Oracle® Communications Network Service Orchestration Solution

Implementation Guide

Release 1.1.1

E72585-02

July 2016

ORACLE®

Oracle Communications Network Service Orchestration Solution Implementation Guide, Release 1.1.1

E72585-02

Contents

Preface

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

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 the Network Service Orchestration solution

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

Related Documentation

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.
- *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.
- *UIM Cartridge Guide*: Provides information about how you use cartridge packs with UIM. Describes the content of the base cartridges.

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

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

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 Solution and describes the solution components and software requirements.

About Network Service Orchestration Solution

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

The Network Service Orchestration solution provides the following functionality:

- **Onboarding of Network Services and VNFs**. You can define network services and VNFs based on any network function that you want to virtualize. See "Designing and Onboarding Network Services and VNFs" for more information.
- **Instantiation, Scaling, and Termination of Network Services**. You can quickly instantiate, scale, or terminate VNFs and network services in response to the 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 and VNFs" for more information.

  The solution supports asynchronous communication with northbound applications. See "Integrating the Solution With Northbound Applications for Asynchronous Communication" for more information.
- **Monitoring and Auto-healing**. You can monitor the performance of the VNFs continuously and configure the solution to heal a failed VNF automatically. See "Monitoring and Healing a VNF" for more information about monitoring and healing a VNF.
- **Resource Orchestration**. The solution 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 and VNFs" for more information.
- **Customization and Extension**. You can customize and extend the solution to support integration with third-party VNF Managers, Virtualized Infrastructure Managers (VIMs), software-defined networking (SDN) controllers, and monitoring engines. The solution also provides extension points that enable you to customize and extend the solution's core functionality. See "Extending the Network Service Orchestration Solution" for more information.

The solution includes a VNF Manager that enables you to manage the life cycles of the VNFs. The solution also supports integration with Oracle and third-party VNF

Managers, VIMs, SDN controllers, and network monitoring applications. By default, the solution provides integration to certain applications and supports integration to additional applications during the implementation. The solution provides RESTful APIs, which communicate over HTTP, to interact and exchange data with the solution's components.

Solution Components

The Network Service Orchestration solution 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 hybrid, virtual, and physical networks.

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

About Network Service Orchestration Entities

The Network Service Orchestration solution 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 the following characteristics: ▪ Disk Total ▪ Memory Total ▪ VCPU Total ▪ Disk Used ▪ Memory Used ▪ VCPU Used	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.
Endpoint	Custom Object	Describes a service access point for the network service.

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

NFV Entity	OCIM Entity	Description
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 the following characteristics: ■ Disk Total ■ Memory Total ■ VCPU Total ■ Disk Used ■ Memory Used ■ VCPU Used	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 the solution. 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 with the following characteristics: ■ Host ■ Port ■ Username ■ Password ■ Domain Name ■ Tenant Name ■ VIM Type	Represents the totality of all hardware and software components that build the environment where VNFs are deployed. Represents a tenant.
Network Service	Service	Represents a composition of network functions.
Network Service Descriptor	Service 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 lifecycle of a network service instance.
NFV Service Request	Business Interaction	Represents an NFV life-cycle action in UIM. Every time you perform a life-cycle action, the solution creates a business interaction for the action in UIM.
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.

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

NFV Entity	OCIM Entity	Description
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 the following characteristics: ■ Disk Total ■ Memory Total ■ VCPU Total	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	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 the following characteristics: ■ Host ■ Port ■ Username ■ Password ■ Domain Name ■ Tenant Name ■ VIM Type	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	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 lifecycle of a VNF instance.

About the Sample Network Protection Service

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

See "About the Sample Network Protection Service Model" for detailed information about the service model and instructions for implementing the sample network protection service.

About the Branding Cartridge

By default, in the UIM user interface, the entities are labeled using the standard UIM names that are based on Oracle Communications Information Model. For example, VNFs are displayed as logical devices and VNF descriptors are displayed as Logical Device specifications. You can customize the UIM user interface to label your entities using the standard NFV terminology.

The Network Service Orchestration solution provides a branding sample cartridge that you can deploy into UIM to display a separate group of links and pages. The group display your resource entities in the NFV-standard terminology. The cartridge provides NFV-specific label names for links, field names, and entities in the UIM user interface. The cartridge also filters entities in the search results pages to display only the solution-specific entities that you work with in the solution.

When you deploy the branding cartridge into UIM, UIM does the following:

- Displays the Network Service Orchestration Solution banner at the top of the UIM screens.
- Displays the **Network Service Orchestration** group in the navigation section that includes the following expandable and collapsible subgroups of links:

  In the **Orchestration** subgroup:

  - **NFV Service Requests**. Clicking this link displays the **Search** page for service requests. The search page returns service requests that are based on your NFV service request specifications.
  - **Network Services**. Clicking this link displays the **Search** page for network services. The search page returns a list of network services that are 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 based on your VNF descriptors.

  In the **Catalog** group:

  - **Network Service Descriptors**. Clicking this link displays the **Search** page for Network Service descriptors. The search page 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.
- Filters the entities that are displayed in the search results pages to retrieve only those entities that are created based on your VNF and network service descriptors that you use in the solution.

See "Configuring UIM for the Network Service Orchestration Solution" for instructions about branding the UIM user interface.

2 Installing and Integrating the Solution Components

This chapter describes the software requirements and instructions for installing and integrating Oracle Communications Network Service Orchestration Solution components.

Planning Your Implementation

Before you implement the Network Service Orchestration solution, 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 with the solution. 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 the solution:

- Install and configure the required software. See "Configuring UIM for the Network Service Orchestration Solution".
- Integrate the Virtual Infrastructure Manager (VIM). See "Integrating the VIM with the Solution".
- 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 and VNFs".
- Write extensions for extending the core functionality and integrate third-party software with the solution. See "Using Extension Points and Java Interface Extensions to Extend the Solution".
- Integrate client applications with the solution for using the RESTful APIs. For details about the solution’s RESTful APIs, see "Network Service Orchestration RESTful API Reference".

Software Requirements

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

- Oracle Communications Unified Inventory Management 7.3.3.

  See *UIM 7.3.3 Installation Guide* for installation instructions.
- Oracle Communications Design Studio 7.3.2.

See *Design Studio 7.3.2 Installation Guide* for installation instructions.

Configuring UIM for the Network Service Orchestration Solution

To configure UIM for the Network Service Orchestration solution:

1. Install UIM on a WebLogic server. See *UIM 7.3.3 Installation Guide* for installation instructions.

   Note: If you are upgrading to UIM 7.3.3 from UIM 7.3.1 or UIM 7.3.2, follow the steps for upgrading from UIM 7.3.*x* to UIM 7.3.2 in the *UIM 7.3.3 Installation Guide*.

   If you are upgrading the solution from 1.1 to 1.1.1, see "Upgrading the Network Service Orchestration Solution".

2. Download and set up the **gson** library:

   a. Download the **gson-2.2.4.jar** file from the following website and copy it to the *UIM_Home*/**lib** folder, where *UIM_Home* is the directory into which UIM is installed:

      `http://repo1.maven.org/maven2/com/google/code/gson/gson/2.2.4/`

   b. Open the *Domain_Home*/**bin**/**setUIMEnv.sh** file and add the following entry, where *Domain_Home* is the directory that contains the configuration for the domain into which UIM is typically installed:

      ```
      CLASSPATH="${CLASSPATH}:${UIM_HOME}/lib/gson-2.2.4.jar"
      export CLASSPATH
      ```

3. In the WebLogic server on which UIM is installed, deploy the *WL_HOME*/**common**/**deployable-libraries**/**jersey-bundle-1.9.war** file as a library and specify the target as the server on which UIM is installed.
4. Set up queues in the WebLogic server. See "Setting Up Queues in WebLogic Server" for detailed instructions.
5. Restart the server on which UIM is installed.
6. Navigate to the *UIM_Home*/**cartridges** directory and deploy the following base UIM cartridges into UIM in the order they are listed:
   - ora_uim_baseextpts
   - ora_uim_basemeasurements
   - ora_uim_basetechnologies
   - ora_uim_basespecifications
   - (Optional) ora_uim_common. Deploy this cartridge if you want to implement a network protection service by using the sample cartridges.

   See *UIM Cartridge Guide* for instructions about deploying cartridges into UIM.
7. Create a local directory (*NSO_Software_Home*).
8. In the UIM software pack, locate the **OracleComms_NSO_1.1.1.0.0.***build_number***.zip** file and extract it into the *NSO_Software_Home* directory.

9. Navigate to the *NSO_Software_Home*/**deploy**/**individualJarsForSuperJar** directory and deploy the following Network Service Orchestration solution cartridges into UIM in the order they are listed:
   - OracleComms_NSO_NFVIAdapter
   - OracleComms_NSO_Common
   - OracleComms_NSO_BaseCartridge
10. (Optional) If you want to use the sample cartridges that are provided with the solution, navigate to the *NSO_Software_Home*/**designStudio**/**cartridgeZips** directory and deploy the following sample cartridges into UIM in the order they are listed:
    - NPaaS_NetworkService

      This sample cartridge contains the functionality to implement Network Protection as a network service.
    - Checkpoint_NG_FW_VNF

      This sample cartridge contains the Checkpoint firewall VNF to use with the Network Protection service.
    - Juniper_vSRX_VNF

      This sample cartridge contains the Juniper vSRX firewall VNF to use with the Network Protection service.
11. (Optional) By default, in the UIM user interface, your entities are labeled using the standard UIM names that are based on the Oracle Communications Information Model. To label your entities using the standard NFV terminology, extract and deploy the *NSO_Software_Home*/**designStudio**/**cartridgeZips/NSOBranding** sample cartridge into UIM.

    Note: After you deploy the branding cartridge into UIM, you cannot undeploy it or return UIM to its previous state.

12. In the WebLogic server on which UIM is installed, deploy the *NSO_Software_Home*/**deploy**/**applications**/**OracleComms_NSO_WebServices.war** file as a web application.

    To deploy the **.war** file into WebLogic server:

    a. Copy the **custom.ear** file from *Domain_Home*/**UIM**/**app**/**7_3_3**/ to a temporary directory.

    b. Navigate to the temporary directory and expand the **custom.ear** archive file by running the following command:

    ```
    jar xvf custom.ear
    ```

    c. Delete the **custom.ear** file and copy the **deploy**/**applications**/**OracleComms_NSO_WebServices.war** file to the temporary directory.

    d. Open the **META_INF/application**.**xml** file in a text editor and add the following text:

    ```
    <module>
              <web>
                  <web-uri>OracleComms_NSO_WebServices.war</web-uri>
                  <context-root>/ocnso/1.1</context-root>
    ```

```
            </web>
    </module>
```

e. Rebuild the **custom.ear** file by running the following command:

   ```
   jar cvf custom.ear *
   ```

f. Log in to Oracle WebLogic Server Console.

g. Click **Lock and Edit**.

h. Click **Deployments**.

i. In the Summary of Deployments section, select **custom** and click **Update**.

j. Select **Redeploy this application using the following deployment files** and click **Change Path**.

k. Browse and select the **custom.ear** file, which is created in the temporary directory.

l. Click **Next**.

m. Click **Finish**.

n. Click **Activate Changes**.

13. (Optional) If you want to use an SDN controller to control data flows for your network service, register the SDN controller with the solution. See "Registering the SDN Controller" for instructions.

14. (Optional) Integrate the solution with northbound applications for asynchronous communication. See "Integrating the Solution With Northbound Applications for Asynchronous Communication".

After you install the required software and configure UIM, integrate the VIM with the solution. See "Integrating the VIM with the Solution" for more information.

Setting Up Queues in WebLogic Server

You set up queues in the WebLogic server to maintain fault tolerance when there is an issue in the activation of resources after a lifecycle action is performed.

To set up queues in WebLogic server, do any one of the following:

- Run the scripts to set up queues in the WebLogic server automatically. See "Running Scripts to Set Up Queues in WebLogic Server" for instructions.
- Configure the WebLogic server manually. See "Setting Up Queues in WebLogic Server Manually" for instructions.

Running Scripts to Set Up Queues in WebLogic Server

The Network Service Orchestration solution includes scripts that you can run to set up queues in the WebLogic server.

Before you run the scripts, do the following:

- Specify the WebLogic server details in the following files:
  - For standalone server, specify the server details in the *NSO_Software_Home*/**tools/jmsWlstScripts/nso_jms_configuration_standalone.properties** file.
  - For cluster setup, specify the server details in the *NSO_Software_Home*/**tools/jmsWlstScripts/nso_jms_configuration_cluster.properties** file.

- If SSL is enabled for secure communication, open all the script files and change the protocol from **t3** to **t3s**:

  For example, change:

  URL="t3://"+AdminServerListenAddress+":"+AdminServerListenPort

  to

  URL="t3s://"+AdminServerListenAddress+":"+AdminServerListenPort.

To set up queues in the WebLogic server:

- If SSL is not enabled, do the following:
  - For standalone server, run the following script:

    WebLogic_Home/**oracle_common/common/bin/wlst.cmd nso_jms_queue_standalone.py**

  - For cluster setup, run the following script:

    WebLogic_Home/**oracle_common/common/bin/wlst.cmd nso_jms_queue_cluster.py**

- If SSL is enabled, do the following:
  1. Set the WebLogic server environment by running the following command:

     WebLogic_Home/**wlserver/server/bin/setWLSEnv.sh**

  2. For standalone server, run the following script:

     ```
     java -Dweblogic.security.SSL.ignoreHostnameVerification=true
     -Dweblogic.security.CustomTrustKeyStoreType="JKS"
     -Dweblogic.security.TrustKeyStore=DemoTrust
     -Dweblogic.security.CustomTrustKeyStoreFileName="Weblogic_
     Home/wlserver/server/
      lib/DemoTrust.jks" weblogic.WLST nso_jms_queue_standalone.py
     ```

     For cluster setup, run the following script:

     ```
     java -Dweblogic.security.SSL.ignoreHostnameVerification=true
     -Dweblogic.security.CustomTrustKeyStoreType="JKS"
     -Dweblogic.security.TrustKeyStore=DemoTrust
     -Dweblogic.security.CustomTrustKeyStoreFileName="Weblogic_
     Home/wlserver/server/
      lib/DemoTrust.jks" weblogic.WLST nso_jms_queue_cluster.py
     ```

Setting Up Queues in WebLogic Server Manually

To set up queues in WebLogic server manually:

1. In the WebLogic server on which UIM is installed, create a JDBC Store with the following parameters:
   - **Name**: **inventoryNSOStore**
   - **Target**: Specify **AdminServer** or the managed server on which UIM is installed.
   - **Datasource**: **InventoryDatSource**

   If you use cluster setup, create a JDBC Store for each managed server and specify a unique prefix name. For example, if you use managed server 1 (MS1) and managed server 2 (MS2), create a JDBC Store with the name **inventoryNSOStore-0** targeting MS1 and another JDBC Store with the name **inventoryNSOStore-1** targeting MS2 with unique prefix names.

2. Create a JMS Server with the following parameters:
   - **Name**: **NSOJMSServer**
   - **Persistent Server**: Specify the persistent store that you created earlier.
   - **Target**: Specify **AdminServer** or the managed server on which UIM is installed.

   If you use cluster setup, create a JMS Server for each managed server and associate the JMS servers to the JDBC stores that you created earlier. For example, if you use managed server 1 (MS1) and managed server 2 (MS2), create a JMS Server with the name **NSOJMSServer-0** targeting MS1 and associate it to the **inventoryNSOStore-0** JDBC Store. Create another JMS Server with the name **NSOJMSServer-1** targeting MS2 and associate it to the **inventoryNSOStore-1** JDBC Store.
3. Activate the changes.
4. Create a JMS Module with the following parameters:
   - **Name**: **NSOModule**
   - **Target**: Specify the **AdminServer** or select all the servers if you use cluster setup.
5. For the NSOModule that you created, create a Connection Factory with the following parameters:
   - **Name**: **NSORequestQueueConnFactory**
   - **JNDI Name**: **NSORequestQueueConnFactory**
   - Select **Advanced Targeting**
   - Create a new sub-deployment with the name **NSOJMSServer**
   - For the sub-deployment that you created, specify **NSOJMServer** as the target. If you use cluster setup, specify all the JMS servers that you created.
6. For the NSOModule, create a JMS Queue for a standalone server setup or create a Distributed JMS Queue for a cluster setup with the following parameters:
   - **Name**: **NSORequestQueue**
   - **JNDI Name**: **NSORequestQueue**
   - Select **Advanced Targeting**.
   - Select **NSOJMSServer** as the sub-deployment.
   - Select **NSOJMSServer** as target. If you use cluster setup, specify all the JMS servers.
7. Activate the changes.

Registering the SDN Controller

If your network service requires implementation of network flows, then you can set up the solution 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 the solution to manage flow control in the network. The Network Service Orchestration solution 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 the solution:

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.

Integrating the Solution With Northbound Applications for Asynchronous Communication

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

With this integration, the solution 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
- Configure a VNF
- Upgrade the software version of a VNF

To integrate the solution with northbound systems for asynchronous communication, you must set up a topic in the WebLogic server. After you set up the topic in the WebLogic server, you can configure your client applications to subscribe to the topic.

Note: You can also customize the solution's asynchronous communication functionality. For information about writing your own implementation for asynchronous communication, see "Implementing a Custom Response Manager".

To set up topics in the WebLogic server, do any one of the following:

- Run the scripts to set up topics in the WebLogic server automatically. See "Running Scripts to Set Up Topics in WebLogic Server" for instructions.
- Configure the WebLogic server manually. See "Setting Up Topics in WebLogic Server Manually" for instructions.

Running Scripts to Set Up Topics in WebLogic Server

The Network Service Orchestration solution includes scripts that you can run to configure the WebLogic server for asynchronous communication with northbound applications.

Before you run the scripts, do the following:

- Specify the WebLogic server details in the following files:
  - For the standalone server, specify the server details in the *NSO_Software_Home*/**tools**/**jmsWlstScripts/nso_jms_configuration_standalone.properties** file.
  - For cluster setup, specify the server details in the *NSO_Software_Home*/**tools**/**jmsWlstScripts/nso_jms_configuration_cluster.properties** file.
- If SSL is enabled for secure communication, open all the script files and change the protocol from **t3** to **t3s**:

  For example, change

  URL="t3://"+AdminServerListenAddress+":"+AdminServerListenPort

  to

  URL="t3s://"+AdminServerListenAddress+":"+AdminServerListenPort.

To set up topics in the WebLogic server:

- If SSL is not enabled, do the following:
  - For a standalone server, run the following script:

    WebLogic_Home/**oracle_common/common/bin/wlst.cmd nso_jms_topic_standalone.py**
  - For a cluster setup, run the following script:

    WebLogic_Home/**oracle_common/common/bin/wlst.cmd nso_jms_topic_cluster.py**
- If SSL is enabled, do the following:
  1. Set the WebLogic server environment by running the following command:

     WebLogic_Home/**wlserver/server/bin/setWLSEnv.sh**
  2. For standalone server, run the following script:

     ```
     java -Dweblogic.security.SSL.ignoreHostnameVerification=true
     -Dweblogic.security.CustomTrustKeyStoreType="JKS"
     -Dweblogic.security.TrustKeyStore=DemoTrust
     -Dweblogic.security.CustomTrustKeyStoreFileName="Weblogic_
     Home/wlserver/server/
      lib/DemoTrust.jks" weblogic.WLST nso_jms_topic_standalone.py
     ```

     For cluster setup, run the following script:

     ```
     java -Dweblogic.security.SSL.ignoreHostnameVerification=true
     -Dweblogic.security.CustomTrustKeyStoreType="JKS"
     -Dweblogic.security.TrustKeyStore=DemoTrust
     -Dweblogic.security.CustomTrustKeyStoreFileName="Weblogic_
     Home/wlserver/server/
      lib/DemoTrust.jks" weblogic.WLST nso_jms_topic_cluster.py
     ```

Setting Up Topics in WebLogic Server Manually

To set up topics in the WebLogic server:

1. In the WebLogic server on which UIM is installed, for the NSOModule, create a Connection Factory with the following parameters:
   - **Name**: **NSOResponseTopicConnFactory**
   - **JNDI Name**: **NSOResponseTopicConnFactory**
   - Select **Advanced Targeting**
2. For the sub-deployment that you created, specify **NSOJMSServer** as the target. If you use cluster setup, specify all the JMS servers that you created.
3. Create a JMS topic for a standalone server setup or create a Distributed JMS topic for a cluster setup with the following parameters:
   - **Name**: **NSOResponseTopic**
   - **JNDI Name**: **NSOResponseTopic**
   - Select **NSOJMSServer** as the sub-deployment.
   - Select **NSOJMSServer** as target. If you use cluster setup, specify all the JMS servers that you created.
4. Activate the changes.

Integrating the VIM with the Solution

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

Integrating the VIM with the solution involves the following tasks:

- Registering the VIM
- Discovering VIM Resources

Registering the VIM

To register a VIM with the solution:

1. Ensure that UIM is started and running.
2. Ensure that the Network Service Orchestration solution cartridges are deployed into UIM.
3. Start the VIM and ensure that you have the IP address, username, and password 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 a VIM" in the "Network Service Orchestration RESTful API Reference" chapter.

   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 the solution contains information about the current status and availability of all the required virtual resources on the network. In UIM, the VIM is represented as a custom object.

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

- Availability zone (OpenStack)
- Flavor
- Host
- VDC

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 the solution 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" in the "Network Service Orchestration RESTful API Reference" chapter.

   The RESTful API client returns a response.

2. In UIM, verify that the following entities are created as Custom Objects:

   - Availability zone
   - Flavor
   - Host
   - VDC

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 the solution.

Enabling Logging for the Network Service Orchestration Solution

You enable logging for the solution 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 the Network Service Orchestration solution:

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.

Upgrading the Network Service Orchestration Solution

If you are using Network Service Orchestration Solution 1.1, you can upgrade to Network Service Orchestration Solution 1.1.1.

To upgrade to Network Service Orchestration Solution 1.1.1:

1. Upgrade UIM to UIM 7.3.3. See *UIM 7.3.3 Installation Guide* for instructions.
2. Follow steps from 4 to 12 in the "Configuring UIM for the Network Service Orchestration Solution" section.
3. In your network service properties file, add or update the following parameters and specify values for the parameters:
   - *NSD_Name*.**default.serviceArea**.*default_service_area*

     where:

     – *NSD_Name* is the name of your network service descriptor file.

     – *default_service_area* is the name of the default service area
   - *NSD_Name*.**default.dataCenter**.*default_data_center*

     where *default_data_center* is the name of the default data center
   - **sdnController**.*NSD_Name*
4. Update your VIM by running the following RESTful API:

   PUT http://*nso_host*:*port*/ocnso/vim/*vimId*

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

For more details about the request parameters, see "Update a VIM" in the "Network Service Orchestration RESTful API Reference" chapter.

Specify values for the following parameters:

- **version**. For OpenStack Keystone version 2.0, specify **2**. For OpenStack Keystone version 3.0, specify **3**.
- **cpuOvercommitRatio**. Specify the ratio of overcommitted vCPU.
- **memoryOverCommitRatio**. Specify the ratio of overcommitted memory.
- **diskOverCommitRatio**. Specify the ratio of overcommitted disk.

5. Discover the VIM resources by running the following RESTful API:

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

   where:

   - *vimId* is the Id of the VIM 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 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" in the "Network Service Orchestration RESTful API Reference" chapter.

Supported Southbound Integration

The Network Service Orchestration solution supports the following southbound integrations:

- For VNF management:
  - VNF Manager, with the ability to manage VNFs through direct integration or by integration with an Element Management System (EMS)
  - Integration with external VNF Managers
- For virtual infrastructure management:
  - Integration to OpenStack Kilo with Keystone version 2 and version 3, and Oracle OpenStack for Oracle Linux Release 2
  - Sample integration to VMware vCloud Director
  - Integration with other Virtual Infrastructure Managers
- For network and SDN controllers:
  - Integration to OpenStack Neutron (Kilo release)
  - Sample integration to OpenDaylight

3

Designing and Onboarding Network Services and VNFs

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

About the Design Components

The design components constitute resources that you create in Oracle Communications Design Studio. The Network Service Orchestration solution uses different types of files that you create in Design Studio to describe the behavior of your network services and VNFs.

- **Entity Specifications**. You create specifications in Design Studio that you use to create instances of VNFs and network services in Oracle Communications Unified Inventory Management (UIM).

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

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

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

- **Technical actions files**. The technical actions files describe the actions for the VNFs and Network Services in the VIM. There is one technical actions file for each network service and VNF.

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

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

About the Descriptor Files

The descriptor files contain metadata about the network services and VNFs. The solution defines network service and VNF descriptors in the form of Design Studio specifications.The Network Service Orchestration solution uses these specifications to manage the life cycles of network services and VNFs.

Network services are assembled from the defined units of behavior provided by the VNFs in the cartridges. Network Service descriptors structure how these network services are populated in the cartridges. VNF descriptors describe the behavior of virtual functions that are defined in the Network Service Orchestration cartridges. There is one descriptor file for each network service and a VNF.

About the Network Service Descriptor Files

Network Service descriptor files describe the deployment requirements, operational behavior, and policies required by network services based on them.

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

In the network service descriptor file, you:

- Define the networks by either creating them or by referencing existing networks and specifying network types. See "Describing Networks" for more information.
- For each network, specify the VNFs the network service should use.
- Specify the network forwarding path for the network traffic. See "Describing Forwarding Graphs" for more information.
- For each VNF in the network service, specify parameters related to CPU utilization and other factors related to performance of the virtual machine on which the VNF is deployed. See "Describing Deployment Flavors" for more information.
- Specify when you want the solution to heal a VNF and scale the network service.

The solution includes the sample **NPaaS_NSD.xml** network service descriptor file.

Note: If you create your own descriptor file for a network service, ensure that the name of the descriptor file ends with **NSD**. For example, if you want to create a network service descriptor file for an IP Multimedia Subsystem (IMS) network service with the name **IMSaaS**, create it as **IMSaaS_NSD**. This enables the solution to filter and return search results for network service descriptors only.

Describing Networks

In the network service descriptor XML 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 pattern in which you describe a virtual link descriptor, which corresponds to a network in the sample **NPaaS_NSD.xml** network service descriptor file:

```
-<virtualLinkDescriptors>
  -<virtualLinkDescriptor name="network_name" type="network_type"
isReferenced="value">
    <numberOfEndPoints>number_of_endpoints</numberOfEndPoints>
    -<connectionPoints>
      <!-- The format is VNFD:ConnectionPoint -->
      <connectionPoint name="vnf_descriptor_name:connection_point_name"
type="connection_point_type" order="connectionPoint_order"/>
    </connectionPoints>
  </virtualLinkDescriptor>
</virtualLinkDescriptors>
```

where:

- *network_name* is the name of the network that you want to create or reference.
- *network_type* is the type of the network that you want to create or reference.
- *value* indicates whether you want to create or reference the network. Specify **true** or **false**.
- *number_of_endpoints* is the number of endpoints that the network provides.
- *vnf_descriptor_name*:*connection_point_name* is the name of the VNF descriptor XML file and the name of the VNF connection point.
- *connection_point_type* is the type of the connection point.
- *connectionPoint_order* is the order of the connection points for the VNF.

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

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

Describing Forwarding Graphs

In the network service descriptor XML file, you 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 the **NPaaS_NSD.xml** network service descriptor file:

```
-<forwardingGraphDescriptors>
  -<forwardingGraphDescriptor name="ForwardingGraphName" default="default">
   -<networkForwardingPath>
    -<vnfd name="vnf_descriptor_name">
     -<connectionPoints>
      <connectionPoint name="connection_point_name" type="type_of_
connectionPoint"/>
     <connectionPoint name="connection_point_name" type="type_of_
connectionPoint"/>
    </connectionPoints>
   </vnfd>
  </networkForwardingPath>
 </forwardingGraphDescriptor>
</forwardingGraphDescriptors>
```

where:

- *ForwardingGraphName* is the name of the forwarding graph.
- *default* indicates if the network service should use this forwarding graph by default or not.

- *vnf_descriptor_name* is the name of the VNF descriptor that you want to use with the network service.
- *connection_point_name* is the name of the connection point defined in the VNF descriptor, that you want to use for the forwarding graph.
- *type_of_connectionPoint* is the type of the connection point.

The following text shows a sample forwarding graph element in the **NPaaS_NSD.xml** 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 XML file, you describe deployment flavors by specifying parameters for CPU utilization and other factors related to the performance 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 the **NPaaS_NSD.xml** network service descriptor file:

```
-<serviceDeploymentFlavors>
 -<serviceDeploymentFlavor name="flavorName">
  -<constituentVNFDs>
   -<vnf>
    <vnfd name="VNFDname"/>
     -<assuranceParameters>
      -<assuranceParameter name="assuranceParameterName" action="action">
       <id>Id</id>
       <value>value</value>
       <condition>condition</condition>
      </assuranceParameter>
    -<assuranceParameter name="assuranceParameterName" action="action">
      <id>Id</id>
      <value>value</value>
      <condition>condition</condition>
     </assuranceParameter>
    </assuranceParameters>
   </vnf>
  </constituentVNFDs>
 </serviceDeploymentFlavor>
</serviceDeploymentFlavors>
```

where:

- *flavorName* is the name of the service deployment flavor.
- *VNFDname* is the name of the VNF Descriptor.

- *assuranceParameterName* is the name of the assurance parameter.
- *action* is the action you want to perform on the VNF. You can specify either to **heal** or **scale** the VNF.
- *Id* is the Id of the assurance parameter.
- *value* is the threshold value.
- *condition* is the condition based on which the action is performed.

The following text shows a sample service deployment flavor element in the **NPaaS_NSD.xml** 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>
```

About the VNF Descriptor Files

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

The solution 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.

> **Note:** If you create your own descriptor file for a VNF, ensure that the name of the descriptor file ends with **VNFD**. For example, if you want to create a VNF descriptor file for Cisco's VRF-aware VNF with the name **CiscoVRF**, create it as **CiscoVRF_VNFD**. This enables the solution to filter and return search results for VNF descriptors only.

In the VNF descriptor file, you specify:

- Deployment flavor parameters
- Connection points for the VNF

- Software version of the VNF

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

```
-<vnfd name="VNFdescriptorName">
 -<deploymentFlavors>
  <deploymentFlavor name="deploymentFlavorName" disk="diskSpace" memory="memory"
vcpus="vcpus"/>
  <deploymentFlavor name="deploymentFlavorName" disk="diskSpace" memory="memory"
vcpus="vcpus"/>
  </deploymentFlavors>
  -<connectionPoints>
    <connectionPoint name="ConnectionPointName"/>
    <connectionPoint name="ConnectionPointName"/>
    <connectionPoint name="ConnectionPointName"/>
   </connectionPoints>
   <defaultDeploymentFlavor>defaultDeploymentFlavorName</defaultDeploymentFlavor>
  -<versions>
   <version imagePasswd="" imageUserName="" imageName="imageName"
number="versionNumber"/>
  </versions>
</vnfd>
```

where:

- *VNFdescriptorName* is the name of the VNF Descriptor.
- *deploymentFlavorName* is the name of the VNF deployment flavor.
- *diskSpace* is the disk space that you want to allocate for the VNF.
- *memory* is the memory you want to allocate for the VNF.
- *vcpus* is the number of virtual CPUs that you want to allocate for the VNF.
- *ConnectionPointName* is the name of the connection point.
- *defaultDeploymentFlavorName* 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 version number of the VNF image.

The following text shows sample VNF elements in the **Juniper_vSRX_VNFD.xml** 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"/>
  </versions>
</vnfd>
```

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 solution cartridges. See "Setting Up Design Studio for the Network Service Orchestration Solution Cartridges" for more information about importing the cartridges into Design Studio.
2. Switch to the Navigator view.
3. In the root directory of the cartridge, 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*_NSD.**xml** for a network service and *ServiceSpecificationName*_VNFD.**xml** for a VNF, where *ServiceSpecificationName* is the name of the service specification.
5. Copy the sample content from the sample cartridge to the XML file and modify it according to your service requirements.

About the Technical Actions File

The technical actions file describe the actions for the VNFs and Network Services in the VIM. There is one technical actions file for each network service and VNF.

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 and a VNF Service specification.

To create a technical actions file:

1. In Design Studio, switch to the Navigator view.
2. In the root directory of the cartridge, 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 to the XML file and modify it according to your service requirements.

About the 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, the solution 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.

The solution generates the *VNFD_Name*.**conf** configuration file based on the *VNFD_Name*__**Template.conf**__ file and the *VNFD_Name*__**Config.xml**__ file.

The solution 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>
```

About the Sample Network Protection Service Model

The Network Service Orchestration solution provides the following sample cartridges that you can use as references for designing and implementing network protection as a service on your network:

- Juniper_vSRX_VNF. This sample cartridge contains the functionality to implement a Juniper vSRX firewall as a VNF.
- Checkpoint_NG_FW_VNF. This sample cartridge that contains the functionality to implement a Checkpoint firewall as a VNF.
- NPaaS_NetworkService. This sample cartridge contains the functionality to implement a Network Protection service that constitutes the Juniper vSRX firewall VNF and the Checkpoint firewall VNF.

Figure 3–1 shows how a VNF service is modeled in the sample VNF cartridge.

Figure 3–1 VNF Service Model

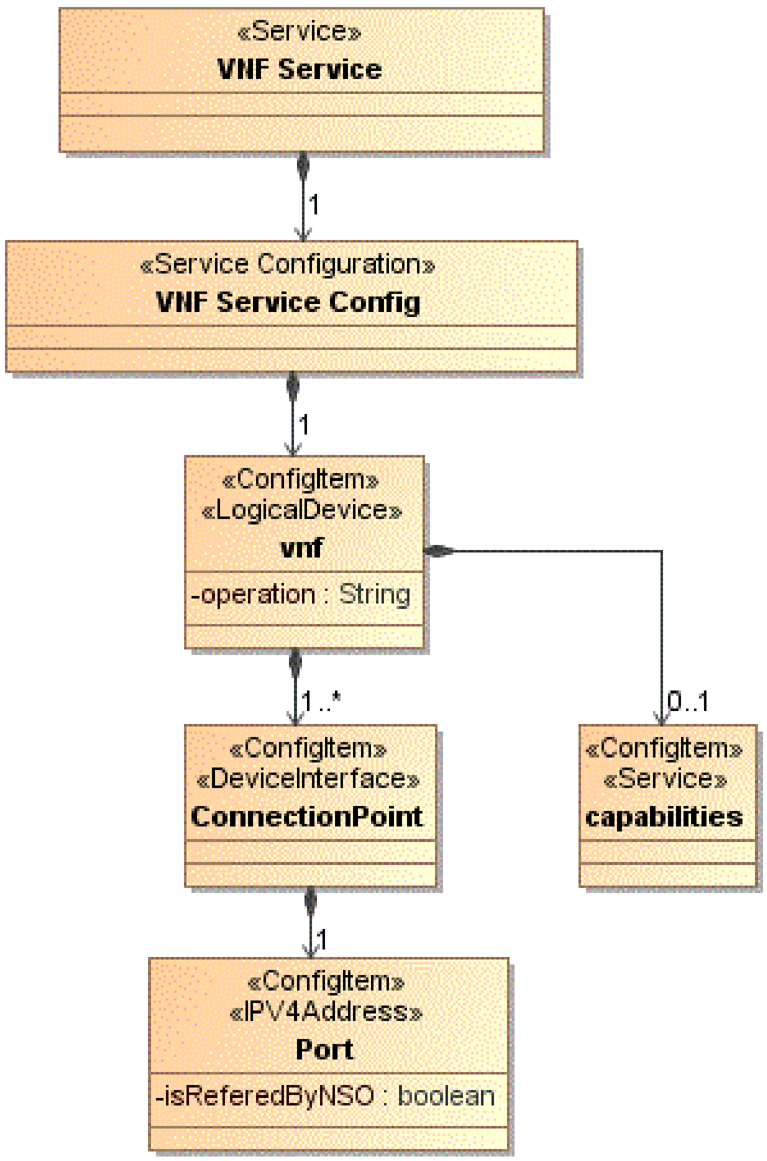
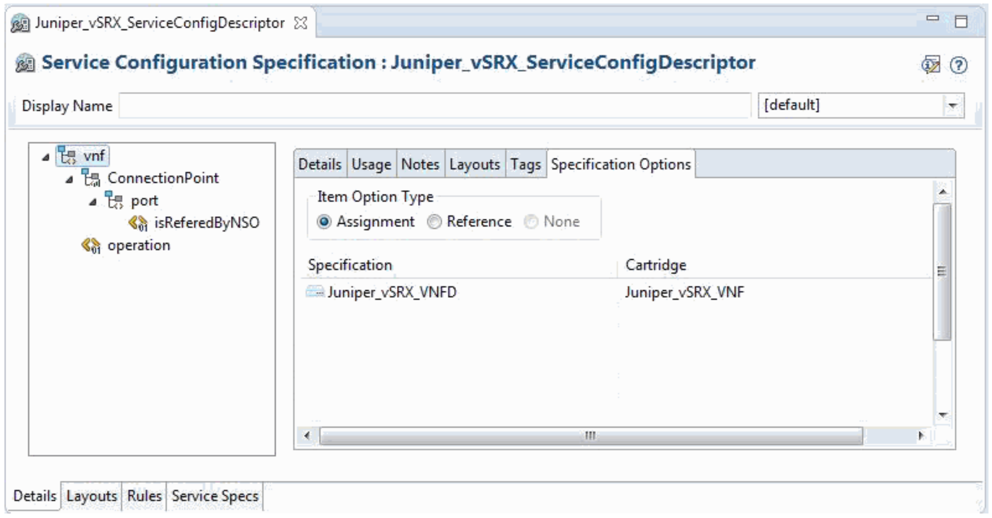


Figure 3–2 shows the Details view of the Juniper vSRX VNF service configuration specification in Design Studio.

Figure 3–2 VNF Service Configuration Details



Item Option Type: [✓ Assignment] Reference None

Specification	Cartridge
Juniper_vSRX_VNFD	Juniper_vSRX_VNF

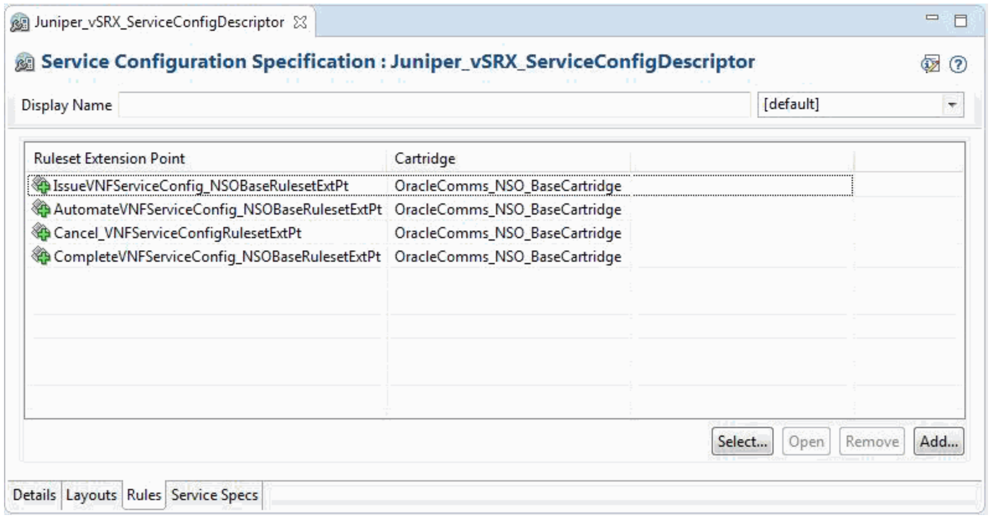
Table 3–1 lists the specifications that are associated to the configuration items in the Juniper vSRX VNF service configuration.

Table 3–1 VNF Service Configuration Specifications

Configuration Item	Associated Specification
vnf	Juniper_vSRX_VNFD logical device specification as assignment. This specification is included in the Juniper sample cartridge.
ConnectionPoint	CPD device interface specification as reference. This specification is included in the OracleComms_NSO_BaseCartridge cartridge.
port	IPV4Address specification as reference. This specification is included in the in OracleComms_NSO_BaseCartridge cartridge.
capabilities	Juniper_vSRX_Capability_ServiceDescriptor specification as assignment. This specification is included in the Juniper sample cartridge.

Figure 3–3 shows the ruleset extension points that are associated to the Juniper vSRX VNF service configuration specification.

Figure 3–3 Rules Associated to the Juniper vSRX VNF Service Configuration Specification


Juniper_vSRX_ServiceConfigDescriptor
Service Configuration Specification : Juniper_vSRX_ServiceConfigDescriptor
Select...
Open
Remove
Add...
Details
Layouts
Rules
Service Specs

Display Name: [default]

Ruleset Extension Point	Cartridge
IssueVNFServiceConfig_NSOBaseRulesetExtPt	OracleComms_NSO_BaseCartridge
AutomateVNFServiceConfig_NSOBaseRulesetExtPt	OracleComms_NSO_BaseCartridge
Cancel_VNFServiceConfigRulesetExtPt	OracleComms_NSO_BaseCartridge
CompleteVNFServiceConfig_NSOBaseRulesetExtPt	OracleComms_NSO_BaseCartridge

Figure 3–4 shows how a VNF capability service is modeled.

Figure 3–4 VNF Capability Service Model

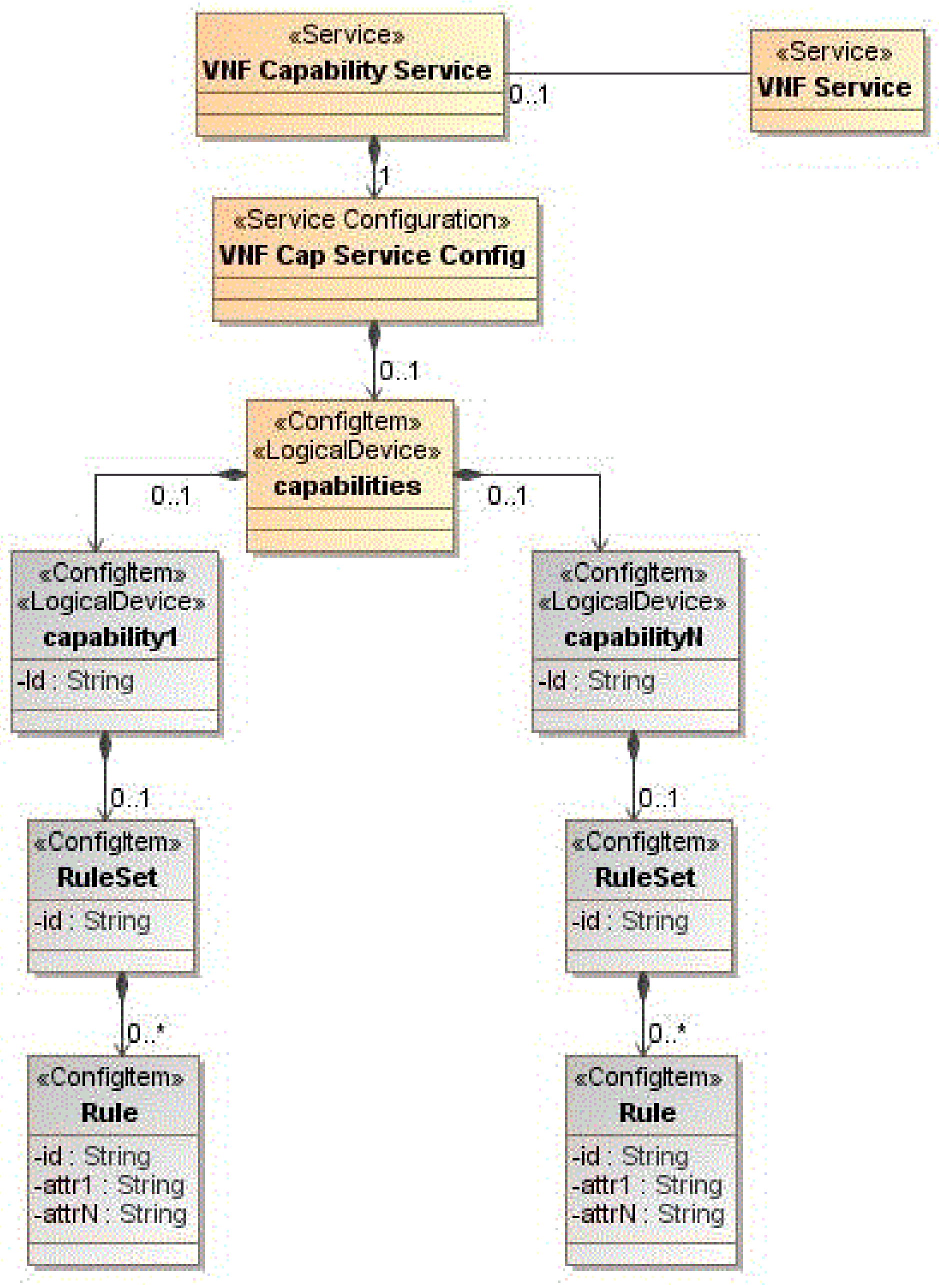
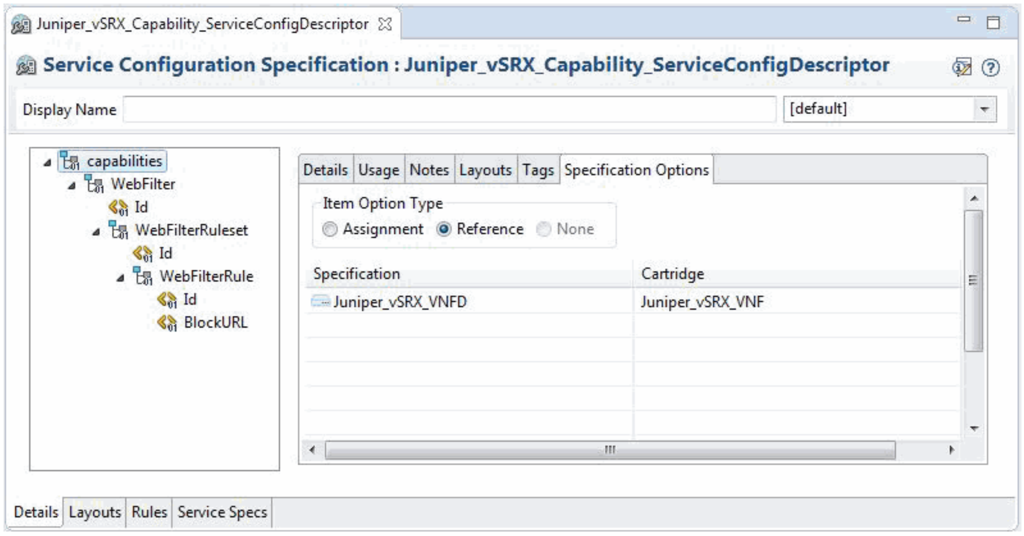


Figure 3–5 shows the Details view of the VNF capability service configuration in Design Studio.

Figure 3–5 VNF Capability Service Configuration Details


Juniper_vSRX_Capability_ServiceConfigDescriptor

Service Configuration Specification : Juniper_vSRX_Capability_ServiceConfigDescriptor

Display Name: [default]

- capabilities
  - WebFilter
    - Id
    - WebFilterRuleset
      - Id
      - WebFilterRule
        - Id
        - BlockURL

Item Option Type: Assignment [✓ Reference] None

Specification	Cartridge
Juniper_vSRX_VNFD	Juniper_vSRX_VNF

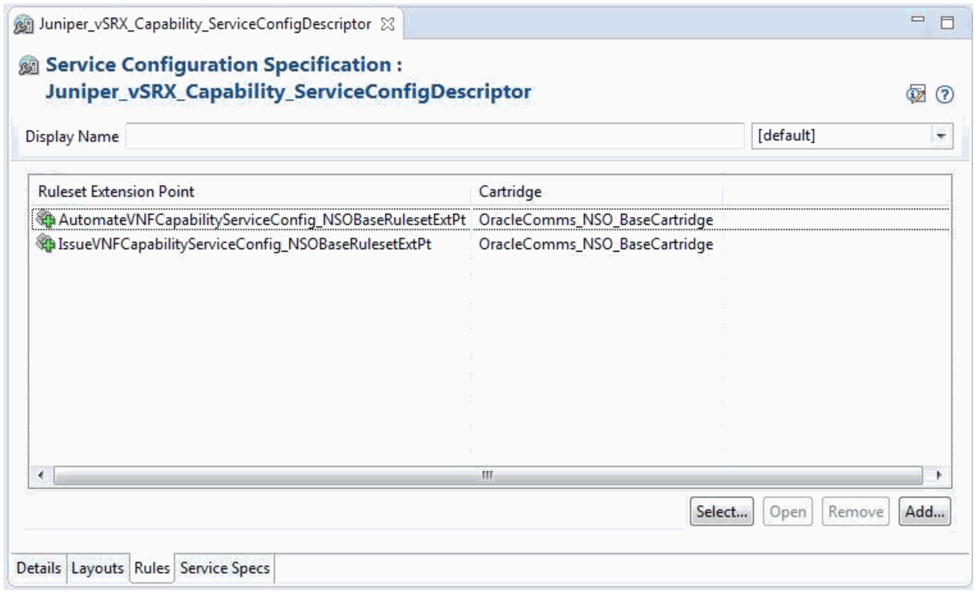
Table 3–2 shows the specifications that are associated to the configuration items in the VNF Capability service configuration.

Table 3–2 VNF Capability Service Configuration Speciations

Configuration Item	Associated Specification
capabilities	Juniper_vSRX_VNFD logical device specification as reference. This specification is included in the Juniper sample cartridge.
WebFilter	Juniper_vSRX_VNFD logical device specification as reference. This specification is included in the Juniper sample cartridge.

Figure 3–6 shows the ruleset extension points that are associated with the VNF capability service configuration.

Figure 3–6 Rules Associated to the VNF Capability Service Configuration


Juniper_vSRX_Capability_ServiceConfigDescriptor

Service Configuration Specification : Juniper_vSRX_Capability_ServiceConfigDescriptor

Display Name: [default]

Ruleset Extension Point	Cartridge
AutomateVNFCapabilityServiceConfig_NSOBaseRulesetExtPt	OracleComms_NSO_BaseCartridge
IssueVNFCapabilityServiceConfig_NSOBaseRulesetExtPt	OracleComms_NSO_BaseCartridge

Figure 3–7 shows how the Network Protection service is modeled in the sample Network Service cartridge.

Figure 3–7 Network Protection Service Model

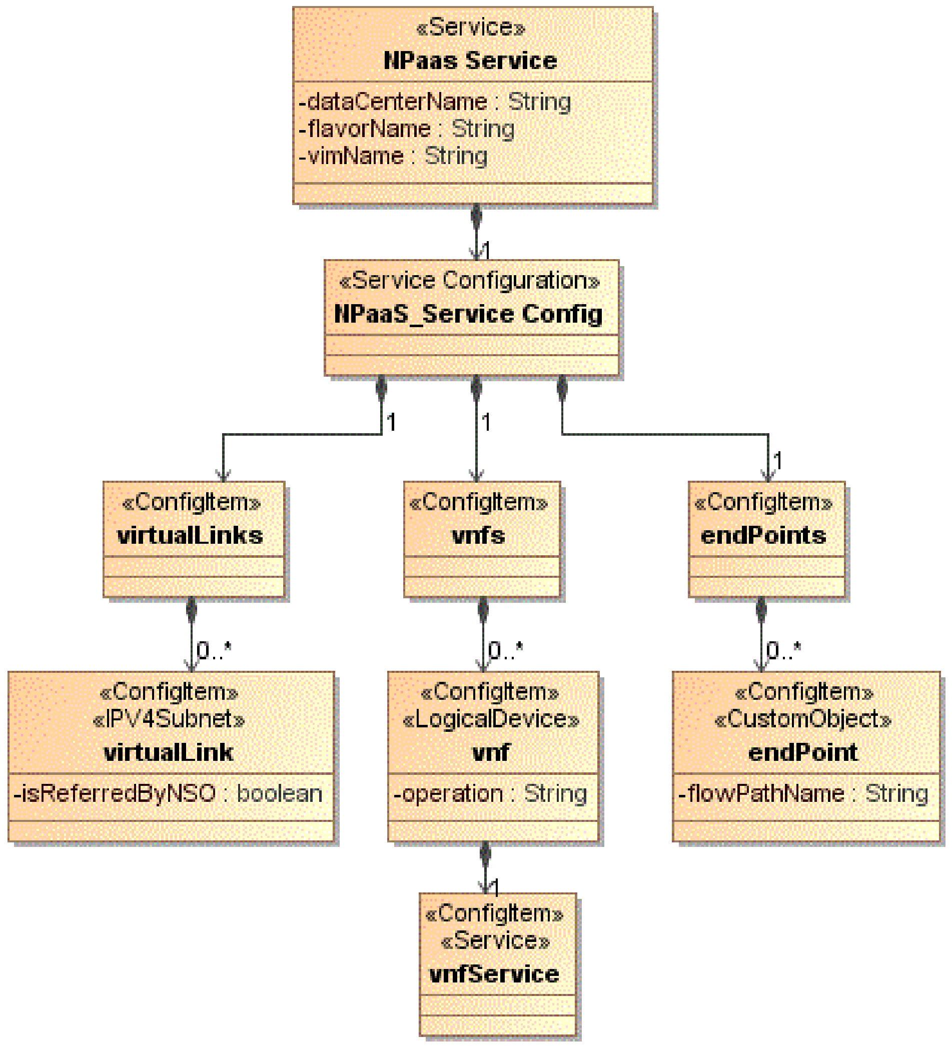
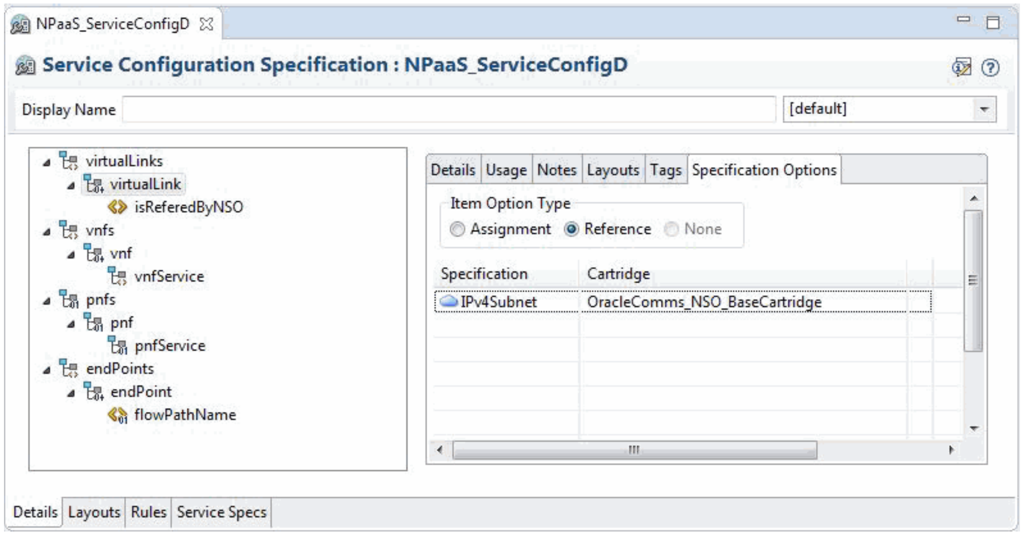


Figure 3–8 shows the details view of the Network Protection service configuration in Design Studio.

Figure 3–8 Network Protection Service Configuration Details



Display Name [default]

- virtualLinks
  - virtualLink
    - isReferedByNSO
- vnfs
  - vnf
    - vnfService
- pnfs
  - pnf
    - pnfService
- endPoints
  - endPoint
    - flowPathName

Item Option Type: Assignment [✓ Reference] None

Specification	Cartridge
IPv4Subnet	OracleComms_NSO_BaseCartridge

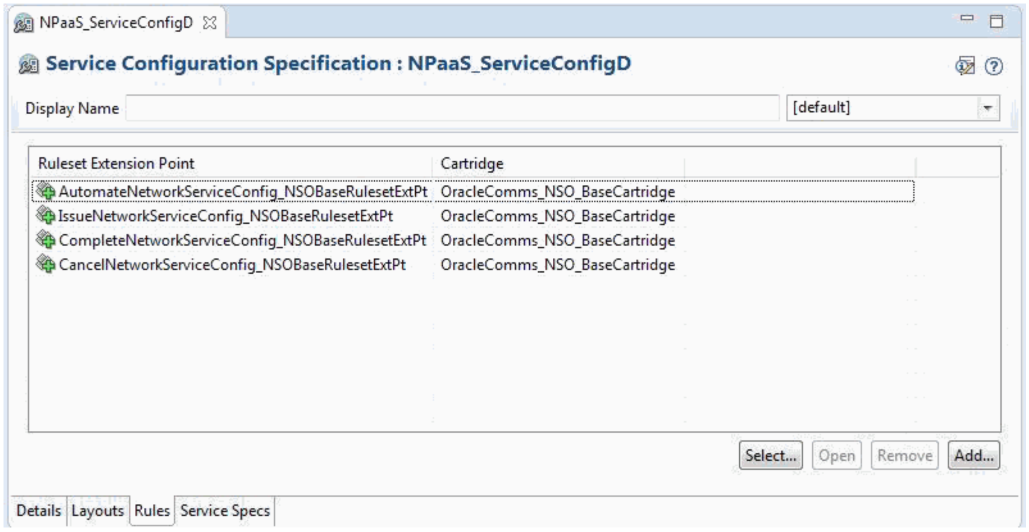
Table 3–3 shows the specifications that are associated to the configuration items in the Network Protection service configuration.

Table 3–3 Network Protection Service Configuration Specifications

Configuration Item	Associated Specification
virtualLink	IPV4Subnet specification as reference This specification is included in the OracleComms_NSO_BaseCartridge cartridge.
vnf	Juniper_vSRX_VNFD logical device specification as reference This specification is included in the Juniper sample cartridge.
endPoint	NSSubscriber custom object specification as reference. This specification is included in the OracleComms_NSO_BaseCartridge cartridge.

Figure 3–9 shows the ruleset extension points that are associated with the Network Protection service configuration.

Figure 3–9 Rules Associated to the Network Protection Service Configuration



NPaaS_ServiceConfigD

Service Configuration Specification : NPaaS_ServiceConfigD

Display Name [default]

Ruleset Extension Point	Cartridge
AutomateNetworkServiceConfig_NSOBaseRulesetExtPt	OracleComms_NSO_BaseCartridge
IssueNetworkServiceConfig_NSOBaseRulesetExtPt	OracleComms_NSO_BaseCartridge
CompleteNetworkServiceConfig_NSOBaseRulesetExtPt	OracleComms_NSO_BaseCartridge
CancelNetworkServiceConfig_NSOBaseRulesetExtPt	OracleComms_NSO_BaseCartridge

The solution includes sample descriptor files for the sample Network Protection service and the constituent VNFs. The descriptor files enable you to define the behavior of the network service and the VNFs.

- **NPaaS_NSD.xml**. This is the descriptor file for the Network Protection service.
- **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.

You open the descriptor files in Design Studio and specify the deployment requirements, operational behavior, and policies required by network services.

In the descriptor file:

- Specify the networks that you want to create or reference. In the descriptor file, networks are represented as virtual link descriptors.
- Specify the VNFs and the flow path that you want the network traffic to pass through.
- Specify the CPU utilization and other parameters for each VNF in the network service.

  See "About the Descriptor Files" for more information about the Network Service descriptor file.

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

The solution includes sample technical actions files for the VNFs. The technical actions files enable you to describe the actions for the VNFs in the VIM.

- **NPaaS_NSD_TechnicalActions.xml**. This is the technical actions file for the Network Protection service.
- **Juniper_vSRX_ServiceDescriptor_TechnicalActions.xml**. This is the technical actions file for the Juniper vSRX firewall VNF.
- **Checkpoint_NG_FW_ServiceDescriptor_TechnicalActions.xml**. This is the technical actions file for the Checkpoint NG virtual firewall.

Implementing a Network Service By Using the Sample Cartridges

The sample network protection service uses the following software components:

- UIM 7.3.3 and the Network Service Orchestration Solution 1.1.1 cartridges
- OpenStack VIM, with Open vSwitch capability
- OpenDaylight SDN Controller
- Software images for the firewall VNFs

Note: By default, the *UIM_Home*/**config**/**nso.properties** file displays the username and password of the Network Service Orchestration solution user in plain text. You must encrypt the password by running the **EncryptText** ruleset in UIM.

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.

To implement the network protection service:

1. In OpenStack, create a tenant or reference an existing tenant with administrator privileges.
2. Create a management network with the name **nfvo-mgmt** or reference an existing management network that can be shared by all the components of the solution. For example,

   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
3. 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.
4. Create a customer-side network that facilitates the customer's network traffic to reach the VNFs.

   Table 3–4 shows examples of IP addresses and IP address ranges of network and subnet configuration for the customer-side network.

Table 3–4 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

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

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

Table 3–5 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

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

   Table 3–6 shows examples of IP addresses and IP address ranges of network and subnet configuration for the packet-in network.

Table 3–6 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 3–7 shows examples of IP addresses and IP address ranges of network and subnet configuration for the packet-out network.

Table 3–7 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

7. Start the OpenDaylight virtual machine on the management network.
8. Start the Open vSwitch virtual machines on the management network, customer-side network, Internet-side network, packet-in network, and packet-out network.
9. On the Open vSwitch virtual machine, run the following commands:
   - 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.

10. Open the *UIM_Home*/**config**/**nso.properties** file and update the following parameters.

    - **startIpAddress**. Specify the subnet start IP address. By default, when the solution 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, the solution 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.

11. Open the *UIM_Home*/**config**/**NPaaS_NSD.properties** file and specify values for the parameters listed in Table 3–8:

Table 3–8 Parameters in the NPaaS Network Service Descriptor Properties File

Parameter	Description
NPaaS_ NSD.default.serviceArea	Specify a default service area for the NPaaS_NSD network service descriptor.
NPaaS_ NSD.default.dataCenter	Specify a default data center for the NPaaS_NSD network service descriptor.
NPaaS_ NSD.ManagementNetwork	Specify the name of the management network. The management network is the VLD Name that is specified in the **NPaaS_NSD.xml** file.
NPaaS_NSD.Data_IN	Specify the name of the data-in network.
NPaaS_NSD.Data_OUT	Specify the name of the data-out network.

Table 3–8 (Cont.) Parameters in the NPaaS Network Service Descriptor Properties File

Parameter	Description
sdnController.NPaaS_NSD	Specify an implementation class for the SDN controller interface. The default implementation class is **com.oracle.communications.inventory.nso.nfvi.sdn.ODLManager**.
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.

12. Redeploy the Network Service Orchestration solution cartridges in Design Studio. See "Configuring UIM for the Network Service Orchestration Solution" for information about deploying the cartridges in the specified order.
13. Register the VIM by calling the corresponding RESTful API. See "Registering the VIM" for instructions.
14. Discover the VIM resources. See "Discovering VIM Resources" for instructions.

Designing New Network Services and VNF Services

You can define and model network services and VNFs depending on the network functions that you want to virtualize on your network.

To define and model network services and VNFs, you work in Design Studio. In Design Studio, you define specifications and properties for your network services, VNFs, and their hierarchical and related components.

To model a network service with a VNF, you create two cartridges in Design Studio: one cartridge for the VNF and one cartridge for the network service.

In the cartridge for the Network Service, do the following:

- Specify the following UIM entity specifications:
  - One Service specification for the Network Service
  - One Service Configuration specification for the network service
- 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.
- Create custom code for extension.

In the cartridge for the VNF Descriptor, do the following:

- Specify the following UIM entity specifications:
  - One Service specification for the VNF
  - One Service Configuration specification for the VNF
  - A Logical Device specification for the VNF
- 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.
- Create a post-configuration template configuration file for the VNF. See "About the VNF Configuration Files" for more information.
- Create a template file for the VNF.
- Create custom code for extension.

4 Working with Network Services and VNFs

This chapter provides instructions for working with network services and VNFs in Oracle Communications Network Service Orchestration Solution.

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

- Instantiating a Network Service
- Upgrading the Software Version of a VNF
- Monitoring and Healing a VNF
- Modifying a Network Service
- Terminating a Network Service
- Retrieving Details About Network Services, VNFs, and Descriptors

Note: Based on the configurations that the VNFs in the network service require, these lifecycle 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.

Instantiating a Network Service

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.

Before you instantiate a network service, ensure that the VIM resources are discovered. See "Discovering VIM Resources" for information about discovering VIM resources.

When you instantiate a network service, you need to provide values for the required parameters in the API request. For details about the values and the parameters, look in the network service and VNF descriptor files that you created in Design Studio. See "Network Service Orchestration RESTful API Reference" for descriptions of the parameters.

To instantiate a network service:

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

   POST http://*host*:*port*/ocnso/1.1/ns

   where *host* is the hostname and *port* is the port number of the machine on which UIM is installed.

For a sample request and response about the network service instantiation API, see "Instantiate a Network Service".

2. In the request, specify values for the following parameters:
   - nsName
   - nsDescriptorName
   - serviceDeploymentFlavorName
   - vnfName
   - deploymentFlavorName
   - vnfDescriptorName
   - version
   - name
   - parameters
3. Ensure that you receive a success message and a response.
4. In Oracle Communications Unified Inventory Management (UIM), verify the following:
   - The network service and its configurations are created and are in **In Service** status.
   - The VNF service with configurations is created and associated to the network service.
   - The VNFs, 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.
5. In your VIM, verify the following:
   - The VNF instance is up and running.
   - The specified networks are either created or referenced.
   - The VNF is linked to the networks.

Based on the configurations you defined in the network service and the VNF descriptor files, the solution 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 the VNF Configuration Files" for more information.
- 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 on the VIM (or for any other reason), the solution rolls back the resources completely.

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.

To upgrade the software version of a VNF:

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

   PUT http://*host*:*port*//ocnso/1.1/vnf/*vnfId*/upgrade/*vnfVersion*

   where:

   - *vnfId* is the ID of the VNF that you want to upgrade
   - *vnfVersion* is the version number of the VNF image that you want to upgrade to

   For a sample request and response of this API, see "Upgrade the Software Version of a VNF".

2. In the request, specify the details about the VNF name and the software version of the VNF image that you want to upgrade to.
3. Ensure that you receive a success message and a response.
4. In UIM, do the following:
   - Verify that the network service is updated with a new service configuration version.
   - Verify that the version number of the VNF image that you upgraded the VNF to is listed.
5. In your VIM, verify that the VNF instance displays the name of the VNF image that you upgraded to.

Monitoring and Healing a VNF

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, the solution 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. If you use OpenStack Ceilometer, when you heal a failed VNF by replacing it, if the new VNF comes up in a different host, the solution performs resource orchestration to deduce the resources from the new host and the availability zone and adds up the resources count to the host.

You can integrate other third-party monitoring engines by using the extensions provided in the solution. 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.

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

   POST http://*host*:*port*//ocnso/1.1/vnf/*vmId*/heal

   where *vmId* is the ID of the VNF virtual machine that you want to heal.

   For a sample request and response of this API, see "Heal a VNF".
3. In the request, specify the details of the VNF that you want to heal and specify whether you want to reboot or replace the VNF.
4. Ensure that you receive a success message and a response.
5. In your VIM, verify that the VNF you rebooted or replaced is listed as active and running.

Modifying a Network Service

You modify a network service to either add or remove VNFs in a network service. You add a VNF to a network service to enable the network service to deliver additional service capabilities.

- Adding a VNF to a Network Service
- Deleting a VNF from a Network Service

Adding a VNF to a Network Service

To add a VNF to a network service:

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

   POST http://*host*:*port*//ocnso/1.1/ns/*networkServiceId*/vnfs

   where *networkServiceId* is the ID of the network service that you want to modify.

   For a sample request and response of this API, see "Add VNFs to a Network Service".
2. In the request, specify the details about the VNF that you want to add to the network service.
3. Ensure that you receive a success message and a response.
4. 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.
5. In your VIM, verify that a new VNF instance is created.

Deleting a VNF from a Network Service

To delete a VNF from a network service:

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

   DELETE http://*host*:*port*//ocnso/1.1/ns/*networkServiceId*/vnfs

   where *networkServiceId* is the ID of the network service that you want to modify.

   For a sample request and response of this API, see "Delete a VNF from a Network Service".

2. In the request, specify the details about the VNF that you want to remove from the network service.
3. Ensure that you receive a success message and a response.
4. 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.
5. In your VIM, verify that the VNF instance is removed and the resources that were assigned to the VNF are freed up.

Terminating a Network Service

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.

To terminate a network service:

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

   DELETE http://host:port/ocnso/1.1/ns/*networkServiceId*

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

   For details about this API, see "Terminate a Network Service".

2. Specify the details of the network service you want to delete.
3. Ensure that you receive a success message and a response.
4. In UIM, verify the following:
   - The status of the network service and the VNF services is changed to Disconnected.
   - The status of the logical device corresponding to the associated VNF is changed to Unassigned.
5. In your VIM, verify that the VNF instance is deleted and all the allocated resources are released.

Retrieving Details About Network Services, VNFs, and Descriptors

You can retrieve and view details about your network services, VNFs, network service descriptors, and VNF descriptors.

The solution provides RESTful APIs that you can call to retrieve and view different types of information about your network services and VNFs.

You can retrieve and view the following details about VNFs, network services, and descriptors:

- Information about a specific network service
- Information about the network forwarding paths for a network service
- List of available network service descriptors
- Network service descriptor information
- List of VNF descriptors supported by a network service descriptor
- List of flavors of a network service descriptor
- VNF descriptor information
- List of versions of the VNF descriptor
- List of VNF descriptor flavors

For details about the RESTful APIs, see "Network Service Orchestration RESTful API Reference".

5 Extending the Network Service Orchestration Solution

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

You can extend the functionality of the solution by:

- Designing cartridges in Oracle Communications Design Studio. See "Designing Cartridges for Custom VNFs and Network Services".

  For more information about designing cartridges:

  - See *UIM Concepts* to understand the concept of extending cartridge packs and the impact of doing so.
  - See *UIM Cartridge Guide* for information about the leading practices for extending cartridge packs.
  - See *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 Concepts*.

- Using extension points and Java interface extensions. See "Using Extension Points and Java Interface Extensions to Extend the Solution".

Setting Up Design Studio for the Network Service Orchestration Solution Cartridges

Before you design and work with cartridges for VNFs and network services, you must set up Design Studio.

To set up Design Studio for the Network Service Orchestration solution:

1. Create a local directory (*UIM_SDK_Home*).
2. Locate the **UIM_SDK.zip** folder in the UIM software pack and extract it into the *UIM_SDK_Home* local directory.

3. Create another local directory (*NSO_SDK_Home*).
4. Locate the **OracleComms_NSO_1.1.1.0.0**.*build_number*.**zip** file and extract it into the *NSO_SDK_Home* local directory.
5. Create another local directory (*OTHER_LIB_Home*) and copy the following WebLogic libraries from your WebLogic installation into the OTHER_LIB_Home 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**
6. Copy other UIM-specific JAR files to the OTHER_LIB_Home directory. See *UIM 7.3.3 Installation Guide* for information about UIM-specific JAR files.
7. In Design Studio, open a new workspace.
8. Navigate to *UIM_SDK_Home*/**cartridges** and import the following UIM base cartridges into Design Studio:
   - ora_uim_baseextpts
   - ora_uim_basemeasurements
   - ora_uim_basespecifications
   - ora_uim_basetechnologies
   - ora_uim_common
   - ora_uim_mds
   - ora_uim_model
9. Navigate to *NSO_SDK_Home*/**designStudio**/**cartridgeZips** and import the following Network Service Orchestration solution cartridges into Design Studio:
   - OracleComms_NSO_BaseCartridge
   - NPaaS_NetworkService
   - Checkpoint_NG_FW_VNF
   - Juniper_vSRX_VNF
10. In Design Studio, for the Network Service Orchestration Solution cartridge projects, configure the following Java build path classpath variables:
    - UIM_LIB. Specify the path as *UIM_SDK_Home*/**lib**
    - OTHER_LIB. Specify the path as *OTHER_LIB_Home*
    - NSO_LIB. Specify the path as *NSO_SDK_Home*/**designStudio**/**nso_lib**

Designing Cartridges for Custom VNFs and Network Services

To design cartridges for custom VNFs and network services:

1. In Design Studio, create new Inventory projects for the VNFs and the network service that you want to design.

   See Design Studio Help for instructions about creating cartridge projects.

2. For each VNF and network service Inventory project, create specifications, metadata, and technical action files by referring to the solution's sample cartridges.
3. For each service specification, create a technical action XML file. See "About the Technical Actions File" for more information.
4. If required, write custom ruleset extension points. See "Writing a Custom Ruleset Extension Point"for more information.
5. If required, extend the core functionality by using Java interface extensions. See "Using Java Interface Extensions" for more information.

Using Extension Points and Java Interface Extensions to Extend the Solution

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

- Writing a custom rule set 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 solution's core functionality by writing a custom rule set extension point and associating the extension point with the rule set in Design Studio.

To extend the solution's core functionality by using the base extension points:

1. In Groovy or Drools, write a ruleset that provides the additional functionality that you want to implement.
2. Write a rule set extension point by integrating the extension point and the ruleset with a placement of BEFORE, INSTEAD, or AFTER.
3. In Design Studio, relate the rule set extension point to the relevant specification.

Table 5–1 describes the Network Service Orchestration solution core APIs that can be extended by using the extension points in the solution.

Table 5–1 Network Service Orchestration Solution 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.

Table 5–1 (Cont.) Network Service Orchestration Solution Core APIs and Extension Points

API	Extension Point	Description
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 solution's core functionality 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.

The solution 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 manager. See "Implementing a Custom VNF Lifecycle 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" for more information.

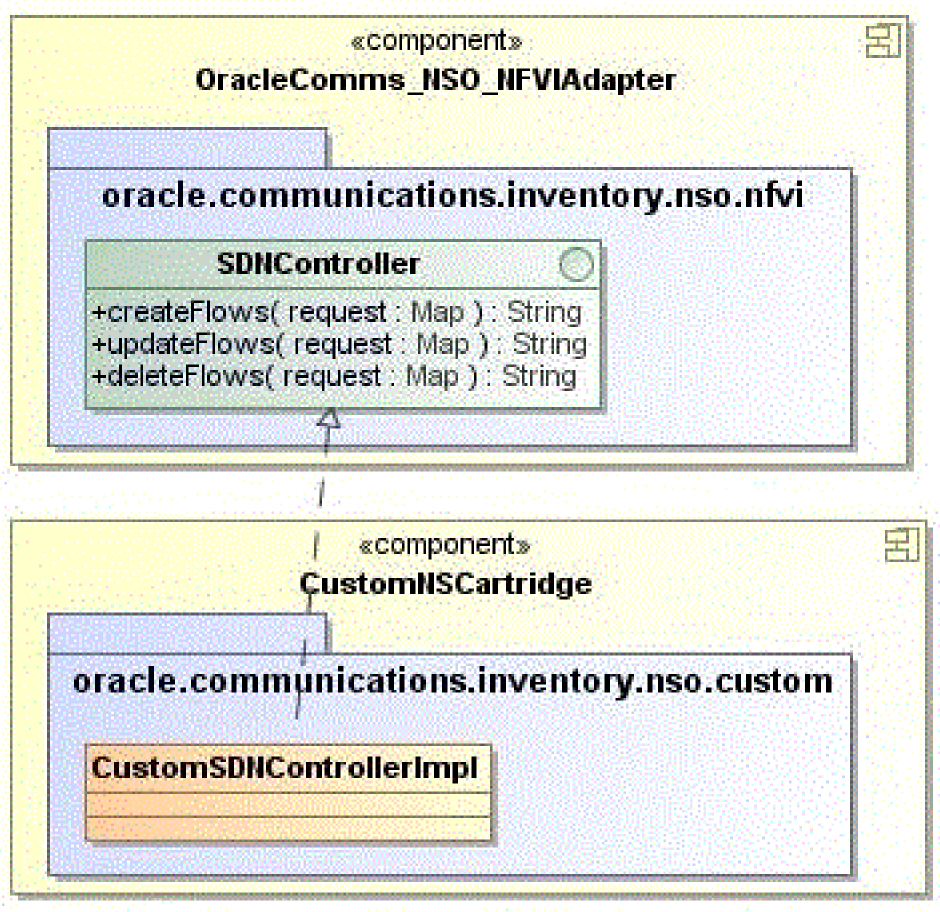
Note: If you change any keys in the **nso.properties** file and the **nfvi.properties** file, before redeploying the solution's core cartridges or upgrading the solution, take a backup of these files and use the latest files.

Implementing a Custom SDN Controller

By default, the solution supports integration with OpenDaylight, but you can also implement a custom SDN controller.

Figure 5–1 shows a model diagram that depicts how you can write an extension for an SDN controller in Design Studio.

Figure 5–1 Custom SDN Controller Model


«component»
OracleComms_NSO_NFVIAdapter
oracle.communications.inventory.nso.nfvi
SDNController
+createFlows(request : Map) : String
+updateFlows(request : Map) : String
+deleteFlows(request : Map) : String
«component»
CustomNSCartridge
oracle.communications.inventory.nso.custom
CustomSDNControllerImpl

To implement a custom SDN controller:

1. In the custom Network Service descriptor cartridge, 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 provided in the OracleComms_NSO_NFVIAdapter cartridge.
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, 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. Redeploy the cartridge.

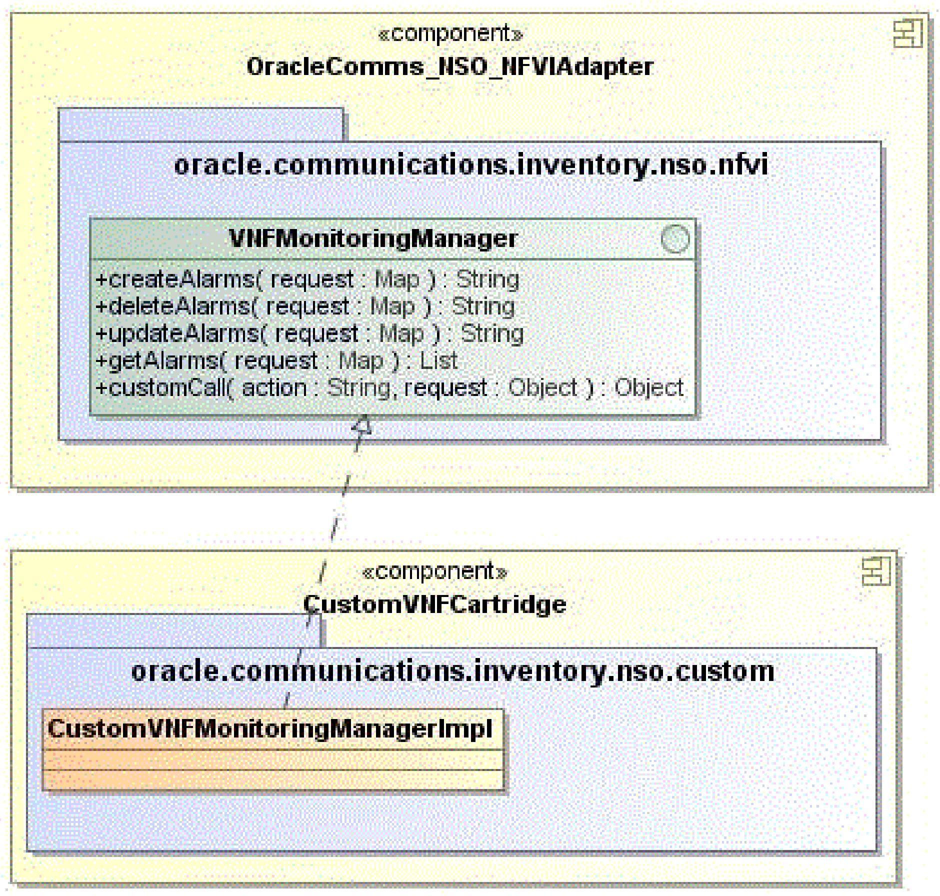
> **Note:** If the **sdnController**.*NSD_Name* key is commented out or if the path of the implementation class is not specified, the solution does not perform the network flow operations such as creation of flows, deletion of flows, and update of flows for the network service.

Implementing a Custom Monitoring Engine

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

Figure 5–2 shows a model diagram that depicts how you can write an extension for a custom VNF monitoring engine in Design Studio.

Figure 5–2 Custom Monitoring Engine Model


«component»
OracleComms_NSO_NFVIAdapter
oracle.communications.inventory.nso.nfvi
VNFMonitoringManager
+createAlarms(request : Map) : String
+deleteAlarms(request : Map) : String
+updateAlarms(request : Map) : String
+getAlarms(request : Map) : List
+customCall(action : String, request : Object) : Object
«component»
CustomVNFCartridge
oracle.communications.inventory.nso.custom
CustomVNFMonitoringManagerImpl

To implement a custom monitoring engine:

1. In the custom VNF descriptor cartridge, 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 provided in the OracleComms_NSO_NFVIAdapter cartridge.
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, 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. Redeploy the cartridge.

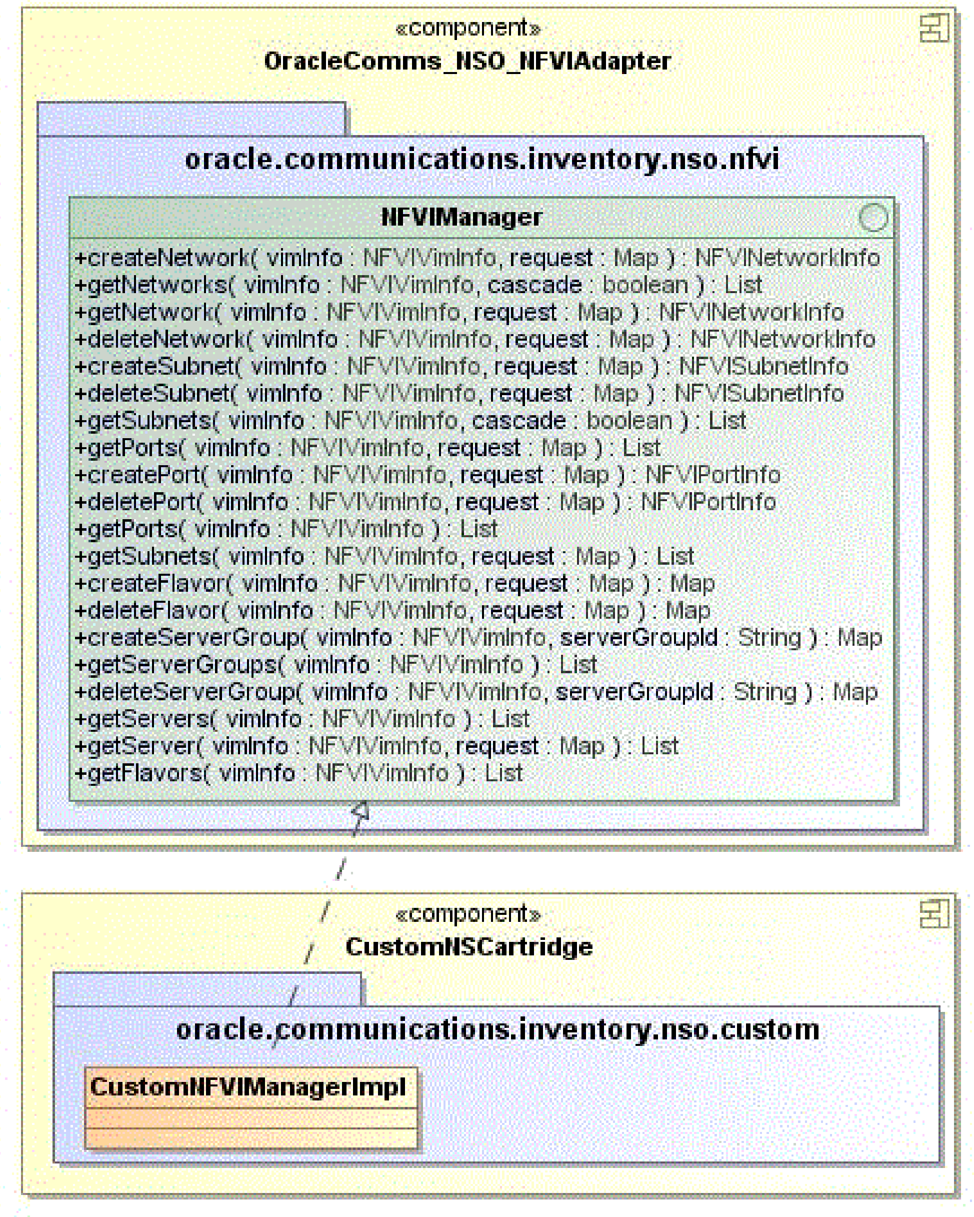
Note: If the **vnfMonitor**.*VNFD_Name* key is commented out or if the path of the implementation class is not specified, the solution does not perform monitoring operations such as creation, deletion, and update of alarms for the network service.

Implementing a Custom VIM

By default, the solution supports integration with OpenStack, but you can also implement a custom VIM.

Figure 5–3 shows a model diagram that depicts how you can write an extension for a custom VIM in Design Studio.

Figure 5–3 Custom VIM Model


«component»
OracleComms_NSO_NFVIAdapter
oracle.communications.inventory.nso.nfvi
NFVIManager
+createNetwork(vimInfo : NFVIVimInfo, request : Map) : NFVINetworkInfo
+getNetworks(vimInfo : NFVIVimInfo, cascade : boolean) : List
+getNetwork(vimInfo : NFVIVimInfo, request : Map) : NFVINetworkInfo
+deleteNetwork(vimInfo : NFVIVimInfo, request : Map) : NFVINetworkInfo
+createSubnet(vimInfo : NFVIVimInfo, request : Map) : NFVISubnetInfo
+deleteSubnet(vimInfo : NFVIVimInfo, request : Map) : NFVISubnetInfo
+getSubnets(vimInfo : NFVIVimInfo, cascade : boolean) : List
+getPorts(vimInfo : NFVIVimInfo, request : Map) : List
+createPort(vimInfo : NFVIVimInfo, request : Map) : NFVIPortInfo
+deletePort(vimInfo : NFVIVimInfo, request : Map) : NFVIPortInfo
+getPorts(vimInfo : NFVIVimInfo) : List
+getSubnets(vimInfo : NFVIVimInfo, request : Map) : List
+createFlavor(vimInfo : NFVIVimInfo, request : Map) : Map
+deleteFlavor(vimInfo : NFVIVimInfo, request : Map) : Map
+createServerGroup(vimInfo : NFVIVimInfo, serverGroupId : String) : Map
+getServerGroups(vimInfo : NFVIVimInfo) : List
+deleteServerGroup(vimInfo : NFVIVimInfo, serverGroupId : String) : Map
+getServers(vimInfo : NFVIVimInfo) : List
+getServer(vimInfo : NFVIVimInfo, request : Map) : List
+getFlavors(vimInfo : NFVIVimInfo) : List
«component»
CustomNSCartridge
oracle.communications.inventory.nso.custom
CustomNFVIManagerImpl

To implement a custom VIM:

1. In your custom cartridge, 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 provided in the OracleComms_NSO_NFVIAdapter cartridge.
3. Override the methods in the custom NFVI manager Java implementation class.
4. Open the *UIM_Home*/**config**/**nfvi.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

Note: If you change any keys in the **nfvi.properties** file, before redeploying the solution's core cartridges or upgrading the solution, take a backup of the **nfvi.properties** file and use the latest file.

---

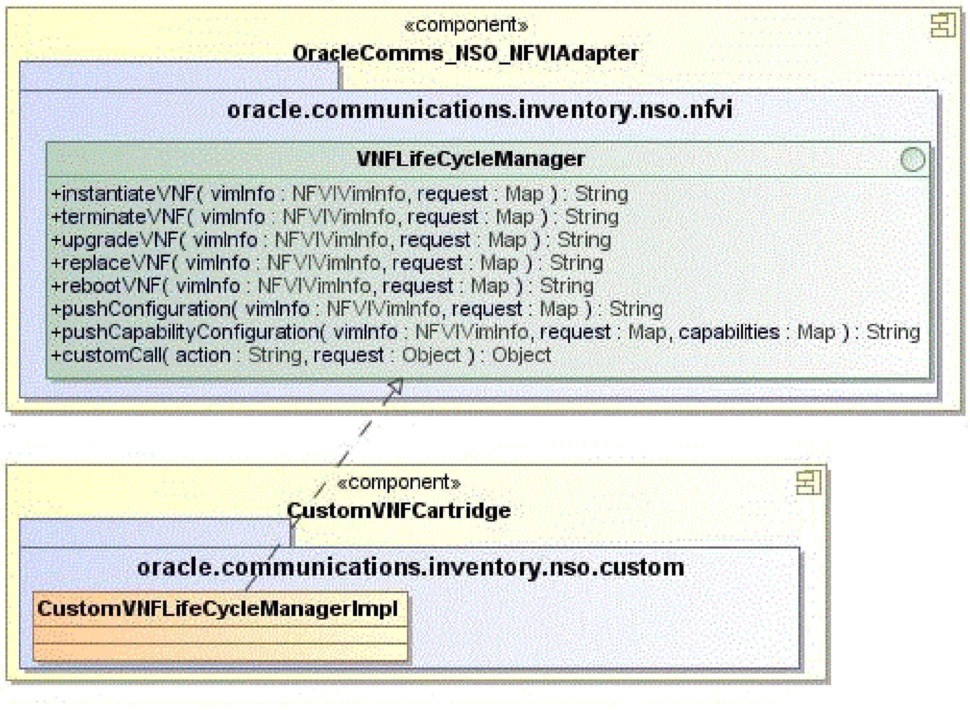
5. Redeploy the cartridge.

Implementing a Custom VNF Lifecycle Manager

By default, the solution manages the VNF lifecycle operations by using OpenStack Compute services (referred to as Nova), but you can also implement and use a custom VNF lifecycle manager with the solution.

Figure 5–4 shows a model diagram that depicts how you can write an extension for a custom VNF lifecycle manager in Design Studio.

Figure 5–4 Custom VNF Manager Model


«component»
OracleComms_NSO_NFVIAdapter
oracle.communications.inventory.nso.nfvi
VNFLifeCycleManager
+instantiateVNF(vimInfo : NFVIVimInfo, request : Map) : String
+terminateVNF(vimInfo : NFVIVimInfo, request : Map) : String
+upgradeVNF(vimInfo : NFVIVimInfo, request : Map) : String
+replaceVNF(vimInfo : NFVIVimInfo, request : Map) : String
+rebootVNF(vimInfo : NFVIVimInfo, request : Map) : String
+pushConfiguration(vimInfo : NFVIVimInfo, request : Map) : String
+pushCapabilityConfiguration(vimInfo : NFVIVimInfo, request : Map, capabilities : Map) : String
+customCall(action : String, request : Object) : Object
«component»
CustomVNFCartridge
oracle.communications.inventory.nso.custom
CustomVNFLifeCycleManagerImpl

To implement a custom VNF lifecycle manager:

1. In your custom cartridge, create a Java implementation class for the VNF lifecycle manager.
2. Configure the custom VNF lifecycle manager class to implement the **oracle.communications.inventory.nso.nfvi.VNFLifeCycleManager** interface, which is provided in the OracleComms_NSO_NFVIAdapter cartridge.
3. Override the methods in the custom VNF lifecycle manager Java implementation class.

4. Open the *UIM_Home*/**config**/**nfvi.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 lifecycle manager

   Note: If you change any keys in the **nfvi.properties** file, before redeploying the solution's core cartridges or upgrading the solution, take a backup of the **nfvi.properties** file and use the latest file.

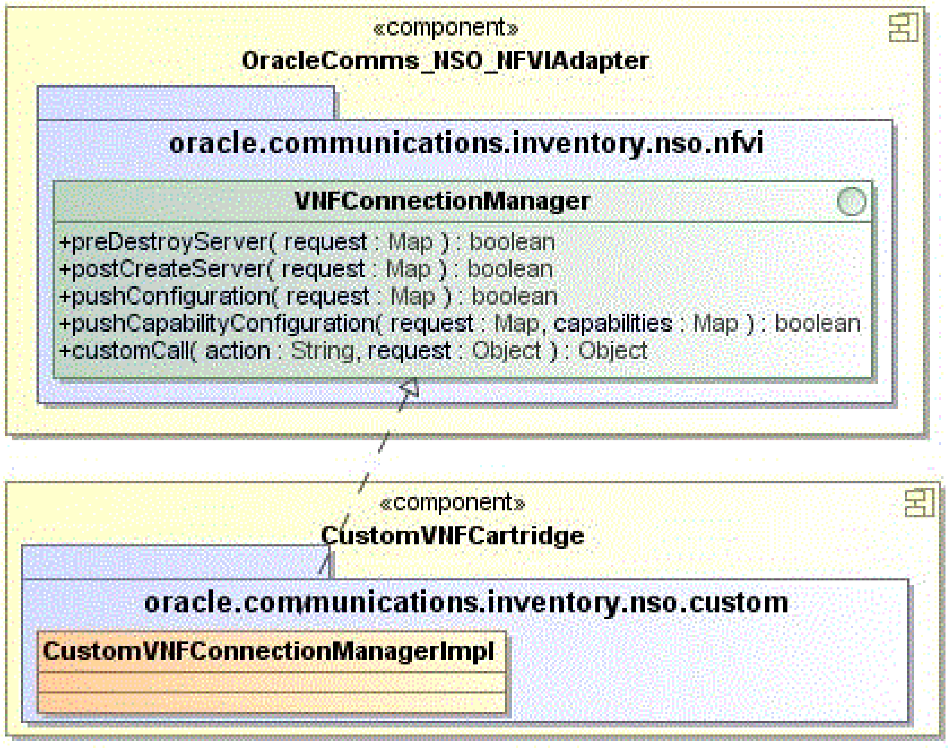
5. Redeploy the cartridge.

Implementing a Custom VNF Connection Manager

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

Figure 5–5 shows a model diagram that depicts how you can write an extension for a custom VNF connection manager in Design Studio.

Figure 5–5 Custom VNF Connection Manager Model


«component»
OracleComms_NSO_NFVIAdapter
oracle.communications.inventory.nso.nfvi
VNFConnectionManager
+preDestroyServer(request : Map) : boolean
+postCreateServer(request : Map) : boolean
+pushConfiguration(request : Map) : boolean
+pushCapabilityConfiguration(request : Map, capabilities : Map) : boolean
+customCall(action : String, request : Object) : Object
«component»
CustomVNFCartridge
oracle.communications.inventory.nso.custom
CustomVNFConnectionManagerImpl

To implement a custom VNF connection manager:

1. In the custom VNF descriptor cartridge, 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 provided in the OracleComms_NSO_NFVIAdapter cartridge.
3. Override the methods in the custom VNF connection manager Java implementation class.
4. In the VNF descriptor cartridge, 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. Redeploy the cartridge.

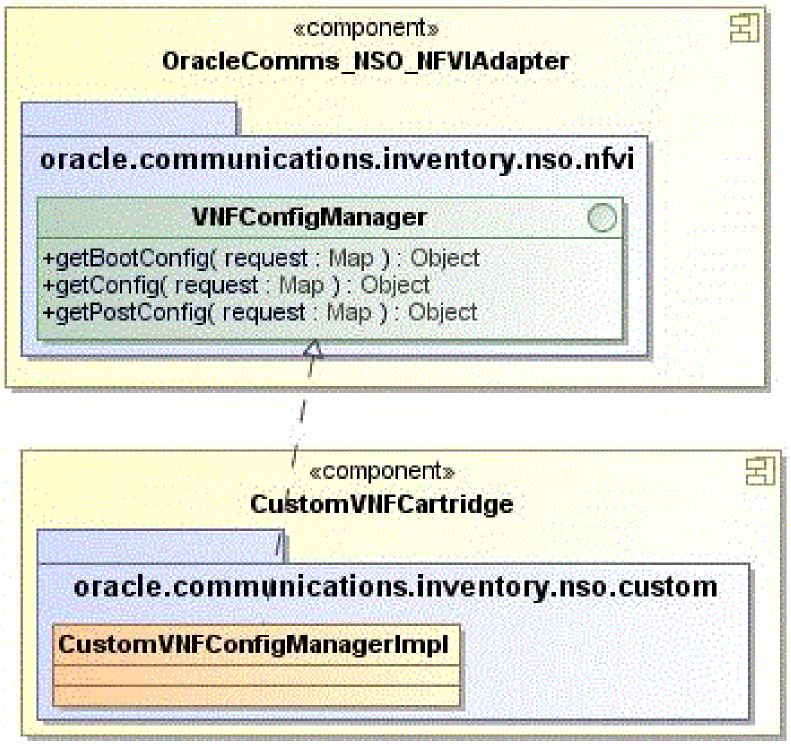
Note: If the **vnfConnectionMgr**.*VNFD_Name* key is commented out or if the path of the implementation class is not specified, the solution does not run configurations on the virtual machines on which the VNFs are deployed.

Implementing a Custom VNF Configuration Manager

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

Figure 5–6 shows a model diagram that depicts how you can write an extension for a custom VNF configuration manager in Design Studio.

Figure 5–6 Custom VNF Configuration Manager Model


«component»
OracleComms_NSO_NFVIAdapter
oracle.communications.inventory.nso.nfvi
VNFConfigManager
+getBootConfig(request : Map) : Object
+getConfig(request : Map) : Object
+getPostConfig(request : Map) : Object
«component»
CustomVNFCartridge
oracle.communications.inventory.nso.custom
CustomVNFConfigManagerImpl

To implement a custom VNF configuration manager:

1. In the custom VNF descriptor cartridge, 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 provided in the OracleComms_NSO_NFVIAdapter cartridge.
3. Override the methods in the custom VNF configuration manager Java implementation class.
4. In the VNF descriptor cartridge, 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. Redeploy the cartridge.

Note: If the **vnfConfigMgr.***VNFD_Name* key is commented out or if the path of the implementation class is not specified, the solution does not generate configurations for the VNF.

Implementing a Custom Response Manager

By default, the solution 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, 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 provided in the OracleComms_NSO_Common cartridge.
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.

   The solution supports multiple implementations of response manager.

   > **Note:** If you change any keys in the **nso.properties** file, before redeploying the solution's core cartridges or upgrading the solution, take a backup of the **nso.properties** file and use the latest file.

5. Redeploy the cartridge.

Localizing the Network Service Orchestration Solution

You can localize the UIM user interface, UIM Help, and the responses that the REST APIs return into your local language. If you have deployed the NSOBranding cartridge, you can also localize the UI labels of the NFV entities that are displayed in the UIM user interface.

To localize the Network Service Orchestration Solution:

1. Localize the UIM user interface and UIM Help. See the chapter about localizing UIM in *UIM 7.3.3 Developer's Guide*.
2. Localize the NFV entities in the UIM user interface. See "Localizing the NFV Entities in the UIM User Interface" for instructions.
3. Localize the responses that the RESTful APIs return. See "Localizing the Responses in RESTful APIs" for instructions.

Localizing the NFV Entities in the UIM User Interface

To localize the NFV entities in the UIM user interface:

1. Navigate to **designStudio****cartridgeZips** and import the NSOBranding sample cartridge into Design Studio.

2. Make a copy of the **model\content\inventory.ear\WEB-INF\classes\oracle\communciations\inventory\common\bundle\InventoryUIBundle.xlf** file and rename it as **InventoryUIBundle**_*localeID*_**.xlf**, where *localeID* is the locale ID of the language into which you want to localize the UI labels of the NFV entities.

   For example, if you want to localize the UI labels into French, rename the file to **InventoryUIBundle**_fr_FR_**.xlf**.

3. Open the file and replace the text in the **<source>** tags with the corresponding text in your local language.

   For example, to localize the Network Services label into French, change:

```
 <trans-unit id="MENU_NETWORK_SERVICES">
        <source>Network Services</source>
        <target/>
      </trans-unit>
```

   to

```
 <trans-unit id="MENU_NETWORK_SERVICES">
        <source>Services réseau</source>
        <target/>
      </trans-unit>
```

4. Compile and deploy the cartridge into UIM and restart the UIM server.

Localizing the Responses in RESTful APIs

To localize the responses in the Network Service Orchestration solution 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*.

6 Contents of the Network Service Orchestration JAR and ZIP Files

This chapter describes the contents of Oracle Communications Network Service Orchestration Solution JAR and ZIP files.

Table 6–1 describes the contents of the Network Service Orchestration JAR and ZIP files.

Table 6–1 Network Service Orchestration JAR and ZIP File Contents

Directory	Directory Content Description
deploy/ **individualJarsForSuperJar**	Contains individual JAR files that comprise the super JAR file. See "Network Service Orchestration Individual JAR Files" for more information.
deploy/ **superJarToDeploy**	Contains the super JAR file. See "Network Service Orchestration Super JAR File" for more information.
deploy/ **applications**	Contains applications. See "Network Service Orchestration Applications" for more information.
designStudio/ **cartridgeZips**	Contains cartridge project ZIP files. See "Network Service Orchestration ZIP Files" for more information.

Network Service Orchestration Individual JAR Files

The Network Service Orchestration cartridge contains individual JAR files that comprise the super JAR file. Each individual JAR file is deployable.

Note: Before deploying the Network Service Orchestration cartridge JAR files, you must deploy the base cartridges if not previously deployed. For information about the base cartridges, see "Configuring UIM for the Network Service Orchestration Solution" and *UIM Cartridge Guide*.

Oracle recommends that you deploy the super JAR file. If you deploy the JAR files individually, you must install them in a specific order.

The Network Service Orchestration cartridge individual JAR files are located in the **deploy/individualJarsForSuperJar** directory and they must be deployed in the following order:

- OracleComms_NSO_NFVIAdapter
- OracleComms_NSO_Common
- OracleComms_NSO_BaseCartridge

The Network Service Orchestration cartridge also contains individual JAR files for sample cartridges that comprise the super JAR file. Each individual JAR file is deployable.

- NPaaS_NetworkService
- Juniper_vSRX_VNF
- Checkpoint_NG_FW_VNF

Network Service Orchestration Super JAR File

The Network Service Orchestration cartridge contains the **OracleComms_NSO_*.jar** super JAR file. The solution super JAR is located in the **deploy\superJarToDeploy** directory.

Note: The asterisk in the JAR file name represents a five-segment release version number followed by a build number. The five-segment release version numbers represent the following:

- Major Version Number
- Minor Version Number
- Maintenance Pack
- Generic Patch
- Customer Patch

The Network Service Orchestration super JAR file contains the entire contents of the solution and is ready for deployment.

See *UIM Cartridge Guide* for more information about deploying cartridges into Oracle Communications Unified Inventory Management (UIM).

Network Service Orchestration Applications

The Network Service Orchestration cartridge contains the **OracleComms_NSO_WebServices.war** application file in the **deploy\applications\** directory.

The **OracleComms_NSO_WebServices.war** file contains the implementation of the Network Service Orchestration solution web services.

Network Service Orchestration ZIP Files

The Network Service Orchestration cartridge contains one project ZIP file for every cartridge or model project, and each ZIP file contains a project in its pre-compiled state (the project name is the root directory).

The solution cartridge contains the following cartridge ZIP files, which are located in the **designStudio\cartridgeZips\** directory:

- NPaaS_NetworkService
- Juniper_vSRX_VNF
- Checkpoint_NG_FW_VNF
- NSOBranding
- OracleComms_NSO_BaseCartridge

7 Network Service Orchestration RESTful API Reference

This chapter provides reference information about the Oracle Communications Network Service Orchestration Solution RESTful APIs.

The Network Service Orchestration RESTful APIs provide the northbound interface to the Network Service Orchestration solution. Operation Support Systems (OSS) and VNF managers query data from the solution's resource inventory.

The solution's RESTful APIs 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 host name

  – *port* is the port number of the machine on which Oracle Communications Unified Inventory Management (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 APIs, in your RESTful API client, choose Basic Authentication and specify the username 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 Oracle WebLogic Server documentation for information about configuring TrustStore.

List of Network Service Orchestration Solution RESTful API Resources

Table 7–1 lists the Network Service Orchestration RESTful API resources.

Table 7–1 Network Service Orchestration Solution RESTful API Resources

Task	Method	Resource	Description
Register a Virtual Infrastructure Manager (VIM)	POST	/ocnso/1.1/vim	Registers the IP address, port, username and password of the VIM with the solution.
Discover VIM resources	POST	/ocnso/1.1/vim/*vimId*/discovery?infoLevel=*vim_information*	Discovers the resources of the registered VIM into the solution.
Update a VIM	POST	/ocnso/1.1/vim/update/*vimId*	Updates the IP address, port, username, password, and project name of an existing VIM in the solution.
Instantiate a network service	POST	/ocnso/1.1/ns	Instantiates a network service and its constituent VNFs.
Terminate a network service	DELETE	/ocnso/1.1/ns/*networkServiceId*	Terminates a network service and the constituent VNFs.
Upgrade VNF software version	PUT	/ocnso/1.1/vnf/*vnfId*/upgrade	Upgrades the software image version of a VNF.
Heal a VNF	POST	/ocnso/vnf/*vnfId*/heal?action=*reboot* /ocnso/vnf/*vnfId*/heal?action=*replace*	Heals a VNF by rebooting or replacing the VM. Available values for the action parameter are: ■ Replace ■ Reboot
Add VNFs to a network service	POST	/ocnso/1.1/ns/*networkServiceId*/vnfs	Adds VNFs to a network service.
Scale a VNF	POST	/ocnso/1.1/ns/*networkServiceId*/scale/*vnfId*	Scales a VNF in a network service.
Configure VNF service capabilities	POST	/ocnso/1.1/vnf/configure	Configures the capabilities of a VNF service.
Delete a VNF from a network service	DELETE	/ocnso/1.1/ns/*networkServiceId*/vnfs	Deletes a VNF from a network service.
Get VIM information	GET	/ocnso/1.1/vim/*vimId*	Returns information about a VIM that is registered with the solution
Get network service information	GET	/ocnso/1.1/nsd/*networkServiceId*	Returns information about a network service.
Get a list of all network service descriptors that are deployed in the solution	GET	/ocnso/1.1/nsd	Returns a list of all network service descriptors that are deployed in the solution.
Get details about a network service descriptor	GET	/ocnso/1.1/nsd/*nsdName*	Returns details about a network service descriptor.
Get a list of VNF descriptors in a network service descriptor	GET	/ocnso/1.1/nsd/*nsdName*/vnfds	Returns a list of VNF descriptors in a network service descriptor.
Get network service descriptor deployment 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 details about a VNF Descriptor	GET	/ocnso/1.1/vnfd/*vnfdName*	Returns details about a VNF descriptor.
Get a list of VNF descriptors in a network service descriptor	GET	/ocnso/1.1/nsd/*nsdName*/vnfds	Returns a list of VNF descriptors in a network service descriptor.
Get a list of supported versions for a VNF descriptor	GET	/ocnso/1.1/vnfd/*vnfdName*/versions	Returns a list of supported versions for a VNF descriptor.
Get VNF descriptor deployment flavors	GET	/ocnso/1.1/vnfd/*vnfdName*/flavors	Returns a list of deployment flavors that are defined for a VNF descriptor.
Get a list of all active network services that are created based on a specific network service descriptor	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 details about a network service	GET	/ocnso/1.1/ns/*networkServiceId*	Returns details about a network service.

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

Task	Method	Resource	Description
Get details about VNFs in a network service	GET	/ocnso/1.1/ns/*networkServiceId*/vnfs	Returns details about VNFs in a network service.
Get details about networks in a network service	GET	/ocnso/1.1/ns/*networkServiceId*/networks	Returns details about networks in a network service.
Get details about endpoints in a network service	GET	/ocnso/1.1/ns/*networkServiceId*/endpoints	Returns details about endpoints in a network service.
Get status information of a network service	GET	/ocnso/1.1/ns/*networkServiceId*/status	Returns status information of a network service.
Get details about a VNF	GET	/ocnso/1.1/ns/vnf/*vnfId*	Returns details about a VNF.
Get status information of a VNF	GET	/ocnso/1.1/ns/*vnfId*/status	Returns status information about a VNF.

HTTP Response Status Codes

Table 7–2 describes the HTTP response status codes for the GET, POST, PUT, and DELETE operations of the Network Service Orchestration Solution RESTful APIs.

Table 7–2 HTTP Response Status Codes for the REST APIs

Response Code	Description
200 OK	The request is successful. The information returned in the response is dependent on the method used in the request. For example: ■ GET. An entity corresponding to the requested resource is sent in the response. ■ POST. An entity describing or containing the result of the action.
202 Accepted	The request has been accepted for processing, but the processing has not completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place.
400 Bad Request	The request could not be understood by the server due to incorrect syntax. Do not repeat the request without correcting the syntax.
404 Not Found	The server has not found a matching request or URI.
500 Internal Server Error	The server encountered an unexpected condition which prevented it from fulfilling the request.

Sample Requests and Responses

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

Register a VIM

Registers the following details about the VIM with the Network Service Orchestration solution:

- IP address
- Port
- Username
- Password

Method

POST

URL

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

Sample JSON Request

```
{
"id":"vimId",
"name":"vimName",
"host":"11.111.111.1",
"port":"12345",
"userName":"nso",
"pswd":"***",
"projectName":"test",
"domainName":"default",
"vimType":"default",
"version":"3",
"sslEnabled":"false",
"cpuOvercommitRatio":"15",
"memoryOvercommitRatio":"1.5",
"diskOvercommitRatio":"1.0"
}
```

Table 7–3 describes the parameters in the request.

Table 7–3 Request Parameters

Parameter	Value	Required	Description
name	String	No	Name for the VIM.
id	String	Yes	Unique ID for the VIM in NSO.
host	String	Yes	IP address or host name of the VIM.
port	String	Yes	Port number of the VIM.
userName	String	Yes	Username of the VIM.
pswd	String	Yes	Password of the VIM.
projectName	String	Yes	Name of the project.
domainName	String	No	Name of the domain.
version	String	No	Version number of the VIM that you use. If you use OpenStack Kilo: ■ For OpenStack Keystone version 2, specify 2. ■ For OpenStack Keystone version 3, specify 3. This is the default.
sslEnabled	String	No	Specify whether you have SSL enabled for the VIM. Values are: ■ **true**. Indicates that you have SSL enabled for the VIM. ■ **false**. Indicates that you do not have SSL enabled for the VIM. This is the default.
vimType	String	Yes	Type of the VIM. The default is OpenStack.

Table 7–3 (Cont.) Request Parameters

Parameter	Value	Required	Description
cpuOvercommitRatio	Double	No	Ratio of over-committed virtual CPUs on the VIM
memoryOvercommitRatio	Double	No	Ratio of over-committed memory on the VIM
diskOvercommitRatio	Double	No	Ratio of over-committed disk on the VIM

Sample JSON Response

vimDataCenter is successfully registered with NSO.

Discover VIM Resources

Discovers the resources that are available on the VIM. In UIM, 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=*vim_information*

where:

- *vimId* is the Id of the VIM 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 values are:
  - **summary**. Retrieves and displays a summary of the VIM resources.
  - **details**. Retrieves and displays complete details about all the VIM resources.

Sample Request

This API does not require any request parameters.

Sample Response

This API returns the following response if you set the **infoLevel** parameter in the URL to **summary**:

```
{
   "data": {
     "summary": {
       "Number of Subnets": 0,
       "Number of Flavors": 7,
       "Number of Hosts": 0,
       "Number of Networks": 4,
       "Number of Zones": 3
```

```
      }
    }
}
```

The API returns the following response if you set the **infoLevel** parameter in the URL to **details**:

```
{
   "data": {
     "availabilityZones": [
       {
         "zone": "CustomerTermination",
         "hosts": [
           "compute2"
         ]
       },
       {
         "zone": "nova",
         "hosts": [
           "compute4"
         ]
       },
       {
         "zone": "InternetTermination",
         "hosts": [
           "compute3"
         ]
       }
     ],
     "networks": [
       {
         "network": "ext-net",
         "subnets": [
           "ext-subnet"
         ]
       },
       {
         "network": "825158_ManagementNetwork",
         "subnets": [
           "825158_ManagementNetwork"
         ]
       },
       {
         "network": "975005_Data_IN",
         "subnets": [
           "975005_Data_IN"
         ]
       },
       {
         "network": "825158_Data_OUT",
         "subnets": [
           "825158_Data_OUT"
         ]
       },
     ],
     "flavors": [
       "m1.small",
       "m1.large",
       "csr_flavor",
       "m1.medium",
       "vsrx.medium",
        "checkpoint",
        "Gold"
      ]
    }
  }
```

Update a VIM

Updates the following details about an existing VIM in the solution:

- IP address
- Port
- Username
- Password
- Project Name

Method

PUT

URL

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

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

Sample Request

```
{
"host":"11.111.1.11",
"port":"12345",
"userName":"admin",
"pswd":"****",
"projectName":"test",
"domainName":"default",
"version":"3",
"sslEnabled":"false",
"cpuOvercommitRatio":"15",
"memoryOvercommitRatio":"1.5",
"diskOvercommitRatio":"1.0"
}
```

Sample Response

VIM is updated successfully.

Instantiate a Network Service

Creates networks and the constituent resources and starts the VNFs in the network service.

Method

POST

URL

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

Sample Request

```
 {
    "nsName":"NPaaS1",
    "nsDescriptorName":"NPaaS_NSD",
    "serviceDeploymentFlavorName":"Checkpoint",
    "vnfs":[
        {
            "vnfName":"Vnf1",
            "deploymentFlavorName":"checkpoint",
            "vnfDescriptorName":"Checkpoint_NG_FW_VNFD",
            "version":"1.0"
        }
    ],
    "endPoints": [
        {
            "name":"Test110301",
            "forwardingGraphDescriptorName":"Data",
            "parameters":
            [
                {
                    "name": "ipAddress",
                    "value": "207.123.34.2"
                },
                {
                    "name": "vlanId",
                    "value": "101"
                },
                {
                    "name": "serviceLocation",
                    "value": "city03"
                }
            ]
        }
    ]
}
```

Table 7–4 describes the parameters in the request.

Table 7–4 Request Parameters

Parameter Name	Value	Required	Description
nsName	String	Yes	Name of the network service that you want to instantiate.
nsDescriptorName	String	Yes	Name of the network service descriptor.
serviceDeployment FlavorName	String	Yes	Name of the Network Service deployment flavor.
vnfs:[vnfName	String	Yes	Name of the VNF in the network service that you want to instantiate.
vnfs:[deploymentFlavorName	String	Yes	Name of the deployment flavor of the VNF service.
vnfs:[vnfDescriptorName	String	Yes	Name of the VNF descriptor that contains the descriptor of the VNF.
vnf:[version	String	Yes	Version number of the VNF image.
endPoints:[name	String	Yes	Name of the endpoint.

Table 7–4 (Cont.) Request Parameters

Parameter Name	Value	Required	Description
endPoints:[forwardingGraphDescriptorName	String	Yes	Name identifier that is defined in the Network Service metadata descriptor for which a sequence of VNFDs is defined for flow.
endPoints:[parameters:[name	String	Yes	Dynamic parameter that can be sent from the request to implement custom logic by using the extensions in the solution.
endPoints:[parameters:[value	String	Yes	Value of the customer-side end point termination points.

Sample Response

```
{
"networkServiceName": "NSO_NPassService_1",
"networkServiceId": "2475008",
"networkServiceStatus": "PENDING",
"businessInteractionId": "2475017",
"message": "[INV-992902] Network Service instantiation is in progress.",
"status": 202,
"vnfs": [
 {
      "vnfId": "2250005",
      "vnfName": "NSO_CheckPointVNF_1",
      "vnfStatus": "PENDING_ASSIGN",
      "vnfDescriptor": "Checkpoint_NG_FW_VNFD",
      "vnfServiceId": "2475009",
      "businessInteractionId": "2475018"
  }
 ]
}
```

Terminate a Network Service

Terminates a network service. 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*

Sample Request

This API does not require parameters. Specify the network service ID in the URL.

Sample Response

```
{
"networkServiceName": "NSO_NPassService_1",
"networkServiceId": "2475008",
"networkServiceStatus": "PENDING_DISCONNECT",
"businessInteractionId": "2475025",
"message": "[INV-992907] Network Service termination is in progress.",
"status": 202,
"vnfs": [
    {
      "vnfId": "2250005",
      "vnfName": "VNF1",
      "vnfStatus": "PENDING_UNASSIGN",
      "vnfDescriptor": "Juniper_vSRX_VNFD",
      "vnfServiceId": "2475009",
      "businessInteractionId": "2475026"
    }
  ]
}
```

Upgrade the Software Version of a 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/*vnf_version*

Sample Request

This API does not require parameters. Specify the VNF ID and the VNF version number you want to upgrade to in the URL.

Table 7–5 describes the parameters in the request.

Table 7–5 Request Parameters

Parameter Name	Value	Required	Description
vnfId	String	Yes	Id of the VNF whose image you want to upgrade.
vnf_Version	String	Yes	Version number of the VNF image that you want to upgrade to. This version number should already be defined in the VNF descriptor file.

Sample Response

```
{
  "vnfId": "975022",
  "interactionId": "975028",
  "message": "Network Service upgrade is under process.",
  "status": 202
}
```

Heal a VNF

Heals a VNF by either rebooting or replacing a VNF in the VIM.

Method

POST

URL

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

- *vnf_Id* is the VNF ID
- *action* is the action that you want to perform on the VNF. Specify any one of the following values:
  - **reboot**. Reboots the VNF.
  - **replace**. Replaces the VNF.

Sample Request

Request parameters are not required for this API.

Sample Response

reboot

```
VNF has been rebooted successfully.
```

replace

```
VNF has been replaced successfully. The new VM Id is
84068628-9d4b-415c-9d63-181cadc9b20d.
```

Add VNFs to a Network Service

Adds VNFs to an existing network service.

Method

POST

URL

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

Sample Request

```
[
    {
         "vnfName":"VNF1",
         "deploymentFlavorName":"checkpoint",
         "vnfDescriptorName":"Checkpoint_NG_FW_VNFD",
         "version":"1.0"
    },
    {
         "vnfName":"VNF2",
         "deploymentFlavorName":"checkpoint",
         "vnfDescriptorName":"Checkpoint_NG_FW_VNFD",
         "version":"1.0"
    }
]
```

Table 7–6 describes the parameters in the request.

Table 7–6 Request Parameters

Parameter Name	Value	Required	Description
vnfName	String	Yes	Name of the VNF that you want to add.
deploymentFlavorName	String	Yes	Name of the VNF deployment flavor.
vnfDescriptorName	String	Yes	Name of the VNF descriptor.
version	String	Yes	Version number of the VNF image.

Sample Response

```
{
"networkServiceName": "NSO_NPassService_1",
"networkServiceId": "2475008",
"businessInteractionId": "2475021",
"message": "[INV-992903] Adding VNF to Network Service is in progress.",
"status": 202,
"vnfs": [
  {
    "vnfId": "2250007",
    "vnfName": "VNF1",
    "vnfStatus": "PENDING_ASSIGN",
    "vnfDescriptor": "Checkpoint_NG_FW_VNFD",
    "vnfServiceId": "2475011",
    "businessInteractionId": "2475022"
  },
  {
   "vnfId": "2250008",
   "vnfName": "VNF2",
   "vnfStatus": "PENDING_ASSIGN",
   "vnfDescriptor": "Checkpoint_NG_FW_VNFD",
   "vnfServiceId": "2475012",
   "businessInteractionId": "2475023"
  }
 ]
}
```

Scale a VNF

Scales a VNF in a network service.

Method

POST

URL

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

Sample Request

Request parameters are not required

Sample Response

```
{
"networkServiceName": "NSO_NPassService_1",
"networkServiceId": "2475008",
"businessInteractionId": "2475019",
"message": "[INV-992913] VNF Scaling is in progress.",
"status": 202,
"vnfs": [
  {
    "vnfId": "2250006",
    "vnfName": "Juniper_vSRX_VNFD_1460550991411",
    "vnfStatus": "PENDING_ASSIGN",
    "vnfDescriptor": "Juniper_vSRX_VNFD",
    "vnfServiceId": "2475010",
    "businessInteractionId": "2475020"
  }
 ]
}
```

Delete a VNF from a Network Service

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

Method

DELETE

URL

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

Sample Request

```
[
{
"vnfId":"11"
}
]
```

where vnfId is the Id of the VNF in UIM. The VNF is represented as a logical device in UIM.

Sample Response

```
{
"networkServiceName": "NSO_NPassService_1",
"networkServiceId": "2475008",
"businessInteractionId": "2475023",
"message": "[INV-992904] Deleting VNF from Network Service is in progress.",
"status": 202,
"vnfs": [
  {
    "vnfId": "11",
    "vnfName": "VNF1",
    "vnfStatus": "PENDING_UNASSIGN",
    "vnfDescriptor": "Juniper_vSRX_VNFD",
    "vnfServiceId": "2475011",
    "businessInteractionId": "2475024"
  }
 ]
}
```

Configure VNF Service 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

```
{
   "vnfId" : "vnfID",
   "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": 202,
    "message": "VNF Configuration is in progress.",
    "vnfId": "12345",
    "vnfName": "VNF1",
    "vnfStatus" : "ASSIGNED",
    "vnfDescriptor":"Juniper_VSRX_VNFD",
    "vnfServiceName":"vSRX123",
    "vnfServiceStatus":"ASSIGNED",
    "vnfServiceDescriptor":"Juniper_vSRX_ServiceDescriptor"
}
```

Get VIM Details

Retrieves the details of a VIM that is registered with the solution.

Method

GET

URL

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

where *vimId* is the ID of the VIM

Sample Response

```
{
  "id": "VIMCloudTest",
  "name": "VIMCloudTest",
  "host": "10.133.0.31",
  "port": "35357",
  "userName": "admin",
  "pswd": "****",
  "projectName": "admin",
  "domainName": "default",
  "vimType": "OpenStack",
  "version": "3",
  "sslEnabled": true,
  "cpuOvercommitRatio": "15",
  "memoryOvercommitRatio": "1.5",
  "diskOvercommitRatio": "1.0"
}
```

Get List of Network Services

Retrieves the list of active network services that are defined in a 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 file

Sample Response

```
[
 {
    "nsID": "17",
    "nsdName": "NPaaS_NSD",
    "nsName": "NSO_QA_NPaaS_mgf_1_Service",
    "status": "IN_SERVICE"
  },
  {
    "nsID": "23",
    "nsdName": "NPaaS_NSD",
    "nsName": "NSO_QA_NPaaS_mgf_3_Service",
    "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 network service ID

Sample Response

```
{
  "nsID": "75191",
  "nsdName": "NPaaS_NSD",
  "nsName": "NPassService_Juniper55_Service",
  "status": "IN_SERVICE",
  "vimId": "VIMCloudTest",
  "biID": "75407",
  "networks": [
    {
      "networkName": "75191_Data_OUT",
      "status": "REFERENCED"
    },
    {
      "networkName": "75191_Data_IN",
        "status": "REFERENCED"
      },
      {
        "networkName": "nfvo-poc3-mgmt",
        "status": "REFERENCED"
      }
    ],
    "vnfs": [
      {
        "vnfId": "75127",
        "vnfName": "JuniperVNF55-1",
        "status": "ASSIGNED",
        "vnfDescriptor": "Juniper_vSRX_VNFD",
        "vnfServiceId": "75192",
        "vmId": "d63486ed-e025-41bf-8259-7294fb043ff5",
        "biID": "75408"
      }
    ],
    "endPoints": [
      {
        "name": "EndPoint55",
        "status": "REFERENCED"
      }
    ]
  }
```

Get Status Information of a Network Service

Retrieves the status information about a network service.

Method

GET

URL

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

where *networkServiceId* is the ID of the network service

Sample Response

```
{
  "nsID": "375005",
  "nsdName": "NPaaS_NSD",
  "nsName": "29_1.3_ns_Service",
  "status": "IN_SERVICE"
}
```

Get List of Network Service Descriptors

Retrieves a list of network service descriptors.

Method

GET

URL

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

Sample Response

```
{
 "NPaaS_NSD",
  "CustomerName_NPaaS_NSD"
}
```

Get Information about a Network Service Descriptor

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

```
{
    "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": "40.0",
                            "action": "scale"
                        }
                    ]
                }
            ]
        }
    ]
}
```

Get VNF Descriptors

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

```
[
 "Juniper_vSRX_VNFD",
  "Checkpoint_NG_FW_VNFD"
]
```

Get Flavors of a Network Service Descriptor

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

```
[
    {
        "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": "40.0",
                        "action": "scale"
                    }
                ]
            }
        ]
    }
]
```

Get Information about a VNF Descriptor

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

```
{
    "deploymentFlavors": [
        {
            "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
        }
    ],
    "connectionPoints": [
        {
            "name": "CP01",
            "isExternal": false,
            "order": -1
        },
        {
            "name": "CP02",
            "isExternal": false,
            "order": -1
        },
        {
            "name": "CP03",
            "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-v0.3",
            "imageUserName": "",
            "imagePasswd": ""
        }
    ]
}
```

Get Versions of a VNF Descriptor

Retrieves details about the versions of a 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

```
{
    "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 Flavors of a VNF Descriptor

Retrieves the list of 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

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

Get Details about VNFs in a Network Service

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 ID of the network service

Sample Response

```
{
  "nsID": "375005",
  "nsdName": "NPaaS_NSD",
  "nsName": "29_1.3_AMns_Service",
  "vnfs": [
    {
      "vnfId": "300003",
      "vnfName": "VNF1",
      "vnfStatus": "INSTALLED",
      "vnfDescriptor": "Juniper_vSRX_VNFD",
      "vnfServiceId": "375006",
      "vnfServiceName": "VNF_Juniper_vSRX_VNFD_Service",
      "vnfServiceStatus": "IN_SERVICE",
      "vnfServiceDescriptor": "Juniper_vSRX_ServiceDescriptor",
      "vnfVersion": "1.0",
      "vmId": "3479b080-6341-425c-b242-ecd14b1dcef8",
      "biID": "375006",
      "status" : "ASSIGNED"
      "deploymentFlavorInfo": {
        "name": "m1.medium",
        "vcpus": 2,
        "memory": "4 MB",
        "disk": "40 GB"
      },
        "connectionPoints": [
          {
            "id": "300003-1",
            "name": "CP01",
            "ipAddress": {
              "address": "192.0.2.132",
              "network": "nfvo-pkt-in-v2",
              "externalID": "8f2468de-c4b1-4656-b23f-ccd5c26b9d83"
            }
          },
          {
            "id": "300003-2",
            "name": "CP02",
            "ipAddress": {
              "address": "192.0.2.120",
              "network": "nfvo-pkt-out-v2",
              "externalID": "8ab6b415-b04a-458c-97bc-d4ef2eb550c3"
            }
          },
          {
            "id": "300003-3",
            "name": "CP03",
            "ipAddress": {
              "address": "192.0.2.8",
              "network": "nfvo-mgmt",
              "externalID": "9e32e48a-439c-4292-a308-9eafa0beeb78"
            }
          }
        ]
      }
    ]
}
```

Get Details about Networks in a Network Service

Retrieves the details about the networks in a network service.

Method

GET

URL

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

where *networkServiceId* is the ID of the network service

Sample Response

```
{
  "nsID": "375005",
  "nsdName": "NPaaS_NSD",
  "nsName": "ns_Service",
  "networks": [
    {
      "networkName": "nfvo-mgmt",
      "networkID": "nfvo-mgmt",
      "externalID": "109ae4cf-3cea-4729-a24f-957c4ed6d3c6",
      "status" : "REFERENCED" ,
      "subnets": [
        {
          "startIP": "192.0.2.0",
          "prefix": "24",
          "externalID": "fb791563-7c8b-454c-a1eb-87399e6837dc"
        }
      ]
    },
    {
      "networkName": "nfvo-pkt-in-v2",
      "networkID": "nfvo-pkt-in-v2",
      "externalID": "2277b6e2-eb2d-4cc2-b80c-6d6c38f35ab0",
      "status" : "REFERENCED" ,
      "subnets": [
        {
          "startIP": "192.0.2.128",
          "prefix": "25",
          "externalID": "d47bf43a-57bd-4b17-b559-505a426d7359"
        }
      ]
    },
    {
      "networkName": "nfvo-pkt-out-v2",
      "networkID": "nfvo-pkt-out-v2",
      "externalID": "3b45febc-4531-4751-ac55-9e43bd53897a",
      "status" : "REFERENCED" ,
      "subnets": [
        {
          "startIP": "192.0.2.0",
          "prefix": "25",
          "externalID": "c04bb488-73cc-4e93-bcab-156030a63a0c"
        }
      ]
    }
  ]
}
```

Get Details about Endpoints in a Network Service

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 ID of the network service

Sample Response

```
{
    "nsID": "75133",
    "nsdName": "NPaaS_NSD",
    "nsName": "NPaaS_Service",
    "biID": "75360",
    "endPoints": [
        {
            "name": "cnsmr_endpoint",
            "ipAddress": "207.123.34.2",
            "status": "PENDING_UNREFERENCE"
        }
    ]
}
```

Get Details about a VNF

Retrieves the details about a VNF.

Method

GET

URL

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

where *vnfId* is the ID of the VNF

Sample Response

```
{
  "vnfId": "300003",
  "vnfName": "VNF1",
  "vnfStatus": "INSTALLED",
  "vnfDescriptor": "Juniper_vSRX_VNFD",
  "vnfServiceId": "375006",
  "vnfServiceName": "vnf_Juniper_vSRX_VNFD_Service",
  "vnfServiceStatus": "IN_SERVICE",
  "vnfServiceDescriptor": "Juniper_vSRX_ServiceDescriptor",
  "biID": "375006",
  "deploymentFlavorInfo": {
    "name": "m1.medium",
    "vcpus": 2,
    "memory": "4 MB",
    "disk": "40 GB"
  },
  "connectionPoints": [
    {
      "id": "300003-1",
      "name": "CP01",
      "ipAddress": {
        "address": "192.0.2.132",
        "network": "nfvo-pkt-in-v2",
        "externalID": "8f2468de-c4b1-4656-b23f-ccd5c26b9d83"
      }
    },
    {
      "id": "300003-2",
      "name": "CP02",
      "ipAddress": {
        "address": "192.0.2.120",
        "network": "nfvo-pkt-out-v2",
        "externalID": "8ab6b415-b04a-458c-97bc-d4ef2eb550c3"
      }
    },
    {
      "id": "300003-3",
      "name": "CP03",
      "ipAddress": {
        "address": "192.0.2.8",
        "network": "nfvo-mgmt",
            "externalID": "9e32e48a-439c-4292-a308-9eafa0beeb78"
          }
        }
      ]
    }
```

Get Status Information of a VNF

Retrieves the status information of a VNF.

Method

GET

URL

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

where *vnfId* is the ID of the VNF

Sample Response

```
{
    "vnfId": "75085",
    "vnfName": "ChkptVNF_CP_B253_BALU_Scale",
    "status": "ASSIGNED",
    "vnfDescriptor": "Checkpoint_NG_FW_VNFD",
    "vnfServiceId": "75134",
    "vnfServiceStatus": "IN_SERVICE",
    "vmStatus": "ACTIVE",
    "vmId": "80cbd3bf-5bc2-45b1-9cc4-b4bbc9e120c2",
    "biID": "75273"
}
```