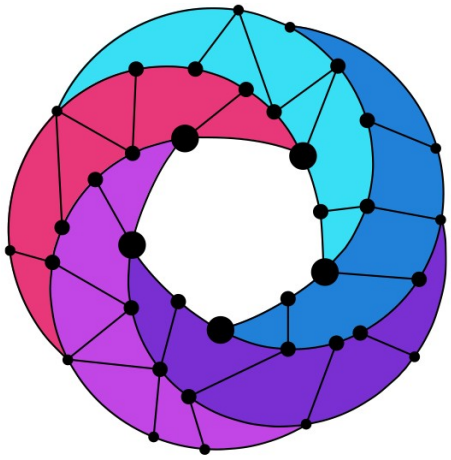


Open Source MANO

OSM Information Model

RELEASE TWO

April 2017

CONTENTS

Open Source MANO Information Model

Today's service providers have a growing interest in migrating custom, proprietary, hardware-based network functions to virtualized hardware in data centers—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).

This cloud model paradigm, Network Function Virtualization (NFV), is an initiative to decouple hardware from software. As a subset of software defined networking (SDN), NFV moves functions from specialized applications that run on COTS equipment (servers, storage, switches) to a virtual cloud environment. With virtualization, you use network resources without having to worry about where assets are physically located or how they are organized.

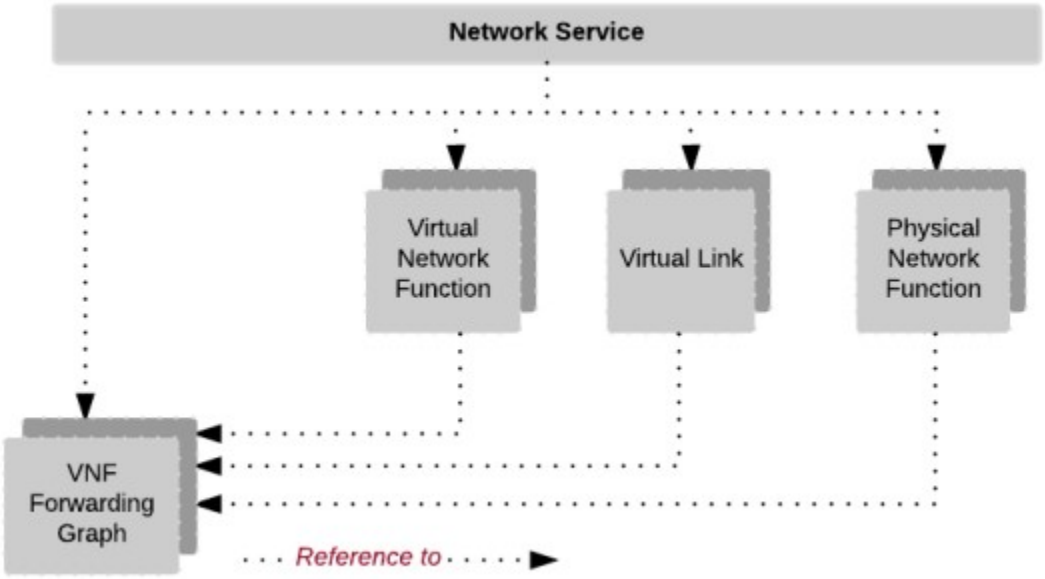
NFV is fundamentally changing how network services are deployed and managed, by promising agile service delivery, faster development cycles, and optimal resource usage. However, there is an obstacle to broader adoption due to a lack of a standard platform for deploying and managing VNFs and network services. In particular, there is a lack of consistency and openness in management and orchestration tools. The promise of NFV can be realized only if the VNF applications behave in a way that can be easily deployed, scaled, and managed.

The European Telecommunications Standards Institute (ETSI) has defined a framework for Network Functions Virtualization and Management and Orchestration Architectures (MANO). These architectures are broadly defined to let vendors interpret and extend in proprietary ways. As a result, SDN/NFV deployments can stall when network operators lack a standard, vendor-neutral way to deploy VNFs, and VNF builders lack a standard platform for delivering VNFs.

MANO Descriptor Overview

A descriptor is a configuration template that defines the main properties of managed objects in a network, such as a virtual network function (VNF) and network service (NS).

A network service describes the relationship between its network functions and the links that connect all network functions implemented in the NFVI network. These links interconnect the VNFs to connection points, which provide an interface to the existing network. Connection points also let you include physical network functions (PNFs) to expedite network expansion and evolution. The links in a network service form a network connection topology (NCT). The following diagram illustrates the high-level structure of the elements in the NS.



Information elements

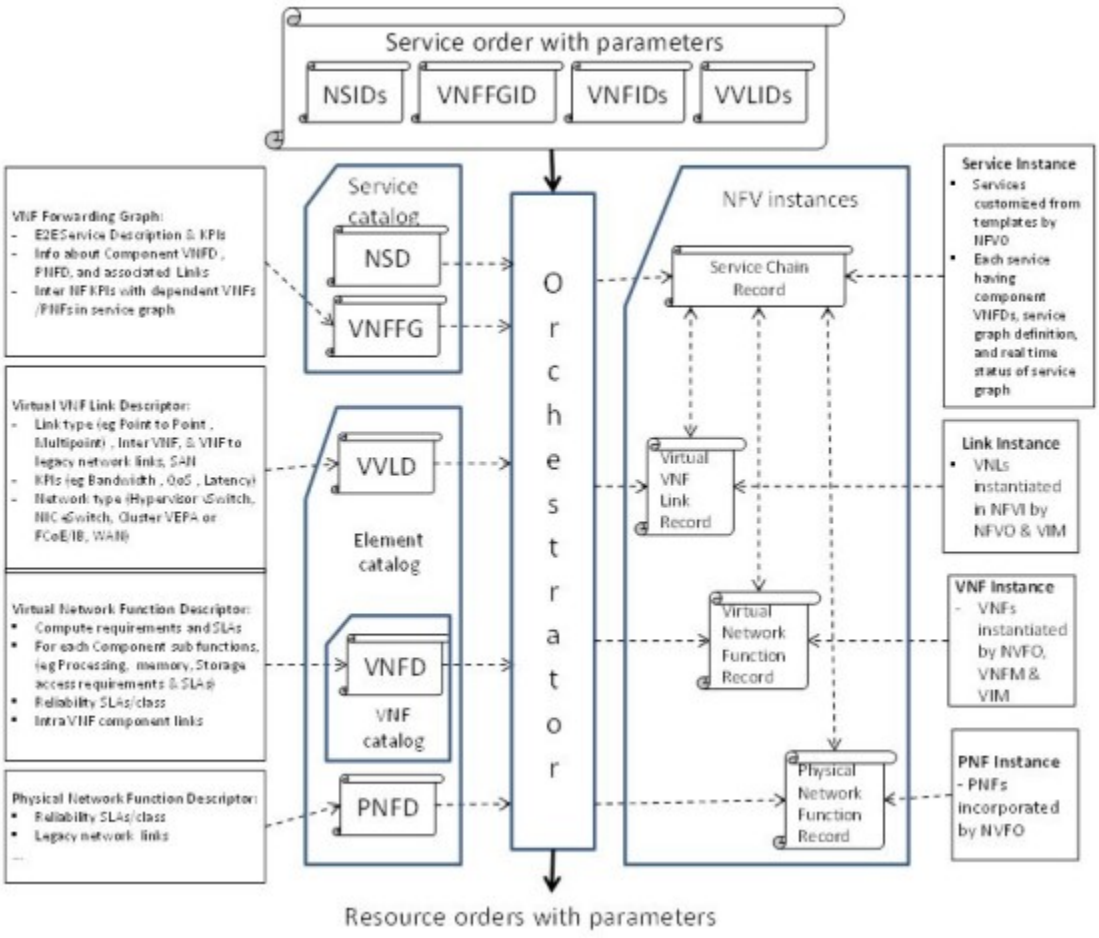
Information in a network service is structured into information elements, which might contain a single value or additional information elements that form a tree structure. Information element are classified as one of the following types.

- **Leaf element** — A single information element that specifies a value within the scope of the present document. The data type of the value is dependent on the information it should carry, such as a string, integer, list, container, choice.
- **Reference element** — An information element that contains a reference to another information element. The reference may be represented by a URI but depends on the concrete implementation of the information model.
- **Sub element** — An information element that specifies another level in the tree.

In each case, the number of occurrences of the same element inside its parent element is specified by the cardinality of the element. If the cardinality is a positive integer *n*, the element occurs exactly *n* times. If the cardinality is specified as a range, then the number of occurrences cannot exceed that range. A range that starts with "0" indicates that the element may be omitted.

Each of these information elements has a unique name along the whole path in the tree that leads to that element from the root of the tree.

The information elements can be used in two different contexts: As descriptors or as run-time instance records.



Source: ETSI NFV MANO WI document

Network Service Descriptor (NSD)

The network service descriptor (NSD) is the top-level construct used for designing the service chains, referencing all other descriptors that describe components that are part of that network service.

The NSD consists of static information elements as defined in the nsd base element and describes deployment flavors of the network service. The NSD is used by the NFV orchestrator to instantiate a network service.

The following four information elements are defined apart from the top-level network service:

- Virtual network function (VNF) information element
- Physical network function (PNF) information element
- Virtual Link (VL) information element.
- VNF forwarding graph (VNFFG) information element

The NSD references one or more VNFDs. These VNFs are connected VLDs, and the VNFFGD determines the traffic flow in the service chain. The NSD also exposes a set of connection points to enable connectivity to other network services or to the external world.

See "Network Service Descriptor (nsd:nsd)" on page 11.

VNF Descriptor (VNFD)

The virtual network function descriptor (VNFD) is a deployment template that describes the attributes of a single VNF. The VNFD is used primarily by the VNF manager (VNFM) in the process of VNF instantiation and lifecycle management of a VNF instance. The information provided in the VNFD is also used by the NFV orchestrator (NFVO) to manage and orchestrate network services and virtualized resources on the NFV infrastructure (NFVI).

The VNFD also contains:

- VNF images, which contain both the application and the Launchpad
- Connectivity (connection points and virtual links), interface, and KPI requirements that can be used by MANO functional blocks to establish appropriate virtual links:
    - Within the NFVI between its VNFC instances
    - Between a VNF instance and the outside network via endpoint interfaces to the other network functions in the network service
- Virtual deployment unit (VDU) that specifies the VM/VNFC compute, storage, and network requirements
- Platform resource requirements, such as CPU, memory, interfaces, and network
- Special characteristics related to EPA attributes and performance capabilities
- Scaling properties

See "VNFD Data Model (vnfd:vnfd)" on page 41.

PNF Descriptor (PNFD)

A PNF Descriptor describe a physical (legacy) network function. The only elements within a PNFD are the interconnections (connection points and virtual links). The PNFD is needed if the network service includes a physical device to support network evolution.

Virtual Link Descriptor (VLD)

A virtual link descriptor (VLD) is a deployment template that describes the resource requirements needed for a link between VNFs, PNFs and endpoints of the network service, which could be met by various link options that are available in the NFVI.

The NFVO can select an option after evaluating the VNFFG to determine the appropriate NFVI to be used based on functional (e.g. dual separate paths for resilience) and other needs (e.g. geography and regulatory requirements).

Network connections are defined by connection points and **virtual links**. There are three types of connection points:

- Connect a network service to the outside world, such as the network service endpoint, described in the NSD
- Connect between VNFs within a network service, such as the external interface of the VNF, described in the VNFD
- Connect between VMs, described in the VNFC

There are also two types of virtual links:

- External virtual links, which can be connected to network service endpoints and external VNF interfaces
- Internal virtual links, which can be connected to external VNF interfaces and VNFCs

Virtual links also follow the Metro Ethernet Forum E-LINE, E-TREE, and E-LAN services. Virtual link descriptors (VLDs) contain the bandwidth and QoS requirements of the interconnection.

VLDs are required for a functioning NSD.

See "Virtual Link Descriptor (nsd:vld)" on page 104.

Virtual Deployment Unit (VDU)

A VDU is a basic part of VNF. VDUs are virtual machines that host the network function, such as: Virtual machine specificationComputation properties (RAM size, disk size, memory page size, number of CPUs, number of cores per CPU, number of threads per core)Storage requirementsInitiation and termination scriptsHigh availability redundancy modelScale out/scale in limits A VDU is deployed as a VM in the VNF. See "VDU Data Model (vnfd:vdu)" on page 61.

Virtual Network Function Component (VNFC)

Software that provides VNFs can be structured into software components, the implementation view of a software architecture. These components can then be packaged into one or more images, the deployment view of a software architecture. These software components are called Virtual Network Function Components (VNFCs). VNFs are implemented with one or more VNFCs, where each VNFC instance generally maps 1:1 to a VM image or a container, as defined in the VDU.

ETSI MANO Common Object Models

The ETSI MANO object model framework supports 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.

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 by ETSI Management Orchestration:

- Maximize efficiency and performance by intelligently placing workloads on advanced NFV infrastructure capabilities, such as Enhanced Platform Awareness (EPA). See "Enhanced Platform Awareness Workload Placement" on page 10.
- 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

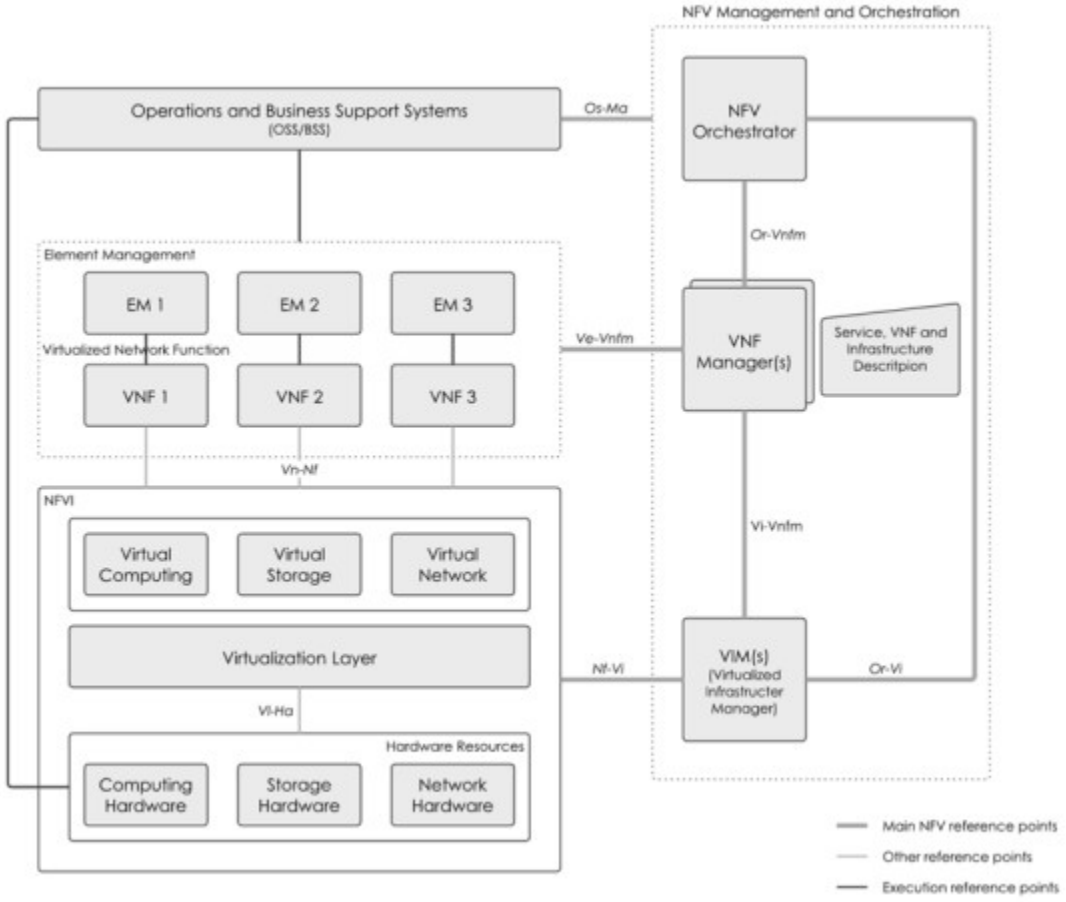
Network Functions Virtualisation (NFV); Management and Orchestration

Network Functions Virtualisation (NFV); Architectural Framework

MANO Software Components

The Virtualized Infrastructure Manager (VIM) is responsible for managing the NFV infrastructure (compute, network and storage resources).

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.



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

VNF Manager

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

- 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

The NFV Orchestrator (NFVO) is responsible for network service management, such as creating virtual function instances to meet service requirements. The NFVO manages network service lifecycle and resource orchestration across multiple VIMs.

Other NFVO functionality includes the following:

- Onboarding new network services and virtual network function packages
- Managing global resources, such as the 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 (NFVI) resource requests
- Managing the network service (NS) service templates in the NS catalog
- Simplifying the job of launching new network services

Interface

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.

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 an 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

Network Service Descriptor (nsd:nsd)

The network service descriptor (NSD) is the top-level construct used for designing the service chains, referencing all other descriptors that describe components that are part of that network service.

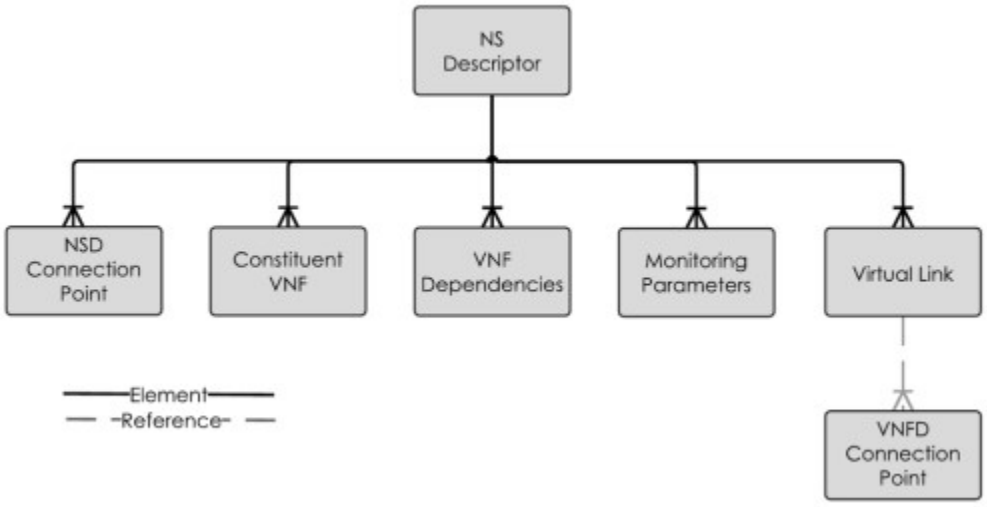
The NSD consists of static information elements as defined in the nsd base element and describes deployment flavors of the network service. The NSD is used by the NFV orchestrator to instantiate a network service.

The following four information elements are defined apart from the top-level network service:

- Virtual network function (VNF) information element
- Physical network function (PNF) information element
- Virtual Link (VL) information element.
- VNF forwarding graph (VNFFG) information element

The NSD references one or more VNFDs. These VNFs are connected VLDs, and the VNFFGD determines the traffic flow in the service chain. The NSD also exposes a set of connection points to enable connectivity to other network services or to the external world.

The following diagram illustrates the high-level object model for NSD.



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

NSD Data Model

Catalog for the network service descriptor.

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 to use as a label in the UI.
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. SVG format is preferred, but PNG is supported. Although there is no hard limit on size and dimension, a square image under 200px by 200px is preferred.
description	string	1	Description of the NSD.
version	string	1	Version of the NSD.
connection-point	list	0..n	A list of references to network service connection points. See "nsd:connection-point" on page 15.
vld	list	0..n	List of Virtual Link Descriptors (VLDs). See "Virtual Link Descriptor (nsd:vld)" on page 104.

ID	Type	Cardinality	Description
constituent-vnfd	list	0..n	List of Virtual Network Function Descriptors (VNFDs) that are part of this network service. See "nsd:constituent-vnfd" on page 16
placement-groups	list	0..n	List of placement groups at the NS level. See "nsd:placement-groups" on page 17.
ip-profiles-list	list	0..n	List of IP profiles. See "nsd:ip-profiles" on page 19.
vnf-dependency	list	0..n	List of VNF dependencies. See "nsd:vnf-dependency" on page 21.
vnffgd	list	0..n	List of VNF forwarding graph descriptor (VNFFGD). See VNF Forwarding Graph Descriptor (nsd:vnffgd).
monitoring-param	list	0..n	List of monitoring parameters at the network service level. See "nsd:monitoring-param" on page 22.
input-parameter-xpath	list	0..n	List of xpath to parameters inside the NSD that can be customized during instantiation. See "nsd:input-parameter-xpath" on page 26.
parameter-pool	list	0..n	Pool of parameter values that must be pulled from during configuration. See "nsd:parameter-pool" on page 27
service-primitive	list	0..n	Network service level configuration primitives. See "nsd:service-primitive" on page 28.

ID	Type	Cardinality	Description
initial-config-primitive	list	0..n	Set of configuration primitives to be executed when the network service comes up. See "nsd:initial-config-primitive" on page 33.
cloud-config	list	0..n	Configure the list of users and public keys to be injected as part of network service instantiation. See "nsd:cloud-config" on page 34.

See also

For an example of full output of the `nsd.yang` file, see "nsd.yang Model" on page 125

nsd:connection-point

A list of references to network service connection points.

Each network service (NS):

- Has one or more external connection points used to link the NS to other NS or to external networks.
- Exposes these connection points to the orchestrator.

The orchestrator can construct network service chains by joining the connection points between different network services.

Fields

ID	Type	Cardinality	Description
name	string	1	Name of the NS connection point.
type	enum	1	Type of connection point. Currently, only value `VPORT` (Virtual Port) is supported

See also

"Network Service Descriptor (nsd:nsd)" on page 11

nsd:constituent-vnfd

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

Fields

ID	Type	Cardinality	Description
member-vnfd-index	uint64	1	[Required] Identifier/index for the VNFD. **Note:** This separate ID is required so that multiple VNFs can be part of a single network service.
vnfd-id-ref	leafref	1	Identifier for the VNFD. This is a leafref to path: "/vnfd:vnfd-catalog/vnfd:vnfd/vnfd:id"
start-by-default	boolean	1	[Default *true*] VNFD is started as part of network service instantiation.

See also

"Network Service Descriptor (nsd:nsd)" on page 11

nsd:placement-groups

A list of placement groups at the network service level.

Fields

ID	Type	Cardinality	Description
name	string	1	[Key] Placement group construct to define the compute resource placement strategy in cloud environment.
requirement	string	1	Describes the intent/rationale behind this placement group. **Note:** This free-text field is for human consumption only
strategy	enum	1	Strategy associated with this placement group. The following values are supported: • COLOCATION: [Default] Share the physical infrastructure (hypervisor/network) among all members of this placement group. • ISOLATION: Do not share the physical infrastructure (hypervisor/network) among the members of this placement group
member-vnfd	list	0..n	List of VNFDs that are part of this placement group. See "member-vnfd" on page 18.

member-vnfd

List of VNFDs that are part of this placement group.

ID	Type	Cardinality	Description
member-vnf-index-ref	leafref	1	Member VNF index of this member VNF. "../../../constituent-vnfd/member-vnf-index"
vnfd-id-ref	leafref	1	Identifier for the VNFD. "../../../constituent-vnfd" + "[member-vnf-index = current()/../member-vnf-index-ref]" + "/vnfd-id-ref"

See also

"Network Service Descriptor (nsd:nsd)" on page 11

nsd:ip-profiles

List of IP profiles that describe the IP characteristics for the virtual link.

Fields

ID	Type	Cardinality	Description
name	string	1	Name of the IP profile.
description	string	1	Description of the IP profile.
ip-profile-params	container	1	Information about the IP profile. See "ip-profile-params" on page 19.

ip-profile-params

Information about the IP profile.

ID	Type	Cardinality	Description
ip-version	enum	1	[Default IPv4] Version of the Internet Protocol used.
subnet-address	inet:ip-prefix	1	Subnet IP prefix associated with this IP profile.
gateway-address	inet:ip-address	1	IP address of the default gateway associated with this IP profile.
security-group	string	1	Name of the security group.
dns-server	list	0..n	List of DNS servers associated with this IP profile. See "ip-profile-params:dns-server" on page 20.

ID	Type	Cardinality	Description
dhcp-params	container	1	Container for DHCP parameters. See "ip-profile-params:dhcp-params" on page 20.
subnet-prefix-pool	string	1	VIM-specific reference to pre-created subnet prefix.

ip-profile-params:dns-server

List of DNS servers associated with this IP profile.

ID	Type	Cardinality	Description
address	inet:ip-address	1	List of DNS servers associated with this IP profile.

ip-profile-params:dhcp-params

Container for DHCP parameters.

ID	Type	Cardinality	Description
enabled	boolean	1	[Default *true*] Indicates if DHCP is enabled.
start-address	inet:ip-address	1	Start IP address of the IP address range associated with DHCP domain.
count	uint32	1	Size of the DHCP pool associated with DHCP domain.

See also

"vnfd:connection-point" on page 59

"Network Service Descriptor (nsd:nsd)" on page 11

nsd:vnf-dependency

List of VNF dependencies in the Network Service Descriptor (NSD).

Fields

ID	Type	Cardinality	Description
vnf-source-ref	leafref	1	List of VNF dependencies. "../../constituent-vnfd/vnfd-id-ref"
vnf-depends-on-ref	leafref	1	Reference to VNFD that on which the source VNF depends. "../../constituent-vnfd/vnfd-id-ref"

See also

"Network Service Descriptor (nsd:nsd)" on page 11

nsd:monitoring-param

List of monitoring parameters from VNFs to propagate to the Network Service Record (NSR).

Fields

ID	Type	Cardinality	Description
id	string	1	Identifier for the parameter.
name	string	1	Name of the monitoring parameter
monitoring-param-ui-data	grouping	1	Grouping of monitoring parameters on the UI. See "monitoring-param-ui-data" on page 23
monitoring-param-value	grouping	1	See "monitoring-param-value" on page 24.
aggregation-type	enum	1	Aggregation type for the monitoring parameter: • AVERAGE • MINIMUM • MAXIMUM • COUNT • SUM
vnfd-monitoring-param	list	0..n	List of VNFD monitoring parameters. See "vnfd-monitoring-param" on page 25.

monitoring-param-ui-data

Grouping of monitoring parameters on the UI.

ID	Type	Cardinality	Description
description	string	1	Description of the monitoring parameter.
group-tag	string	1	Tag to group monitoring parameters.
widget-type	enum	1	Type of the widget, typically used by the UI: • HISTOGRAM • BAR • GAUGE • SLIDER • COUNTER • TEXTBOX
units	string	1	Units for the monitoring parameter, such as megabits per second.

monitoring-param-value

ID	Type	Cardinality	Description
value-type	enum	1	Type of the parameter value: • INT (default) • DECIMAL • STRING
numeric-constraints	container	1	Constraints for the numbers. See "monitoring-param-value:numeric-constraints" on page 24.
text-constraints	container	1	Constraints for the text strings. See "monitoring-param-value:text-constraints" on page 25.
value-integer	int64	1	Current value for integer parameter.
value-decimal	decimal164	1	Current value for decimal parameter, up to 4 fraction digits.
value-string	string	1	Current value for the string parameter.

monitoring-param-value:numeric-constraints

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

monitoring-param-value: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-monitoring-param

A list of VNFD monitoring parameters.

ID	Type	Cardinality	Description
vnfd-id-ref	leafref	1	A reference to a VNFD. "../../../constituent-vnfd" + "[member-vnf-index = current()/../member-vnf-index-ref]" + "/vnfd-id-ref"
vnfd-monitoring-param-ref	leafref	1	A reference to the VNFD monitoring parameter. "/vnfd:vnfd-catalog/vnfd:vnfd" + "[vnfd:id = current()/../vnfd-id-ref]" + "/vnfd:monitoring-param/vnfd:id"
member-vnf-index-ref	leafref	1	Optional reference to member-vnf within constituent-vnfds. "../../../constituent-vnfd/member-vnf-index"

See also

"Network Service Descriptor (nsd:nsd)" on page 11

nsd:input-parameter-xpath

List of XPaths to parameters inside the Network Service Descriptor that can be customized during instantiation.

Fields

ID	Type	Cardinality	Description
xpath	string	1	Xpath that specifies the element in a descriptor.
label	string	1	Descriptive string.
default-value	string	1	Default value for this input parameter.

See also

"Network Service Descriptor (nsd:nsd)" on page 11

nsd:parameter-pool

Pool of parameter values from which to pull during network service configuration.

Fields

ID	Type	Cardinality	Description
name	string	1	[Key] Name of the configuration value pool.
range	container	1	Create a range of values from which to populate the pool. See "range" on page 27.

range

Range of values from which to populate the pool

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.

See also

"Network Service Descriptor (nsd:nsd)" on page 11

nsd:service-primitive

Network service-level service primitives for the Network Service Descriptor.

Fields

ID	Type	Cardinality	Description
name	string	1	[Key] Name of the service primitive.
parameter	list	0..n	List of parameters for the service primitive. See "parameter" on page 29.
parameter-group	list	0..n	Grouping of parameters that are logically grouped in UI. See "parameter-group" on page 30.
vnf-primitive-group	list	0..n	List of service primitives grouped by the VNF. See "vnf-primitive-group" on page 31.
used-defined-script	string	1	A user-defined script.

parameter

List of parameters for the service primitive.

ID	Type	Cardinality	Description
name	string	1	Name of NSD parameter.
data-type	enum	1	Data type associated with the NSD parameter name: • STRING • INTEGER • BOOLEAN
mandatory	boolean	1	[Default *false*] Specifies whether this field is mandatory.
default-value	string	1	The default value for the field. **Note:** This value is required to set the read-only and hidden parameters.
parameter-pool	string	1	NSD parameter pool name to use for this parameter.
read-only	boolean	1	[Default *false*] The value should be dimmed in the UI. **Note:** Applies only to parameters with default values.
hidden	boolean	1	The value should be hidden in the UI. **Note:** Applies only to parameters with default values.
out	boolean	1	[Default *false*] Specifies if this is an output of the primitive execution

parameter-group

Grouping of parameters that are logically grouped in UI.

ID	Type	Cardinality	Description
name	string	1	Name of parameter group.
parameter	list	0..n	List of parameters to the service primitive. See "parameter-group-parameter" on page 30.
mandatory	boolean	1	[Default *true*] Specifies whether this parameter group is mandatory.

parameter-group-parameter

List of parameters to the service primitive.

ID	Type	Cardinality	Description
name	string	1	Name of NSD parameter.
data-type	enum	1	Data type associated with the NSD parameter name: • STRING • INTEGER • BOOLEAN
mandatory	boolean	1	[Default *false*] Specifies whether this field is mandatory.
default-value	string	1	The default value for the field.

ID	Type	Cardinality	Description
parameter-pool	string	1	NSD parameter pool name to use for this parameter.
read-only	boolean	1	[Default *false*] The value should be dimmed in the UI. **Note:** Applies only to parameters with default values.
hidden	boolean	1	The value should be hidden in the UI. **Note:** Applies only to parameters with default values.
out	boolean	1	[Default *false*] Specifies if this is an output of the primitive execution

vnf-primitive-group

List of service primitives grouped by the VNF.

ID	Type	Cardinality	Description
member-vnf-index-ref	uint64	1	Reference to member-vnf within constituent-vnfds. "../../../constituent-vnfd/member-vnf-index"
vnfd-id-ref	string	1	A reference to a VNFD. "../../../../nsd:constituent-vnfd + [nsd:id = current()/../nsd:id-ref] + /nsd:vnfd-id-ref"

ID	Type	Cardinality	Description
vnfd-name	leafref	1	Name of the VNFD. "/vnfd:vnfd-catalog/vnfd:vnfd" + "[vnfd:id = current()/../vnfd-id-ref]" + "/vnfd:name"
primitive	list	1	A list of VNF primitives. See "vnf-primitive-group:primitive" on page 32.

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.

See also

"Network Service Descriptor (nsd:nsd)" on page 11

nsd:initial-config-primitive

Initial set of configuration primitives for the NSD to be executed when the network service comes up.

Fields

ID	Type	Cardinality	Description
seq	uint64	1	Sequence number for the configuration primitive.
name	string	1	[Required] Name of the configuration primitive.
user-defined-script	string	1	A user-defined script.
parameter	list	0..n	List of parameters to the primitive. See "parameter" on page 33.

parameter

List of parameters to the primitive.

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

See also

"Network Service Descriptor (nsd:nsd)" on page 11

nsd:cloud-config

Configure the NSD cloud configuration parameters to include a list of public keys you want to inject into each VM as part of network service instantiation

Fields

ID	Type	Cardinality	Description
key-pair	list	0..n	Used to configure the list of public keys. See "key-pair" on page 34.
user	list	0..n	List of users to be added through cloud-config. See "user" on page 34.

key-pair

List of public keys to be injected as part of network service instantiation.

ID	Type	Cardinality	Description
name	string	1	[Key] Name of this key pair.
key	string	1	Key associated with this key pair.

user

List of users to be added through cloud-config.

ID	Type	Cardinality	Description
name	string	1	[Key] Name of the user.
user-info	string	1	The user's real name.

ID	Type	Cardinality	Description
key-pair	string	1	Used to configure the list of public keys to be injected as part of network service instantiation. See "user:key-pair" on page 35.

user:key-pair

Used to configure the list of public keys to be injected as part of metwprk service instantiation.

ID	Type	Cardinality	Description
name	string	1	Name of this key pair.
key	string	1	Key associated with this key pair.

See also

"vnfd:image-properties" on page 84

"Network Service Descriptor (nsd:nsd)" on page 11

Virtual Network Function Descriptor

The virtual network function descriptor (VNFD) is a deployment template that describes the attributes of a single VNF. The VNFD is used primarily by the VNF manager (VNFM) in the process of VNF instantiation and lifecycle management of a VNF instance. The information provided in the VNFD is also used by the NFV orchestrator (NFVO) to manage and orchestrate network services and virtualized resources on the NFV infrastructure (NFVI).

The VNFD also contains:

- VNF images, which contain both the application and the Launchpad
- Connectivity (connection points and virtual links), interface, and KPI requirements that can be used by MANO functional blocks to establish appropriate virtual links:
    - Within the NFVI between its VNFC instances
    - Between a VNF instance and the outside network via endpoint interfaces to the other network functions in the network service
- Virtual deployment unit (VDU) that specifies the VM/VNFC compute, storage, and network requirements
- Platform resource requirements, such as CPU, memory, interfaces, and network
- Special characteristics related to EPA attributes and performance capabilities
- Scaling properties

The VNFM uses a VNFD during the process of VNF instantiation and during lifecycle management of a VNF instance. Information provided in the VNFD is also used by the NFVO to manage and orchestrate network services and virtualized resources on the NFVI.

These VNFs are connected by Virtual Link Descriptors (VLDs). The NSD exposes a set of connection points to enable connectivity to other network services or to the external world

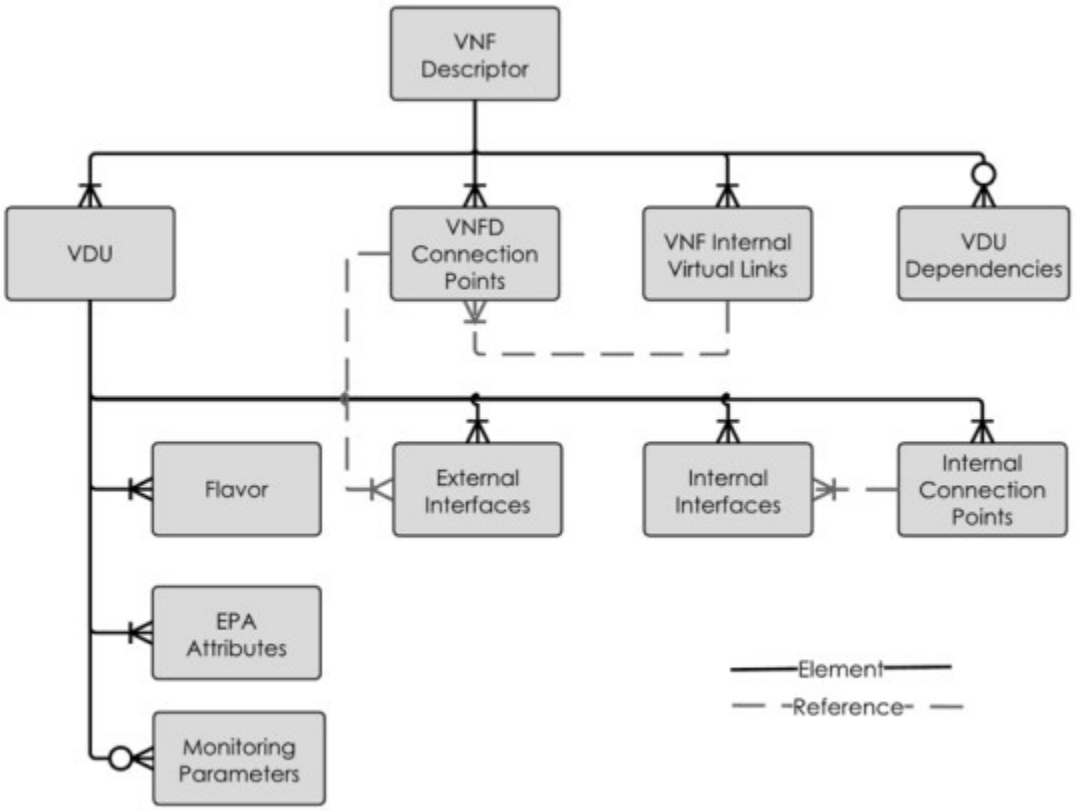
Each VNF is constructed from a set of discrete VNF Components (VNFCs). There can be one or more instance of each VNFC in the VNF. The VNFC is realized using a virtualized compute resource from the VIM. The virtualized compute resource can be either a Virtual Machine (VM) or a container. The VNFC includes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

The elements contained in the Virtual Deployment Unit (VDU) define the compute resource and software components. Specifically, the 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 connected the VNF1, VNF2 and VNF3. Virtual links VL1 and VL3 are used to connect the network services.

The following diagram illustrates the high-level object model for the VNFD. The VNFD contains lists of VDUs, internal connection points, internal virtual links, and 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 are used by the NSD to chain VNFs. The VDUs define the individual VNF components and capture 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 Amazon Web Services, some 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

Intel 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

Intel 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 (vnfd:vnfd)

Descriptor details for the Virtual Network Function (VNF).

The VNF includes one or more VDUs (the VM that hosts the network function), virtual links, and connection points. Each of these components (called nodes) has specific requirements, attributes, and capabilities, such as computational properties, that are defined in the VNF descriptor.

Fields

ID	Type	Cardinality	Description
id	string	1	Identifier for the VNFD.
name	string	1	VNFD name.
short-name	string	1	VNFD short name to use as a label in the UI.
vendor	string	1	Provider 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. SVG format is preferred, but PNG is supported. Although there is no hard limit on size and dimension, a square image under 200px by 200px is preferred.
description	string	1	Description of the VNFD.
version	string	1	Version of the VNFD.
vnf-configuration	container		Information about the VNF configuration for the management interface. See "vnfd:vnf-configuration" on page 44.

ID	Type	Cardinality	Description
mgmt-interface	container	1	Interface over which the VNF is managed. See "vnfd:mgmt-interface" on page 51.
internal-vld	list	0..n	List of Internal Virtual Link Descriptors (VLD). See "vnfd:internal-vld" on page 53.
ip-profiles	list	0..n	List of IP profiles. An IP profile describes the IP characteristics for the virtual-link. See "vnfd:ip-profiles" on page 57.
connection-point	list	0..n	The list for external connection points. See "vnfd:connection-point" on page 59.
vdu	list	0..n	List of virtual deployment units (VDUs). See "VDU Data Model (vnfd:vdu)" on page 61.
vdu-dependency	list	0..n	List of VDU dependencies, from which the orchestrator determines the order of startup for VDUs. See "vnfd:vdu-dependency" on page 95.
service-function-chain	enum	1	Type of node in the service function chaining (SFC) architecture: • UNAWARE (default) • CLASSIFIER: Function that classifies packets based on contents, and optionally local policies, such as subscriber aware. • SF: A service function that is responsible for specific treatment of received packets. • SFF: Service function forwarder delivers traffic received from the SFC network forwarder to one or more connected service functions through information carried in the SFC header.

ID	Type	Cardinality	Description
service-function-type	string	1	Type of service function. This field is temporarily set to string data type for ease of use. **Note:** On the OpenDaylight platform, this value must map to a service function type to support VNFFG.
monitoring-param	list	0..n	List of monitoring parameters for the VNF. See "vnfd:monitoring-param" on page 96.
placement-groups	list	0..n	Placement group construct to define the compute resource placement strategy in cloud environment. See "vnfd:placement-groups" on page 102.

See also

"vnfd.yang Model" on page 139

vnfd:vnf-configuration

Information about the VNF configuration for the management interface.

Note: If the network service contains multiple instances of the same VNF, each VNF instance may have a different configuration.

Fields

ID	Type	Cardinality	Description
config-method	choice	1	Defines the configuration method for the VNF. See "config-method" on page 45.
config-access	container	1	IP address to be used to configure this VNF. See "config-access" on page 46.
config-attributes	container	1	Miscellaneous input parameters to be considered while processing the NSD to apply configuration. See "config-attributes" on page 46.
service -primitive	list	0..n	List of service primitives supported by the configuration agent for this VNF. See "service-primitive" on page 46.
initial-config-primitive	list	0..n	Initial set of configuration primitives. See "initial-config-primitive" on page 48.
config-template	string	1	Configuration template for each VNF.

config-method

ID	Type	Cardinality	Description
script	container	1	Use a custom script for configuring the VNF. **Note:** This script is executed in the Launchpad. All required dependencies for the script must be available in the Launchpad system before the script runs. See "config-method:script" on page 45.
juju	container	1	Use Juju for configuring the VNF. See "config-method:juju" on page 45.

config-method:script

Script container for configuring the VNF. This script will be executed in the Launchpad, and all required dependencies for the script should be available in the Launchpad system.

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

config-method:juju

Juju container for configuring the VNF.

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

config-access

VNF configuration access.

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

config-attributes

Miscellaneous input parameters to be considered while processing the NSD.

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

service-primitive

List of primitives supported by the configuration agent for this VNF.

ID	Type	Cardinality	Description
name	string	1	[Key] Name of the config primitive.

ID	Type	Cardinality	Description
parameter	list	0..n	List of parameters to the configuration primitive. See "service-primitive:parameter" on page 47
user-defined-script	string	1	A user-defined script. If a script is defined, the script will be executed using bash.

service-primitive:parameter

List of parameters to the primitive.

ID	Type	Cardinality	Description
name	string	1	Name of NSD parameter.
data-type	enum	1	Data type associated with the NSD parameter name: • STRING • INTEGER • BOOLEAN
mandatory	boolean	1	[Default *false*] Specifies whether this field is mandatory.
default-value	string	1	The default value for the field. **Note:** This value is required to set the read-only and hidden parameters.
parameter-pool	string	1	NSD parameter pool name to use for this parameter.

ID	Type	Cardinality	Description
read-only	boolean	1	[Default *false*] The value should be dimmed in the UI. **Note:** Applies only to parameters with default values.
hidden	boolean	1	The value should be hidden in the UI. **Note:** Applies only to parameters with default values.
out	boolean	1	[Default *false*] Specifies if this is an output of the primitive execution.

initial-config-primitive

Initial set of configuration primitives.

ID	Type	Cardinality	Description
seq	uint64	1	[Key] Sequence number for the configuration primitive.
primitive-type	choice	1	Primitive type to use. See "initial-config-primitive:primitive-type" on page 49
config-primitive-ref	leafref	1	Reference to a config primitive name. **Note:** The referenced config primitive should have all the input parameters predefined either with default values or dependency references

initial-config-primitive:primitive-type

ID	Type	Cardinality	Description
name	string	1	Name of the configuration primitive.
parameter	list	0..n	List of parameters to the configuration primitive. See "initial-config-primitive:parameter" on page 49.
user-defined-script	string	1	A user-defined script

initial-config-primitive:parameter

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

See also

"vnfd:mgmt-interface" on page 51

"VNFD Data Model (vnfd:vnfd)" on page 41

vnfd:mgmt-interface

Interface over which the VNF is managed.

Fields

ID	Type	Cardinality	Description
endpoint-type	choice	1	Specifies the type of management endpoint to use. See "endpoint-type" on page 51
port	inet:port-number	1	Port number for the management interface.
dashboard-params	container	1	Parameters for the VNF dashboard of the management interface. See "dashboard-params" on page 52.

endpoint-type

Specifies the type of management endpoint to use:

ID	Type	Cardinality	Description
ip-address	inet:ip-address	1	Use static IP address for managing the VNF.
vdu-id	leafref	1	Use the default management interface on this VDU: "../../vdu/id".
cp	leafref	1	Use the IP address for the VNFD associated with this connection point endpoint: "../../connection-point/name"

dashboard-params

Parameters for the VNF dashboard of the management interface.

ID	Type	Cardinality	Description
path	string	1	The HTTP path for the dashboard.
https	boolean	1	[Default *false*] Choose HTTPS instead of HTTP.
port	uint16	1	The HTTP port for the dashboard.

See also

"VNFD Data Model (vnfd:vnfd)" on page 41

vnfd:internal-vld

List of internal Virtual Link Descriptors (VLDs). Internal VLDs describe the basic topology of the connectivity—such as E-LAN— between internal VNF Components (VNFC) within the system.

Fields

ID	Type	Cardinality	Description
id	string	1	Identifier for the internal VLD.
name	string	1	Name of the internal VLD.
short-name	string	1	Short name for the internal VLD to display as a label in the UI.
vendor	string	1	Provider of the VLD.
description	string	1	Description of the internal VLD.
version	string	1	Version of the VLD.
type	enum	1	Type of virtual link: ELAN – A multipoint service that connects 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	list	0..n	List of internal connection points in this VLD: See "internal-connection-point" on page 54

ID	Type	Cardinality	Description
virtual-connection-points	list	0..n	A list of virtual connection points associated with the virtual link. These connection points are not directly associated with VDUs. See "virtual-connection-points" on page 55.
provider-network	container	1	Container for the provider network. See "provider-network" on page 56
init-params	choice	1	Extra parameters for VLD instantiation. See "init-params" on page 56.

internal-connection-point

List of internal connection points in this VLD:

ID	Type	Cardinality	Description
id-ref	leafref	1	Reference to the internal connection point ID: "../../../vdu/internal-connection-point/id"

virtual-connection-points

List of virtual-connection points associated with this virtual Link. These connection points are not directly associated with any VDUs.

ID	Type	Cardinality	Description
name	string	1	Name of the connection point.
id	string	1	Identifier for the internal connection points.
short-name	string	1	Short name of the connection point to display as a label in the UI.
type	enum	1	Type of connection point: VPORT: Virtual Port
port-security-enabled	boolean	1	Enables or disables the port security for the connection-point. When set to *True*, the resource orchestrator passes the value to the VIM when the connection-point is created to filter traffic. **Note:** This value is supported on OpenStack only.
static-ip-address	inet:ip-address	1	Static IPv4 or IPv6 address for the internal/external connection point in the VNFD. When you instantiate a VNF, the static IP for the connection point is passed to the VIM.
associated-cps	list	0..n	List of connection points associated with virtual connection point. "../../internal-connection-point/id-ref"

provider-network

Container for the 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	1	Identifies the type of the overlay network, which is a virtual network that is built on top of an existing network and is supported by its infrastructure. Supported values are: • LOCAL — Provider network implemented in a single compute node. • FLAT — Provider network shared by all tenants. • VLAN — Provider network implemented using 802.1Q tagging. • VXLAN — Provider networks implemented using RFC 7348. • GRE — Provider networks implemented using GRE tunnels.
segmentation-id	uint32	1	Segmentation ID

init-params

Extra parameters for VLD instantiation.

ID	Type	Cardinality	Description
vim-network-name	string	1	Name of pre-provisioned network in the VIM (cloud) account.
ip-profile-ref	string	1	Named reference to an ip-profile object.

vnfd:ip-profiles

List of IP profiles that describe the IP characteristics for the virtual link.

Fields

ID	Type	Cardinality	Description
name	string	1	Name of the IP profile.
description	string	1	Description of the IP profile.
ip-profile-params	container	1	Information about the IP profile. See "ip-profile-params" on page 57.

ip-profile-params

Information about the IP profile.

ID	Type	Cardinality	Description
ip-version	enum	1	[Default IPv4] Version of the Internet Protocol used.
subnet-address	inet:ip-prefix	1	Subnet IP prefix associated with this IP profile.
gateway-address	inet:ip-address	1	IP address of the default gateway associated with this IP profile.
security-group	string	1	Name of the security group.
dns-server	list	0..n	List of DNS servers associated with this IP profile. See "ip-profile-params:dns-server" on page 58.

ID	Type	Cardinality	Description
dhcp-params	container	1	Container for DHCP parameters. See "ip-profile-params:dhcp-params" on page 58.
subnet-prefix-pool	string	1	VIM-specific reference to pre-created subnet prefix.

ip-profile-params:dns-server

List of DNS servers associated with this IP profile.

ID	Type	Cardinality	Description
address	inet:ip-address	1	List of DNS servers associated with this IP profile.

ip-profile-params:dhcp-params

Container for DHCP parameters.

ID	Type	Cardinality	Description
enabled	boolean	1	[Default *true*] Indicates if DHCP is enabled.
start-address	inet:ip-address	1	Start IP address of the IP address range associated with DHCP domain.
count	uint32	1	Size of the DHCP pool associated with DHCP domain.

vnfd:connection-point

List of external connection points, in which each VNF:

- Has one or more points that are used to connect a VNF to other VNFs or to external networks
- Exposes these connection points to the orchestrator (NFVO)

The orchestrator constructs network services by joining the connection points between different VNFs.

The orchestrator uses VLDs and VNFFGs at the network service level to construct network services.

Fields

ID	Type	Cardinality	Description
name	string	1	Name of the connection point.
id	string	1	Identifier for the internal connection points.
short-name	string	1	Short name of the connection point to display as a label in the UI.
type	enum	1	Type of connection point: VPORT: Virtual Port
port-security-enabled	boolean	1	Enables or disables the port security for the connection-point. When set to *True*, the resource orchestrator passes the value to the VIM when the connection-point is created to filter traffic. **Note:** This value is supported on OpenStack only.
static-ip-address	inet:ip-address	1	Static IPv4 or IPv6 address for the internal/external connection point in the VNFD. When you instantiate a VNF, the static IP for the connection point is passed to the VIM.

See also

"nsd:ip-profiles" on page 19

"VNFD Data Model (vnfd:vnfd)" on page 41

VDU Data Model (vnfd:vdu)

A VDU is a basic part of VNF. VDUs are virtual machines that host the network function, such as:

- Virtual machine specification
- Computation properties (RAM size, disk size, memory page size, number of CPUs, number of cores per CPU, number of threads per core)
- Storage requirements
- Initiation and termination scripts
- High availability redundancy model
- Scale out/scale in limits

Fields

ID	Type	Cardinality	Description
id	string	1	Unique identifier for the 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 virtual machine (VM) instance. See "vnfd:vm-flavor" on page 67.
guest-epa	container	1	EPA attributes for the guest operating system. See "vnfd:guest-epa" on page 68.

ID	Type	Cardinality	Description
vswitch-epa	container	1	EPA attributes for Open vSwitch. See "vnfd:vswitch-epa" on page 73.
hypervisor-epa	container	1	EPA attributes for the hypervisor. See "vnfd:hypervisor-epa" on page 74.
host-epa	container	1	EPA attributes for the host operating system. See "vnfd:host-epa" on page 75.
alarm	list	0..n	A list of alarms. See "vnfd:alarm" on page 80.
image-properties	container	1	Image properties, such as name and checksum. See "vnfd:image-properties" on page 84
cloud-init-input	choice	1	Specifies how the contents of `cloud-init` script are provided. See "vnfd:cloud-init-input" on page 85.
supplemental-boot-data	container	1	Container for custom VIM metadata. **Note:** This container is provided for convenience. You should use cloud-init ("vnfd:image-properties" on page 84 and) and VCA ("vnfd:vnf-configuration" on page 44 and "nsd:initial-config-primitive" on page 33) to define VNF configuration. See "vnfd:supplemental-boot-data" on page 86.
internal-connection-point	list	0..n	List for internal connection points. See "vnfd:internal-connection-point" on page 87.

ID	Type	Cardinality	Description
internal-interface	list	0..n	List of internal interfaces for the VNF. See "vnfd:internal-interface" on page 88.
external-interface	list	0..n	List of external interfaces for the VNF. See "vnfd:external-interface" on page 90.
volumes	list	0..n	List of disk-volumes to be attached to VDU. See "vnfd:volumes" on page 92.

Schema

```
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;
```

```
        list alarm {
          key "alarm-id";

          uses manotypes:alarm;
        }

        uses manotypes:image-properties;

        choice cloud-init-input {
          description
            "Indicates how the contents of cloud-init script are provided.
             There are 2 choices - inline or in a file";

          case inline {
            leaf cloud-init {
              description
                "Contents of cloud-init script, provided inline, in cloud-config
format";
              type string;
            }
          }

          case filename {
            leaf cloud-init-file {
              description
                "Name of file with contents of cloud-init script in cloud-config
format";
                type string;
            }
          }
        }

        uses manotypes:supplemental-boot-data;

        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;
        }

        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 volumes {
          key "name";

          leaf name {
            description "Name of the disk-volumes, e.g. vda, vdb etc";
            type string;
          }

          uses manotypes:volume-info;
        }
      }
```

See also

"VNFD Data Model (vnfd:vnfd)" on page 41

vnfd:vm-flavor

Flavor is an alternative term for a VM instance type.

Fields

ID	Type	Cardinality	Description
vcpu-count	uint16	1	Number of VCPUs for the VM.
memory-mb	uint64	1	Amount of memory in MB to allocate to the VM.
storage-gb	uint64	1	Amount of disk space in GB to allocate to the VM.

See also

"VDU Data Model (vnfd:vdu)" on page 61

"VNFD Data Model (vnfd:vnfd)" on page 41

vnfd:guest-epa

EPA attributes for the guest operating system.

Fields

ID	Type	Cardinality	Description
trusted-execution	boolean	1	If set to *true*, indicates this VM should be allocated from trusted pool.
mempage-size	enum	1	Memory page allocation size for the VM: • LARGE: Requires hugepages (either 2MB or 1GB) • SMALL: Doesn't require hugepages • SIZE_2MB: Requires 2MB hugepages • SIZE_1GB: Requires 1GB hugepages • PREFER_LARGE: Prefers hugepages **Note:** If a VM requires hugepages, choose LARGE or SIZE_2MB or SIZE_1GB. If the VM prefers hugepages, choose PREFER_LARGE.
cpu-pinning-policy	enum	1	Defines the association between virtual CPUs in the 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

ID	Type	Cardinality	Description
cpu-thread-pinning-policy	enum	1	Defines 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.
pcie-device	list	0..1	List of PCIE passthrough devices. See "pcie-device" on page 70
numa-unaware	empty		Details about the numa-node-policy are null.
numa-node-policy	container	1	Defines NUMA topology of the guest, specifying if the guest should run on a host with one numa node or multiple NUMA nodes. **Example:** A guest might need 8 VCPUs and 4 GB of memory with the VCPUs and memory distributed across multiple NUMA nodes. In this scenario, NUMA node 1 could run with 6 VCPUs and 3GB, and NUMA node 2 could run with 2 vcpus and 1GB. See "numa-node-policy" on page 70.

pcie-device

List of PCIE passthrough devices.

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

numa-node-policy

Defines NUMA topology of the guest.

ID	Type	Cardinality	Description
node-cnt	uint16	1	Number of NUMA nodes to expose to the VM.
mem-policy	enum	1	Defines how to allocate memory in a multi-node scenario: • STRICT: The memory must be allocated from the memory attached to the NUMA node. • PREFERRED: The memory should be allocated from the memory attached to the NUMA node
node	list	0..n	List of NUMA nodes. See "numa-node-policy:node" on page 70.

numa-node-policy:node

ID	Type	Cardinality	Description
id	uint64	1	NUMA node identification. Typically 0 or 1.

ID	Type	Cardinality	Description
vpcu	list	0..n	List of VPCUs to allocate on this NUMA node. See "numa-node-policy:node:vcpu" on page 71.
memory-mb	uint64	1	Memory size in MB for this NUMA node.
num-cores	uint8	1	Number of cores.
paired-threads	container	1	Container for paired threads. See "numa-node-policy:node:paired-threads" on page 71.
num-threads	uint8	1	OpenMANO NUMA type selection.

numa-node-policy:node:vcpu

ID	Type	Cardinality	Description
id	uint64	1	List of VCPUs IDs to allocate on this NUMA node.

numa-node-policy:node: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. See "numa-node-policy:node:paired-threads:paired-thread-ids" on page 72.

numa-node-policy:node:paired-threads:paired-thread-ids

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

See also

"VDU Data Model (vnfd:vdu)" on page 61

"VNFD Data Model (vnfd:vnfd)" on page 41

vnfd:vswitch-epa

EPA attributes for Open vSwitch.

Fields

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

See also

"VDU Data Model (vnfd:vdu)" on page 61

"VNFD Data Model (vnfd:vnfd)" on page 41

vnfd:hypervisor-epa

EPA attributes for the hypervisor.

Fields

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

See also

"VDU Data Model (vnfd:vdu)" on page 61

"VNFD Data Model (vnfd:vnfd)" on page 41

vnfd:host-epa

Specifies the host-level EPA attributes.

Fields

ID	Type	Cardinality	Description
cpu-model	enum	1	Host CPU model. **Supported values:** • 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

ID	Type	Cardinality	Description
cpu-arch	enum	1	Host CPU architecture. **Supported values:** • PREFER_X86 • REQUIRE_X86 • PREFER_X86_64 • REQUIRE_X86_64 • PREFER_I686 • REQUIRE_I686 • PREFER_IA64 • REQUIRE_IA64 • PREFER_ARMV7 • REQUIRE_ARMV7 • PREFER_ARMV8 • REQUIRE_ARMV8
cpu-vendor	enum	1	Host CPU vendor. **Supported values:** • PREFER_INTEL • REQUIRE_INTEL • PREFER_AMD • REQUIRE_AMD
cpu-socket-count	uint64	1	Number of sockets on the host.
cpu-core-count	uint64	1	Number of cores on the host.
cpu-core-thread-count	uint64	1	Number of threads per cores on the host.

ID	Type	Cardinality	Description
cpu-feature	list	0..1	List of CPU features. See "cpu-feature" on page 77
om-cpu-model-string	string	1	OpenMano CPU model string.
om-cpu-feature	list	0..n	OpenMano CPU features. See "om-cpu-feature" on page 79.

cpu-feature

List of CPU features.

ID	Type	Cardinality	Description

ID	Type	Cardinality	Description
feature	enum	1	**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. **Note:** 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. **Supported values:** • PREFER_AES • REQUIRE_AES • PREFER_CAT • REQUIRE_CAT • PREFER_CMT • REQUIRE_CMT • PREFER_DDIO • REQUIRE_DDIO • REQUIRE_VME • PREFER_VME • REQUIRE_DE • PREFER_DE • REQUIRE_PSE • PREFER_PSE

om-cpu-feature

OpenMANO CPU features

ID	Type	Cardinality	Description
feature	string	1	CPU feature.

See also

"VDU Data Model (vnfd:vdu)" on page 61

"VNFD Data Model (vnfd:vnfd)" on page 41

vnfd:alarm

Information about alarms.

Fields

ID	Type	Cardinality	Description
alarm-id	string	1	Reserved field for the identifier assigned by the VIM provider.
name	string	1	A human-readable string to identify the alarm.
description	string	1	Description of the alarm.
vdur-id	string	1	Identifier of the VDU record (VDUR) associated with this alarm.
actions	container	1	Actions related to the alarm. See "actions" on page 82.
repeat	boolean	1	[Default *true*] Indicates whether the alarm should emit repeatedly after the associated threshold has been crossed.
enabled	boolean	1	[Default *true*] Indicates whether the alarm has been enabled or disabled.
severity	enum	1	Defines a measure of the important or urgency of the alarm: • LOW • MODERATE • CRITICAL

ID	Type	Cardinality	Description
metric	enum	1	Defines metric types that can be tracked by this alarm. • CPU_UTILIZATION • MEMORY_UTILIZATION • STORAGE_UTILIZATION
statistic	enum	1	Defines type of statistic to use to measure a metric, which determines threshold crossing for an alarm. • AVERAGE • MINIMUM • MAXIMUM • COUNT • SUM
operation	enum	1	Defines the relational operator to use if an alarm should be triggered when the metric statistic goes above or below a specified threshold value. • GE — Greater than or equal to • LE — Less than or equal to • GT — Greater than • LT — Less than • EQ — Equal
value	decimal164	1	Defines the threshold (up to 4 fraction digits) that, if crossed, will trigger the alarm.
period	uint32	1	Defines the length of time (seconds) for which metric data are collected to evaluate the chosen statistic.

ID	Type	Cardinality	Description
evaluation	uint32	1	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. **Note:** This value can be used to mitigate spikes in the metric that may skew the statistic of interest.

actions

ID	Type	Cardinality	Description
ok	list	0..n	See "actions" on page 82
insufficient-data	list	0..n	See "actions:insufficient-data" on page 83
alarm	list	0..n	See "actions:alarm" on page 83

actions:ok

ID	Type	Cardinality	Description
url	string	1	[Key]

actions:insufficient-data

ID	Type	Cardinality	Description
url	string	1	[Key]

actions:alarm

ID	Type	Cardinality	Description
url	string	1	[Key]

See also

"VDU Data Model (vnfd:vdu)" on page 61

"VNFD Data Model (vnfd:vnfd)" on page 41

vnfd:image-properties

Provides image name and checksum file of a cloud instance (VM) during launch.

Fields

ID	Type	Cardinality	Description
image	string	1	Image name for the software image. If the image name is found within the VNF package, it will be uploaded to all cloud accounts during the 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 VIM.

See also

"VDU Data Model (vnfd:vdu)" on page 61

vnfd:cloud-init-input

Specifies how the contents of `cloud-init` script are provided.

Fields

ID	Type	Cardinality	Description
cloud-init	string	1	Provide contents of cloud-init script inline, in cloud-config format.
cloud-init-file	string	1	Pass the cloud-init script as a separate file to the resource orchestrator, outside the descriptor, in cloud-config format. The VNF package may reference the cloud-init file by name in the `vnfd:vdu` descriptor.

See also

"VDU Data Model (vnfd:vdu)" on page 61

vnfd:supplemental-boot-data

Grouping for VIM data lets you pass additional data to the VIM to enable a VDU to bootstrap itself.

Note: This container is provided for convenience. You should use cloud-init ("vnfd:image-properties" on page 84 and "vnfd:cloud-init-input" on page 85) and VCA ("vnfd:vnf-configuration" on page 44 and "nsd:initial-config-primitive" on page 33) to define VNF configuration.

Fields

ID	Type	Cardinality	Description
config-file	list	0..n	List of configuration files to be written on an additional drive. These files reside in the cloud-init directory described in Launchpad Package Formats in VNF Configuration & Integration. See "config-file" on page 86.
boot-data-drive	boolean	1	[Default *false*] Specifies whether the VIM should implement an additional drive to host user config data (represented by cloud-init or cloud-init-file in the YANG model) and metadata.

config-file

List of configuration files to be written on an additional drive.

ID	Type	Cardinality	Description
source	string	1	Name of the configuration file.
dest	string	1	Full path of the destination on the guest.

See also

"vnfd:image-properties" on page 84 and "VDU Data Model (vnfd:vdu)" on page 61

vnfd:internal-connection-point

List of 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.

Fields

ID	Type	Cardinality	Description
name	string	1	Name of the connection point.
id	string	1	[Key] Identifier for the internal connection points.
short-name	string	1	Short name to use as a label in the UI.
type	enum	1	Type of connection point: VPORT: Virtual Port
port-security-enabled	boolean		Specifies whether to enable port security for the port.
static-ip-address	inet:ip-address	1	Static IP address for the connection point

See also

"VDU Data Model (vnfd:vdu)" on page 61

"VNFD Data Model (vnfd:vnfd)" on page 41

vnfd:internal-interface

List of internal interfaces for virtual network functions, which enable intra-VNF traffic.

Fields

ID	Type	Cardinality	Description
name	string	1	[Key] Name of the internal interface inside the VDU. **Note:** This name has only local significance to the VDU.
vdu-internal-connection-point-ref	leafref	1	Reference to an internal connection point: "../../internal-connection-point/id"
virtual-interface	container	1	Container for the virtual interface properties. See "virtual-interface" on page 89.

virtual-interface

Container for the virtual interface properties.

ID	Type	Cardinality	Description
type	enum	1	Specifies the type of virtual interface between VM and host: • VIRTIO: [Default] Use the traditional VIRTIO interface • PCI-PASSTHROUGH: Use PCI-PASSTHROUGH interface • SR-IOV: Use SR-IOV interface • E1000 : Emulate E1000 interface • RTL8139 : Emulate RTL8139 interface • PCNET : Emulate PCNET interface • OM-MGMT: Used to specify OpenMANO management internal-connection type
vpci	string	1	Specifies the virtual PCI address in format dddd:dd:dd.d. For example: 0000:00:12.0 **Note:** This information can be used to pass as metadata during VM creation.
bandwidth	uint64	1	Specifies the aggregate bandwidth requirement for the NIC.

See also

"VDU Data Model (vnfd:vdu)" on page 61

"VNFD Data Model (vnfd:vnfd)" on page 41

vnfd:external-interface

List of external interfaces for virtual network functions, which enable intra-VNF traffic.

Fields

ID	Type	Cardinality	Description
name	string	1	[Key] Name of the external interface inside the VDU. **Note:** This name has only local significance to the VDU.
vnfd-connection-point-ref	leafref	1	Reference to an external connection point: "../../../connection-point/name"
virtual-interface	container	1	Container for the virtual interface properties.

virtual-interface

Container for the virtual interface properties.

ID	Type	Cardinality	Description
type	enum	1	Specifies the type of virtual interface between VM and host: • VIRTIO: [Default] Use the traditional VIRTIO interface • PCI-PASSTHROUGH: Use PCI-PASSTHROUGH interface • SR-IOV: Use SR-IOV interface • E1000 : Emulate E1000 interface • RTL8139 : Emulate RTL8139 interface • PCNET : Emulate PCNET interface • OM-MGMT: Used to specify OpenMANO management internal-connection type
vpci	string	1	Specifies the virtual PCI address in format dddd:dd:dd.d. For example: 0000:00:12.0 **Note:** This information can be used to pass as metadata during VM creation.
bandwidth	uint64	1	Specifies the aggregate bandwidth requirement for the NIC.

See also

"VDU Data Model (vnfd:vdu)" on page 61

"VNFD Data Model (vnfd:vnfd)" on page 41

vnfd:volumes

Defines disk volumes to be attached to the VDU, such as when a VNF requires multiple disks to boot the virtual machine.

Note: If you use vnfd:volumes, do not specify the top-level image properties in vnfd:vdu. The data required to instantiate a virtual machine is derived from the volumes descriptor. This means that the `image` and `image-checksum` fields specified in vnfd:volumes replace the same fields in the vnfd:vdu descriptor.

If you have a descriptor that uses the vnfd:vdu image fields and you want to add a second volume, create a volume to represent the vnfd:vdu image fields, leave the vnfd:vdu image fields blank, and add a second volume.

Volumes are automatically deleted when the VDU is destroyed.

View a list of volumes by volume-id on the Launchpad UI Compute Topology page.

Fields

ID	Type	Cardinality	Description
name	string	1	[Required] Name of the disk volumes. You must specify the appropriate device name ("dev/NAME") of the block device exposed within the VM, such as "vda", "vdb", "sda", "sdb", and so on.
description	string	1	Description of the volume.
size	uint64	1	[Required] Size of the volume, in GB.
volume-source	choice	1	[Required] Defines the source of the volume. Supported values: • ephemeral – Empty disk • image – Reference to the image to use for the volume See "volume-source" on page 93.

ID	Type	Cardinality	Description
device_bus	enum	1	[Required] Type of disk-bus on which this disk is exposed to the guest operating system: • IDE • VIRTIO • SCSI
device_type	enum	1	[Required] Type of device as exposed to the guest operating system: • DISK • CDROM (not supported by Brocade vCPE CAL)

volume-source

Specify either ephemeral or image or volume-ref as the volume source.

ID	Type	Cardinality	Description
ephemeral	empty	1	Blank volume. **Note:** ephemeral is not a supported volume source on Brocade vCPE VIM.
image	string	1	Image name of the software image to use. If the image name is found within the VNF package, it will be uploaded to all cloud accounts during the onboarding process. Otherwise, the image must be added to the VIM account with the same name as entered in this field.

ID	Type	Cardinality	Description
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. **Note:** The image-checksum field is used only if you specify the `image` field as the volume source.

See also

"VDU Data Model (vnfd:vdu)" on page 61

"Virtual Network Function Descriptor" on page 36

vnfd:vdu-dependency

List of virtual deployment unit (VDU) dependencies, from which the orchestrator determines the order of startup among the VDUs.

Fields

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

See also

"VDU Data Model (vnfd:vdu)" on page 61

"VNFD Data Model (vnfd:vnfd)" on page 41

vnfd:monitoring-param

List of monitoring parameters at the VNF level.

Fields

ID	Type	Cardinality	Description
http-endpoint	list	0..n	List of http endpoints to be used by monitoring params. See "http-endpoint" on page 96.
monitoring-param	list	0..n	List of monitoring parameters. See "monitoring-param" on page 98
monitoring-param-ui-data	grouping	1	Grouping of monitoring parameters on the UI. See "monitoring-param-ui-data" on page 99
monitoring-param-value	grouping	1	

http-endpoint

List of http endpoints to be used by monitoring params.

ID	Type	Cardinality	Description
path	string	1	The HTTP path on the management server.
https	boolean	1	[Default *false*] Pick HTTPS instead of HTTP.
port	inet:port-number	1	HTTP port to connect to.

ID	Type	Cardinality	Description
username	string	1	HTTP basic auth user name.
password	string	1	HTTP basic auth password.
polling_interval_secs	uint8	1	[Default 2] HTTP polling interval in seconds.
method	enum	1	Method type of the HTTP operation: • GET (default) • POST • PUT • DELETE • PATCH • OPTIONS
data	string	1	Data to send in the POST body.
headers	list	0..n	List of custom HTTP headers to put on the HTTP request. See "http-endpoint:headers" on page 98.

http-endpoint:headers

List of custom HTTP headers to put on the HTTP request.

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

monitoring-param

List of monitoring parameters.

ID	Type	Cardinality	Description
id	string	1	[Key] Identifier for the monitoring parameter.
name	string	1	Name of the monitoring parameter.
http-endpoint-ref	leafref	1	Reference to the HTTP endpoint. "../../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 implementation to extract a value. • OBJECTPATH: Use objectpath implementation to extract a value.
json-query-params	container	1	Object for JSON query parameters. See "monitoring-param:json-query-params" on page 99.

monitoring-param:json-query-params

Object for JSON query parameters.

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.

monitoring-param-ui-data

Grouping of monitoring parameters on the UI.

ID	Type	Cardinality	Description
description	string	1	Description of the monitoring parameter.
group-tag	string	1	Tag to group monitoring parameters.
widget-type	enum	1	Type of the widget, typically used by the UI: • HISTOGRAM • BAR • GAUGE • SLIDER • COUNTER • TEXTBOX
units	string	1	Units for the monitoring parameter, such as megabits per second.

monitoring-param-value

ID	Type	Cardinality	Description
value-type	enum	1	Type of the parameter value: • INT (default) • DECIMAL • STRING
numeric-constraints	container	1	Constraints for the numbers. See "monitoring-param-value:numeric-constraints" on page 100.
text-constraints	container	1	Constraints for the text strings. See "monitoring-param-value:text-constraints" on page 101.
value-integer	int64	1	Current value for integer parameter.
value-decimal	decimal164	1	Current value for decimal parameter, up to 4 fraction digits.
value-string	string	1	Current value for the string parameter.

monitoring-param-value:numeric-constraints

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

monitoring-param-value: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.

See also

"VNFD Data Model (vnfd:vnfd)" on page 41

vnfd:placement-groups

List of placement groups at VNF level. The placement group construct defines the compute resource placement strategy in a cloud environment.

Fields

ID	Type	Cardinality	Description
name	string	1	Placement group name.
requirement	string	1	Describes the intent/rationale behind this placement group. **Note:** This free-text field is for human consumption only.
strategy	enum	1	Strategy associated with this placement group: • COLOCATION: [Default] Share the physical infrastructure, such as hypervisor/network, among all members of this group. • ISOLATION: Do not share the physical infrastructure among the members of this group
member-vdus	list	0..n	List of VDUs that are part of this placement group. See "member-vdus" on page 103.

member-vdus

List of VDUs that are part of this placement group

ID	Type	Cardinality	Description
member-vdu-ref	leafref	1	Reference to the VDU in the VNF. "../../../vdu/id"

See also

"VNFD Data Model (vnfd:vnfd)" on page 41

Virtual Link Descriptor (nsd:vld)

List of Virtual Link Descriptors (VLDs).

A virtual link descriptor (VLD) is a deployment template that describes the resource requirements needed for a link between VNFs, PNFs and endpoints of the network service, which could be met by various link options that are available in the NFVI.

The NFVO can select an option after evaluating the VNFFG to determine the appropriate NFVI to be used based on functional (e.g. dual separate paths for resilience) and other needs (e.g. geography and regulatory requirements).

Network connections are defined by connection points and **virtual links**. There are three types of connection points:

- Connect a network service to the outside world, such as the network service endpoint, described in the NSD
- Connect between VNFs within a network service, such as the external interface of the VNF, described in the VNFD
- Connect between VMs, described in the VNFC

There are also two types of virtual links:

- External virtual links, which can be connected to network service endpoints and external VNF interfaces
- Internal virtual links, which can be connected to external VNF interfaces and VNFCs

Virtual links also follow the Metro Ethernet Forum E-LINE, E-TREE, and E-LAN services. Virtual link descriptors (VLDs) contain the bandwidth and QoS requirements of the interconnection.

VLDs are required for a functioning NSD.

Fields

ID	Type	Cardinality	Description
id	string	1	[Key] Identifier for the VLD.
name	string	1	VLD name.
short-name	string	1	Short name to display as a label in the UI.
vendor	string	1	Provider of the VLD.

ID	Type	Cardinality	Description
description	string	1	Description of the VLD.
version	string	1	Version of the VLD.
type	enum	1	Type of the virtual link. Values can be: ELAN: A multipoint service connecting a set of VNFs.
root-bandwidth	uint64	1	Aggregate bandwidth for ELAN.
leaf-bandwidth	uint64	1	Bandwidth of branches for ELAN.
vnfd-connection-point-ref	list	0..n	A list of references to connection points. See "vnfd-connection-point-ref" on page 106.
virtual-connection-points	list	0..n	A list of virtual connection points associated with the virtual link. These connection points are not directly associated with VNFs. See "virtual-connection-points" on page 106.
provider-network	container	1	Container for the provider network. See "provider-network" on page 108.
mgmt-network	boolean	1	[Default *false*] Indicates whether this network is a VIM management network.
init-params	choice	1	Extra parameters for VLD instantiation. See "init-params" on page 109.

vnfd-connection-point-ref

References to connection points.

ID	Type	Cardinality	Description
vnf-id-ref	leafref	1	Reference to a VNFD. "../../../constituent-vnfd" + "[member-vnf-index = current()/../member-vnf-index-ref]" + "/vnfd-id-ref";
member-vnf-index-ref	leafref	1	[Key] Reference to member-vnf within constituent VNFDs. "../../../constituent-vnfd/member-vnf-index"
vnfd-connection-point-ref	leafref	1	[Key] Reference to a connection point name in a VNFD. This is a leafref to path: "/vnfd:vnfd-catalog/vnfd:vnfd" + "[vnfd:id = current()/../vnfd-id-ref]/" + "vnfd:connection-point/vnfd:name"

virtual-connection-points

List of virtual-connection points associated with the virtual link. These connection points are not directly associated with any VNFs.

ID	Type	Cardinality	Description
name	string	1	Name of the connection point.
id	string	1	Identifier for the internal connection points.

ID	Type	Cardinality	Description
short-name	string	1	Short name of the connection point to display as a label in the UI.
type	enum	1	Type of connection point: VPORT: Virtual Port
port-security-enabled	boolean	1	Enables or disables the port security for the connection-point. When set to *True*, the resource orchestrator passes the value to the VIM when the connection-point is created to filter traffic. **Note:** This value is supported on OpenStack only.
static-ip-address	inet:ip-address	1	Static IPv4 or IPv6 address for the internal/external connection point in the VNFD. When you instantiate a VNF, the static IP for the connection point is passed to the VIM.
associated-cps	list	0..n	List of connection points associated with virtual connection point. "../../vnfd-connection-point-ref/vnfd-connection-point-ref"

provider-network

Container for the 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	1	Identifies the type of the overlay network, which is a virtual network that is built on top of an existing network and is supported by its infrastructure. Supported values are: • LOCAL — Provider network implemented in a single compute node. • FLAT — Provider network shared by all tenants. • VLAN — Provider network implemented using 802.1Q tagging. • VXLAN — Provider networks implemented using RFC 7348. • GRE — Provider networks implemented using GRE tunnels.
segmentation-id	uint32	1	Segmentation ID.

init-params

Extra parameters for VLD instantiation

ID	Type	Cardinality	Description
vim-network-name	string	1	Name of network in VIM account. This is used to indicate pre-provisioned network name in cloud account.
ip-profile-ref	string	1	Named reference to IP-profile object.

See also

"Network Service Descriptor (nsd:nsd)" on page 11

VNF Forwarding Graph Descriptor (nsd:vnffgd)

A virtual network function forwarding graph (VNFFG) is a graph, specified by a network service provider, of bi-directional logical links that connect network function nodes, where at least one node is a VNF through which network traffic is directed.

The VNFFG model is defined at the network service level. One or more VNFFG descriptors can be defined in the network service descriptor (NSD).

A VNFFG model consists of a list of rendered service path (RSP) and list of classifier components.

Fields

ID	Type	Cardinality	Description
id	string	1	Unique identifier for the VNFFGD.
name	string	1	VNFFGD name.
short-name	string	1	VNFFGD short name to use as label in 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	Defines the ordered list of references to service functions (SF) that must be traversed as part of RSP. The SF reference exists in the form of references to connection-points of the constituent VNFDs, which are part of the NSD. See "rsp" on page 111.

ID	Type	Cardinality	Description
classifier	list	0..n	List of classifier rules for the VNFFGD. **Note:** A classifier is a network function that matches traffic flows against policy for subsequent application of the required set of network service functions. See "classifier" on page 113.

rsp

List of the Rendered Service Paths (RSP) for the VNFFGD.

ID	Type	Cardinality	Description
id	string	1	Unique identifier for the RSP.
name	string	1	RSP name.
vnfd-connection-point-ref	list	0..n	List of references to connection points. See "rsp:vnfd-connection-point-ref" on page 112.

rsp:vnfd-connection-point-ref

List of references to connection points.

ID	Type	Cardinality	Description
member-vnf-index-ref	leafref	1	Reference to member VNF within constituent VNFDs. "../../../../constituent-vnfd/member-vnf-index"
order	uint8	1	A number that denotes the VNF in a chain.
vnfd-id-ref	leafref	1	A reference to a VNFD. "../../../../constituent-vnfd" + "[member-vnf-index = current()/../member-vnf-index-ref]" + "/vnfd-id-ref"
vnfd-connection-point-ref	leafref	1	A reference to a connection point name in a VNFD. "/vnfd:vnfd-catalog/vnfd:vnfd" + "[vnfd:id = current()/../vnfd-id-ref]/" + "vnfd:connection-point/vnfd:name"

classifier

List of classifier rules for the VNFFGD.

ID	Type	Cardinality	Description
id	string	1	Unique identifier for the classifier rule.
name	string	1	Name of the classifier.
rsp-id-ref	leafref	1	A reference to the RSP defined within the VNFFG. The packets identified by the packet filters are steered through this RSP. "../../rsp/id"
member-vnf-index-ref	leafref	1	Reference to member-vnf within constituent-vndfs. "../../../constituent-vnfd/member-vnf-index"
vnfd-id-ref	leafref	1	A reference to a VNFD. "../../../constituent-vnfd" + "[member-vnf-index = current()/../member-vnf-index-ref]" + "/vnfd-id-ref"
vnfd-connection-point-ref	leafref	1	A reference to a connection point name in a VNFD. "/vnfd:vnfd-catalog/vnfd:vnfd" + "[vnfd:id = current()/../vnfd-id-ref]/" + "vnfd:connection-point/vnfd:name"
match-attributes	list	0..n	A list of packet filters that identifies the packet stream to be consumed by the RSP. See "classifier:match-attributes " on page 114

classifier:match-attributes

A list of packet filters that identifies the packet stream to be fed to the RSP.

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-address	1	Destination IP address.
source-port	inet:port-number	1	Source port number.
destination-port	inet:port-number	1	Destination port number.

Schema

```
grouping vnfd-descriptor {
      leaf id {
        description "Identifier for the VNFD.";
        type string;
      }

      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;
      }

      uses manotypes:vnf-configuration;

      uses config-parameter;

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

        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 "../../vdu/id";
              }
            }
          }

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

          list internal-connection-point {
            key "id-ref";
            description "List of internal connection points in this VLD";
            leaf id-ref {
              description "reference to the internal connection point id";
              type leafref {
                path "../../../vdu/internal-connection-point/id";
              }
            }
          }

          list virtual-connection-points {
            description
                "A list of virtual-connection points associated with Virtual
Link.
                 These connection points are not directly associated with any
VDUs";
            key name;
            uses common-connection-point;

            leaf-list associated-cps {
              description
                  "A List of connection points associated with virtual
connection point";
              type leafref {
                path "../../internal-connection-point/id-ref";
              }
            }
          }

          uses manotypes:provider-network;
          choice init-params {
            description "Extra parameters for VLD instantiation";

            case vim-network-ref {
              leaf vim-network-name {
                description
                    "Name of network in VIM account. This is used to indicate
                      pre-provisioned network name in cloud account.";
                type string;
```

```
            }
          }

          case vim-network-profile {
            leaf ip-profile-ref {
              description "Named reference to IP-profile object";
              type string;
            }
          }

        }
      }

      uses manotypes:ip-profile-list;

      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;

          list alarm {
            key "alarm-id";

            uses manotypes:alarm;
          }

          uses manotypes:image-properties;

          choice cloud-init-input {
            description
              "Indicates how the contents of cloud-init script are provided.
               There are 2 choices - inline or in a file";

            case inline {
              leaf cloud-init {
                description
                  "Contents of cloud-init script, provided inline, in cloud-
config format";
                type string;
              }
            }

            case filename {
              leaf cloud-init-file {
                description
                  "Name of file with contents of cloud-init script in cloud-
config format";
                type string;
              }
            }
          }

          uses manotypes:supplemental-boot-data;

          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;
      }

      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 volumes {
        key "name";

        leaf name {
          description "Name of the disk-volumes, e.g. vda, vdb etc";
          type string;
        }
```

```
          uses manotypes:volume-info;
        }
      }

      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 member-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";
            }
          }
        }
      }
```

See also

"Network Service Descriptor (nsd:nsd)" on page 11

MANO YANG Models

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

This section provides yang output of the models used by the MANO descriptors.

nsd.yang Model

The `nsd.yang` file defines the Network Service Descriptor (NSD), the top-level deployment of a network service. The nsd module contains attributes for a group of network functions, which together constitute a service definition. These attributes contain the relationship requirements of the VNFs chained together as a service. The NSD references one or more VNFDs, as well as other descriptors that are used for designing the service chains.

```
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";
  }

  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 parameter";
      type string;
    }
  }

  grouping nsd-descriptor {
    leaf id {
      description "Identifier for the NSD.";
      type string;
    }

    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
        "File path for  the vendor specific logo. For example
icons/mylogo.png.
         The logo  should be part of 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;
      }
    }

    /* Model limitation,
       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 vnfd-connection-point-ref";

        leaf member-vnf-index-ref {
          description "Reference to member-vnf within constituent-vnfds";
          type leafref {
            path "../../../constituent-vnfd/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 resolved this will switched to use
                leafref";
            type leafref {
              path "../../../constituent-vnfd" +
                   "[member-vnf-index = current()/../member-vnf-index-ref]" +
                   "/vnfd-id-ref";
            }
          }

          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 resolved this will switched to use
                  leafref";
            type string;
          }
        }

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

        leaf mgmt-network {
           description "Flag indicating whether this network is a VIM
management network";
           type boolean;
           default false;
        }

        choice init-params {
          description "Extra parameters for VLD instantiation";

          case vim-network-ref {
            leaf vim-network-name {
              description
                  "Name of network in VIM account. This is used to indicate
                     pre-provisioned network name in cloud account.";
              type string;
            }
          }

          case vim-network-profile {
            leaf ip-profile-ref {
              description "Named reference to IP-profile object";
              type string;
            }
          }

        }
      }
```

```
    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;
      }
    }


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

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

      list member-vnfd {
        description
            "List of VNFDs that are part of this placement 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 "../../../constituent-vnfd" +
                 "[member-vnf-index = current()/../member-vnf-index-ref]" +
                 "/vnfd-id-ref";
          }
        }
      }
    }

    uses manotypes:ip-profile-list;

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

    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 vnfd-connection-point-ref";

          leaf member-vnf-index-ref {
            description "Reference to member-vnf within constituent-vnfds";
            type leafref {
              path "../../../../constituent-vnfd/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 resolved this will switched to use
                    leafref";
                type leafref {
                   path "../../../../constituent-vnfd" +
                        "[member-vnf-index = current()/../member-vnf-index-
ref]" +
                        "/vnfd-id-ref";
                }
              }

              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 resolved this will switched to use
                    leafref";
                type leafref {
                  path "/vnfd:vnfd-catalog/vnfd:vnfd" +
                       "[vnfd:id = current()/../vnfd-id-ref]/" +
                       "vnfd:connection-point/vnfd:name";
                }
              }
            }
          } //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 "../../rsp/id";
              }
```

```
        }

        leaf member-vnf-index-ref {
          description "Reference to member-vnf within constituent-vnfds";
          type leafref {
            path "../../../constituent-vnfd/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 resolved this will switched to use
                  leafref";
          type leafref {
              path "../../../constituent-vnfd" +
                   "[member-vnf-index = current()/../member-vnf-index-ref]"
+
                   "/vnfd-id-ref";
          }
        }

        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 resolved this will switched to use
                  leafref";
          type leafref {
              path "/vnfd:vnfd-catalog/vnfd:vnfd" +
                   "[vnfd:id = current()/../vnfd-id-ref]/" +
                   "vnfd:connection-point/vnfd:name";
          }
        }

        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;
          }
        } //match-attributes
      } // classifier
    } // vnffgd

    list monitoring-param {
      description
        "List of monitoring parameters from VNFs that should be
        propagated up into NSR";
      key "id";

      leaf id {
        type string;
      }

      leaf name {
        type string;
      }

      uses manotypes:monitoring-param-value;
      uses manotypes:monitoring-param-ui-data;
      uses manotypes:monitoring-param-aggregation;

      list vnfd-monitoring-param {
```

```
        description "A list of VNFD monitoring params";
        key "member-vnf-index-ref vnfd-monitoring-param-ref";

        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 resolved this will switched to use
              leafref";
          type leafref {
            path "../../../constituent-vnfd" +
                 "[member-vnf-index = current()/../member-vnf-index-ref]" +
                 "/vnfd-id-ref";
          }
        }

        leaf vnfd-monitoring-param-ref {
          description "A reference to the VNFD monitoring param";
          type leafref {
            path "/vnfd:vnfd-catalog/vnfd:vnfd"
              + "[vnfd:id = current()/../vnfd-id-ref]"
              + "/vnfd:monitoring-param/vnfd:id";
          }
        }

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

    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;
        }
      }
    }

    uses manotypes:ns-service-primitive;

    list initial-config-primitive {
      rwpb:msg-new NsdInitialConfigPrimitive;
      description
        "Initial set of configuration primitives for NSD.";
      key "seq";

      uses manotypes:initial-config;
    }



  container nsd-catalog {

    list nsd {
      key "id";

      uses nsd-descriptor;
    }
  }

}
```

vnfd.yang Model

The `vnfd.yang` file defines the Virtual Network Function (VNF) attributes. The vnfd module defines VNF platform resource requirements, such as CPU, memory, interfaces, and network. It also contains special characteristics related to EPA attributes and performance capabilities and connectivity, interface, and KPI requirements that can be used to establish virtual links within the NFVI between its Virtual Network Function Component (VNFC) instances, or between a VNF instance and the endpoint interface to the other network functions.

```
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;
    }
    leaf port-security-enabled {
      description "Enables the port security for the port";
      type boolean;
    }

    leaf static-ip-address {
      description "Static IP address for the connection point";
      type inet:ip-address;
    }
  }

  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.
             E1000           : Emulate E1000 interface.
             RTL8139         : Emulate RTL8139 interface.
             PCNET           : Emulate PCNET interface.
             OM-MGMT         : Used to specify openmano mgmt external-
connection type";

        type enumeration {
          enum OM-MGMT;
          enum PCI-PASSTHROUGH;
          enum SR-IOV;
          enum VIRTIO;
          enum E1000;
          enum RTL8139;
          enum PCNET;
        }
        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;
      }
    }
  }



  grouping vnfd-descriptor {
      leaf id {
        description "Identifier for the VNFD.";
        type string;
      }

      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;
      }

      uses manotypes:vnf-configuration;

      uses config-parameter;

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

        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 "../../vdu/id";
              }
            }
          }

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

        list internal-connection-point {
          key "id-ref";
          description "List of internal connection points in this VLD";
          leaf id-ref {
            description "reference to the internal connection point id";
            type leafref {
              path "../../../vdu/internal-connection-point/id";
            }
          }
        }
```

```
          list virtual-connection-points {
            description
                "A list of virtual-connection points associated with Virtual
Link.
                These connection points are not directly associated with any
VDUs";
            key name;
            uses common-connection-point;

            leaf-list associated-cps {
              description
                  "A List of connection points associated with virtual
connection point";
              type leafref {
                path "../../internal-connection-point/id-ref";
              }
            }
          }

          uses manotypes:provider-network;
          choice init-params {
            description "Extra parameters for VLD instantiation";

            case vim-network-ref {
              leaf vim-network-name {
                description
                    "Name of network in VIM account. This is used to indicate
                      pre-provisioned network name in cloud account.";
                type string;
              }
            }

            case vim-network-profile {
              leaf ip-profile-ref {
                description "Named reference to IP-profile object";
                type string;
              }
            }

          }
        }

        uses manotypes:ip-profile-list;

        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;

      list alarm {
        key "alarm-id";

        uses manotypes:alarm;
      }

      uses manotypes:image-properties;

      choice cloud-init-input {
        description
          "Indicates how the contents of cloud-init script are provided.
           There are 2 choices - inline or in a file";
```

```
            case inline {
              leaf cloud-init {
                description
                  "Contents of cloud-init script, provided inline, in cloud-
config format";
                type string;
              }
            }

            case filename {
              leaf cloud-init-file {
                description
                  "Name of file with contents of cloud-init script in cloud-
config format";
                type string;
              }
            }
          }

          uses manotypes:supplemental-boot-data;

          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;
          }

          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 volumes {
          key "name";

          leaf name {
            description "Name of the disk-volumes, e.g. vda, vdb etc";
            type string;
          }

          uses manotypes:volume-info;
        }
      }

      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 mandatory 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 member-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";
              }
            }
          }
        }
    }

    container vnfd-catalog {
      description
          "Virtual Network Function Descriptor (VNFD).";

      list vnfd {
        key "id";

        uses vnfd-descriptor;
      }
    }
  }
```

mano-types.yang Model

The `mano-types.yang` file defines the common types and definitions used by both network and VNF descriptors, shown in "nsd.yang Model" on page 125 and "vnfd.yang Model" on page 139, respectively.

```
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 package-type {
       description "Type of descriptor being on-boarded";
       type enumeration {
         enum NSD;
         enum VNFD;
       }
     }

   typedef parameter-data-type {
     type enumeration {
       enum STRING;
       enum INTEGER;
       enum BOOLEAN;
     }
   }

   grouping primitive-parameter-value {
     list parameter {
       description
           "List of parameters to the configuration primitive.";
       key "name";
       leaf name {
         description
             "Name of the parameter.";
         type string;
       }
```

```
      leaf value {
        description
            "Value associated with the name.";
        type string;
      }
    }
  }

  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 parameter";
      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 event-config {
    leaf seq {
      description
```

```
          "Sequence number for the configuration primitive.";
      type uint64;
    }

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

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

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

      leaf value {
        type string;
      }
    }
  }

  grouping image-properties {
    leaf image {
      description
            "Image name for the software image.
             If the image name is found within the VNF package 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.
            ";
      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;
    }
  }

  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
          "Miscellaneous input parameters to be considered
           while processing the NSD to apply configuration";

      leaf config-priority {
        description
            "Configuration priority - order of configuration
             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 service-primitive {
        rwpb:msg-new ServicePrimitive;
        description
          "List of service primitives supported by the
          configuration agent for this VNF.";
        key "name";

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

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

      list initial-config-primitive {
        rwpb:msg-new InitialConfigPrimitive;
        description
            "Initial set of configuration primitives.";
        key "seq";
        uses event-config;
      }

      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 implementation to extract a value.
       OBJECTPATH - Use objectpath implementation 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: Physical 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;
        enum REQUIRE_VME;
        enum PREFER_VME;
        enum REQUIRE_DE;
        enum PREFER_DE;
        enum REQUIRE_PSE;
        enum PREFER_PSE;
        enum REQUIRE_TSC;
        enum PREFER_TSC;
        enum REQUIRE_MSR;
        enum PREFER_MSR;
        enum REQUIRE_PAE;
        enum PREFER_PAE;
        enum REQUIRE_MCE;
        enum PREFER_MCE;
        enum REQUIRE_CX8;
        enum PREFER_CX8;
        enum REQUIRE_APIC;
        enum PREFER_APIC;
        enum REQUIRE_SEP;
        enum PREFER_SEP;
        enum REQUIRE_MTRR;
        enum PREFER_MTRR;
        enum REQUIRE_PGE;
        enum PREFER_PGE;
        enum REQUIRE_MCA;
        enum PREFER_MCA;
        enum REQUIRE_CMOV;
        enum PREFER_CMOV;
        enum REQUIRE_PAT;
        enum PREFER_PAT;
        enum REQUIRE_PSE36;
        enum PREFER_PSE36;
        enum REQUIRE_CLFLUSH;
        enum PREFER_CLFLUSH;
        enum REQUIRE_DTS;
        enum PREFER_DTS;
        enum REQUIRE_ACPI;
        enum PREFER_ACPI;
```

```
        enum REQUIRE_MMX;
        enum PREFER_MMX;
        enum REQUIRE_FXSR;
        enum PREFER_FXSR;
        enum REQUIRE_SSE;
        enum PREFER_SSE;
        enum REQUIRE_SSE2;
        enum PREFER_SSE2;
        enum REQUIRE_SS;
        enum PREFER_SS;
        enum REQUIRE_HT;
        enum PREFER_HT;
        enum REQUIRE_TM;
        enum PREFER_TM;
        enum REQUIRE_IA64;
        enum PREFER_IA64;
        enum REQUIRE_PBE;
        enum PREFER_PBE;
        enum REQUIRE_RDTSCP;
        enum PREFER_RDTSCP;
        enum REQUIRE_PNI;
        enum PREFER_PNI;
        enum REQUIRE_PCLMULQDQ;
        enum PREFER_PCLMULQDQ;
        enum REQUIRE_DTES64;
        enum PREFER_DTES64;
        enum REQUIRE_MONITOR;
        enum PREFER_MONITOR;
        enum REQUIRE_DS_CPL;
        enum PREFER_DS_CPL;
        enum REQUIRE_VMX;
        enum PREFER_VMX;
        enum REQUIRE_SMX;
        enum PREFER_SMX;
        enum REQUIRE_EST;
        enum PREFER_EST;
        enum REQUIRE_TM2;
        enum PREFER_TM2;
        enum REQUIRE_SSSE3;
        enum PREFER_SSSE3;
        enum REQUIRE_CID;
        enum PREFER_CID;
        enum REQUIRE_FMA;
        enum PREFER_FMA;
        enum REQUIRE_CX16;
        enum PREFER_CX16;
        enum REQUIRE_XTPR;
        enum PREFER_XTPR;
        enum REQUIRE_PDCM;
        enum PREFER_PDCM;
        enum REQUIRE_PCID;
        enum PREFER_PCID;
        enum REQUIRE_DCA;
        enum PREFER_DCA;
        enum REQUIRE_SSE4_1;
        enum PREFER_SSE4_1;
```

```
      enum REQUIRE_SSE4_2;
      enum PREFER_SSE4_2;
      enum REQUIRE_X2APIC;
      enum PREFER_X2APIC;
      enum REQUIRE_MOVBE;
      enum PREFER_MOVBE;
      enum REQUIRE_POPCNT;
      enum PREFER_POPCNT;
      enum REQUIRE_TSC_DEADLINE_TIMER;
      enum PREFER_TSC_DEADLINE_TIMER;
      enum REQUIRE_XSAVE;
      enum PREFER_XSAVE;
      enum REQUIRE_AVX;
      enum PREFER_AVX;
      enum REQUIRE_F16C;
      enum PREFER_F16C;
      enum REQUIRE_RDRAND;
      enum PREFER_RDRAND;
      enum REQUIRE_FSGSBASE;
      enum PREFER_FSGSBASE;
      enum REQUIRE_BMI1;
      enum PREFER_BMI1;
      enum REQUIRE_HLE;
      enum PREFER_HLE;
      enum REQUIRE_AVX2;
      enum PREFER_AVX2;
      enum REQUIRE_SMEP;
      enum PREFER_SMEP;
      enum REQUIRE_BMI2;
      enum PREFER_BMI2;
      enum REQUIRE_ERMS;
      enum PREFER_ERMS;
      enum REQUIRE_INVPCID;
      enum PREFER_INVPCID;
      enum REQUIRE_RTM;
      enum PREFER_RTM;
      enum REQUIRE_MPX;
      enum PREFER_MPX;
      enum REQUIRE_RDSEED;
      enum PREFER_RDSEED;
      enum REQUIRE_ADX;
      enum PREFER_ADX;
      enum REQUIRE_SMAP;
      enum PREFER_SMAP;
    }
  }

  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 uint64;
      }

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

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

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


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

      list om-cpu-feature {
        key "feature";
        description "List of openmano CPU features";
        leaf feature {
          description "CPU feature";
          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 prefers 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-unaware {
          leaf numa-unaware {
            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
                               preferentially 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;
              }

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

              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 physical 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 ns-service-primitive {
        list service-primitive {
          description
```

```
          "Network service level service primitives.";

      key "name";

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

      list parameter {
        description
            "List of parameters for the service 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 for the service 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 service 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 resolved 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;
    }
  }
}

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;
        }
      }

      uses monitoring-param-ui-data;
      uses monitoring-param-value;

    }
  }

  grouping monitoring-param-aggregation {
    typedef aggregation-type {
      description "aggregation-type";
      type enumeration {
        enum AVERAGE;
        enum MINIMUM;
        enum MAXIMUM;
        enum COUNT;
        enum SUM;
      }
    }

    leaf aggregation-type {
      type aggregation-type;
    }
  }

  grouping monitoring-param-ui-data {
      leaf description {
        type string;
      }
```

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


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

      leaf units {
        type string;
      }
  }

  grouping monitoring-param-value {
      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
              "Maximum 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;
      }
    }

    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
              "Maximum 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 specifies 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 agency 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;
    }

    leaf vdur-id {
      description
          "The identifier of the VDUR that the alarm is associated with";
      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 order 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;
    }
  }

  typedef cloud-account-type {
```

```
    description "cloud account type";
    type enumeration {
      enum aws;
      enum cloudsim;
      enum cloudsim_proxy;
      enum mock;
      enum openmano;
      enum openstack;
      enum vsphere;
      enum openvim;
      enum prop_cloud1;
    }
  }

  grouping host-aggregate {
    list host-aggregate {
      description "Name of the Host Aggregate";
      key "metadata-key";

      leaf metadata-key {
        type string;
      }
      leaf metadata-value {
        type string;
      }
    }
  }

  grouping placement-group-input {
    leaf cloud-type {
      type manotypes:cloud-account-type;
    }
    choice cloud-provider {
      case openstack {
        container availability-zone {
          description "Name of the Availability Zone";
          leaf name {
            type string;
          }
        }
        container server-group {
          description "Name of the Affinity/Anti-Affinity Server Group";
          leaf name {
            type string;
          }
        }
        uses host-aggregate;
      }
      case aws {
        leaf aws-construct {
          type empty;
        }
      }
      case openmano {
        leaf openmano-construct {
          type empty;
```

```
        }
      }
      case vsphere {
        leaf vsphere-construct {
          type empty;
        }
      }
      case mock {
        leaf mock-construct {
          type empty;
        }
      }
      case cloudsim {
        leaf cloudsim-construct {
          type empty;
        }
      }
    }
  }

  grouping cloud-config {
    list key-pair {
      key "name";
      description "Used to configure the list of public keys to be injected
as part
          of ns instantiation";

      leaf name {
        description "Name of this key pair";
        type string;
      }

      leaf key {
        description "Key associated with this key pair";
        type string;
      }
    }

    list user {
      rwpb:msg-new CloudConfigUser;
      key "name";
      description "List of users to be added through cloud-config";

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

      leaf user-info {
        description "The user name's real name";
        type string;
      }

      list key-pair {
        key "name";
        description "Used to configure the list of public keys to be
```

```
injected as part
             of ns instantiation";

         leaf name {
           description "Name of this key pair";
           type string;
         }

         leaf key {
           description "Key associated with this key pair";
           type string;
         }
       }
     }
   }

   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";
     }
```

```
   }

   grouping ip-profile-info {
     description "Grouping for IP-Profile";
     container ip-profile-params {

       leaf ip-version {
         type inet:ip-version;
         default ipv4;
       }

       leaf subnet-address {
         description "Subnet IP prefix associated with IP Profile";
         type inet:ip-prefix;
       }

       leaf gateway-address {
         description "IP Address of the default gateway associated with IP
Profile";
         type inet:ip-address;
       }

       leaf security-group {
         description "Name of the security group";
         type string;
       }

       list dns-server {
         key "address";
         leaf address {
                                description "List of DNS Servers associated
with IP Profile";
                                type inet:ip-address;
         }
       }

       container dhcp-params {
         leaf enabled {
           description "This flag indicates if DHCP is enabled or not";
           type boolean;
           default true;
         }

         leaf start-address {
           description "Start IP address of the IP-Address range associated
with DHCP domain";
           type inet:ip-address;
         }

         leaf count {
           description "Size of the DHCP pool associated with DHCP domain";
           type uint32;
         }
       }

       leaf subnet-prefix-pool {
```

```
        description "VIM Specific reference to pre-created subnet prefix";
        type string;
      }
    }
  }

  grouping ip-profile-list {
    list ip-profiles {
      description
          "List of IP Profiles.
             IP Profile describes the IP characteristics for the Virtual-
Link";

      key "name";

      leaf name {
        description "Name of the IP-Profile";
        type string;
      }

      leaf description {
        description "Description for IP profile";
        type string;
      }

      uses ip-profile-info;
    }
  }

  grouping volume-info {
    description "Grouping for Volume-info";

    leaf description {
      description "Description for Volume";
      type string;
    }

    leaf size {
      description "Size of disk in GB";
      type uint64;
    }

    choice volume-source {
      description
            "Defines the source of the volume. Possible options are
             1. Ephemeral -- Empty disk
             2. Image     -- Refer to image to be used for volume
             3. Volume    -- Reference of pre-existing volume to be used
            ";

      case ephemeral {
        leaf ephemeral {
          type empty;
        }
      }
```

```
       case image {
         uses image-properties;
       }

       case volume {
         leaf volume-ref {
           description "Reference for pre-existing volume in VIM";
           type string;
         }
       }
     }

     container guest-params {
       description "Guest virtualization parameter associated with volume";

       leaf device_bus {
         description "Type of disk-bus on which this disk is exposed to
guest";
         type enumeration {
           enum ide;
           enum usb;
           enum virtio;
           enum scsi;
         }
       }

       leaf device_type {
         description "The type of device as exposed to guest";
         type enumeration {
           enum disk;
           enum cdrom;
           enum floppy;
           enum lun;
         }
       }
     }
   }

 }
```