Oracle® Communications Network Service Orchestration

Implementation Guide

Release 7.3.4

E67130-01

September 2016

ORACLE

Oracle Communications Network Service Orchestration Implementation Guide, Release 7.3.4

E67130-01

Contents

Preface vii

Audience vii
Related Documentation vii
Documentation Accessibility viii

1 Overview

About Network Service Orchestration 1-1
Network Service Orchestration Components 1-2
About Network Service Orchestration Entities 1-2
About the UIM User Interface 1-4
About UIM Help 1-5
About the Sample Network Services 1-5

2 Setting Up Network Service Orchestration

Planning Your Implementation 2-1
Software Requirements 2-1
Installing and Integrating the Network Service Orchestration Components 2-2
Integrating Network Service Orchestration With Northbound Applications for Asynchronous Communication 2-2
Integrating the VIM with Network Service Orchestration 2-3
Registering the VIM 2-3
Discovering VIM Resources 2-4
Registering the SDN Controller 2-5
Setting Network Service Orchestration Properties 2-5
Enabling Logging for Network Service Orchestration 2-6
Supported Southbound Integration 2-6

3 Designing and Onboarding Network Services, VNFs, and PNFs

About Design Components 3-1
About Descriptor Files 3-1
About Network Service Descriptor Files 3-2
About VNF Descriptor Files 3-8
About PNF Descriptor Files 3-9
Creating a Descriptor File 3-10
About Technical Actions Files 3-11

Creating a Technical Actions File ... 3-12
About VNF Configuration Files ... 3-13
Setting Network Service Descriptor Properties ... 3-14
Onboarding VNFs Using TOSCA VNF Descriptor Templates ... 3-15
Sample TOSCA VNF Descriptor Template ... 3-16
Installing Python ... 3-17
Importing the TOSCA VNFD Template into Design Studio ... 3-18
Tagging Network Service Orchestration Specifications ... 3-19
Designing Custom Network Services ... 3-19
Creating Cartridges for VNFs ... 3-20
Logical Device Specification ... 3-20
Service Specification ... 3-21
Service Configuration Specification ... 3-23
Capability Service ... 3-23
Capability Service Configuration Specification ... 3-24
Creating Cartridges for PNFs ... 3-25
Logical Device Specification ... 3-25
Service Specification ... 3-26
Service Configuration Specification ... 3-27
Creating Cartridges for Network Services ... 3-28
Network Service Specification ... 3-28
Network Service Configuration Specification ... 3-30

4 Working with Network Services, VNFs, and PNFs

Instantiating Network Services ... 4-1
Modifying Network Services ... 4-3
Adding VNFs to Network Services ... 4-3
Removing VNFs from Network Services ... 4-3
Terminating Network Services ... 4-4
Scaling VNFs ... 4-4
Monitoring and Healing VNFs ... 4-4
About the Monitoring Tabs in the User Interface ... 4-5
Upgrading the Software Version of a VNF ... 4-5
Working with PNFs in Network Services ... 4-6
Retrieving Details About Network Services, VNFs, PNFs, and Descriptors ... 4-6

5 Implementing the Sample Network Services

Configuring the Juniper vSRX Base Image ... 5-1
Implementing the Network Protection Service ... 5-3
Implementing the Residential Gateway Network Service ... 5-7

6 Extending Network Service Orchestration

Setting Up Design Studio for Extending Network Service Orchestration ... 6-1
Using Extension Points and Java Interface Extensions to Extend Network Service Orchestration ... 6-2
Writing a Custom Ruleset Extension Point ... 6-2

Using Java Interface Extensions 6-3
Implementing a Custom SDN Controller 6-4
Implementing a Custom Monitoring Engine 6-5
Implementing a Custom VIM 6-7
Implementing a Custom VNF Life Cycle Manager 6-8
Implementing an Adapter for a Custom VNF Manager 6-9
Implementing a Custom VNF Connection Manager 6-10
Implementing a Custom VNF Configuration Manager 6-11
Implementing a Custom Response Manager 6-12
Localizing Network Service Orchestration 6-13
Localizing the Responses in RESTful APIs 6-13

7 Network Service Orchestration RESTful API Reference

About the Network Service Orchestration RESTful APIs 7-1
Network Service Orchestration RESTful API Resources 7-2
RESTful API Responses 7-3
Sample Requests and Responses 7-5
Register VIM 7-5
Discover VIM Resources 7-6
Update VIM 7-7
Get VIM Details 7-8
Instantiate Network Service 7-9
Scale Network Service 7-13
Get Network Services 7-14
Get Network Service Details 7-14
Get Network Service VNFs 7-15
Get Network Service Networks 7-17
Get Network Service End Points 7-18
Get Network Service Status 7-19
Terminate Network Service 7-20
Add VNF to Network Service 7-21
Terminate VNF in a Network Service 7-22
Heal VNF 7-23
Configure VNF Capabilities 7-25
Upgrade VNF 7-26
Get VNF Details 7-27
Get VNF Status 7-28
Register PNF 7-29
Update PNF 7-30
Get PNFs 7-30
Get PNF Details 7-32
Unregister PNF 7-32
Register EMS 7-33
Update EMS 7-33
Get EMSs 7-34
Get EMS Details 7-35
Unregister EMS 7-36

Get Network Service Descriptors 7-36
Get Network Service Descriptor Details 7-36
Get Network Service Descriptor VNFDs 7-38
Get Network Service Descriptor Flavors 7-38
Get VNF Descriptor Details 7-39
Get VNF Descriptor Versions 7-40
Get VNF Descriptor Flavors 7-41

Preface

This guide explains how to implement and use Oracle Communications Network Service Orchestration.

Audience

This document is intended for:

- Network operations and management personnel who install, configure, and maintain physical and virtual network infrastructure
- Data modelers who define specifications for entities that represent Virtual Network Functions (VNFs), network services, and other related and dependant items in the inventory
- Engineers who model resources in Design Studio
- Systems integrators who implement and integrate Oracle Communications Unified Inventory Management (UIM) and third-party software as part of Network Service Orchestration

The guide assumes that you have a working knowledge of UIM and Network Functions Virtualization (NFV) architecture and concepts.

Related Documentation

For step-by-step instructions to perform tasks, log in to each application to see the following:

- UIM Help: Provides step-by-step instructions for tasks you perform in UIM.
- Design Studio Help: Provides step-by-step instructions for tasks you perform in Design Studio.

For more information, see the following documentation:

- *UIM Installation Guide*: Describes the requirements for installing UIM, installation procedures, and post-installation tasks.
- *UIM System Administrator's Guide*: Describes administrative tasks such as working with cartridge packs, maintaining security, managing the database, configuring Oracle Map Viewer, and troubleshooting.
- *Design Studio Installation Guide*: Describes the requirements for installing Design Studio, installation procedures, and post-installation tasks.
- *UIM Security Guide*: Provides guidelines and recommendations for setting up UIM in a secure configuration.

- *UIM Concepts*: Provides an overview of important concepts and an introduction to using both UIM and Design Studio.
- *UIM Developer's Guide*: Explains how to customize and extend many aspects of UIM, including the data model, life-cycle management, topology, security, rulesets, user interface, and localization.
- *Design Studio Developer's Guide*: Describes how to customize, extend, and work with cartridges.
- *UIM Web Services Developer's Guide*: Describes the UIM Web Service operations and how to use them, and describes how to create custom Web services.
- *UIM Information Model Reference*: Describes the UIM information model entities and data attributes, and explains patterns that are common across all entities. This document is available on the Oracle Software Delivery Cloud as part of the Oracle Communications Unified Inventory Management Developer Documentation package.
- *Oracle Communications Information Model Reference*: Describes the Oracle Communications information model entities and data attributes, and explains patterns that are common across all entities. The information described in this reference is common across all Oracle Communications products. This document is available on the Oracle Software Delivery Cloud as part of the Oracle Communications Unified Inventory Management Developer Documentation package.
- *UIM Cartridge Guide*: Provides information about how you use cartridge packs with UIM. Describes the content of the base cartridges.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

1

Overview

This chapter provides an overview of Oracle Communications Network Service Orchestration.

About Network Service Orchestration

Oracle Communications Network Service Orchestration is a functional module of Oracle Communications Unified Inventory Management (UIM). You use Network Service Orchestration to model network services, Virtual Network Functions (VNFs), and Physical Network Functions (PNFs). You also use Network Service Orchestration to manage the life cycles of network services and VNFs.

Network Service Orchestration enables you to create, implement, and manage the life cycles of network services and deploy the network services as interconnected VNFs and PNFs on virtual resources.

Network Service Orchestration provides the following functionality:

- **Onboarding of Network Services, VNFs, and PNFs**. You can define network services, VNFs, and PNFs based on any network function that you want to virtualize. See "Designing and Onboarding Network Services, VNFs, and PNFs" for more information.
- **Instantiation, Scaling, and Termination of Network Services**. You can quickly instantiate, scale, and terminate VNFs and network services in response to demand on your network. You can manage the life cycles of your VNFs and network services and control the resources that they use. See "Working with Network Services, VNFs, and PNFs" for more information.

  Network Service Orchestration supports asynchronous communication with northbound applications. See "Integrating Network Service Orchestration With Northbound Applications for Asynchronous Communication" for more information.
- **Monitoring and Auto-healing**. You can monitor the performance of VNFs continuously and configure Network Service Orchestration to heal a failed VNF automatically. See "Monitoring and Healing VNFs" for more information about monitoring and healing a VNF.
- **Resource Orchestration**. Network Service Orchestration manages the resources across your data centers to ensure that each network service is allocated the required resources to meet the needs of the VNFs. See "Working with Network Services, VNFs, and PNFs" for more information.
- **Customization and Extension**. You can customize and extend Network Service Orchestration to support integration with third-party VNF Managers, Virtualized

Infrastructure Managers (VIMs), software-defined networking (SDN) controllers, and monitoring engines. Network Service Orchestration also provides extension points that enable you to customize and extend its core functionality. See "Extending Network Service Orchestration" for more information.

Network Service Orchestration Components

Network Service Orchestration builds on Oracle Communications Unified Inventory Management (UIM), taking advantage of its inventory and workflow capabilities to perform run-time orchestration of Network Functions Virtualization (NFV) environments, including virtual, physical, and hybrid networks.

Oracle Communications Design Studio provides the design-time environment for onboarding VNFs and composing network services. Network Service Orchestration is extensible and allows integration with third-party VNF managers, VIMs, monitoring engines, and SDN Controllers.

Network Service Orchestration includes a VNF Manager that enables you to manage the life cycles of the VNFs. Network Service Orchestration also supports integration with Oracle and third-party VNF Managers, VIMs, SDN controllers, and network monitoring applications. By default, Network Service Orchestration provides integration to certain applications and supports integration to additional applications during the implementation.

Network Service Orchestration provides RESTful APIs, which communicate over HTTP and HTTPS, to interact and exchange data between various components.

About Network Service Orchestration Entities

Network Service Orchestration uses the Oracle Communications Information Model (OCIM) to represent inventory items and business practices. The Oracle Communications Information Model is based on the Shared Information Data (SID) model developed by the TeleManagement Forum. The information model contains resource entities, service entities, common patterns, definitions, and common business entities.

For details about the Oracle Communications Information Model (OCIM), see *Oracle Communications Information Model Reference* and *UIM Information Model Reference*.

Table 1–1 describes the NFV entities and their corresponding OCIM entities.

Table 1–1 Mapping of NFV Entities and OCIM Entities

NFV Entity	OCIM Entity	Description
Availability Zone	Custom Object with characteristics.	Represents a grouping of resources based on availability characteristics, for example Availability Zone (OpenStack), Resource Pool (VMware). In OpenStack, availability zones enable you to arrange OpenStack compute hosts into logical groups and provides a form of physical isolation and redundancy from other availability zones, such as by using a separate power supply or network equipment.
Connection Point	Device Interface	Represents a port on the VNF. Connection points connect Virtual Links to VNFs. They represent the virtual interfaces and physical interfaces of the VNFs and their associated properties and other metadata
Deployment Flavor	Custom Object	Represents a specific deployment of a network service or VNF supporting specific key performance indicators (KPIs), such as capacity and performance.
Element Management System (EMS)	Custom Object	Represents the Element Management System, which performs the typical management functionality for one or several VNFs.
Endpoint	Custom Object	Describes a service access point for the network service.
Flavor	Custom Object	Defines the compute, memory, and storage capacity of computing instances. A flavor is an available hardware configuration for a server. It defines the size of a virtual server that can be launched.
Host	Custom Object with characteristics.	Represents a compute host, a physical host dedicated to running compute nodes.
Infrastructure Domain	Network Address Domain	Represents the domain within the NFV Infrastructure that includes all networking that interconnects compute and storage infrastructure.
IP Network Infrastructure	■ Network Address Domain ■ IP Network ■ IP Subnet ■ IP Address	Represents the network, subnet, and IP address of the VNF in Network Service Orchestration. The networks are either created or referenced in the service configuration. During activation, the corresponding network, subnet, and ports are created in the VIM on which the VNF virtual machine is deployed.
IP Address	IP Address	Represents an IPv4Address and an IPv6Address in the OCIM domain model.
Network Functions Virtualization Infrastructure (NFVI)	Custom object	Represents the totality of all hardware and software components that build the environment where VNFs are deployed. NFVI can span across several locations.
Network Service	Service	Represents a composition of network functions.
Network Service Descriptor	■ Service Specification ■ Service Config Version Specification	Describes a network service in terms of its deployment and operational behavior. Used in the process of network service on-boarding and managing the life cycle of a network service instance.
Orchestration Request	Business Interaction	Represents an NFV life-cycle action in UIM. Every time you perform a life-cycle action, Network Service Orchestration creates a business interaction for the action in UIM.

Table 1–1 (Cont.) Mapping of NFV Entities and OCIM Entities

NFV Entity	OCIM Entity	Description
Physical Network Function (PNF)	Logical Device Service	Represents an implementation of a network function that is a tightly-coupled hardware and software system. A network function is a functional building block within a network infrastructure that has well-defined external interfaces and a well-defined functional behavior.
PNF Descriptor	■ Logical Device Specification ■ Service Specification ■ Service Config Version Specification	Describes a PNF in terms of its deployment and operational behavior. The PNF Descriptor is used for onboarding PNFs.
SDN Controller	Custom Object	Centralizes some or all of the control and management functionality of a network domain. An SDN controller can also provide an abstract view of its domain to other functional components through well-defined interfaces.
Subnet	IP Subnet	Represents an administrative or functional boundary on a range of network addresses. A subnet is defined by a base range whose sequence is often appended to a fixed prefix.
Virtual Data Center (VDC)	Custom Object with characteristics.	Represents the resources managed by a VIM under a specific tenant (for example, OpenStack) or Organization Virtual Data Center (VMware).
Virtual Link	IP Network	Describes the basic topology of connectivity between VNFs and target parameters, such as bandwidth, latency, and QoS. Virtual links connect to VNFs using Connection Points (CPs).
Virtual Network Function (VNF)	■ Logical Device ■ Logical Device Service ■ Service Config Version	Represents an implementation of a network function that can be deployed on a Network Function Virtualization Infrastructure (NFVI). A network function is a functional building block within a network infrastructure that has well-defined external interfaces and a well-defined functional behavior.
Virtualized Infrastructure Manager (VIM)	Custom Object with characteristics.	Represents a functional component that is responsible for controlling and managing the NFVI compute, storage and network resources, usually within an operator's infrastructure domain.
VNF Descriptor	■ Logical Device Specification ■ Service Specification ■ Service Config Version Specification	Describes a VNF in terms of its deployment and operational behavior. The VNF Descriptor is used in the process of VNF onboarding and managing the life cycle of a VNF instance.

About the UIM User Interface

The UIM user interface provides a group of links and pages for performing network service and VNF life cycle operations and for managing your data center resources.

The UIM user interface displays the **Network Service Orchestration** group in the navigation section that includes the following expandable and collapsible subgroups of links:

- In the **Orchestration** subgroup:

  - **Orchestration Requests**. Clicking this link displays the **Search** page for orchestration requests. From the Search page, you can create new orchestration requests. The Search page also returns service requests that are created based on your NFV service request specifications.
  - **Network Services**. Clicking this link displays the **Search** page for network services. From the Search page, you can create new network services. The Search page also returns a list of network services that are created based on your network service descriptors.
  - **Virtual Network Functions**. Clicking this link displays the **Search** page for VNFs. The search page returns a list of VNFs that are created based on your VNF descriptors.
- In the **Catalog** group:
  - **Network Service Descriptors**. Clicking this link displays the **Search** page for Network Service descriptors. From the Search page, you can create and instantiate new network services. The search page also returns a list of network service descriptors.
  - **VNF Descriptors**. Clicking this link displays the **Search** page for VNF descriptors. The search page returns a list of VNF descriptors.

For more information about the user interface, see "UIM User Interface Overview" in *UIM Concepts*. See UIM Help for instructions about performing tasks related to network services, VNFs, and PNFs.

About UIM Help

UIM includes a Help system that you use to get step-by-step instructions. You can find the information you need by searching or by navigating through the table of contents. See "Using the UIM Help" in *UIM Concepts* for more information about the UIM Help system.

About the Sample Network Services

Network Service Orchestration includes the following sample cartridges that you can use as references for designing and implementing your own network services:

- **Juniper_vSRX_VNF**. This sample cartridge contains the Juniper vSRX firewall VNF to use with the network protection service.
- **Checkpoint_NG_FW_VNF**. This sample cartridge contains the Checkpoint firewall VNF to use with the network protection service.
- **Cisco_xRV_PNF**. This sample cartridge contains the Cisco XRV router PNF to use with the residential gateway network service.
- **NPaaS_NetworkService**. This sample cartridge contains the functionality to implement network protection as a service.
- **ResidentialGateway_NetworkService**. This sample cartridge contains the functionality to implement a residential gateway service.

See "Implementing the Sample Network Services" for detailed information about the sample network services.

2

Setting Up Network Service Orchestration

This chapter describes the instructions for setting up Oracle Communications Network Service Orchestration.

Planning Your Implementation

Before you implement Network Service Orchestration, you must identify the required software, ensure that the required network infrastructure is available and ready, and identify the third-party software that you want to use. Your choices are based on the network services you want to deliver on your network.

Use the following list of tasks as a checklist to ensure that you have all the required components for a successful implementation of Network Service Orchestration:

- Install and integrate the Network Service Orchestration components. See "Software Requirements" and "Installing and Integrating the Network Service Orchestration Components".
- Integrate your Virtual Infrastructure Manager (VIM). See "Integrating the VIM with Network Service Orchestration".
- Integrate the SDN controller if your network service requires configuration of network flows. See "Registering the SDN Controller".
- Onboard Network Services and VNFs. See "Designing and Onboarding Network Services, VNFs, and PNFs".
- Write extensions for extending the core functionality and integrate third-party software with Network Service Orchestration. See "Using Extension Points and Java Interface Extensions to Extend Network Service Orchestration".
- Integrate client applications with Network Service Orchestration for using the RESTful APIs. For details about the RESTful APIs, see "Network Service Orchestration RESTful API Reference".

Software Requirements

To implement Network Service Orchestration, you require the following software:

- Oracle Communications Unified Inventory Management 7.3.4.

  See *UIM Installation Guide* for installation instructions.
- Oracle Communications Design Studio 7.3.4.1

  See *Design Studio Installation Guide* for installation instructions.

Installing and Integrating the Network Service Orchestration Components

To install and integrate the Network Service Orchestration components:

1. Install UIM on a WebLogic server. See *UIM Installation Guide* for installation instructions.
2. Navigate to the *UIM_Home*/**cartridges/base** directory and deploy the following UIM cartridges into UIM in the order they are listed:
   - ora_uim_baseextpts
   - ora_uim_basemeasurements
   - ora_uim_basetechnologies
   - ora_uim_basespecifications
   - ora_uim_baserulesets
   - OracleComs_NSO_BaseCartridge

   See *UIM Cartridge Guide* for instructions about deploying cartridges into UIM.
3. (Optional) If you want to use the sample cartridges that are provided with Network Service Orchestration, navigate to the *UIM_Home*/**cartridges**/**sample** directory and deploy the sample cartridges into UIM.

   Note: Before deploying the sample cartridges, deploy the ora_uim_common cartridge.

   See "About the Sample Network Services" for more information about the sample cartridges provided with Network Service Orchestration.

   See "Implementing the Sample Network Services" for information about implementing the sample network services.
4. (Optional) Integrate Network Service Orchestration with northbound applications for asynchronous communication. See "Integrating Network Service Orchestration With Northbound Applications for Asynchronous Communication".
5. Integrate the VIM with Network Service Orchestration. See "Integrating the VIM with Network Service Orchestration" for more information.
6. (Optional) Integrate the SDN controller with Network Service Orchestration. See "Registering the SDN Controller" for more information.

Integrating Network Service Orchestration With Northbound Applications for Asynchronous Communication

Some VNF and network service life cycle operations perform long-running processes. Network Service Orchestration supports integration with northbound applications in asynchronous communication for such life cycle operations.

With this integration, Network Service Orchestration provides the final and actual status of the following network service life-cycle actions so that northbound systems can perform and complete service fulfillment:

- Instantiate a network service
- Terminate a network service

- Add one or more VNFs to network service
- Delete one or more VNFs from a network service
- Scale a VNF
- Replace a VNF
- Upgrade the software version of a VNF

To integrate Network Service Orchestration with northbound systems for asynchronous communication:

1. Configure your client applications to subscribe to the **NSOResponseTopic** topic in the WebLogic server. During installation, UIM creates the JMS Module and **NSOResponseTopic**.

   You can implement a custom response topic and configure your applications to subscribe to it. See "Implementing a Custom Response Manager" for more information about implementing a custom response topic.

2. On the WebLogic server, in the JMS Module, create a Durable Subscriber to capture the messages.
3. Open the *UIM_Home*/**config**/**nso.properties** file and uncomment the following property:

   ```
   #nso.ResponseManager.list.1=oracle.communications.inventory.nso.client.vnfm.NSO
   ResponseTopicImpl
   ```

Integrating the VIM with Network Service Orchestration

Network Service Orchestration supports OpenStack and provides integration points for integrating other third-party VIMs. See "Implementing a Custom VIM" for more information about implementing a custom VIM.

Before you integrate the VIM with Network Service Orchestration, ensure that you set up and configure the VIM to use with Network Service Orchestration. After your VIM infrastructure is set up, register the VIM and discover the VIM resources into Network Service Orchestration.

> **Note:** If you use multiple VIMs, register all of them with Network Service Orchestration and discover resources. In the properties file of the network service descriptor, specify the VIM that you want to use by default.

Integrating the VIM with Network Service Orchestration involves the following tasks:

- Registering the VIM
- Discovering VIM Resources

Registering the VIM

To register a VIM with Network Service Orchestration:

1. Ensure that UIM is started and running.
2. Ensure that the required Network Service Orchestration base cartridges are deployed into UIM.

3. Ensure that the VIM is running and that you have the IP address, username, password, and other details of the VIM instance.
4. In a RESTful API client, call the following RESTful API using the POST method:

   POST http://*nso_host*:*port*/ocnso/1.1/vim

   where:

   - *nso_host* is the IP address of the machine on which UIM is installed
   - *port* is the port number of the machine on which UIM is installed
5. Specify the VIM details in the request. For details about the request parameters, see "Register VIM".

   The RESTful API client returns a response.
6. In UIM, verify that a custom object with the details of the VIM is created.

Discovering VIM Resources

You discover VIM resources into UIM so that Network Service Orchestration contains information about the current status and availability of all the required virtual resources on the network. In UIM, VIMs are represented as custom objects.

When you discover VIM resources, the details of the following resources are populated into UIM:

- Availability zone (OpenStack)
- Flavor
- Host
- Networks and Subnets

To discover VIM resources into UIM:

1. In a RESTful API client, call the following RESTful API using the POST method:

   POST http://*nso_host*:*port*/ocnso/1.1/vim/*vimId*/discovery?infoLevel=*vim_information*

   where:

   - *nso_host* is the IP address or the domain name of the machine on which UIM is installed
   - *port* is the port number of the machine on which UIM is installed
   - *vimId* is the Id of the VIM that you registered with Network Service Orchestration and whose resources you want to discover
   - *vim_information* is the level of information about the VIM that you want to retrieve and view in the response. The available values are:
     - **summary**. Retrieves and displays a summary of the VIM resources.
     - **details**. Retrieves and displays complete details about all the VIM resources.

   For more details about the request parameters, see "Discover VIM Resources".

   The RESTful API client returns a response.
2. In UIM, verify that the following entities are created:
   - Availability zone

- Flavor
- Host
- VDC
- Network address domains
- IP subnets

Note: Whenever you add, upgrade, modify, or delete the compute, memory, and network resources in your NFV Infrastructure (NFVI), run the VIM discovery RESTful API to ensure that details about the currently available resources on your NFVI are reflected correctly in Network Service Orchestration.

Registering the SDN Controller

If your network service requires implementation of network flows, then you can set up Network Service Orchestration to use an SDN controller. SDN controllers are based on protocols, such as OpenFlow, that enable servers to instruct switches where to send network traffic. You should register the SDN controller with Network Service Orchestration to manage flow control in the network. Network Service Orchestration supports OpenDaylight and provides integration points for integrating other third-party SDN controllers. See "Implementing a Custom SDN Controller" for more information about implementing a custom SDN controller.

To register your SDN controller with Network Service Orchestration:

1. In UIM, create a custom object based on the SDN specification and specify the following details about the SDN controller that you want to use:
   - Host
   - Port number
   - Username of the SDN controller
   - Password of the SDN controller
   - Type of the SDN controller
2. Associate the VIM custom object as a parent custom object to the SDN controller custom object.

Setting Network Service Orchestration Properties

Network Service Orchestration provides the *UIM_Home*/**config**/**nso.properties** file that you use to specify properties for your implementation of Network Service Orchestration.

To set the properties, open the **nso.properties** file in a text editor and update the following parameters:

- **startIpAddress**. Specify the subnet start IP address. By default, when Network Service Orchestration creates a network, the subnet IP address starts with 192.168.0.0.
- **NSO_HOST**: *IPv4address*

where *IPv4address* is the host on which UIM is installed. By default, Network Service Orchestration considers the host on which the UIM server is running. If the server is running on a private network that is unavailable to external network, specify a reachable IP address for the server.

- **NSO_USERNAME**: *username*

  where *username* is the username of the UIM user.

- **NSO_PASSWORD**: *password*

  where *password* is the encrypted password of the UIM user.

  To encrypt the password:

  1. Create a text file and type the password.
  2. Save and close the file.
  3. In UIM, in the **Administration** group of the navigation section, click **Execute Rulesets**.
  4. In the **Ruleset** list, select the **EncryptText** ruleset, and enter the path and file name of the text file that contains the password in plain text and click **Process**.

     UIM displays the encrypted password. Copy the encrypted password and specify it in the **nso.properties** file.

Enabling Logging for Network Service Orchestration

You enable logging for Network Service Orchestration to log debug messages.

For more information about logging, see the chapter about improving UIM performance in *UIM System Administrator's Guide*.

To enable logging for Network Service Orchestration:

1. Open the *UIM_Home*/**config/loggingconfig.xml** file in a text editor.
2. Add the following text:

```
<logger name="oracle.communications.inventory.nso" additivity="false">
        <level value="debug" />
        <appender-ref ref="stdout"/>
        <appender-ref ref="rollingFile"/>
</logger>
```

3. Save and close the file.

Supported Southbound Integration

Network Service Orchestration supports some integrations by default, while others require customization. Network Service Orchestration supports the following southbound integrations:

- For Virtual Infrastructure Management:
  - Integration to OpenStack Mitaka and Liberty releases
  - Integration to Oracle OpenStack for Oracle Linux Release 2
  - Sample integration to VMware vCloud Director
  - A framework for integration to other Virtual Infrastructure Managers
- For Network and SDN controllers:

  - Integration to OpenStack Neutron (Mitaka and Liberty releases)
  - Sample integration to OpenDaylight
- For VNF Management:
  - Network Service Orchestration includes a built-in VNF Manager that supports the lifecycle management of VNFs. The VNF manager calls the VIM to perform life-cycle actions. The in-built VNF Manager supports direct integration to the VNF or integration to an Element Management System (EMS) that manages the VNF.
  - A framework that supports integration to external VNF Managers

3

Designing and Onboarding Network Services, VNFs, and PNFs

This chapter provides information about designing and onboarding network services, VNFs, and PNFs.

About Design Components

Design components are files that you create in Oracle Communications Design Studio. Network Service Orchestration uses different types of files that you create in Design Studio to describe the behavior of your network services, VNFs, and PNFs.

- **Descriptor files**. The descriptor files describe the attributes of the VNF, PNF, and Network Service specifications.

  See "About Descriptor Files" for more information about the descriptor files.

- **Technical actions files**. The technical actions files describe the actions for the VNFs, PNFs, and network services in the VIM. There is one technical actions file for each network service, VNF, and PNF.

  See "About Technical Actions Files" for more information about the technical actions files.

- **Configuration and template files**. The configuration files contain the configuration and post-configuration details for the VNFs.

  See "About VNF Configuration Files" for more information about the descriptor files.

- **Entity Specifications**. In Design Studio, you create entity specifications that you use to create instances of VNFs and network services in Network Service Orchestration. You package the entity specifications into cartridges. You create one cartridge for each VNF, network service, and PNF. See "Designing Custom Network Services" for more information.

  See Design Studio Help for information about creating entity specifications in Design Studio.

- **Custom extensions**. See "Extending Network Service Orchestration" for information about implementing custom extensions.

About Descriptor Files

Descriptor files contain metadata about network services, VNFs, and PNFs. Network Service Orchestration defines descriptors as Design Studio specifications and uses these specifications to manage the life cycles of network services and VNFs.

Descriptors describe the behavior of virtual network functions that are defined in the Network Service Orchestration cartridges. There is one descriptor file for each network service, VNF, and PNF.

About Network Service Descriptor Files

Network Service descriptor files describe the deployment requirements, operational behavior, and policies required by network services based on them. You use one descriptor file for each network service.

When you instantiate, scale, or terminate a network service, Network Service Orchestration deploys, scales, and undeploys the constituent VNFs based on the parameters and policies specified in the descriptor file.

You use network service descriptor files for the following purposes:

- Defining the networks by either creating them or by referencing existing networks and specifying network types. See "Describing Networks" for more information.
- For each network, specifying the VNF connection points that the network service should use.
- Specifying the network forwarding path for the network traffic. See "Describing Forwarding Graphs" for more information.
- For each VNF in the network service, specifying parameters related to CPU utilization and when you want Network Service Orchestration to heal a VNF and scale the network service. See "Describing Deployment Flavors" for more information.

Describing Networks

In the network service descriptor file, you define networks by creating them or by referencing existing networks and specifying their types. You represent networks as virtual links. You can create or reference any number of networks based on your service requirements. You can also specify the number of end points the networks can have.

The following text shows the parameters that you specify for a virtual link descriptor in a network service descriptor XML file:

```
<virtualLinkDescriptors>
 <virtualLinkDescriptor name="network_name" isReferenced="value_reference"
isDHCPEnabled="value_DHCP" type="network_type">
   <numberOfEndPoints>number_of_endpoints</numberOfEndPoints>
   <connectionPoints>
    <!-- The format is VNFD:ConnectionPoint -->
    <connectionPoint name="vnf_descriptor_name:connection_point_name"
type="connection_point_type" order="connection_point_order" isExternal="value_
external">
     <extensions>
      <portSecurity enabled="value_port_security"/>
      <!-- port-level security groups -->
      <securityGroup name="name_security_group"/>
      <securityGroup name="name_security_group"/>
      <securityGroup name="name_security_group"/>
     </extensions>
   </connectionPoint>
 </connectionPoints>
  <extensions>
   <providerNetwork name="name_provider_network"/>
   <externalRouter name="name_external_router"/>
```

```
  </extensions>
 </virtualLinkDescriptor>
</virtualLinkDescriptors>
```

Table 3–1 describes the parameters you specify for a virtual link descriptor in the network service descriptor XML file.

Table 3–1 Virtual Link Descriptor Parameters

Parameter	Description
network_name	Specify a unique name of the network that you want to create or reference.
value_reference	Specify whether you want to create the network or reference an existing network in run-time. Specify **true** if you want to reference an existing network. Otherwise, specify **false**. If you reference an existing network, specify the networks using the following key in your network service descriptor properties file: *vim_Id.network_service_descriptor.network_name* See "Setting Network Service Descriptor Properties" for more information about setting parameters in the network service descriptor properties file.
value_DHCP	Specify whether the network you create or reference should support DHCP or not. Specify **true** if the network should support DHCP. Otherwise, specify **false**.
network_type	Specify the type of the network that you want to create or reference. For example, specify **MANAGEMENT** for the management network and **DATA** for the packet-in network.
number_of_endpoints	Specify the number of endpoints that the network provides. Network Service Orchestration determines the subnet prefix to be created within the network.
vnf_descriptor_ name:connection_point_ name	Specify the name of the VNF descriptor XML file and the name of the VNF connection point.
connection_point_type	Specify the type of the connection point. For example, specify **IN** or **OUT**.
connection_point_order	Specify the order of the connection point for the VNF. The order of the connection points is determined by the VNF vendor.
value_external	Specify whether this connection point requires access to external traffic or not. Specify **true** if this connection point requires access to external traffic to ensure that Network Service Orchestration allocates a floating IP address.
value_port_security	Specify whether ports should be created with security enabled or disabled. Specify **true** to enable security for the ports. Otherwise, specify **false**. By default, this field is set to **false**.
name_security_group	Specify the name of the security group. Security groups are configured in OpenStack. If the security requirements for each connection point in the VNF are different, define separate security groups for each security requirement and specify the security group at the port level.

Table 3–1 (Cont.) Virtual Link Descriptor Parameters

Parameter	Description

name_provider_network	Specify the name of the external network that is configured in your VIM. Specify a value for this parameter if the **isExternal** parameter in the **connectionPoint** element is set to **true**.
	If you use multiple VIMs, add and specify multiple instances of the following parameter in your network service descriptor properties file:
	`vim_Id.network_service_descriptor.name_provider_network`
	For example, if you implement the Residential Gateway network service using OpenStack Mitaka, specify:
	`OpenStackMitaka.ResidentialGateway_NSD.ext-net = publicNet`

name_external_router	Specify the name of the router that is configured in your VIM and connects to the external network. Specify a value for this if the **isExternal** tag in the **connectionPoint** parameter is set to **true**.
	If you use multiple VIMs, add and specify multiple instances of the following parameter in your network service properties file:
	`vim_Id.network_service_descriptor.name_external_router`
	For example, if you implement the Residential Gateway network service using OpenStack Mitaka, specify:
	`OpenStackMitaka.ResidentialGateway_NSD.AccessExternal = publicRouter`

The following text shows a virtual link descriptor element in the **NPaaS_NSD.xml** sample network service descriptor file:

```
<virtualLinkDescriptors>
 <virtualLinkDescriptor name="Data_IN" type="DATA" isDHCPEnabled="false"
isReferenced="false">
   <numberOfEndPoints>20</numberOfEndPoints>
    <connectionPoints>
      <!-- The format is VNFD:ConnectionPoint -->
      <connectionPoint name="Juniper_vSRX_VNFD:CP01" type="IN" order="2"/>
      <connectionPoint name="Checkpoint_NG_FW_VNFD:CP01" type="IN" order="1"/>
      <connectionPoint name="Oracle_VNS_VNFD:CP01" type="IN" order="3"/>
   </connectionPoints>
 </virtualLinkDescriptor>
</virtualLinkDescriptors>
```

The following text shows a virtual link descriptor element in the **ResidentialGateway_NSD.xml** sample network service descriptor file:

```
<virtualLinkDescriptor name="Data1" isReferenced="false" isDHCPEnabled="false"
type="DATA">
   <numberOfEndPoints>2</numberOfEndPoints>
   <connectionPoints>
    <!-- The format is VNFD:ConnectionPoint -->
    <connectionPoint name="Cisco_xRV_PNFD:CP02" order="2" type="OUT"/>
    <connectionPoint name="Juniper_vSRX_VNFD:CP02" order="2" type="IN"
isExternal="true">
    <extensions>
      <portSecurity enabled="true" />
      <!-- port-level security groups -->
      <securityGroup name="open" />
     </extensions>
    </connectionPoint>
```

```
     </connectionPoints>
     <!-- extensions is for FloatingIP Assignment-->
     <extensions>
    <providerNetwork name="ext-net" />
    <externalRouter name="AccessExternal" />
   </extensions>
  </virtualLinkDescriptor>
```

Describing Forwarding Graphs

In the network service descriptor file, you also describe forwarding graphs by specifying the network forwarding path for the network traffic.

The following text shows the pattern in which you describe a forwarding graph in a network service descriptor file:

```
<forwardingGraphDescriptors>
  <forwardingGraphDescriptor name="name_forwarding_graph" default="default">
   <networkForwardingPath>
    <vnfd name="name_vnf_descriptor">
     <connectionPoints>
      <connectionPoint name="name_connection_point" type="type_connectionPoint"/>
     <connectionPoint name="name_connection_point" type="type_connectionPoint"/>
    </connectionPoints>
   </vnfd>
  </networkForwardingPath>
 </forwardingGraphDescriptor>
</forwardingGraphDescriptors>
```

where:

- **name_forwarding_graph** is the unique name of the forwarding graph.
- **default** indicates if the network service should use this forwarding graph by default or not.
- **name_vnf_descriptor** is the name of the VNF descriptor that you want to use with the network service.
- **name_connection_point** is the name of the connection point defined in the VNF descriptor, that you want to use for the forwarding graph.
- **type_connectionPoint** is the type of the connection point.

The following text shows a forwarding graph element in the **NPaaS_NSD.xml** sample network service descriptor file:

```
<forwardingGraphDescriptors>
 <forwardingGraphDescriptor name="Data" default="true">
  <networkForwardingPath>
   <vnfd name="Checkpoint_NG_FW_VNFD">
    <connectionPoints>
     <connectionPoint name="CP01" type="IN"/>
     <connectionPoint name="CP02" type="OUT"/>
    </connectionPoints>
   </vnfd>
  </networkForwardingPath>
 </forwardingGraphDescriptor>
</forwardingGraphDescriptors>
```

Describing Deployment Flavors

In the network service descriptor file, you also describe deployment flavors where in you specify constituent VNFs and PNFs. You also define assurance parameters for various factors, such as CPU utilization of the virtual machine on which the VNFs are deployed. You also specify when you want to heal a VNF and scale the network service.

The following text shows the pattern in which you describe deployment flavors in a network service descriptor file:

```
<serviceDeploymentFlavors>
 <serviceDeploymentFlavor name="name_flavor">
  <constituentPNFDs>
    <pnf>
      <pnfd name="name_PNFD" />
    </pnf>
  </constituentPNFDs>
  <constituentVNFDs>
   <vnf>
    <vnfd name="name_VNFD"/>
     <assuranceParameters>
      <assuranceParameter name="name_assurance_parameter" action="action">
       <id>Id</id>
       <value>value</value>
       <condition>condition</condition>
      </assuranceParameter>
      <assuranceParameter name="name_assurance_parameter" action="action">
       <id>Id</id>
       <value>value</value>
       <condition>condition</condition>
     </assuranceParameter>
    </assuranceParameters>
   -<extensions>
      <!-- instance-level security groups -->
      <securityGroup name="name_security_group"/>
      <securityGroup name="name_security_group"/>
    </extensions>
   </vnf>
  </constituentVNFDs>
 </serviceDeploymentFlavor>
</serviceDeploymentFlavors>
```

Table 3–2 describes the parameters you specify for a service deployment flavor in the network service descriptor XML file.

Table 3–2 Service Deployment Flavor Parameters

Parameter	Description
name_flavor	Specify a unique name for the service deployment flavor.
name_PNFD	Specify the name of the PNF descriptor.
name_VNFD	Specify the name of the VNF descriptor.
name_assurance_parameter	Specify the name of the assurance parameter. For example specify **Low CPU Utilization** and **High CPU Utilization** to define actions for the CPU utilization parameter.
action	Specify the action that you want Network Service Orchestration to perform on the VNF. For example, specify **heal** for the low CPU utilization assurance parameter and **scale** for the high CPU utilization assurance parameter.

Table 3–2 (Cont.) Service Deployment Flavor Parameters

Parameter	Description
Id	Specify the Id of the assurance parameter.
value	Specify the threshold value for the assurance parameter.
condition	Specify the condition based on which the defined action, either **heal** or **scale**, should be performed.
name_security_group	Specify the name of the security group. Security groups are configured in OpenStack. If security groups are not specified at the connection point level, this parameter enables security groups at the VNF level, provided that the **isEnabled** parameter is set to **true** at the connection point level and the security group is specified. If the security requirements are common for all the connection points of a VNF, define a common security group and use this group.

The following text shows a service deployment flavor element in the **NPaaS_NSD.xml** sample network service descriptor file:

```
<serviceDeploymentFlavors>
 <serviceDeploymentFlavor name="Checkpoint">
  <constituentVNFDs>
   <vnf>
    <vnfd name="Checkpoint_NG_FW_VNFD"/>
     <assuranceParameters>
      <assuranceParameter name="Low CPU Utilization" action="heal">
       <id>cpu_util</id>
       <value>0.0</value>
       <condition>eq</condition>
      </assuranceParameter>
      <assuranceParameter name="High CPU Utilization" action="scale">
       <id>cpu_util</id>
       <value>80.0</value>
       <condition>gt</condition>
      </assuranceParameter>
     </assuranceParameters>
   </vnf>
  </constituentVNFDs>
 </serviceDeploymentFlavor>
</serviceDeploymentFlavors>
```

The following text shows a service deployment flavor element in the **ResidentialGateway_NSD.xml** sample network service descriptor file that includes a PNF:

```
<serviceDeploymentFlavors>
 <serviceDeploymentFlavor name="RGW">
  <constituentPNFDs>
   <pnf>
    <pnfd name="Cisco_xRV_PNFD"/>
   </pnf>
  </constituentPNFDs>
  <constituentVNFDs>
   <vnf>
    <vnfd name="Juniper_vSRX_VNFD"/>
     <assuranceParameters>
      <assuranceParameter name="Low CPU Utilization" action="heal">
       <id>cpu_util</id>
```

```
       <value>0.0</value>
       <condition>eq</condition>
      </assuranceParameter>
      <assuranceParameter name="High CPU Utilization" action="scale">
       <id>cpu_util</id>
       <value>80.0</value>
       <condition>gt</condition>
      </assuranceParameter>
     </assuranceParameters>
    -<extensions>
     <!-- instance-level security groups -->
     <securityGroup name="ALL"/>
     <securityGroup name="default"/>
     </extensions>
   </vnf>
  </constituentVNFDs>
 </serviceDeploymentFlavor>
</serviceDeploymentFlavors>
```

About VNF Descriptor Files

VNF descriptor files describe the deployment requirements, operational behavior, and policies required by VNFs that are based on them.

Network Service Orchestration includes the following sample VNF descriptor files:

- **Juniper_vSRX_VNFD.xml**. This is the descriptor file for the Juniper vSRX firewall VNF.
- **Checkpoint_NG_FW_VNFD.xml**. This is the descriptor file for the Checkpoint NG firewall VNF.

In the VNF descriptor file, you specify:

- Deployment flavor parameters
- Connection points for the VNF
- The software version of the VNF

The following text shows the pattern in which you describe a VNF in the VNF descriptor file:

```
<vnfd name="name_VNFD">
 <deploymentFlavors>
  <deploymentFlavor name="name_deploymentFlavor" disk="diskSpace" memory="memory"
vcpus="vcpus"/>
  <deploymentFlavor name="name_deploymentFlavor" disk="diskSpace" memory="memory"
vcpus="vcpus"/>
  </deploymentFlavors>
   <connectionPoints>
    <connectionPoint name="name_ConnectionPoint"/>
    <connectionPoint name="name_ConnectionPoint"/>
    <connectionPoint name="name_ConnectionPoint"/>
   </connectionPoints>
   <defaultDeploymentFlavor>name_default_
deploymentFlavor</defaultDeploymentFlavor>
  <versions>
   <version imagePasswd="" imageUserName="" imageName="imageName"
number="versionNumber"/>
   <version imagePasswd="" imageUserName="" imageName="imageName"
number="versionNumber"/>
```

```
  </versions>
</vnfd>
```

where:

- **name_VNFD** is the name of the VNF Descriptor.
- **name_deploymentFlavor** is the unique name for the VNF deployment flavor.
- **diskSpace** is the disk space that you want to allocate for the VNF. Specify the disk space in GB.
- **memory** is the memory you want to allocate for the VNF. Specify the memory in GB.
- **vcpus** is the number of virtual CPUs that you want to allocate for the VNF.
- **name_ConnectionPoint** is the unique name of the connection point.
- **name_default_deploymentFlavor** is the name of the deployment flavor that you want to use for the VNF by default.
- **imageName** is the name of the VNF image.
- **versionNumber** is the unique version number for the VNF image.

The following text shows the elements in the **Juniper_vSRX_VNFD.xml** sample VNF descriptor file:

```
<vnfd name="Juniper_vSRX_VNFD">
 <deploymentFlavors>
  <deploymentFlavor name="vsrx.medium" disk="20" memory="4" vcpus="2"/>
   <deploymentFlavor name="m1.medium" disk="40" memory="4" vcpus="2"/>
  </deploymentFlavors>
  <connectionPoints>
    <connectionPoint name="CP01"/>
    <connectionPoint name="CP02"/>
    <connectionPoint name="CP03"/>
   </connectionPoints>
   <defaultDeploymentFlavor>vsrx.medium</defaultDeploymentFlavor>
  <versions>
   <version imagePasswd="" imageUserName=""
imageName="vsrx-12.1X47-D20.7-npaas-v0.3" number="1.0"/>
   <version imagePasswd="" imageUserName=""
imageName="vsrx-12.1X47-D20.7-rgw-v0.3" number="2.0"/>
  </versions>
</vnfd>
```

About PNF Descriptor Files

PNF descriptor files describe the deployment requirements, operational behavior, and policies required by PNFs that are based on them.

In the PNF descriptor file, you specify:

- Vendor details
- Connection points for the PNF
- The software version of the PNF

The following text shows the pattern in which you describe a PNF in the PNF descriptor file:

```
<pnfd vendor="name_PNF_vendor" name="name_PNFD">
```

```
 <connectionPoints>
  <connectionPoint name="name_ConnectionPoint"/>
  <connectionPoint name="name_ConnectionPoint"/>
  <connectionPoint name="name_ConnectionPoint"/>
 </connectionPoints>
 <versions>
  <version number="version"/>
 </versions>
</pnfd>
```

where:

- **name_PNF_vendor** is the name of the PNF vendor.
- **name_PNFD** is the name of the PNF descriptor.
- **name_ConnectionPoint** is the unique name for the connection point.
- **version** is the unique version number of the PNF.

Network Service Orchestration includes the following sample PNF descriptor file:

- **Cisco_xRV_PNFD.xml**. This descriptor file can be used for the Cisco xRV router PNF.

The following text shows the elements in the **Cisco_xRV_PNFD.xml** sample PNF descriptor file:

```
<pnfd vendor="CISCO" name="Cisco_xRV_PNFD">
 <connectionPoints>
  <connectionPoint name="CP01"/>
  <connectionPoint name="CP02"/>
  <connectionPoint name="CP03"/>
  </connectionPoints>
 <versions>
  <version number="1.0"/>
 </versions>
</pnfd>
```

Creating a Descriptor File

In Design Studio, you create a descriptor file for each Network Service specification and VNF Service specification.

To create a descriptor file:

1. In Design Studio, import all the Network Service Orchestration cartridges. See "Setting Up Design Studio for Extending Network Service Orchestration" for more information about importing the cartridges into Design Studio.
2. Switch to the Navigator view.
3. In the root directory of the cartridge project, create the following folder structure:

   model/content/product_home/config

4. Right-click on the **config** folder and create an XML file with the name *ServiceSpecificationName*.**xml** for a network service, *LogicalDeviceSpecificationName*.**xml** for a VNF, and *LogicalDeviceSpecificationName*.**xml** for a PNF.

   where:

   - *ServiceSpecificationName* is the name of the service specification

- *LogicalDeviceSpecificationName* is the name of the logical device specification, which represents a VNF or a PNF.

5. Copy the sample content from the sample cartridge project to the XML file and modify it according to your service requirements.

About Technical Actions Files

Technical actions files describe the actions for the VNFs, PNFs, and network services in a VIM. There is one technical actions file for each network service, VNF, and PNF.

In the technical actions file, for each technical action, you define the following elements:

- **action**: This element declares a technical action, its signature (which contains the name and type of each parameter), and the type of its subject and target.
- **match**: This element declares configuration differences that match an XPath expression.
- **generator**: This element defines all the bindings of the configuration to the parameters, subject, and target of the action to be generated.

The following example shows the elements in the technical actions file:

```
<technicalActionCalculator
    xmlns="http://xmlns.oracle.com/communications/inventory/actioncalculator"

xmlns:invactcalc="http://xmlns.oracle.com/communications/inventory/actioncalculato
r"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.oracle.com/communications/inventory/actioncalcula
tor schemas/TechnicalActionCalculator.xsd">
    <invactcalc:action>
        <name>DEPLOY_VNF</name>
        <actionCode>DEPLOY_VNF</actionCode>
        <subject>
            <class>LogicalDevice</class>
        </subject>
        <target>
            <class>LogicalDevice</class>
        </target>
        <parameter>
            <name>serviceID</name>
            <type>string</type>
        </parameter>
        <parameter>
            <name>vnfID</name>
            <type>string</type>
        </parameter>
        <parameter>
            <name>vnfName</name>
            <type>string</type>
        </parameter>
        <parameter>
            <name>vnfdName</name>
            <type>string</type>
        </parameter>
        <parameter>
            <name>imageName</name>
            <type>string</type>
```

```
        </parameter>
    </invactcalc:action>

    <invactcalc:match>
        <invactcalc:diff>
            <invactcalc:path>/root/after/vnf/Assignment[@State='PENDING_ASSIGN'
                and /root/service[state!='PENDING_
DISCONNECT']]/..</invactcalc:path>
        </invactcalc:diff>
        <invactcalc:action>DEPLOY_VNF</invactcalc:action>
        <invactcalc:anchor>.</invactcalc:anchor>
    </invactcalc:match>

    <invactcalc:generator>
        <invactcalc:action>DEPLOY_VNF</invactcalc:action>
        <invactcalc:condition>/root/after/vnf/Assignment[@State='PENDING_
ASSIGN']</invactcalc:condition>
        <subject>.</subject>
        <target>.</target>
        <binding>
            <parameter>serviceID</parameter>
            <path>/root/service/id</path>
        </binding>
        <binding>
            <parameter>vnfID</parameter>
            <path>Assignment/id</path>
        </binding>
        <binding>
            <parameter>vnfName</parameter>
            <path>Assignment/name</path>
        </binding>
        <binding>
            <parameter>vnfdName</parameter>
            <path>Assignment/specification</path>
        </binding>
        <binding>
            <parameter>imageName</parameter>
            <path>Assignment/imageName</path>
        </binding>
    </invactcalc:generator>
</technicalActionCalculator>
```

Creating a Technical Actions File

In Design Studio, you create a technical actions file for each Network Service specification, a VNF Service specification, and a PNF specification.

To create a technical actions file:

1. In Design Studio, switch to the Navigator view.
2. In the root directory of the cartridge project, create the following folder structure:

   model/content/product_home/config
3. Right-click on the **config** folder and create an XML file with the name *ServiceSpecificationName*_**TechnicalActions.xml**, where *ServiceSpecificationName* is the name of the service specification.
4. Copy the sample content from the sample cartridge project to the XML file and modify it according to your service requirements.

About VNF Configuration Files

Depending on the functionality that they deliver, some VNFs in a network service may require configuration after they are deployed. After a VNF is deployed, you can configure the VNF based on its configuration requirements.

> **Note:** Post-deployment configuration of VNFs is not always required.

To configure a VNF, Network Service Orchestration requires the following configuration files to be created:

- *VNFD_Name***Template.conf**

  This is a VNF-specific configuration template in which you specify the placeholder fields for instance-specific parameters.

- *VNFD_Name***Config.xml**

  This is a configuration file in which you specify the VNF instance-specific configuration parameter values as name-value pairs.

Network Service Orchestration generates the *VNFD_Name*.**conf** configuration file based on the *VNFD_Name***Template.conf** file and the *VNFD_Name***Config.xml** file.

Network Service Orchestration reads all the name-value pairs in the *VNFD_Name***Config.xml** file and replaces the placeholder fields in the *VNFD_Name***Template.conf** file and generates the *VNFD_Name*.**conf** file.

The following text shows a sample configuration template for the Juniper vSRX VNF in the **Juniper_vSRX_VNFDTemplate.conf** configuration file:

```
<rpc>
 <edit-config>
     <target>
         <candidate/>
     </target>
     <config>
         <configuration>
             <security>
                 <utm>
                     <custom-objects>
                         <url-pattern>
                             <name>bad-sites</name>
                             <value>{{site-name}}</value>
                         </url-pattern>
                     </custom-objects>
                 </utm>
             </security>
         </configuration>
     </config>
 </edit-config>
 </rpc>
```

The following example shows a sample configuration for the Juniper vSRX VNF in the **Juniper_vSRX_VNFDConfig.xml** configuration file:

```
<vnfConfiguration>
    <config>
        <param>
            <name>site-name</name>
```

```
            <value>www.example.com</value>
        </param>
        <sbiToPushConfiguration>
            <interface>netconf</interface>
            <interface-script></interface-script>
        </sbiToPushConfiguration>
        <action>null</action>
    </config>
    </vnfConfiguration>
```

Setting Network Service Descriptor Properties

You define and specify properties for your network service in the *UIM_Home*/**config**/*network_service_descriptor*.**properties** file, where *network_service_descriptor* is the name of your network service descriptor. You create one properties file for each network service that you want to create and implement.

Table 3–3 describes the parameters that you specify for a network service.

Table 3–3 Network Service Descriptor Parameters

Parameter	Description
network_service_descriptor.**default.dataCenter**	Used to specify the default data center if you use multiple VIMs. Otherwise, leave blank. *network_service_descriptor* indicates the name of the network service descriptor. For example, NPaaS_NSD.
VIM_Id.network_service_descriptor.**VLD_Name**	Used to specify the name of the management network. In the properties files of the samples, by default, the VIM ID is **OpenStack**. The management network is the VLD Name that is specified in the **NPaaS_NSD.xml** file. If you use multiple VIMs, add another entry of the same parameter and specify the VIM ID and the management network. Add multiple instances of this parameter for specifying more VLDs.
VIM_Id.network_service_descriptor.**Data_IN**	Used to specify the VIM ID and the name of the data-in network. By default, the VIM ID is **OpenStack**. If you use multiple VIMs, add another entry of the same parameter and specify the VIM ID and the data-in network.
VIM_Id.network_service_descriptor.**Data_OUT**	Used to specify the VIM ID and the name of the data-out network. By default, the VIM ID is **OpenStack**. If you use multiple VIMs, add another entry of the same parameter and specify the VIM ID and the data-out network.
(Optional) **sdnController**.*network_service_descriptor*	Used to specify an implementation class for the SDN controller interface. The default implementation class is **com.oracle.communications.inventory.nso.nfvi.sdn.ODLManager**.
(Optional) *network_service*.**ovs.pktInToOVSPort**	Used to specify the Open vSwitch port number of the packet-in network.
(Optional) *network_service*.**ovs.pktOutToOVSPort**	Used to specify the Open vSwitch port number of the packet-out network.
(Optional) *network_service*.**ovs.custNetToOVSPort**	Used to specify the Open vSwitch port number of the customer-side network.

Table 3–3 (Cont.) Network Service Descriptor Parameters

Parameter	Description
(Optional) *network_service*.**ovs.internetToOVSPort**	Used to specify the Open vSwitch port number of the internet-side network.
(Optional) **npaas.ovs.bridge_id**	Specify the bridge ID for the Open VSwitch and prefix it with **openflow**. For example, **openflow**:*OpenFlow_ID*, where *OpenFlow_ID* is the OpenFlow ID. To retrieve the OpenFlow ID, in OpenDaylight call the following OpenDaylight REST API: http://*odlIPaddress*:*port*/restconf/operational/opendaylight-inventory:nodes/ where *odlIPaddress* is the IP address and *port* is the port number of the OpenDaylight virtual machine.

Network Service Orchestration provides properties files for the following sample network services:

- *UIM_Home*/**config/NPaaS_NSD.properties**. This properties file defines the properties for the NPaaS sample network service.
- *UIM_Home*/**config/ResidentialGateway_NSD.properties**. This properties file defines the properties for the Residential Gateway sample network service.

Onboarding VNFs Using TOSCA VNF Descriptor Templates

Topology and Orchestration Specification for Cloud Applications (TOSCA) is an OASIS standard language to describe the topology of cloud-based services. TOSCA provides specifications for defining NFV descriptors.

For more information about TOSCA, see the standards section on the OASIS web site: https://www.oasis-open.org/standards#toscav1.0

Network Service Orchestration supports onboarding of VNFs using TOSCA VNF descriptor templates that are in YAML format.

To onboard VNFs using TOSCA VNF descriptor templates, you import the TOSCA VNF descriptor templates that are in YAML format into Design Studio. When you import a TOSCA VNF descriptor template, Network Service Orchestration processes the YAML file that contains the VNF structural and topology details (connection points and connectivity requirements) and creates a VNF cartridge project with the required UIM entity specifications (such as Logical Device, Service, and Service Configuration specifications), assigned or referenced resource specifications, and characteristic specifications.

Before you import the TOSCA VNF descriptor templates into Design Studio, do the following:

1. Install Python 3.5.1. See "Installing Python" for instructions.
2. Set the **TOSCA_TRANSLATOR_HOME** environment variable to *UIM_SDK*/**NSO tools/ToscaTranslator**, where the **OracleCommsToscaTranslator.pyz** TOSCA translator file is available.

Note: If your operating system does not consider an environment variable defined with a blank space in the directory names, you can move the **OracleCommsToscaTranslator.pyz** TOSCA translator file to any location and provide the path in the **TOSCA_TRANSLATOR_HOME** environment variable.

3. If your Design Studio workspace is already open, close and reopen the workspace.
4. Import the required UIM base cartridges into Design Studio.

   See "Setting Up Design Studio for Extending Network Service Orchestration" for information about the required UIM base cartridges.

Sample TOSCA VNF Descriptor Template

The following text shows the TOSCA VNF descriptor template in YAML format for a sample firewall VNF. This sample shows the supported properties.

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: firewall

metadata:
  template_name: Firewall_VNF

topology_template:
  node_templates:
    Firewall_VNF:
      type: tosca.nodes.nfv.VNF
      properties:
        id: Firewall_Id
        vendor: Firewall_VNF_Vendor
        version: 1.0
      requirements:

    Firewall_VDU:
      type: tosca.nodes.nfv.VDU
      artifacts:
          VM_image: Firewall-vnf-image-v1
      capabilities:
        host:
          properties:
            num_cpus: 2
            mem_size: 4 GB
            disk_size: 20 GB

    CP1:
       type: tosca.nodes.nfv.CP
       properties:
        type: vPort
       requirements:
          - virtualLink: mgmt-net
          - virtualBinding: Firewall_VDU
    CP2:
       type: tosca.nodes.nfv.CP
       properties:
        type: vPort
       requirements:
          - virtualLink: pkt-in
```

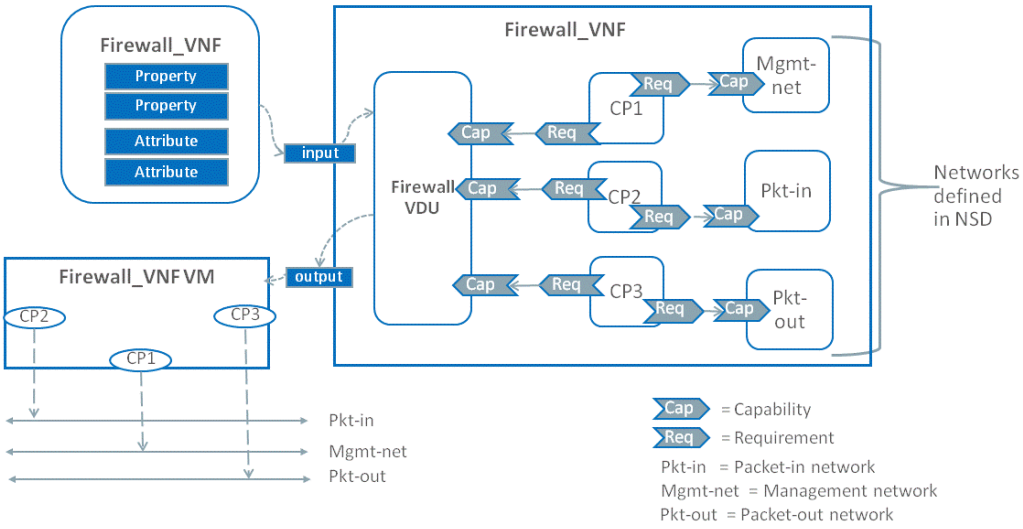
```
      - virtualBinding: Firewall_VDU

CP3:
  type: tosca.nodes.nfv.CP
  properties:
    type: vPort
  requirements:
    - virtualLink: pkt-out
    - virtualBinding: Firewall_VDU

mgmt-net:
  type: tosca.nodes.nfv.VL
  properties:
    vendor: <Vendor_mgmt-net>
pkt-in:
  type: tosca.nodes.nfv.VL
  properties:
    vendor: <Vendor_pkt-in>
pkt-out:
   type: tosca.nodes.nfv.VL
   properties:
     vendor: <Vendor_pkt-out>
```

Figure 3–1 depicts the TOSCA VNF Template to the VNF virtual machine mapping.

Figure 3–1 TOSCA VNF Template to VNF Virtual Machine Mapping



In the illustration, the **Firewall_VNF** VNF comprises one Virtual Deployment Unit (VDU) that is connected to three virtual links. The VDU has three connection points: CP1 (connection point 1) is connected to Mgmt-net (management network), CP2 (connection point 2) is connected to Pkt-in (packet-in network), and CP3 (connection point 3) is connected to Pkt-out (packet-out network). The networks are defined in the network service descriptor.

Installing Python

To install python:

1. Download and unzip **UIM_SDK.zip**.

2. Navigate to the **UIM_SDK\NSO tools\ToscaTranslator** directory and unzip the **OracleCommsToscaTranslatorEnvSetup.zip** file.

   The unzipped **OracleCommsToscaTranslatorEnvSetup** directory contains different directories for Windows and Linux operating systems.

3. Run the following command:

   Note: Before you run the commands, ensure that you have a working internet connection.

   - For Windows machine, launch command prompt and run the following command as administrator:

     ```
     install.bat
     ```

     If you use proxy and a target directory, run the following command:

     ```
     install.bat -proxy="www-proxy.example.com:port" -targetdir="C:\python35"
     ```

     where:

     – *www-proxy.example.com:port* is the proxy

     – *C:\python35* is the target directory of Python

     See Table 3–4 for details about the command line argument options.

   - For Linux machine, from terminal, run the following command as a pseudo user or with root permission:

     ```
     install.sh
     ```

   The installation script installs python and the required libraries on the machine.

   Table 3–4 describes the command line arguments.

Table 3–4 Command Line Arguments

For Windows	For Linux	Usage
-proxy	-p or --proxy	To connect to the internet through proxy.
-targetdir	Not Applicable	Specify the Python installation directory (on Windows only).
-internet	-i or --internet	Set Y if you have a working internet connection.

Importing the TOSCA VNFD Template into Design Studio

To import the TOSCA VNF template into Design Studio:

1. In Design Studio, from the **Studio** menu, select **Show Design Perspective**.
2. Click the **Studio Projects** tab.

   The Studio Projects view appears.

3. Right-click in the Studio Projects view and select **Import** and then select **Import VNFD TOSCA Template**.

   The Import VNFD TOSCA Template dialog box appears.

4. Click **Browse** and select the TOSCA VNF descriptor file in YAML format.

   Design Studio creates the VNF cartridge project with the required specifications.

After the VNF cartridge project is generated, to use this VNF cartridge with a network service, make appropriate changes in your network service cartridge project. For more information about working with network service cartridges, see "Designing Custom Network Services".

Tagging Network Service Orchestration Specifications

UIM uses tags to differentiate between inventory entities and entities used by Network Service Orchestration. You apply these tags to Network Service Orchestration specifications in Design Studio.

When you tag specifications with Network Service Orchestration tags, UIM filters the entities based on the tags and lists and displays only the relevant entities in Network Service Orchestration pages.

The tags for Network Service Orchestration specifications are included in the **OracleComms_NSO_BaseTags** cartridge. See "NSO Base Tags Cartridge" in *UIM Cartridge Guide* for more information.

For instructions about tagging specifications, see Design Studio Help.

Table 3–5 lists and describes the tags for the Network Service Orchestration specifications.

Table 3–5 Network Service Orchestration Specifications and Tags

Tag	Specification Type	Description
EMS	Custom Object	Tags a Custom Object specification as an EMS specification.
Endpoint	Custom Object	Tags a Custom Object specification as an Endpoint specification.
Network Service	Service	Tags a Service specification as a Network Service Orchestration Network Service specification.
Orchestration Request	Business Interaction	Tags a Business Interaction specification as an Orchestration Request specification.
PNF	▪ Service ▪ Logical Device	Tags a Service specification as a PNF Service specification. Tags a Logical Device specification as a PNF specification.
VNF	▪ Service ▪ Logical Device	Tags a Service specification as a VNF Service specification. Tags a Logical Device specification as a VNF device specification.

Designing Custom Network Services

You can use Design Studio to design and implement custom network services based on your business requirements. Designing a network service requires designing the service itself as well as the VNFs and PNFs it uses.

In Design Studio, you create a cartridge project for each network service, VNF, and PNF that you design. These cartridge projects include specifications and other artifacts. You compile the cartridge projects into cartridges for deployment into UIM.

To work properly with Network Service Orchestration, the specifications must include certain characteristics, relationships, and rulesets. See the following sections for more information:

- Creating Cartridges for VNFs
- Creating Cartridges for PNFs
- Creating Cartridges for Network Services

Creating Cartridges for VNFs

For each VNF that you want to use with a network service, create a cartridge project in Design Studio. In each VNF cartridge project, do the following:

- Create the following UIM entity specifications:
  - A Logical Device specification that represents the VNF. See "Logical Device Specification" for more information about the logical device specification.
  - A Service specification that represents the VNF. See "Service Specification" for more information.
  - A Service Configuration specification for the VNF. See "Service Configuration Specification" for more information
- A Capability service for the VNF service. See "Capability Service".
- A Service Configuration specification for the Capability service. See "Capability Service Configuration Specification".
- Create a technical actions file for the VNF Service specification. See "Creating a Technical Actions File" for more information.
- Create a VNF descriptor file for the VNF Service specification. See "Creating a Descriptor File" for more information.
- Create a configuration file for the VNF, if the VNF requires configuration. See "About VNF Configuration Files" for more information.
- Create a post-configuration template configuration file for the VNF. See "About VNF Configuration Files" for more information.
- Create a template file for the VNF. See "About VNF Configuration Files" for more information.
- Create custom code for extension. See "Extending Network Service Orchestration" for more information.

Logical Device Specification

Create a Logical Device specification to represent the VNF. This specification must include the characteristics listed in Table 3–6. These characteristics are provided in the **OracleComms_NSO_BaseCartridge** cartridge. You can optionally define and include additional characteristics.

See "Working with Characteristics" in *UIM Concepts* and Design Studio Help for more information about characteristics.

Table 3–6 Logical Device Specification Characteristics

Characteristic	Type	Description
availabilityZoneName	String	The name of the availability zone where the VNF gets instantiated.
dataCenterName	String	The name of the data center where the VNF gets instantiated.
deploymentFlavor	String	The deployment flavor used to deploy the VNF.

Table 3–6 (Cont.) Logical Device Specification Characteristics

Characteristic	Type	Description
externalID	String	The VNF ID.
host	String	The host ID where VNF is instantiated.
imageName	String	The name of the VNF image for instantiate.
version	String	The VNF version.
securityGroups	String	The security groups for the VNF.

There are no rulesets required for the VNF Logical Device specification. You can create custom rulesets to extend the default capabilities, however.

A VNF Logical Device specification must include the specification relationships listed in Table 3–7.

Table 3–7 Logical Device Specification Relationships

Specification	Name	Description
DeviceInterface	CPD	Multiplicity is from 0 to 100. This specification is available in the **OracleComms_NSO_BaseCartridge** cartridge.
ServiceSpecification	*user created*	Set this to the name of the VNF Service specification that you design.

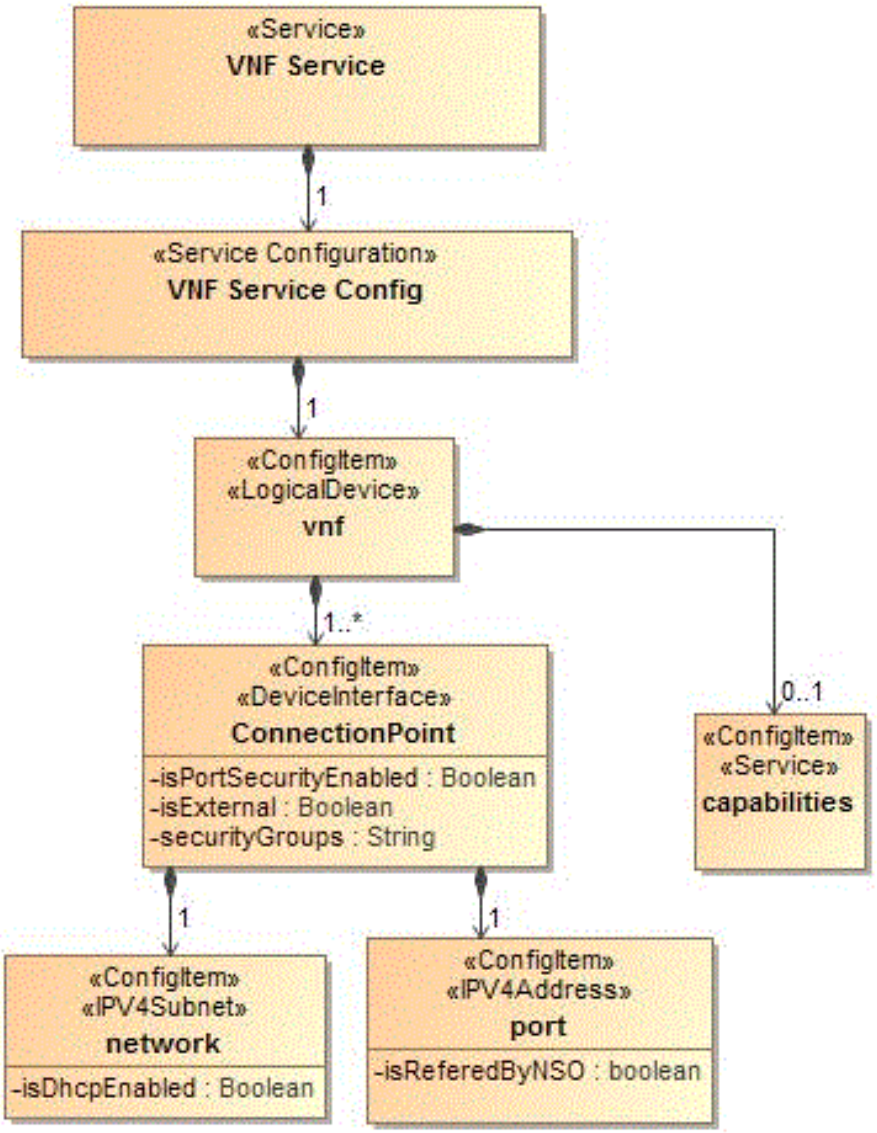
Associate the **VNF** tag to the VNF Logical Device specification to ensure that UIM correctly handles entities based on this specification. The **VNF** tag is provided in the **OracleComms_NSO_BaseTags** cartridge. See "Tagging Network Service Orchestration Specifications" for more information.

Service Specification

Create a Service specification to represent the VNF service. No characteristics or rulesets are required, but you can optionally add them to extend the default capabilities.

Figure 3–2 illustrates the VNF service model.

Figure 3–2 VNF Service Model



A VNF Service specification must include the specification relationships listed in Table 3–8.

Table 3–8 VNF Service Specification Relationships

Specification	Name	Description
ServiceSpecification	*user created*	The associated capability service specification, used to configure capabilities.
ServiceConfigurationSpecification	*user created*	The associated service configuration specification.

Apply the **VNF** tag to the VNF Service specification to ensure that UIM correctly handles entities based on this specification. The **VNF** tag is provided in the **OracleComms_NSO_BaseTags** cartridge. See "Tagging Network Service Orchestration Specifications" for more information.

Service Configuration Specification

Create a Service Configuration specification to accompany the VNF Service specification. The Service Configuration specification must include the configuration items listed in Table 3–9.

Table 3–9 VNF Service Configuration Items

Name	Parent Item	Multiplicity	Characteristics
vnf	null	Required	None
ConnectionPoint	vnf	1 to 100	▪ isExternal ▪ isPortSecurityEnabled ▪ securityGroups
port	ConnectionPoint	Required	None
network	ConnectionPoint	Required	isDhcpEnabled
capabilities	vnf	Optional. 0 to 1.	None

Define the specification options for the configuration items as shown in Table 3–10.

Table 3–10 VNF Service Configuration Specification Options

Item	Item Option Type	Specification	Specification Type
vnf	Assignment	VNF	Logical Device specification
ConnectionPoint	Reference	CPD	DeviceInterface specification
port	Reference	IPv4Address	IPv4Address specification
network	Reference	IPv4Subnet	IPv4Subnet specification
capabilities	Assignment	VNF Capability Service	Service specification

Associate the following rulesets with the Service Configuration specification. These rulesets are included in the **OracleComms_NSO_BaseCartridge** cartridge:

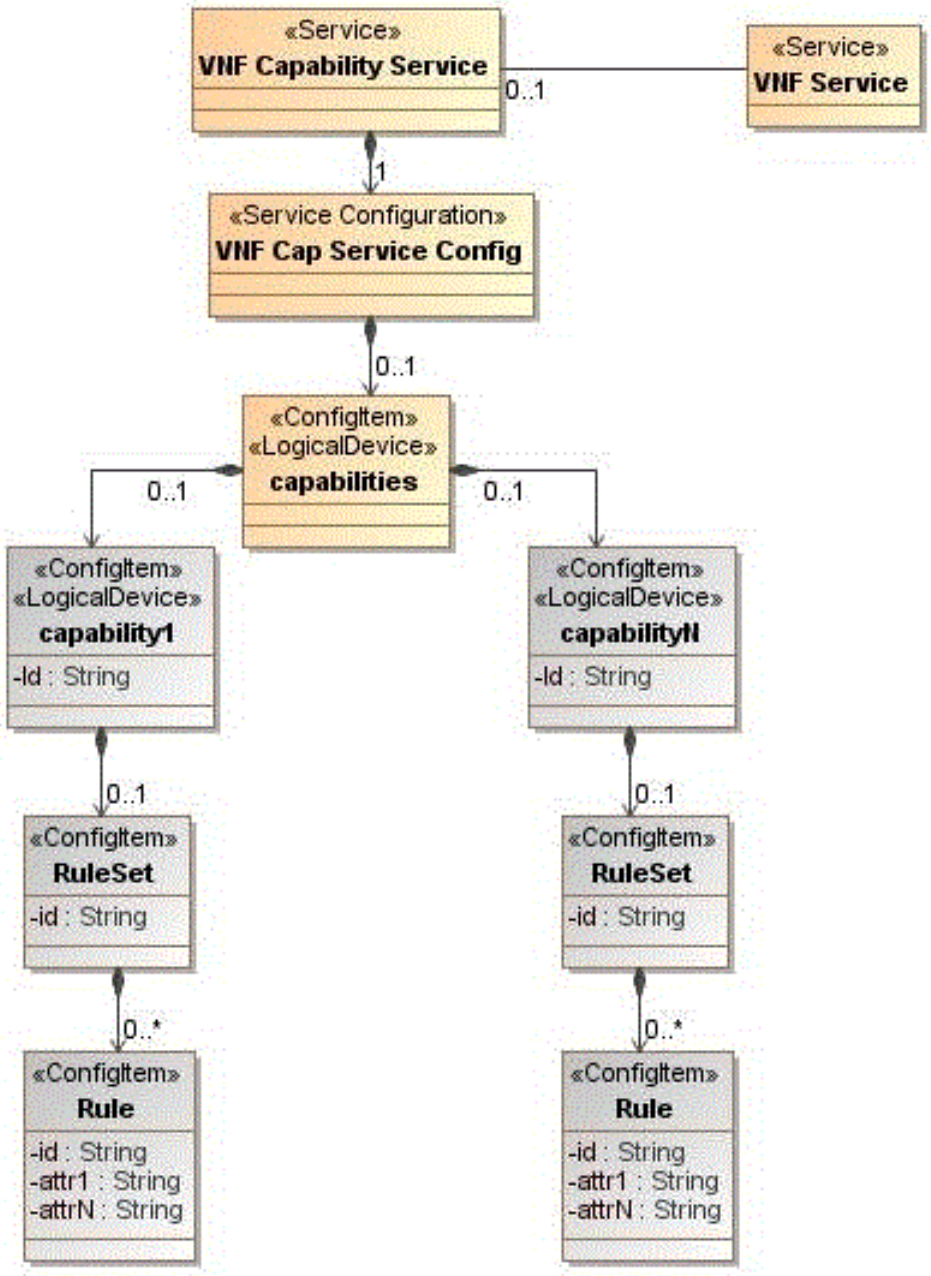
- IssueVNFServiceConfig_NSOBaseRulesetExtPt
- AutomateVNFServiceConfig_NSOBaseRulesetExtPt
- Cancel_VNFServiceConfigRulesetExtPt
- CompleteVNFServiceConfig_NSOBaseRulesetExtPt

Capability Service

Create a capability service to configure the VNF Service capabilities. No characteristics or rulesets are required, but you can optionally add them to extend the default capabilities.

Figure 3–3 shows how a VNF capability service is modeled.

Figure 3–3 VNF Capability Service Model



The Capability service must include the specification relationships listed in Table 3–11.

Table 3–11 Capability Service Specification Relationships

Specification Type	Name	Availability
ServiceConfigurationSpecification	*user created*	The associated capability service configuration specification.

Capability Service Configuration Specification

Create a Service Configuration specification to accompany the Capability Service specification.

Associate the following rulesets with the Capability Service configuration specification. These rulesets are included in the **OracleComms_NSO_BaseCartridge** cartridge.

- AutomateVNFCapabilityServiceConfig_NSOBaseRulesetExtPt
- IssueVNFCapabilityServiceConfig_NSOBaseRulesetExtPt

Creating Cartridges for PNFs

For each PNF that you want to use with a network service, create a cartridge project in Design Studio. In each PNF cartridge project, do the following:

- Create the following UIM entity specifications:
  - A Logical Device specification that represents the PNF. See "Logical Device Specification".
  - A Service specification that represents the PNF service. See "Service Specification".
  - A Service Configuration specification for the PNF service. See "Service Configuration Specification".
- Create a technical actions file for the PNF Service specification. See "Creating a Technical Actions File" for more information.
- Create a network service descriptor file for the Network Service specification. See "Creating a Descriptor File" for more information.
- Create custom code for extension. See "Extending Network Service Orchestration" for more information.

Logical Device Specification

Create a Logical Device specification to represent the PNF. This specification must include the characteristics listed in Table 3–12. These characteristics are provided in the **OracleComms_NSO_BaseCartridge** cartridge. You can optionally define and include additional characteristics. See "Working with Characteristics" in *UIM Concepts* and Design Studio Help for more information about characteristics.

Table 3–12 PNF Logical Device Specification Characteristics

Characteristic	Type	Description
ipAddress	String	The IP address of the PNF.
password	String	The password of the PNF.
username	String	The username of the PNF.
sshkey	String	The ssh key for the PNF. This characteristic is available in the PNF sample cartridge.
sslEnabled	Boolean	Indicates whether SSL is enabled for the PNF or not.

Associate the following rulesets with the PNF Logical Device specification. These rulesets are included in the **OracleComms_NSO_BaseCartridge** cartridge.

- CreatePNF_Ruleset. This ruleset validates the create PNF request.
- UpdatePNF_Ruleset. This ruleset validates the update PNF request.
- CreateEMS_Ruleset. This ruleset validates the create EMS request.

- UpdateEMS_Ruleset. This ruleset validates the update EMS request.

A PNF Logical Device specification must include the specification relationships listed in Table 3–13.

Table 3–13 PNF Logical Device Specification Relationships

Specification	Name	Description
DeviceInterfaceSpecification	CPD	Multiplicity is from 0 to 100. This specification is available in the **OracleComms_NSO_BaseCartridge** cartridge.
ServiceSpecification	*user created*	Set this to the name of the PNF Service specification that you design.

Apply the following tags to the PNF Logical Device specification to ensure that UIM correctly handles entities based on this specification. These tags are provided in the **OracleComms_NSO_BaseTags** cartridge.

- PNF
- EMS

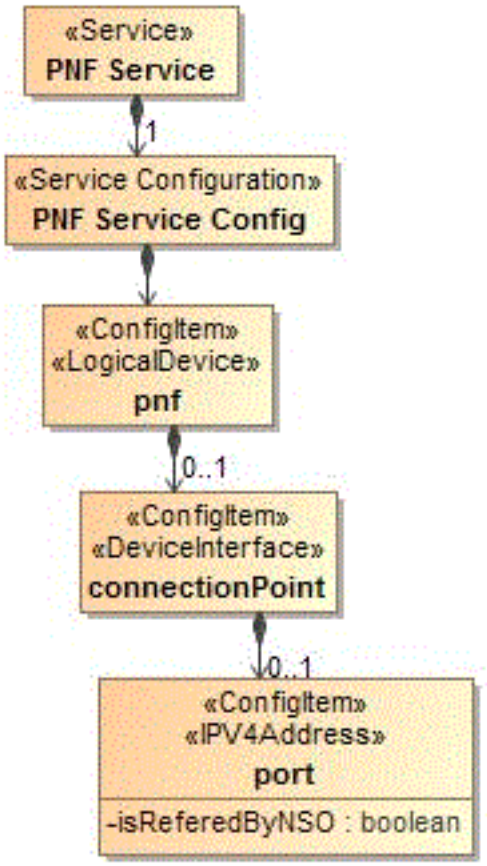
See "Tagging Network Service Orchestration Specifications" for more information.

Service Specification

Create a Service specification to represent the PNF service. No characteristics or rulesets are required, but you can optionally add them to extend the default capabilities.

Figure 3–4 shows how a PNF service is modeled.

Figure 3–4 PNF Service Model



A PNF Service specification must include the specification relationships listed in Table 3–14.

Table 3–14 PNF Service Specification Relationships

Specification	Name	Description
ServiceConfigurationSpecification	Cisco_xRV_Service_Config	The associated service configuration specification.

Apply the **PNF** tag to the PNF Service specification to ensure that UIM correctly handles entities based on this specification. The **PNF** tag is provided in the **OracleComms_NSO_BaseTags** cartridge. See "Tagging Network Service Orchestration Specifications" for more information.

Service Configuration Specification

Create a Service Configuration specification to accompany the PNF Service specification. The Service Configuration specification must include the configuration items listed in Table 3–15.

Table 3–15 PNF Service Configuration Items

Item	Parent Item	Multiplicity	Characteristics
pnf	null	Required	None
ConnectionPoint	pnf	1 to 100	None
port	ConnectionPoint	Required	None

Define the specification options for the configuration items as shown in Table 3–16.

Table 3–16 PNF Service Configuration Specification Options

Item	Item Option Type	Specification
pnf	Assignment	Logical Device
ConnectionPoint	Reference	DeviceInterface
port	Reference	IPv4Address

Associate the following rulesets with the Service Configuration Version specification. These rulesets are included in the **OracleComms_NSO_BaseCartridge** cartridge.

- IssuePNFServiceConfig_NSOBaseRulesetExtPt
- AutomatePNFServiceConfig_NSOBaseRulesetExtPt
- Cancel_PNFServiceConfigRulesetExtPt
- CompletePNFServiceConfig_NSOBaseRulesetExtPt

Creating Cartridges for Network Services

For each network service, create a cartridge project in Design Studio. In the cartridge project for the network service, do the following:

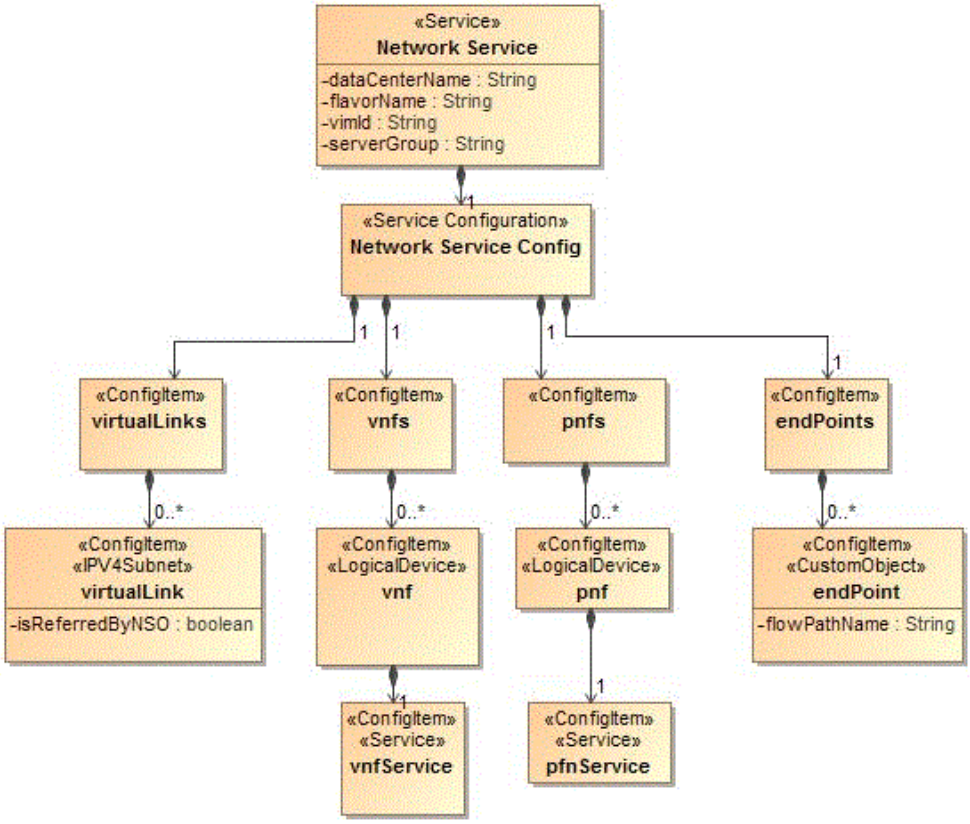
- Create the following UIM entity specifications:
  - A Service specification to represent the network service. See "Network Service Specification".
  - A Service Configuration specification to accompany the network service specification. See "Network Service Configuration Specification".
- Create a technical actions file for the Network Service specification. See "Creating a Technical Actions File" for more information.
- Create a network service descriptor file for the Network Service specification. See "Creating a Descriptor File" for more information.
- Create a custom properties file for the Network Service specification. See "Setting Network Service Descriptor Properties" for more information.
- Create custom code for extension. See "Extending Network Service Orchestration" for more information.

Network Service Specification

Create a Network Service specification to represent the network service.

Figure 3–5 illustrates the network service model.

Figure 3–5 Network Service Model



The Network Service specification must include the characteristics listed in Table 3–17. These characteristics are provided in the **OracleComms_NSO_BaseCartridge** cartridge.

Table 3–17 Network Service Specification Characteristics

Characteristic	Type	Description
dataCenterName	String	The name of the data center.
flavorName	String	The name of the service flavor.
serverGroup	String	The name of the server group.
vimId	String	The unique identifier of VIM.

The Network Service specification does not require any rulesets, but you can create custom rulesets to extend the default capabilities.

The Network Service specification must include the specification relationships listed in Table 3–18.

Table 3–18 Network Service Specification Relationships

Specification	Name	Description
ServiceConfigurationSpecification	*user created*	This is the associated service configuration specification.

Apply the **NetworkService** tag to the Network Service specification to ensure that UIM correctly handles entities based on this specification. This tag is provided in the **OracleComms_NSO_BaseTags** cartridge. See "Tagging Network Service Orchestration Specifications" for more information.

Network Service Configuration Specification

Create a Service Configuration specification to accompany the Network Service specification.

The Service Configuration specification must include the configuration items listed in Table 3–19.

Table 3–19 Network Service Configuration Items

Item	Parent Item	Multiplicity	Characteristics
virtualLinks	null	Required	None
virtualLink	virtualLinks	0-unbounded	isReferencedByNSO
vnfs	null	Required	None
vnf	vnfs	0 to unbounded	None
vnfService	vnf	Required	None
pnfs	null	Optional	None
pnf	pnfs	0 to unbounded	None
pnfService	pnf	Required	None
endPoints	null	Required	None
endPoint	endPoints	0 to unbounded	flowPathName

Define the specification options for the configuration items as shown in Table 3–20.

Table 3–20 Network Service Configuration Specification Options

Item	Item Option Type	Specification	Specification Type
virtualLinks	None	None	None
virtualLink	Reference	IPv4Subnet	IPv4Subnet specification
vnfs	None	None	None
vnf	Reference	VNF Logical Device	Logical Device specification
vnfService	Assignment	VNF Service	Service specification
pnfs	None	None	None
pnf	Reference	PNF Logical Device	Logical Device specification
pnfService	Assignment	PNF Service	Service specification
endPoints	None	None	None
endPoint	Reference	NSSubscriber	CustomObject specification

Associate the following rulesets with the Service Configuration Version specification. These rulesets are provided in the **OracleComms_NSO_BaseCartridge** cartridge.

- AutomateNetworkServiceConfig_NSOBaseRulesetExtPt
- IssueNetworkServiceConfig_NSOBaseRulesetExtPt

- CompleteNetworkServiceConfig_NSOBaseRulesetExtPt
- CancelNetworkServiceConfig_NSOBaseRulesetExtPt

4

Working with Network Services, VNFs, and PNFs

This chapter provides information about working with network services, VNFs, and PNFs in Oracle Communications Network Service Orchestration.

To work with network services, VNFs, and PNFs in Network Service Orchestration, you can use either the UIM user interface or the REST APIs. See UIM Help for instructions about performing tasks using the user interface.

When you use REST APIs, you use a REST API client and provide values for the required parameters in the API request. The values and the parameters are defined in the network service and VNF descriptor files that you created in Design Studio. See "Network Service Orchestration RESTful API Reference" for details about the REST APIs that you can use to perform various tasks.

You perform the following tasks related to network services, VNFs, and PNFs:

- Instantiating Network Services
- Terminating Network Services
- Modifying Network Services
- Scaling VNFs
- Monitoring and Healing VNFs
- Upgrading the Software Version of a VNF
- Working with PNFs in Network Services
- Retrieving Details About Network Services, VNFs, PNFs, and Descriptors

> **Note:** Based on the configurations that the VNFs in the network service require, some VNF life cycle operations may take some time to complete. In UIM and in your VIM, the resources may not be created, deleted, or updated immediately after you send the API request or complete the operation using the user interface.

Instantiating Network Services

You instantiate a network service to start a VNF on the network. A network service can have multiple VNFs that are connected to each other. When you instantiate a network service that has multiple VNFs, all the VNFs in the network service are started on the network. You can also include PNFs in your network services. See "Working with PNFs in Network Services" for information about including PNFs in

network services.

Before you instantiate a network service, ensure that your VIM is registered and the data center resources that your VIM manages are discovered. If you use multiple VIMs, register all your VIMs and discover the resources that the VIMs manage. See "Discovering VIM Resources" for information about discovering VIM resources.

If you use multiple VIMs to instantiate your network services, do the following:

- In UIM, create an Inventory Group entity and associate the VDC custom object that was generated during the discovery of VIM resources to the inventory group. See UIM Help for instructions.
- Create a ServiceLocation entity based on the Place specification and associate it to the Inventory Group entity.
- When you instantiate a network service, specify the service location of the VDC in which you want to instantiate the network service.

After you instantiate a network service, verify the following in UIM:

- The network service and its configurations are created and are in **In Service** status. You can see this in the Network Service Summary page of the network service.
- The VNF service and its configurations are created and associated to the network service.
- The VNFs and PNFs, which are represented as logical devices, are created.
- The specified networks are either created or referenced.
- The details of the endpoints are updated in the service configuration.

In your VIM, verify the following:

- The VNF instances are up and running.
- The specified networks are either created or referenced.
- The VNFs are linked to the networks.

Based on the configurations you defined in the network service and the VNF and PNF descriptor files, Network Service Orchestration does the following tasks during the instantiation of a network service:

- Finds the best suitable data center for the network service from among the data centers that you registered.
- Performs resource orchestration to find the best suitable availability zone where constituent VNFs can be deployed.
- Creates new networks or references existing networks that are required for connectivity among the VNFs.
- Manages IP addresses of all the resources.
- Configures the VNFs based on pre-defined parameters. See "About VNF Configuration Files" for more information.
- If the network service includes a PNF, configures the PNF and checks access to the PNF using the management IP address.
- If you integrated a monitoring engine, configures the monitoring engine to trigger alarms for VNFs that reach a specified threshold to enable healing of VNFs.
- If you integrated an SDN controller, configures routing paths for end-to-end packet flow.

Note: If the instantiation of a network service fails at any stage of the transaction due to insufficient ports or other resources in the VIM (or for any other reason), Network Service Orchestration rolls back the resources completely.

See "Modifying Network Services" for information about adding and removing VNFs, endpoints, and PNFs in network services.

Modifying Network Services

You modify a network service to add or remove endpoints, VNFs, and PNFs in the network service. You add a VNF to a network service to enable the network service to deliver additional service capabilities. You remove a VNF from a network service when it is no longer required. Similarly, you can add and remove endpoints and PNFs in your network services.

Using the user interface and the REST APIs, you can perform the following the tasks:

- Add VNFs to network services
- Remove VNFs from network services
- Add PNFs to network services
- Remove PNFs from network services
- Add endpoints to network services
- Remove endpoints from network services

See UIM Help for instructions about performing tasks using the user interface. See "Network Service Orchestration RESTful API Reference" for details about the Network Service Orchestration REST APIs that you can use to modify network services.

Adding VNFs to Network Services

You add VNFs to a network service when you create it. Every network service requires one or more VNFs.

With one exception, you can add VNFs to network services by using the user interface or the REST APIs. Before you instantiate a network service, you can add constituent VNFs in a network service by using the Network Service Summary pages. After a network service is instantiated, you can add VNFs to the network service by using the **scale** options only.

After you add a VNF to a network service, do the following:

1. In UIM, verify the following:
   - The network service is updated with a new service configuration version showing the VNF that you added.
   - The status of the new service configuration version shows completed.
2. In your VIM, verify that a new VNF instance is created.

Removing VNFs from Network Services

Before you instantiate a network service, you can remove any VNF that you have already added to your network service.

Note: You cannot remove VNFs from a network service that is already instantiated. To remove VNFs from a network service that is already instantiated, use the **scale** options.

After you delete a VNF from a network service, do the following:

1. In UIM, verify the following:
   - The network service is updated with a new service configuration version showing that the VNF is deleted.
   - The status of the service configuration version shows completed.
2. In your VIM, verify that the VNF instance is removed and the resources that were assigned to the VNF are freed up.

Terminating Network Services

You terminate a network service to deactivate all the constituent VNFs in the network service. When you terminate a network service, all the resources that were allocated to the VNFs are released and become available for consumption by other network services.

After you terminate a network service, do the following:

1. In UIM, verify the following:
   - The status of the network service and the VNF and PNF services is changed to Disconnected.
   - The statuses of the logical devices corresponding to the associated VNFs and PNFs are changed to Unassigned.
2. In your VIM, verify that the VNF instance is deleted and all the allocated resources are released.

Scaling VNFs

You scale VNFs in a network service when there is high or low utilization of CPU and memory on the machine on which the network service is instantiated. When you scale VNFs, you can either clone existing VNF instances in a network service or remove VNFs from a network service.

Monitoring and Healing VNFs

You monitor VNFs in a network service to track their performance and take actions based on their CPU utilization, number of requests handled, and other key performance indicator (KPI) parameters.

To monitor VNFs, you configure and use monitoring engines. You also configure and specify the relevant parameters in the Network Service descriptor file. See "Describing Deployment Flavors" for information about defining assurance parameters for monitoring and healing a VNF.

By default, Network Service Orchestration supports integration with OpenStack Ceilometer, which monitors VNFs and reboots failed VNFs automatically based on KPI thresholds that are defined in the network service descriptor file.

You can integrate other third-party monitoring engines by using the extensions provided in Network Service Orchestration. See "Implementing a Custom Monitoring Engine" for more information about implementing a third-party monitoring engine.

When the monitoring engine identifies a failed VNF in a network service, you can heal the failed VNF by either rebooting or replacing the virtual machine on which the VNF is deployed. In OpenStack Ceilometer, when you heal a failed VNF by replacing it, the new VNF may come up in a different host. Network Service Orchestration performs resource orchestration to deduce the resources from the new host and the availability zone and adds up the resources count to the host.

To heal a VNF:

1. Ensure that you have defined the assurance parameters for the VNFs in the network service descriptor file. See "Describing Deployment Flavors" for information about defining assurance parameters.
2. Do one of the following:
   - Use the user interface to reboot or replace the VNF. See UIM Help for instructions.
   - Call the RESTful API. See "Network Service Orchestration RESTful API Reference" for more information.
3. In your VIM, verify that the VNF you rebooted or replaced is listed as active and running.

About the Monitoring Tabs in the User Interface

When you create specifications for your VNFs and network services in Design Studio, you can add characteristics to the specifications to capture URLs of web pages of your monitoring systems. You can define the characteristics to capture any number of URLs of web pages. See Design Studio documentation about working with characteristics and specifications.

In the UIM user interface, when you create network services, you specify the URLs of web pages of your monitoring systems. After the network service is instantiated, each URL that you specified for your monitoring system displays an embedded page in a tab in the Network Service Summary page.

You can use the monitoring tabs to view service topologies of your network services, and the following metrics about your VNFs:

- CPU
- Memory
- Disk space

See UIM Help for more information about the monitoring tabs.

Upgrading the Software Version of a VNF

You upgrade the software version of a VNF in a network service to utilize the functional capabilities that a later software version of the VNF provides.

You can use the user interface or the REST API to upgrade the software version of a VNF. See UIM Help and "Network Service Orchestration RESTful API Reference" for instructions.

After you upgrade the software version of a VNF, do the following:

1. In UIM, verify the following:
   - The network service is updated with a new service configuration version.
   - The version number of the VNF image that you upgraded the VNF to is listed.
2. In your VIM, verify that the VNF instance displays the name of the VNF image that you upgraded to.

Working with PNFs in Network Services

You can include Physical Network Functions (PNFs) in your network services.

To include PNFs in a network service, do the following:

1. If your PNF is managed by an EMS, register the Element Management System (EMS) with Network Service Orchestration by using the REST API. See "Network Service Orchestration RESTful API Reference" for information about registering EMSs.
2. Register the PNF with Network Service Orchestration by using the REST API.
3. When you create a network service in the user interface, add the PNF to the network service. See UIM Help for instructions about adding PNFs to network services.

If you use REST APIs to instantiate a network service with PNFs, specify the details of the PNFs in the API request. See "Network Service Orchestration RESTful API Reference" for information about the API request.

Retrieving Details About Network Services, VNFs, PNFs, and Descriptors

You can retrieve and view details about your network services, VNFs, PNFs, network service descriptors, and VNF descriptors by using the user interface and the REST APIs. In the user interface, you can search for and view details by using standard UIM techniques. See UIM Help for more information.

Network Service Orchestration provides RESTful APIs that you can call to retrieve and view different types of information about your network services, VNFs, and PNFs. For details about the RESTful APIs, see "Network Service Orchestration RESTful API Reference".

5

Implementing the Sample Network Services

This chapter provides information about the sample network services that are provided with Network Service Orchestration.

Network Service Orchestration includes the following sample cartridges that you can use as references for designing and implementing your own network services:

- **Juniper_vSRX_VNF**. This sample cartridge contains the Juniper vSRX firewall VNF to use with the network protection service.
- **Checkpoint_NG_FW_VNF**. This sample cartridge contains the Checkpoint firewall VNF to use with the network protection service.
- **Cisco_xRV_PNF**. This sample cartridge contains the Cisco XRV router PNF to use with the residential gateway network service or the network protection service.
- **NPaaS_NetworkService**. This sample cartridge provides the functionality to implement a Network Protection as a Service (NPaaS) network service.
- **ResidentialGateway_NetworkService**. This sample cartridge provides the functionality to implement a Residential Gateway network service

Configuring the Juniper vSRX Base Image

Before you implement the sample network services, you must configure the software image of the Juniper vSRX firewall VNF. You use this VNF with the Network Protection and the Residential Gateway network services.

To configure the Juniper vSRX base image:

1. Download the Juniper vSRX base image from Juniper's web site.
2. Install OpenStack and source the tenant's credentials file.
3. In OpenStack, upload the downloaded base image to the Glance repository by running the following command:

   ```
   glance image-create --name vsrx-vmdisk-15.1X49-D40_base --is-public true
   --container-format bare --disk-format qcow2 --file
   media-vsrx-vmdisk-15.1X49-D40.6.qcow2
   ```

   where:

   - **vsrx-vmdisk-15.1X49-D40_base** is the name of the image uploaded into the repository
   - **media-vsrx-vmdisk-15.1X49-D40.6.qcow2** is the name of the base image downloaded from the vendor's web portal.

4. In OpenStack, create a flavor with the following specifications by running the following command:

   Specifications:

   - Name: **vsrx.medium**
   - VCPUs: **2**
   - Root Disk: **20** GB
   - Ephemeral Disk: **0** GB
   - RAM: **4096** MB

   Command:

   ```
   nova flavor-create vsrx.medium auto 4096 20 2
   ```

5. Boot the image by running the following command:

   ```
   nova boot --flavor vsrx.medium --image vsrx-vmdisk-15.1X49-D40_base --nic net_
   ID=109ae4cf-3cea-4729-a24f-957c4ed6d3c6 vsrx_base_instance
   ```

   where:

   - *net_ID* is the ID of your management network in OpenStack.
   - **vsrx_base_instance** is the name of the vsrx instance you are spawning in OpenStack.
   - **vsrx-vmdisk-15.1X49-D40_base** is the name of the base image that is uploaded into the repository.

6. After the image boots up, navigate to the Instances console in OpenStack and run the following commands:

   ```
   root@%cli
   root>config
   root#
   delete security
   set system root-authentication plain-text-password
   New password: Enter a password
   Retype new password:Enter a password
   ```

   OpenStack prompts for a password.

7. Enter any password and run the following commands:

   ```
   set system login user admin class super-user authentication plain-text-password
   New password:password
   Retype new password:password
   ```

   OpenStack prompts for a password.

8. Enter any password.

   The username and the password that you specify here become the username and password of the VNF image that you specify in the VNF descriptor. Network Service Orchestration uses these credentials to update the configuration.

9. Run the following commands:

   ```
   set system services netconf ssh
   set interfaces fxp0 description "Managament Interface" unit 0 family inet dhcp
   set interfaces ge-0/0/0 description "Customer Interface" unit 0 family inet
   dhcp
   ```

```
set interfaces ge-0/0/1 description "Internet interface" unit 0 family inet
dhcp
set security zones security-zone Customer host-inbound-traffic system-services
ping
set security zones security-zone Internet host-inbound-traffic system-services
ping
set security zones security-zone Customer interfaces ge-0/0/0.0
host-inbound-traffic system-services dhcp
set security zones security-zone Customer interfaces ge-0/0/0.0
host-inbound-traffic system-services ping
set security zones security-zone Internet interfaces ge-0/0/1.0
host-inbound-traffic system-services dhcp
set security zones security-zone Internet interfaces ge-0/0/1.0
host-inbound-traffic system-services ping
set routing-instances Traffic instance-type virtual-router
set routing-instances Traffic interface ge-0/0/0.0
set routing-instances Traffic interface ge-0/0/1.0
set groups security-rules security policies from-zone <*> to-zone <*> policy
<*> then log session-init session-close
set security policies apply-groups security-rules
set security policies from-zone Customer to-zone Internet policy
Customer-Internet-Access match source-address any destination-address any
application any
set security policies from-zone Customer to-zone Internet policy
Customer-Internet-Access then permit
set security policies from-zone Internet to-zone Customer policy Deny-All match
source-address any destination-address any application any
set security policies from-zone Internet to-zone Customer policy Deny-All then
deny
set security utm custom-objects url-pattern bad-sites value
http://www.example.com
set security utm custom-objects custom-url-category bad-category value
bad-sites
set security utm feature-profile web-filtering juniper-local profile wf-profile
custom-block-message "Website blocked by NPaaS. Powered by Oracle" default
log-and-permit fallback-settings default block too-many-requests block
set security utm utm-policy utm-protect web-filtering http-profile wf-profile
commit
exit
exit
```

10. Create a snapshot of the running instance of the Juniper vSRX image by running the following command:

    `nova image-create --poll` **`vsrx_base_instance vsrx-vmdisk-15.1X49-D40_updated`**

    where:

    - **vsrx_base_instance** is the name of the vsrx instance
    - **vsrx-vmdisk-15.1X49-D40_updated** is the name of the vsrx image snapshot uploaded to OpenStack Glance.

    Use this snapshot as the software image for instantiation of the Juniper vSRX VNF.

Implementing the Network Protection Service

Network Service Orchestration provides sample cartridges that you can use as references for designing and implementing a network protection service.

The **NPaaS_NetworkService** sample cartridge contains the functionality to implement the sample Network Protection as a Service (NPaaS) network service.

The network protection service constitutes and uses the following VNFs:

- Juniper vSRX firewall

  The **Juniper_vSRX_VNF** sample cartridge contains the functionality to implement a Juniper vSRX firewall as a VNF.

- Checkpoint firewall

  The **Checkpoint_NG_FW_VNF** sample cartridge contains the functionality to implement a Checkpoint firewall as a VNF.

The network protection service requires and uses the following software components:

- UIM 7.3.4 and the Network Service Orchestration 7.3.4 cartridges
- OpenStack VIM, with Open vSwitch capability
- OpenDaylight SDN Controller
- Software images of the firewall VNFs

To implement the network protection service:

1. Configure the Juniper vSRX base image. See "Configuring the Juniper vSRX Base Image" for instructions.
2. In OpenStack, create a tenant or reference an existing tenant with administrator privileges.
3. Reference an existing management network that can be shared by all the components of Network Service Orchestration.

   The management network requires, at a minimum:

   - One IP address for each:
     - Machine on which UIM is installed
     - Virtual machine on which Open vSwitch is installed
     - Machine on which OpenDaylight is installed
   - One IP address for each virtual machine on which you want to bring up the VNFs
4. Connect the management network and the external network to a virtual router. This enables you to use floating IP addresses for providing access to the data center.
5. Create a customer-side network that facilitates the customer's network traffic to reach the VNFs.

   Table 5–1 shows examples of IP addresses and IP address ranges of network and subnet configuration for the customer-side network.

Table 5–1 Example of Network and Subnet Configuration for Customer-side Network

CIDR	IP Allocation Pool	Gateway IP	DHCP Enabled	Additional Routes	DNS Name Server
192.168.2.0/24	Start 192.0.2.145 End 192.0.2.250	192.0.2.1	Yes	None	None

6. Create an Internet-side network that facilitates the traffic from the customer-side network to the Internet.

   Table 5–2 shows examples of IP addresses and IP address ranges of network and subnet configuration for the Internet-side network.

Table 5–2 Example of Network and Subnet Configuration for Internet-side Network

CIDR	IP Allocation Pool	Gateway IP	DHCP Enabled	Additional Routes	DNS Name Server
192.168.2.0/24	Start 192.0.2.2 End 192.0.2.254	192.0.2.1	No	None	None

7. Create packet-in and packet-out networks.

   Table 5–3 shows examples of IP addresses and IP address ranges of network and subnet configuration for the packet-in network.

Table 5–3 Example of Network and Subnet Configuration for Packet-in Network

CIDR	IP Allocation Pool	Gateway IP	DHCP Enabled	Additional Routes	DNS Name Server
192.168.2.128/25	Start 192.0.2.129 End 192.0.2.140	-	Yes	None	None

   Table 5–4 shows examples of IP addresses and IP address ranges of network and subnet configuration for the packet-out network.

Table 5–4 Example of Network and Subnet Configuration for Packet-out Network

CIDR	IP Allocation Pool	Gateway IP	DHCP Enabled	Additional Routes	DNS Name Server
192.168.2.0/25	Start 192.0.2.115 End 192.0.2.126	192.0.2.1	Yes	None	None

8. Start the OpenDaylight virtual machine on the management network.
9. Start the Open vSwitch virtual machines on the management network, customer-side network, Internet-side network, packet-in network, and packet-out network.
10. On the Open vSwitch virtual machine, do the following:
    - Create a steering bridge:

      ```
      ovs-vsctl add-br steering
      ```

      where *steering* is the name of the integration bridge.
    - Add the interfaces of the networks you created to the steering bridge:

      ```
      ovs-vsctl add-port steering networkInterface
      ```

      where *networkInterface* is the name of the network interface. For example, *eth1*.

      ```
      ovs-vsctl add-port steering eth1
      ovs-vsctl add-port steering eth2
      ovs-vsctl add-port steering eth3
      ovs-vsctl add-port steering eth4
      ovs-vsctl add-port steering eth5
      ```

- Set the IP address and port number of the OpenDaylight virtual machine as the controller to the steering bridge:

  ovs-vsctl set-controller steering tcp:*OpenDaylight_IPAddress*

  ovs-vsctl set bridge steering protocols="OpenFlow13"

  where *OpenDaylight_IPAddress* is the IP address of the OpenDaylight virtual machine.

- Get the port numbers:

  ovs-vsctl -- --columns=*name_of_port* list Interface

  where *name_of_port* is the name of the Open vSwitch port.

11. Open the *UIM_Home*/**config/nso.properties** file and update the following parameters.
    - **startIpAddress**. Specify the subnet start IP address. By default, when Network Service Orchestration creates a network, the subnet IP address starts with 192.168.0.0.
    - **NSO_HOST**: *IPv4address*. Specify the host on which UIM is installed. By default, Network Service Orchestration considers the host on which the UIM server is running. If the server is running on a private network that is unavailable to external network, specify a reachable IP address for the server.
    - **NSO_USERNAME**: *username*

      where *username* is the username of the server on which UIM is installed.
    - **NSO_PASSWORD**: *encrypted_password*

      where *encrypted_password* is the encrypted password of the server on which UIM is installed. See "Setting Network Service Orchestration Properties" for information about encrypting the password.
12. Open the *UIM_Home*/**config/NPaaS_NSD.properties** file and specify values for the parameters listed in Table 5–5:

Table 5–5 Parameters in the NPaaS Network Service Descriptor Properties File

Parameter	Description
NPaaS_ NSD.default.dataCenter	Specify a default data center.
VIM_Id.**NPaaS_ NSD.ManagementNetwork**	Specify the VIM ID and the name of the management network. By default, the VIM ID is **OpenStack**. The management network is the VLD Name that is specified in the **NPaaS_NSD.xml** file. If you use multiple VIMs, add another entry of the same parameter and specify the VIM ID and the management network.
VIM_Id.**NPaaS_NSD.Data_IN**	Specify the VIM ID and the name of the data-in network. By default, the VIM ID is **OpenStack**. If you use multiple VIMs, add another entry of the same parameter and specify the VIM ID and the data-in network.

Table 5–5 (Cont.) Parameters in the NPaaS Network Service Descriptor Properties File

Parameter	Description
VIM_Id.**NPaaS_NSD.Data_OUT**	Specify the VIM ID and the name of the data-out network. By default, the VIM ID is **OpenStack**. If you use multiple VIMs, add another entry of the same parameter and specify the VIM ID and the data-out network.
sdnController.NPaaS_NSD	Specify an implementation class for the SDN controller interface. The default implementation class is **com.oracle.communications.inventory.nso.nfvi.sdn.ODL Manager**.
npaas.ovs.pktInToOVSPort	Specify the Open vSwitch port number of the packet-in network.
npaas.ovs.pktOutToOVSPort	Specify the Open vSwitch port number of the packet-out network.
npaas.ovs.custNetToOVSPort	Specify the Open vSwitch port number of the customer-side network.
npaas.ovs.internetToOVSPort	Specify the Open vSwitch port number of the internet-side network.
npaas.ovs.bridge_id	Specify the bridge ID for the Open VSwitch and prefix it with **openflow**. For example, **openflow**:*OpenFlow_ID*, where *OpenFlow_ID* is the OpenFlow ID. To retrieve the OpenFlow ID, in OpenDaylight call the following OpenDaylight REST API: http://*odlIPaddress*:*port*/restconf/operational/opendaylight-inventory:nodes/ where *odlIPaddress* is the IP address and *port* is the port number of the OpenDaylight virtual machine.

13. Deploy the Network Service Orchestration cartridges into UIM. See "Installing and Integrating the Network Service Orchestration Components" for information about deploying the cartridges in the specified order.
14. Register the VIM by calling the corresponding RESTful API. See "Registering the VIM" for instructions.
15. Discover the VIM resources. See "Discovering VIM Resources" for instructions.

    The Network Protection service is ready for instantiation.

Implementing the Residential Gateway Network Service

Network Service Orchestration provides sample cartridges that you can use as references for designing and implementing a residential gateway network service.

The **ResidentialGateway_NetworkService** sample cartridge contains the functionality to implement the Residential Gateway network service.

The Residential Gateway network service constitutes and uses the following VNFs and PNFs:

- Juniper vSRX firewall VNF

  The **Juniper_vSRX_VNF** sample cartridge contains the functionality to implement a Juniper vSRX firewall as a VNF.

- Cisco xRV router PNF

The **Cisco_xRV_PNF** sample cartridge contains the functionality to implement a Cisco xRV router as a PNF.

The Residential Gateway network service requires and uses the following software components:

- UIM 7.3.4 and the Network Service Orchestration 7.3.4 cartridges
- OpenStack VIM, with Open vSwitch capability
- Software image of the Juniper firewall VNF
- Cisco xRV PNF. Ensure that the PNF is up and running on a management IP address.

To implement the Residential Gateway network service:

1. Configure the Juniper vSRX base image. See "Configuring the Juniper vSRX Base Image" for instructions.
2. In OpenStack, create a tenant or reference an existing tenant with administrator privileges.
3. Reference an existing management network that can be shared by all the components of Network Service Orchestration.
4. Reference the existing provider network and create or reference a virtual router that can provide external access to the PNF.
5. Specify the details of the provider network and the virtual router in the Residential Gateway network service descriptor file. This enables you to use floating IP addresses for providing access to the PNF.
6. Open the *UIM_Home*/**config**/**ResidentialGateway_NSD.properties** file and specify values for the parameters listed in Table 5–6.

Table 5–6 Parameters in the Residential Gateway Descriptor Properties File

Parameter	Description
ResidentialGateway_ NSD.default.dataCenter	Specify a default data center.
VIM_Id.**ResidentialGateway_ NSD.ManagementNetwork**	Specify the VIM ID and the name of the management network. By default, the VIM ID is **OpenStack**. The management network is the VLD Name that is specified in the **ResidentialGateway_NSD.xml** file. If you use multiple VIMs, add another entry of the same parameter and specify the VIM ID and the management network.
VIM_Id.**ResidentialGateway_ NSD.Data_IN**	Specify the VIM ID and the name of the data-in network. By default, the VIM ID is **OpenStack**. If you use multiple VIMs, add another entry of the same parameter and specify the VIM ID and the data-in network.
VIM_Id.**ResidentialGateway_ NSD.Data_OUT**	Specify the VIM ID and the name of the data-out network. By default, the VIM ID is **OpenStack**. If you use multiple VIMs, add another entry of the same parameter and specify the VIM ID and the data-out network.
sdnController.ResidentialGateway_NSD	Specify an implementation class for the SDN controller interface. The default implementation class is **com.oracle.communications.inventory.nso.nfvi.sdn.ODLManager**.

7. If you use multiple data centers, override the provider network and the virtual router details specified in the network service descriptor file with the provider network and virtual router details of your data center:

   ```
   vim_Id.network_service_descriptor.external_network_name = provider_network
   vim_Id.network_service_descriptor.virtual_router_name = virtual_router
   ```

   For example, if you deploy the Residential Gateway network service on an OpenStack VIM with VIM ID **pod5**, specify:

   ```
   pod5.ResidentialGateway_NSD.ext-net = publicNet
   ```

8. Deploy the Network Service Orchestration cartridges into UIM. See "Installing and Integrating the Network Service Orchestration Components" for information about deploying the cartridges in the specified order.
9. Register the PNF by using the REST API. See "Network Service Orchestration RESTful API Reference" for a sample request for registering PNFs.

   See "Working with PNFs in Network Services" for more information about working with PNFs.
10. Register the VIM by using the REST API. See "Registering the VIM" for instructions.
11. Discover the VIM resources. See "Discovering VIM Resources" for instructions.
12. To enable connectivity between the VNF and PNF, the VNF is assigned with a floating IP address. Configure the static routes corresponding to the floating IP in the PNF manually or by extending the cartridges.

    The Residential Gateway network service is ready for instantiation.

6

Extending Network Service Orchestration

This chapter describes how you can customize and extend Oracle Communications Network Service Orchestration to meet the business needs of your organization.

You can extend the functionality of Network Service Orchestration by:

- Designing cartridges in Oracle Communications Design Studio. See "Designing Custom Network Services".

  For more information about designing cartridges:

  – See *UIM Cartridge Guide* for information about the leading practices for extending cartridge packs.

  – See "About Cartridges and Cartridge Packs" in *UIM Developer's Guide* for information about how to extend cartridge packs.

  – See Design Studio Help for instructions on how to extend cartridge packs through specifications, characteristics, and rulesets.

  Important: To ensure that your extensions can be upgraded and supported, you must follow the guidelines and policies described in *UIM Cartridge Guide*.

- Using extension points and Java interface extensions. See "Using Extension Points and Java Interface Extensions to Extend Network Service Orchestration".

Setting Up Design Studio for Extending Network Service Orchestration

To extend Network Service Orchestration, you build an Inventory cartridge in Design Studio. The UIM Software Developer's Kit (UIM SDK) provides the resources required to build an Inventory cartridge in Design Studio.

To set up Design Studio for extending Network Service Orchestration:

1. Follow the steps described in "Building an Inventory Cartridge Using the UIM SDK" in the *UIM Developer's Guide*.
2. Create a local directory (*NSO_CartridgePack_Home*).
3. Locate the **OracleComms_NSO_CartridgePacks.zip** file and extract it into the *NSO_CartridgePack_Home* local directory.
4. Copy the following WebLogic libraries from your WebLogic installation into the **OTHER_LIB** local directory:
   - *WL_Home*/**oracle_common/modules/groovy-all-2.0.5.jar**

- *WL_Home*/**oracle_common/modules/jersey-client-1.18.jar**
- *WL_Home*/**oracle_common/modules/jettison-1.1.jar**
- *WL_Home*/**wlserver/modules/features/weblogic.server.merged.jar**

5. In Design Studio, open a new workspace.
6. Import the following UIM base cartridges into Design Studio from *UIM_SDK_Home*/**cartridges**:
   - ora_uim_baseextpts
   - ora_uim_basemeasurements
   - ora_uim_basespecifications
   - ora_uim_basetechnologies
   - ora_uim_common
   - ora_uim_mds
   - ora_uim_model
   - OracleComms_NSO_BaseCartridge
7. Import the following Network Service Orchestration sample cartridges into Design Studio from *NSO_CartridgePack_Home*/**designStudio/cartridgeZips**:
   - Juniper_vSRX_VNF
   - Checkpoint_NG_FW_VNF
   - Cisco_xRV_PNF
   - ResidentialGateway_NetworkService
   - NPaaS_NetworkService
8. In Design Studio, configure the following Java build path classpath variables for the Network Service Orchestration cartridge projects:
   - UIM_LIB. Specify the path as *UIM_SDK_Home*/**lib**
   - OTHER_LIB. Specify the path as *OTHER_LIB_Home*

See "Designing Custom Network Services" for information about creating cartridges for new network services.

Using Extension Points and Java Interface Extensions to Extend Network Service Orchestration

You can extend the core functionality of Network Service Orchestration by:

- Writing a custom ruleset extension point. See "Writing a Custom Ruleset Extension Point".
- Using Java interface extensions. See "Using Java Interface Extensions".

Writing a Custom Ruleset Extension Point

You can extend the core functionality of Network Service Orchestration by writing a custom ruleset extension point and associating the extension point with the ruleset in Design Studio. See “Extending UIM Through Rulesets” in *UIM Developer’s Guide* for more information.

Table 6–1 describes the Network Service Orchestration core APIs that can be extended by using extension points in Network Service Orchestration.

Table 6–1 Network Service Orchestration Core APIs and Extension Points

API	Extension Point	Description
NetworkServiceDesignManager.processCreate	NetworkServiceDesignManager_processCreate	Implements the design-and-assign logic for a network service when the network service is instantiated.
NetworkServiceDesignManager.processDisconnect	NetworkServiceDesignManager_processDisconnect	Cleans up the network service resources when the network service is terminated.
NetworkServiceDesignManager.processChange	NetworkServiceDesignManager_processChange	Implements the design-and-assign logic or cleans up the resources when a network service is updated.
VNFServiceDesignManager.processCreate	VNFServiceDesignManager_processCreate	Implements the design-and-assign logic for the VNF service when a network service is instantiated with a VNF.
VNFServiceDesignManager.processDisconnect	VNFServiceDesignManager_processDisconnect	Cleans up the VNF service resources when a network service is terminated.
VNFServiceDesignManager.processChange	VNFServiceDesignManager_processChange	Implements the design-and-assign logic for a VNF service when the network service is updated.
VNFServiceManager.processTechnicalActions	VNFServiceManager_processTechnicalActions	Activates or removes the resources in a VIM for each VNF service.
NetworkServiceManager.processTechnicalActions	NetworkServiceManager_processTechnicalActions	Activates or removes the resources in a VIM for each network service.
ConsumerHelper.getDataCenterForConsumer	ConsumerHelper_getDataCenterForConsumer	Looks up the data center based on the NS endpoint.
VNFServiceHelper.createVNF	VNFServiceHelper_createVNF	Creates a VNF.
ConsumerHelper.getDataCenterLookupIdentifier	ConsumerHelper_getDataCenterLookupIdentifier	Returns the string representation of the dynamic property in the JSON request for NS instantiation.
NetworkServiceManager.designInstantiate	NetworkServiceManager_designInstantiate_Global	Used to design the network service for instantiation.
NetworkServiceManager.designUpdate	NetworkServiceManager_designUpdate_Global	Used to design the network service for update.

Using Java Interface Extensions

You can extend the core functionality of Network Service Orchestration by using Java interface extensions. You write a new Java implementation class for a core interface and implement the core interface for a specific network service or VNF descriptor. See *UIM API Overview* for more information about the Network Service Orchestration Java manager classes and package locations.

Network Service Orchestration supports the following functionality through custom Java implementation classes:

- Implementation of a custom SDN controller. See "Implementing a Custom SDN Controller".
- Implementation of a custom VNF monitoring engine. See "Implementing a Custom Monitoring Engine".

- Implementation of a custom VIM. See "Implementing a Custom VIM".
- Implementation of a custom VNF life cycle manager. See "Implementing a Custom VNF Life Cycle Manager".
- Implementation of an adapter for a custom VNF manager. See "Implementing an Adapter for a Custom VNF Manager".
- Implementation of a custom VNF connection manager. See "Implementing a Custom VNF Connection Manager".
- Implementation of a custom VNF configuration manager. See "Implementing a Custom VNF Configuration Manager".
- Implementation of a custom response manager. See "Implementing a Custom Response Manager".

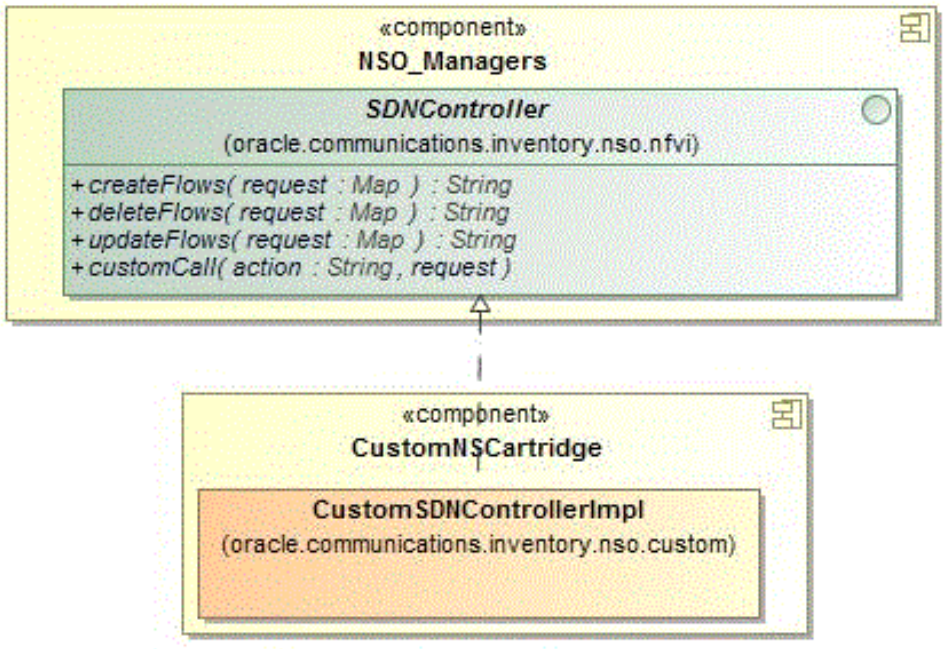
Note: Implementing some custom managers requires changes to the **nso.properties** file. Make a backup copy of the file before modifying it.

Implementing a Custom SDN Controller

By default, Network Service Orchestration supports integration with OpenDaylight, but you can also implement a custom SDN controller.

Figure 6–1 shows a model diagram that depicts the relationship between the Java Manager interface for the SDN Controller and your new custom Java implementation class.

Figure 6–1 Custom SDN Controller Model



To implement a custom SDN controller:

1. In the custom Network Service descriptor cartridge project, create a Java implementation class for the SDN controller.
2. Configure the custom SDN controller class to implement the **oracle.communications.inventory.nso.nfvi.SDNController** interface, which is available in *UIM_SDK*/**lib**/**nso-managers.jar**.
3. Override the following methods in the custom SDN controller Java implementation class:

```
public String createFlows(Map request) throws Exception
public String deleteFlows(Map request) throws Exception
public String updateFlows(Map request) throws Exception
```

4. In your Network Service descriptor cartridge project, create or update the network service properties file and add the following entry:

sdnController.*NSD_Name=SDNController_ImplementationClassPath*

where:

- *NSD_Name* is the name of the network service descriptor
- *SDNController_ImplementationClassPath* is the path of the implementation class of your custom SDN controller

5. Build and redeploy the cartridge.

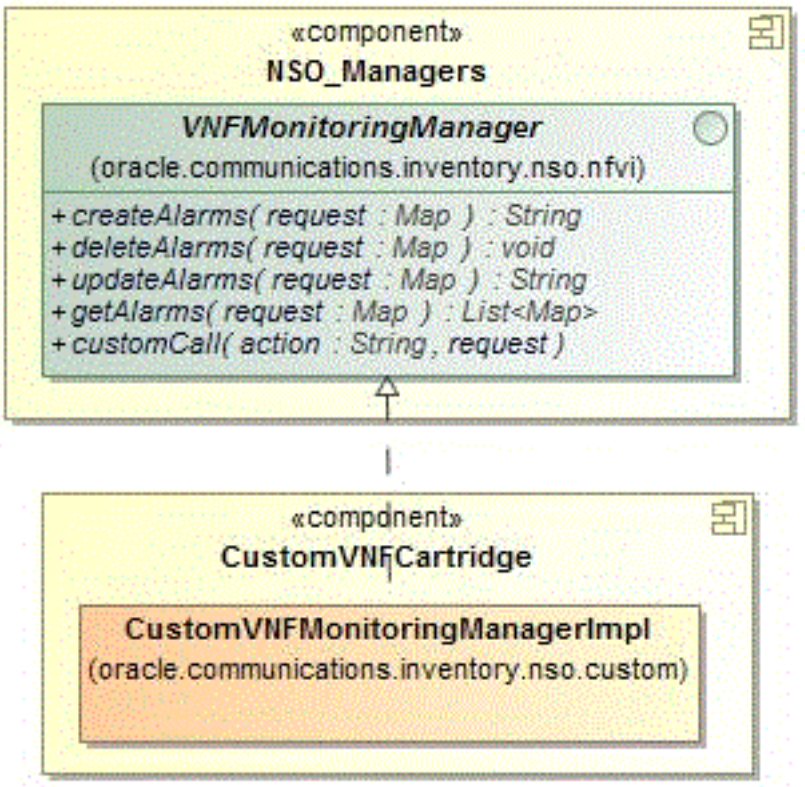
Note: If the **sdnController**.*NSD_Name* entry is commented out or if the path of the implementation class is not specified, Network Service Orchestration does not perform the network flow operations such as creation of flows, deletion of flows, and update of flows for the network service.

Implementing a Custom Monitoring Engine

By default, Network Service Orchestration supports integration with OpenStack Ceilometer, but you can also implement and use a custom monitoring engine.

Figure 6–2 shows a model diagram that depicts the relationship between the Java Manager interface for the Monitoring Manager and your new custom Java implementation class.

Figure 6–2 Custom Monitoring Engine Model



To implement a custom monitoring engine:

1. In the custom VNF descriptor cartridge project, create a Java implementation class for the VNF monitoring manager.
2. Configure the VNFMonitoringManager class to implement the **oracle.communications.inventory.nso.nfvi.VNFMonitoringManager** interface, which is available in *UIM_SDK*/**lib**/**nso-managers.jar**.
3. Override the following methods in the custom VNF monitoring engine Java implementation class:

```
public String createAlarms(Map request) throws Exception
public String deleteAlarms(Map request) throws Exception
public String updateAlarms(Map request) throws Exception
public String getAlarms(Map request) throws Exception
public String customCall(Map request) throws Exception
```

4. In the VNF descriptor cartridge project, create or update the VNF properties file and add the following entry:

   vnfMonitor.*VNFD_Name=MonitoringEngine_ImplementationClassPath*

   where:

   - *VNFD_Name* is the name of the VNF descriptor
   - *MonitoringEngine_ImplementationClassPath* is the path of the implementation class of your monitoring engine

5. Build and redeploy the cartridge.

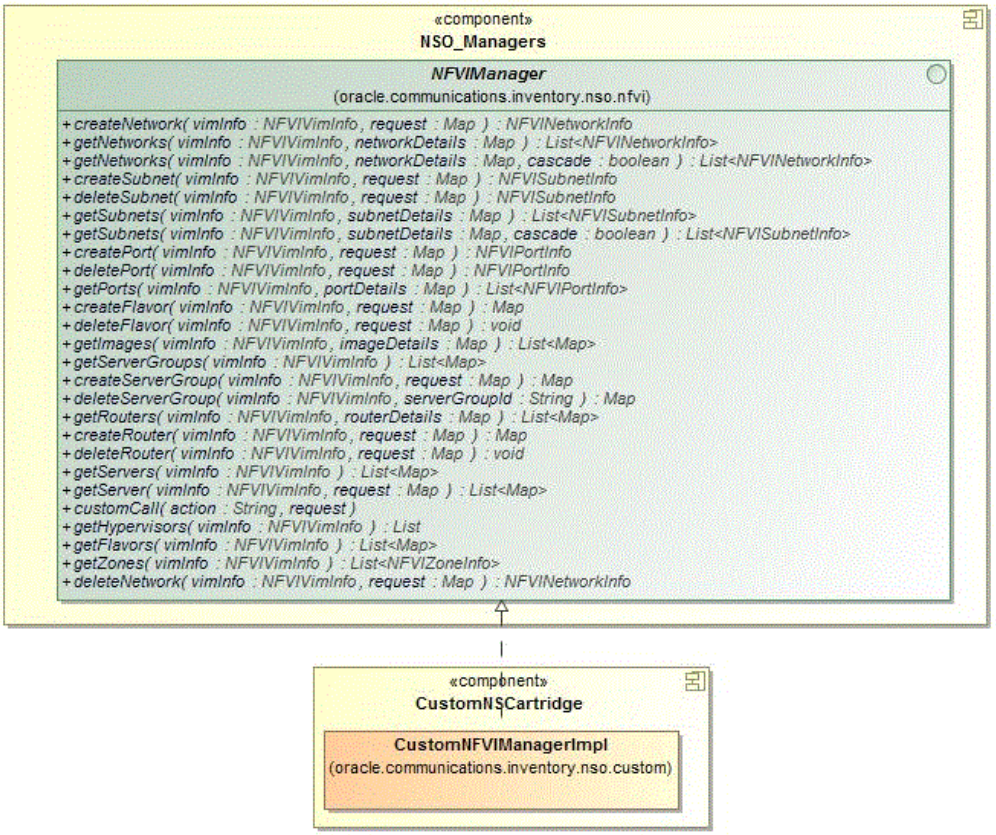
Note: If the **vnfMonitor**.*VNFD_Name* entry is commented out or if the path of the implementation class is not specified, Network Service Orchestration does not perform monitoring operations such as creation, deletion, and update of alarms for the network service.

Implementing a Custom VIM

By default, Network Service Orchestration supports integration with OpenStack, but you can also implement a custom VIM.

Figure 6–3 shows a model diagram that depicts the relationship between the Java Manager interface for the VIM and your new custom Java implementation class.

Figure 6–3 Custom VIM Model



To implement a custom VIM:

1. In your custom cartridge project, create a Java implementation class for the NFVIManager interface.
2. Configure the NFVIManager class to implement the **oracle.communications.inventory.nso.nfvi.NFVIManager** interface, which is available in *UIM_SDK*/**lib**/**nso-managers.jar**.
3. Override the methods in the custom NFVI manager Java implementation class.
4. Open the *UIM_Home*/**config**/**nso.properties** file, and add or update the following entry:

   nfviMgr.*nfviType=VIM_ImplementationClassPath*

where:

- *nfviType* is the type of VIM. For example, OpenStack or VMware.
- *VIM_ImplementationClassPath* is the path of the implementation class of your VIM

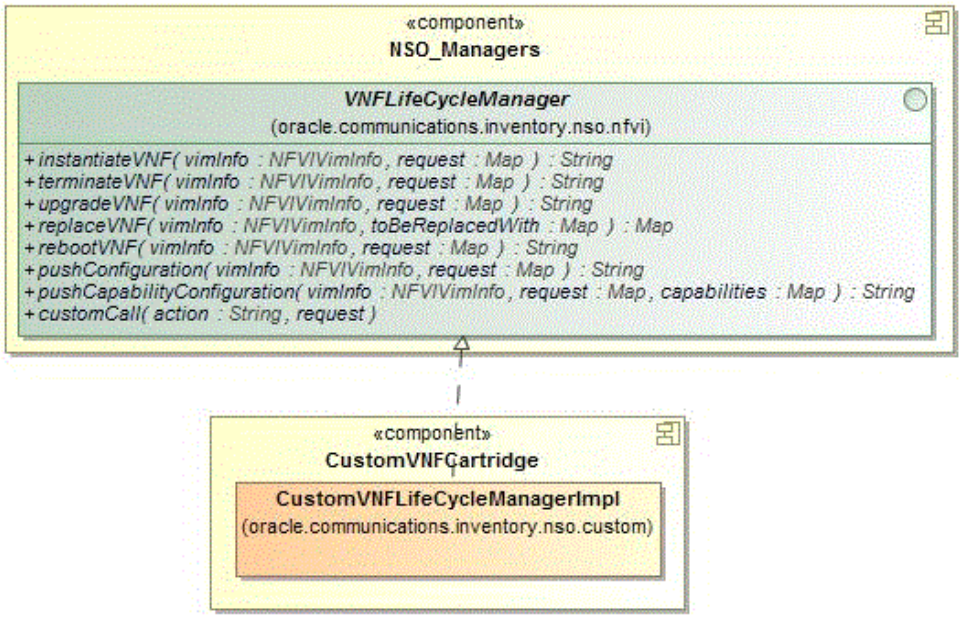
5. Build and redeploy the cartridge.

Implementing a Custom VNF Life Cycle Manager

By default, Network Service Orchestration manages VNF life cycle operations (such as instantiate, reboot, and terminate) by using OpenStack Compute services (referred to as Nova), but you can also implement and use a custom VNF life cycle manager with Network Service Orchestration.

Figure 6–4 shows a model diagram that depicts the relationship between the Java Manager interface for the Life Cycle Manager and your new custom Java implementation class.

Figure 6–4 Custom VNF Life Cycle Manager Model



To implement a custom VNF life cycle manager:

1. In your custom cartridge project, create a Java implementation class for the VNF life cycle manager.
2. Configure the custom VNF life cycle manager class to implement the **oracle.communications.inventory.nso.nfvi.VNFLifeCycleManager** interface, which is available in *UIM_SDK*/**lib**/**nso-managers.jar**.
3. Override the methods in the custom VNF life cycle manager Java implementation class.
4. Open the *UIM_Home*/**config**/**nso.properties** file, and add or update the following entry:

   vnflcMgr.*vimType=VNFLifecycleManager_ImplementationClassPath*

where:

- *vimType* is the type of VIM
- *VNFLifecycleManager_ImplementationClassPath* is the path of the implementation class of your custom VNF life cycle manager

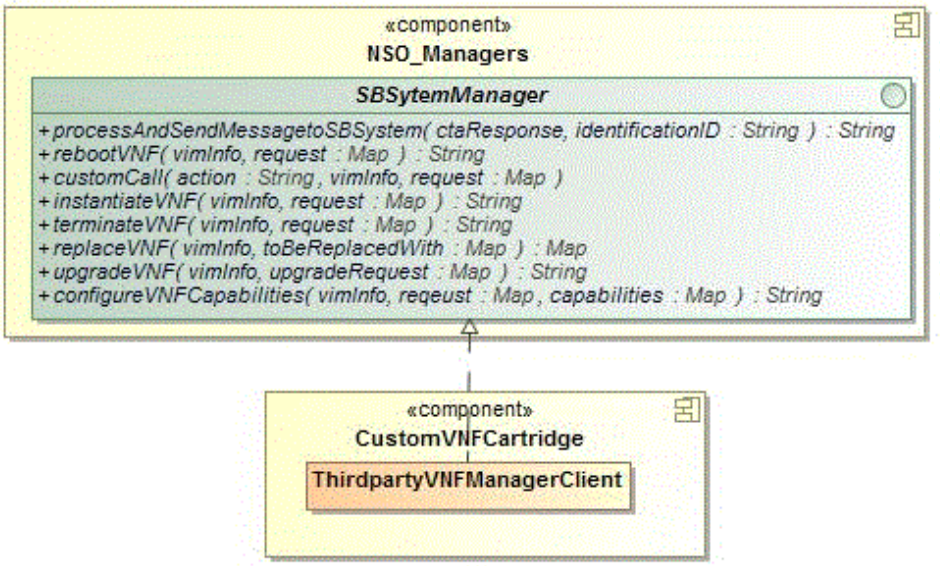
5. Build and redeploy the cartridge.

Implementing an Adapter for a Custom VNF Manager

By default, Network Service Orchestration contains and uses the built-in adapter for the built-in VNF manager to manage the VNFs in your network services, but you can also implement an adapter to integrate with third-party VNF managers.

Figure 6–5 shows a model diagram that depicts the relationship between the Java Manager interface for the VNF Manager and your new custom Java implementation class.

Figure 6–5 Custom Adapter for VNF Manager Model



To implement an adapter for custom VNF manager:

1. In your custom cartridge project, create a Java implementation class for the custom adapter.
2. Configure the custom adapter class to implement the **oracle.communications.inventory.nso.api.sb.SBSystemManager** interface, which is available in *UIM_SDK*/**lib**/**nso-managers.jar**.
3. Override the methods in the custom adapter Java implementation class:
4. Open the *UIM_Home*/**config**/**nso.properties** file, and add or update the following entry:

   sbClient.*name_VNFD=vnf_manager_adapter_ImplementationClassPath*

   where:

   - *name_VNFD* is the name of the VNF descriptor

- *vnf_manager adapter_ImplementationClassPath* is the path of the implementation class of your custom

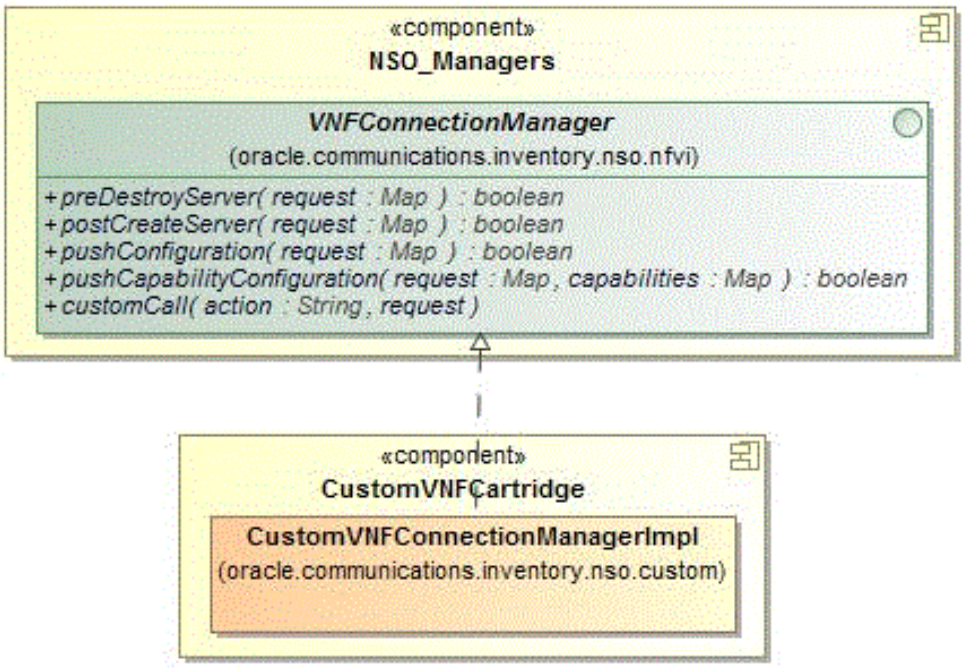
5. Build and redeploy the cartridge.

Implementing a Custom VNF Connection Manager

Network Service Orchestration includes a VNF connection manager that enables Network Service Orchestration to establish a communication channel with VNFs for deploying configurations during VNF life cycle operations. You can also implement a custom VNF connection manager for Network Service Orchestration by writing an extension.

Figure 6–6 shows a model diagram that depicts the relationship between the Java Manager interface for the VNF Connection Manager and your new custom Java implementation class.

Figure 6–6 Custom VNF Connection Manager Model



To implement a custom VNF connection manager:

1. In the custom VNF descriptor cartridge project, create a Java implementation class for the custom VNF connection manager.
2. Configure the custom VNF connection manager class to implement the **oracle.communications.inventory.nso.nfvi.VNFConnectionManager** interface, which is available in *UIM_SDK*/**lib**/**nso-managers.jar**.
3. Override the methods in the custom VNF connection manager Java implementation class.
4. In the VNF descriptor cartridge project, create or update the VNF properties file and add the following entry:

   vnfConnectionMgr.*VNFD_Name=VNFConnectionManager_ImplementationClassPath*

where:

- *VNFD_Name* is the name of the VNF descriptor
- *VNFConnectionManager_ImplementationClassPath* is the path of the implementation class of your custom VNF connection manager

5. Build and redeploy the cartridge.

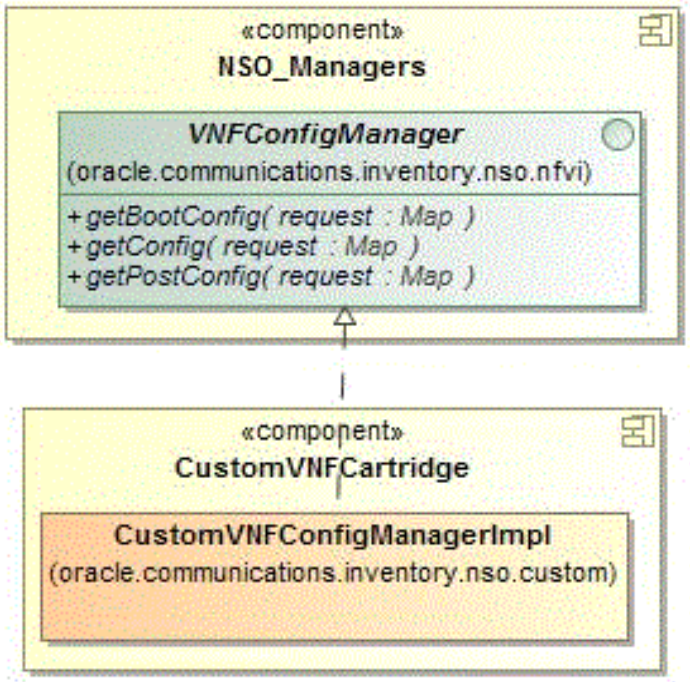
> **Note:** If the **vnfConnectionMgr**.*VNFD_Name* key is commented out or if the path of the implementation class is not specified, Network Service Orchestration does not run configurations on the virtual machines on which the VNFs are deployed.

Implementing a Custom VNF Configuration Manager

Network Service Orchestration includes a VNF configuration manager that generates configuration content for VNF configuration. You can also implement a custom VNF configuration manager for Network Service Orchestration by writing an extension.

Figure 6–7 shows a model diagram that depicts the relationship between the Java Manager interface for the VNF Configuration Manager and your new custom Java implementation class.

Figure 6–7 Custom VNF Configuration Manager Model



To implement a custom VNF configuration manager:

1. In the custom VNF descriptor cartridge project, create a Java implementation class for the custom VNF configuration manager.

2. Configure the custom VNF configuration manager class to implement the **oracle.communications.inventory.nso.nfvi.VNFConfigManager** interface, which is available in *UIM_SDK*/**lib**/**nso-managers.jar**.
3. Override the methods in the custom VNF configuration manager Java implementation class.
4. In the VNF descriptor cartridge project, create or update the VNF properties file and add the following entry:

   vnfConfigMgr.*VNFD_Name*=*VNFConfigurationManager_ImplementationClassPath*

   where:

   - *VNFD_Name* is the name of the VNF descriptor
   - *VNFConfigurationManager_ImplementationClassPath* is the path of the implementation class of your custom VNF configuration manager
5. Build and redeploy the cartridge.

> **Note:** If the **vnfConfigMgr**.*VNFD_Name* entry is commented out or if the path of the implementation class is not specified, Network Service Orchestration does not generate configurations for the VNF.

Implementing a Custom Response Manager

By default, Network Service Orchestration includes a response manager that publishes the status of the VNF and network service life-cycle operations to a topic in the WebLogic server. You can also implement a custom response manager by writing an extension.

To implement a custom response manager:

1. In your custom cartridge project, create a Java implementation class for the custom response manager.
2. Configure the custom response manager class to implement the **oracle.communications.inventory.nso.nfvi.NSOResponseManager** interface, which is available in *UIM_SDK*/**lib**/**nso-managers.jar**.
3. Override the following method in the custom response manager Java implementation class:

   ```
   public void processRequest(NSResponseInfo response)   throws
   ValidationException
   ```

4. Open the *UIM_Home*/**config**/**nso.properties** file, and add or update the following entry:

   nso.ResponseManager.list.1=*ResponseManager_ImplementationClassPath*

   where *ResponseManager_ImplementationClassPath* is the path of the implementation class of your custom response manager.

   Network Service Orchestration supports multiple implementations of response manager.
5. Build and redeploy the cartridge.

Localizing Network Service Orchestration

You can localize the UIM user interface, UIM Help, and the responses that the REST APIs return into your local language.

To localize Network Service Orchestration:

1. Localize the UIM user interface and UIM Help. For instructions, see the chapter about localizing UIM in *UIM Developer's Guide*.
2. Localize the responses that the RESTful APIs return. See "Localizing the Responses in RESTful APIs" for instructions.

Localizing the Responses in RESTful APIs

To localize the responses in the Network Service Orchestration RESTful APIs:

1. Make a copy of the *UIM_Home*/**config**/**resources**/**logging**/**nsoresourcebundle.properties** file in the same directory and rename it as **nsoresourcebundle**_*localeID*.**properties**, where *localeID* is the locale ID of your local language. For example, rename it to **nsoresourcebundle**_*fr_FR*.**properties** to localize the responses into French.
2. Open the **nsoresourcebundle**_*localeID*.**properties** file and localize the messages.
3. (Optional) If you want to implement the sample Network Protection service by using the sample cartridges, make a copy of the *UIM_Home*/**config**/**resources**/**logging**/**npassresourcebundle.properties** file in the same directory and name it as **npaasresourcebundle**_*localeID*.**properties** and localize the messages.
4. Restart the UIM server.
5. In your RESTful API client, update the Accept-Language header with the locale ID. For example, for French, specify *fr-FR*.

7

Network Service Orchestration RESTful API Reference

This chapter provides reference information about the Oracle Communications Network Service Orchestration RESTful API resources.

About the Network Service Orchestration RESTful APIs

The Network Service Orchestration RESTful API requests provide the northbound interface to Network Service Orchestration. Operation support systems (OSS) and VNF managers query the resource inventory for data.

The RESTful API requests enable you to perform various functions by using a RESTful API client.

The root URL for the Network Service Orchestration RESTful API resources is:

- HTTP connection: http://*nso_host*:*port*/ocnso/1.1
- SSL connection: https://*nso_host*:*ssl_port*/ocnso/1.1

  where:

  - *nso_host* is the name of the host on which Oracle Communications Unified Inventory Management (UIM) is installed
  - *port* is the port number of the machine on which UIM is installed
  - *ssl_port* is the SSL port number of the machine on which UIM is installed

To access the Network Service Orchestration RESTful API resources, in your RESTful API client, choose Basic Authentication and specify the user name and password of the machine on which UIM is installed.

> **Note:** If you use HTTPS-enabled OpenStack Keystone RESTful APIs, add the Certified Authority certificate to the TrustStore that your application server uses. If OpenStack Keystone is configured with self-signed certificate, then add the self-signed certificate to the TrustStore of the application server. See the Oracle WebLogic Server documentation for information about configuring the TrustStore.

The **ora_nso_webservices.war** file is included in the **inventory.ear** file and is deployed once the UIM installation is complete.

Network Service Orchestration RESTful API Resources

Table 7–1 lists the Network Service Orchestration RESTful API requests provided, along with the method, resource and description for each. This table is ordered first by entity and then by the method with the order of POST, PUT, GET and DELETE.

Table 7–1 Network Service Orchestration RESTful API Resources

Request	Method	Resource	Description
Register VIM	POST	/ocnso/1.1/vim	Registers the IP address, port, user name and password of the Virtual Infrastructure Manager (VIM) with Network Service Orchestration.
Discover VIM Resources	POST	/ocnso/1.1/vim/*vimId*/discovery?infoLevel=*vim_information*	Discovers the resources of the registered VIM and adds them to inventory.
Update VIM	PUT	/ocnso/1.1/vim/*vimId*	Updates the attributes (for example IP address, port, user name, password, and project name) of an existing VIM in Network Service Orchestration.
Get VIM Details	GET	/ocnso/1.1/vim/*vimId*	Returns information about a VIM that is registered with Network Service Orchestration.
Instantiate Network Service	POST	/ocnso/1.1/ns	Instantiates a network service and its constituent Virtual Network Functions (VNFs).
Scale Network Service	POST	/ocnso/1.1/ns/*networkServiceId*/scale/*vnfId*	Scales a VNF in a network service.
Get Network Services	GET	/ocnso/1.1/ns/nsdName=*nsdName*	Returns a list of all active network services that are created based on the given network service descriptor.
Get Network Service Details	GET	/ocnso/1.1/ns/*networkServiceId*	Returns details about a network service.
Get Network Service VNFs	GET	/ocnso/1.1/ns/*networkServiceId*/vnfs	Returns details about VNFs in a network service.
Get Network Service Networks	GET	/ocnso/1.1/ns/*networkServiceId*/networks	Returns details about networks in a network service.
Get Network Service End Points	GET	/ocnso/1.1/ns/*networkServiceId*/endpoints	Returns details about endpoints in a network service.
Get Network Service Status	GET	/ocnso/1.1/ns/*networkServiceId*/status	Returns status information of a network service.
Terminate Network Service	DELETE	/ocnso/1.1/ns/*networkServiceId*	Terminates a network service and the constituent VNFs.
Add VNF to Network Service	POST	/ocnso/1.1/ns/*networkServiceId*/vnfs	Adds VNFs to a network service.
Terminate VNF in a Network Service	DELETE	/ocnso/1.1/ns/*networkServiceId*/vnfs	Terminates a VNF in a network service.
Heal VNF	POST	/ocnso/1.1/vnf/*vnfId*/heal?action=*action*	Heals a VNF by rebooting or replacing the VM. Available values for the action parameter are: ■ replace ■ reboot
Configure VNF Capabilities	POST	/ocnso/1.1/vnf/configure	Configures the capabilities of a VNF service.
Upgrade VNF	PUT	/ocnso/1.1/vnf/*vnfId*/upgrade/*vnfVersion*	Upgrades the software image version of a VNF.
Get VNF Details	GET	/ocnso/1.1/vnf/*vnfId*	Returns details about a VNF.
Get VNF Status	GET	/ocnso/1.1/vnf/*vnfId*/status	Returns status information about a VNF.
Register PNF	POST	/ocnso/1.1/pnfs	Registers or creates a Physical Network Function (PNF) in inventory.
Update PNF	PUT	/ocnso/1.1/pnfs/*pnfId*	Updates an existing registered PNF.

Table 7–1 (Cont.) Network Service Orchestration RESTful API Resources

Request	Method	Resource	Description
Get PNFs	GET	/ocnso/1.1/pnfs?descriptorName=*pnfdName*	Returns the list of PNFs given the input descriptor name.
Get PNF Details	GET	/ocnso/1.1/pnfs/*pnfId*	Returns the PNF details given the input PNF identifier.
Unregister PNF	DELETE	/ocnso/1.1/pnfs/*pnfId*	Unregisters an existing PNF.
Register EMS	POST	/ocnso/1.1/ems	Registers or creates an Element Management System (EMS) in inventory.
Update EMS	PUT	/ocnso/1.1/ems/*emsId*	Updates an existing registered EMS.
Get EMSs	GET	/ocnso/1.1/ems?descriptorName=*emsdName*	Returns the list of EMSs given the input descriptor name.
Get EMS Details	GET	/ocnso/1.1/ems/*emsId*	Returns the EMS details given the input EMS identifier.
Unregister EMS	DELETE	/ocnso/1.1/ems/*emsId*	Unregisters an existing EMS.
Get Network Service Descriptors	GET	/ocnso/1.1/nsd	Returns a list of all network service descriptors that are deployed in Network Service Orchestration.
Get Network Service Descriptor Details	GET	/ocnso/1.1/nsd/*nsdName*	Returns details about a network service descriptor.
Get Network Service Descriptor VNFDs	GET	/ocnso/1.1/nsd/*nsdName*/vnfds	Returns a list of VNF descriptors in a network service descriptor.
Get Network Service Descriptor Flavors	GET	/ocnso/1.1/nsd/*nsdName*/flavors	Returns a list of all constituent service flavors that are defined for a network service descriptor.
Get VNF Descriptor Details	GET	/ocnso/1.1/vnfd/*vnfdName*	Returns details about a VNF descriptor.
Get VNF Descriptor Versions	GET	/ocnso/1.1/vnfd/*vnfdName*/versions	Returns a list of supported versions for a VNF descriptor.
Get VNF Descriptor Flavors	GET	/ocnso/1.1/vnfd/*vnfdName*/flavors	Returns a list of deployment flavors that are defined for a VNF descriptor.

RESTful API Responses

This section describes the fields and possible values for the responses returned from Network Service Orchestration RESTful API calls.

Table 7–2 describes the fields that are contained in a response.

Table 7–2 Fields in the Response

Response Field	Description
status	This field contains the status of the request processing. The following are the valid values: ■ SUCCESS ■ ACCEPTED ■ FAILURE
code	This field contains the HTTP status code value. Table 7–3 describes the possible values.
message	This optional field contains a high-level error message and is present only for a failed request.
data	This optional field contains the response information from the request and it only exists for successful requests. This field may not be relevant to all requests.

Table 7–2 (Cont.) Fields in the Response

Response Field	Description
exception	This optional field contains information when an exception occurs for a failed request.
errors	This optional field contains an array of information when an error occurs for a failed request.
warnings	This optional field contains an array of information when a warning occurs for a failed request.

Example 7–1 shows a success response for a network service instantiation request. The data portion contains the operation-specific information and message about the successful request.

Example 7–1 Sample Response Snippet on Success

```
{
  "status": "SUCCESS",
  "code": "202",
  "message": "[INV-430902] Network Service instantiation is in progress.",
  "data": {
    "id": "1125001",
    "descriptorName": "NPaaS_NSD",
    "name": "NSO_NPassService_1",
    "status": "PENDING",
    "businessInteractionId": "1275003",
    "businessInteractionStatus": "IN_PROGRESS"
     .
     .
     .
  }
}
```

Example 7–2 shows a response to a failed request.

Example 7–2 Sample Response on Failure

```
{
  "status": "FAILURE",
  "code": "500",
  "message": "No Network service exists with provided Network Service id 100.",
  "errors": [],
  "exception": {
    "code": "INV-430355",
    "message": "No Network service exists with provided Network Service id 100."
  }
}
```

Table 7–3 describes the HTTP response status codes for the RESTful API resources.

Table 7–3 HTTP Response Status Codes for the RESTful API Resources

Response Code	Description
200 OK	The request is successful. The information returned in the response depends on the method used in the request. For example: ■ GET returns one or more entities corresponding to the requested resource. ■ POST returns an entity describing or containing the result of the action.
202 Accepted	The request has been accepted for processing, but processing has not yet started. The request might or might not eventually be acted upon; it might be disallowed when the request is processed.
400 Bad Request	The request could not be understood by the server due to incorrect syntax. Correct the syntax before repeating the request.
401 Unauthorized	The request could not be authorized by the server due to invalid authentication or the absence of a valid user name and password. All requests require valid user authentication. See "About the Network Service Orchestration RESTful APIs" for more information.
404 Not Found	The server has not found a matching request or URI.
500 Internal Server Error	The server encountered an unexpected condition that prevented it from fulfilling the request.

Sample Requests and Responses

The following sections provide sample JSON requests and responses for the Network Service Orchestration RESTful API resources.

Register VIM

Registers details about the VIM with Network Service Orchestration such as the following:

- Host IP address
- Port
- User name
- Password

This resource results in the creation of a VIM.

Method

POST

URL

http://*nso_host*:*port*/ocnso/1.1/vim

Sample Request

```
{
  "id": "VimID_1",
  "name": "OpenStack VIM-1",
  "host": "192.0.2.251",
  "port": "12345",
  "userName": "nso",
  "pswd": "password",
```

```
  "projectName": "test",
  "domainName": "default",
  "version": "3",
  "sslEnabled": "false",
  "type": "OpenStack",
  "cpuOvercommitRatio": "15",
  "memoryOvercommitRatio": "1.5",
  "diskOvercommitRatio": "1.0"
}
```

Sample Response

```
{
  "status": "SUCCESS",
  "code": "200",
  "message": "[INV-430914] OpenStack VIM-1 is successfully registered with NSO."
}
```

Discover VIM Resources

Discovers the resources that are available on the VIM. For example, it creates the following resources as custom objects:

- Availability zones
- Flavors
- Hosts
- Virtual Data Center (VDC)

Method

POST

URL

http://*nso_host*:*port*/ocnso/1.1/vim/*vimId*/discovery?infoLevel=*vimInformation*

where:

- *vimId* is the identifier of the VIM whose resources you want to discover
- *vimInformation* is the level of information about the VIM that you want to receive in the response. The values for **infoLevel** are:
  - **summary**: Include a summary of the VIM resources.
  - **details**: Include complete details about all the VIM resources.

Sample Request

This API does not require any request parameters.

Sample Response

This is a sample response when the **infoLevel** parameter in the URL is set to **summary**:

```
{
   "status": "SUCCESS",
   "code": "200",
   "data": {
      "summary": {
         "Number of Subnets": 80,
```

```
         "Number of Flavors": 26,
         "Number of Hosts": 4,
         "Number of Networks": 80,
         "Number of Zones": 3
      }
   }
}
```

The following is a sample response when the **infoLevel** parameter in the URL is set to **details**:

```
{
  "data": {
    "availabilityZones": [
      {
        "zone": "CustomerTermination",
        "hosts": [
          "compute2"
        ]
      },
      {
        "zone": "nova",
        "hosts": [
          "compute4"
        ]
      }
    ],
    "networks": [
      {
        "network": "ext-net",
        "subnets": [
          "ext-subnet"
        ]
      },
      {
        "network": "QANetwork",
        "subnets": [
          "QASubnet"
        ]
      }
    ],
    "flavors": [
      "m1.tiny",
      "m1.small",
      "m1.large"
    ]
  }
}
```

Update VIM

Updates details about an existing VIM in Network Service Orchestration. For example, you can update the following attributes:

- Host IP address
- Port
- User name
- Password

- Project name

The update persists the new attribute values to the inventory database.

Method
PUT

URL
http://*nso_host*:*port*/ocnso/1.1/vim/*vimId*

where *vimId* is the identifier of the VIM that you want to update.

Sample Request
```
{
   "host": "192.0.2.252",
   "port": "12345",
   "userName": "nso",
   "pswd": "password",
   "projectName": "test",
   "domainName": "default",
   "version": "3",
   "sslEnabled": "false",
   "cpuOvercommitRatio": "15",
   "memoryOvercommitRatio": "1.5",
   "diskOvercommitRatio": "1.0"
}
```

Sample Response
```
{
   "status": "SUCCESS",
   "code": "200",
   "message": "[INV-430921] OpenStack VIM-2 updated successfully."
}
```

Get VIM Details

Retrieves the details of a VIM that is registered in inventory.

Method
GET

URL
http://*nso_host*:*port*/ocnso/1.1/vim/*vimId*

where *vimId* is the identifier of the VIM.

Sample Response
```
{
   "status": "SUCCESS",
   "code": "200",
   "data": {
      "id": "OpenStack",
      "name": "OpenStack",
      "host": "192.0.2.249",
      "port": "8001",
      "userName": "admin",
      "pswd": "password",
```

```
      "projectName": "admin",
      "domainName": "default",
      "type": "OpenStack",
      "version": "3",
      "sslEnabled": false,
      "cpuOvercommitRatio": "15",
      "memoryOvercommitRatio": "1.5",
      "diskOvercommitRatio": "1.0"
   }
}
```

Instantiate Network Service

Creates a network service, the related resources, and starts the VNFs in the network service. This resource can optionally include PNFs. See the additional sample request and response including PNFs.

Method

POST

URL

http://*nso_host*:*port*/ocnso/1.1/ns

Sample Request with VNFs

```
{
   "name": "NSO_NPassService_1",
   "descriptorName": "NPaaS_NSD",
   "flavorName": "Checkpoint",
   "vnfs": [
      {
         "name": "NSO_CheckPointVNF_1",
         "flavorName": "checkpoint",
         "descriptorName": "Checkpoint_NG_FW_VNFD",
         "version": "1.0"
      }
   ],
   "endPoints": [
      {
         "name": "Test1103",
         "parameters": [
            {
               "name": "ipAddress",
               "value": "192.0.2.250"
            },
            {
               "name": "vlanId",
               "value": "101"
            },
            {
               "name": "serviceLocation",
               "value": "Chicago 04"
            }
         ]
      }
   ]
}
```

Sample Response with VNFs


```
{
  "status": "SUCCESS",
  "code": "202",
  "message": "[INV-430902] Network Service instantiation is in progress.",
  "data": {
    "id": "15",
    "name": "NSO_NPassService_1",
    "descriptorName": "NPaaS_NSD",
    "status": "PENDING",
    "businessInteractionId": "30",
    "businessInteractionStatus": "IN_PROGRESS",
    "serviceDeploymentFlavorName": "Juniper",
    "parameters": [
      {
        "name": "vimId",
        "value": "ONAP21"
      },
      {
        "name": "dataCenterName",
        "value": "ONAP21"
      }
    ],
    "networks": [
      {
        "name": "15_Data_IN(ONAP21)",
        "status": "PENDING_REFERENCE"
      },
      {
        "name": "15_Data_OUT(ONAP21)",
        "status": "PENDING_REFERENCE"
      },
      {
        "name": "nfvo-poc3-mgmt(ONAP21)",
        "status": "PENDING_REFERENCE"
      }
    ],
    "endpoints": [
      {
        "id": "14",
        "name": "NSO_SGPcnsmr2",
        "descriptorName": "NSSubscriber",
        "status": "Pending Reference"
      }
    ],
    "vnfs": [
      {
        "id": "2",
        "name": "NSO_NPassService_VNF1",
        "status": "Pending Assign",
        "descriptorName": "Juniper_vSRX_VNFD",
        "version": "1.0",
        "imageName": "vsrx-12.1X47-D20.7-npaas-v0.3",
        "availabilityZoneName": "CustomerTermination",
        "dataCenterName": "ONAP21",
        "deploymentFlavorInfo": {
          "name": "vsrx.small",
          "vcpus": 2,
          "memory": "2 GB",
          "disk": "20 GB"
```

```
        }
      }
    ]
  }
}
```

Sample Request with VNFs and PNFs

```
{
  "name": "NSO_NPassService_2_with_PNFs",
  "descriptorName": "ResidentialGateway_NSD",
  "flavorName": "RGW",
  "vnfs": [
    {
      "name": "NSO_NPassService_VNF",
      "flavorName": "vsrx.small",
      "descriptorName": "Juniper_vSRX_VNFD",
      "version": "1.0"
    }
  ],
  "pnfs": [
    {
      "id": "675001",
      "descriptorName": "Cisco_xRV_PNFD"
    }
  ],
  "endPoints": [
    {
      "name": "NSO_EP_2",
      "parameters": [
        {
          "name": "ipAddress",
          "value": "192.0.2.231"
        },
        {
          "name": "vlanId",
          "value": "101"
        },
        {
          "name": "serviceLocation",
          "value": "CityPQ03"
        }
      ]
    }
  ]
}
```

Sample Response with VNFs and PNFs

```
{
  "status": "SUCCESS",
  "code": "202",
  "message": "[INV-430902] Network Service instantiation is in progress.",
  "data": {
    "id": "300085",
    "name": "NSO_NPassService_2_with_PNFs",
    "descriptorName": "ResidentialGateway_NSD",
    "status": "PENDING",
    "businessInteractionId": "150091",
    "businessInteractionStatus": "IN_PROGRESS",
    "serviceDeploymentFlavorName": "RGW",
```

```
"parameters": [
  {
    "name": "vimId",
    "value": "OpenStack"
  },
  {
    "name": "dataCenterName",
    "value": "OpenStack"
  }
],
"networks": [
  {
    "name": "300085_Data1(OpenStack)",
    "status": "PENDING_REFERENCE"
  },
  {
    "name": "nfvo-poc3-mgmt(OpenStack)",
    "status": "PENDING_REFERENCE"
  },
  {
    "name": "300085_Data2(OpenStack)",
    "status": "PENDING_REFERENCE"
  }
],
"endpoints": [
  {
    "id": "450002",
    "name": "NSO_EP_2",
    "descriptorName": "NSSubscriber",
    "status": "Pending Reference"
  }
],
"vnfs": [
  {
    "id": "675029",
    "name": "NSO_NPassService_VNF",
    "status": "Pending Assign",
    "descriptorName": "Juniper_vSRX_VNFD",
    "version": "1.0",
    "imageName": "vsrx-12.1X47-D20.7-npaas-v0.3",
    "availabilityZoneName": "nova",
    "dataCenterName": "OpenStack",
    "deploymentFlavorInfo": {
      "name": "vsrx.small",
      "vcpus": 2,
      "memory": "2 GB",
      "disk": "20 GB"
    },
    "securityGroups": "sgall,default"
  }
],
"pnfs": [
  {
    "id": "675001",
    "name": "xtv862",
    "descriptorName": "Cisco_xRV_PNFD",
    "ipAddress": "192.0.2.155",
    "userName": "user",
    "pswd": "password",
    "sslEnabled": false,
```

```
        "sshKey": "",
        "parameters": [
          {}
        ]
      }
    ]
  }
}
```

Scale Network Service

Scales a network service by either cloning a VNF instance or removing an existing VNF.

Method

POST

URL

http://*nso_host*:*port*/ocnso/1.1/ns/*networkServiceId*/scale/*vnfId*?action=*action*

where:

- *networkServiceId* is the required identifier of the network service for the VNF.
- *vnfId* is the required identifier of the VNF to scale.
- *action* is the scale action. Specify one of the following values:
  - **scale-in**: Removes the specified VNF.
  - **scale-out**: Clones the specified VNF. This is the default action if the action is not specified.

Sample Request

This API does not require a request body.

Sample Response

```
{
  "status": "SUCCESS",
  "code": "202",
  "message": "[INV-431002] VNF scale operation is in progress.",
  "data": {
    "id": "18",
    "name": "NSO_NPassService_1",
    "descriptorName": "NPaaS_NSD",
    "status": "IN_SERVICE",
    "vnfs": [
      {
        "id": "5",
        "name": "Juniper_vSRX_VNFD_1471441956301",
        "status": "Pending Assign",
        "descriptorName": "Juniper_vSRX_VNFD"
      }
    ]
  }
}
```

Get Network Services

Retrieves the list of active network services that are related to the input network service descriptor.

Method

GET

URL

http://*nso_host*:*port*/ocnso/1.1/ns?nsdName=*nsdName*

where *nsdName* is the name of the network service descriptor.

Sample Response

```
{
   "status": "SUCCESS",
   "code": "200",
   "data": [
      {
         "id": "1050001",
         "descriptorName": "NPaaS_NSD",
         "name": "NSO_NPassService_1",
         "status": "IN_SERVICE"
      },
      {
         "id": "1050005",
         "descriptorName": "NPaaS_NSD",
         "name": "NSO_NPassService_2",
         "status": "IN_SERVICE"
      },
      {
         "id": "1125001",
         "descriptorName": "NPaaS_NSD",
         "name": "NSO_NPassService_3",
         "status": "IN_SERVICE"
      }
   ]
}
```

Get Network Service Details

Retrieves the details of a network service.

Method

GET

URL

http://*nso_host*:*port*/ocnso/1.1/ns/*networkServiceId*

where *networkServiceId* is the identifier of the network service whose details you want to retrieve.

Sample Response

```
{
  "status": "SUCCESS",
  "code": "200",
  "data": {
```

```
    "id": "22",
    "name": "NPassService_Juniper55_Service",
    "descriptorName": "NPaaS_NSD",
    "status": "IN_SERVICE",
    "businessInteractionId": "47",
    "businessInteractionStatus": "COMPLETED",
    "serviceDeploymentFlavorName": "Juniper",
    "parameters": [
      {
        "name": "vimId",
        "value": "ONAP21"
      },
      {
        "name": "dataCenterName",
        "value": "ONAP21"
      }
    ],
    "networks": [
      {
        "name": "22_Data_OUT(ONAP21)",
        "status": "REFERENCED"
      },
      {
        "name": "22_Data_IN(ONAP21)",
        "status": "REFERENCED"
      },
      {
        "name": "nfvo-poc3-mgmt(ONAP21)",
        "status": "REFERENCED"
      }
    ],
    "endpoints": [
      {
        "id": "14",
        "name": "NSO_SGPcnsmr2",
        "descriptorName": "NSSubscriber",
        "status": "Referenced"
      }
    ],
    "vnfs": [
      {
        "id": "6",
        "name": "Juniper_VNF_6",
        "status": "Assigned",
        "descriptorName": "Juniper_vSRX_VNFD"
      }
    ]
  }
}
```


Get Network Service VNFs

Retrieves the details about the VNFs in a network service.

Method

GET

URL

http://*nso_host*:*port*/ocnso/1.1/ns/*networkServiceId*/vnfs

where *networkServiceId* is the identifier of the network service for the VNFs.

Sample Response


```
{
  "status": "SUCCESS",
  "code": "200",
  "data": {
    "id": "22",
    "name": "29_1.3_AMns_Service",
    "descriptorName": "NPaaS_NSD",
    "status": "IN_SERVICE",
    "vnfs": [
      {
        "id": "6",
        "name": "VNF_1",
        "status": "Assigned",
        "descriptorName": "Juniper_vSRX_VNFD",
        "vmStatus": "ACTIVE",
        "version": "1.0",
        "externalId": "6e941ad7-c45f-4dfb-bf26-2bb50234fadc",
        "imageName": "vsrx-12.1X47-D20.7-npaas-v0.3",
        "availabilityZoneName": "CustomerTermination",
        "dataCenterName": "ONAP21",
        "host": "b73c2706400313820fd7db7086d90025ed6c2ab95a558d589c578c0a",
        "deploymentFlavorInfo": {
          "name": "vsrx.small",
          "vcpus": 2,
          "memory": "2 GB",
          "disk": "20 GB"
        },
        "connectionPoints": [
          {
            "id": "6-1",
            "name": "CP01",
            "ipAddress": {
              "address": "192.0.2.231",
              "network": "22_Data_IN(ONAP21)",
              "externalId": "c83177ff-632e-4a19-97a7-3539c96c3d4c"
            }
          },
          {
            "id": "6-2",
            "name": "CP02",
            "ipAddress": {
              "address": "192.0.2.232",
              "network": "22_Data_OUT(ONAP21)",
              "externalId": "f937cdc6-11bb-42b7-9b9e-f9685f7b71e2"
            }
          },
          {
            "id": "6-3",
            "name": "CP03",
            "ipAddress": {
              "address": "192.0.2.233",
              "network": "nfvo-poc3-mgmt(ONAP21)",
              "externalId": "c5272351-905d-47ca-b7ac-0f18f01d724d"
            }
          }
        ],
        "capabilities": [
```

```
          {
            "serviceId": "24",
            "serviceName": "PRAYAVAR_Aug17_06_Capabilities_24",
            "serviceStatus": "PENDING_ASSIGN",
            "serviceDescriptorName": "Juniper_vSRX_Capabilities_
ServiceDescriptor",
            "businessInteractionId": "50"
          }
        ]
      }
    ]
  }
}
```

Get Network Service Networks

Retrieves the details about the networks within a network service.

Method

GET

URL

http://*nso_host*:*port*/ocnso/1.1/ns/*networkServiceId*/networks

where *networkServiceId* is the identifier of the network service.

Sample Response

```
{
  "status": "SUCCESS",
  "code": "200",
  "data": {
    "id": "750001",
    "name": "NSO_NetworkService",
    "descriptorName": "NPaaS_NSD",
    "status": "IN_SERVICE",
    "networks": [
      {
        "name": "750001_Data_IN(OpenStack)",
        "id": "750001_Data_IN(OpenStack)",
        "externalId": "b246b28c-9e98-490f-b046-4601da09a28a",
        "status": "REFERENCED",
        "subnets": [
          {
            "startIP": "192.0.2.209",
            "prefix": "27",
            "externalId": "d8bb391a-f216-4fc2-8465-fec0cccbb5ef"
          }
        ]
      },
      {
        "name": "nfvo-poc3-mgmt(OpenStack)",
        "id": "nfvo-poc3-mgmt(OpenStack)",
        "externalId": "109ae4cf-3cea-4729-a24f-957c4ed6d3c6",
        "status": "REFERENCED",
        "subnets": [
          {
            "startIP": "192.0.2.208",
            "prefix": "24",
```

```
            "externalId": "fb791563-7c8b-454c-a1eb-87399e6837dc"
          }
        ]
      },
      {
        "name": "750001_Data_OUT(OpenStack)",
        "id": "750001_Data_OUT(OpenStack)",
        "externalId": "ba96e7e3-4080-49c8-aa4d-e9af6e5ab2e9",
        "status": "REFERENCED",
        "subnets": [
          {
            "startIP": "192.0.2.207",
            "prefix": "27",
            "externalId": "f88688ee-aecb-47fc-bb58-e48c6a277eb9"
          }
        ]
      }
    ]
  }
}
```

Get Network Service End Points

Retrieves the details about the endpoints in a network service.

Method

GET

URL

http://*nso_host*:*port*/ocnso/1.1/ns/*networkServiceId*/endpoints

where *networkServiceId* is the identifier of the network service.

Sample Response

```
{
  "status": "SUCCESS",
  "code": "200",
  "data": {
    "id": "22",
    "name": "NPaaS_Service",
    "descriptorName": "NPaaS_NSD",
    "status": "IN_SERVICE",
    "endpoints": [
      {
        "id": "14",
        "name": "NSO_SGPcnsmr2",
        "descriptorName": "NSSubscriber",
        "ipAddress": "192.0.2.218",
        "forwardingGraphDescriptorName": "Data",
        "status": "Referenced",
        "serviceLocation": "MTRLPQ03"
      }
    ]
  }
}
```

Get Network Service Status

Retrieves the status information for a network service.

Method

GET

URL

http://*nso_host*:*port*/ocnso/1.1/ns/*networkServiceId*/status

where *networkServiceId* is the identifier of the network service whose status information you want to retrieve.

Sample Response

```
{
   "status": "SUCCESS",
   "code": "200",
   "data": {
      "id": "22",
      "name": "Sample NS",
      "descriptorName": "NPaaS_NSD",
      "status": "IN_SERVICE",
      "businessInteractionId": "47",
      "businessInteractionStatus": "COMPLETED",
      "networks": [
         {
            "name": "22_Data_OUT(ONAP21)",
            "status": "REFERENCED"
         },
         {
            "name": "22_Data_IN(ONAP21)",
            "status": "REFERENCED"
         },
         {
            "name": "nfvo-poc3-mgmt(ONAP21)",
            "status": "REFERENCED"
         }
      ],
      "endpoints": [
         {
            "id": "14",
            "name": "NSO_smr2",
            "descriptorName": "NSSubscriber",
            "status": "Referenced"
         }
      ],
      "vnfs": [
         {
            "id": "6",
            "name": "VNF-1_06",
            "status": "Assigned",
            "descriptorName": "Juniper_vSRX_VNFD",
            "vmStatus": "ACTIVE",
            "externalId": "6e941ad7-c45f-4dfb-bf26-2bb50234fadc"
         }
      ]
   }
}
```

Terminate Network Service

Terminates a network service. This API undeploys the constituent VNFs in the network service and releases all the resources that were allocated to the service.

Method

DELETE

URL

http://*nso_host*:*port*/ocnso/1.1/ns/*networkServiceId*

where *networkServiceId* is the identifier of the network service that you want to terminate.

Sample Request

This API does not require a request body. Specify the network service identifier in the URL.

Sample Response


```
{
   "status": "SUCCESS",
   "code": "202",
   "message": "[INV-430907] Network Service termination is in progress.",
   "data": {
      "id": "750001",
      "name": "NSO_NPassService_1",
      "descriptorName": "NPaaS_NSD",
      "status": "PENDING_DISCONNECT",
      "businessInteractionId": "1275001",
      "businessInteractionStatus": "IN_PROGRESS",
      "serviceDeploymentFlavorName": "Juniper",
      "parameters": [
         {
            "name": "vimId",
            "value": "OpenStack"
         },
         {
            "name": "flavorName",
            "value": "Juniper"
         },
         {
            "name": "dataCenterName",
            "value": "OpenStack"
         }
      ],
      "networks": [
         {
            "name": "750001_Data_IN(OpenStack)",
            "status": "PENDING_UNREFERENCE"
         },
         {
            "name": "nfvo-poc3-mgmt(OpenStack)",
            "status": "PENDING_UNREFERENCE"
         },
         {
            "name": "750001_Data_OUT(OpenStack)",
            "status": "PENDING_UNREFERENCE"
         }
```

```
      ],
      "vnfs": [
      {
         "id": "675001",
         "name": "Juniper VNF_10",
         "status": "Pending Unassign",
         "descriptorName": "Juniper_vSRX_VNFD",
         "vmStatus": "ACTIVE",
         "version": "1.0",
         "externalId": "db788029-e3cc-4651-bab7-da2511ff6e9b",
         "imageName": "vsrx-12.1X47-D20.7-npaas-v0.3",
         "availabilityZoneName": "CustomerTermination",
         "dataCenterName": "ONAP21",
         "host": "b73c2706400313820fd7db7086d90025ed6c2ab95a558d589c578c0a",
         "deploymentFlavorInfo": {
            "name": "vsrx.small",
            "vcpus": 2,
            "memory": "2 GB",
            "disk": "20 GB"
            }
         }
      ],
      "endpoints": [
         {
            "id": "1",
            "name": "NSO_EP_2",
            "descriptorName": "NSSubscriber",
            "status": "PENDING_UNREFERENCE"
         }
      ]
   }
}
```


Add VNF to Network Service

Adds new VNFs to an existing network service.

Method

POST

URL

http://*nso_host*:*port*/ocnso/1.1/ns/*networkServiceId*/vnfs

where *networkServiceId* is the identifier of the existing network service to add the VNF to.

Sample Request


```
[
   {
      "name": "VNF1",
      "flavorName": "vsrx.small",
      "descriptorName": "Juniper_vSRX_VNFD",
      "version": "1.0"
   }
]
```


Sample Response


```
{
```

```
   "status": "SUCCESS",
   "code": "202",
   "message": "[INV-430903] Adding VNF to Network Service is in progress.",
   "data": {
      "id": "750001",
      "name": "NSO_NPassService_1",
      "descriptorName": "NPaaS_NSD",
      "status": "IN_SERVICE",
      "vnfs": [
         {
            "id": "675004",
            "name": "VNF1",
            "status": "Pending Assign",
            "descriptorName": "Juniper_vSRX_VNFD"
         }
      ]
   }
}
```

Terminate VNF in a Network Service

Terminate a VNF in an existing network service and undeploys it in the VIM.

Method

DELETE

URL

http://*nso_host*:*port*/ocnso/1.1/ns/*networkServiceId*/vnfs

where *networkServiceId* is the identifier of the existing network service for the VNF.

Sample Request

```
[
  {
    "id": "675004"
  }
]
```

The input **id** is the identifier of the VNF to terminate, represented as a logical device in UIM.

Sample Response

```
{
   "status": "SUCCESS",
   "code": "202",
   "message": "[INV-430904] Deleting VNF from Network Service is in progress.",
   "data": {
      "id": "450002",
      "name": "NSO_NPassService_1",
      "descriptorName": "NPaaS_NSD",
      "status": "IN_SERVICE",
      "vnfs": [
         {
            "id": "675004",
            "name": "VNF1",
            "status": "Pending Unassign",
            "descriptorName": "Juniper_vSRX_VNFD"
         }
```

```
      ]
   }
}
```

Heal VNF

Heals a VNF by either rebooting or replacing the virtual machine on which the VNF is deployed.

Method

POST

URL

http://*nso_host*:*port*/ocnso/1.1/vnf/*vnfId*/heal?action=*action*

where:

- *vnfId* is the required identifier of the VNF.
- *action* is the optional action that you want to perform on the VNF. Specify one of the following values:
  - **reboot**: Reboots the virtual machine on which the VNF is deployed. This is the default action if the action is not specified.
  - **replace**: Undeploys the existing VNF and deploys a new VNF with the same attributes.

Sample Request

This API does not require a request body.

Sample Response

The following is a sample response when the **action** parameter in the URL is set to **reboot**:

```
{
  "status": "SUCCESS",
  "code": "200",
  "message": "VNF has been rebooted successfully.",
  "data": {
    "id": "4",
    "name": "VNF_05",
    "status": "Assigned",
    "descriptorName": "Juniper_vSRX_VNFD",
    "vmStatus": "REBOOT",
    "version": "1.0",
    "externalId": "8d303b96-4f21-4172-aeaf-1b7ac36970e0",
    "imageName": "vsrx-12.1X47-D20.7-npaas-v0.3",
    "availabilityZoneName": "CustomerTermination",
    "dataCenterName": "ONAP21",
    "host": "b73c2706400313820fd7db7086d90025ed6c2ab95a558d589c578c0a",
    "deploymentFlavorInfo": {
      "name": "vsrx.small",
      "vcpus": 2,
      "memory": "2 GB",
      "disk": "20 GB"
    },
    "connectionPoints": [
      {
```

```
        "id": "4-1",
        "name": "CP01",
        "ipAddress": {
          "address": "192.0.2.168",
          "network": "18_Data_IN(ONAP21)",
          "externalId": "2c901de9-c868-401d-9238-4a880dd7c266"
        }
      },
      {
        "id": "4-2",
        "name": "CP02",
        "ipAddress": {
          "address": "192.0.2.34",
          "network": "18_Data_OUT(ONAP21)",
          "externalId": "c7689e70-059a-437f-86d0-a5f773489dea"
        }
      },
      {
        "id": "4-3",
        "name": "CP03",
        "ipAddress": {
          "address": "192.0.2.35",
          "network": "nfvo-poc3-mgmt(ONAP21)",
          "externalId": "7dce6c90-7aae-4c7e-829e-b8a35f806398"
        }
      }
    ]
  }
}
```


The following is a sample response when the **action** parameter in the URL is set to **replace**:


```
{
  "status": "SUCCESS",
  "code": "202",
  "message": "Replace VNF is in progress.",
  "data": {
    "id": "5",
    "name": "Juniper_vSRX_VNFD_1471",
    "status": "Assigned",
    "descriptorName": "Juniper_vSRX_VNFD",
    "vmStatus": "ACTIVE",
    "version": "1.0",
    "externalId": "5159e90e-4166-4d1e-879f-da1109e8073d",
    "imageName": "vsrx-12.1X47-D20.7-npaas-v0.3",
    "availabilityZoneName": "CustomerTermination",
    "dataCenterName": "ONAP21",
    "host": "b73c2706400313820fd7db7086d90025ed6c2ab95a558d589c578c0a",
    "deploymentFlavorInfo": {
      "name": "vsrx.small",
      "vcpus": 2,
      "memory": "2 GB",
      "disk": "20 GB"
    },
    "connectionPoints": [
      {
        "id": "5-1",
        "name": "CP01",
        "ipAddress": {
          "address": "192.0.2.87",
```

```
          "network": "18_Data_IN(ONAP21)",
          "externalId": "b4fe843c-e122-4f44-90ec-6853f6a939de"
        }
      },
      {
        "id": "5-2",
        "name": "CP02",
        "ipAddress": {
          "address": "192.0.2.44",
          "network": "18_Data_OUT(ONAP21)",
          "externalId": "64409652-5f29-45ae-a26e-7170598409a7"
        }
      },
      {
        "id": "5-3",
        "name": "CP03",
        "ipAddress": {
          "address": "192.0.2.4",
          "network": "nfvo-poc3-mgmt(ONAP21)",
          "externalId": "ba07082a-d8f7-48f0-9e7a-d38e8fcd9549"
        }
      }
    ]
  }
}
```


Configure VNF Capabilities

Configures the capabilities of a VNF in a network service.

Method

POST

URL

http://*nso_host*:*port*/ocnso/1.1/vnf/configure

Sample Request


```
{
   "id": "12345",
   "capabilities": [
      {
         "name": "WebFilter",
         "parameters": [
            {
               "name": "Id",
               "value": "WebFilter-vnfID"
            },
            {
               "name": "Action",
               "value": "Create"
            }
         ],
         "configuration": {
            "items": [
               {
                  "name": "WebFilterRuleset",
                  "parameters": [
                     {
```

```
                        "name": "Id",
                        "value": "WebFilter_RulesetvnfID"
                     },
                     {
                        "name": "Action",
                        "value": "Add"
                     }
                  ],
                  "items": [
                     {
                        "name": "WebFilterRule",
                        "parameters": [
                           {
                              "name": "Id",
                              "value": "WebFilter_Rule_UniqueNumber"
                           },
                           {
                              "name": "Action",
                              "value": "Add"
                           },
                           {
                              "name": "BlockURL",
                              "value": "www.example.com"
                           }
                        ]
                     }
                  ]
               }
            ]
         }
      }
   ]
}
```

Sample Response

```
{
  "status": "SUCCESS",
  "code": "200",
  "message": "VNF Configuration is in progress",
  "data": {
    "id": "123",
    "name": "Sample_VNF_Name_123",
    "status": "Assigned",
    "descriptorName": "Juniper_vSRX_VNFD",
    "capabilities": [
      {
        "serviceId": "24",
        "serviceName": "Service_Capabilities_24",
        "serviceStatus": "PENDING_ASSIGN",
        "serviceDescriptorName": "Juniper_vSRX_Capabilities_ServiceDescriptor"
      }
    ]
  }
}
```

Upgrade VNF

Upgrades the software version of a VNF image in a network service.

Method
PUT

URL
http://*nso_host*:*port*/ocnso/1.1/vnf/*vnfId*/upgrade/*vnfVersion*

where:

- *vnfId* is the required identifier of the VNF whose image you want to upgrade.
- *vnfVersion* is the required version number of the VNF image that you want to upgrade to. The version number must already exist in the VNF descriptor file.

Sample Request
This API does not require a request body. Specify the VNF identifier and the VNF version number you want to upgrade to in the URL.

Sample Response
```
{
   "status": "SUCCESS",
   "code": "202",
   "data": {
      "id": "675001",
      "businessInteractionId": "975001"
   }
}
```

Get VNF Details

Retrieves the details about a VNF given the input VNF identifier.

Method
GET

URL
http://*nso_host*:*port*/ocnso/1.1/vnf/*vnfId*

where *vnfId* is the identifier of the VNF.

Sample Response
```
{
   "status": "SUCCESS",
   "code": "200",
   "data": {
      "id": "6",
      "name": "VNF_17_06",
      "status": "Assigned",
      "descriptorName": "Juniper_vSRX_VNFD",
      "vmStatus": "ACTIVE",
      "version": "1.0",
      "externalId": "6e941ad7-c45f-4dfb-bf26-2bb50234fadc",
      "imageName": "vsrx-12.1X47-D20.7-npaas-v0.3",
      "availabilityZoneName": "CustomerTermination",
      "dataCenterName": "ONAP21",
      "host": "b73c2706400313820fd7db7086d90025ed6c2ab95a558d589c578c0a",
      "deploymentFlavorInfo": {
         "name": "vsrx.small",
```

```
            "vcpus": 2,
            "memory": "2 GB",
            "disk": "20 GB"
         },
         "connectionPoints": [
            {
               "id": "6-1",
               "name": "CP01",
               "ipAddress": {
                  "address": "192.0.2.240",
                  "network": "22_Data_IN(ONAP21)",
                  "externalId": "c83177ff-632e-4a19-97a7-3539c96c3d4c"
               }
            },
            {
               "id": "6-2",
               "name": "CP02",
               "ipAddress": {
                  "address": "192.0.2.241",
                  "network": "22_Data_OUT(ONAP21)",
                  "externalId": "f937cdc6-11bb-42b7-9b9e-f9685f7b71e2"
               }
            },
            {
               "id": "6-3",
               "name": "CP03",
               "ipAddress": {
                  "address": "192.0.2.242",
                  "network": "nfvo-poc3-mgmt(ONAP21)",
                  "externalId": "c5272351-905d-47ca-b7ac-0f18f01d724d"
               }
            }
         ],
         "capabilities": [
            {
               "serviceId": "24",
               "serviceName": "Capabilities_24",
               "serviceStatus": "PENDING_ASSIGN",
               "serviceDescriptorName": "Juniper_vSRX_Capabilities_
ServiceDescriptor",
               "businessInteractionId": "50"
            }
         ]
      }
}
```


Get VNF Status

Retrieves the status information of a VNF and the VIM given the input VNF identifier.

Method

GET

URL

http://*nso_host*:*port*/ocnso/1.1/vnf/*vnfId*/status

where *vnfId* is the identifier of the VNF.

Sample Response

```
{
   "status": "SUCCESS",
   "code": "200",
   "data": {
      "id": "75085",
      "name": "ChkptVNF_CP_B253",
      "status": "Assigned",
      "descriptorName": "Checkpoint_NG_FW_VNFD"
   }
}
```

Register PNF

Registers or creates a new PNF in inventory.

Method

POST

URL

http://*nso_host*:*port*/ocnso/1.1/pnfs

Sample Request

```
{
  "name": "xtv59",
  "descriptorName": "Cisco_xRV_PNFD",
  "description": "",
  "userName": "user",
  "pswd": "password",
  "sslEnabled": false,
  "sshKey": "7b:ab:75:32:9e:b6:6c:4b:29:dc",
  "ipAddress": "192.0.2.252",
  "management": {
    "id": "225001",
    "name": "ems5111",
    "mgmtInterface": "EMS",
    "descriptorName": "Cisco_xRV_EMS"
  },
  "parameters": []
}
```

Sample Response

```
{
  "status": "SUCCESS",
  "code": "200",
  "data": {
    "id": "525003",
    "name": "xtv598912",
    "descriptorName": "Cisco_xRV_PNFD",
    "description": "",
    "ipAddress": "192.0.2.252",
    "userName": "user",
    "pswd": "password",
    "sslEnabled": false,
    "sshKey": "7b:ab:75:32:9e:b6:6c:4b:29:dc",
    "parameters": [
```

```
      {}
    ],
    "management": {
      "id": "225001",
      "name": "ems5111",
      "mgmtInterface": "EMS",
      "descriptorName": "Cisco_xRV_EMS"
    }
  }
}
```

Update PNF

Updates an existing registered PNF. The update persists the new attribute values to the inventory database.

Method

PUT

URL

http://*nso_host*:*port*/ocnso/1.1/pnfs/*pnfId*

where *pnfId* is the identifier of the PNF that you want to update.

Sample Request

```
{
   "pswd": "password",
   "sshKey": "7b:ab:75:32:9e:b6:6c:4b:29:dc"
}
```

Sample Response

```
{
  "status": "SUCCESS",
  "code": "200",
  "data": {
    "id": "525003",
    "name": "xtv598912",
    "descriptorName": "Cisco_xRV_PNFD",
    "description": "",
    "ipAddress": "192.0.2.252",
    "userName": "user",
    "pswd": "password",
    "sslEnabled": false,
    "sshKey": "7b:ab:75:32:9e:b6:6c:4b:29:dc",
    "parameters": [
      {}
    ]
  }
}
```

Get PNFs

Retrieves the list of active PNFs that are related to the input PNF descriptor.

Method

GET

URL

http://*nso_host*:*port*/ocnso/1.1/pnfs?descriptorName=*pnfdName*

where *pnfdName* is the name of the PNF descriptor.

Sample Response


```
{
   "status": "SUCCESS",
   "code": "200",
   "data": [
      {
         "id": "22001",
         "name": "xtv0018",
         "descriptorName": "Cisco_xRV_PNFD",
         "ipAddress": "192.0.2.228",
         "userName": "user2",
         "pswd": "password",
         "sslEnabled": false,
         "sshKey": "",
         "parameters": [
            {}
         ],
         "management": {}
      },
      {
         "id": "300012",
         "name": "xtv592",
         "descriptorName": "Cisco_xRV_PNFD",
         "ipAddress": "192.0.2.226",
         "userName": "user8",
         "pswd": "password",
         "sslEnabled": false,
         "sshKey": "",
         "parameters": [
            {}
         ],
         "management": {
            "id": "75007",
            "name": "ems363",
            "mgmtInterface": "EMS",
            "descriptorName": "Cisco_xRV_EMS"
         }
      },
      {
         "id": "300016",
         "name": "xtv595",
         "descriptorName": "Cisco_xRV_PNFD",
         "ipAddress": "192.0.2.227",
         "userName": "user5",
         "pswd": "password",
         "sslEnabled": false,
         "sshKey": "",
         "parameters": [
            {}
         ],
         "management": {}
      }
   ]
}
```

Get PNF Details

Retrieves the details of a PNF given the PNF identifier.

Method

GET

URL

http://<host>:<port>/ocnso/1.1/pnfs/*pnfId*

where *pnfdId* is the identifier of the PNF that you want to retrieve.

Sample Response

```
{
   "status": "SUCCESS",
   "code": "200",
   "data": {
      "id": "525003",
      "name": "xtv59",
      "descriptorName": "Cisco_xRV_PNFD",
      "description": "",
      "ipAddress": "192.0.2.224",
      "userName": "user",
      "pswd": "password",
      "sslEnabled": false,
      "sshKey": "",
      "parameters": [
         {}
      ],
      "management": {
         "id": "225001",
         "name": "ems5111",
         "mgmtInterface": "EMS",
         "descriptorName": "Cisco_xRV_EMS"
      }
   }
}
```

Unregister PNF

Unregisters an existing PNF. The request deletes the PNF matching the input identifier value from the inventory database.

Method

DELETE

URL

http://*nso_host*:*port*/ocnso/1.1/pnfs/*pnfId*

where *pnfId* is the identifier of the PNF that you want to unregister.

Sample Request

This API does not require a request body.

Sample Response

```
{
   "status": "SUCCESS",
```

```
   "code": "200",
   "message": "[INV-430925] PNF 300007 is deleted successfully."
}
```

Register EMS

Register or create a new EMS in inventory.

Method

POST

URL

http://*nso_host*:*port*/ocnso/1.1/ems

Sample Request

```
{
   "name": "ems574",
   "descriptorName": "Cisco_xRV_EMS",
   "description": "Sample Description",
   "userName": "user",
   "pswd": "password",
   "ipAddress": "192.0.2.212",
   "port": "7001",
   "sslEnabled": false,
   "protocol": "REST"
}
```

Sample Response

```
{
   "status": "SUCCESS",
   "code": "200",
   "data": {
      "id": "375003",
      "name": "ems574",
      "descriptorName": "Cisco_xRV_EMS",
      "description": "Sample Description",
      "ipAddress": "192.0.2.212",
      "port": "7001",
      "userName": "user",
      "pswd": "password",
      "sslEnabled": false,
      "protocol": "REST",
      "parameters": [
         {}
      ]
   }
}
```

Update EMS

Updates the details of a registered EMS in inventory. The update persists the new attribute values to the inventory database.

Method

PUT

URL

http://*nso_host*:*port*/ocnso/1.1/ems/*emsId*

where *emsId* is the identifier of the EMS that you want to update.

Sample Request

```
{
   "description": "New EMS description",
   "userName": "sys_user_1",
   "pswd": "password"
}
```

Sample Response

```
{
  "status": "SUCCESS",
  "code": "200",
  "data": {
    "id": "375003",
    "name": "ems574236312",
    "descriptorName": "Cisco_xRV_EMS",
    "description": "New EMS description",
    "ipAddress": "192.0.2.212",
    "port": "7001",
    "userName": "sys_user_1",
    "pswd": "password",
    "sslEnabled": false,
    "protocol": "REST",
    "parameters": [
      {}
    ]
  }
}
```

Get EMSs

Retrieves the list of active EMSs that are related to the input EMS descriptor.

Method

GET

URL

http://*nso_host*:*port*/ocnso/1.1/ems?descriptorName=*emsdName*

where *emsdName* is the name of the EMS descriptor.

Sample Response

```
{
   "status": "SUCCESS",
   "code": "200",
   "data": [
      {
         "id": "225001",
         "name": "ems5111",
         "descriptorName": "Cisco_xRV_EMS",
         "description": "description",
         "ipAddress": "192.0.2.221",
         "port": "1234",
```

```
            "userName": "user",
            "pswd": "password",
            "sslEnabled": false,
            "protocol": "REST",
            "parameters": [
               {}
            ]
         },
         {
            "id": "375001",
            "name": "ems5742",
            "descriptorName": "Cisco_xRV_EMS",
            "description": "description",
            "ipAddress": "192.0.2.222",
            "port": "12345",
            "userName": "user",
            "pswd": "password",
            "sslEnabled": false,
            "protocol": "REST",
            "parameters": [
               {}
            ]
         }
      ]
}
```

Get EMS Details

Retrieves the details of an EMS given the EMS identifier.

Method

GET

URL

http://<host>:<port>/ocnso/1.1/ems/*emsId*

where *emsId* is the identifier of the EMS that you want to retrieve.

Sample Response

```
{
   "status": "SUCCESS",
   "code": "200",
   "data": {
      "id": "375003",
      "name": "ems5742",
      "descriptorName": "Cisco_xRV_EMS",
      "description": "description",
      "ipAddress": "192.0.2.220",
      "port": "1234",
      "userName": "user11",
      "pswd": "password",
      "sslEnabled": false,
      "protocol": "REST",
      "parameters": [
         {}
      ]
   }
}
```

Unregister EMS

Unregisters an existing EMS. The request deletes the EMS matching the input identifier value from the inventory database.

Method

DELETE

URL

http://*nso_host*:*port*/ocnso/1.1/ems/*emsId*

where *emsId* is the identifier of the EMS that you want to unregister.

Sample Request

This API does not require a request body.

Sample Response

```
{
  "status": "SUCCESS",
  "code": "200",
  "message": "[INV-430936] EMS 75003 is deleted successfully."
}
```

Get Network Service Descriptors

Retrieves a list of deployed network service descriptors.

Method

GET

URL

http://*nso_host*:*port*/ocnso/1.1/nsd

Sample Response

```
{
   "status": "SUCCESS",
   "code": "200",
   "data": [
      "NPaaS_NSD"
   ]
}
```

Get Network Service Descriptor Details

Retrieves details about a specified network service descriptor.

Method

GET

URL

http://*nso_host*:*port*/ocnso/1.1/nsd/*nsdName*

where *nsdName* is the name of the network service descriptor.

Sample Response


```
{
   "status": "SUCCESS",
   "code": "200",
   "data": {
      "referencedVnfds": [
         "Checkpoint_NG_FW_VNFD",
         "Juniper_vSRX_VNFD"
      ],
      "serviceDeploymentFlavors": [
         {
            "name": "Checkpoint",
            "constituentVNFDs": [
               {
                  "vnfd": {
                     "name": "Checkpoint_NG_FW_VNFD",
                     "vNetworkInterfaces": 0
                  },
                  "assuranceParameters": [
                     {
                        "name": "Low CPU Utilization",
                        "id": "cpu_util",
                        "condition": "eq",
                        "value": "0.0",
                        "action": "heal"
                     },
                     {
                        "name": "High CPU Utilization",
                        "id": "cpu_util",
                        "condition": "gt",
                        "value": "80.0",
                        "action": "scale"
                     }
                  ]
               }
            ]
         },
         {
            "name": "Juniper",
            "constituentVNFDs": [
               {
                  "vnfd": {
                     "name": "Juniper_vSRX_VNFD",
                     "vNetworkInterfaces": 0
                  },
                  "assuranceParameters": [
                     {
                        "name": "Low CPU Utilization",
                        "id": "cpu_util",
                        "condition": "eq",
                        "value": "0.0",
                        "action": "heal"
                     },
                     {
                        "name": "High CPU Utilization",
                        "id": "cpu_util",
                        "condition": "gt",
                        "value": "80.0",
                        "action": "scale"
                     }
```

```
                ]
             }
          ]
       }
    ]
 }
}
```

Get Network Service Descriptor VNFDs

Retrieves a list of VNF descriptors that a network service descriptor references.

Method

GET

URL

http://*nso_host*:*port*/ocnso/1.1/nsd/*nsdName*/vnfds

where *nsdName* is the name of the network service descriptor.

Sample Response

```
{
   "status": "SUCCESS",
   "code": "200",
   "data": [
      "Checkpoint_NG_FW_VNFD",
      "Juniper_vSRX_VNFD"
   ]
}
```

Get Network Service Descriptor Flavors

Retrieves a list of deployment flavors for a specified network service descriptor.

Method

GET

URL

http://*nso_host*:*port*/ocnso/1.1/nsd/*nsdName*/flavors

where *nsdName* is the name of the network service descriptor.

Sample Response

```
{
   "status": "SUCCESS",
   "code": "200",
   "data": [
      {
         "name": "Checkpoint",
         "constituentVNFDs": [
            {
               "vnfd": {
                  "name": "Checkpoint_NG_FW_VNFD",
                  "vNetworkInterfaces": 0
               },
               "assuranceParameters": [
                  {
```

```
                        "name": "Low CPU Utilization",
                        "id": "cpu_util",
                        "condition": "eq",
                        "value": "0.0",
                        "action": "heal"
                     },
                     {
                        "name": "High CPU Utilization",
                        "id": "cpu_util",
                        "condition": "gt",
                        "value": "80.0",
                        "action": "scale"
                     }
                  ]
               }
            ]
         },
         {
            "name": "Juniper",
            "constituentVNFDs": [
               {
                  "vnfd": {
                     "name": "Juniper_vSRX_VNFD",
                     "vNetworkInterfaces": 0
                  },
                  "assuranceParameters": [
                     {
                        "name": "Low CPU Utilization",
                        "id": "cpu_util",
                        "condition": "eq",
                        "value": "0.0",
                        "action": "heal"
                     },
                     {
                        "name": "High CPU Utilization",
                        "id": "cpu_util",
                        "condition": "gt",
                        "value": "80.0",
                        "action": "scale"
                     }
                  ]
               }
            ]
         }
      ]
}
```


Get VNF Descriptor Details

Retrieves details about a specified VNF descriptor.

Method

GET

URL

http://*nso_host*:*port*/ocnso/1.1/vnfd/*vnfdName*

where *vnfdName* is the name of the VNF descriptor.

Sample Response


```
{
   "status": "SUCCESS",
   "code": "200",
   "data": {
      "deploymentFlavors": [
         {
            "name": "vsrx.small",
            "vcpus": 2,
            "memory": 2,
            "disk": 20
         },
         {
            "name": "vsrx.medium",
            "vcpus": 2,
            "memory": 4,
            "disk": 20
         }
      ],
      "connectionPoints": [
         {
            "name": "CP01",
            "isExternal": false,
            "order": -1
         },
         {
            "name": "CP02",
            "isExternal": false,
            "order": -1
         }
      ],
      "versions": [
         {
            "number": "1.0",
            "imageName": "vsrx-12.1X47-D20.7-npaas-v0.3",
            "imageUserName": "",
            "imagePasswd": ""
         },
         {
            "number": "1.1",
            "imageName": "vsrx-12.1X47-D20.7-npaas-v0.4",
            "imageUserName": "",
            "imagePasswd": ""
         }
      ]
   }
}
```


Get VNF Descriptor Versions

Retrieves the list of VNF versions of the specified VNF descriptor.

Method

GET

URL

http://*nso_host*:*port*/ocnso/1.1/vnfd/*vnfdName*/versions

where *vnfdName* is the name of the VNF descriptor.

Sample Response


```
{
   "status": "SUCCESS",
   "code": "200",
   "data": [
      {
         "number": "1.0",
         "imageName": "npaas-srx-poc3-nso",
         "imageUserName": "",
         "imagePasswd": ""
      },
      {
         "number": "1.1",
         "imageName": "npaas-srx-poc3-nso2",
         "imageUserName": "",
         "imagePasswd": ""
      },
      {
         "number": "1.3",
         "imageName": "vsrx-12.1X47-D20.7-npaas-v0.3",
         "imageUserName": "",
         "imagePasswd": ""
      },
      {
         "number": "1.4",
         "imageName": "vsrx-npaas-v0.4",
         "imageUserName": "",
         "imagePasswd": ""
      }
   ]
}
```


Get VNF Descriptor Flavors

Retrieves the list of VNF flavors of a specified VNF descriptor.

Method

GET

URL

http://*nso_host*:*port*/ocnso/1.1/vnfd/*vnfdName*/flavors

where *vnfdName* is the name of the VNF descriptor.

Sample Response


```
{
   "status": "SUCCESS",
   "code": "200",
   "data": [
      {
         "name": "vsrx.small",
         "vcpus": 2,
         "memory": 2,
         "disk": 20
      },
      {
         "name": "vsrx.medium",
         "vcpus": 2,
```

```
            "memory": 4,
            "disk": 20
        },
        {
            "name": "m1.medium",
            "vcpus": 2,
            "memory": 4,
            "disk": 40
        }
    ]
}
```