WHITE PAPER

Communications Service Providers
VOLTE

VOLTE Orchestration and Validation Reference Architecture

Reference design validates the functionality of Tata Consultancy Service's TCS NFV Concerto™ on Intel®-powered servers with technology from Brocade,* Metaswitch,* Ixia,* and Red Hat.*

TATA
TATA CONSULTANCY SERVICES
Experience certainty.

Audience and Purpose

Telecommunications networks are witnessing a technology paradigm shift with the evolution of network functions virtualization (NFV). Adopting NFV significantly increases service agility and reduces operational expenses. In addition, with most communications service providers (CommSPs) moving to long term evolution (LTE) for data and IoT scenarios, successful voice over LTE (VOLTE) service deployment is emerging as a critical business necessity. The key challenge is successful migration or expansion from physical to virtual LTE mobile core networks, and successful deployment and verification of VOLTE services without compromising customer quality of experience (QoE). This virtualization enables operators to introduce new services more rapidly and compete with "over-the-top" (OTT) competitors more effectively. Therefore, testing of VOLTE services in a virtual LTE mobile core network must take place across the entire service lifecycle including evaluation, field-trial, commercial deployment, and optimization phases. Timely verification of these phases is vital to ensure scalability and interoperability. This technical reference architecture document describes how verification can be achieved with TCS NFV Concerto™ from Tata Consultancy Services (TCS) with integrated Brocade* virtual Evolved Packet Core (vEPC), Metaswitch's* virtual IP Multimedia Subsystem (vIMS), and Ixia's* IxLoad* Virtual Edition (VE). This complete virtual VOLTE solution is deployed on Intel® Xeon® processor-based off-the-shelf servers, and leverages Red Hat's* OpenStack* platform for virtual infrastructure manager (VIM) functionality.

Executive Summary

CommSPs are finding it difficult to keep pace with growing customer demands because of their existing siloed and proprietary networks. This is because network elements have traditionally been optimized for high packet throughput at the expense of flexibility, thus hampering the development and deployment of innovative new services. Another concern is that rapid advances in technology and services are accelerating the obsolescence of installed hardware; and in turn, hardware isn't keeping up with other modes of feature evolution, which constrains innovation in a more network-centric, connected world. With existing networks, scalability is limited and deployment is often sluggish as expensive new equipment must be acquired and provisioned. Staffing costs escalate, as increased expertise is needed to design, integrate, operate, and maintain the various network function appliances. All of these issues make it difficult to innovate and compete. Moreover, OTT competitors and app developers are deploying new services with minimal deployment timelines. All of these factors are driving CommSPs to improve service deployment cycles.

Table of Contents

Network functions virtualization (NFV) can provide the infrastructure flexibility and agility needed to successfully compete in today's evolving communications landscape. Top drivers for NFV are new agility for service deployments for increased revenue, in addition to new operational efficiencies and reduced capital expenses stemming from the use of commercial servers rather than special-purpose network equipment. This virtualized approach decouples the network hardware from the network functions and results in increased infrastructure flexibility and reduced hardware costs. Because the infrastructure is simplified and streamlined, new and expanded services can be created quickly and with less expense.

Intel and TCS are collaborating to demonstrate the technical and business viability of NFV deployment and service orchestration using cloud computing technologies. The collaboration delivers a reference architecture and proof of concept (PoC) that demonstrates a novel NFV-based orchestration solution for a CommSP's cloud-based VOLTE network. Any mobile network operator wanting to migrate to the cloud and operationalize a VOLTE solution in its end-to-end network is a potential customer for this solution.

Reference Architecture Participants

Intel

The servers used in the reference architecture depend on the processing power and network and virtualization capabilities of the Intel® Xeon® processors E5-2600 and 10 Gigabit Ethernet Intel® technology. Intel architecture CPUs provide CommSPs with a standard, reusable, shared platform for NFV applications that is easy to upgrade and maintain. Intel architecture-based servers, enable MNOs to take advantage of the proven scalability of modern, virtualized computing technology. Advantages of this approach include a streamlined network and cost savings from hardware reusability and power reduction. Particularly, data plane processing has been greatly accelerated by open source Data Plane Development Kit optimization techniques originally developed by Intel.

TCS

Tata Consultancy Services is an IT services, consulting, and business solutions organization that delivers real results to global business, ensuring an amazing level of certainty. TCS offers a consulting-led, integrated portfolio of IT and IT-enabled infrastructure, engineering, and assurance services. This is delivered through its unique Global Network Delivery Model™, recognized as the benchmark of excellence in software development. A part of the Tata Group, India's largest industrial conglomerate, TCS has a global footprint and is listed on the National Stock Exchange and Bombay Stock Exchange in India. For more information, visit www.tcs.com.

Brocade

Brocade Communications Systems Inc. is an American technology company specializing in data and storage networking products. Originally known for its leadership in fibre channel storage networks, the company has expanded its focus to include a wide range of products for new IP and third-platform technologies. Brocade provides a full-function evolved packet core (EPC) designed from the ground-up for a virtualized environment that consists of independent slices of control, data, and session management.

Ixia

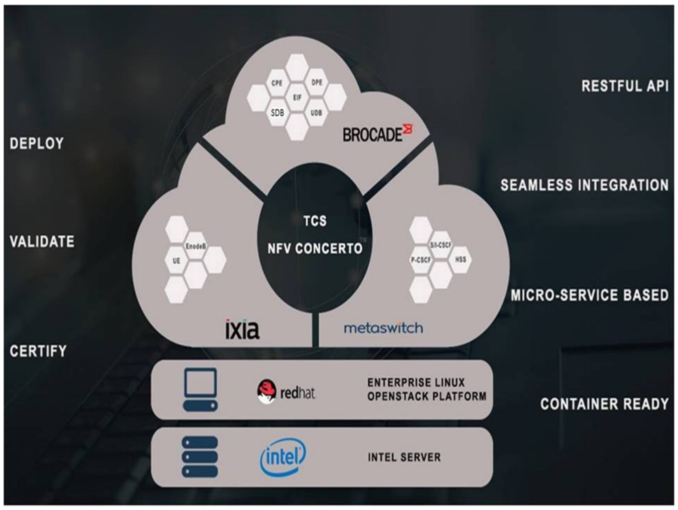
Ixia provides testing, visibility, and security solutions, strengthening applications across physical and virtual networks for enterprises, service providers, and network equipment manufacturers. To support network operator transformation and speed the introduction of new revenue-generating services, Ixia provides both physical and virtual solutions for SDN, NFV, IoT, Wireless, and DevOps. Ixia plays an active role in the ETSI (NFV) Industry Standards Group and is broadly involved in leading industry organizations focused on standards, open source, and new technologies for operators.

Metaswitch

Metaswitch, a cloud native communications software company with award-winning solutions, develops commercial and open-source software solutions that are constructively disrupting the way that service providers build, scale, innovate and account for communication services. By working with Metaswitch, visionary service providers are realizing the full economic, operational and technology benefits of becoming cloud based and software-centric. Metaswitch's solutions are powering more than 1,000 service providers in today's global, ultra-competitive, and rapidly changing communications marketplace. For more information, please visit www.metaswitch.com.

Red Hat

Red Hat plays a leading role in working with industry leaders to deliver carrier-grade OpenStack solutions. They have a proven record in developing commercial ecosystems around an open infrastructure platform, fostering a strong network of technology providers to deliver enterprise features needed for mission-critical deployments.



Overview of vVOLTE NFV Use Case

The core components that are necessary to facilitate a VOLTE call include user equipment (UE), enhanced NodeB (eNB), evolved packet core (EPC), and IP multimedia subsystem (IMS).

- The UE is the user's mobile device, which is subscribed to the VOLTE services and is used to place a call request. The typical UE is a mobile handset.
- The eNB is the radio access component of an LTE system that is based on universal mobile telecommunications system (UMTS). Each eNB contains at least one radio transmitter, receiver, control section, and power supply. In addition to radio transmitters and receivers, eNBs contain resource management and logic control functions. This capability allows eNBs to directly communicate with each other.
- The EPC communicates with packet data networks in the outside world such as the Internet, private corporate networks, or the IP multimedia subsystem. The interfaces between the different parts of the system are denoted Uu, S1, and SGi.
- The IMS has two primary functions: the call session control function (CSCF) that is responsible for the SIP session setup and tear down, and the home subscriber server (HSS) that is responsible for provisioning, authentication and location services.

These, along with policy and charging rules function (PCRF), are needed to provide the end-to-end architecture to work with the EPC and other IP networks. Online charging systems (OCS) and offline charging systems (OFCS) collect and present the data needed to charge for services as a part of session management. IMS uses the session initiation protocol (SIP) for session setup and teardown while DIAMETER signalling is used as the authorization, authentication, and accounting (AAA) protocol. The IMS provides access independence with the IMS core network serving as a common 'glue' layer for access aggregation for service delivery over various access media—Wi-Fi, broadband, LTE, and others as they evolve.

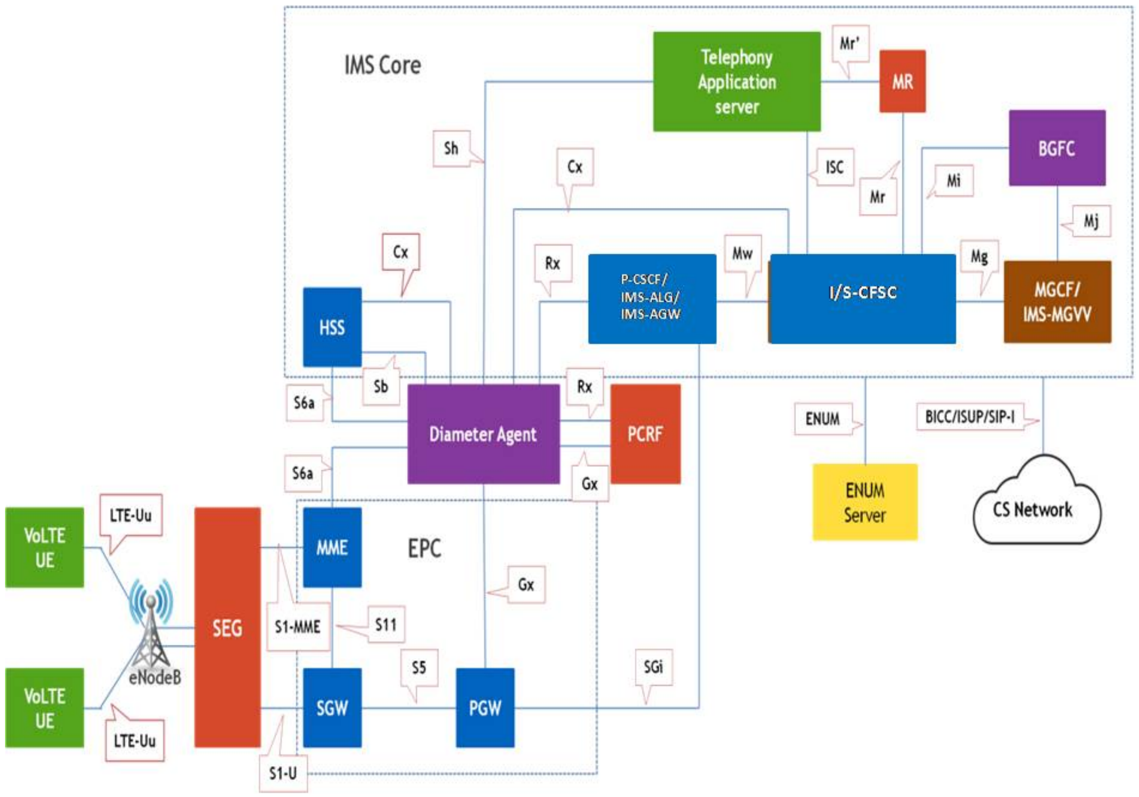


Figure 2. Description of vVOLTE Core. Blue blocks indicate the elements in the reference architecture that are orchestration and test targets

The European Telecommunications Standards Institute (ETSI) use cases #5 and #6 describe the virtualization of the mobile core and virtualization of mobile base station, including the EPC and the IMS elements. The goal of this virtualization is to achieve reduced total cost of operation (TCO), efficient and flexible resource allocation, scale to achieve higher availability and resiliency, and dynamic network reconfiguration. In addition, this can allow for dynamic reallocation of resources from one service to another to address spikes in demand in a particular service (e.g., a natural disaster or other major event). These virtualized solutions will have to coexist with legacy systems for some time as most operators will have mixed environments—NFV-based deployments and legacy equipment—for many years. Any or all of the IMS elements—in any of the core network, application, or transport layers are good candidates for virtualization.

Key Reference Architecture Components

TCS NFV Concerto

TCS' NFV Concerto is a comprehensive framework and a complete validation platform that provides end-to-end VOLTE verification, in a virtual LTE mobile core, with fully automated orchestration. The following are the salient features of TCS NFV Concerto.

- Comprehensive verification and validation of functional, system, performance, load, and resilience features of a VNF
- Vendor neutrality
- Enablement of isolated certification sandbox environments
- Zero-touch execution
- Pre-Integrated Industry standard test frameworks
- Provisioning DevOps interface for closed-loop validation of network functions

Key Features

The following are the key feature of NFV Concerto:

Reference Implementations: TCS NFV Concerto comes with reference implementations for key components, Network function virtualization orchestrator (NFVO) and virtual network function manager (VNFM) in the ETSI MANO stack. Since NFV Concerto comes with the reference implementations for key components in the ETSI MANO stack, it thus has the industry standard orchestration and VNFM engines that take care of end-to-end network orchestration of the VNF components.

Modular Framework: The inherent framework modularity brings in innovation and intelligence from any layer of the ETSI stack while offering the flexibility, when needed, to fall back and use the industry standard reference implementations already in place. Any of the ETSI MANO stack can be plugged in and out of the framework.

Vendor Neutrality and Interoperability: TCS NFV Concerto is a complete vendor-neutral platform that deploys, manages, and validates virtual network functions (VNFs) that belong to multiple vendors, making the platform interoperable and providing the flexibility to orchestrate a complete network service involving components from across multiple vendors.

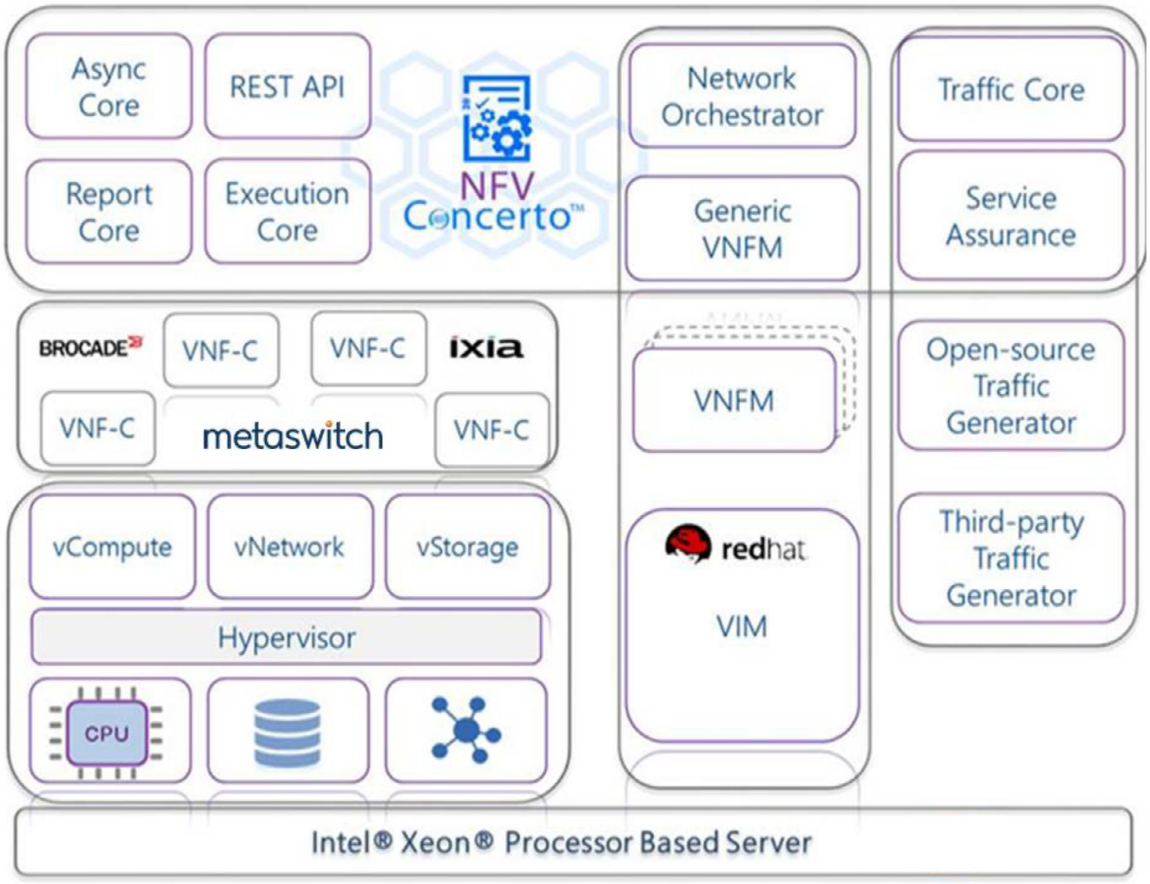


Figure 3. TCS NFV Concerto Block Diagram

TCS OpenVNFManager

The ETSI NFV MANO specification for NFV Service Orchestration Framework details the VNF manager as the entity responsible for lifecycle management of the virtual network functions. Each VNF instance is associated with a VNF manager. A VNF manager may be assigned the management of a single VNF instance, or the management of multiple VNF instances of the same or different types. TCS OpenVNFManager is an open-source, automated service orchestration framework for NFV. It encompasses NFV orchestration and lifecycle management and is fully compliant with the ETSI MANO specification and works with the OpenStack REST API. The solution is completely vendor neutral and self-installing, requiring minimal pre-configuration. It is a scalable and modular framework that interoperates with existing service orchestration solutions via standard OpenStack-like northbound REST API, enabling fully automated service provisioning. With this solution, both TEMs and CommSPs gain the responsiveness and service agility needed to meet customer demands. OpenVNFManager manages VNF occurrences via instance-specific plug-ins that communicate with the VNF instance over NETCONF, SNMP, or proprietary interfaces. Each VNF manager is capable of managing multiple VNF instances of the same or different type with the help of the plug-in framework. A sample plug-in is provided in the OpenVNFManager github for reference.

Key Features

Though the ETSI NFV MANO reference architecture does not mandate any specific realization of the NFV MANO architectural framework, it recommends leveraging a few best practices. Key best practices and their relevance in the current framework are elaborated on below.

The architectural framework should

- Lend itself to virtualization and be implementable in software only.
- Lend itself to possible distribution and scaling across the NFVI in order to improve service availability and support different locations.
- Lend itself to full automation (ability to react to events within reasonable time delays without human intervention, and execute actions associated with those events, based on pre-provisioned templates and policies).
- Lend itself to implementations that do not contain any single points of failures with the potential to endanger service continuity.
- Lend itself to an implementation with an open architecture approach, which includes exposing standard or "de-facto" standard interfaces.
- Support and help realize the feasible decoupling of VNF software from hardware.
- Support management and orchestration of VNFs and network services using NFVI resources from a single or across multiple NFVI points of presence (PoPs).
- Support modelling of NFVI resource requirements of VNFs in an abstracted way.
- Support modelling of NFVI resources so that they can be abstracted with the resources being exposed by functionality in one layer, to functionality in a different layer.
- Support service assurance by correlating multilayer fault, performance, logging & service impact analysis. This feedback is available to the policy management component of the orchestration system for recovery.
- Integrate with DevOps for closed-loop validation of network functions.

OpenVNFManager is a software-based VNF management framework that is scalable within a NFVI-PoP. It has a centralized component – VNF service daemon (vnfsvc) and a distributed VNFManager component that makes this possible. The framework enables automated service creation and life-cycle management of the VNF instances. It exposes standard OpenStack-like REST APIs. The NFV resource requirements and other parameters are captured in service descriptors as advised in the ETSI NFV MANO reference documentation. Sample descriptors are added at the end of the document for reference.

Open Platform for NFV Orchestration: OpenVNFManager is open source software under the Apache 2.0 license and this solution is available on github. The objective is to promote a truly open and vendor neutral platform for service orchestration.

Complements OpenStack: The solution does not by-pass any features of OpenStack. It works in coordination with OpenStack in the role of the Virtual Infrastructure Manager. It provides OpenStack-like northbound REST APIs to enable interoperability with other orchestration applications and systems.

Standard telco interface-based VNF configuration: The solution offers configuration of the VNF using NETCONF, a standard interface protocol that is widely used in CommSP environments. For VNFs that do not support this interface, custom drivers can be used for the VNF configuration and life-cycle management.

Architectural Details

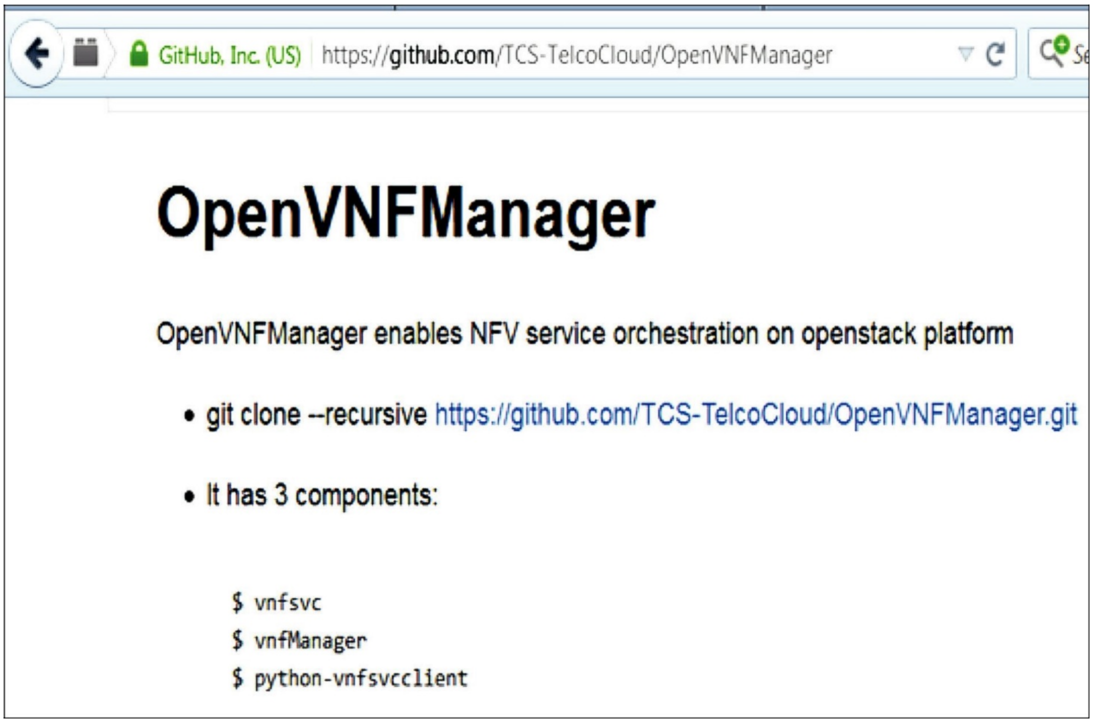
Key steps in service orchestration via the OpenVNFManager framework are as follows:

Service On-boarding

For a system to enable service creation, it is important to have all the required resources readily available. For a VNF, the VNF descriptor elaborately describes the configuration and dependencies. Similarly for a network service, the network service descriptor (NSD) describes the topology, VNFs required, service end-points, and policies applicable to that service. In order to ensure that the service creation is performed properly, these dependencies are validated before the VNF is made available in the system. This is called service on-boarding and it is enabled by a REST API.

In this phase, the descriptors are validated and the dependencies are verified by the system and services are made available in the catalogue. Successful on-boarding indicates that the service is available for deployment. The descriptors supported by this framework are ETSI-compliant YAML-based network service descriptors and virtual network function descriptors. Sample descriptors are provided in the examples section of Appendix B.

During the on-boarding phase, the descriptor is parsed to validate the availability of the required dependencies. For example, the image files to launch the virtual network functions etc. The system also creates required references in the system for the network service being on-boarded. The database is updated with the identifier for the service descriptor. Service on-boarding is performed by the administrator. Once the on-boarding is successful, a user can request service creation. On-boarded services are made available via a service catalogue or exposed via REST APIs. OpenStack Heat* enables the orchestration of services that are successfully on-boarded into the system. As mentioned in the help documentation, the templates.json file should be updated with the relevant descriptor information. For more details please refer to Appendix B for setup and installation instructions.



OpenVNFManager

OpenVNFManager enables NFV service orchestration on openstack platform

- git clone --recursive https://github.com/TCS-TelcoCloud/OpenVNFManager.git
- It has 3 components:

```
$ vnfsvc
$ vnfManager
$ python-vnfsvcclient
```

Figure 4. OpenVNFManager GitHub Screenshot

Service Provisioning

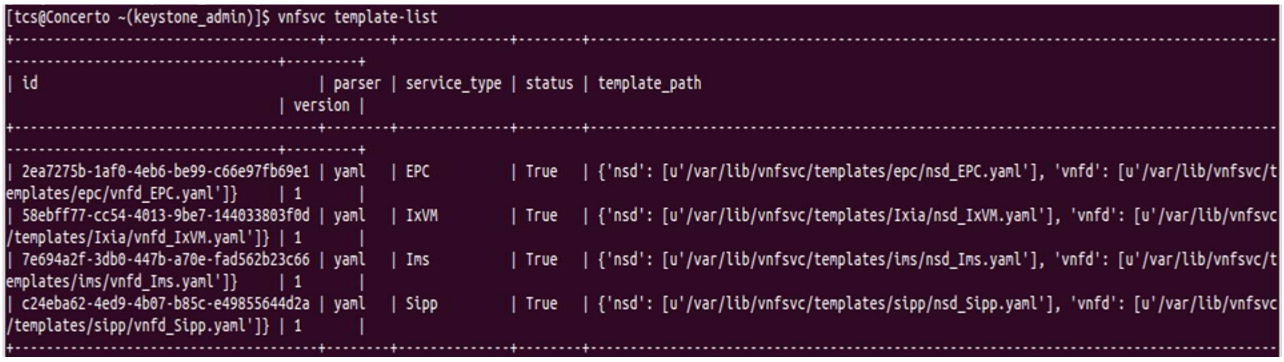
Once a service is successfully on-boarded and made available in the system, a user can request service creation. Upon request from the user via a client API, REST API or Heat service, the service orchestration is triggered. The request is validated for privileges and availability of resources using standard OpenStack Nova* REST APIs. The required networks and ports are created via the OpenStack Neutron* API. VM boot request is sent via the OpenStack novaclient.

Upon successful boot-up, the VNFManager connects to the VNF instance and pushes the required configuration via the plug-in framework. The configuration of the network functions is carried over the management network as illustrated in the architecture diagram below. Standard CommSP interfaces are used for service configuration, for example, in this use-case, NETCONF. This platform provides open, vendor-neutral interfaces for service on-boarding and orchestration. This is achieved via the plug-in framework. Examples of the plug-in modules can be found in the git repository. For example, users can refer to the example for a plug-in module for HAProxy load balancer here and example driver for Ellis node in Project Clearwater IMS here.

As mentioned above, service configuration is achieved via the plug-in framework. Based on the inputs provided for management interface in the VNF descriptor, valid service configuration drivers are used for configuring the VNFs. Configure_ service method is implemented to push the initial configuration to the VNF instance via the NETCONF interface. This method should be available in the plug-in that the user provides for the VNF configuration. Configure_service implementation in the HAProxy and Ellis example drivers listed above can be referred to. An example YANG model is also provided. The core components of TCS' OpenVNFManager [OpenVNFM] solution are listed in Figure 4.

The configuration parameters, both compute and network, are coded in a YAML configuration file with which the Brocade vEPC solution is on-boarded to perform further validations.

After a successful service onboard and provisioning, the service templates can be queried from the vnfsvc command line, the installation steps for which are mentioned in Appendix B. The sample output is shown below:



```
[tcs@Concerto ~(keystone_admin)]$ vnfsvc template-list
+--------------------------------------+--------+--------------+--------+--------------------------------------------------------------------------------------------
-----------------------------------+---------+
| id                                   | parser | service_type | status | template_path
                                   | version |
+--------------------------------------+--------+--------------+--------+--------------------------------------------------------------------------------------------
-----------------------------------+---------+
| 2ea7275b-1af0-4eb6-be99-c66e97fb69e1 | yaml   | EPC          | True   | {'nsd': [u'/var/lib/vnfsvc/templates/epc/nsd_EPC.yaml'], 'vnfd': [u'/var/lib/vnfsvc/t
emplates/epc/vnfd_EPC.yaml']}      | 1       |
| 58ebff77-cc54-4013-9be7-144033803f0d | yaml   | IxVM         | True   | {'nsd': [u'/var/lib/vnfsvc/templates/Ixia/nsd_IxVM.yaml'], 'vnfd': [u'/var/lib/vnfsvc
/templates/Ixia/vnfd_IxVM.yaml']} | 1       |
| 7e694a2f-3db0-447b-a70e-fad562b23c66 | yaml   | Ims          | True   | {'nsd': [u'/var/lib/vnfsvc/templates/ims/nsd_Ims.yaml'], 'vnfd': [u'/var/lib/vnfsvc/t
emplates/ims/vnfd_Ims.yaml']}      | 1       |
| c24eba62-4ed9-4b07-b85c-e49855644d2a | yaml   | Sipp         | True   | {'nsd': [u'/var/lib/vnfsvc/templates/sipp/nsd_Sipp.yaml'], 'vnfd': [u'/var/lib/vnfsvc
/templates/sipp/vnfd_Sipp.yaml']} | 1       |
+--------------------------------------+--------+--------------+--------+--------------------------------------------------------------------------------------------
```

vnfsvc

This is the service orchestrator that performs the following functionality:

» Descriptor validation and on-boarding

Descriptor is parsed and each parameter of the network service descriptor is validated. The information elements of the descriptor are detailed in the ETSI NFV MANO reference document. The NFV-MANO architectural framework identifies the following data repositories:

* NS catalogue

* VNF catalogue

* NFV instances repository

* NFVI resources repository

Ideally, during on-boarding the new service is added to the network service catalogue. There are no updates to the instances repositories. Upon successful validation, vnfsvc creates internal references for the service and the descriptors are added to the data repository. A new vnfsvc database is created as part of setting up the service. Details are specified in the Appendix B. The descriptors are added to the data repository of vnfsvc. As described in the setup procedure, the templates.json file should be updated with the correct resource entries for NSD and VNFD. The service is now ready for instantiation.

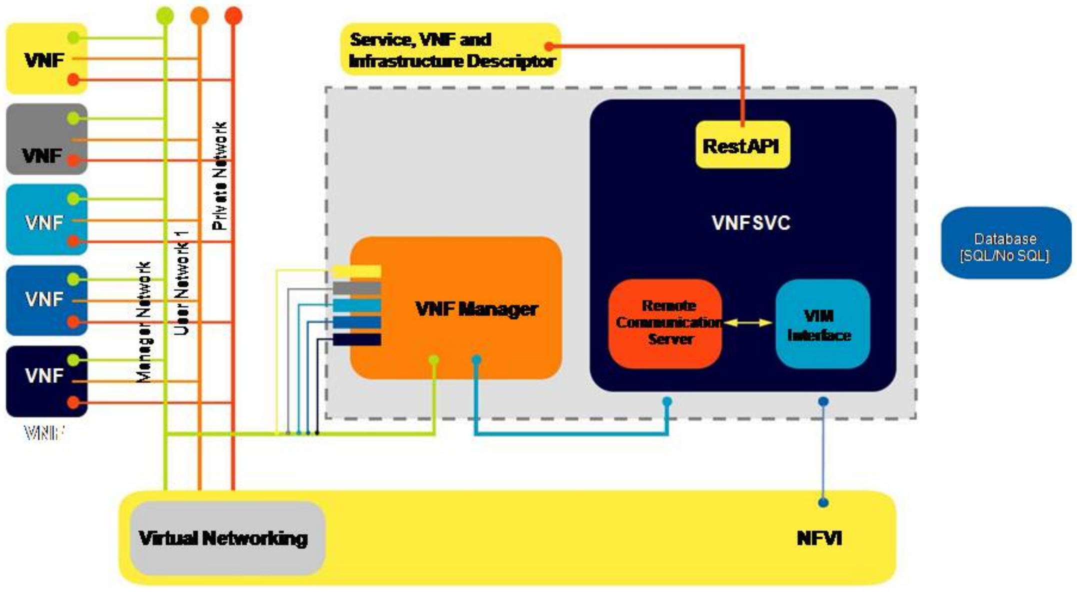


Figure 5. Service Initiation Block Diagram

» Network service instantiation and life-cycle management

The service instantiation request can be triggered via Heat or vnfsvc client or REST API. This triggers the creation of the VNF instances via the OpenStack REST API. The networks are created first via the OpenStack Neutron API. After this, VNFs are instantiated via the OpenStack Nova REST API.

VNF Manager

The VNF Manager communicates with the virtual network functions over the management network. Communication between the VNFManager and the virtual network functions/components can be over any standard management interface like NETCONF. OpenVNFM is responsible for the following:

- Configuring VNFs over standard interfaces via plug-ins. Each VNF should provide an extension of the plug-in framework. The OpenVNFManager communicates with the VNF instances via the extensions.
- For lifecycle management, the OpenVNFManager supports initialization of the VNF instances via the plug-in framework. The initialization is automatically performed via the plug-in framework as soon as the VNF is up and running. The initial configuration is captured in the VNF descriptor.

Heat

Heat is an OpenStack service that enables orchestration across various services. Clients can use Heat templates to create services. The Heat engine parses templates and enables the required OpenStack resources to create the stack. Heat allows CommSPs to extend the capabilities of the orchestration service by writing their own resource plug-ins. A resource plug-in can extend a base resource class and implements relevant life cycle handler methods. The resource class is responsible for managing the overall life cycle of the plug-in. It defines methods corresponding to the life cycle as well as the basic hooks for plug-ins to handle the work of communicating with specific down-stream services. In our implementation, heat.engine. resource.vnfsvc.vnfsvc.VNFSvcResource is the resource plug-in that extends the base resource class heat.engine.resource resource and implements the required Heat life cycle handler methods. When the Heat engine determines it is time to create a resource, it calls the create method of the applicable plug-in. This method is implemented in the resource base class, which further calls a handle_create method defined in the plug-in class (heat.engine.resources.vnfsvc.vnf_template.Service), which is responsible for using a specific service call.

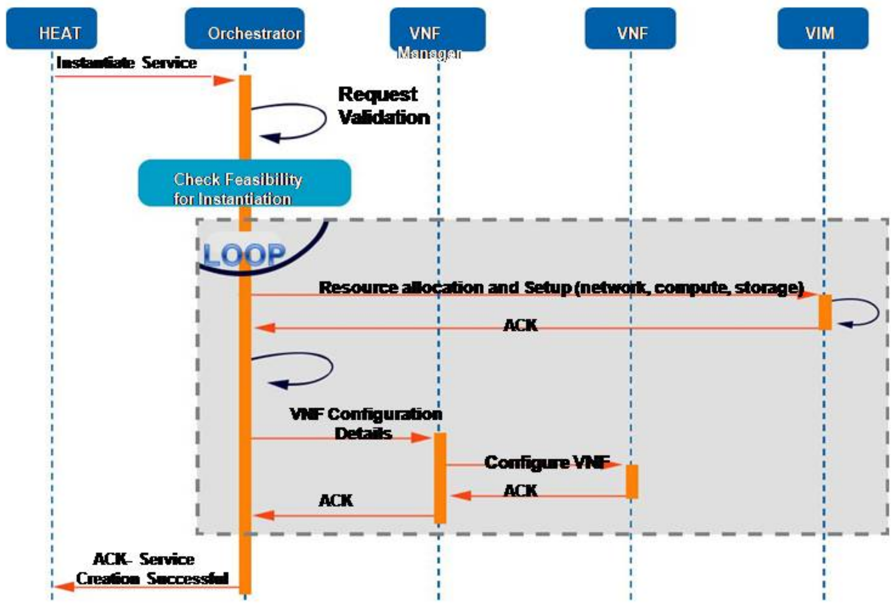

HEAT
Orchestrator
VNF Manager
VNF
VIM
Instantiate Service
Request Validation
Check Feasibility for Instantiation
LOOP
Resource allocation and Setup (network, compute, storage)
ACK
VNF Configuration Details
Configure VNF
ACK
ACK
ACK- Service Creation Successful

Figure 6. Using Heat to Instantiate a Service

The steps to set up the above three modules—Heat, vnfsvc, and VNFManager—are detailed in Appendix B. The required source code and sample templates are provided in this document for reference. The latest version of this software, with patches for updates, is available in the git repository at https://github.com/TCS-TelcoCloud. It is recommended to fetch the code, examples, and sample descriptors from the git repository. Figure 6 illustrates the steps in the creation of a service via Heat. Once the descriptor is on-boarded successfully by the administrator, the service can be instantiated by the user. In this reference architecture, a Heat template is used to instantiate the service.

A sample Heat template is available in Appendix B for reference, and installation instructions are also provided. On triggering the Heat orchestration, the request is first validated by the vnfsvc. A valid request is translated into Nova and Neutron API calls, and the required VNF instances are created. This process is repeated until all the instances are created. Once the VNF instance is up and running, the VNFManager configures the instance via the plug-in provided, which is a NETCONF driver in this reference architecture. The configuration data is extracted from the descriptors.

VOLTE Subscriber Attach Validation and Auto Scaling

To bring up the VOLTE, which is the system under test or test target, as a virtual network function (VNF), the configuration parameters of the vVOLTE components need to be input to TCS NFV Concerto in the form of service descriptors. YAML is one of the popular service descriptor types used. TCS NFV Concerto reads the configuration files and requests the service orchestrator to deploy all the components as virtual machines by making REST calls to the virtual infrastructure manager (VIM), a first step in a three step process. Once the system under test components are deployed as virtual machines on the host infrastructure, TCS NFV Concerto configures them to work as a functional block. This process might involve configuring the components in multiple iterations based on the complexity and the way the system is constructed. Once the above two phases are completed, NFV Concerto introduces the test suite that is planned to be executed on the network service as a whole.

NFV Concerto initiates VOLTE subscriber attaches using the UE components through the eNB provided by Ixia on the Brocade EPC solution. As part of the validation process, as many as 100,000 subscriber attach requests are placed on the EPC network service through multiple UE simulators. A policy configuration parameter is applied on to the network service in step one of the above three step process, through which TCS NFV Concerto monitors the number of subscriber attach requests that are placed on the EPC solution continuously throughout the lifetime of the network service, and notifies those events to subscribers of the notification events. As soon as the number of subscriber attaches reaches the threshold value that is configured as part of policy configuration, TCS NFV Concerto or the subscriber to the notification service can take an action. The action defined in the policy for the verification suite discussed here is Auto Scale of Control Plane Slice component. Once the auto scale is successful, TCS NFV Concerto configures the latest Control Plane Slice component to make it load ready.

The high level logical architecture of vEPC is as follows:

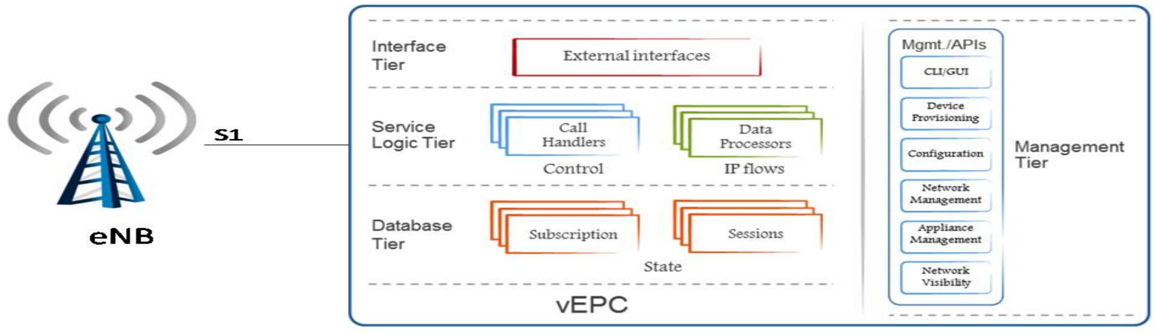


Figure 7. vEPC logical architecture

Appendix A: Abbreviation Glossary

This section gives the glossary of abbreviations and terms used in the document.

ABBREVIATION	DESCRIPTION
ACK	Acknowledgement
API	Application Programming Interface
BGCF	Breakout Gateway Control Function
CLI	Command Line Interface
CSCF	Call Session Control Function
DC	Data Center
DNS	Domain Name Server/Service
eNB	Evolved NodeB
GUI	Graphical User Interface
HS	Home Subscriber
HSS	Home Subscriber Server
I-CSCF	Integrating Call Session Control Function
KVM	Kernel-based Virtual Machine
MAA	Multimedia-Authorization-Answer
MANO	Management and Orchestration
MAR	Multimedia-Authorization-Request
MME	Mobility Management Entity
NETCONF	Network Configuration Protocol
NFV	Network Function Virtualization
NIC	Network Interface Card
NS	Network Service
NSD	Network Service Descriptor
OVS	Open vSwitch
P-CSCF	Proxy Call Session Control Function
PGW	Packet Data Network Gateway
PoC	Proof Of Concept
SGW	Software Gateway
UE	User Equipment
VLAN	Virtual LAN
VM	Virtual Machine
VNF	Virtualized Network Function
Ve-Vnfm	A reference point between VNF and VNF
Vi-Vnfm	A reference point between VIM and VNF

Appendix B: Installation Instructions

Installation Instructions for vnfsvc

OpenVNFManager enables NFV service orchestration on OpenStack platform

git clone --recursive https://github.com/TCS-TelcoCloud/OpenVNFManager.git

It has 3 components:

» `vnfsvc`

» `vnfManager`

» `python-vnfsvcclient`

`vnfsvc` runs as a service [similar to OpenStack Neutron, etc.] on the OpenStack controller node.

To install, execute the following commands

```
$ git clone https://github.com/TCS-TelcoCloud/OpenVNFManager.git
$ python setup.py install
```

Post Installation verify the following to ensure successful installation `[for Red Hat Linux/Centos7/Fedora]`:

»»`/etc/vnfsvc/ should contain`

```
* api-paste.ini,
* rootwrap.conf,
* rootwrap.d,
* templates.json,
* vnfsvc.conf
```

»»`/etc/vnfsvc/vnfsvc.conf` should contain correct passwords and URLs for OpenStack services.

»» Create keystone endpoints using following commands:

```
$ keystone service-create --name vnfsvc --type vnfservice --description "VNF service"
$ keystone endpoint-create --region RegionOne --service-id <vnfsvc_service_id> --publicurl
"http://<your_ip>:9010" --internalurl "http://<your_ip>:9010" --adminurl http://<your_ip>:9010
$ keystone user-create --tenant-id <service_tenant_id> --name vnfsvc --pass <passsword>
$ keystone user-role-add --user-id <vnfsvc_user_id> --tenant-id <service_tenant_id> --role-id
<admin_role_id>
Execute the following commands for database configuration:
$ mysql> create database vnfsvc;
$ mysql> grant all privileges on vnfsvc.* to 'vnfsvc'@'localhost' identified by <database
password>;
$ mysql> grant all privileges on vnfsvc.* to 'vnfsvc'@'%' identified by <database password>;
$ vnfsvc-db-manage --config-file /etc/vnfsvc/vnfsvc.conf upgrade head
$ mkdir /var/log/vnfsvc
Run with the following command to start the server:
$ python /usr/bin/vnfsvc-server --config-file /etc/vnfsvc/vnfsvc.conf --log-file /var/log/
vnfsvc/server.log
```

Installation Instructions for VNFManager

VNFManager interfaces with VNFs and `vnfsvc` for configuration and lifecycle management of virtual network functions. In the current setup, `init` is the only supported lifecycle event.

• Sample descriptors and explanations are provided in the `vnfsvc_examples` folder. It has:

```
»» NSD
»» VNFD
»» Heat template
»» README for running the installation
```

After installing `vnfsvc, python-vnfsvcclient` and Heat updates, run the setup as detailed in `vnfsvc_examples`.

• To install:

```
$ git clone https://github.com/TCS-TelcoCloud/vnfmanager.git
$ python setup.py install
$ git clone https://github.com/TCS-TelcoCloud/vnfmanager.git
$ python setup.py install
```

Installation Instructions for python-vnfsvcclient

This is a client for the `vnfsvc` API. Execute the below commands to install:

```
$ git clone https://github.com/TCS-TelcoCloud/OpenVNFManager.git
$ python setup.py install
```

Command-line API

• The complete description of the `vnfsvc` command usage can be found by running:

```
$ vnfsvc help
```

• Create, List, Show and Delete are currently supported. Usage of the operations supported can be found by appending "-h":

```
$ vnfsvc service-create -h
```

Example command for the create operation is given below:

```
$ vnfsvc service-create --name webservice --qos Silver --networks mgmt-if='fce9ee06-a6cd-
4405-ba0f-d8491dd38e2a' --networks public='b481ac9c-19bb-4216-97b5-25f5bd8be4ae' --networks
private='6458b56a-a6a2-42d5-8634-bdec253edf4e' --router 'router' --subnets mgmt-if='0c8ccdf2-
3808-462c-ab1e-1e1b621b0324' --subnets public='baf8bae2-3e4c-4b8b-bdb9-964fb1594203' --subnets
private='ad09ac00-c4d7-473f-94ec-2ad22153d1ca'
```

• Networks, subnets and routers given in the command should be created and corresponding UUIDs should be specified in the command-line.

• Command for the list operation is given below:

```
$ vnfsvc service-list <service-id>
```

• Command for the show operation is given below:

```
$ vnfsvc service-show <service-id>
```

• Command for the delete operation is given below:

```
$ vnfsvc service-delete <service-id>
```

• After installing `vnfsvc, python-vnfsvcclient` and Heat updates, run the setup as detailed in `vnfsvc_examples`.

Installation Instructions for Heat

The Heat module is updated to enable orchestration of VNFs with `vnfsvc`. Details of the setup for Red Hat/Fedora/CentOS platforms is as follows:

```
$ git clone https://github.com/TCS-TelcoCloud/OpenVNFManager.git
1. Copy heat/heat/common/config.py to
   /usr/lib/python2.7/site-packages/heat/common/config.py
2. Copy heat/heat/engine/clients/__init__.py to
   /usr/lib/python2.7/site-packages/heat/engine/clients/__init__.py
3. Copy heat/heat/engine/clients/os/vnfsvc.py to
   /usr/lib/python2.7/site-packages/heat/engine/clients/os/vnfsvc.py
4. Copy heat/heat/engine/resource.py to
   /usr/lib/python2.7/site-packages/heat/engine/resource.py
5. Copy heat/heat/engine/resources/vnfsvc/__init__.py to
   /usr/lib/python2.7/site-packages/heat/engine/resources/vnfsvc/__init__.py
6. Copy heat/heat/engine/resources/vnfsvc/vnfsvc.py to
   /usr/lib/python2.7/site-packages/heat/engine/resources/vnfsvc/vnfsvc.py
7. Copy heat/heat/engine/resources/vnfsvc/vnf_template.py to
   /usr/lib/python2.7/site-packages/heat/engine/resources/vnfsvc/vnf_template.py
8. Update [heat.clients] section in entry_points.txt with
   "vnfsvc = heat.engine.clients.os.vnfsvc:VnfsvcClientPlugin"
9. Update [clients_vnfsvc] in /etc/heat/heat.conf.
```

Examples

Sample NSD Template (in YAML) for IMS orchestration

```
nsd:
  name: ims_template
  vendor: ETSI
  description: "Ims service"
  version: "1.0"
monitoring-parameter:
  param-id: "num-requests"
  description: "Number of http requests
  load balancer can handle"
 endpoints:
  apn-router-gateway:
    end-point-id: Router-gateway
    description: Router gateway
 apn-webserver:
    end-point-id: WebServer
    vnf: vLB:pkt_in
    description: web server
 flavors:
  Silver:
  flavor-id: Silver
  description: "Silver Service flavor"
  monitoring:
    param-id: "num-requests"
  description: "Number of http requests
  load balancer can handle"
```

```
assurance-params:
  param-id: "num-requests"
  value: 1000
  member-vnfs:
   • name: Ims
     member-vdu-id: vHS
   • name: Ims
     member-vdu-id: vBono
     dependency: Ims:vSprout
   • name: Ims
     member-vdu-id: vSprout
     dependency:
        - Ims:vHomer
        - Ims:vHS
   • name: Ims
     member-vdu-id: vHomer
   • name: Ims
     member-vdu-id: vHomer
   • name: Ims
     member-vdu-id: vEllis
     dependency:
        - Ims:vHomer
        - Ims:vHS
member-vlds:
  mgmt-if:
     property: internal
     member-vnfs:
        Ims:vHomer:
         connection-point: 'mgmt-if'
        Ims:vHS:
         connection-point: 'mgmt-if'
        Ims:vEllis:
         connection-point: 'mgmt-if'
        Ims:vSprout:
         connection-point: 'mgmt-if'
        Ims:vBono:
         connection-point: 'mgmt-if'
    private:
     property: internal
     member-vnfs:
        Ims:vHomer:
         connection-point: 'private'
        Ims:vHS:
```

```
         connection-point: 'private'
        Ims:vEllis:
         connection-point: 'private'
        Ims:vSprout:
         connection-point: 'private'
        Ims:vBono:
         connection-point: 'private'
    Router: apn-router-gateway
  forwarding-graphs:
    XYZAccess:
      direction: bidirection
      network-forwarding-path:
        • name: apn-router-gateway
          type: endpoint
        • name: Ims:vBono
          type: vnf
        connection-point: private
```

Sample VNFD (in YAML) for IMS

```
vnfd:
 id: Ims
 vendor: TCSL
 description: "Ims service"
 version: "1.0"
 connection-point:
  private:
   name: private
   description:   "Private interface"
  mgmt-if:
   name: "mgmt-if"
   description: "Management interface"
 flavors:
  Silver:
   description: 'Silver service flavor'
   flavor-id: Silver
   assurance-params:
    param-id: "num-requests"
    value: 100
   vdus:
    vHomer:
     vdu-id: vHomer
     implementation_artifact:
      cfg_engine: puppet
      deployment_artifact: "/home/ccc/
      userdata_ims.yaml"
    num-instances: 1
    lifecycle_events:
     init: ""
```

```
     vm-spec:
      container_format:   "bare"
      disk_format:  "qcow2"
      image: "/home/XYZ/vnfd.img.tar.gz"
      is_public: "True"
      min_disk: 8
      min_ram: 512
      name: vHomer
      password: tcs
      username: "tcs"
     storage: 8
     memory:
      total-memory-mb: 512
     cpu:
      num-vcpu: 1
     network-interfaces:
       pkt-in:
         name: "pkt-in"
         description: "Packet in interface"
         connection-point-ref: "connection-
         points/private"
      management-interface:
       name: "management-interface"
       description: "Interface Used
       for management interface"
       connection-point-ref:
       "connection-points/mgmt-if"
       properties:
         driver: ''
```

The above descriptor is derived from the ETSI MANO reference document. Some of the tags are described in the table below:

IDENTIFIER	DESCRIPTION
Dependency	This identifier is used in the member-vnfs section of the network service descriptor. It lists the dependencies of a given vdu and would mandate that those VNF instances be instantiated before this VDU.
Driver	This identifier is used in the network-interfaces section of the VNFD. It describes the management driver required to manage and configure the VNF instance via the management network (the TCS OpenVNFManager).
Endpoints	This descriptor is used in the NSD to define the start and termination points for a service.
Forwarding-graphs	This identifier is used in the NSD to identify a static path traversal within this service chain. This enables the orchestrator to block or allow traffic between the virtual network function instances.
Flavors	This identifier is a list with the possible flavors to be supported for this service. Each flavor offers a particular quality of service and configuration. However the entry points and the forwarding graph remain the same and hence are defined globally within the network service descriptor.

Examples are indicated for reference. For the latest version of source, configuration and updates please refer to the github repository.

The following parameters should be updated as per the local settings before on-boarding the service.

IDENTIFIER	DESCRIPTION
deployment_artifact	Specify path of the userdata file
driver	Specify the class path to the specific driver which is placed under vnfmanager/drivers
image	Specify path of the image
image-id	Specifiy image uuid present in glance
vm-spec	The following child tags should be updated: min_ram: RAM to be allocated to the VNF min_disk: Disk to be allocated to the VNF username: User ID with pseudo privileges to enable NETCONF configuration on the VNF password: Password for the above user ID storage: Any additional storage to be provisioned above minimum disk

NOTE: Either image or image-id has to be specified but not both.

Sample Heat Template

```
heat_template_version: 2013-05-23

description: A simple template which creates
a device template and a device

parameters:
 mgmt-if:
  type:string
  description : management Subnet
  default : 30.0.0.0/24

private:

  type: string
  description: Packet-IN Subnet
  default : 20.0.0.0/24

name:

  type: string
  description: Name of the service
  default : Ims

router:

  type: string
  description: Router name
  default : VNF_Router

quality_of_service:
  type: string
  description: Quality of service
  default : Silver

resources:
 router_n:
  type: OS::Neutron::Router
  properties:
  name:VNF_Router

network_m:
  type: OS::Neutron::Net
  properties: {name:management}

subnet_m:
  type: OS::Neutron::Subnet
  properties:
   name : management _subnet
   network _id: {Ref: network_m}
   cidr: { get_param: mgmt-if }
   ip_version: 4

network_private:
  type: OS::Neutron::Net
  properties: {name: private}

subnet_private:
  type: OS::Neutron::Subnet
  properties:
   name : private_subnet
   network _id: {Ref: network_private}
   cidr: { get_param: private }
   ip_version: 4

router_intrfc_m:
  type: OS::Neutron::RouterInterface
```

```
  properties:
   router_id: {Ref: router_n}
   subnet_id: {Ref: subnet_m}

service:

  type: OS::VNFSvc::Service
  properties:
   name:{get_param: name}
   description: VNF Service quality_of_
   service: {get_param: quality_
   of_ service attributes: {‘networks’:{‘mgmt-
   if’: {get_ attr: [subnet_m, network_id]},
   ‘private’:{get_ attr: [subnet_private,
   network_id]}}, ‘router’ : {get_param :
   router},’subnets’:{‘mgmt-if’:
   {“Ref”: “subnet_m”}, ‘private’:{“Ref”:
   “subnet_private”}, ‘router-iface’:{Ref:
   router_intrfc_m}}}

zopier_network:
  type: OS::Neutron::Net
  properties:
   name: user_network_1
  depends_on: service

zopier_subnet:

  type: OS::Neutron::Subnet
  properties:
   name: user_subnet_1
   network_id: { Ref: zopier_network }
   cidr: 80.0.0.0/24

my_router_interface:

  type: OS::Neutron::RouterInterface
  properties:
   router_id: {Ref: router_n}
   subnet_id: {Ref: zopier_subnet}

my_instance:

  type: OS::Nova::Server
  properties:
   name: zopier_1
   image: 2a2eb7af-1989-4865-a8d4-038f36fca5e2
   flavor: m1.medium
   networks: [network:Ref:zopier_network}

zopier_network_1:

  type: OS::Neutron::Net

  properties:

   name: user_network_2

  depends_on: service

zopier_subnet_1:

  type: OS::Neutron::Subnet
  properties:
   name: user_subnet_2
   network_id:{Ref: zopier_network_1}
   cidr: 90.0.0.0/24
```

```
my_router_interface_1:
  type: OS::Neutron::RouterInterface
  properties:
   router_id:{Ref: router_n}
   subnet_id:{Ref: zopier_subnet_1}

my_instance_w1:
  type: OS::Nova::Server
  properties:
   name: zopier_2
   image: 2a2eb7af-1989-4865-a8d4-038f36fca5e2
   flavor: m1.medium
   networks:[network:{Ref: zopier_net work_1}]
```

References

IDENTIFIER	SOURCE
[NFV WP]	Network Function Virtualization White paper - Network Operator Perspectives on Industry Progress (http://portal.etsi.org/NFV/NFV_White_Paper2.pdf)
[NFV E2E Arch]	Network Function Virtualization Reference Architecture (http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf)
[IMS]	Project Clearwater (http://www.projectclearwater.org/)
[OpenVNFManager]	TCS' OpenVNFManager (https://github.com/TCS-TelcoCloud/OpenVNFManager)
[OpenStack]	http://www.openstack.org