Introduction

Network Function Virtualization (NFV) and Cloud native network functions (CNF) continue to draw immense attention from the telecom sector. From the beginning, virtualization is trying to fulfill the promises of reducing CAPEX (Capital Expenditure) and OPEX (operating expenses) by decoupling Network Functions (NFs) from the underlying infrastructure and ensuring flexibility and scalability. However, despite the huge traction it has gained, there are still obstacles to overcome before it can be part of day-to-day operations.

One of the important challenges associated with this transformation is to implement the process for management and orchestration of network functions. Telcos need to build virtualized network functions while maintaining a high quality of service (QoS) and keeping up with a constantly evolving technology landscape. The 5G stage will likely last for the next 5-8 years before the transition to 6G, and so on. The switch to a new generation network is costly, and if the average revenue per user (ARPU) does not grow as fast as Capex, telcos will not be able to maintain profitability.

To cut the costs and in the same time improve security and quality, companies can turn to open source initiatives such as Open Source MANO (OSM), Open Networking Foundation (ONF), Open Radio Access Network (O-RAN), and OpenStack, etc. However, the integration of open source resources poses other challenges. If telcos generally appreciate the lower costs and the no vendor lock-in aspects of open source, the lack of support and interoperability with their solution can be seen as problematic. Also, these various open-source technologies often present single components, whereas telcos prefer e2e solution providers. This is what sets the Canonical open source stack apart, providing a model-driven approach and making the integration between all the components much straightforward and effortless.

In this whitepaper, we will introduce how to build network functions and onboard them to the MANO (management and network orchestration) framework that has been proposed by the European Telecommunications Standards Institute (ETSI) to meet challenges of interoperability and support. Also, by considering the fact that network function is not alone and it has many of its non functional requirements to make it ready for a production environment, we will see how we can fulfill these requirements like monitoring, logging, scaling and upgrades etc. by proposing a model-driven operations based complete open source solution instead of single components.
Network functions

Network functions are defined as one or many nodes in a network infrastructure that have well-defined interfaces and functional networking capability. The services provided by telcos are highly dependent on the use of functions that include routers, switches, firewalls, as well as algorithms to receive and forward calls. These services are provided primarily through physical devices (physical network functions) but the use of Virtual Network Functions (VNFs) has also increased over the past few years. While VNFs offer certain advantages, they can be enhanced by the use of Cloud-Native Network Functions (CNFs) that employ container and cloud technologies to ease operations such as update, upgrade, scaling, monitoring, logging, etc.

- **Physical Network Functions**: PNF refers to an actual piece of networking hardware such as a router, switch, firewall, etc. In a PNF, the hardware and software are tightly bound to each other. PNFs are widely deployed and are preferred when the consumer can afford vendor lock-in and does not want to get involved with the installation of the network.

- **Virtual Network Function**: VNF refers to the software implementation of a networking function/module. These allow the features of the PNF to be replicated on software. VNF runs on Virtual Machines (VMs) that allow it to be run on commodity hardware, essentially eliminating vendor lock-in. VNFs are relatively new as compared to PNFs and are not as widely deployed. These are ideal for scenarios where the consumer wants to have control over installation and deployment. They are also preferred when portability/migration is required.

- **Cloud-native network functions**: CNF refers to containerizing a PNF or VNF and running it on the cloud. This includes creating microservices that communicate with each other to form a network. These microservices run on docker containers and management of the containers is done using Kubernetes (k8s) as well as the CI/CD principles in a cloud architecture. CNF and KNF (Kubernetes-based network functions) are used interchangeably in the NFV domain as both are based on cloud-native concepts. CNFs are preferred for scenarios where the consumer requires added features to VNFs such as elasticity, scalability, availability, and ease of installation and deployment. All of these features are enabled by cloud and container technologies.

In this paper, we will be focusing on the virtualized and containerized network functions.

<table>
<thead>
<tr>
<th>Type</th>
<th>Key Features</th>
<th>Ideal Usage Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>PNF</td>
<td>Tightly bound software and hardware</td>
<td>When installation from the vendor is desired</td>
</tr>
<tr>
<td>VNF</td>
<td>Virtual implementation of PNFs</td>
<td>When eliminating the dependency on dedicated hardware</td>
</tr>
<tr>
<td>CNF</td>
<td>Containerized implementation of VNFs/PNFs</td>
<td>When microservices architecture and DevOps are a major requirement</td>
</tr>
</tbody>
</table>
Network function onboarding with Open Source MANO (OSM)

ETSI OSM is an operator-led ETSI community that is delivering an open source Management and Orchestration (MANO) stack aligned with ETSI NFV Information Models and that meets the requirements of production NFV networks. It helps accelerate your migration to NFV with network function onboarding and then after migration ensuring the automated orchestration of the network functions using Day-2 operations.

Design patterns and properties

The design decisions for making the network functions cloud-ready and resilient before onboarding are key elements to achieve the desired performance. Here is a list of the main elements needed to set up network functions before orchestration, as described in NFV-SWA 001.

- **Infrastructure planning**: The infrastructure on which the network function can be best deployed and give expected results. It can be bare metal, kubernetes clusters, and virtual machines.

- **Choice of data models**: Network functions need to be described in a standardized way for supporting multiple virtual infrastructure. These data models are provided by different NFV standardization institutions to meet the challenges of interoperability between NFV components.

- **Network function evaluation**: Network function needs to be tested under specific criteria for compute network and storage resources of infrastructure and it should keep delivering the same KPI (Key Performance Indicators) under different circumstances.

- **Network function internal structure**: To have a better microservices experience, separate the functionality into different VMs and containers according to the requirements of the network function.

- **Network function instantiation**: Choose your instantiation methodology for failover scenarios, some network functions can best work with parallel instances and some with non-parallel instances according to the effect it has on the functionality.

Operations management: operations include monitoring, logging, scaling, auditing, upgrades, and security, etc. To achieve exceptional performance, these Day-2 operations should be automated as manual management can quickly become inadequate.

Onboarding Stages

Once the prerequisites are met systematically, the onboarded network function should now aim to fulfill the lifecycle stages to function properly, which then, the NFV MANO layer will automate. The on-boarding process includes the creation of descriptor files that define how to initialize/terminate, scale, and monitor the network functions. The overview of stages for onboarding is as follows:
• Create the network function descriptors as templates to instantiate/terminate, scale, update or upgrade.

• Create packages for network services i.e. templates for images and configurations of the virtualized infrastructure using Enhanced Platform Awareness provided by OSM.

• Configure the day-1 and day-2 operations or actions according to the required services.

• Validate and load into the service catalog of OSM.

All of the stages for onboarding NFs (network functions) can be achieved through OSM by creating a NF package that specifies Day-0 (basic instantiation), Day-1 (service initialization), and Day-2 (runtime operations) configurations. The Day-1 and Day-2 configurations are described in scripts called charms to run operations (refer day-1 for more details) according to the functional requirements. These charms are then integrated into the OSM descriptors.

The details for each stage is given below:

Day 0

The Day-0 configuration refers to the instantiation of the network services, or set of network functions, and the initial parameters for infrastructure setup that will be needed later in the onboarding process.

• Build an initial package compliant with the ETSI SOL006 standards and populate the basic parameters for the instantiation of network service.

• After generating the network function descriptor (NFD) and Network service descriptor (NSD) in packages define the connection points parameters through which the network function components are interconnected.

• Integrate the cloud-init scripts in the descriptors for primary configurations needed to build up a network service like the OS boot requirements, setting up a hostname, adding SSH keys, configuring network devices and users.

• OSM supports the EPA (enhanced platform awareness capabilities) in its descriptors, so this is time to enable Hugepages, CPU pinning, SR-IOV, or any other data accelerated features in your network functions.

Day 1

OSM component VCA (VNF configuration and abstraction) is responsible for the day-1 and day-2 operations and the driver behind the VCA is Juju. Juju manages the charms and charms are the operations code inside the network function descriptors. These are a set of scripts for deploying and operating another software, network function in our case.
OSM supports two types of charms, native and proxy charms, which should be chosen according to the nature of the network function (refer to figure below). For a proxy charm, if you are using a fixed image for your NF which cannot be modified, then the charm has to be allocated not in the NF but in a different LXD (Linux container manager) or K8s cloud. However, in the case of native charm, it can be modified, and the charm should be deployed in the same workload or NF.

Day-1 configurations initiate the service and ensure the expected functionality from the network functions. The process includes:

- Create a charm (proxy or native) as an operator for the network function.
- Define the primitives/actions in charms that need to be performed right after the instantiation of service i.e.
  - Initial charm configuration
  - Authentication to configure remote in case of proxy charms
  - Actions required after the service is up and running etc.
- Integrate the configured charm in the OSM descriptors
Day 2

As described above, Juju is responsible for the day-2 operations and the mechanism for including the primitives is the same as discussed in the day-1 configurations process. The difference will be in the nature of primitives, as the day-2 actions are more focused on the runtime operations or the configuration required after instantiations of the network service which can be the following:

- Reconfigurations needed after the service is running
- Monitoring of the specific metrics for the infrastructure
- Scaling on the basis of monitoring analysis
- Operations to enable closed-loop automation.

Now the network function is onboarded and managed by OSM. But, the challenge here is that the requirements of network function are not only to get onboarded in an automated way and perform the intended functionality. There are some other non-functional requirements like monitoring, logging, auditing of the network function that can also be done in simpler ways using Juju relations as discussed in the next sections.

Non-functional requirements of the network functions

The real benefit of having the charms and following the standardization by ETSI can be best utilized here when we need compatible and automated tools for fulfilling the non-functional requirements of the network function. These non-functional requirements can be monitoring, scaling, logging, and security, etc. Although there are multiple open-source tools for the LMA (Logging, Monitoring, and Alerting) of NFs like Prometheus, Grafana, and Graylog which can serve the purpose but reconfiguring these tools with every update in your network functions can be tiresome.

As the application modeling tool, Juju offers model-driven open-source tools which can be easily integrated with the network functions by using juju relations. So if the charmed network function has been deployed, it can now be integrated into the other open-source monitoring tools by relating it with the charms of required tools which is further explained below.

Logging, monitoring, and alerting (LMA)

The network function is now operator-driven and onboarded to OSM. For the continuous monitoring and logging of the network function, a model-driven open-source LMA infrastructure is needed. This makes it more convenient to re-use the same models instead of reconfiguring the whole LMA stack when the NF is updated. It can be done using Juju relations. Relations are a communication channel between charms and provide an abstraction that enables applications to interoperate.
Another benefit of juju relations is the reusability and seamless integration of new components. If the network function gets scaled up and more instances need to be added, then the same LMA stack can be used and there is no need to reconfigure the whole LMA stack for the newly added units in the juju model.

Scaling

Scaling refers to increasing or decreasing resources to match the demands of the user or application. Juju supports horizontal scaling, which involves increasing the number of VMs or containers in a particular network function. It enables the scaling of onboarded network functions with just a single command. The following mechanism will create an additional three units to meet the demands of network function.

Security

Despite all the well-recognized benefits of network functions, including CAPEX reduction and network efficiency, security remains a major concern of network service providers.

Security can be provided on multiple levels in any solution. The security at the operating system level can be easily achieved by deploying the network function on an OS with regular security updates like Ubuntu. Extended Security Maintenance (ESM) support in Ubuntu LTS systems receives patches for high and critical CVEs (Common Vulnerabilities and Exposures) in the Ubuntu base OS and scale-out infrastructure.
For security on the application layer for example to ensure the secure functionality of network functions different open source intrusion detection systems can be used as charms for Snort, Zeek and Suricata, etc., and integrate them with your network function. The deployment and integration will be the same as described above for the LMA stack.

Use Case: Charmed 5G testbed orchestration with Charmed OSM

The telecom industry is advancing rapidly from bare metal toward 5G cloud-native networks. To ensure the effectiveness of network function virtualization in 5G, the network functions should be virtualized or containerized in the best possible way so that they can be modified, managed, monitored, and optimized with minimum effort.

TATA Elxsi has immense expertise in delivering flagship programs in Wireless, 5G, and is a strong community member in Open Source MANO. TATA, in collaboration with Canonical, has deployed the 5G testbed on Charmed OSM to test the performance of the Canonical Juju operator framework in configuring and monitoring the 5G CNFs (cloud-native network functions).

Charmed OSM is an upstream Open Source MANO distribution, developed and maintained by Canonical, which uses Juju charms to simplify its deployments and operations. Charmed OSM enables TSPs to easily deploy pure upstream Open Source MANO in highly available, production-grade, and scalable clusters.

Solution overview

The functional ability of the Juju controller and Charmed OSM made possible the deployment of 5G Kubernetes network functions (KNFs) on MicroK8s, infrastructure. All the components, i.e. Radio Access Network (RAN) and 5G Core, are based on a model-driven operator framework that can be reused, upgraded, and integrated according to the current demands. The functional features include:
- 5G Core, 5G RAN Emulator and IMS onboarding using Charmed OSM
- 5G E2E Network Service chaining (RAN Emulator to Core to IMS)
- the Day 1 and Day 2 configuration using generic VNF manager (Juju Controller & Charms),
- Juju controller Charm-based KNF Monitoring and Grafana visualization.

The use cases validated as a part of the deployment included the PDN (Packet Data Network) connectivity, UE (User Equipment) IP acquisition, and Voice SIP (Session Initiation Protocol) calls.

The components details are as follows:

**Open source Components**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5G Core</td>
<td>Free5GC</td>
</tr>
<tr>
<td>IMS</td>
<td>Kamilio</td>
</tr>
<tr>
<td>5G RAN Emulator</td>
<td>Free5GC</td>
</tr>
<tr>
<td>Grafana</td>
<td>Grafana labs</td>
</tr>
<tr>
<td>Web client</td>
<td>Jmeter</td>
</tr>
<tr>
<td>SIP client</td>
<td>PJSIP</td>
</tr>
</tbody>
</table>

**Canonical’s distributions**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charmed OSM</td>
<td>ETSI MANO Orchestrator</td>
</tr>
<tr>
<td>Juju framework</td>
<td>Deployment and configuration manager</td>
</tr>
<tr>
<td>Microk8s</td>
<td>Cloud infra manager</td>
</tr>
</tbody>
</table>
Open-source ecosystem for network function builders

Open-source technologies help in reducing the CAPEX for shifting between technologies but telcos and mobile operators need a full solution provider. As we discussed before, the network function is not alone, it needs non-functional requirements to help operators prepare it for deployment and orchestration.

Canonical is focusing on the full-stack solution combining the required open source technologies for building up the network functions effectively. It provides all the necessary elements from the bottom up, Ubuntu for the OS, LXD clustering to provide an abstract layer of virtualization, Ceph for distributed storage, and MicroK8s and micro clouds to deliver the workloads as containers or machines. On top of all, Charmed OSM covers the management layer for the deployed applications. All of these support model-driven approaches using Juju and charms that can be deployed and reused according to the network functions requirements.

Other than provisioning the managed infrastructure for network function onboarding, Canonical offers an observability stack for non-functional requirements like Prometheus for metric collections, Grafana for metrics analysis, Graylog for auditing logs, and Filebeat to aggregate logs from different components or databases. So, in parallel to developing innovative and advanced open-source tools, Canonical is also focusing on providing operators that simplify deployment and operations of the existing ones.

References:
5. https://osm.etsi.org/docs/vnf-onboarding-guidelines/02-day0.html
8. https://juju.is/docs/sdk
9. https://juju.is/docs/relations

© Canonical Limited 2021. Ubuntu, Kubuntu, Canonical and their associated logos are the registered trademarks of Canonical Ltd. All other trademarks are the properties of their respective owners. Any information referred to in this document may change without notice and Canonical will not be held responsible for any such changes.