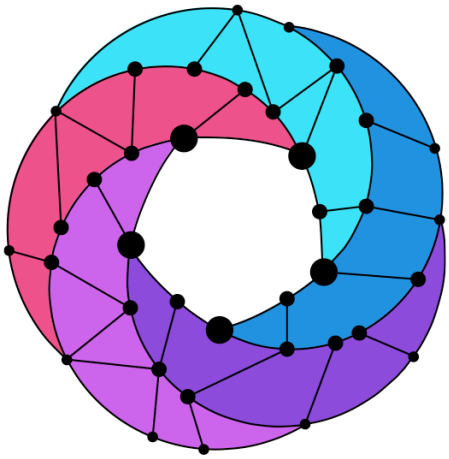


Open Source MANO

OSM Data Model

Release 0

May 1, 2016

Table of Contents

NFV MANO Descriptors

Today's service providers have a growing interest in migrating custom, hardware-based network functions to the cloud. Cloud-based network functions—referred to as virtual network functions (VNFs)—are the software implementation of network functions that can be deployed on a network function virtualization infrastructure (NFVI). VNFs break the dependence on custom hardware appliances, allowing network functions, such as DNS, caching, and firewalls, to run in software.

This new cloud model paradigm is referred to as **Network Function Virtualization (NFV)**.

NFV shows promise as a way to deliver VNFs on virtual machines on commodity off-the-shelf (COTS) infrastructure. NFV promises agile service delivery, faster development cycles, and optimal resource utilization. However, obstacles remain to successful NFV adoption, particularly for wireless and secure connectivity use cases at scale.

The promise of NFV can be realized only if the Virtual Network Function (VNF) applications behave in a way that can be easily deployed, scaled, and managed. This document describes various components of the ETSI MANO model. It also provides concrete data models for various descriptors, including VNF descriptor, Virtual Link Descriptor, VNF Forwarding Graph Descriptor, and Network Service descriptor.

ETSI MANO Common Object Models

The ETSI MANO object models support simple onboarding, deployment, and management of virtual network functions. With extensive support for Enhanced Platform Awareness (EPA), the descriptors enable network function suppliers and network service providers to deploy virtual network functions quickly and easily, in the most cost-efficient manner.

VNF Descriptors (VNFDs) contain the attributes of a VNF necessary to define its platform resource requirements (CPU, memory, interfaces, network, and so on), plus special characteristics related to EPA attributes and performance capabilities.

Network Service Descriptors (NSDs) contain the attributes for a group of VNFs, which together constitute a service definition. These attributes contain the relationship requirements of the VNFs chained together as a service.

Virtual Link Descriptors (VLDs) define L2 or L3 links between VNFs in a chain and/or network interface boundaries (NIBs), which provide interconnectivity in and out of the platform.

Physical Network Function Descriptors (PNFDs) contain the attributes of a PNF necessary to integrate with legacy network hardware devices and to define its capabilities, networking requirements, and performance capabilities.

ETSI MANO objects are modeled as YANG objects, which the tool chain can automatically convert into NETCONF objects, XML objects, protocol buffers (protobufs), and GObject introspect-capable data objects by multiple languages (C, C++, Python, LUA).

The following advanced functions are supported:

- Maximize efficiency and performance by intelligently placing workloads on advanced NFV infrastructure capabilities, such as Enhanced Platform Awareness (EPA)
- Interface physical network functions (PNFs) and chain them together logically to create service chains made entirely of VNFs, PNFs, or combinations of both

See Also

Network Functions Virtualisation (NFV); Management and Orchestration; ETSI GS NFV-MAN 001 V1.1.1 (2014-12)

Network Functions Virtualisation (NFV); Architectural Framework; ETSI GS NFV 002 V1.2.1 (2014-12)

Enhanced Platform Awareness Workload Placement

In a legacy, chassis-based architecture, network function suppliers have chosen a specific CPU for the network function. CPUs and the CPU cards are connected through a point-to-point, redundant fabric, such as a Dual Star backplane. The bandwidth and latency are guaranteed across the chassis fabric, and there is often a separate management fabric for separation of data and control. In such architecture, network functions do not have to deal with much variability.

By contrast, datacenter architectures are highly variable and nondeterministic. Virtual machines may be allocated from physical hosts anywhere within the same datacenter, and both the host and the physical links between these hosts can be oversubscribed. The CPU cores within a virtual machine might belong to different sockets on the physical host, leading to cache and memory access issues. This variability can lead to VNFs with completely different performance characteristics, even when they are placed in the same cloud infrastructure.

To ensure deterministic performance, OpenStack Enhanced Platform Awareness (EPA) attributes can increase the efficiency of the network function for high-touch tasks, such as packet forwarding and security. EPA attributes are discovered during the initial allocation of virtual machines from the Virtualized Infrastructure Manager (VIM).

During the VNF instantiation process, the VNF request characteristics are compared to the virtual machine capabilities in order to allocate workload placement across the corresponding VMs. This design supports advanced placement such as:

- Placing high data rate workloads, such as load balancing and bearer plane forwarding, on VMs that support NUMA affinity, hugepage setup, CPU pinning, and PCI pass through or single root I/O virtualization (SR-IOV)
- Placing best-effort workloads, such as statistics gathering or log output, on “vanilla” VMs
- Placing workloads that form part of the same network service (same service chain) in the same switching domain
- Distributing workloads, such as firewalling, DHCP, or other premise-related tasks, to a remote customer premise device
- Providing advanced security capabilities, such as Quick Assist Technology (QAT) crypto assist and Trusted Platform Module

MANO Software Components

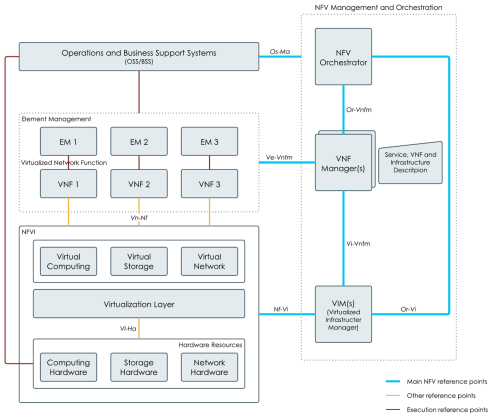
Virtualized Infrastructure Manager (VIM) is responsible for managing compute, network and storage resources. There can be many different implementations of VIM, including OpenStack, AWS and VMware. The VIM provides a northbound interface for the VNF Manager and NFV Orchestrator, and abstracts rest of the system from the details of underlying cloud management platform. The southbound interface should support Nf-Vi interface. The VNF Manager and NFV Orchestrator use the Vi-Vnfm and Or-Vi interfaces to interact with the VIM.

VNF Manager (VNFM) is the process of lifecycle management of the components and services. The VNFM oversees the lifecycle management of VNF instances, as well as:

- Starting the VNF from its descriptor and managing the VNF
- Scaling out/in and up/down of VNFs
- Monitoring and collecting parameters that determine the health of the VNF

NFV Orchestrator (NFV-O) is the process of creating virtual function instances to meet service requirements. The NFV-O manages network service lifecycle and resource orchestration across multiple VIMs. Other NFV-O functionality includes:

- Onboarding new network services and virtual network function packages
- Managing global resources (physical and logical network topology of how various VNFs and PNFs connect)
- Handling policy management related to scalability, reliability, and high availability for network service instances
- Authorizing network functions virtualization infrastructure resource requests
- Manages the NS service templates in the NS catalog
- Simplifies the job of launching new NSs



Reference: ETSI GS NFV 002 V1.2.1 (2014-12)

Designed with open, standards-based APIs, such as NETCONF and REST, and common information models, such as YANG, the Os-ma-nfvo interface is exposed through open, standards-based interfaces such as REST. This design enables upper-level orchestrators, such as Business Process Orchestrators or Service Orchestrators, to automate the entire service bring-up process.

Network Service Descriptor

The Network Service Descriptor (NSD) is the top-level construct used for designing the service chains. The NSD is constructed by using other descriptors, including:

- VNF Descriptor (VNFD)
- PNF Descriptor (PNFD)
- Virtual Link Descriptor (VLD)
- VNF Forwarding Graph Descriptor (VNF-FGD)

The NSD references one or more VNFDs. These VNFs are connected VLDs, and the VNF-FGD determines the traffic flow in the service chain. In addition, the NSD exposes a set of connection points to enable connectivity to other Network Services or to the external world.

NSD Data Model

ID	Type	Cardinality	Description
id	string	1	Unique identifier for the Network Service Descriptor (NSD).
name	string	1	NSD name.
short_name	string	1	NSD short name.
vendor	string	1	Vendor of the NSD.
logo	string	1	File path of the vendor-specific logo. For example, `icons/mylogo.png` The logo should be part of the network service package
description	string	1	Description of the NSD.
version	string	1	Version of the NSD.
connection-point	list	0..n	List for network service (NS) connection points. Each NS has one or more external connection points used to link the NS to other NS or to external networks. Each NS exposes these connection points to the orchestrator. The orchestrator can construct network service chains by connecting the connection points between different NS. See "NS Connection Point " on page 9.
vld	list	0..n	List of Virtual Link Descriptors (VLD). The VLD describes how VNFs in the NSD are connected. See "Virtual Link Descriptor" on page 10.

ID	Type	Cardinality	Description
constituent-vnfd	list	0..n	List of Virtual Network Function Descriptors (VNFDs) that are part of this network service. See "Constituent VNFD" on page 11
scaling-group-descriptor	list	0..n	Scaling group descriptor within this network service. Scaling group defines a group of VNFs, and the ratio of VNFs in the network service that is used as target for scaling action. See "Scaling Group" on page 12.
placement-groups	list	0..n	List of placement groups at the NS level. See "NSD Placement Groups" on page 11.
vnf-dependency	list	0..n	List of VNF dependencies. See "VNF Dependency " on page 13.
vnffgd	list	0..n	List of VNF forwarding graph descriptor (VNFFD). See "VNF Forwarding Graph Descriptor" on page 19.
monitoring-param	grouping	0..n	List of monitoring parameters at the NS level. See "NSD Monitoring Parameter " on page 13.
input-parameter-xpath	list	0..n	List of xpath to parameters inside the NSD that can be customized during instantiation. See "Input Parameter xpath" on page 16.
parameter-pool	list	0..n	Pool of parameter values from which configuration pulls. See "Parameter Pool" on page 16
service-primitive	list	0..n	Network service level configuration primitives. See "Service Primitive" on page 17.

See Also

"MANO YANG Models" on page 45

"NSD YANG Model" on page 72

NS Connection Point

A list for Network Service (NS) connection points. Each NS has one or more external connection points used to link the NS to other NS or to external networks. Each NS exposes these connection points to the orchestrator. The orchestrator can construct network service chains by connecting the connection points between different network services.

nsd:connection-point

ID	Type	Cardinality	Description
name	string	1	Identifier for the external connection point.
type	enum	1	Type of connection point: VPORT: Virtual Port

Virtual Link Descriptor

A list of Virtual Link Descriptors (VLD). The VLD describes the basic topology of connectivity between one or more virtual network functions.

nsd:vld

ID	Type	Cardinality	Description
id	string	1	Unique identifier for the VLD.
name	string	1	VLD name.
short_name	string	1	NSD short name
vendor	string	1	Vendor of the VLD.
description	string	1	Description of the VLD.
version	string	1	Version of the VLD.
type	enum	1	Type of the virtual link. ELAN: A multipoint service connecting a set of VNFs.
root-bandwidth	uint64	1	For ELAN this is the aggregate bandwidth.
leaf-bandwidth	uint64	1	For ELAN this is the bandwidth of branches.
vnfd-connection-point-ref	list	0..n	Reference to VNFD connection points. See "nsd:vld:vnfd-connection-point-ref" below.
provider-network	container	1	Information about the provider network. See "nsd:vld:provider-network " on page 11.

nsd:vld:vnfd-connection-point-ref

ID	Type	Cardinality	Description

ID	Type	Cardinality	Description
member-vnf-index-ref	reference	1	Reference to member-vnf within constituent units. path "../../../nsd:constituent-vndf/nsd:member-vnf-index"
vnf-id-ref	string	1	Reference to a VNFD.
vnfd-connection-point-ref	string	1	Reference to a connection point name in a VNFD.

nsd:vld:provider-network

ID	Type	Cardinality	Description
physical-network	string	1	Name of the physical network on which the provider network is built.
overlay-type	enum	0..1	Identifies the type of the overlay network. Value can be: • LOCAL • FLAT • VLAN • VXLAN • GRE
segmentation-id	uint32	1	Segmentation ID.

Constituent VNFD

A list of Virtual Network Function Descriptors (VNFDs) that are part of this network service.

nsd:constituent-vnfd

ID	Type	Cardinality	Description
member-vnfd-index	uint64	1	Identifier/index for the Virtual Network Function Descriptor (VNFD). This separate ID is required to ensure that multiple VNFs can be part of a single network service.
vnfd-id-ref	reference	1	Identifier for the VNFD.
start-by-default	boolean	1	VNFD is started as part of the NS instantiation. Default is *true*.

NSD Placement Groups

A list of placement groups at the network service level.

nsd:placement-groups

ID	Type	Cardinality	Description
name	string	1	Place group construct to define the compute resource placement strategy in cloud environment.
requirement	string	1	This is free text space used to describe the intent/rationale behind this placement group. This is for human consumption only
strategy	enum	1	Strategy associated with this placement group. Following values are possible • COLOCATION: [Default] Colocation strategy imply intent to share the physical infrastructure (hypervisor/network) among all members of this group. • ISOLATION: Isolation strategy imply intent to not share the physical infrastructure (hypervisor/network) among the members of this group
constituent-vnfd	list	0..n	List of VNFDs that are part of this placement group. See "nsd:placement-groups:constituent-vnfd" below.

nsd:placement-groups:constituent-vnfd

ID	Type	Cardinality	Description
member-vnf-index-ref	reference	1	Member VNF index of this member VNF. path "../../../constituent-vnfd/member-vnf-index"
vnfd-id-ref	reference	1	Identifier for the VNFD. path "/vnfd:vnfd-catalog/vnfd:vnfd/vnfd:id"

Scaling Group

The scaling group descriptor within this network service. A scaling group defines a group of VNFs and the ratio of VNFs in the network service that is used as target for scaling action.

nsd:scaling-group-descriptor

ID	Type	Cardinality	Description
name	string	1	Name of the scaling group.
vnfd-member	list	0..n	List of the VNFs in this scaling group. See "nsd:scaling-group-descriptor:vnfd-member" on page 13.
min-instance-count	uint32	1	Minimum instances of the scaling group which are allowed. These instances are created by default when the network service is instantiated. Default is 0.
max-instance-count	uint32	1	Maximum instance of this scaling group that are allowed in a single network service. The network service scaling will fail when the number of service group instances exceeds the max-instance-count specified. Default is 10.

ID	Type	Cardinality	Description
scaling-config-action	list	0..n	List of scaling config actions. See "nsd:scaling-group-descriptor:scaling-config-action" below.

nsd:scaling-group-descriptor:vnfd-member

ID	Type	Cardinality	Description
member-vnf-index-ref	reference	1	Member VNF index of this member VNF. path "../../../constituent-vnfd-member-vnfd-index"
count	uint32	1	Count of this member VNF within this scaling group/ The count allows to define the number of instances when a scaling action targets this scaling group. Default is 1.

nsd:scaling-group-descriptor:scaling-config-action

ID	Type	Cardinality	Description
trigger	scaling-trigger	1	Scaling trigger.
ns-config-primitive-name-ref	reference	1	Reference to the NS config name primitive. path "../../../config-primitive/name

VNF Dependency

nsd:vnf-dependency

ID	Type	Cardinality	Description
vnf-source-ref	reference	1	Reference to a VNFD. path "../../../vnfd:vnfd-catalog/vnfd:vnfd/vnfd:id"
vnf-depends-on-ref	reference	1	Reference to VNFD that on which the source VNF depends. path "../../../vnfd:vnfd-catalog/vnfd:vnfd/vnfd:id"

NSD Monitoring Parameter

nsd:monitoring-param

ID	Type	Cardinality	Description
http-endpoint	list	0..n	List of HTTP endpoints to be used by the monitoring parameters. See "nsd:monitoring-param:http-endpoint" on page 14
monitoring-param	list	0..n	List of monitoring params at the network service level.

ID	Type	Cardinality	Description
			See "nsd:monitoring-param:monitoring-param" below

nsd:monitoring-param:http-endpoint

ID	Type	Cardinality	Description
path	string	1	The HTTP path on the management server.
https	boolean	1	Pick HTTPS instead of HTTP. Default is *false*.
port	inet:port-number	1	HTTP port to connect to.
username	string	1	HTTP basic auth user name.
password	string	1	HTTP basic auth password.
polling_interval_secs	uint8	1	HTTP polling interval in seconds. Default is 2.
method	enum	1	Method to be performed at the URI. Values can be: • `GET` (default) • `POST` • `PUT` • `GET` • `DELETE` • `PATCH` • `OPTIONS`
headers	list	0..n	List of custom HTTP headers to put on the HTTP request. See "nsd:monitoring-param:http-endpoint:headers" below

nsd:monitoring-param:http-endpoint:headers

ID	Type	Cardinality	Description
key	string	1	HTTP header key.
value	string	1	HTTP header value.

nsd:monitoring-param:monitoring-param

ID	Type	Cardinality	Description
id	string	1	Unique identifier for the monitoring parameter.
name	string	1	Name of the monitoring parameter.
http-endpoint-ref	reference	1	path "../../../http-endpoint/path"

ID	Type	Cardinality	Description
json-query-method	enum	1	The method to extract a value from a JSON response. • NAMEKEY: [Default] Use the name as the key for a non-nested value. • JSONPATH: Use jsonpath-rw implemenation to extract a value. • OBJECTPATH: Use objectpath implemenation to extract a value.
json-query-params	container	1	Object for JSON query parameters. See "nsd:monitoring-param:monitoring-param:json-query-params" below
description	string	1	Description of the monitoring parameter.
group-tag	string	1	Tag to group monitoring parameters.
value-type	enum	1	The type of the parameter value. Value can be: • INT (default) • DECIMAL • STRING
numeric-constraints	container	1	Constraints for the numbers. See "nsd:monitoring-param:monitoring-param:numeric-constraints" on page 16
text-constraints	container	1	Constraints for the string. See "nsd:monitoring-param:monitoring-param:text-constraints" on page 16
value-integer	int64	1	Current value for integer parameter description.
value-decimal	decimal164	1	Current value for decimal parameter.
value-string	string	1	Current value for the string parameter.
widget-type	enum	1	Type of the widget used by the RIFT.ware UI. Value can be: • HISTOGRAM • BAR • GAUGE • SLIDER • COUNTER • TEXTBOX
units	string	1	Units for the monitoring parameter, such as megabits per second.

nsd:monitoring-param:monitoring-param:json-query-params

ID	Type	Cardinality	Description
json-path	string	1	The JSON path used to extract value from the JSON structure.

ID	Type	Cardinality	Description
object-path	string	1	The object path to use to extract value form the JSON structure.

nsd:monitoring-param:monitoring-param:numeric-constraints

ID	Type	Cardinality	Description
min-value	uint64	1	Minimum value for the parameter.
max-value	uint64	1	Maximum value for the parameter.

nsd:monitoring-param:monitoring-param:text-constraints

ID	Type	Cardinality	Description
min-length	uint8	1	Minimum string length for the parameter.
max-length	uint8	1	Maximum string length for the parameter.

Input Parameter xpath

nsd:input-parameter-xpath

ID	Type	Cardinality	Description
xpath	string	1	An xpath that specifies the element in a descriptor.
label	string	1	A descriptive string.
default-value	string	1	A default value for this input parameter.

Parameter Pool

nsd:parameter-pool

ID	Type	Cardinality	Description
name	string	1	Name of the configuration value pool.
range	container	1	Create a range of values from which to populate the pool. See "nsd:parameter-pool:range" below

nsd:parameter-pool:range

ID	Type	Cardinality	Description
start-value	uint32	1	[Required] Generated pool values start at this value.
end-value	uint32	1	[Required] Generated pool values end at this value.

Service Primitive

nsd:service-primitive

ID	Type	Cardinality	Description
name	string	1	Name of the configuration primitive.
parameter	list	0..n	List of parameters to the configuration primitive. See "nsd:service-primitive:parameter" below.
parameter-group	list	0..n	Grouping of parameters that are logically grouped in UI. See "nsd:service-primitive:parameter-group" below.
vnf-primitive-group	list	0..n	List of configuration primitives grouped by VNF. See "nsd:service-primitive:vnf-primitive-group" on page 18.
used-defined-script	string	1	A user defined script

nsd:service-primitive:parameter

ID	Type	Cardinality	Description
name	string	1	Name of parameter.
data-type	enum	1	Data type associated with the name. Value can be: • STRING • INTEGER • BOOLEAN
mandatory	boolean	1	Specifies whether this field is mandatory. Default is *false*.
default-value	string	1	The default value for the field.
parameter-pool	string	1	NSD parameter pool name to use for this parameter.
read-only	boolean	1	The value should be dimmed by the UI. Applies only to parameters with default values.
hidden	boolean	1	The value should be hidden by the UI. Applies only to parameters with default values

nsd:service-primitive:parameter-group

ID	Type	Cardinality	Description
name	string	1	Name of parameter group.
parameter	list	0..n	List of parameters to the configuration primitive. See "nsd:service-primitive:parameter-group:parameter" on page 18

ID	Type	Cardinality	Description
mandatory	boolean	1	Specifies whether this group mandatory. Default is *true*.

nsd:service-primitive:parameter-group:parameter

ID	Type	Cardinality	Description
name	string	1	Name of parameter.
data-type	enum	1	Data type associated with the name. Value can be: • STRING • INTEGER • BOOLEAN
mandatory	boolean	1	Specifies whether this field is mandatory. Default is *false*.
default-value	string	1	The default value for the field.
parameter-pool	string	1	NSD parameter pool name to use for this parameter.
read-only	boolean	1	The value should be dimmed by the UI. Applies only to parameters with default values.
hidden	boolean	1	The value should be hidden by the UI. Applies only to parameters with default values

nsd:service-primitive:vnf-primitive-group

ID	Type	Cardinality	Description
member-vnf-index-ref	uint64	1	Reference to member-vnf within constituent-vnfds.
vnfd-id-ref	string	1	A reference to a VNFD.
vnfd-name	string	1	Name of the VNFD.
primitive	list	1	A list of VNF primitives. See "nsd:service-primitive:vnf-primitive-group:primitive" below

nsd:service-primitive:vnf-primitive-group:primitive

ID	Type	Cardinality	Description
index	uint32	1	Index of this primitive
name	string	1	Name of the primitive in the VNF primitive

VNF Forwarding Graph Descriptor

A Virtual Network Function Forwarding Graph (VNFFG) is a graph specified by a Network Service Provider of bi-directional logical links connecting NF nodes, where at least one node is a VNF through which network traffic is directed.

The VNFFG descriptor contains metadata about the VNF forwarding graph itself, as well as references to VLs, VNFs and PNFs, and network forwarding path elements. These elements include policies, such as MAC forwarding rules and routing entries, and references to connection points, such a virtual ports and virtual NIC addresses.

VNFFGD Data Model

ID	Type	Cardinality	Description
id	string	1	Unique identifier for the VNFFGD.
name	string	1	VNFFGD name.
short_name	string	1	VNFFGD short name for the UI.
vendor	string	1	Provider of the VNFFGD.
description	string	1	Description of the VNFFGD.
version	string	1	Version of the VNFFGD.
rsp	list	0..n	List of the Rendered Service Paths (RSP). See "Rendered Service Path" on page 20.
classifier	list	0..n	List of classifier rules. See "Classifier" on page 20.

See Also

"MANO YANG Models" on page 45

"NSD YANG Model" on page 72

Rendered Service Path

vnffg:rsp

ID	Type	Cardinality	Description
id	string	1	Unique identifier for the RSP.
name	string	1	RSP name.
vnfd-connection-point-ref	list	0..n	See "vnffg:rsp:vnfd-connection-point-ref" below

vnffg:rsp:vnfd-connection-point-ref

ID	Type	Cardinality	Description
member-vnf-index-ref	reference	1	Reference to member VNF within constituent VNFDs path "../../../nsd:constituent-vnfd-nsd_member-vnf-index"
order	uint8	1	A number that denotes the VNF in a chain.
vnfd-id-ref	string	1	A reference to a VNFD.
vnfd-connection-point-ref	string	1	A reference to a connection point name in a VNFD.

Classifier

vnffg:classifier

ID	Type	Cardinality	Description
id	string	1	Unique identifier for the classifier rule.
name	string	1	Name of the classifier.
rsp-id-ref	reference	1	A reference to the RSP. path "../../nsd:rsp/nsd:id"
member-vnf-index-ref	reference	1	Reference to member-vnf within constituent-vndfs. path "../../../nsd:constituent-vnfd/nsd:member-vnf-index"
vnfd-id-ref	string	1	A reference to a VNFD.
vnfd-connection-point-ref	string	1	A reference to a connection point name in a VNFD.
match-atrributes	list	0..n	List of match attributes. See "vnffg:classifier:match-attributes" on page 21.

vnffg:classifier:match-attributes

ID	Type	Cardinality	Description
id	string	1	Unique identifier for the classifier match-attribute rule.
ip-proto	uint8	1	IP protocol
source-ip-address	inet:ip-address	1	Source IP address.
destination-ip-address	inet:ip-addres	1	Destination IP address.
source-port	inet:port-number	1	Source port.
destination-port	inet:port-number	1	Destination port.

Virtual Network Function Descriptor

The Virtual Network Function Descriptor (VNFD) describes the virtual network function and contains the following information:

- Individual VNF components that are part of the VNF
- Internal and external connectivity details
- KPI requirements and auto-scaling properties

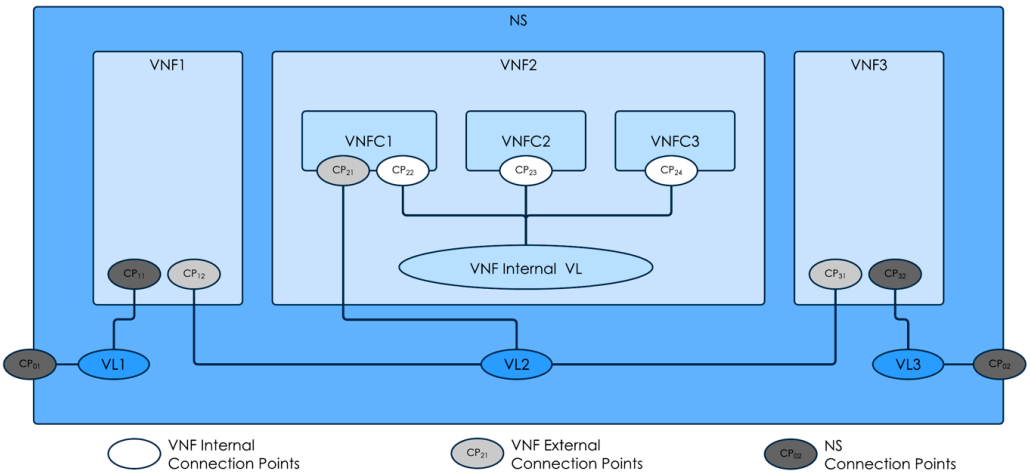
The VNF Manager uses the VNFD to start a new instance of a VNF.

Each VNF is constructed from a set of discrete VNF Components (VNFCs). There can be one or more instance of each VNFC in the VNF. The VNFC is realized using a virtualized compute resource from the VIM. The virtualized compute resource can be either a Virtual Machine (VM) or a container. The VNFC includes a full description of software components that will run inside the virtualized compute resource.

The elements contained in the Virtual Deployment Unit (VDU) define the compute resource and software components. Specifically, VDU captures information about storage, memory, CPU, networking resources, and the software components inside the VM.

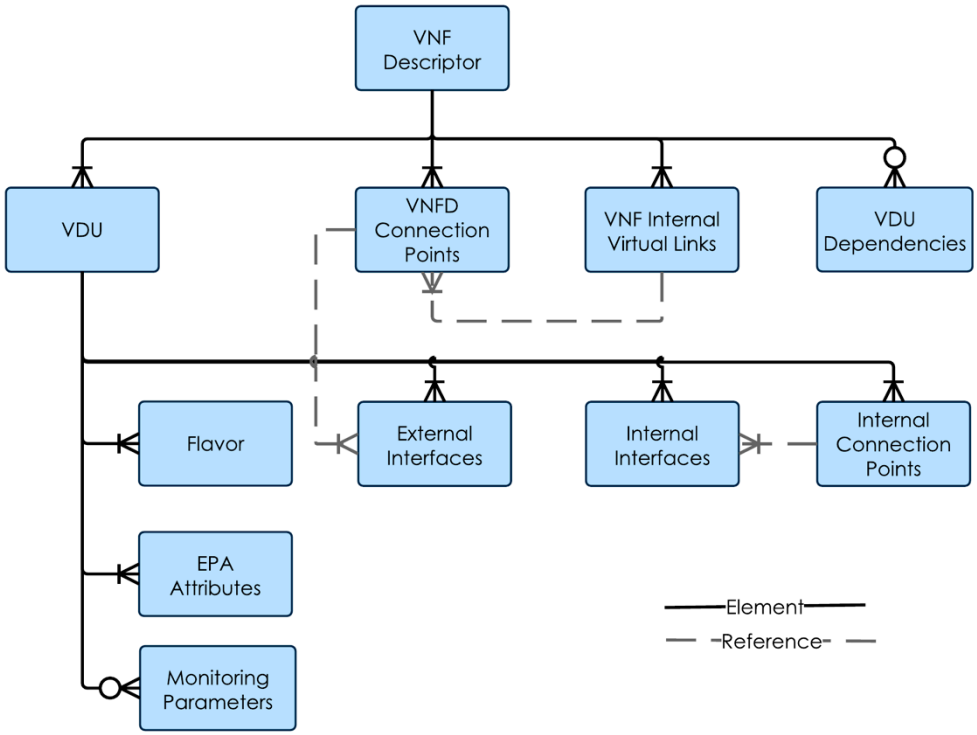
Each VNF has a set of internal and external connection points, which abstract the actual virtual interface used by the VM/container. The virtual interface in the VM is assigned a connection point. Internal connection points are used to connect the VNFC internals to the VNF. The connection points, whether internal or external, are connected using virtual links. Each virtual link has references to two or more connection points.

The following diagram illustrates how VNFs are connected using internal and external virtual links. VL1, VL2 and VL3 represent external virtual links. VL2 is used to connect the VNF1, VNF2 and VNF3. Virtual links VL1 and VL3 are used to connect the network services.



(Reference: ETSI GS NFV 001 V1.1.1 (2014-12))

The following diagram illustrates the high-level object model for the VNFD. The VNFD contains a list of VDUs, a list of internal connection points, a list of internal virtual links and a list of external connection points. The internal connection points and internal virtual links define how the VMs inside the VNF will be connected. The external connection points will be used by the NSD to chain VNFs. The VDUs define the individual VNF components. The VDU captures information about VM image, VM flavor, and EPA attributes.



(VNFD high-level object model)

VNFD Enhanced Platform Awareness Elements

Enhanced Platform Awareness (EPA) is designed to improve the performance of guest virtual machines on the hypervisors by enabling fine-grained matching of workload requirements to platform capabilities, before the VM is launched. EPA includes the ability to:

- Launch cryptographic workload on a VM with cryptographic resources
- Launch high-bandwidth workload on a VM with DPDK capabilities
- Use vCPU to pCPU pinning and hugepages to improve NFV workload performance
- Configure PCI pass-through

EPA capabilities are captured in the VNFD Virtual Deployment Unit. The VNFD descriptor contains elements to capture these EPA attributes.

Note: On public clouds, such as AWS, many EPA features are not applicable.

Hugepages

The use of hugepages can improve network performance. Because fewer pages are needed there are fewer Translation Look-aside Buffers (TLBs, high speed translation caches). Without standard 4k pages, high TLB miss rates could affect performance.

OpenStack `hw:mem_page_size` flavor filter supports the allocation of hugepages. The following page sizes are supported:

- Large (2MB or 1GB)
- Small (4K)
- Any
- 2MB
- 1GB

CPU Pinning

Often in OpenStack deployments, hosts are configured to permit over-commit of CPUs. If conflict occurs between two guests VMs, there could be extended periods when the guest vCPU is not scheduled by the host. This scenario can result in unpredictable latency, which could affect certain types of workloads. To avoid a latency issue, the guest should be pinned to a dedicated physical CPU.

OpenStack `hw:cpu_policy` and `hw:cpu_threads_policy` filters control the CPU pinning behavior of the guest VMs. The `hw:cpu_policy` supports the shared and dedicated options. In the shared mode, the guest CPUs are not pinned to the physical CPUs. In the dedicated mode, the guest CPUs are pinned to the physical CPUs. For example, to launch a VM with vCPUs pinned to the physical CPUs, the following flavor key must be set:

```
hw:cpu_policy=dedicated
```

The `hw:cpu_threads_policy` supports avoid, separate, isolate, and prefer options. The avoid thread policy avoids placing the guest on a host with CPU threads. The separate thread policy places the vCPUs on different cores and avoids placing two vCPUs on two threads of the same core. The isolate policy places each vCPU on a different core, and places no vCPUs from a different guest on the same core. The prefer policy places the vCPUs on the same core.

Guest NUMA Awareness

When running workloads on NUMA hosts, the CPUs executing the processes should be on the same node as the memory used. This configuration ensures that all memory accesses are local to the NUMA node and not consumed by the limited cross-node memory bandwidth, which adds latency to memory accesses. PCI devices are directly associated with specific NUMA nodes for the purposes of DMA. When you use PCI device assignment, place the guest VM on the

same NUMA node as any device that is assigned to it. If the RAM/vCPUs associated with a flavor are larger than any single NUMA node, it is important to expose NUMA topology to the guest so that the guest operating system can schedule workloads it runs. For this setup to work, the guest NUMA nodes must be directly associated with host NUMA nodes.

OpenStack uses the following filters to configure the guest NUMA topology:

- `hw:numa_nodes`
- `hw:numa_mempolicy`
- `hw:numa_cpus`
- `hw:numa_mem`

Example

```
hw:numa_nodes=NN - numa of NUMA nodes to expose to the guest.
 hw:numa_mempolicy=preferred|strict - memory allocation policy
 hw:numa_cpus.0=<cpu-list> - mapping of vCPUS N-M to NUMA node
 hw:numa_cpus.1=<cpu-list> - mapping of vCPUS N-M to NUMA node 1
 hw:numa_mem.0=<ram-size> - mapping N GB of RAM to NUMA node 0
 hw:numa_mem.1=<ram-size> - mapping N GB of RAM to NUMA node 1
```

The `hw:numa_mempolicy` is either preferred or strict. In the strict mode, the memory for the NUMA node in the guest must come from the corresponding NUMA node on the host.

PCI Pass-Through

Guest virtual machines might need direct access to the PCI devices to avoid contention with other VMs. In addition, PCI pass-through significantly enhances performance since the hypervisor layer is bypassed. For example, PCI pass-through may be used to provide a guest exclusive and direct access to a NIC or Crypto resource.

OpenStack uses `pci_pass-through:alias` filter to assign a PCI device to the VM. This also requires configuration of PCI pass-through whitelist on each compute node and PCI pass-through alias on the controller node. The following examples show configuration of PCI pass-through whitelist and PCI alias.

```
pci_pass-through_whitelist=[{ "vendor_id":"8086","product_id":"1520"}]
 pci_alias={"vendor_id":"8086", "product_id":"1520", "name":"a1"}
```

Data Direct I/O

Intel Data Direct I/O (DDIO) enables Ethernet server NICs. Controllers talk directly to the processor cache without a detour through system memory. Intel DDIO makes the processor cache the primary destination and source of I/O data rather than main memory. By avoiding system memory, Intel DDIO reduces latency and increases system I/O. DDIO is enabled in the BIOS of the host.

Cache Monitoring Technology

Cache Monitoring Technology (CMT) allows an operating system, hypervisor, or similar system management agent to determine the usage of cache based on applications running on the platform. The implementation is directed at L3 cache monitoring (currently the last-level cache in most server and client platforms).

Cache Allocation Technology

Cache Allocation Technology (CAT) allows an operating system, hypervisor, or similar system management agent to specify the amount of L3 cache (currently the last-level cache in most server and client platforms) space an application can fill.

Note: As a hint to hardware functionality, certain features such as power management, may override CAT settings.

VNFD Data Model

ID	Type	Cardinality	Description
id	string	1	Identifier for the VNFD.
name	string	1	VNFD name.
short-name	string	1	VNFD short name.
vendor	string	1	Vendor of the VNFD.
logo	string	1	File path of the vendor-specific logo. For example, `icons/mylogo.png` The logo should be part of the VNF package.
description	string	1	Description of the VNFD.
version	string	1	Version of the VNFD.
mgmt-interface	container	1	Interface over which the VNF is managed. See "Management Interface" on page 27.
internal-vld	list	0..n	A list of internal virtual links to connect various VNF components. The MANO document [1] refers this as “virtual-link”, however we are calling this field “internal-vlds” to explicitly indicate that these are internal virtual links. Also, we plan to use the same virtual link descriptor for both internal and external descriptors. If the VNF is composed of only one VNF component/VM, no internal VLD may be required. See "Internal VLD " on page 32.

ID	Type	Cardinality	Description
connection-point	list	0..n	The list for external connection points. Each VNF has one or more external connection points. As the name implies the external connection points are used for connecting the VNF to other VNFs or to external networks. Each VNF exposes these connection points to the orchestrator. The orchestrator can construct network services by connecting the connection points between different VNFs. The NFVO will use VLDs and VNFFGs at the network service level to construct network services. See "VNFD Connection Point " on page 32.
vdu	list	0..n	List of virtual deployment units. See "VDU Data Model" on page 35.
vdu-dependency	list	0..n	List of VDU dependencies. The orchestrator uses this list to determine the order of startup for VDUs. See "VDU Dependency " on page 44.
service-function-chain	enum	1	Type of node in Service Function Chaining Architecture. Type can be: • UNAWARE (default) • CLASSIFIER • SF • SFF
service-function-type	string	1	Type of service function. **Note:** This value needs to map with service function type in ODL to support VNFFG. Service Function Type is manadatory param in ODL SFC. This is temporarily set to string for ease of use.
monitoring-param	list	0..n	List of monitoring parameters for the VNF. See "NSD Monitoring Parameter " on page 13.
placement-groups	list	0..n	List of placement groups at the VNF level. See

See Also

"MANO YANG Models" on page 45

"VNFD YANG Model" on page 85

Management Interface

vnfd:mgmt-interface

ID	Type	Cardinality	Description

ID	Type	Cardinality	Description
vnf-configuration	container	1	Information regarding the VNF configuration is captured here. See "VNF Configuration" on page 29.
endpoint-type	choice	1	Indicates the type of management endpoint. Values can be: • IP (see "vnfd:mgmt-interface:endpoint-type:ip" below) • VDU-ID (see "vnfd:mgmt-interface:endpoint-type:vdu-id" below) • CP (see "vnfd:mgmt-interface:endpoint-type:cp" below)
port	inet:port-number	1	Port number for the management interface.
dashboard-params	container	1	Parameters for the VNF dashboard. See "vnfd:mgmt-interface:dashboard-params" below.

vnfd:mgmt-interface:endpoint-type:ip

ID	Type	Cardinality	Description
ip	inet:ip-address	1	Specifies the static IP address for managing the VNF

vnfd:mgmt-interface:endpoint-type:vdu-id

ID	Type	Cardinality	Description
vdu-id	reference	1	Use the default management interface on this VDU. path "/vnfd:vnfd-catalog/vnfd:vnfd/vnfd:vdu/vnfd:id"

vnfd:mgmt-interface:endpoint-type:cp

ID	Type	Cardinality	Description
cp	reference	1	Use the ip address associated with this connection point. path "/vnfd:vnfd-catalog/vnfd:vnfd/vnfd:connection-point/vnfd:name"

vnfd:mgmt-interface:dashboard-params

ID	Type	Cardinality	Description
path	string	1	The HTTP path for the dashboard.
https	boolean	1	Pick HTTPS instead of HTTP. Default is *false*.
port	inet:port-number	1	The HTTP port for the dashboard

VNF Configuration

vnfd:mgmt-interface:vnf-configuration

ID	Type	Cardinality	Description
config-method	choice	1	Defines the configuration method for a VNF. Choices are: • NETCONF (see "vnfd:mgmt-interface:config-method:netconf" below) • REST (see "vnfd:mgmt-interface:config-method:rest" on page 30) • SCRIPT (see "vnfd:mgmt-interface:config-method:script" on page 30) • JUJU (see "vnfd:mgmt-interface:config-method:juju" on page 30)
config-access	container	1	IP address to be used to configure this VNF. **Note:** This parameter is optional if it is possible to dynamically resolve the IP. See "vnfd:mgmt-interface:vnf-configuration:config-access" on page 30
input-params	container	1	Miscellaneous input parameters to be considered while processing the NSD to apply configuration. See "vnfd:mgmt-interface:vnf-configuration:input-params" on page 30.
config-attributes	container	1	Miscellaneous config attributes to be considered while processing the NSD to apply configuration. See "vnfd:mgmt-interface:vnf-configuration:config-atrributes" on page 30.
config-primitive	list	0..n	List of configuration primitives supported by the configuration agent for this VNF. See "vnfd:mgmt-interface:vnf-configuration:config-primitive" on page 31
initial-config-primitive	list	0..n	Initial set of configuration primitives. See "vnfd:mgmt-interface:vnf-configuration:initial-config-primitive" on page 31.
config-template	string	1	Configuration template for each VNF.

vnfd:mgmt-interface:config-method:netconf

ID	Type	Cardinality	Description
target	enum	1	NETCONF configuration target. Values are: • RUNNING • CANDIDATE
protocol	enum	1	Protocol to use, such as for NETCONF. Values are: • NONE • SSH

ID	Type	Cardinality	Description
port	inet:port-number	1	Port for the NETCONF server.

vnfd:mgmt-interface:config-method:rest

ID	Type	Cardinality	Description
port	inet_port-number	1	Port for the REST server

vnfd:mgmt-interface:config-method:script

ID	Type	Cardinality	Description
script-type	enum	1	Script type to use. Values are: • BASH • EXPECT

vnfd:mgmt-interface:config-method:juju

ID	Type	Cardinality	Description
charm	string	1	Juju charm to use to use with the VNF.

vnfd:mgmt-interface:vnf-configuration:config-access

ID	Type	Cardinality	Description
mgmt-ip-address	inet:ip-address	1	IP address to be used to configure this VNF. Optional if it is possible to resolve dynamically.
username	string	1	User name for configuration.
password	string	1	Password for configuration access authentication.

vnfd:mgmt-interface:vnf-configuration:config-atrributes

ID	Type	Cardinality	Description
config-priority	uint64	1	Configuration priority, order of configuration to be applied to each VNF in this NS. A low number takes precedence over a high number.
config-delay	uint64	1	Wait time (in seconds) before applying the configuration to this VNF.

vnfd:mgmt-interface:vnf-configuration:input-params

ID	Type	Cardinality	Description
config-priority	uint64	1	Configuration priority , order of configuration to be applied to each VNF in this NS. A low number takes precedence over a high number.

ID	Type	Cardinality	Description
config-delay	uint64	1	Wait time (in seconds) before applying the configuration to this VNF.

vnfd:mgmt-interface:vnf-configuration:config-primitive

ID	Type	Cardinality	Description
name	string	1	Name of the configuration primitive.
parameter	list	0..n	List of parameters to the configuration primitive. See "vnfd:mgmt-interface:vnf-configuration:config-primitive:parameter" below

vnfd:mgmt-interface:vnf-configuration:config-primitive:parameter

ID	Type	Cardinality	Description
name	string	1	Name of parameter
data-type	parameter-data-type	1	Data type associated with the name
mandatory	boolean	1	Defines whether this field is mandatory. Default is *false*.
default-value	string	1	The default value for the field.
parameter-pool	string	1	Parameter pool name to use for this parameter.
read-only	boolean	1	The value should be dimmed by the UI. Applies only to parameters with default values.
hidden	boolean	1	The value should be hidden by the UI. Applies only to parameters with default values.

vnfd:mgmt-interface:vnf-configuration:initial-config-primitive

ID	Type	Cardinality	Description
seq	uint64	1	Sequence number for the configuration primitive.
name	string	1	Name of the configuration primitive.
parameter	list	0..n	See "vnfd:mgmt-interface:vnf-configuration:config-primitive:parameter" above

vnfd:mgmt-interface:vnf-configuration:initial-config-primitive:parameter

ID	Type	Cardinality	Description
name	string	1	Name of the parameter.
value	string	1	Value of the parameter.

Internal VLD

vnfd:internal-vld

ID	Type	Cardinality	Description
id	string	1	Unique identifier for the internal virtual link descriptor (VLD).
name	string	1	Name of the internal VLD.
short-name	string	1	Short name for the internal VLD.
description	string	1	Description of the internal VLD.
type	enum	1	Type of virtual link. Currently supports ELAN. ELAN is a multipoint service connecting a set of VDUs.
root-bandwidth	uint64	1	For ELAN this is the aggregate bandwidth.
leaf-bandwidth	uint64	1	For ELAN this is the bandwidth of branches.
internal-connection-point-ref	reference	1	Reference to one or more internal connection points.

VNFD Connection Point

vnfd:connection-point

ID	Type	Cardinality	Description
name	string	1	Name of the connection point.
id	string	1	Unique identifier of the connection point
short-name	string	1	Short name of the connection point for display in the RIFT.ware UI.
type	enum	1	Type of connection point. ELAN: A multipoint service connecting a set of VNFs.

VNFD Monitoring Parameter

vnfd:monitoring-param

ID	Type	Cardinality	Description
http-endpoint	list	0..n	List of HTTP endpoints to be used by the monitoring parameters. See "vnfd:monitoring-param:http-endpoint" on page 33
monitoring-param	list	0..n	List of monitoring params at the network service level. See "vnfd:monitoring-param:monitoring-param" on page 33

vnfd:monitoring-param:http-endpoint

ID	Type	Cardinality	Description
path	string	1	The HTTP path on the management server.
https	boolean	1	Pick HTTPS instead of HTTP. Default is false.
port	inet:port-number	1	HTTP port to connect to.
username	string	1	HTTP basic auth user name.
password	string	1	HTTP basic auth password.
polling_interval_secs	uint8	1	HTTP polling interval in seconds. Default is 2.
method	enum	1	Method to be performed at the URI. Values can be: • GET (default) • POST • PUT • GET • DELETE • PATCH • OPTIONS
headers	list	0..n	List of custom HTTP headers to put on the HTTP request. See "vnfd:monitoring-param:http-endpoint:headers" below

vnfd:monitoring-param:http-endpoint:headers

ID	Type	Cardinality	Description
key	string	1	HTTP header key.
value	string	1	HTTP header value.

vnfd:monitoring-param:monitoring-param

ID	Type	Cardinality	Description
id	string	1	Unique identifier for the monitoring parameter.
name	string	1	Name of the monitoring parameter.
http-endpoint-ref	reference	1	path "../../../http-endpoint/path"
json-query-method	enum	1	The method to extract a value from a JSON response. • NAMEKEY: [Default] Use the name as the key for a non-nested value. • JSONPATH: Use jsonpath-rw implemenation to extract a value. • OBJECTPATH: Use objectpath implemenation to extract a value.

ID	Type	Cardinality	Description
json-query-params	container	1	Object for JSON query parameters. See "vnfd:monitoring-param:monitoring-param:json-query-params" below
description	string	1	Description of the monitoring parameter.
group-tag	string	1	Tag to group monitoring parameters.
value-type	enum	1	The type of the parameter value • INT (default) • DECIMAL • STRING
numeric-constraints	container	1	Constraints for the numbers. See "vnfd:monitoring-param:monitoring-param:numeric-constraints" below
text-constraints	container	1	Constraints for the string. See "vnfd:monitoring-param:monitoring-param:text-constraints" on page 35
value-integer	int64	1	Current value for integer parameter description.
value-decimal	decimal164	1	Current value for decimal parameter.
value-string	string	1	Current value for the string parameter.
widget-type	enum	1	Type of the widget used by the RIFT.ware UI. Value can be: • HISTOGRAM • BAR • GAUGE • SLIDER • COUNTER • TEXTBOX
units	string	1	Units for the monitoring parameter, such as megabits per second.

vnfd:monitoring-param:monitoring-param:json-query-params

ID	Type	Cardinality	Description
json-path	string	1	The JSON path used to extract value from the JSON structure.
object-path	string	1	The object path to use to extract value form the JSON structure.

vnfd:monitoring-param:monitoring-param:numeric-constraints

ID	Type	Cardinality	Description

ID	Type	Cardinality	Description
min-value	uint64	1	Minimum value for the parameter.
max-value	uint64	1	Maximum value for the parameter.

vnfd:monitoring-param:monitoring-param:text-constraints

ID	Type	Cardinality	Description
min-length	uint8	1	Minimum string length for the parameter.
max-length	uint8	1	Maximum string length for the parameter.

VNFD Placement Groups

A list of placement groups at the VNF level.

vnfd:placement-groups

ID	Type	Cardinality	Description
name	string	1	Place group construct to define the compute resource placement strategy in cloud environment.
requirement	string	1	This is free text space used to describe the intent/rationale behind this placement group. This is for human consumption only
strategy	enum	1	Strategy associated with this placement group. Following values are possible • COLOCATION: [Default] Colocation strategy imply intent to share the physical infrastructure (hypervisor/network) among all members of this group. • ISOLATION: Isolation strategy imply intent to not share the physical infrastructure (hypervisor/network) among the members of this group
constituent-vdus	list	0..n	List of VDUs that are part of this placement group. See "vnfd:placement-groups:constituent-vdus" below.

vnfd:placement-groups:constituent-vdus

ID	Type	Cardinality	Description
member-vdu-ref	reference	1	Reference to the VDU in the VNF. path "../../../constituent-vnfd/member-vnf-index"

VDU Data Model

vnfd:vdu

ID	Type	Cardinality	Description

ID	Type	Cardinality	Description
id	string	1	Unique identifier for the Virtual Deployment Unit (VDU).
name	string	1	Unique name for the VDU.
description	string	1	Description of the VDU.
count	uint64	1	Number of instances of the VDU.
mgmt-vpci	string	1	Specifies the virtual PCI address, expressed in the following format dddd:dd:dd.d. For example 0000:00:12.0. This information can be used to pass as metadata during the VM creation.
vm-flavor	container	1	Flavor of the VM, which captures information such as number of VCPUs, memory and disk space. See "VM Flavor " on page 37.
guest-epa	container	1	Captures guest EPA attributes. See "Guest EPA" on page 40.
vswitch-epa	container	1	Captures vSwitch EPA attributes. See "vSwitch EPA " on page 37.
hypervisor-epa	container	1	Captures hypervisor EPA attributes. See "Hypervisor EPA " on page 37.
host-epa	container	1	Captures host EPA attributes. See "Host EPA " on page 38.
image	string	1	[Required] File/URL path or name to the software image. If the image name is found within the VNF package, it will be uploaded to all cloud accounts during tje onboarding process. Otherwise, the image must be added to the cloud account with the same name as entered in this field.
image-checksum	string	1	Image md5sum for the software image. The md5sum, if provided, along with the image name uniquely identifies an image uploaded to the CAL.
cloud-init	string	1	Content of cloud-init script.
internal-connection-point	list	0..n	List for internal connection points. Each VNFC has zero or more internal connection points. Internal connection points are used for connecting the VNF components internal to the VNF. If a VNF has only one VNFC, it may not have any internal connection points. See "VDU Internal Connection Point " on page 43.
internal-interface	list	0..n	List of internal interfaces for the VNF. The external interfaces enable sending traffic to and from VNF.

ID	Type	Cardinality	Description
			See "VDU Internal Interface " on page 43.
external-interface	list	0..n	List of eternal interfaces for the VNF. See "VDU External Interface " on page 44.

See Also

"MANO YANG Models" on page 45

"VNFD YANG Model" on page 85

VM Flavor

vnfd:vdu:vm-flavor

ID	Type	Cardinality	Description
vpcu-count	uint16	1	Number of VCPUs for the VM.
memory-mb	uint16	1	Amount of memory in MB.
storage-gb	uint16	1	Amount of disk space in GB.

vSwitch EPA

vnfd:vdu:vswitch-epa

ID	Type	Cardinality	Description
ovs-acceleration	enum	0..1	Specifies Open vSwitch acceleration mode. • MANDATORY: OVS acceleration is required • PREFERRED: OVS acceleration is preferred • DISABLED: OVS acceleration is disabled.
ovs-offload	enum	0..1	Specifies Open vSwitch hardware offload mode. • MANDATORY: OVS offload is required • PREFERRED: OVS offload is preferred • DISABLED: OVS offload is disabled

Hypervisor EPA

vnfd:vdu:hypervisor-epa

ID	Type	Cardinality	Description

ID	Type	Cardinality	Description
type	enum	0..1	Specifies the type of hypervisor. Value can be: • KVM: KVM • XEN: XEN
version	string	1	Version of the hypervisor.

Host EPA

Specifies the host level EPA attributes.

vnfd:vdu:host-epa

ID	Type	Cardinality	Description
cpu-model	enum	0..1	Host CPU model. The following values are supported: • PREFER_WESTMERE • REQUIRE_WESTMERE • PREFER_SANDYBRIDGE • REQUIRE_SANDYBRIDGE • PREFER_IVYBRIDGE • REQUIRE_IVYBRIDGE • PREFER_HASWELL • REQUIRE_HASWELL • PREFER_BROADWELL • REQUIRE_BROADWELL • PREFER_NEHALEM • REQUIRE_NEHALEM • PREFER_PENRYN • REQUIRE_PENRYN • PREFER_CONROE • REQUIRE_CONROE • PREFER_CORE2DUO • REQUIRE_CORE2DUO
cpu-arch	enum	0..1	Host CPU architecture. The following values are supported: • PREFER_X86 • REQUIRE_X86 • PREFER_X86_64 • REQUIRE_X86_64 • PREFER_I686

ID	Type	Cardinality	Description
			• REQUIRE_I686 • PREFER_IA64 • REQUIRE_IA64 • PREFER_ARMV7 • REQUIRE_ARMV7 • PREFER_ARMV8 • REQUIRE_ARMV8
cpu-vendor	enum	0..1	Host CPU vendor. The following values are supported: • PREFER_INTEL • REQUIRE_INTEL • PREFER_AMD • REQUIRE_AMD
cpu-socket-count	enum	0..1	Number of sockets on the host. The following values are supported: • PREFER_ONE • PREFER_TWO • REQUIRE_ONE • REQUIRE_TWO
cpu-core-count	uint64	1	Number of cores on the host.
cpu-feature	enum	0..1	The following values are supported: • PREFER_AES • REQUIRE_AES • PREFER_CAT • REQUIRE_CAT • PREFER_CMT • REQUIRE_CMT • PREFER_DDIO • REQUIRE_DDIO **NOTE:** Provide both REQUIRE and PREFER options for these features. • AES: CPU supports advanced instruction set for AES (Advanced Encryption Standard). • CAT: Cache Allocation Technology (CAT) allows an operating system, hypervisor, or similar system management agent to specify the amount of L3 cache (currently the last-level cache in most server and client platforms) space an application can fill. As a hint to hardware functionality, certain features such as power management may override CAT settings). • CMT: Cache Monitoring Technology (CMT) allows an Operating System, Hypervisor, or similar system management agent to determine the usage of cache based on applications running on the

ID	Type	Cardinality	Description
			platform. The implementation is directed at L3 cache monitoring (currently the last-level cache in most server and client platforms). • DDIO: Intel Data Direct I/O (DDIO) enables Ethernet server NICs and controllers talk directly to the processor cache without a detour via system memory. This enumeration specifies if the VM requires a DDIO capable host.
om-cpu-model-string	string	1	OpenMano CPU model string
om-cpu-feature	string	1	OpenMano CPU features

Guest EPA

EPA attributes for the guest.

vnfd:vdu:guest-epa

ID	Type	Cardinality	Description
trusted-execution	boolean	1	This VM should be allocated from trusted pool.
mempage-size	enum	0..1	Memory page allocation size. If a VM requires hugepages, it should choose LARGE or SIZE_2MB or SIZE_1GB. If the VM prefers hugepages it should chose PREFER_LARGE. • LARGE: Require hugepages (either 2MB or 1GB) • SMALL: Doesn't require hugepages • SIZE_2MB: Requires 2MB hugepages • SIZE_1GB: Requires 1GB hugepages • PREFER_LARGE: Application prefers hugepages
cpu-pinning-policy	enum	0..1	CPU pinning policy describes association between virtual CPUs in guest and the physical CPUs in the host. • DEDICATED: Virtual CPUs are pinned to physical CPUs • SHARED: Multiple VMs may share the same physical CPUs. • ANY: (Default) Any policy is acceptable for the VM
cpu-thread-pinning-policy	enum	0..1	CPU thread pinning policy describes how to place the guest CPUs when the host supports hyper threads: • AVOID: Avoids placing a guest on a host with threads. • SEPARATE: Places vCPUs on separate cores, and avoids placing two vCPUs on two threads of same core. • ISOLATE: Places each vCPU on a different core, and places no vCPUs from a different guest on the same core. • PREFER: Attempts to place vCPUs on threads of the same core.

ID	Type	Cardinality	Description
pcie-device	list	0..1	List of pcie passthrough devices. See "vnfd:vdu:guest-epa:pcie-device" below
numa-policy	choice	1	Specifies whether the VM is NUMA aware. When NUMA aware, the numa-node-policy container captures the details about the numa-node-policy. Choices are: • NUMA-UNAWARE • NUMA-AWARE

vnfd:vdu:guest-epa:pcie-device

ID	Type	Cardinality	Description
device-id	string	1	Device identifier
count	uint64	1	Number of devices to attach to the VM.

vnfd:vdu:guest-epa:numa-policy:numa-unaware

ID	Type	Cardinality	Description
numa-unaware	empty	1	Specifies that no NUMA policy is requested.

vnfd:vdu:guest-epa:numa-policy:numa-aware

ID	Type	Cardinality	Description
numa-node-policy	container	1	This policy defines numa topology of the guest. Specifically identifies if the guest should be run on a host with one numa node or multiple numa nodes. As an example a guest may want 8 vcpus and 4 GB of memory. But may want the vcpus and memory distributed across multiple numa nodes. The NUMA node 1 may run with 6 vcpus and 3GB, and NUMA node 2 may run with 2 vcpus and 1GB. See "vnfd:vdu:guest-epa:numa-policy:numa-aware:numa-node-policy" below

vnfd:vdu:guest-epa:numa-policy:numa-aware:numa-node-policy

ID	Type	Cardinality	Description
node-cnt	uint16	1	Number of NUMA nodes to expose to the VM.
mem-policy	enum	1	This policy specifies how the memory should be allocated in a multi-node scenario. • STRICT: The memory must be allocated strictly from the memory attached to the NUMA node. • PREFERRED: The memory should be allocated preferentially from the memory attached to the NUMA node
node	list	0..n	List of NUMA nodes. See "vnfd:vdu:guest-epa:numa-policy:numa-aware:numa-node-policy:node" on page

ID	Type	Cardinality	Description
			42.

vnfd:vdu:guest-epa:numa-policy:numa-aware:numa-node-policy:node

ID	Type	Cardinality	Description
id	uint64	1	NUMA node identification. Typically 0 or 1.
vpcu	list	1	List of VPCUs to allocate on this NUMA node.
memory-mb	uint64	1	Memory size expressed in MB for this NUMA node.
om-numa-type	choice	1	Openmano NUMA type selection: • CORES • PAIRED-THREADS • THREADS See "vnfd:vdu:guest-epa:numa-policy:numa-aware:numa-node-policy:node" above

vnfd:vdu:guest-epa:numa-policy:numa-aware:numa-node-policy:node:om-numa-type:cores

ID	Type	Cardinality	Description
num-cores	uint8	1	Number of cores.

vnfd:vdu:guest-epa:numa-policy:numa-aware:numa-node-policy:node:om-numa-type:paired-threads

ID	Type	Cardinality	Description
num-paired-threads	uint8	1	Number of paired-threads.
paired-thread-ids	list	0..n	List of thread paired to use in case of paired thread NUMA.

vnfd:vdu:guest-epa:numa-policy:numa-aware:numa-node-policy:node:om-numa-type:paired-threads:paired-thread-ids

ID	Type	Cardinality	Description
thread-a	uint8	1	Thread ID
thread-b	uint8	1	Thread ID

vnfd:vdu:guest-epa:numa-policy:numa-aware:numa-node-policy:node:om-numa-type:threads

ID	Type	Cardinality	Description
nym-threads	uint8	1	Number of threads.

VDU Internal Connection Point

vnfd:vdu:internal-connection-point

ID	Type	Cardinality	Description
name	string	1	Name of the connection point.
id	string	1	Unique identifier for the connection point.
short-name	string	1	Short name of the connection point for display in the RIFT.wareUI.
type	enum	1	Type of connection point. ELAN: A multipoint service connecting a set of VNFs
internal-vld-ref	reference	1	path "../../../internal-vld/id"

VDU Internal Interface

vnfd:vdu:internal-interface

ID	Type	Cardinality	Description
name	string	1	Name of the internal interface inside the VDU. **Note:** This name has only local significance to the VDU.
vdu-internal-connection-point-ref	reference	1	Reference to an internal connection point. path "../../internal-connection-point/id"
virtual-interface	container	1	Container for the virtual interface properties. See "vnfd:vdu:internal-interface:virtual-interface" below.

vnfd:vdu:internal-interface:virtual-interface

ID	Type	Cardinality	Description
type	enum	1	Specifies the type of virtual interface between VM and host. • `VIRTIO`: Use the traditional VIRTIO interface • `PCI-PASSTHROUGH`: Use PCI-PASSTHROUGH interface • `SR-IOV`: Use SR-IOV interface • `OM-MGMT`: Used to specify openmano mgmt internal-connection type
vpci	string	1	Specifies the virtual PCI address. Expressed in the following format dddd:dd:dd.d. For example 0000:00:12.0. This information can be used to pass as metadata during the VM creation
bandwidth	integer	1	Specifies the bandwidth requirement for the NIC.

VDU External Interface

vnfd:vdu:external-interface

ID	Type	Cardinality	Description
name	string	1	Name of the external interface inside the VDU. **Note:** This name has only local significance.
vnfd-connection-point-ref	reference	1	Reference to a connection point. path "../../../connection-point/name"
virtual-interface	container	1	Container for the virtual interface properties. See "vnfd:vdu:external-interface:virtual-interface" below.

vnfd:vdu:external-interface:virtual-interface

ID	Type	Cardinality	Description
type	enum	1	Specifies the type of virtual interface between VM and host. • `VIRTIO`: Use the traditional VIRTIO interface • `PCI-PASSTHROUGH`: Use the PCI-PASSTHROUGH interface • `SR-IOV`: Use the SR-IOV interface • `OM-MGMT`: Used to specify openmano mgmt external-connection type
vpci	string	1	Specifies the virtual PCI address. Expressed in the following format dddd:dd:dd.d. For example 0000:00:12.0. This information can be used to pass as metadata during the VM creation
bandwidth	integer	1	Specifies the bandwidth requirement for the NIC.

VDU Dependency

vnfd:vdu-dependency

ID	Type	Cardinality	Description
vdu-source-ref	reference	1	Identifier of the VDU. path "../../vdu/id"
vdu-depends-on-ref	reference	1	Reference to the VDU on which the source VDU depends. path "../../vdu/id"

MANO YANG Models

YANG is a data modeling language used to model configuration and state data manipulated by the Network Configuration (NETCONF) Protocol [RFC 6241], NETCONF remote procedure calls, and NETCONF notifications.

MANO Types

```
module mano-types
{
  namespace "urn:ietf:params:xml:ns:yang:nfvo:mano-types";
  prefix "manotypes";

  import ietf-inet-types {
    prefix "inet";
  }

  import rw-pb-ext {
    prefix "rwpb";
  }

  revision 2015-04-23 {
    description
      "Initial revision. This YANG file defines
       the reusable base types for VNF Management
       and Orchestration (MANO).";
    reference
      "Derived from earlier versions of base YANG files";
  }

  typedef parameter-data-type {
    type enumeration {
      enum string;
      enum integer;
      enum boolean;
    }
  }

  grouping primitive-parameter {
    leaf name {
      description
          "Name of the parameter.";
      type string;
    }

    leaf data-type {
      description
          "Data type associated with the name.";
      type manotypes:parameter-data-type;
    }

    leaf mandatory {
      description "Is this field mandatory";
      type boolean;
      default false;
    }
```

```
  leaf default-value {
    description "The default value for this field";
    type string;
  }

  leaf parameter-pool {
    description "NSD Parameter pool name to use for this paramter";
    type string;
  }

  leaf read-only {
    description
      "The value should be greyed out by the UI.
      Only applies to parameters with default values.";
    type boolean;
  }

  leaf hidden {
    description
      "The value should be hidden by the UI.
      Only applies to parameters with default values.";
    type boolean;
  }
}


grouping vnf-configuration {
  container vnf-configuration {
    rwpb:msg-new VnfConfiguration;
    description
        "Information regarding the VNF configuration
         is captured here. Note that if the NS contains
         multiple instances of the same VNF, each instance
         of the VNF may have different configuration";

    choice config-method {
      description
          "Defines the configuration method for the VNF.";
      case netconf {
        description
            "Use NETCONF for configuring the VNF.";
        container netconf {
          leaf target {
            description
                "Netconf configuration target";
            type enumeration {
              enum running;
              enum candidate;
            }
          }

          leaf protocol {
            description
                "Protocol to use for netconf (e.g. ssh)";
            type enumeration {
              enum None;
```

```
              enum ssh;
            }
          }

          leaf port {
            description
                "Port for the netconf server.";
            type inet:port-number;
          }
        }
      }

      case rest {
        description
            "Use REST for configuring the VNF.";
        container rest {
          leaf port {
            description
                "Port for the REST server.";
            type inet:port-number;
          }
        }
      }

      case script {
        description
            "Use custom script for configuring the VNF.
             This script is executed in the context of
             Orchestrator.";
        container script {
          leaf script-type {
            description
                "Script type - currently supported : bash, expect";
            type enumeration {
              enum bash;
              enum expect;
            }
          }
        }
      }

      case juju {
        description
          "Configure the VNF through Juju.";
        container juju {
          leaf charm {
            description "Juju charm to use with the VNF.";
            type string;
          }
        }
      }
    }

    container config-access {
      leaf mgmt-ip-address {
        description
            "IP address to be used to configure this VNF,
```

```
            optional if it is possible to resolve dynamically.";
        type inet:ip-address;
      }

      leaf username {
        description
            "username for configuration.";
        type string;
      }

      leaf password {
        description
            "Password for configuration access authentication.";
        type string;
      }
    }

    container config-attributes {
      description
          "Miscelaneous input parameters to be considered
           while processing the NSD to apply configuration";

      leaf config-priority {
        description
            "Configuration priority - order of confgiration
             to be applied to each VNF in this NS,
             low number gets precedence over high number";
        type uint64;
      }

      leaf config-delay {
        description
            "Wait (seconds) before applying the configuration to VNF";
        type uint64;
      }
    }

    list config-primitive {
      rwpb:msg-new ConfigPrimitive;
      description
          "List of configuration primitives supported by the
          configuration agent for this VNF.";
      key "name";

      leaf name {
        description
            "Name of the configuration primitive.";
        type string;
      }

      list parameter {
        description
            "List of parameters to the configuration primitive.";
        key "name";
        uses primitive-parameter;
      }
    }
```

```
    list initial-config-primitive {
      rwpb:msg-new InitialConfigPrimitive;
      description
          "Initial set of configuration primitives.";
      key "seq";
      leaf seq {
        description
            "Sequence number for the configuration primitive.";
        type uint64;
      }

      leaf name {
        description
            "Name of the configuration primitive.";
        type string;
      }

      list parameter {
        key "name";
        leaf name {
          type string;
        }

        leaf value {
          type string;
        }
      }
    }

    leaf config-template {
      description
          "Configuration template for each VNF";
      type string;
    }
  }
} // END - grouping vnf-configuration

typedef virtual-link-type {
  description
      "Type of virtual link
       ELAN: A multipoint service connecting a set of VNFs
       // ELINE: For a simple point to point connection
       //        between a VNF and the existing network.
       // ETREE: A multipoint service connecting one or
       //        more roots and a set of leaves, but
       //        preventing inter-leaf communication.";
  type enumeration {
    enum ELAN;
    // enum ETREE;
    // enum ELINE;
  }
}

grouping named-value {
  leaf name {
    type string;
```

```
    }

    leaf value {
      type string;
    }
  }

  typedef http-method {
    description
      "Type of HTTP operation";

    type enumeration {
      enum POST;
      enum PUT;
      enum GET;
      enum DELETE;
      enum OPTIONS;
      enum PATCH;
    }
  }

  typedef api-type {
    description
      "Type of API to fetch monitoring params";

    type enumeration {
      enum HTTP;
      enum NETCONF;
      enum SOAP;
    }
  }

  typedef json-query-method {
    description
      "The method to extract a value from a JSON response

       NAMEKEY - Use the name as the key for a non-nested value.
       JSONPATH - Use jsonpath-rw implemenation to extract a value.
       OBJECTPATH - Use objectpath implemenation to extract a value.";
      type enumeration {
        enum NAMEKEY;
        enum JSONPATH;
        enum OBJECTPATH;
      }
  }

  typedef param-value-type {
    description
      "The type of the parameter value";
    type enumeration {
       enum INT;
       enum DECIMAL;
       enum STRING;
    }
  }

  typedef connection-point-type {
```

```
    description
        "Type of connection point
        VPORT: Virtual Port
        // VNIC_ADDR: Virtual NIC Address
        // PNIC_ADDR: Physical NIC Address
        // PPORT: Phsical Port.";

    type enumeration {
      enum VPORT;
    }
  }

  typedef widget-type {
    description
        "Type of the widget, typically used by the UI.";
    type enumeration {
      enum HISTOGRAM;
      enum BAR;
      enum GAUGE;
      enum SLIDER;
      enum COUNTER;
      enum TEXTBOX;
    }
  }

  typedef cpu-feature-type {
    description
        "Enumeration for CPU features.

         AES: CPU supports advanced instruction set for
         AES (Advanced Encryption Standard).

         CAT: Cache Allocation Technology (CAT) allows
         an Operating System, Hypervisor, or similar
         system management agent to specify the amount
         of L3 cache (currently the last-level cache
         in most server and client platforms) space an
         application can fill (as a hint to hardware
         functionality, certain features such as power
         management may override CAT settings).

         CMT: Cache Monitoring Technology (CMT) allows
         an Operating System, Hypervisor, or similar
         system management agent to determine the
         usage of cache based on applications running
         on the platform. The implementation is
         directed at L3 cache monitoring (currently
         the last-level cache in most server and
         client platforms).

         DDIO: Intel Data Direct I/O (DDIO) enables
         Ethernet server NICs and controllers talk
         directly to the processor cache without a
         detour via system memory. This enumeration
         specifies if the VM requires a DDIO
         capable host.";
```

```
    type enumeration {
      enum PREFER_AES;
      enum REQUIRE_AES;
      enum PREFER_CAT;
      enum REQUIRE_CAT;
      enum PREFER_CMT;
      enum REQUIRE_CMT;
      enum PREFER_DDIO;
      enum REQUIRE_DDIO;
    }
  }

  grouping vm-flavor {
    container vm-flavor {
      leaf vcpu-count {
        description
            "Number of vcpus for the VM.";
        type uint16;
      }

      leaf memory-mb {
        description
            "Amount of memory in MB.";
        type uint64;
      }

      leaf storage-gb {
        description
            "Amount of disk space in GB.";
        type uint64;
      }
    }
  } //grouping vm-flavor

  grouping vswitch-epa {
    container vswitch-epa {
      leaf ovs-acceleration {
        description
            "Specifies Open vSwitch acceleration mode.
             MANDATORY: OVS acceleration is required
             PREFERRED: OVS acceleration is preferred";
        type enumeration {
          enum MANDATORY;
          enum PREFERRED;
          enum DISABLED;
        }
      }

      leaf ovs-offload {
        description
            "Specifies Open vSwitch hardware offload mode.
             MANDATORY: OVS offload is required
             PREFERRED: OVS offload is preferred";
        type enumeration {
          enum MANDATORY;
          enum PREFERRED;
          enum DISABLED;
```

```
      }
    }
  }
}

grouping hypervisor-epa {
  container hypervisor-epa {
    leaf type {
      description
          "Specifies the type of hypervisor.
           KVM: KVM
           XEN: XEN";
      type enumeration {
        enum PREFER_KVM;
        enum REQUIRE_KVM;
      }
    }
    leaf version {
      type string;
    }
  }
}

grouping host-epa {
  container host-epa {
    description "Specifies the host level EPA attributes.";
    leaf cpu-model {
      description
          "Host CPU model. Examples include: SandyBridge,
           IvyBridge";
      type enumeration {
        enum PREFER_WESTMERE;
        enum REQUIRE_WESTMERE;
        enum PREFER_SANDYBRIDGE;
        enum REQUIRE_SANDYBRIDGE;
        enum PREFER_IVYBRIDGE;
        enum REQUIRE_IVYBRIDGE;
        enum PREFER_HASWELL;
        enum REQUIRE_HASWELL;
        enum PREFER_BROADWELL;
        enum REQUIRE_BROADWELL;
        enum PREFER_NEHALEM;
        enum REQUIRE_NEHALEM;
        enum PREFER_PENRYN;
        enum REQUIRE_PENRYN;
        enum PREFER_CONROE;
        enum REQUIRE_CONROE;
        enum PREFER_CORE2DUO;
        enum REQUIRE_CORE2DUO;
      }
    }

    leaf cpu-arch {
      description "Host CPU architecture.";
      type enumeration {
        enum PREFER_X86;
        enum REQUIRE_X86;
```

```
          enum PREFER_X86_64;
          enum REQUIRE_X86_64;
          enum PREFER_I686;
          enum REQUIRE_I686;
          enum PREFER_IA64;
          enum REQUIRE_IA64;
          enum PREFER_ARMV7;
          enum REQUIRE_ARMV7;
          enum PREFER_ARMV8;
          enum REQUIRE_ARMV8;
        }
      }

      leaf cpu-vendor {
        description "Host CPU Vendor.";
        type enumeration {
          enum PREFER_INTEL;
          enum REQUIRE_INTEL;
          enum PREFER_AMD;
          enum REQUIRE_AMD;
        }
      }

      leaf cpu-socket-count {
        description "Number of sockets on the host.";
        type enumeration {
          enum PREFER_ONE;
          enum PREFER_TWO;
          enum REQUIRE_ONE;
          enum REQUIRE_TWO;
        }
      }

      leaf cpu-core-count {
        description "Number of cores on the host.";
        type uint64;
      }

      leaf-list cpu-feature {
        description
            "List of CPU features.";
        type cpu-feature-type;
      }

      leaf om-cpu-model-string {
        description "Openmano CPU model string";
        type string;
      }

      leaf-list om-cpu-feature {
        description "Openmano CPU features";
        type string;
      }
    }
  }

  grouping guest-epa {
```

```
    description "EPA attributes for the guest";
    container guest-epa {
      leaf trusted-execution {
        description "This VM should be allocated from trusted pool";
        type boolean;
      }

      leaf mempage-size {
        description
            "Memory page allocation size. If a VM requires
             hugepages, it should choose LARGE or SIZE_2MB
             or SIZE_1GB. If the VM prefers hugepages it
             should chose PREFER_LARGE.
             LARGE        : Require hugepages (either 2MB or 1GB)
             SMALL        : Doesn't require hugepages
             SIZE_2MB     : Requires 2MB hugepages
             SIZE_1GB     : Requires 1GB hugepages
             PREFER_LARGE : Application perfers hugepages";
        type enumeration {
          enum LARGE;
          enum SMALL;
          enum SIZE_2MB;
          enum SIZE_1GB;
          enum PREFER_LARGE;
        }
      }

      leaf cpu-pinning-policy {
        description
            "CPU pinning policy describes association
             between virtual CPUs in guest and the
             physical CPUs in the host.
             DEDICATED : Virtual CPUs are pinned to
                         physical CPUs
             SHARED    : Multiple VMs may share the
                         same physical CPUs.
             ANY       : Any policy is acceptable for the VM";
        type enumeration {
          enum DEDICATED;
          enum SHARED;
          enum ANY;
        }
        default "ANY";
      }

      leaf cpu-thread-pinning-policy {
          description
            "CPU thread pinning policy describes how to
             place the guest CPUs when the host supports
             hyper threads:
             AVOID   : Avoids placing a guest on a host
                       with threads.
             SEPARATE: Places vCPUs on separate cores,
                       and avoids placing two vCPUs on
                       two threads of same core.
             ISOLATE : Places each vCPU on a different core,
                       and places no vCPUs from a different
```

```
                    guest on the same core.
          PREFER  : Attempts to place vCPUs on threads
                    of the same core.";
      type enumeration {
        enum AVOID;
        enum SEPARATE;
        enum ISOLATE;
        enum PREFER;
      }
    }

    list pcie-device {
      description
          "List of pcie passthrough devices.";
      key device-id;
      leaf device-id {
        description
            "Device identifier.";
        type string;
      }
      leaf count {
        description
            "Number of devices to attach to the VM.";
        type uint64;
      }
    }

    choice numa-policy {
      case numa-unware {
        leaf numa-unware {
          type empty;
        }
      }

      case numa-aware {
        container numa-node-policy {
          description
              "This policy defines numa topology of the
               guest. Specifically identifies if the guest
               should be run on a host with one numa
               node or multiple numa nodes. As an example
               a guest may want 8 vcpus and 4 GB of
               memory. But may want the vcpus and memory
               distributed across multiple numa nodes.
               The NUMA node 1 may run with 6 vcpus and
               3GB, and NUMA node 2 may run with 2 vcpus
               and 1GB.";

          leaf node-cnt {
            description
                "The number of numa nodes to expose to the VM.";
            type uint16;
          }

          leaf mem-policy {
            description
                "This policy specifies how the memory should
```

```
                    be allocated in a multi-node scenario.
                    STRICT    : The memory must be allocated
                                strictly from the memory attached
                                to the NUMA node.
                    PREFERRED : The memory should be allocated
                                perferentially from the memory
                                attached to the NUMA node";
                type enumeration {
                  enum STRICT;
                  enum PREFERRED;
                }
              }

            list node {
               key id;
               leaf id {
                 description
                     "NUMA node identification. Typically
                      it's 0 or 1";
                 type uint64;
               }

               leaf-list vcpu {
                 description
                     "List of vcpus to allocate on
                      this numa node.";
                 type uint64;
               }

               leaf memory-mb {
                 description
                     "Memory size expressed in MB
                      for this NUMA node.";
                 type uint64;
               }

               choice om-numa-type {
                 description
                     "Openmano Numa type selection";

                 case cores {
                   leaf num-cores {
                     type uint8;
                   }
                 }

                 case paired-threads {
                   container paired-threads {
                     leaf num-paired-threads {
                       type uint8;
                     }

                     list paired-thread-ids {
                       description
                           "List of thread pairs to use in case of paired-
thread numa";
                       max-elements 16;
```

```
                    key thread-a;

                    leaf thread-a {
                        type uint8;
                    }

                    leaf thread-b {
                        type uint8;
                    }
                  }
                }
              }
              case threads {
                leaf num-threads {
                  type uint8;
                }
              }
            }
          }

        }
      }
    }
  }
}

grouping provider-network {
  container provider-network {
    description "Container for the provider network.";
    leaf physical-network {
      description
          "Name of the phsyical network on which the provider
           network is built.";
      type string;
    }

    leaf overlay-type {
      description
          "Type of the overlay network.";
      type enumeration {
        enum LOCAL;
        enum FLAT;
        enum VLAN;
        enum VXLAN;
        enum GRE;
      }
    }
    leaf segmentation_id {
      description
          "Segmentation ID";
          type uint32;
    }
  }
}

grouping monitoring-param {
  list http-endpoint {
```

```
      description
          "List of http endpoints to be used by monitoring params";
      key path;

      leaf path {
        description "The HTTP path on the management server";
        type string;
      }

      leaf https {
        description "Pick HTTPS instead of HTTP , Default is false";
        type boolean;
        default "false";
      }

      leaf port {
        description "The HTTP port to connect to";
        type inet:port-number;
      }

      leaf username {
        description "The HTTP basic auth username";
        type string;
      }

      leaf password {
        description "The HTTP basic auth password";
        type string;
      }

      leaf polling_interval_secs {
        description "The HTTP polling interval in seconds";
        type uint8;
        default 2;
      }

      leaf method {
        description
          "This is the method to be performed at the uri.
           GET by default for action";

        type manotypes:http-method;
        default "GET";
      }

      list headers {
        description "Custom HTTP headers to put on HTTP request";
        key key;
        leaf key{
          description "HTTP header key";
          type string;
        }

        leaf value{
          description "HTTP header value";
          type string;
        }
```

```
      }
    }

    list monitoring-param {
      description
          "List of monitoring parameters at the NS level";
      key id;
      leaf id {
        type string;
      }

      leaf name {
        type string;
      }

      leaf http-endpoint-ref {
        type leafref {
          path "../../http-endpoint/path";
        }
      }

      leaf json-query-method {
        type json-query-method;
        default "NAMEKEY";
      }

      container json-query-params {
        leaf json-path {
          description
            "The jsonpath to use to extract value from JSON structure";
          type string;
        }
        leaf object-path {
          description
            "The objectpath to use to extract value from JSON structure";
          type string;
        }
      }

      leaf description {
        type string;
      }

      leaf group-tag {
        description "A simple tag to group monitoring parameters";
        type string;
      }

      leaf value-type {
        type param-value-type;
        default "INT";
      }

      container numeric-constraints {
        leaf min-value {
          description
              "Minimum value for the parameter";
```

```
          type uint64;
        }
        leaf max-value {
          description
              "Maxium value for the parameter";
          type uint64;
        }
      }

      container text-constraints {
        leaf min-length {
          description
              "Minimum string length for the parameter";
          type uint8;
        }
        leaf max-length {
          description
              "Maximum string length for the parameter";
          type uint8;
        }
      }

      leaf value-integer {
        description
            "Current value for an integer parameter";
        type int64;
      }

      leaf value-decimal {
        description
            "Current value for a decimal parameter";
        type decimal64 {
          fraction-digits 4;
        }
      }

      leaf value-string {
        description
            "Current value for a string parameter";
        type string;
      }

      leaf widget-type {
        type manotypes:widget-type;
      }

      leaf units {
        type string;
      }
    }
  }

  grouping control-param {
    list control-param {
      description
          "List of control parameters to manage and
           update the running configuration of the VNF";
```

```
      key id;

      leaf id {
        type string;
      }

      leaf name {
        type string;
      }

      leaf description {
        type string;
      }

      leaf group-tag {
        description "A simple tag to group control parameters";
        type string;
      }

      leaf min-value {
        description
            "Minimum value for the parameter";
        type uint64;
      }

      leaf max-value {
        description
            "Maxium value for the parameter";
        type uint64;
      }

      leaf current-value {
        description
            "Current value for the parameter";
        type uint64;
      }

      leaf step-value {
        description
            "Step value for the parameter";
        type uint64;
      }

      leaf units {
        type string;
      }

      leaf widget-type {
        type manotypes:widget-type;
      }

      leaf url {
        description
          "This is the URL where to perform the operation";

        type inet:uri;
      }
```

```
    leaf method {
      description
        "This is the method to be performed at the uri.
         POST by default for action";

      type manotypes:http-method;
      default "POST";
    }

    leaf payload {
      description
        "This is the operation payload or payload template as stringified
         JSON. This field provides the data  to be sent for this operation
         call";

      type string;
    }
  }
}

grouping action-param {
  list action-param {
    description
        "List of action parameters to
         control VNF";
    key id;
    leaf id {
      type string;
    }

    leaf name {
      type string;
    }

    leaf description {
      type string;
    }

    leaf group-tag {
      description "A simple tag to group monitoring parameter";
      type string;
    }

    leaf url {
      description
        "This is the URL where to perform the operation";
      type inet:uri;
    }

    leaf method {
      description
        "This is the method to be performed at the uri.
         POST by default for action";

      type manotypes:http-method;
      default "POST";
```

```
      }

      leaf payload {
        description
          "This is the operation payload or payload template to be sent in
           the data for this operation call";

        type string;
      }
    }
  }

  grouping input-parameter {
    description "";

    list input-parameter {
      description
          "List of input parameters";

      key xpath;


      leaf xpath {
        description
          "A an xpath that specfies which element in a descriptor is to be
          modified.";
        type string;
      }

      leaf value {
        description
          "The value that the element specified by the xpath should take
when a
          record is created.";
        type string;
      }
    }
  }

  grouping input-parameter-xpath {
    list input-parameter-xpath {
      description
          "List of xpaths to parameters inside the NSD
           the can be customized during the instantiation.";

      key "xpath";
      leaf xpath {
        description
            "An xpath that specifies the element in a descriptor.";
        type string;
      }

      leaf label {
        description "A descriptive string";
        type string;
      }
```

```
      leaf default-value {
        description " A default value for this input parameter";
        type string;
      }
    }
  }

  grouping nfvi-metrics {
    container vcpu {
      leaf label {
        description
          "Label to show in UI";
        type string;
        default "VCPU";
      }

      leaf total {
        description
          "The total number of VCPUs available.";
        type uint64;
      }

      leaf utilization {
        description
          "The VCPU utilization (percentage).";
        type decimal64 {
          fraction-digits 2;
          range "0 .. 100";
        }
      }
    }

    container memory {
      leaf label {
        description
          "Label to show in UI";
        type string;
        default "MEMORY";
      }

      leaf used {
        description
          "The amount of memory (bytes) currently in use.";
        type uint64;
      }

      leaf total {
        description
          "The amount of memory (bytes) available.";
        type uint64;
      }

      leaf utilization {
        description
          "The memory utilization (percentage).";
        type decimal64 {
          fraction-digits 2;
```

```
        range "0 .. 100";
      }
    }
  }

  container storage {
    leaf label {
      description
        "Label to show in UI";
      type string;
      default "STORAGE";
    }

    leaf used {
      description
        "The amount of storage (bytes) currently in use.";
      type uint64;
    }

    leaf total {
      description
        "The amount of storage (bytes) available.";
      type uint64;
    }

    leaf utilization {
      description
        "The storage utilization (percentage).";
      type decimal64 {
        fraction-digits 2;
        range "0 .. 100";
      }
    }
  }

  container external-ports {
    leaf label {
      description
        "Label to show in UI";
      type string;
      default "EXTERNAL PORTS";
    }

    leaf total {
      description
        "The total number of external ports.";
      type uint64;
    }
  }

  container internal-ports {
    leaf label {
      description
        "Label to show in UI";
      type string;
      default "INTERNAL PORTS";
    }
```

```
      leaf total {
        description
          "The total number of internal ports.";
        type uint64;
      }
    }

    container network {
      leaf label {
        description
          "Label to show in UI";
        type string;
        default "NETWORK TRAFFIC";
      }

      container incoming {
        leaf label {
          description
            "Label to show in UI";
          type string;
          default "INCOMING NETWORK TRAFFIC";
        }

        leaf bytes {
          description
            "The cumulative number of incoming bytes.";
          type uint64;
        }

        leaf packets {
          description
            "The cumulative number of incoming packets.";
          type uint64;
        }

        leaf byte-rate {
          description
            "The current incoming byte-rate (bytes per second).";
          type decimal64 {
            fraction-digits 2;
          }
        }

        leaf packet-rate {
          description
            "The current incoming packet (packets per second).";
          type decimal64 {
            fraction-digits 2;
          }
        }
      }

      container outgoing {
        leaf label {
          description
            "Label to show in UI";
```

```
          type string;
          default "OUTGOING NETWORK TRAFFIC";
        }

        leaf bytes {
          description
            "The cumulative number of outgoing bytes.";
          type uint64;
        }

        leaf packets {
          description
            "The cumulative number of outgoing packets.";
          type uint64;
        }

        leaf byte-rate {
          description
            "The current outgoing byte-rate (bytes per second).";
          type decimal64 {
            fraction-digits 2;
          }
        }

        leaf packet-rate {
          description
            "The current outgoing packet (packets per second).";
          type decimal64 {
            fraction-digits 2;
          }
        }
      }
    }
  }

  typedef alarm-severity-type {
    description "An indication of the importance or ugency of the alarm";
    type enumeration {
      enum LOW;
      enum MODERATE;
      enum CRITICAL;
    }
  }

  typedef alarm-metric-type {
    description "The type of metrics to register the alarm for";
    type enumeration {
      enum CPU_UTILIZATION;
      enum MEMORY_UTILIZATION;
      enum STORAGE_UTILIZATION;
    }
  }

  typedef alarm-statistic-type {
    description
        "The type of statistic to used to measure a metric to determine
        threshold crossing for an alarm.";
```

```
    type enumeration {
      enum AVERAGE;
      enum MINIMUM;
      enum MAXIMUM;
      enum COUNT;
      enum SUM;
    }
  }

  typedef alarm-operation-type {
    description
        "The relational operator used to define whether an alarm should be
        triggered when, say, the metric statistic goes above or below a
        specified value.";
    type enumeration {
      enum GE; // greater than or equal
      enum LE; // less than or equal
      enum GT; // greater than
      enum LT; // less than
      enum EQ; // equal
    }
  }

  grouping alarm {
    leaf alarm-id {
      description
          "This field is reserved for the identifier assigned by the cloud
          provider";

      type string;
    }

    leaf name {
      description "A human readable string to identify the alarm";
      type string;
    }

    leaf description {
      description "A string containing a description of this alarm";
      type string;
    }

    container actions {
      list ok {
        key "url";
        leaf url {
          type string;
        }
      }

      list insufficient-data {
        key "url";
        leaf url {
          type string;
        }
      }
```

```
      list alarm {
        key "url";
        leaf url {
          type string;
        }
      }
    }

    leaf repeat {
      description
          "This flag indicates whether the alarm should be repeatedly
emitted
          while the associated threshold has been crossed.";

      type boolean;
      default true;
    }

    leaf enabled {
      description
          "This flag indicates whether the alarm has been enabled or
          disabled.";

      type boolean;
      default true;
    }

    leaf severity {
      description "A measure of the important or urgency of the alarm";
      type alarm-severity-type;
    }

    leaf metric {
      description "The metric to be tracked by this alarm.";
      type alarm-metric-type;
    }

    leaf statistic {
      description "The type of metric statistic that is tracked by this
alarm";
      type alarm-statistic-type;
    }

    leaf operation {
      description
          "The relational operator that defines whether the alarm should be
          triggered when the metric statistic is, say, above or below the
          specified threshold value.";
      type alarm-operation-type;
    }

    leaf value {
      description
          "This value defines the threshold that, if crossed, will trigger
          the alarm.";
      type decimal64 {
        fraction-digits 4;
```

```
      }
    }

    leaf period {
      description
          "The period defines the length of time (seconds) that the metric
          data are collected over in oreder to evaluate the chosen
          statistic.";
      type uint32;
    }

    leaf evaluations {
      description
          "This is the number of samples of the metric statistic used to
          evaluate threshold crossing. Each sample or evaluation is equal to
          the metric statistic obtained for a given period. This can be used
          to mitigate spikes in the metric that may skew the statistic of
          interest.";
      type uint32;
    }
  }

  grouping placement-group-info {
    description "";

    leaf name {
      description
          "Place group construct to define the compute resource placement
strategy
           in cloud environment";
      type string;
    }

    leaf requirement {
      description "This is free text space used to describe the
intent/rationale
                   behind this placement group. This is for human
consumption only";
      type string;
    }

    leaf strategy {
      description
          "Strategy associated with this placement group
             Following values are possible
               - COLOCATION: Colocation strategy imply intent to share the
physical
                             infrastructure (hypervisor/network) among all
members
                             of this group.
               - ISOLATION: Isolation strategy imply intent to not share the
physical
                            infrastructure (hypervisor/network) among the
members
                            of this group.
          ";
      type enumeration {
```

```
        enum COLOCATION;
        enum ISOLATION;
      }
      default "COLOCATION";
    }
  }
}
```

NSD YANG Model

```
module nsd
{
  namespace "urn:ietf:params:xml:ns:yang:nfvo:nsd";
  prefix "nsd";

  import rw-pb-ext {
    prefix "rwpb";
  }

  import vld {
    prefix "vld";
  }

  import vnfd {
    prefix "vnfd";
  }

  import ietf-inet-types {
    prefix "inet";
  }

  import ietf-yang-types {
    prefix "yang";
  }

  import mano-types {
    prefix "manotypes";
  }

  revision 2014-10-27 {
    description
      "Initial revision. This YANG file defines
       the Network Service Descriptor (NSD)";
    reference
      "Derived from earlier versions of base YANG files";
  }

  typedef scaling-trigger {
    type enumeration {
      enum pre-scale-in {
        value 1;
      }
      enum post-scale-in {
        value 2;
      }
```

```
      enum pre-scale-out {
        value 3;
      }
      enum post-scale-out {
        value 4;
      }
    }
  }

  container nsd-catalog {

    list nsd {
      key "id";

      leaf id {
        description "Identifier for the NSD.";
        type yang:uuid;
      }

      leaf name {
        description "NSD name.";
        mandatory true;
        type string;
      }

      leaf short-name {
        description "NSD short name.";
        type string;
      }


      leaf vendor {
        description "Vendor of the NSD.";
        type string;
      }

      leaf logo {
        description
          "Vendor logo for the Network Service";
        type string;
      }

      leaf description {
        description "Description of the NSD.";
        type string;
      }

      leaf version {
        description "Version of the NSD";
        type string;
      }

      list connection-point {
        description
            "List for external connection points.
            Each NS has one or more external connection
            points. As the name implies that external
```

```
            connection points are used for connecting
            the NS to other NS or to external networks.
            Each NS exposes these connection points to
            the orchestrator. The orchestrator can
            construct network service chains by
            connecting the connection points between
            different NS.";

          key "name";
          leaf name {
            description
                "Name of the NS connection point.";
            type string;
          }

          leaf type {
            description
                "Type of the connection point.";
            type manotypes:connection-point-type;
          }
        }

        leaf-list vld-ref {
          type leafref {
            path "/vld:vld-catalog/vld:vld/vld:id";
          }
        }

        /* Still having issues modelling this,
           see the comments under vnfd-connection-point-ref
         */
        list vld {
          description
              "List of Virtual Link Descriptors.";

          key "id";

          leaf id {
            description
                "Identifier for the VLD.";
            type string;
          }

          leaf name {
            description
                "Virtual Link Descriptor (VLD) name.";
            type string;
          }

          leaf short-name {
            description
                "Short name for VLD for UI";
            type string;
          }

          leaf vendor {
            description "Provider of the VLD.";
```

```
          type string;
        }

        leaf description {
          description "Description of the VLD.";
          type string;
        }

        leaf version {
          description "Version of the VLD";
          type string;
        }

        leaf type {
          type manotypes:virtual-link-type;
        }

        leaf root-bandwidth {
          description
              "For ELAN this is the aggregate bandwidth.";
          type uint64;
        }

        leaf leaf-bandwidth {
          description
              "For ELAN this is the bandwidth of branches.";
          type uint64;
        }

        list vnfd-connection-point-ref {
          description
              "A list of references to connection points.";
          key "member-vnf-index-ref";

          leaf member-vnf-index-ref {
            description "Reference to member-vnf within constituent-vnfds";
            type leafref {
              path "../../../nsd:constituent-vnfd/nsd:member-vnf-index";
            }
          }

           leaf vnfd-id-ref {
             description
                 "A reference to a vnfd. This is a
                  leafref to path:
                      ../../../nsd:constituent-vnfd
                      + [nsd:id = current()/../nsd:id-ref]
                      + /nsd:vnfd-id-ref
                  NOTE: An issue with confd is preventing the
                  use of xpath. Seems to be an issue with leafref
                  to leafref, whose target is in a different module.
                  Once that is resovled this will switched to use
                  leafref";
             type yang:uuid;
           }

           leaf vnfd-connection-point-ref {
```

```
            description
                "A reference to a connection point name
                 in a vnfd. This is a leafref to path:
                     /vnfd:vnfd-catalog/vnfd:vnfd
                     + [vnfd:id = current()/../nsd:vnfd-id-ref]
                     + /vnfd:connection-point/vnfd:name
                 NOTE: An issue with confd is preventing the
                 use of xpath. Seems to be an issue with leafref
                 to leafref, whose target is in a different module.
                 Once that is resovled this will switched to use
                 leafref";
             type string;
          }
        }

        // replicate for pnfd container here
        uses manotypes:provider-network;
      }

      list constituent-vnfd {
        description
            "List of VNFDs that are part of this
            network service.";

        key "member-vnf-index";

        leaf member-vnf-index {
          description
            "Identifier/index for the VNFD. This separate id
             is required to ensure that multiple VNFs can be
             part of single NS";
          type uint64;
        }

        leaf vnfd-id-ref {
          description
            "Identifier for the VNFD.";
          type leafref {
            path "/vnfd:vnfd-catalog/vnfd:vnfd/vnfd:id";
          }
        }

        leaf start-by-default {
          description
            "VNFD is started as part of the NS instantiation";
          type boolean;
          default true;
        }

        // Provide this VNF configuration parameters
        uses manotypes:vnf-configuration;
      }

      list scaling-group-descriptor {
        description
            "scaling group descriptor within this network service.
             The scaling group defines a group of VNFs,
```

```
            and the ratio of VNFs in the network service
            that is used as target for scaling action";

        key "name";

        leaf name {
          description "Name of this scaling group.";
          type string;
        }

        list vnfd-member {
          description "List of VNFs in this scaling group";
          key "member-vnf-index-ref";

          leaf member-vnf-index-ref {
            description "member VNF index of this member VNF";
            type leafref {
              path "../../../constituent-vnfd/member-vnf-index";
            }
          }

          leaf count {
            description
              "count of this member VNF  within this scaling group.
               The count allows to define  the number of instances
               when a scaling action targets this scaling group";
            type uint32;
            default 1;
          }
        }

        leaf min-instance-count {
          description
            "Minimum instances of the scaling group which are allowed.
            These instances are created by default when the network service
            is instantiated.";
          type uint32;
          default 0;
        }

        leaf max-instance-count {
          description
            "Maximum instances of this scaling group that are allowed
             in a single network service. The network service scaling
             will fail, when the number of service group instances
             exceed the max-instance-count specified.";
          type uint32;
          default 10;
        }

        list scaling-config-action {
          description "List of scaling config actions";
          key "trigger";

          leaf trigger {
            description "scaling trigger";
            type scaling-trigger;
```

```
      }

      leaf ns-config-primitive-name-ref {
        description "Reference to the NS config name primitive";
        type leafref {
          path "../../../config-primitive/name";
        }
      }
    }
  }

  list placement-groups {
    description "List of placement groups at NS level";

    key "name";
    uses manotypes:placement-group-info;

    list constituent-vnfd {
      description
          "List of VNFDs that are part of thisplacement group";

      key "member-vnf-index-ref";

      leaf member-vnf-index-ref {
        description "member VNF index of this member VNF";
        type leafref {
          path "../../../constituent-vnfd/member-vnf-index";
        }
      }

      leaf vnfd-id-ref {
        description
            "Identifier for the VNFD.";
        type leafref {
          path "/vnfd:vnfd-catalog/vnfd:vnfd/vnfd:id";
        }
      }
    }
  }


  list vnf-dependency {
    description
        "List of VNF dependencies.";
    key vnf-source-ref;
    leaf vnf-source-ref {
      type leafref {
        path "/vnfd:vnfd-catalog/vnfd:vnfd/vnfd:id";
      }
    }
    leaf vnf-depends-on-ref {
      description
          "Reference to VNF that sorce VNF depends.";
      type leafref {
        path "/vnfd:vnfd-catalog/vnfd:vnfd/vnfd:id";
      }
    }
```

```
    }

    list vnffgd {
      description
          "List of VNF Forwarding Graph Descriptors (VNFFGD).";

      key "id";

      leaf id {
        description
            "Identifier for the VNFFGD.";
        type string;
      }

      leaf name {
        description
            "VNFFGD name.";
        type string;
      }

      leaf short-name {
        description
            "Short name for VNFFGD for UI";
        type string;
      }

      leaf vendor {
        description "Provider of the VNFFGD.";
        type string;
      }

      leaf description {
        description "Description of the VNFFGD.";
        type string;
      }

      leaf version {
        description "Version of the VNFFGD";
        type string;
      }

      list rsp {
        description
            "List of Rendered Service Paths (RSP).";

        key "id";

        leaf id {
          description
              "Identifier for the RSP.";
          type string;
        }

        leaf name {
          description
              "RSP name.";
          type string;
```

```
            }

            list vnfd-connection-point-ref {
              description
                    "A list of references to connection points.";
              key "member-vnf-index-ref";

              leaf member-vnf-index-ref {
                description "Reference to member-vnf within constituent-
vnfds";
                type leafref {
                  path "../../../../nsd:constituent-vnfd/nsd:member-vnf-
index";
                }
              }

              leaf order {
                type uint8;
                description
                    "A number that denotes the order of a VNF in a chain";
              }

               leaf vnfd-id-ref {
                 description
                     "A reference to a vnfd. This is a
                      leafref to path:
                          ../../../../nsd:constituent-vnfd
                          + [nsd:id = current()/../nsd:id-ref]
                          + /nsd:vnfd-id-ref
                      NOTE: An issue with confd is preventing the
                      use of xpath. Seems to be an issue with leafref
                      to leafref, whose target is in a different module.
                      Once that is resovled this will switched to use
                      leafref";
                 type yang:uuid;
               }

               leaf vnfd-connection-point-ref {
                 description
                     "A reference to a connection point name
                      in a vnfd. This is a leafref to path:
                          /vnfd:vnfd-catalog/vnfd:vnfd
                          + [vnfd:id = current()/../nsd:vnfd-id-ref]
                          + /vnfd:connection-point/vnfd:name
                      NOTE: An issue with confd is preventing the
                      use of xpath. Seems to be an issue with leafref
                      to leafref, whose target is in a different module.
                      Once that is resovled this will switched to use
                      leafref";
                  type string;
                }
            }
          } //rsp

          list classifier {
            description
                "List of classifier rules.";
```

```
        key "id";

        leaf id {
          description
              "Identifier for the classifier rule.";
          type string;
        }

        leaf name {
          description
              "Name of the classifier.";
          type string;
        }

        leaf rsp-id-ref {
          description
              "A reference to the RSP.";
          type leafref {
            path "../../nsd:rsp/nsd:id";
          }
        }


        leaf member-vnf-index-ref {
          description "Reference to member-vnf within constituent-vnfds";
          type leafref {
            path "../../../nsd:constituent-vnfd/nsd:member-vnf-index";
          }
        }

        leaf vnfd-id-ref {
          description
              "A reference to a vnfd. This is a
                   leafref to path:
                       ../../../nsd:constituent-vnfd
                       + [nsd:id = current()/../nsd:id-ref]
                       + /nsd:vnfd-id-ref
                   NOTE: An issue with confd is preventing the
                   use of xpath. Seems to be an issue with leafref
                   to leafref, whose target is in a different module.
                   Once that is resovled this will switched to use
                   leafref";
          type yang:uuid;
        }

        leaf vnfd-connection-point-ref {
          description
              "A reference to a connection point name
                   in a vnfd. This is a leafref to path:
                       /vnfd:vnfd-catalog/vnfd:vnfd
                       + [vnfd:id = current()/../nsd:vnfd-id-ref]
                       + /vnfd:connection-point/vnfd:name
                   NOTE: An issue with confd is preventing the
                   use of xpath. Seems to be an issue with leafref
                   to leafref, whose target is in a different module.
                   Once that is resovled this will switched to use
```

```
                leafref";
          type string;
        }

        list match-attributes {
          description
              "List of match attributes.";

          key "id";

          leaf id {
            description
                "Identifier for the classifier match attribute rule.";
            type string;
          }

          leaf ip-proto {
            description
                "IP Protocol.";
            type uint8;
          }

          leaf source-ip-address {
            description
                "Source IP address.";
            type inet:ip-address;
          }

          leaf destination-ip-address {
            description
                "Destination IP address.";
            type inet:ip-address;
          }

          leaf source-port {
            description
                "Source port number.";
            type inet:port-number;
          }

          leaf destination-port {
            description
                "Destination port number.";
            type inet:port-number;
          }
          //TODO: Add more match criteria
        } //match-attributes
      } // classifier
    } // vnffgd

    uses manotypes:monitoring-param;
    uses manotypes:input-parameter-xpath;

    list parameter-pool {
      description
        "Pool of parameter values which must be
         pulled from during configuration";
```

```
        key "name";

        leaf name {
          description
              "Name of the configuration value pool";
          type string;
        }

        container range {
          description
              "Create a range of values to populate the pool with";

          leaf start-value {
            description
                "Generated pool values start at this value";
            type uint32;
            mandatory true;
          }

          leaf end-value {
            description
                "Generated pool values stop at this value";
            type uint32;
            mandatory true;
          }
        }
      }

      list config-primitive {
        description
            "Network service level configuration primitives.";

        key "name";

        leaf name {
          description
              "Name of the configuration primitive.";
          type string;
        }

        list parameter {
          description
              "List of parameters to the configuration primitive.";

          key "name";
          uses manotypes:primitive-parameter;
        }

        list parameter-group {
          description
              "Grouping of parameters which are logically grouped in UI";
          key "name";

          leaf name {
            description
                "Name of the parameter group";
            type string;
```

```
      }

      list parameter {
        description
            "List of parameters to the configuration primitive.";
        key "name";
        uses manotypes:primitive-parameter;
      }

      leaf mandatory {
        description "Is this parameter group mandatory";
        type boolean;
        default true;
      }
    }

    list vnf-primitive-group {
      description
          "List of configuration primitives grouped by VNF.";

      key "member-vnf-index-ref";
      leaf member-vnf-index-ref {
        description
            "Reference to member-vnf within constituent-vnfds";
        type uint64;
      }

      leaf vnfd-id-ref {
        description
            "A reference to a vnfd. This is a
             leafref to path:
                 ../../../../nsd:constituent-vnfd
                 + [nsd:id = current()/../nsd:id-ref]
                 + /nsd:vnfd-id-ref
             NOTE: An issue with confd is preventing the
             use of xpath. Seems to be an issue with leafref
             to leafref, whose target is in a different module.
             Once that is resovled this will switched to use
             leafref";

        type string;
      }

      leaf vnfd-name {
        description
            "Name of the VNFD";
        type string;
      }

      list primitive {
        key "index";

        leaf index {
          description "Index of this primitive";
          type uint32;
        }
```

```
            leaf name {
              description "Name of the primitive in the VNF primitive ";
              type string;
            }
          }
        }

        leaf user-defined-script {
          description
              "A user defined script.";
          type string;
        }
      }
    }
  }
 }
```

VNFD YANG Model

```
module vnfd
 {
   namespace "urn:ietf:params:xml:ns:yang:nfvo:vnfd";
   prefix "vnfd";

   import mano-types {
     prefix "manotypes";
   }

   import rw-pb-ext {
     prefix "rwpb";
   }

   import ietf-yang-types {
     prefix "yang";
   }

   import ietf-inet-types {
     prefix "inet";
   }

   revision 2015-09-10 {
     description
       "Initial revision. This YANG file defines
        the Virtual Network Function (VNF)";
     reference
       "Derived from earlier versions of base YANG files";
   }

   grouping common-connection-point {
     leaf name {
       description "Name of the connection point";
       type string;
     }

     leaf id {
```

```
      description "Identifier for the internal connection points";
      type string;
    }

    leaf short-name {
      description "Short name of the connection point";
      type string;
    }

    leaf type {
      description "Type of the connection point.";
      type manotypes:connection-point-type;
    }
  }

  grouping virtual-interface {
    container virtual-interface {
      description
          "Container for the virtual interface properties";

      leaf type {
        description
            "Specifies the type of virtual interface
             between VM and host.
             VIRTIO          : Use the traditional VIRTIO interface.
             PCI-PASSTHROUGH : Use PCI-PASSTHROUGH interface.
             SR-IOV          : Use SR-IOV interface.
             OM-MGMT         : Used to specify openmano mgmt external-
connection type";

        type enumeration {
          enum OM-MGMT;
          enum PCI-PASSTHROUGH;
          enum SR-IOV;
          enum VIRTIO;
        }
        default "VIRTIO";
      }

      leaf vpci {
        description
            "Specifies the virtual PCI address. Expressed in
             the following format dddd:dd:dd.d. For example
             0000:00:12.0. This information can be used to
             pass as metadata during the VM creation.";
        type string;
      }

      leaf bandwidth {
        description
            "Aggregate bandwidth of the NIC.";
        type uint64;
      }
    }
  }

  container vnfd-catalog {
```

```
    description
        "Virtual Network Function Descriptor (VNFD).";

    list vnfd {
      key "id";

      leaf id {
        description "Identifier for the VNFD.";
        type yang:uuid;
      }

      leaf name {
        description "VNFD name.";
        mandatory true;
        type string;
      }

      leaf short-name {
        description "VNFD short name.";
        type string;
      }

      leaf vendor {
        description "Vendor of the VNFD.";
        type string;
      }

      leaf logo {
        description
            "Vendor logo for the Virtual Network Function";
        type string;
      }

      leaf description {
        description "Description of the VNFD.";
        type string;
      }

      leaf version {
        description "Version of the VNFD";
        type string;
      }

      container mgmt-interface {
        description
            "Interface over which the VNF is managed.";

        uses manotypes:vnf-configuration;

        choice endpoint-type {
          description
              "Indicates the type of management endpoint.";

          case ip {
            description
                "Specifies the static IP address for managing the VNF.";
```

```
            leaf ip-address {
              type inet:ip-address;
            }
          }

          case vdu-id {
            description
                "Use the default management interface on this VDU.";
            leaf vdu-id {
              type leafref {
                path "/vnfd:vnfd-catalog/vnfd:vnfd/vnfd:vdu/vnfd:id";
              }
            }
          }

          case cp {
            description
                "Use the ip address associated with this connection point.";
            leaf cp {
              type leafref {
                path "/vnfd:vnfd-catalog/vnfd:vnfd/vnfd:connection-
point/vnfd:name";
              }
            }
          }
        }

        leaf port {
          description
              "Port for the management interface.";
          type inet:port-number;
        }

        container dashboard-params {
          description "Parameters for the VNF dashboard";

          leaf path {
            description "The HTTP path for the dashboard";
            type string;
          }

          leaf https {
            description "Pick HTTPS instead of HTTP , Default is false";
            type boolean;
          }

          leaf port {
            description "The HTTP port for the dashboard";
            type inet:port-number;
          }
        }
      }

      list internal-vld {
        key "id";
        description
            "List of Internal Virtual Link Descriptors (VLD).
```

```
            The internal VLD describes the basic topology of
            the connectivity (e.g. E-LAN, E-Line, E-Tree)
            between internal VNF components of the system.";

          leaf id {
            description "Identifier for the VLD";
            type string;
          }

          leaf name {
            description "Name of the internal VLD";
            type string;
          }

          leaf short-name {
            description "Short name of the internal VLD";
            type string;
          }

          leaf description {
            type string;
          }

          leaf type {
            type manotypes:virtual-link-type;
          }

          leaf root-bandwidth {
            description
                "For ELAN this is the aggregate bandwidth.";
            type uint64;
          }

          leaf leaf-bandwidth {
            description
                "For ELAN this is the bandwidth of branches.";
            type uint64;
          }

          leaf-list internal-connection-point-ref {
            type leafref {
              path "../../vdu/internal-connection-point/id";
            }
          }

          uses manotypes:provider-network;
        }

        list connection-point {
          key "name";
          description
              "List for external connection points. Each VNF has one
              or more external connection points. As the name
              implies that external connection points are used for
              connecting the VNF to other VNFs or to external networks.
              Each VNF exposes these connection points to the
              orchestrator. The orchestrator can construct network
```

```
            services by connecting the connection points between
            different VNFs. The NFVO will use VLDs and VNFFGs at
            the network service level to construct network services.";

        uses common-connection-point;
      }

      list vdu {
        description "List of Virtual Deployment Units";
        key "id";

        leaf id {
          description "Unique id for the VDU";
          type string;
        }

        leaf name {
          description "Unique name for the VDU";
          type string;
        }

        leaf description {
            description "Description of the VDU.";
            type string;
        }

        leaf count {
          description "Number of instances of VDU";
          type uint64;
        }

        leaf mgmt-vpci {
          description
              "Specifies the virtual PCI address. Expressed in
             the following format dddd:dd:dd.d. For example
             0000:00:12.0. This information can be used to
             pass as metadata during the VM creation.";
          type string;
        }


        uses manotypes:vm-flavor;
        uses manotypes:guest-epa;
        uses manotypes:vswitch-epa;
        uses manotypes:hypervisor-epa;
        uses manotypes:host-epa;

        leaf image {
          description
            "Image name for the software image.
             If the image name is found within the VNF packaage it will
             be uploaded to all cloud accounts during onboarding process.
             Otherwise, the image must be added to the cloud account with
             the same name as entered here.
            ";
          mandatory true;
          type string;
```

```
        }

        leaf image-checksum {
          description
            "Image md5sum for the software image.
            The md5sum, if provided, along with the image name uniquely
            identifies an image uploaded to the CAL.
            ";
          type string;
        }

        leaf cloud-init {
          description "Content of cloud-init script";
          type string;
        }

        list internal-connection-point {
          key "id";
          description
              "List for internal connection points. Each VNFC
              has zero or more internal connection points.
              Internal connection points are used for connecting
              the VNF components internal to the VNF. If a VNF
              has only one VNFC, it may not have any internal
              connection points.";

          uses common-connection-point;

          leaf internal-vld-ref {
            type leafref {
              path "../../../internal-vld/id";
            }
          }
        }

        list internal-interface {
          description
              "List of internal interfaces for the VNF";
          key name;

          leaf name {
            description
                "Name of internal interface. Note that this
                name has only local significance to the VDU.";
            type string;
          }

          leaf vdu-internal-connection-point-ref {
            type leafref {
              path "../../internal-connection-point/id";
            }
          }
          uses virtual-interface;
        }

        list external-interface {
          description
```

```
              "List of external interfaces for the VNF.
              The external interfaces enable sending
              traffic to and from VNF.";
            key name;

            leaf name {
              description
                  "Name of the external interface. Note that
                  this name has only local significance.";
              type string;
            }

            leaf vnfd-connection-point-ref {
              description
                "Name of the external connection point.";
              type leafref {
                path "../../../connection-point/name";
              }
            }
            uses virtual-interface;
          }
        }

        list vdu-dependency {
          description
              "List of VDU dependencies.";

          key vdu-source-ref;
          leaf vdu-source-ref {
            type leafref {
              path "../../vdu/id";
            }
          }

          leaf vdu-depends-on-ref {
            description
                "Reference to the VDU that
                source VDU depends.";
            type leafref {
              path "../../vdu/id";
            }
          }
        }

        leaf service-function-chain {
          description "Type of node in Service Function Chaining
Architecture";

          type enumeration {
            enum UNAWARE;
            enum CLASSIFIER;
            enum SF;
            enum SFF;
          }
          default "UNAWARE";
        }
```

```
      leaf service-function-type {
        description
            "Type of Service Function.
             NOTE: This needs to map with Service Function Type in ODL to
             support VNFFG. Service Function Type is manadatory param in ODL
             SFC. This is temporarily set to string for ease of use";
            type string;
      }

      uses manotypes:monitoring-param;

      list placement-groups {
        description "List of placement groups at VNF level";

        key "name";
        uses manotypes:placement-group-info;

        list constituent-vdus {
          description
              "List of VDUs that are part of this placement group";
          key "member-vdu-ref";

          leaf member-vdu-ref {
            type leafref {
              path "../../../vdu/id";
            }
          }
        }
      }
    }
  }
}
```