Oracle® Communications Network Service Orchestration Solution

Implementation Guide

Release 1.1

E65331-02

July 2016

Oracle Communications Network Service Orchestration Solution Implementation Guide, Release 1.1

E65331-02

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 their deployment 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 and model 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.
- **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 centres to ensure that each network service is allocated the required resources to meet the needs of the VNFs.
- **Customization and Extension**. You can customize and extend the solution to support integration with third-party VNF Managers, Virtualized Infrastructure Managers (VIMs), SDN Controllers, and monitoring engines. The solution 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 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 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 NFV Entities and OCIM Entities

NFV Entity	OCIM Entity	Description
Network Service	Service	Represents a network service formed by VNFs on the NFV infrastructure.
Virtual Network Function (VNF)	Logical Device	Represents a virtual network function. A VNF is a network function capable of running on an NFV infrastructure and being orchestrated by a NFV Orchestrator and VNF Manager. A VNF may implement a single network entity with interfaces and behavior defined by standardization organizations like 3GPP and IETF, while another VNF may implement groups of network entities.
NFV Infrastructure	Custom Object with the following characteristics: ■ Host ■ Port ■ Username ■ Password ■ Domain Name ■ Tenant Name ■ VIM Type Represents a tenant.	Network Functions Virtualization Infrastructure (NFVI) is the totality of all hardware and software components that build the environment where VNFs are deployed.
Network Service Descriptor (NSD)	Service	Describes the deployment requirements, operational behavior, and policies required by network services that are created based on this descriptor.
VNF Descriptor (VNFD)	■ Logical Device ■ Service	Describes the deployment requirements, operational behavior, and policies required by VNFs that are created based on this descriptor.

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

NFV Entity	OCIM Entity	Description
Virtualized Infrastructure Manager (VIM)	Custom Object with the following characteristics: ■ Host ■ Port ■ Username ■ Password ■ Domain Name ■ Tenant Name ■ VIM Type	Represents a Virtualized Infrastructure Manager that controls and manages the compute, storage, and network resources.
SDN Controller	Custom Object	Represents a Software Defined Network (SDN) controller that manages the flow control to enable intelligent networking.
Virtual Data Center (VDC)	Custom Object with the following characteristics: ■ Disk Total ■ Memory Total ■ VCPU Total	Represents a virtual machine in a data center.
Availability Zone	Custom Object with the following characteristics: ■ Disk Total ■ Memory Total ■ VCPU Total ■ Disk Used ■ Memory Used ■ VCPU Used	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.
Host	Custom Object with the following characteristics: ■ Disk Total ■ Memory Total ■ VCPU Total ■ Disk Used ■ Memory Used ■ VCPU Used	Represents the compute host.
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 VIM on which the VNF virtual machine is deployed.
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 termination endpoint for the network service.
Connection Point	Device Interface	Represents a port on the VNF.

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.

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 the required software. See "Configuring UIM for the Network Service Orchestration Solution".
- Integrate the VIM. See "Integrating the Network Service Orchestration Solution Components".
- Integrate the SDN Controller. See "Integrating the SDN Controller With the Solution".
- 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 7.3.2.

  See *UIM 7.3.2 Installation Guide* for installation instructions.
- Oracle Communications Design Studio 7.3.1.

See *Design Studio 7.3.1 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 7.3.2 Installation Guide* for UIM installation instructions.

   Note: If you are upgrading to UIM 7.3.2 from UIM 7.3.0 or UIM 7.3.1, follow the steps for upgrading from UIM 7.3.*x* to UIM 7.3.2 in the *UIM 7.3.2 Installation Guide*.

2. In UIM, deploy the base UIM cartridges in the following order:
   - 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.

3. 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/

4. 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
   ```

5. Restart the server on which UIM is installed.

6. In WebLogic server, 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.

7. Go to **deploy**/**individualJarsForSuperJar** and deploy the following Network Service Orchestration solution cartridges into UIM in the order they are listed:
   - OracleComms_NSO_Common
   - OracleComms_NSO_NFVIAdapter
   - OracleComms_NSO_BaseCartridge

8. (Optional) If you want to use the sample cartridges that are provided with the solution, 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.

9. In the WebLogic server on which UIM is installed, deploy the **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_2/** 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**.

After you install the required software, integrate the solution components. See "Integrating the Network Service Orchestration Solution Components" for more information.

Integrating the Network Service Orchestration Solution Components

This section describes the steps that you follow to integrate the Network Service Orchestration solution components.

Integrating the solution components involves the following tasks:

- Integrating the VIM with the Solution
- Integrating the SDN Controller With the Solution

Integrating the VIM with the Solution

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.

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:

   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.

Note: Register your VIM with the solution only once. Do not register a VIM that you have already registered. In UIM, if there are multiple VIM custom objects that refer to the same VIM, resource orchestration may return errors.

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. When you discover a VIM, the details of the virtual resources are populated into UIM. In UIM, the VIM is represented as a custom object.

To discover VIM resources into UIM:

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

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

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
- *vimName* is the name of the VIM that you registered with the solution and whose resources you want to discover

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.

Integrating the SDN Controller With the Solution

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 integrate 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.

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)
  - Framework to integrate with external VNF Managers

- For Virtual Infrastructure Management
  - Integration to OpenStack Kilo and Oracle OpenStack for Oracle Linux Release 2
  - Sample integration to VMware vCloud Director
  - Framework to integrate to 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.

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

  See "About the Technical Actions File" for more information about the descriptor 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 a number of NFV descriptors in the form of Design Studio specifications. These specifications are used by the Network Service Orchestration solution to create NFV events for managing the life cycles of VNFs and network services.

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

About the Network Service Descriptor

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 reference existing networks and specifying network types.
- For each network, specify the VNFs the network service should use.
- Specify the flow path for the network traffic.
- 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.
- Specify when you want to heal a VNF and scale the network service.

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

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

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

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

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

To create a descriptor file:

1. In Design Studio, import all the required 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:

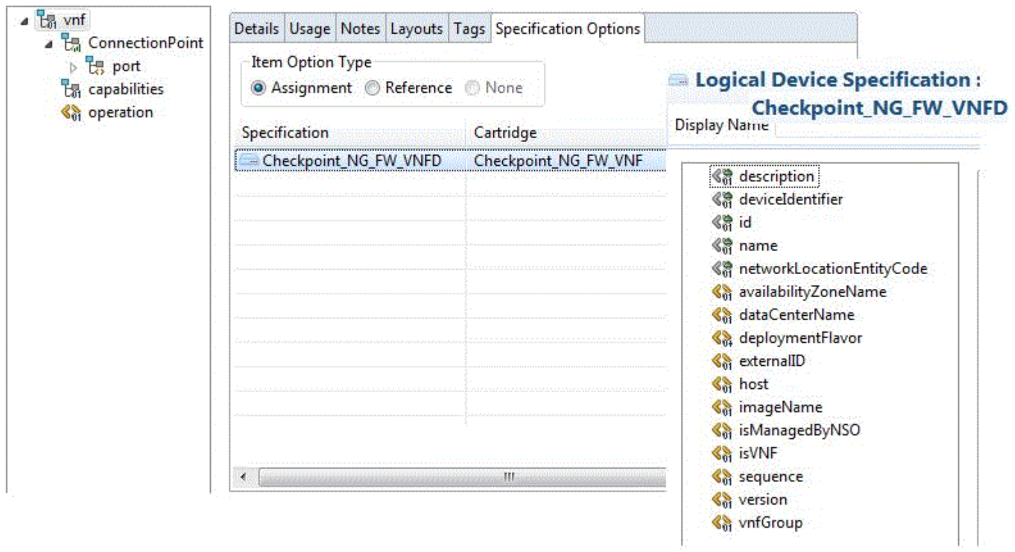
   model\content\product_home\config
4. Right-click on the **config** folder and create an XML file with the name *ServiceSpecificationName*.**xml**.
5. Copy the sample content from the sample cartridge to the XML file and modify it according to your solution requirements.

About the Technical Actions File

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

Figure 3–1 shows a sample VNF service configuration model in Design Studio.

Figure 3–1 VNF Service Configuration Model



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>
```

About Technical Actions

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.

Creating a Technical Action File

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

To create a technical action 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**.
4. Copy the sample content from the sample cartridge to the XML file and modify it according to your solution requirements.

About the VNF Configuration Files

After a VNF is deployed, you can configure the VNF based on the configuration requirements of the VNF.

Note: Configuration is not required for all VNFs. Configuration is required only for those VNFs that need additional configurations other than the configuration in the VNF's software image.

To configure the 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:

- NPaaS_NetworkService. This sample cartridge contains functionality to implement a Network Protection 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.

Figure 3–2 shows how the Network Protection service is modeled in the sample Network Service cartridge.

Figure 3–2 Network Service Model

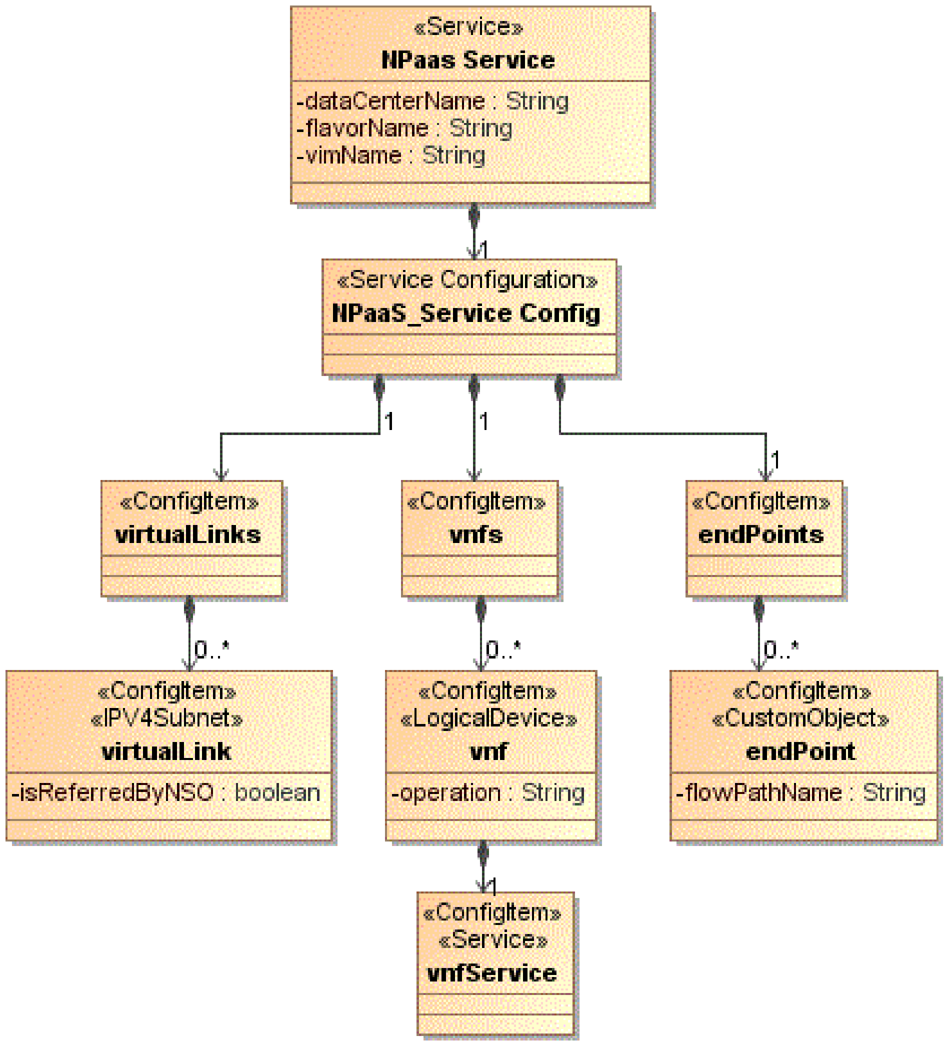
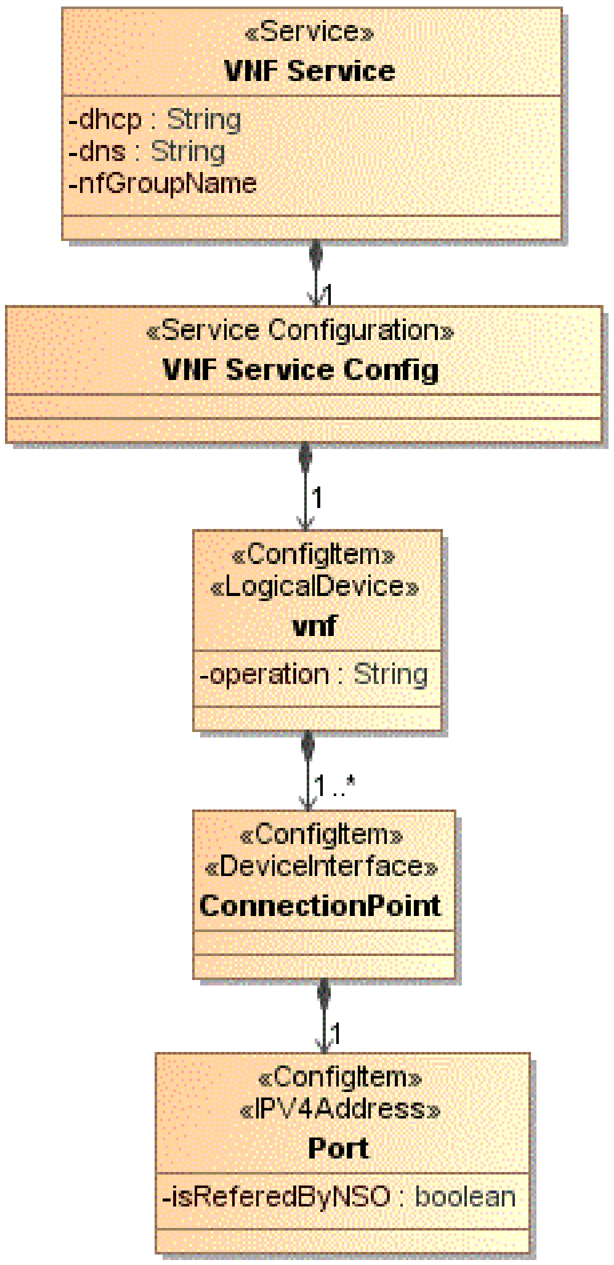


Figure 3–3 shows how a VNF service is modeled in the sample VNF cartridge.

Figure 3–3 VNF Service Model



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" for more information about the VNF descriptor file.

The solution includes sample technical action files for the VNFs. The technical action 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 7.3.2 and the Network Service Orchestration Solution 1.1 cartridges
- OpenStack VIM, with Open vSwitch capability
- OpenDaylight SDN Controller
- Software images for the firewall VNFs

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 or reference an existing management network that can be shared by all the components of the solution.

   The management network requires, at a minimum:

   - One IP address for each:
     – Virtual machine on which UIM is installed
     – Virtual machine on which Open vSwitch is installed
     – Virtual 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 3–1 shows a sample network and subnet configuration for the customer-side network.

Table 3–1 Sample Network and Subnet Configuration for Customer-side Network

CIDR	IP Allocation Pool	Gateway IP	DHCP Enabled	Additional Routes	DNS Name Server
192.0.2.0/24	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 3–2 shows a sample network and subnet configuration for the Internet-side network.

Table 3–2 Sample Network and Subnet Configuration for Internet-side Network

CIDR	IP Allocation Pool	Gateway IP	DHCP Enabled	Additional Routes	DNS Name Server
192.0.2.0/24	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 3–3 shows a sample network and subnet configuration for the packet-in network.

Table 3–3 Sample Network and Subnet Configuration for Packet-in Network

CIDR	IP Allocation Pool	Gateway IP	DHCP Enabled	Additional Routes	DNS Name Server
192.0.2.128/24	Start 192.0.2.129 End 192.0.2.140	-	Yes	None	None

   Table 3–4 shows a sample network and subnet configuration for the packet-out network.

Table 3–4 Sample Network and Subnet Configuration for Packet-out Network

CIDR	IP Allocation Pool	Gateway IP	DHCP Enabled	Additional Routes	DNS Name Server
192.0.2.0/24	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*

     8 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. In Design Studio, open the **nso.properties** file and do the following:

   - Update the corresponding Open vSwitch port numbers of the network interfaces:

     – **nso.ovs.pktInToOVSPort** = *ovsPort*

       where *ovsPort* is the Open vSwitch port number of the network interface that you attached to the steering bridge.

     – **nso.ovs.pktInToOVSPort** = *4*

     – **nso.ovs.pktOutToOVSPort** = *1*

     – **nso.ovs.custNetToOVSPort** = *5*

     – **nso.ovs.internetToOVSPort** = 2

   - Update the following server details:

     – **NSO_HOST**: *IPv4address*

     – **NSO_USERNAME**: *username*

       where *IPV4address* is the IP address and *username* is the username of the machine on which UIM is installed.

Note: By default, the **nso.properties** file displays the username and password of the Network Service Orchestration solution user in plain text. You can 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, run the **EncryptText** ruleset, and browse and specify the text file that contains the password in plain text.

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

- Specify the following networks:
  - **NPaaS_NSD.ManagementNetwork**=*management_network*

    where *management_network* is the name of the management network.
  - **NPaaS_NSD.Data_IN**=*packet_in_network*

    where *packet_in_network* is the name of the packet-in network.
  - **NPaaS_NSD.Data_OUT**=*packet_out_network*

    where *packet_out_network* is the name of the packet-out network.
- Specify the name of the network service descriptor:

  ```
  ServiceDescriptors = NPaaS_NSD
  ```

  where *NPaaS_NSD* is the name of the Network Service descriptor.

11. Open the **NPaaS_NSD.xml** file and set **isReferenced** to **true**.
12. Open the **nfvi.properties** file and update the odlManager path:

    ```
    sdnController.ODL=com.oracle.communications.inventory.nso.nfvi.sdn.ODLManager
    ```

13. In OpenDaylight, retrieve the OpenFlow ID by calling the following OpenDaylight REST API:

    http://*odl_IPaddress*:*port*/restconf/operational/opendaylight-inventory:nodes/

    where *odl_IPaddress* is the IP address and *port* is the port number of the OpenDaylight virtual machine.
14. In the **nso.properties** file, update the OpenFlow ID:

    ```
    nso.ovs.bridge_id=openflow:OpenFlow_ID
    ```

15. Redeploy the cartridges in Design Studio. See "Configuring UIM for the Network Service Orchestration Solution" for information about deploying the cartridges in the specified order.
16. Register the VIM by calling the corresponding RESTful API. See "Integrating the VIM with the Solution" for instructions.
17. 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 Action 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 Action 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

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.

To instantiate a network service:

1. In a REST API client, run the following URL 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 by looking in the network service and VNF descriptor files:

   - nsName
   - nsDescriptorName
   - serviceDeploymentFlavor
   - vnfName
   - deploymentFlavorName

- vnfDescriptorName
- version
- name
- parameters

For details about these parameters, navigate to *UIM_Home*/**config**/ and look up the sample network service and VNF descriptor XML files.

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

To upgrade the software version of a VNF in a network service:

1. In a REST API client, run the following URL using the PUT method:

   PUT http://*host*:*port*//ocnso/1.1/ns/*networkServiceId*/upgrade

   where *networkServiceId* is the ID of the network service that the VNF is part of.

   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 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. 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.

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

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 REST API client, run the following URL 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.

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.

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 REST API client, run the following URL 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 REST API client, run the following URL 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 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 REST API client, run the following URL 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 Virtual Network Function 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. From the Oracle Software Delivery Cloud, download the UIM SDK into the *UIM_SDK_Home* local directory.
2. Extract the downloaded **UIM_SDK.zip** file into the *UIM_SDK_Home* local directory to get the **lib** folder.

3. Go to *build_folder*/**designStudio** and extract the **nso_lib** folder.
4. Create a local directory named **OTHER_LIB**.
5. Copy the following WebLogic libraries from your WebLogic installation into the **OTHER_LIB** local directory:
   - *WL_Home*/**oracle_common/modules/groovy-all-2.0.5.jar**
   - *WL_Home*/**oracle_common/modules/jersey-client-1.18.jar**
   - *WL_Home*/**oracle_common/modules/jettison-1.1.jar**
   - *WL_Home*/**wlserver/modules/features/weblogic.server.merged.jar**
6. Download the **gson-2.2.4.jar** file from the following Website and copy it into the **OTHER_LIB** local directory:

   http://repo1.maven.org/maven2/com/google/code/gson/gson/2.2.4/
7. Copy other UIM-specific JAR files to the **OTHER_LIB** directory. See *UIM Installation Guide 7.3.2* for information about UIM-specific JAR files.
8. In Design Studio, open a new workspace and import the following base cartridges:
   - ora_uim_baseextpts
   - ora_uim_basemeasurements
   - ora_uim_basespecifications
   - ora_uim_basetechnologies
   - ora_uim_common
   - ora_uim_mds
   - ora_uim_model
9. Import the following Network Service Orchestration Solution cartridges:
   - OracleComms_NSO_BaseCartridge
   - OracleComms_NSO_Common
   - OracleComms_NSO_NFVIAdapter
   - 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 Library variables:
    - UIM_LIB. Specify the path as UIM_SDK_Home/**lib**.
    - OTHER_LIB. Specify the directory that you created.
    - NSO_LIB. Specify the path as *build_folder*/**designStudio/nso_lib**.
11. Build the inventory project.

Designing Cartridges for Custom VNFs and Network Services

To design cartridges for custom VNFs and network services:

1. In Design Studio, create cartridge 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 cartridge project, create specifications, metadata, and technical action files.
3. For each service specification, create a technical action xml file. See "About the Technical Actions File" for more information.
4. Write the design-and-assign logic for the service configuration.
5. Write the issue logic for the service configuration.
6. Develop the adapter for the monitoring engine. See "Implementing a Custom Monitoring Engine" for more information.
7. Develop the adapter for the VNF life-cycle manager. See "Implementing a Custom VNF Manager" 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.

The solution supports the following extensions with extension points:

- The Design and Assign extension point for VNF and Network Service service configurations.
- The Issue Configuration extension point for VNF and Network Service service configurations.
- Data center lookup based on the dynamic property provided in the network service request.

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.
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 Manager".

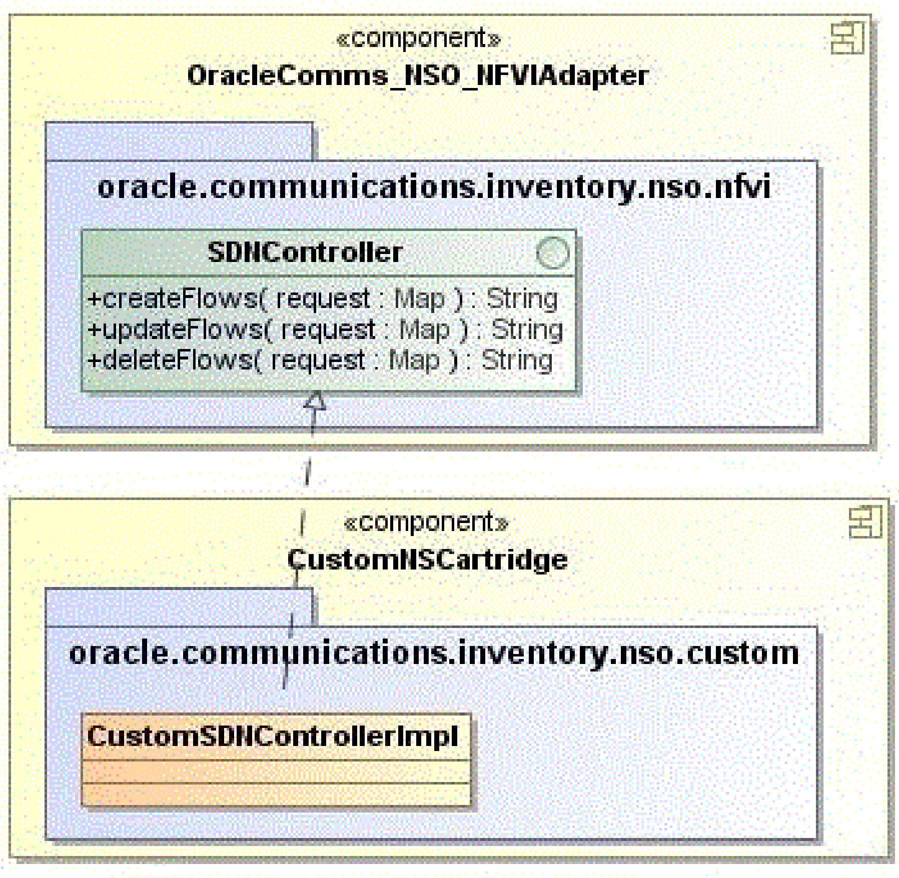
- Implementation of a custom VNF connection manager. See "Implementing a Custom VNF Connection Manager".
- Implementation of a custom VNF configuration manager. See "Implementing a Custom VNF Configuration Manager".

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 catalog 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
```

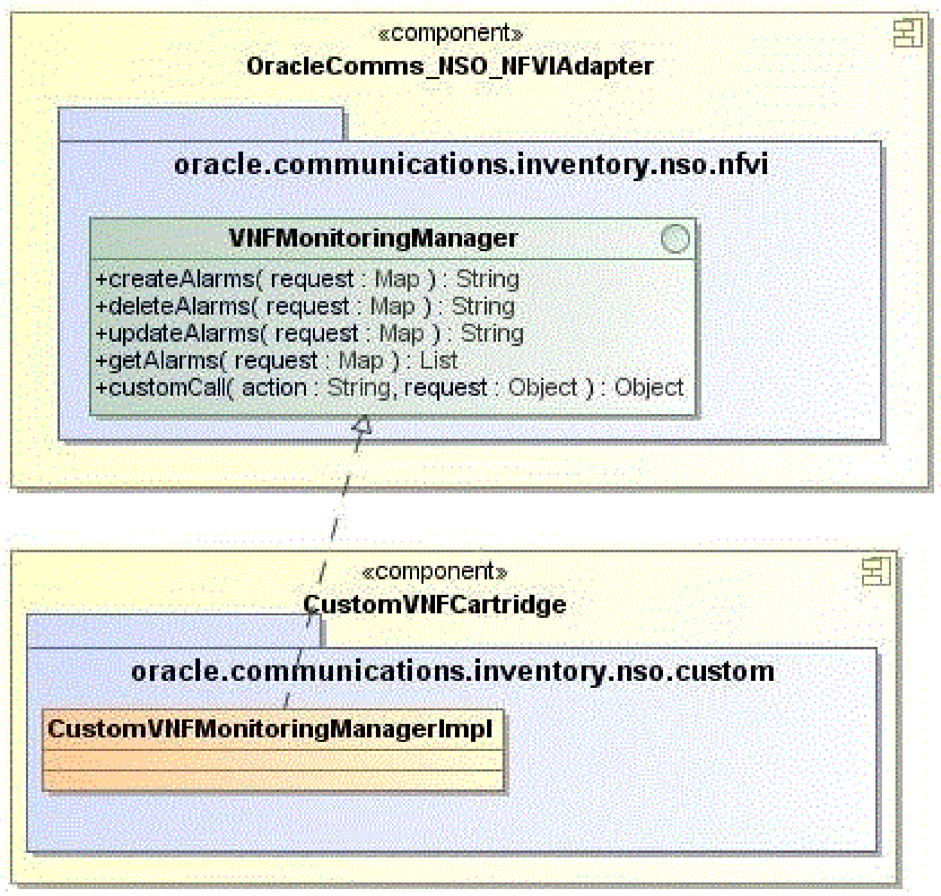
4. Go to *UIM_Home*/**config** and open the **nfvi.properties** file in a text editor.
5. Update the **sdnController**.*sdnType* key with the name of custom SDN controller.

Implementing a Custom Monitoring Engine

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

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 VNFMonitoringManager.
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
```

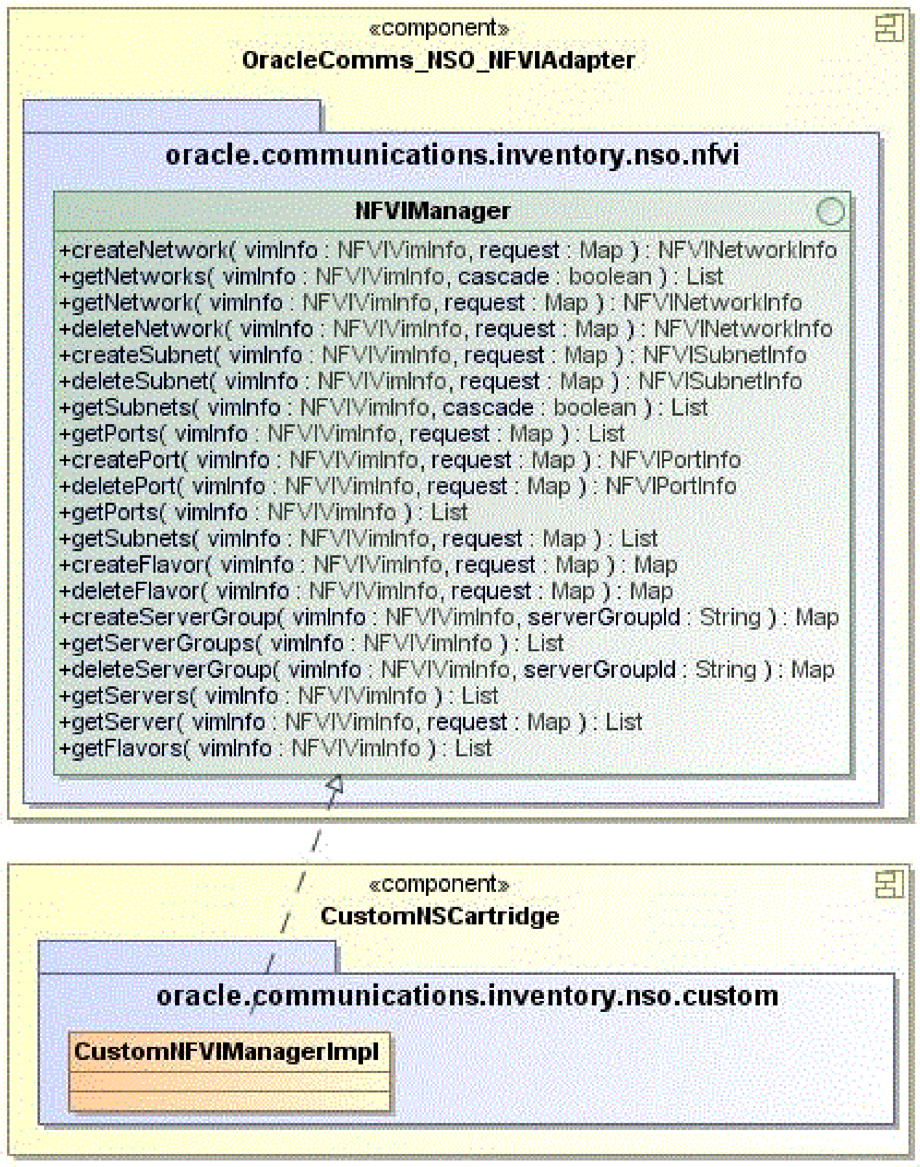
4. Go to *UIM_Home*/**config** and open the **nfvi.properties** file in a text editor.
5. Update the **vnfMonitor.***vnfdName* key with the name of the custom monitoring engine.

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
«component»
CustomNSCartridge
oracle.communications.inventory.nso.custom
CustomNFVIManagerImpl

```
+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
```

To implement a custom VIM:

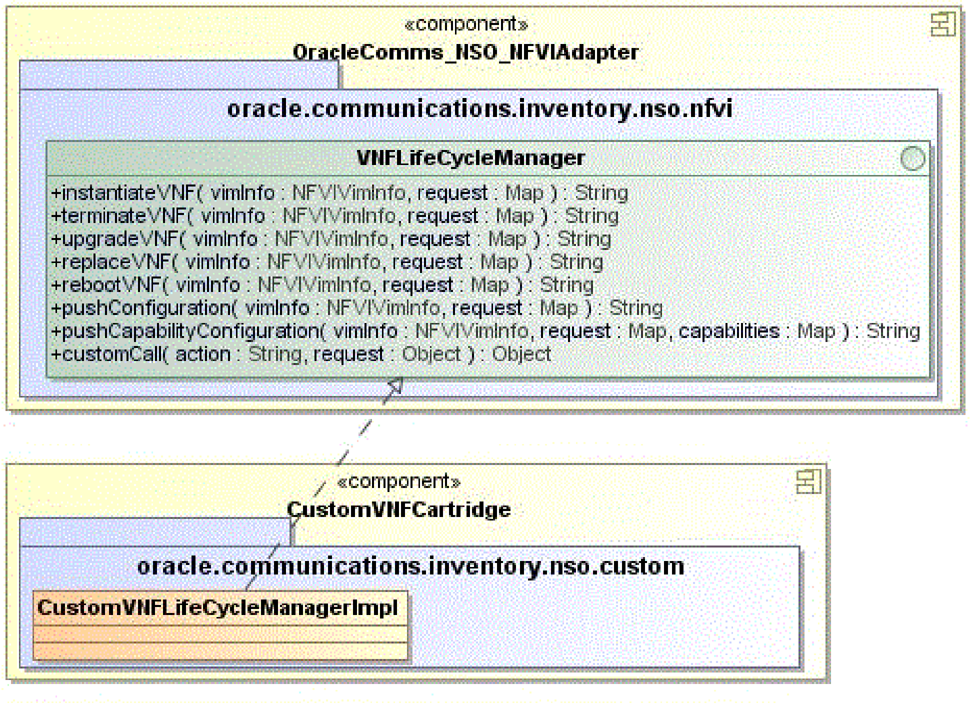
1. In the custom Network Service descriptor catalog cartridge, create a Java implementation class for the NFVIManager interface.
2. Configure the NFVIManager class to implement the **oracle.communications.inventory.nso.nfvi.NFVIManager** interface, which is provided in the OracleComms_NSO_NFVIAdapter cartridge.
3. Override the methods in the custom NFVI manager Java implementation class.
4. Go to *UIM_Home*/**config** and open the **nfvi**.**properties** file in a text editor.
5. Update the **nfviMgr**.*nfviType* key with the name of the custom VIM.

Implementing a Custom VNF Manager

The Network Service Orchestration solution uses UIM to manage the lifecycle of the VNFs. The solution supports integration with third-party VNF managers by using extensions.

Figure 5–4 shows a model diagram that depicts how you can write an extension for a custom VNF 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

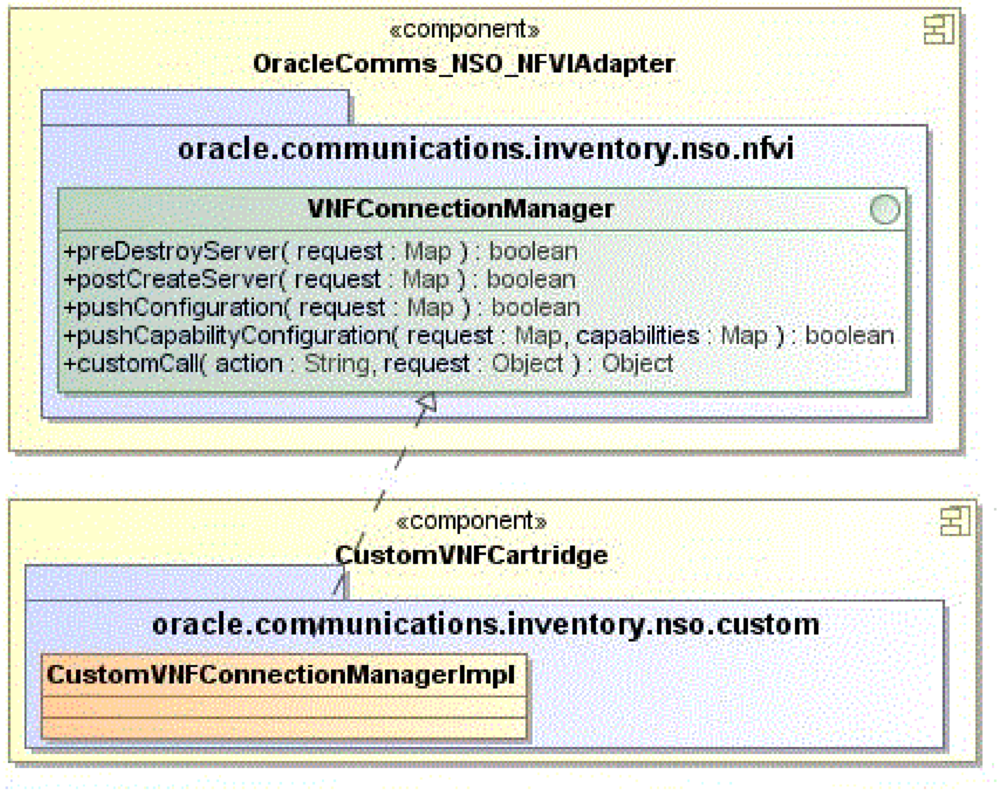
To implement a custom VNF manager:

1. In the custom VNF descriptor cartridge, create a Java implementation class for the VNF manager.
2. Configure the custom VNF 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 manager Java implementation class.
4. Go to *UIM_Home*/**config** and open the **nfvi.properties** file in a text editor.
5. Update the **vnflcMgr**.*vnfdName* key with the name of the custom VNF manager.

Implementing a Custom VNF Connection Manager

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

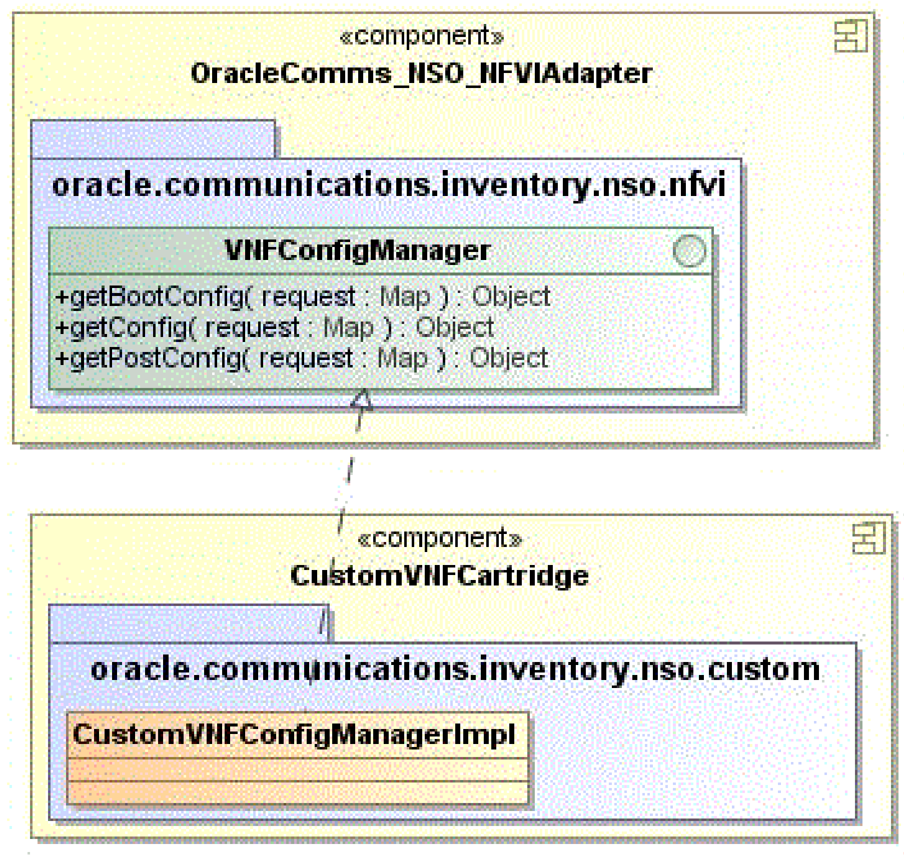
To implement a custom VNF connection manager:

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. Go to *UIM_Home*/**config** and open the **nfvi.properties** file in a text editor.
5. Update the **vnfConnectionMgr**.*vnfdName* key with the name of the custom VNF connection manager.

Implementing a Custom VNF Configuration Manager

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. Go to *UIM_Home*/**config** and open the **nfvi.properties** file in a text editor.
5. Update the **vnfConfigMgr**.*vnfdName* key with the name of the custom VNF configuration manager.

Localizing the Network Service Orchestration Solution

You can localize Oracle Communications Unified Inventory Management (UIM) user interface and the UIM Help. Localizing UIM involves modifying a specific set of files that UIM uses to display text in the UI and in the Help.

To localize the response messages 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 language code. For example, *fr-FR* indicates the locale ID for 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, specify *fr-FR* for French.

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	Individual JAR files that comprise the super JAR file. See "Network Service Orchestration Individual JAR Files" for more information.
deploy/ superJarToDeploy	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	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 following application in the **deploy\applications\** directory:

- **OracleComms_NSO_WebServices.war**

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
- 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

---

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/discover/*vim_name*	Discovers the resources of the registered VIM into 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.
Update VNF software version	POST	/ocnso/1.1/ns/*networkServiceId*/upgrade	Updates the software image version of a VNF.
Heal a VNF	POST	/ocnso/1.1/vnf/*vmId*/heal /ocnso/1.1/vnf/heal	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.
Delete a VNF from a network service	DELETE	/ocnso/1.1/ns/*networkServiceId*/vnfs	Deletes a VNF from a network service.
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.
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.

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

Task	Method	Resource	Description
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 (retrieve), POST (create), PUT (modify), and DELETE operations of the Network Service Orchestration Solution RESTful APIs.

Table 7–2 HTTP Response Status Codes for the Network Service Orchestration Solution RESTful 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. There is no facility for re-sending a status code from an asynchronous operation such as this.
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
```
{
"name":"vimDataCenter",
"host":"11.111.111.1",
"port":"12345",
"userName":"nso",
"pswd":"***",
"projectName":"test",
"domainName":"default"
"vimType":"default"
}
```

Table 7–3 describes the parameters in the request.

Table 7–3 Request Parameters

Parameter	Value	Required	Description
name	String	Yes	Name of the VIM.
vimType	String	No	Type of the VIM. The default is OpenStack.
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	Yes	Name of the domain.

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/discover/*vimDataCenter*

where *vimDataCenter* is the name of the VIM whose resources you want to discover

Sample Request

This API does not require any request parameters.

Sample Response

VIM resources are discovered 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":"NSO_NPassService_1",
    "nsDescriptorName":"NPaaS_NSD",
    "serviceDeploymentFlavorName":"Checkpoint",

    "vnfs":[
                {
                    "vnfName":"NSO_CheckPointVNF_1",
                    "deploymentFlavorName":"checkpoint",
                    "vnfDescriptorName":"Checkpoint_NG_FW_VNFD",
                    "version":"1.0"
                }

            ],
        "endPoints":[
          {
                "name":"NSO_SGPcnsmr2",
                "flowPath":"Red",
                "parameters":
[
                {
"name": "ipAddress",
"value": "207.123.34.2"
},

{
"name": "vlanId",
"value": "101"
},

{
"name": "serviceLocation",
"value": "CityPQ03"
}
]
        }
    ]
}
```

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.
endPoints:[forwardingGraphDescriptorName	String	Yes	Name identifier that is defined in the Network Service meta-data 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	Actual value of the customer-side end point termination points.

Sample Response

```
{
  "networkServiceName": "24_11.6_AMns_Service",
  "networkServiceId": "975022",
  "networkServiceStatus": "PENDING",
  "message": "Network Service Instantiation is in progress",
  "status": 202,
  "vnfs": [
    {
      "vnfId": "1050008",
      "vnfName": "24_11.6_AMvnf",
      "vnfStatus": "INSTALLED",
      "vnfDescriptor": "Juniper_vSRX_VNFD",
      "vnfServiceId": "975021",
      "vnfServiceName": "24_11.6_AMvnf_Juniper_vSRX_VNFD_Service",
      "vnfServiceStatus": "PENDING_ASSIGN",
      "vnfServiceDescriptor": "Juniper_vSRX_ServiceDescriptor",
      "message": null,
      "status": 0,
      "capabilities": []
    }
  ]
}
```

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

```
{
  "networkServiceId": "975022",
  "message": "Terminate request is under process.",
  "status": 202
}
```

Upgrade the Software Version of a VNF

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

Method

POST

URL

http://*nso_host_name*:*port*/ocnso/1.1/ns/*networkService_Id*/upgrade

Sample Request

```
 {
   "vnfName":"01.9_vSRXA_002",
   "upgradeToVersion":"1.6" // Upgrades to the version defined in VNF descriptor
}
```

Table 7–5 describes the parameters in the request.

Table 7–5 Request Parameters

Parameter Name	Value	Required	Description
vnfName	String	Yes	Name of the VNF whose image you want to upgrade.
upgradeToVersion	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

```
{
  "networkServiceId": "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_name*:*port*/ocnso/1.1/vnf/*vnfExternalId*/heal

where *vnfExternalId* is the VNF external ID

Sample Request

```
{
   "action":"reboot"
}
```

Table 7–6 describes the parameters in the request.

Table 7–6 Request Parameters

Parameter Name	Value	Required	Description
action	String	Yes	Specify whether you want to reboot or replace the VNF. Values are: ■ reboot ■ replace

Sample Response

reboot

```
VNF is 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_name*:*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 7–7 describes the parameters in the request.

Table 7–7 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

```
{
  "status": 202,
  "message": "[INV-992903] Adding VNF...",
  "networkServiceId": "30"
}
```

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_name*:*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

```
 {
  "status": 202,
  "message": "[INV-992904] Deleting VNF...",
  "networkServiceId": "33"
}
```

Get Network Service Information

Retrieves the details of a network service.

Method

GET

URL

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

where *networkServiceId* is the network service ID

Sample Response

```
{
  "nsID": "975022",
  "nsName": "24_11.6_AMns_Service",
  "status": "IN_SERVICE",
  "aIPAddress": null,
  "zIPAddress": null,
  "aIPDomain": null,
  "zIPDomain": null,
  "vimName": "datacntr_test_2",
  "biID": "975023",
  "networks": [
    {
      "networkName": "nfvo-poc3-pkt-in",
      "networkID": "nfvo-poc3-pkt-in",
      "externalID": "cf8d4e3d-2775-4b8a-b38a-25410e8bbba4",
      "subnets": [
        {
          "startIP": "192.0.2.0",
          "prefix": "24",
          "externalID": "576a7110-d628-455c-932b-9586f688c22c"
        }
      ]
    },
    {
      "networkName": "nfvo-poc3-mgmt",
      "networkID": "nfvo-poc3-mgmt",
      "externalID": "109ae4cf-3cea-4729-a24f-957c4ed6d3c6",
      "subnets": [
        {
          "startIP": "192.0.2.0",
          "prefix": "24",
          "externalID": "fb791563-7c8b-454c-a1eb-87399e6837dc"
        }
      ]
    },
    {
      "networkName": "nfvo-poc3-pkt-out",
      "networkID": "nfvo-poc3-pkt-out",
      "externalID": "14403e50-4f3d-484c-9743-82cdde768ece",
      "subnets": [
        {
          "startIP": "192.0.2.0",
          "prefix": "24",
          "externalID": "a0790554-6a67-42c4-a2f4-c1b1ddfa5800"
        }
      ]
    }
  ],
  "vnfs": [
    {
      "vnfId": "1050008",
      "vnfName": "24_11.6_AMvnf",
      "vnfStatus": "INSTALLED",
      "vnfDescriptor": "Juniper_vSRX_VNFD",
      "vnfServiceId": "975021",
      "vnfServiceName": "24_11.6_AMvnf_Juniper_vSRX_VNFD_Service",
      "vnfServiceStatus": "ASSIGNED",
      "vnfServiceDescriptor": "Juniper_vSRX_ServiceDescriptor",
      "biID": "975024",
      "deploymentFlavorInfo": {
        "name": "m1.medium",
        "vcpus": 2,
        "memory": "4 MB",
        "disk": "40 GB"
      },
      "connectionPoints": [
        {
          "id": "1050008-1",
          "name": "CP01",
          "ipAddress": {
            "address": "111.111.1.11",
            "network": "nfvo-poc3-pkt-in",
            "externalID": "3ffdad5d-83cb-42cb-a389-179bae00b255"
          }
        },
        {
          "id": "1050008-2",
          "name": "CP02",
          "ipAddress": {
            "address": "111.111.1.11",
            "network": "nfvo-poc3-pkt-out",
            "externalID": "07249a45-4c18-4f01-9c6b-8c61bc5ae4e2"
          }
        },
        {
          "id": "1050008-3",
          "name": "CP03",
          "ipAddress": {
            "address": "111.111.1.11",
            "network": "nfvo-poc3-mgmt",
            "externalID": "f6e17f07-d956-4c97-b036-60777df8a429"
          }
        }
      ],
      "capabilityServices": null
    }
  ],
  "consumers": [
    {
      "name": "24_11.6_AMcnsmr",
      "ipAddress": "111.111.11.1",
      "ipDomain": null,
      "flowPathName": "",
      "flowPath": "24_11.6_AMvnf",
      "status": "REFERENCED"
    }
  ]
}
```

Get Network Service Descriptors

Retrieves a list of network service descriptors.

Method

GET

URL

http://*nso_host_name*:*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_name*:*port*/ocnso/1.1/nsd/*nsdName*

where *nsdName* is the name of the Network Service descriptor

Sample Response

```
{
  "referencedVnfds": [
    "Juniper_vSRX_VNFD",
    "Checkpoint_NG_FW_VNFD"
  ],
  "serviceFlavors": [
    {
      "name": "S1",
      "constituentVNFDs": [
        {
          "name": "Juniper_vSRX_VNFD",
          "maxInstances": "5",
          "maxConsumersServed": "2"
        },
        {
          "name": "Checkpoint_NG_FW_VNFD",
          "maxInstances": "5",
          "maxConsumersServed": null
        }
      ]
    },
    {
      "name": "S2",
      "constituentVNFDs": [
        {
          "name": "Juniper_vSRX_VNFD",
          "maxInstances": "5",
          "maxConsumersServed": "2"
        }
      ]
    }
  ]
}
```

Get VNF Descriptors

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

Method

GET

URL

http://*nso_host_name*:*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_name*:*port*/ocnso/1.1/nsd/*nsdName*/flavors

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

Sample Response

```
 {
    "name": "S1",
    "constituentVNFDs": [
      {
        "name": "Juniper_vSRX_VNFD",
        "maxInstances": "5",
        "maxConsumersServed": "2"
      },
      {
        "name": "Checkpoint_NG_FW_VNFD",
        "maxInstances": "5",
        "maxConsumersServed": null
      }
    ]
  },
  {
    "name": "S2",
    "constituentVNFDs": [
      {
        "name": "Juniper_vSRX_VNFD",
        "maxInstances": "5",
        "maxConsumersServed": "2"
      }
    ]
  }
```

Get Information about a VNF Descriptor

Retrieves details about a specified VNF descriptor.

Method

GET

URL

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

where *vnfdName* is the name of the VNF descriptor

Sample Response

```
 {
  "deploymentFlavours": [
    {
      "vcpus": 2,
      "memory": 4,
      "disk": 20,
      "name": "vsrx.medium"
    },
    {
      "vcpus": 2,
      "memory": 4,
      "disk": 40,
      "name": "m1.medium"
    }
  ],
  "connectionPoints": [
    {
      "name": "CP01",
      "isExternal": false
    },
    {
        "name": "CP02",
        "isExternal": false
      },
      {
        "name": "CP03",
        "isExternal": false
      }
    ],
    "versions": [
      {
        "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 Versions of a VNF Descriptor

Retrieves details about the versions of a specified VNF descriptor.

Method

GET

URL

http://*nso_host_name*:*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_name*:*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 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_name*:*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 Details about a Network Service

Retrieves the details about a network service.

Method

GET

URL

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

where *networkServiceId* is the ID of the network service

Sample Response

```
{
  "nsID": "375005",
  "nsdName": "NPaaS_NSD",
  "nsName": "29_1.3_AMns_Service",
  "status": "IN_SERVICE",
  "vimName": "OpenStack",
  "networks": [
    {
      "networkName": "nfvo-poc3-mgmt"
    },
    {
      "networkName": "nfvo-demo-pkt-in-v2"
    },
    {
      "networkName": "nfvo-demo-pkt-out-v2"
    }
  ],
  "vnfs": [
    {
      "vnfServiceId": "375006",
      "vnfServiceName": "29_1.3_AMvnf_Juniper_vSRX_VNFD_Service",
      "vnfServiceDescriptor": "Juniper_vSRX_ServiceDescriptor",
      "vmId": "3479b080-6341-425c-b242-ecd14b1dcef8"
    }
  ],
  "endPoints": [
    {
      "name": "29_1.3_AMcnsmr"
    }
  ]
}
```

Get Details about VNFs in a Network Service

Retrieves the details about the VNFs in a network service.

Method

GET

URL

http://*nso_host_name*:*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": "29_1.3_AMvnf",
      "vnfStatus": "INSTALLED",
      "vnfDescriptor": "Juniper_vSRX_VNFD",
      "vnfServiceId": "375006",
      "vnfServiceName": "29_1.3_AMvnf_Juniper_vSRX_VNFD_Service",
      "vnfServiceStatus": "IN_SERVICE",
      "vnfServiceDescriptor": "Juniper_vSRX_ServiceDescriptor",
      "vmId": "3479b080-6341-425c-b242-ecd14b1dcef8",
      "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-demo-pkt-in-v2",
            "externalID": "8f2468de-c4b1-4656-b23f-ccd5c26b9d83"
          }
        },
        {
          "id": "300003-2",
          "name": "CP02",
          "ipAddress": {
            "address": "192.0.2.120",
            "network": "nfvo-demo-pkt-out-v2",
            "externalID": "8ab6b415-b04a-458c-97bc-d4ef2eb550c3"
          }
        },
        {
          "id": "300003-3",
          "name": "CP03",
          "ipAddress": {
            "address": "192.0.2.8",
            "network": "nfvo-poc3-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_name*:*port*/ocnso/1.1/ns/*networkServiceId*/networks

where *networkServiceId* is the ID of the network service

Sample Response

```
{
  "nsID": "375005",
  "nsdName": "NPaaS_NSD",
  "nsName": "29_1.3_AMns_Service",
  "networks": [
    {
      "networkName": "nfvo-poc3-mgmt",
      "networkID": "nfvo-poc3-mgmt",
      "externalID": "109ae4cf-3cea-4729-a24f-957c4ed6d3c6",
      "subnets": [
        {
          "startIP": "192.0.2.0",
          "prefix": "24",
          "externalID": "fb791563-7c8b-454c-a1eb-87399e6837dc"
        }
      ]
    },
    {
      "networkName": "nfvo-demo-pkt-in-v2",
      "networkID": "nfvo-demo-pkt-in-v2",
      "externalID": "2277b6e2-eb2d-4cc2-b80c-6d6c38f35ab0",
      "subnets": [
        {
          "startIP": "192.0.2.128",
          "prefix": "25",
          "externalID": "d47bf43a-57bd-4b17-b559-505a426d7359"
        }
      ]
    },
    {
      "networkName": "nfvo-demo-pkt-out-v2",
      "networkID": "nfvo-demo-pkt-out-v2",
      "externalID": "3b45febc-4531-4751-ac55-9e43bd53897a",
      "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_name*:*port*/ocnso/1.1/ns/*networkServiceId*/endPoints

where *networkServiceId* is the ID of the network service

Sample Response

```
{
  "nsID": "375005",
  "nsdName": "NPaaS_NSD",
  "nsName": "29_1.3_AMns_Service",
  "endPoints": [
    {
      "name": "29_1.3_AMcnsmr",
      "ipAddress": "207.123.34.2"
    }
  ]
}
```

Get Status Information of a Network Service

Retrieves the status information about a network service.

Method

GET

URL

http://*nso_host_name*:*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_AMns_Service",
  "status": "IN_SERVICE"
}
```

Get Details about a VNF

Retrieves the details about a VNF.

Method

GET

URL

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

where *vnfId* is the ID of the VNF

Sample Response

```
{
  "vnfId": "300003",
  "vnfName": "29_1.3_AMvnf",
  "vnfStatus": "INSTALLED",
  "vnfDescriptor": "Juniper_vSRX_VNFD",
  "vnfServiceId": "375006",
  "vnfServiceName": "29_1.3_AMvnf_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-demo-pkt-in-v2",
        "externalID": "8f2468de-c4b1-4656-b23f-ccd5c26b9d83"
      }
    },
    {
      "id": "300003-2",
      "name": "CP02",
      "ipAddress": {
        "address": "192.0.2.120",
        "network": "nfvo-demo-pkt-out-v2",
        "externalID": "8ab6b415-b04a-458c-97bc-d4ef2eb550c3"
      }
    },
    {
      "id": "300003-3",
      "name": "CP03",
      "ipAddress": {
        "address": "192.0.2.8",
        "network": "nfvo-poc3-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_name*:*port*/ocnso/1.1/vnf/*vnfId*/status

where *vnfId* is the ID of the VNF

Sample Response

```
{
  "vnfServiceId": "375006",
  "vnfServiceName": "29_1.3_AMvnf_Juniper_vSRX_VNFD_Service",
  "vnfServiceStatus": "IN_SERVICE",
  "vnfServiceDescriptor": "Juniper_vSRX_ServiceDescriptor",
  "vmStatus": "ACTIVE"
}
```