Virtual VoIP Orchestration with OpenVNFManager Reference Architecture

VNF Lifecycle Orchestration of a Virtual IMS Use Case on Intel® Architecture

Audience and Purpose

With the transition to Network Function Virtualization (NFV), as the focus shifts from hardware appliances to Virtualized Network Functions (VNFs) deployed on cloud-computing software platforms like OpenStack®, the challenge arises for VNF lifecycle management. These orchestration solutions enable service providers to reap the real benefit of transition to standard high volume servers while still being able to integrate into legacy OSS/BSS deployments. This technical reference architecture describes how this can be achieved with the open source contributions of OpenVNFManager from Tata Consultancy Services (TCS)* and community OpenStack running on servers powered with Intel® Xeon® E5-2600 family of processors. Project Clearwater’s open source implementation of an IP Multimedia Subsystem (IMS) was used as the vIMS.

This collaboration was demonstrated at IBM Interconnect 2015.

Executive Summary

Today’s networks are overly complex, partly due to an increasing variety of proprietary, fixed-function appliances that are unable to deliver the agility and economics needed to address constantly changing market requirements. 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 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. Conventional communications infrastructures that rely on dedicated proprietary hardware to implement each network function not only increase cost and complexity, but also this hardware-centric, siloed infrastructure approach can impede business agility and innovation. 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.

Network function virtualization (NFV) can provide the infrastructure flexibility and agility needed to successfully compete in today’s evolving communications landscape. NFV implements network functions in software running on a pool of shared standard, off the shelf servers...
Table of Contents

Partners ........................................... 2
Intel .................................................. 2
TCS ................................................... 2
Overview of vIMS NFV use case ............. 3
Key Ingredients .................................... 3
Intel ................................................... 3
TCS ................................................... 4
Project Clearwater ................................. 9
Test Bed Network ................................ 10
Equipment and Software ....................... 11
Testing ............................................. 11
Appendix A ......................................... 16
Appendix B ......................................... 16
Installation Instructions for vnfsvc ......... 16
Installation Instructions for VNF Manager .. 18
Installation Instructions for python-vnfsvcclient .. 18
Installation Instructions for Heat ............. 19
Examples .......................................... 20
Sample NSD Template (in YAML) for IMS Orchestration ........ 20
Sample VNFD (in YAML) for IMS .................. 21
Sample Heat Template ......................... 23
Appendix C ......................................... 24
Installation of Softphone ....................... 24
Softphone Account Signup/Registration .... 24
Configuration of Softphone ................. 24
References ....................................... 25

instead of using dedicated proprietary hardware. 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 Tata Consultancy Services [TCS] are collaborating to demonstrate the technical and business viability of NFV deployment and service orchestration using cloud computing technologies. The collaboration delivers a Proof of Concept (PoC) that demonstrates a novel NFV-based orchestration solution for an operator’s cloud-based IP Multimedia Subsystem [IMS]. Any mobile operator wanting to migrate to the cloud and operationalize an IMS solution in their end to end network is a potential customer for this solution. With the help of TCS, Intel® Xeon® processor-based servers help save up-front acquisition costs (CapEx) due to the economies of scale associated with Commercial Off-The-Shelf (COTS) hardware, reduce power and cooling requirements (OpEx), and provide a common infrastructure, significantly simplifying maintenance compared with the traditional fixed-form, hardware-based different appliances.

Partners

This reference architecture demonstrates an open network function virtualization infrastructure (NFVI) ecosystem based on Intel Xeon processor based servers with ingredients contributed by TCS to provision, deploy, and manage a virtual IP Multimedia System (vIMS) service. This section provides an overview of the vendors that partnered to demonstrate that the NFV vIMS use case can be operationalized in a carrier network using VNF life cycle management and integrated with legacy orchestration for ease of integration, deployment, management, scaling and monitoring.

Intel

The hardware components of the NFV infrastructure (NFVI) are powered with the processing power and network and virtualization capabilities of Intel® Xeon® E5-2600 class of processors and 10 gigabit Intel® Ethernet technology. Intel® Architecture provides operators a standard, reusable, shared platform for NFV applications that is easy to upgrade and maintain. Recent improvements in Intel architecture have significantly reduced the need for specialized silicon, enabling network operators to take advantage of the proven scalability of modern, virtualized data center technology. Advantages of this approach include a streamlined network, and cost savings through hardware reusability and power reductions. Particularly, data plane processing has been greatly accelerated by optimization techniques developed over several years at Intel. Developers can access these tools from the Data Plane Development Kit (DPDK).

TCS

Tata Consultancy Services brings in rich experience of over 30 years in telecommunication industry with more than 300 consultants serving customers in the SDN and NFV domains across the globe where TCS offers end-to-end product engineering capability. TCS’ Centers of Excellence (CoEs) specializing in virtualization, networking and orchestration enable cutting edge research and development of enablers and solutions. TCS also contributes to open-source communities such as OpenStack, OpenDaylight, Open vSwitch, OPNFV and Open Network Operating System (ONOS), building their technology know-how and extending thought-leadership positioning to real-world
implementations. TCS' extensive partner ecosystem helps consolidate telco cloud insights and customize solutions based on customers' requirements.

Overview of vIMS NFV Use Case

The IMS core network 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 Evolved Packet Core (EPC) and other IP networks. Online Charging System (OCS) and Offline Charging System (OFCS) provide the charging functionality as a part of session management. IMS uses the Session Initiation Protocol (SIP) for session setup and teardown while DIAMETER is used as the AAA (Authorization, Authentication and Accounting) 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 – WiFi, broadband, LTE, and others as they evolve.

The European Telecommunications Standards Institute (ETSI) use case #5 describes the virtualization of the mobile core including the Evolved Packet Core and the IMS elements to achieve reduced total cost of operation (TCO), efficiencies of flexible resource allocation and scale to achieve higher availability and resiliency with dynamic network reconfiguration.¹

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 Ingredients

Intel

Intel Xeon E5-2600 v3 family of processors provide the foundation for innovation in NFV by powering the Software Defined Infrastructure (SDI) with energy efficient high performance building blocks with system visibility for monitoring and control to address the imminent need of greater flexibility with higher levels of automation and orchestration; along with significant benefits in performance, power efficiency, virtualization, and security meeting the needs for compute, networking and storage. They come with the first DDR4 implementation and Intel® Advanced Vector Instructions 2.0 (AVX 2.0); enhanced Intel® Data Direct IO (DDIO), Intel® Quick Path Interconnect (QPI), Intel® Intelligent Storage Acceleration Library and Intel® Quick Assist Technology.²

- Advanced multi-core, multi-threaded processing – with up to 18 cores and 36 threads per socket
- Larger cache and faster memory with up to 45 MB of LLC for fast access to frequently used data and 24 DIMMs per two-socket server to support multiple data-hungry VMs
- Faster maximum memory speeds than the previous generation (2133 MHz versus 1866 MHz)
- Higher performance for diverse workloads with Intel® Turbo Boost Technology takes advantage of power and thermal headroom to increase processor frequencies for diverse workloads

---

DDDR4 next generation memory technology improves platform performance on memory intensive workloads with up to 1.4X higher bandwidth versus previous-generation platforms. Adopting DDR4 enables solutions to meet data center energy efficiency requirements.

**Intel® Data Direct I/O Technology** provides intelligent, system-level I/O performance improvements. It allows Intel® Ethernet Controllers and adapters to talk directly with processor cache - making the processor cache the primary destination and source of I/O data rather than main memory, helping to deliver increased bandwidth, lower latency, and reduced power consumption.

**Intel AVX2 with new Fused Multiply-Add (FMA) instructions** in Intel Xeon Processor E5-2600 v3 product family doubles the floating point operations (Flops) from first generation Intel AVX, and doubles the width of vector integer instructions to 256 bits, expanding the benefits of Intel AVX2 into enterprise computing with up to 1.9x performance gains.

**Hardware-accelerated nested virtualization** – Intel® Virtual Machine Control Structure (Intel® VMCS Shadowing) extends root virtual machine monitor (VMM)-like privileges to a guest VM, enabling legacy OS, applications, security software, and other code not supported on the platform root VMM to be run on the system.

**Intel Integrated I/O** provides up to 80 PCIe* lanes per two socket server, and supports the PCIe 3.0 specification with atomic operations support for improved peer-to-peer (P2P) bandwidth

**The Non-Volatile Memory Express® (NVMe*) specification** that is supported by the Intel® Solid-State Drive Data Center Family for PCIe overcomes SAS and SATA SSD performance limitations through an optimized register interface, command set, and feature set for PCI Express (PCIe*)-based Solid-State Drives (SSDs).

**The Intel® Ethernet Controller XL710 Series** – Delivers proven 10 and 40 Gigabit Ethernet connectivity for the platform, extending Intel® Virtualization technologies beyond server virtualization to network virtualization. They reduce I/O bottlenecks by providing intelligent offloads for networking traffic per VM, enabling near native performance and VM scalability. Supported technologies include: VMDq for emulated path, SR-IOV for direct assignment, Virtual Ethernet Port Aggregator (VEPA), Virtual Ethernet Bridge support; VxLAN, NVGRE, GENEVE offloads in addition to TCP stateless offloads and flow classification.

**Intel and Network Transformation**

Intel has been defining and building the software and hardware infrastructure to enable this transformation over a number of years. Intel has recognized the need and is helping to move the industry with contributions to open source and standards initiatives to enable solutions for service providers. Intel is helping communications service providers (Telcos) implement their four primary workloads—application processing, control processing, packet processing, and signal processing—on Intel® architecture. In realizing this vision, Intel is a leading contributor to standards organizations: ETSI NFV ISG, IETF SFC, ONF and open source efforts: OpenStack, OpenDaylight*, DPDK and Open vSwitch.

To be able to achieve the benefits of hardware in a virtualized environment Intel has contributed Enhanced Platform Awareness (EPA) features in OpenStack that allow the scheduler to make provisioning and placement decisions on a host node based on availability of certain hardware capabilities. OpenStack Havana release supported features like:

- PCIe* SR-IOV Accelerators: (Intel® QuickAssist Technology enabler)
- Enable assignment of dedicated accelerator resources via virtual functions to VMs (e.g., for crypto/ compression workloads)

OpenStack Icehouse has support for features like:

- CPU Feature Discovery: Drive smarter VM placement decisions based on understanding of CPU capabilities available in the infrastructure. For example to leverage AVX/SSE for Deep Packet Inspection (DPI) workloads, AES-NI, and SecureKey (rdrand) for security workloads.

**TCS**

**TCS OpenVNFManager – NFV Service Orchestration Framework**

ETSI NFV MANO specification 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 the NFV orchestration and lifecycle management fully compliant with the ETSI MANO specification and works with 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 north-bound REST API, enabling fully automated service provisioning. With this solution, both
TEMs and Service Providers gain the responsiveness and service agility to meet customer demands.

OpenVNFManager manages VNF occurrences via instance specific plug-ins that communicate with the specific 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 a few best practices to be leveraged. 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 and should expose standard or “de-facto” standard interfaces.
- support and help realize the feasible decoupling of VNF software from the hardware.
- support management and orchestration of VNFs and network services using NFVI resources from a single or across multiple NFVI-PoPs (NFVI-Point of Presence).
- support modelling of NFVI resource requirements of VNFs in an abstracted way.
- support modelling of NFVI resources in a way that an abstraction of them can be exposed by functionality in one layer, to functionality in a different layer.

OpenVNFManager is a software based VNF Management framework which is scalable within a NFVI-PoP. It has a centralized component – vnfsvc (VNF service daemon) and a distributed...
VNFM component which makes this possible. The framework enables automated service creation and life-cycle management of the VNF instances. It exposes a 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:** TCS has open-sourced OpenVNFManager 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 Virtual Network Function, based on NETCONF, as is currently used in telco environments. For VNFs that do not support such an 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 Virtual Network Function, 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. In this phase the descriptors are validated and the dependencies are verified by the system and services made available in the catalog. Successful on-boarding indicates that the service is available for deployment. REST API enables on-boarding of the Network Service.

  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 for the service creation. On-boarded services are made available via a service catalog 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.

- **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 Neutron API. VM boot request is sent via

![GitHub](https://github.com/TCS-TelcoCloud/OpenVNFManager)

**Figure 3:** TCS OpenVNFManager in Github
the OpenStack novaclient. Upon successful boot-up, the VNFMManager connects to the VNF instance and pushes the required configuration via the plug-in framework. All the configuration of the network functions is carried over the management network as illustrated in the architecture diagram below. Standard Telco 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, user can refer to the example for a plug-in module for HAProxy loadbalancer [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 virtual network functions. `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’ OpenVNFMManager [OpenVNFM] solution are illustrated in Figure 4.

- **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.
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 virtual network function 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.

• VNFManager

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 Virtual Network Functions over standard interfaces via plug-ins. Each VNF should provide an extension of the plug-in framework. The OpenVNFM communicates with the VNF instances via the extensions.

  » Lifecycle Management

The OpenVNFM 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 to enable 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 service providers 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 which 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.vnfsvc. Service) which is responsible for using a specific service call.
The steps to setup 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 errors, 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 5 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 PoC 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.

Project Clearwater – An Open Source Core IMS Implementation

Project Clearwater (http://www.projectclearwater.org/) is an open source implementation of an IMS that is built for the cloud and hence can achieve massive scalability and cost effectiveness. Clearwater provides SIP-based call control for voice and video communications and for SIP-based messaging applications. Clearwater follows IMS architectural principles and supports all of the key standardized interfaces expected of an IMS core network. When deployed as an IMS core, Clearwater does everything that you’d expect an IMS core to do, incorporating Proxy CSCF (Call Session Control Function), Interrogating CSCF and Serving CSCF, together with Breakout Gateway Control Function. Clearwater also includes a WebRTC gateway, and natively supports interworking between WebRTC clients and standard SIP-based clients, using SIP over WebSocket signaling. Since Clearwater was designed from the ground up to run in virtualized environments and take full advantage of the flexibility of the Cloud, it is extremely well suited for NFV. Its architectural features include the following:

The user equipment (UE) makes a request to Bono – that is the proxy CSCF node providing a SIP IMS Gm compliant interface.

Bono uses Sprout (SIP Router) that is a horizontally scalable, combined SIP registrar and authoritative routing proxy, that handles client authentication and the ISC interface to application servers. Sprout implements most of the I-CSCF and S-CSCF functionality. The Sprout cluster includes a redundant memcached cluster storing client registration data and other long-lived state. SIP transactions are load

Figure 6: Project Clearwater Architecture
balanced across the Sprout cluster, so there is no long-lived association between a client and a particular Sprout node.

Homestead (HSS Mirror) provides a Web services interface to Sprout for retrieving authentication credentials and user profile information. Homestead nodes run as a cluster using Cassandra as the store for mastered/mirrored data. In the IMS architecture, the HSS mirror functions are considered to be part of the I-CSCF and S-CSCF components, so in Clearwater I-CSCF and S-CSCF functions are implemented with a combination of Sprout and Homestead.

Ralf (Rf CTF) provides Rf Charging Trigger Function, which is used in IMS to provide offline billing. Bono and Sprout report P-CSCF and I-CSCF/S-CSCF chargeable events respectively to Ralf, which then reports these over Rf to an external Charging Data Function (CDF). As the other components, Ralf nodes run as a cluster, with session state stored in memcached cluster. (Storage of session state is required to conform to the Rf protocol.)

Homer (XDMS) is a standard XML Document Management Server used to store multimedia telephony (MMTel) service settings documents for each user of the system and uses a standard XCAP interface. Homer nodes also run as a cluster using Cassandra as the data store.

Ellis is a sample provisioning portal providing self sign-up, password management, line management and control of MMTel service settings.

**Test Bed Network**

This section provides an overview of the service provider point of presence (PoP) where the vIMS service is deployed on industry standard Intel processor based servers and cloud management technologies are used.

A simplified view of the system is illustrated Figure 7.

Intel Xeon processor-based servers and software vendors like Red Hat with their enterprise operating system, and OpenStack-based cloud management systems form the basic NFV Infrastructure as a service (NFVaaS) platform. TCS’ OpenVNFManager serves as the Orchestrator and VNF Manager. Project Clearwater VNFs provide the vIMS service.

Sample service descriptors are available at [https://github.com/TCS-TelcoCloud/vnfsvc_examples](https://github.com/TCS-TelcoCloud/vnfsvc_examples).
Service creation is initiated via a Heat template. As illustrated in Figure 5, the sequence of events to completely orchestrate the IMS service are performed. The Examples Section of Appendix B details the sample descriptors to on-board and provision the IMS service. Once the Virtual machines are created, the applications are installed via any Infrastructure Manager (in this case OpenStack). Alternately pre-configured images can be used to deploy the service. The Virtual machines are instantiated via the VIM interface.

The VNFManager is provisioned with all the required plug-in components to successfully serve the IMS Service configuration and life-cycle management. Configuration of the VNFs can be achieved via standard interfaces like NETCONF, SNMP or a CLI supported by a custom SDK. The `vnfsvc` daemon continuously monitors the service deployment status and ensures successful deployment and configuration of the IMS. The user equipment connects over IP to the service providers cloud based vIMS. Zoiper client based soft phones were used in this PoC.

**Testing**

Start with the installation and setup of the OpenVNFManager.

Latest setup instructions, scripts and code-base are available at [https://github.com/TCS-TelcoCloud](https://github.com/TCS-TelcoCloud).

There are four main components that are being setup:

- **vnfsvc**
  This runs as an OpenStack-like service on the controller node. The setup instructions are available [here](https).

- **VNFManager**
  This is the lifecycle manager for the virtual network functions. The setup instructions are available [here](https).

- **Client**
  As in OpenStack, a command line interface is provided for the VNFManager service via a python client. The setup instructions for the client are available [here](https).

- **Heat**
  To enable the user to instantiate and on-boarded service, the Heat updates should be added. The detailed steps for this are available [here](https). It is required that the user install the Heat module when installing OpenStack Heat client.

Once all of these modules are setup successfully, the following steps need to be followed next.

### Equipment & Software

<table>
<thead>
<tr>
<th>NFV COMPONENT</th>
<th>COMPONENT DESCRIPTION</th>
</tr>
</thead>
</table>
| NFVI/Hardware       | Intel® Xeon® Processor E5-2600 V3  
|                     | Intel® XL710 10 Gigabit Ethernet Controller with dual- or quad-core ports             |
| NFVI/OS & Hypervisor| RHEL OS v6.1  
|                     | RHEL 6 KVM                                                                               |
| VNF                 | IP Multimedia System  
|                     | Open Source Project Clearwater                                                        |
| VIM                 | Red Hat OpenStack 6.0                                                                  |
| VNF Manager(s)      | TCS OpenVNFManager                                                                    |
| Solution Integrator | TCS                                                                                   |
Update the nsd and vnfd templates to point to the correct resource locations in your setup (exists in vnfsvc_examples in github) path in /etc/vnfsvc/templates.json
Update the images (exists in sample_templates) paths in vnfd_vIMS.yaml for flavor “Silver” as indicated in the templates.
Update the userdata (exists in vnfsvc_examples) path under deployment_artifact tag for “Silver” flavor in vnfd_vLB.yaml

The VNF Service component of OpenVNFManager can be started using CLI or as a service on a controller node. The following configuration parameters are required before starting the vnfsvc service:

- OpenStack Controller information
- Keystone connection URI
- Credentials
- Network Service and VNF templates – YAML templates of all the Network Services supported

Perform the updates mentioned in the box above to the configuration file – vnfsvc.conf and launch VNFSvc.

After that, the setup is ready, and a user can launch the IMS service and execute a call through the Zopier softphones to validate the setup.

To start the IMS service, the first step is to launch a Heat template.

Heat Template
From the dashboard, navigate to Orchestration and choose the “Launch Stack” option. The example Heat template specified in Appendix B should be loaded and the following parameters should be filled in the form [illustrated in Figure 8]:

Update the image path(any desktop image) and flavor details under user_instance tag in heat.yaml
Enter the values similar to examples given below for heat template attributes before uploading it.

```
name - IMS
flavor - Silver
private - 192.168.1.0/24 (Any IPv4 CIDR)
mgmt-if - 192.168.3.0.24 (Any IPv4 CIDR)
user-nw - 192.168.4.0/24 (Any IPv4 CIDR)
router - <Router name>
```

Upload the heat.yaml to HEAT. This triggers the stack deployment.
Switch to the Network Topology tab and observer the network being created and then the virtual network function instances are spawned. The completely launched network topology would be as illustrated in the following figure.

Figure 8: Heat Template

Two user networks are also created. Virtual Machines running Zoiper clients are hosted on these networks.

[Zoiper can be installed as detailed in the Appendix C followed by configuration of the Zoiper phones.]
Once the network is created, create a floating IP pool in OpenStack. Next, associate floating IPs to Ellis and Bono Virtual Network Function instances. This indicates that the test network has been setup.

In the Zoiper client proceed with registration of the client with Ellis Virtual Network Function. Upon successful registration of the first client, repeat the steps to register the second client.

Both clients would indicate successful registration. After this, user 1 can dial the contact number for user 2 and dial-tone can be received on user 2 Zoiper. Figures 10a and 10b illustrate the call between the two Zoiper test clients.
As seen, registered numbers are dialed and end-to-end VoIP call is successfully made over the soft phones.

The logs in the IMS nodes are monitored for successful registration of the softphones.

Navigate to the instances tab and open the console of the Sprout node. Execute the following command after logging into the Sprout node:

```
$> tailf /var/log/sprout/log_current.txt
```

As illustrated in Figure 11, the logs indicate the SIP messaging with the registered mobile numbers, indicating that the call is going through the IMS network that was provisioned and deployed.

The log shown here depicts the messages that are shown between registered entities with registration numbers as 6505550942 and 6505550322.

Figure 11
### Appendix A

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

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

### Appendix B

#### 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:

```bash
$ keystone service-create --name vnfsvc --type vnfservice --description "VNF service"
$ keystone user-create --tenant-id <service_tenant_id> --name vnfsvc --pass <password>
$ 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:

```bash
$ 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
  $ git clone https://github.com/TCS-TelcoCloud/OpenVNFManager.git
  $ python setup.py install
  $ vnfsvc help
  $ vnfsvc service-create -h
  $ 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-able-1e1b621b0324' --subnets public='baf8bae2-3e4c-4b8b-bdb9-964fbi594203' --subnets private='ad09ac00-c4d7-473f-94ec-2ad22153d1ca'
  ```

**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-able-1e1b621b0324' --subnets public='baf8bae2-3e4c-4b8b-bdb9-964fbi594203' --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
  commandline.

• 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/OpenVNFM/Heat

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


7. Copy heat/heat/engine/resources/vnfsvc/vnf_template.py to /usr/lib/python2.7/site-packages/heat/engine/resources/vnfsvc/vnf_template.py


Examples

Sample NSD Template (in YAML) for IMS orchestration

```yaml
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
  member-vlds:
    mgmt-if:
      property: internal
    end-point-id: Router-gateway
    description: Router gateway
    network-forwarding-path:
      - name: apn-router-gateway
        type: endpoint
        connection-point: private
```
Sample VNFD (in YAML) for IMS

```yaml
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
  vdu:
    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: ""
```

End to End NFV-vEPC Service Orchestration Reference Architecture

21
The above descriptor is derived from the ETSI MANO Reference document. Some of the tags are described in the table below and the rest can be referred from the information model published in the ETSI reference.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependency</td>
<td>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.</td>
</tr>
<tr>
<td>Driver</td>
<td>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).</td>
</tr>
<tr>
<td>Endpoints</td>
<td>This descriptor is used in the NSD to define the start and termination points for a service.</td>
</tr>
<tr>
<td>Forwarding-graphs</td>
<td>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.</td>
</tr>
<tr>
<td>Flavors</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>deployment_artifact</td>
<td>Specify path of the userdata file</td>
</tr>
<tr>
<td>driver</td>
<td>Specify the class path to the specific driver which is placed under vnfm\manager\drivers</td>
</tr>
<tr>
<td>image</td>
<td>Specify path of the image</td>
</tr>
<tr>
<td>image-id</td>
<td>Specify image uuid present in glance</td>
</tr>
<tr>
<td>vm-spec</td>
<td>The following child tags should be updated: Ram to be allocated to the VNF, Disk to be allocated to the VNF, User ID with pseudo privileges to enable NETCONF configuration on the VNF, Password for the above user ID, Any additional storage to be provisioned above minimum disk</td>
</tr>
</tbody>
</table>
Sample Heat Template

```yaml
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:
      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:
            type: OS::Neutron::Subnet
            properties:
              name: private_subnet
              network_id: {Ref: network_private}
              cidr: { get_param: private }
              ip_version: 4

        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 }
```
Appendix C

Installation of Softphone

These instructions detail how to configure the Zoiper Android/iOS SIP client to work with a Clearwater IMS system.

Download the application from the Play Store or iTunes.

Softphone Account Signup/Registration

- Enter Ellis node ip in the URL (http://<ellis-ip>)
- Click on the Signup
- Enter Name, Password, Email, Signup code and click on "Sign up"

Note the account name and password created.
(Ex: 6505550XXX@test.com, Password:YYYYYYY)

Configuration of Softphone

Download the Zoiper application from the Play Store or iTunes.

- Once installed, go to Config -> Accounts -> Add account -> SIP
- Fill in the following details:-
  » Account name: 6505550XXX@test.com
  » Host: the root domain of your Clearwater deployment
  » Username: 6505550XXX
  » Password:YYYYYYY
  » Authentication user: 6505550XXX@test.com
  » Outbound proxy: <SIP PROXY>

- Click 'Network Settings' and fill in the following details:
  » Transport type: TCP/UDP
  » STUN Server: <Server ip>

- Hit Save
- If your account was successfully enabled you should see a green tick notification
- Go back to the main Config menu and select Codecs
  » Unselect everything except uLaw and aLaw.
  » Hit Save

You are now ready to make calls.
## References

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>[NFV E2E Arch]</td>
<td>Network Function Virtualization Reference Architecture</td>
</tr>
<tr>
<td></td>
<td>(<a href="http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf">http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf</a>)</td>
</tr>
<tr>
<td>[OpenVNFManager]</td>
<td>TCS' OpenVNFManager (<a href="https://github.com/TCS-TelcoCloud/OpenVNFManager">https://github.com/TCS-TelcoCloud/OpenVNFManager</a>)</td>
</tr>
<tr>
<td>[OpenStack]</td>
<td><a href="http://www.openstack.org">http://www.openstack.org</a></td>
</tr>
</tbody>
</table>