with complex startup sequencing. These challenges are exacerbated by CNFs' modular nature and the volatile resource dynamics in cloud-native environments. To overcome these issues, we propose a Two-Tiered Network Service Orchestration (TTNSO) architecture, utilizing Helm for orchestrated deployment and a RESTful Information Exposure Interface within Kubernetes. TTNSO optimizes CNF orchestration, streamlining deployment, scalability, and resource allocation in OSM environments.

Keywords: MANO · NFV · Orchestration.

1 Introduction

The evolution from Physical Network Functions (PNFs) to Virtual Network Functions (VNFs) and further to Cloud-Native Network Functions (CNFs) signifies a dynamic progression within virtualization technologies [1]. In the era of PNFs, network functions were heavily dependent on dedicated hardware, leading to constraints in flexibility, extended deployment timelines, and difficulties in achieving scalable solutions [2]. Despite the transformative impact of VNFs, which uncoupled functions from hardware and enabled deployment on virtualized infrastructure, challenges persist in optimizing resource utilization, streamlining provisioning efficiency, and achieving seamless scalability [3].

The emergence of CNFs represents the culmination of virtualization progression. CNFs encapsulate functionalities within lightweight, containerized units,

aligning with cloud-native principles that prioritize agility, scalability, and resilience [4]. This paradigmatic shift facilitates accelerated development processes and continuous deployment practices [5].

The effective management and orchestration of CNFs are critical to harnessing their full potential in modern network infrastructures. Open Source Management and Orchestration (OSM), an open source Management and Orchestration (MANO) stack aligned with ETSI NFV Information Models frameworks, has played a pivotal role in addressing the challenges associated with CNF deployment, configuration, and scaling [6].

However, limitations persist in using OSM to orchestrate large-scale CNF applications. Deploying multi-component apps remains intricate due to complex dependencies and startup sequences. Post-deployment, uncertainties arise around capacity planning and load distribution of each CNF, demanding precise orchestration. For intricate multi-CNF apps, conventional deployment lacks agility for modern networks. The modular microservices nature of CNFs exacerbates this challenge. Meticulous coordination is needed to deploy such composite applications.

Moreover, the dynamic nature of cloud infrastructure introduces resource volatility. This necessitates adaptive orchestration and auto-scaling mechanisms. Ensuring optimal resource allocation across varying CNF demands is an intricate challenge conventional strategies struggle to address.

To address the aforementioned challenges, we propose a novel Two-Tiered Network Service Orchestration (TTNSO) architecture, that aims to enhance the deployment, management, and scalability of CNFs for complex applications in OSM environments. TTNSO architecture comprises two tiers for effective CNF orchestration. The first tier utilizes Helm, a Kubernetes package manager, to deploy multiple VNFs in precise startup sequences in OSM [7]. This addresses complex interdependent application compositions. The second tier, within Kubernetes, introduces a RESTful Information Exposure Interface (IEI) enabling dynamic CNF control. It facilitates dynamic scaling, graceful restarts, and more. Integrating both tiers bridges application deployment and runtime management, streamlining processes and enhancing scalability, fault tolerance, and resource efficiency.

The subsequent sections of this paper unfold as follows: Section 2 introduces pertinent concepts encompassing CNF, OSM, Helm, and RESTful API. In Section 3, an in-depth exposition of our TTNSO architecture is presented, accompanied by insights into the conducted experiments outlined in Section 4. The paper culminates with the conclusions in Section 5.

2 Backgrounds

In this section, the related concepts including CNF, OSM, Helm and are introduced in detail.

2.1 Cloud-Native Network Funcion

CNFs adopt a modular and microservices-oriented design, decomposing network functions into smaller, independently deployable components. This microservices approach provides greater flexibility to develop, update, and scale individual components. CNFs leverage containers and orchestrators like Kubernetes for automated deployment, scaling, and management.

Their lightweight containerized packaging allows rapid startup times and near instantaneous scaling of instances to handle fluctuations in traffic. CNFs emphasize the ability to elastically scale up or down on-demand, ensuring high availability and resiliency through replicating and distributing components.

CNFs aim to align network functions with cloud-native principles like agility, resilience, and resource efficiency. The cloud-native attributes of CNFs are key to supporting dynamic, 5G, and edge computing environments.

2.2 Open Source MANO

As an open source MANO platform, OSM plays a pivotal role in orchestrating and managing virtualized network workloads. It aligns with ETSI NFV Information Models to cover the entire VNF/CNF lifecycle.

OSM's capabilities include automated VNF onboarding, Day-0/Day-1 configuration, instantiation of VNF instances, elastic scaling, monitoring, and closed-loop remediation. It utilizes VIMs like OpenStack, VMware, and Kubernetes to interact with virtual resources and infrastructure.

The service orchestrator engine is a key component, stitching together end-to-end network services across diverse VIM environments. It handles multi-VNF composition, cloud onboarding, NFVI resource allocation, and Day-2 operations like scaling and healing.

While OSM provides sophisticated orchestration, challenges exist in deploying large-scale CNF applications with complex dependencies and startup sequences. The dynamic nature of cloud-native infrastructure also demands more intelligent and adaptive orchestration capabilities.

2.3 Helm

Helm is an open source package manager that streamlines deploying Kubernetes applications. Helm charts define dependencies between components, specify startup sequences, and encapsulate configuration details.

Helm handles intricate application deployments by orchestrating the release cycle, coordinating across components, and managing upgrades. Its declarative model and sequencing capabilities simplify deploying large, multi-CNF applications.

Integrating Helm's orchestration into OSM can enhance CNF deployment and lifecycle management. But runtime operations like scaling remain challenging in cloud-native environments, which TTNSO aims to address.

2.4 RESTful Information Exposure Interface

To enable dynamic and adaptive orchestration of CNFs, a RESTful Information Exposure Interface (IEI) can be introduced within each VNF/CNF.

This IEI exposes runtime information about the CNF to the orchestrator, including resource utilization, network/storage IO metrics, and request rates. The IEI provides a standard way for CNFs to report dynamic metrics and telemetry data.

The IEI follows a REST architectural style, providing resource-based APIs that return JSON/YAML payloads containing current runtime info. Kubernetes can leverage this info to make data-driven decisions around scaling, healing, and resource allocation.

By querying the RESTful IEI of each CNF, the orchestrator gains greater visibility into runtime dynamics. The IEI data powers advanced orchestration capabilities like Horizontal autoscaling of CNFs based on demand, predictive scaling and pre-emptive resource allocation, optimized placement and distribution of CNF instances, and traffic steering and request routing. Overall, the RESTful IEI enables closing the feedback loop between the orchestrator and application infrastructure. This allows dynamic optimization and adaptive management of CNFs in response to changing conditions.

3 TTNSO Architecture

The first tier of our TTNSO architecture leverages Open Source Management and Orchestration (OSM) as its foundation. Within this tier, we harness the power of Helm, a package manager for Kubernetes applications, to facilitate the orchestrated deployment of multiple Virtual Network Functions (VNFs) while ensuring precise startup sequencing. Helm's capability to define and manage CNF dependencies and startup orders addresses the intricacies of applications composed of interdependent components. By utilizing Helm's declarative approach, we can effectively coordinate the initialization of CNFs, mitigating the challenges of composite application deployment.

In the second tier, which operates within the Kubernetes deployment layer, we introduce a Restful Information Exposure Interface (IEI) as a pivotal component. This interface empowers the Kubernetes management layer to control various aspects of individual CNFs dynamically. Through this Restful IEI, functionalities such as dynamic scaling, graceful restarts, migrations, and other lifecycle management operations are made feasible. Kubernetes, renowned for its orchestration prowess, gains the ability to adjust the scale and state of CNFs in response to varying demands, ensuring optimal performance while maintaining resilience.

By integrating these two tiers, our TTNSO architecture tackles the intricacies of orchestrating CNFs within intricate applications. The coordination of CNF startup sequencing in the OSM tier, coupled with the dynamic control enabled by the Restful IEI in the Kubernetes tier, bridges the gap between composite

application deployment and runtime management. This comprehensive approach not only streamlines the deployment process but also facilitates agile scalability, fault tolerance, and efficient resource utilization.

4 TTNSO demonstration

In this chapter, the deployment of standalone 5G core network components serves as a case study to highlight the superiority of TTNSO in managing the lifecycle of complex network functions. The deployment of a standalone 5G core network involves intricate dependency relationships between multiple components.

OpenAirInterface (OAI) is an open-source initiative that holds a central position in fostering advancements in next-generation wireless network technologies. This paper employs TTNSO to deploy the OAI open-source project that aligns with 3GPP standards, showcasing the enhancements it brings to complex network function deployments based on OSM.

Conventionally, when deploying a Network Service (NS) with multiple Network Functions (NF) using OSM, all NFs initiate instantiation simultaneously. This simultaneous instantiation becomes problematic when deploying core network components, such as AMF, which require preceding dependencies. Failures occur due to dependencies not being prepared in time for deployment.

To enforce a strict orchestration order during component deployment, this paper enhances the NF instantiation process in OSM using Helm. At the Helm level, NF packages exist in the form of Charts. Under the umbrella of a parent Chart, each component's sub-Chart is assigned a weight value. This weight is defined within the configuration file of the parent Chart, as illustrated in the given figure. OSM strictly adheres to this weighted order to ensure a sequential instantiation of components. The complete NF package topology within OSM is depicted in To enforce a strict orchestration order during component deployment, this paper enhances the NF instantiation process in OSM using Helm. At the Helm level, NF packages exist in the form of Charts. Under the umbrella of a parent Chart, each component's sub-Chart is assigned a weight value. This weight is defined within the configuration file of the parent Chart, as illustrated in Table 1. OSM strictly adheres to this weighted order to ensure a sequential instantiation of components. The complete NF package topology within OSM is depicted in Figure 1

Instantiation marks the inception of the entire network function lifecycle. In the context of OSM, we refer to this phase as the Day-1 Operation. Subsequent runtime maintenance, termed Day-2 Operation, is another pivotal aspect ensuring the persistent and stable delivery of network services. OSM utilizes 'Charm' for Day-2 operations on services deployed on cloud platforms like OpenStack. However, its support for cloud-native services based on Kubernetes remains suboptimal. The complexity of code development and its lackluster performance have rendered Charm less than ideal for genuine use in cloud-native settings.

To realize Day-2 operations in a Kubernetes environment, we introduced a component, designed around the Information Exposure Interface (IEI) and the

Table 1. Weight and order of 5GCN components

Components	Order	Weight
AMF	1	4
SMF	7	6
UPF	6	5
NRF	1	0
AUSF	5	3
UDM	4	2
UDR	3	1
NSSF	2	0

Table 2. AMF Prometheus Metrics and HPA Configuration

Metric Name	**Description**	**HPA Rule Configuration**
`amf_request_rate_total`	Requests processed per second	>1000 requests/sec
`amf_response_time_average`	Average response time	>300 ms
`amf_cpu_usage_percentage`	CPU utilization rate	>70%
`amf_concurrent_sessions`	Current number of sessions	>1000 sessions

Horizontal Pod Autoscaler (HPA). This component automatically detects load and conducts scaling actions accordingly.

In the core network components discussed in the preceding section, we designed four performance indicators for the Access and Mobility Management Function (AMF) to serve as data references for HPA. These indicators are depicted in Table **??**. Proactive instrumentation within AMF has been integrated to collect the necessary data, which is subsequently exposed to HPA through IEI.

During the phase of data collection for indicators, the Time Series Database (TSDB) is employed to gather and store the data provided by the IEI. By organizing and storing the continuously fed metric data from the NF's operational state with time stamps, TSDB significantly enhances the detection efficiency of the HPA.

In the metric detection phase, the HPA generated during the NS initialization phase polls the associated metric data from TSDB at fixed intervals. This data serves as input to a decision tree generated based on predefined rules. The output from this tree dictates the subsequent actions the HPA should undertake. In essence, the HPA aims to maintain the metrics as close as possible to a desired value, either by scaling out or scaling in.

As previously discussed, the deployed AMF component by default instantiates two replicas. These replicas share service requests evenly through the load balancing provided by Kubernetes. As indicated in the table, our objective is to maintain the ratio of *amf_concurrent_sessions/replica* below 1100,

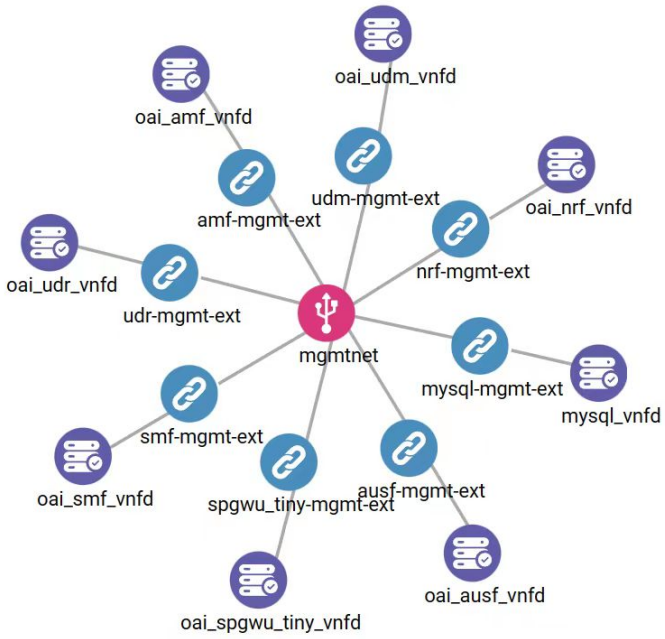

Fig. 1. 5g core network component topology

meaning each replica upholds a maximum of 1000 sessions concurrently. During our experimentation, when we simulated an increased session request leading to a *amf_ concurrent_ sessions* value of 4028, the HPA's scaling-out action was triggered. In an impressively short span of time, two instances, identical to the pre-existing AMF instances, were created. This adjustment brought the *amf_ concurrent_ sessions/replica* ratio back to approximately 1000, aligning with our expectations.

5 Conclusion

References

1. Bernstein, D.: Containers and cloud: From lxc to docker to kubernetes. IEEE cloud computing **1**(3), 81–84 (2014)
2. Mijumbi, R., Serrat, J., Gorricho, J.L., Bouten, N., De Turck, F., Boutaba, R.: Network function virtualization: State-of-the-art and research challenges. IEEE Communications surveys & tutorials **18**(1), 236–262 (2015)
3. Addis, B., Belabed, D., Bouet, M., Secci, S.: Virtual network functions placement and routing optimization. In: 2015 IEEE 4th International Conference on Cloud Networking (CloudNet). pp. 171–177. IEEE (2015)

4. Hirai, S., Tojo, T., Seto, S., Yasukawa, S.: Automated provisioning of cloud-native network functions in multi-cloud environments. In: 2020 6th IEEE Conference on Network Softwarization (NetSoft). pp. 1–3. IEEE (2020)
5. Knafl, K.A., Deatrick, J.A., Havill, N.L.: Continued development of the family management style framework. Journal of Family Nursing **18**(1), 11–34 (2012)
6. Karamichailidis, P., Choumas, K., Korakis, T.: Enabling multi-domain orchestration using open source mano, openstack and opendaylight. In: 2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN). pp. 1–6. IEEE (2019)
7. Yilmaz, O.: Extending the kubernetes api. In: Extending Kubernetes: Elevate Kubernetes with Extension Patterns, Operators, and Plugins, pp. 99–141. Springer (2021)