Datacom Network Open Programmability

Lab Guide

Issue: 1.0



Huawei Technologies Co., Ltd.

|  |
| --- |
| Copyright © Huawei Technologies Co., Ltd. 2021. All rights reserved.  No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.  Trademarks and Permissions  C:\Users\jwx341670\AppData\Local\Microsoft\Windows\INetCache\Content.Word\HW_POS_RBG_Vertical-150ppi.png and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.  All other trademarks and trade names mentioned in this document are the property of their respective holders.  Notice  The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.  The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied. |

|  |  |
| --- | --- |
| **Huawei Technologies Co., Ltd.** | |
| Address: | Huawei Industrial Base Bantian, Longgang Shenzhen 518129  People's Republic of China |
| Website: | http://[e](http://e.huawei.com/).huawei.com |

About This Document

Background Knowledge Required

This document is intended for network automation engineers seeking advanced learning. You are expected to have the following knowledge and skills:

* Python programming basics
* RESTful fundamentals
* NETCONF YANG fundamentals
* Knowledge about Datacom Network Open Programmability

Lab Environment

Environment Description

This document describes how to use iMaster NCE for practices of open programmability in the local lab environment.

Obtain the Datacom Network Open Programmability version from Huawei and deploy it in the local lab environment.

Preparing the Local Compilation Environment

Install Python 3, PyCharm, Java 1.8, Gpg4Win, and Datacom Network Open Programmability SDK.

* Python 3

Download the Python 3 installation package from the official website (https://www.python.org/downloads/release/python-382/), select the default configuration, and complete the installation as prompted.

* PyCharm

Download the PyCharm installation package from the official website (https://www.jetbrains.com/pycharm/), select the default configuration, and complete the installation as prompted.

* Java 1.8

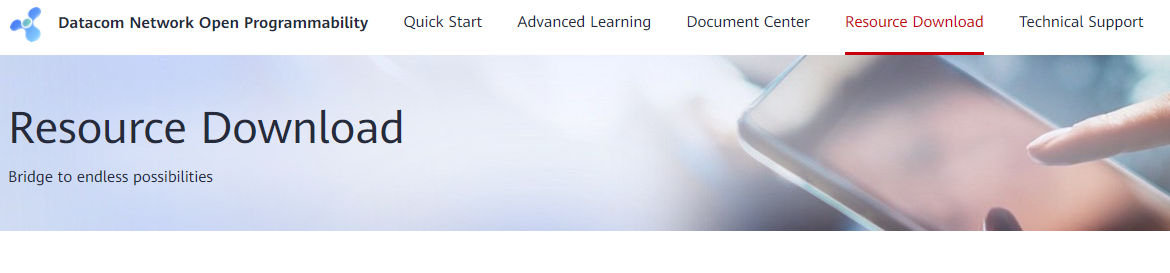
Download Java 1.8 from the official website (<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>), select the default configuration, and complete the installation as prompted.

* Gpg4Win

Download the Gpg4Win installation package from the official website (<https://www.gpg4win.org/>), select the default configuration, and complete the installation as prompted.

* Datacom Network Open Programmability SDK

Choose **Datacom Network Open Programmability** > **Resource Download** in the Developer Community to download the SDK to the local PC.



Open the **cmd** window, and go to the directory where the **python-aoc-api-*xxx*.rar** package is decompressed. Enter the **dir** command and press **Enter** to view files in the directory.

D:\AOC\SDK>dir

2020/04/09 16:02 121,569 aoc\_api-2.0.0-py3-none-any.whl

1 File(s) 121,569 bytes

2 Dir(s) 103,275,847,680 bytes free

Run the **pip install aoc\_api-2.0.0-py3-none-any.whl** command and press **Enter** to install the SDK file.

D:\AOC\SDK>pip install aoc\_api-2.0.0-py3-none-any.whl

Looking in indexes: http://mirrors.tools.huawei.com/pypi/simple

Processing d:\download\sdk\aoc\_api-2.0.0-py3-none-any.whl

….

Installing collected packages: netaddr, protobuf, aoc-api

Successfully installed aoc-api-2.0.0 netaddr-0.8.0 protobuf-3.12.2

If the preceding information is displayed, the installation is successful.

Contents

[About This Document 3](#_Toc74659494)

[1 Building New Services – SSP Package Practice 2](#_Toc74659495)

[1.1 Background 2](#_Toc74659496)

[1.1.1 SSP Package 2](#_Toc74659497)

[1.1.2 Open Service Capabilities 3](#_Toc74659498)

[1.2 Introduction 4](#_Toc74659499)

[1.2.1 Networking 4](#_Toc74659500)

[1.2.2 Objectives 4](#_Toc74659501)

[1.2.3 Procedure 9](#_Toc74659502)

[1.3 Environment Preparation 9](#_Toc74659503)

[1.4 Compiling an SSP Package 10](#_Toc74659504)

[1.4.1 Creating an SSP Package Template 10](#_Toc74659505)

[1.4.2 Exporting the SSP Package Template to the Local IDE 10](#_Toc74659506)

[1.4.3 Compiling the Service YANG Model 11](#_Toc74659507)

[1.4.4 Verifying the YANG Model File 17](#_Toc74659508)

[1.4.5 Creating a Southbound Template 17](#_Toc74659509)

[1.4.6 Compiling Python Mapping Code 26](#_Toc74659510)

[1.4.7 Compiling Test Cases 30](#_Toc74659511)

[1.4.8 Configuring Keys 34](#_Toc74659512)

[1.4.9 Generating an SSP Package 39](#_Toc74659513)

[1.5 Delivering Network Services on the GUI 40](#_Toc74659514)

[1.5.1 Delivering a Network Service 40](#_Toc74659515)

[1.5.2 Modifying a Network Service 45](#_Toc74659516)

[1.5.3 Deleting a Network Service 48](#_Toc74659517)

[1.6 Delivering Network Services Through the Northbound API 49](#_Toc74659518)

[1.6.1 Creating a Northbound User 49](#_Toc74659519)

[1.6.2 Creating Configurations 51](#_Toc74659520)

[1.6.3 Checking Configurations 58](#_Toc74659521)

[1.6.4 Modifying Configurations 60](#_Toc74659522)

[1.6.5 Deleting Configurations 67](#_Toc74659523)

[1.7 Quiz 69](#_Toc74659524)

# Building New Services – SSP Package Practice

## Background

Based on the YANG model-driven open architecture, Huawei Datacom Network Openness Programmability enables open programmability at the network element (NE) and network layers in the form of specific NE driver (SND) and specific service plug-in (SSP) packages. It automatically generates configuration pages and northbound APIs, implementing fast interconnection with new devices, building of new network services, and opening of network service capabilities.

This document provides guidance for you to compile an SSP package to open network service capabilities, that is, use iMaster NCE to manage network devices, customize network services, and generate northbound APIs. Upon completion of this course, you will be able to:

* Master the process of compiling an SSP package based on service requirements.
* Develop a service YANG file.
* Develop a Jinja2 template.

### SSP Package

The SSP package is a type of iMaster NCE software packages, which defines a data model for configuring network-layer services. The data model usually contains a Jinja2 template file, a Python mapping script, and a service YANG model.

* The Jinja2 template describes the data structure of an NE. In the template, variables reference service parameters or new parameters obtained through Python service processing. In addition, Jinja2 syntax is used to perform operations such as interpolation, condition-based determination, and looping.
* The Python mapping script describes how to process submitted data and fill the data into the Jinja2 template.
* The service YANG model describes service parameters and is constructed based on service input.



### Open Service Capabilities

iMaster NCE provides open network service capabilities. Based on service YANG models, Datacom Network Open Programmability automatically generates southbound and northbound APIs to quickly build new services. With the Easymap algorithm, you only need to write the creation process. The Easymap algorithm automatically implements the modification and deletion processes through comparison, simplifying the programming process and facilitating development.



This lab mainly introduces the process and operations of opening service capabilities on iMaster NCE.

## Introduction

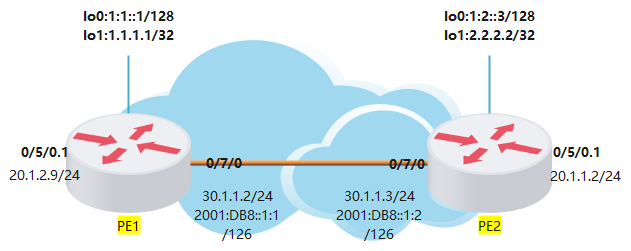
### Networking



This lab involves three objects: NE8000 M8, iMaster NCE, and local compilation environment.

### Objectives

In this lab, you will customize a VPN service (to be delivered to PEs) and access the network service through a northbound IP address. The network topology is as follows.



1. The global VPN configurations have been performed on the device. You need to analyze the configurations that need to be delivered for VPN services, create a VPN instance and a sub-interface, bind the sub-interface to the VPN instance, and import VPN routes (BGP-related configurations). The commands to be run on the corresponding device are as follows:
2. Create VPN instance **5G-RAN** and set the RT and RD.

ip vpn-instance 5G-RAN

ipv4-family

route-distinguisher 100:1

vpn-target 100:11 export-extcommunity

vpn-target 100:11 import-extcommunity

1. Create a sub-interface, configure the sub-interface address and VLAN, and bind the sub-interface to the VPN instance.

interface GigabitEthernet 0/5/0.1

vlan-type dot1q 20

ip binding vpn-instance 5G-RAN

ip address 20.1.2.9 255.255.255.0

1. Configure VPN routes, import static routes and direct routes, and configure BGP peers.

bgp 100

ipv4-family vpn-instance 5G-RAN

import-route direct

import-route static

peer 2.2.2.2 as-number 100

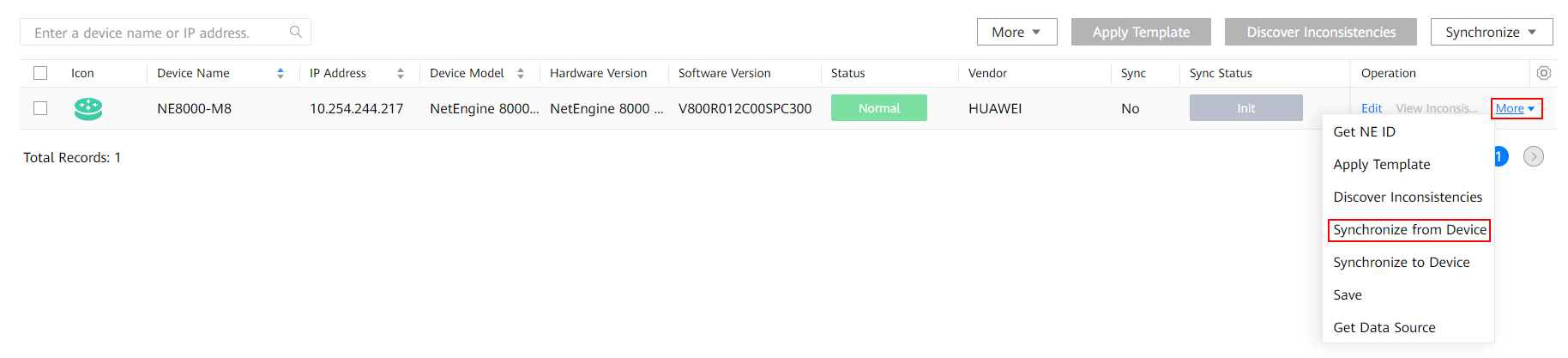
1. Enable the YANG interface to deliver BGP configurations.

bgp yang-mode enable

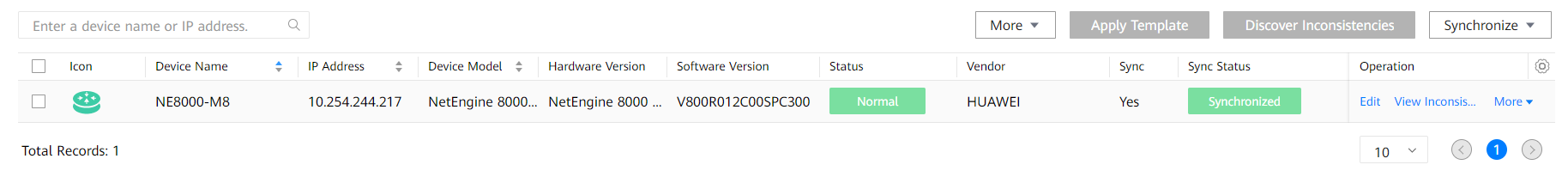
1. Analyze how to deliver the preceding commands to devices through NETCONF. That is, analyze the YANG model mapped to the commands and the XML data corresponding to the YANG model.

After the devices are managed, device configurations are synchronized, the preceding commands are configured on the device, and the XML data template corresponding to the commands is exported based on the NE configuration difference discovery capability.

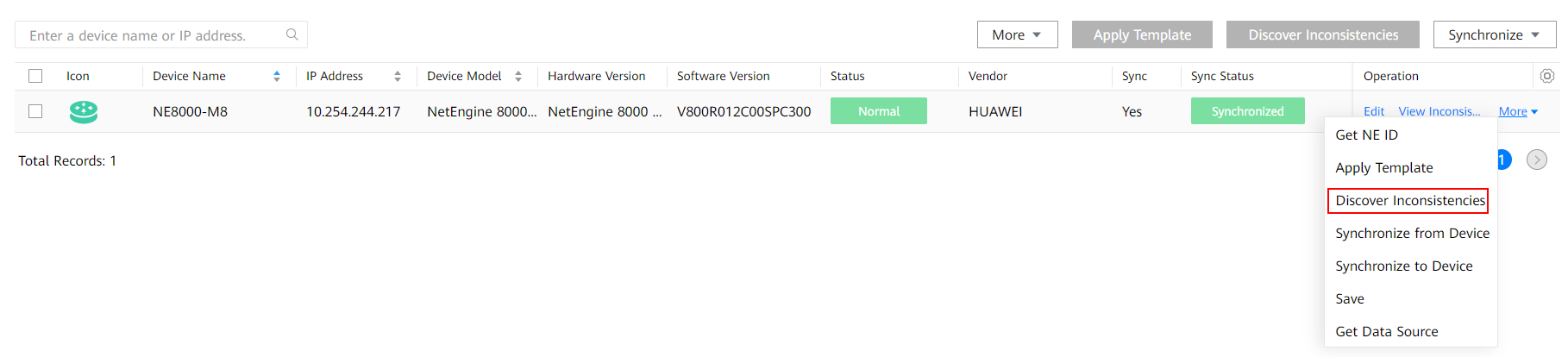
1. Perform synchronization.



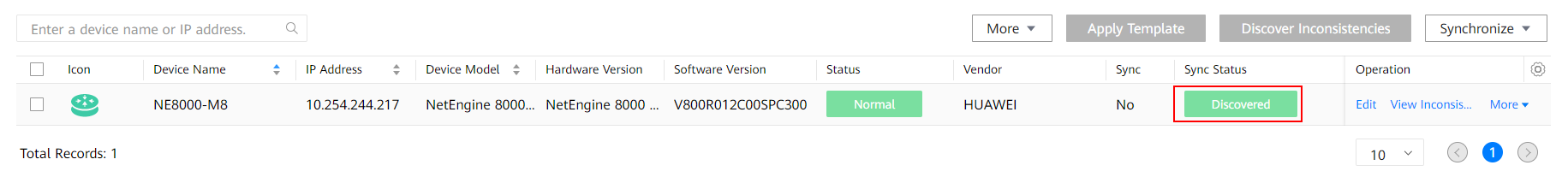
1. Wait until synchronization is completed.



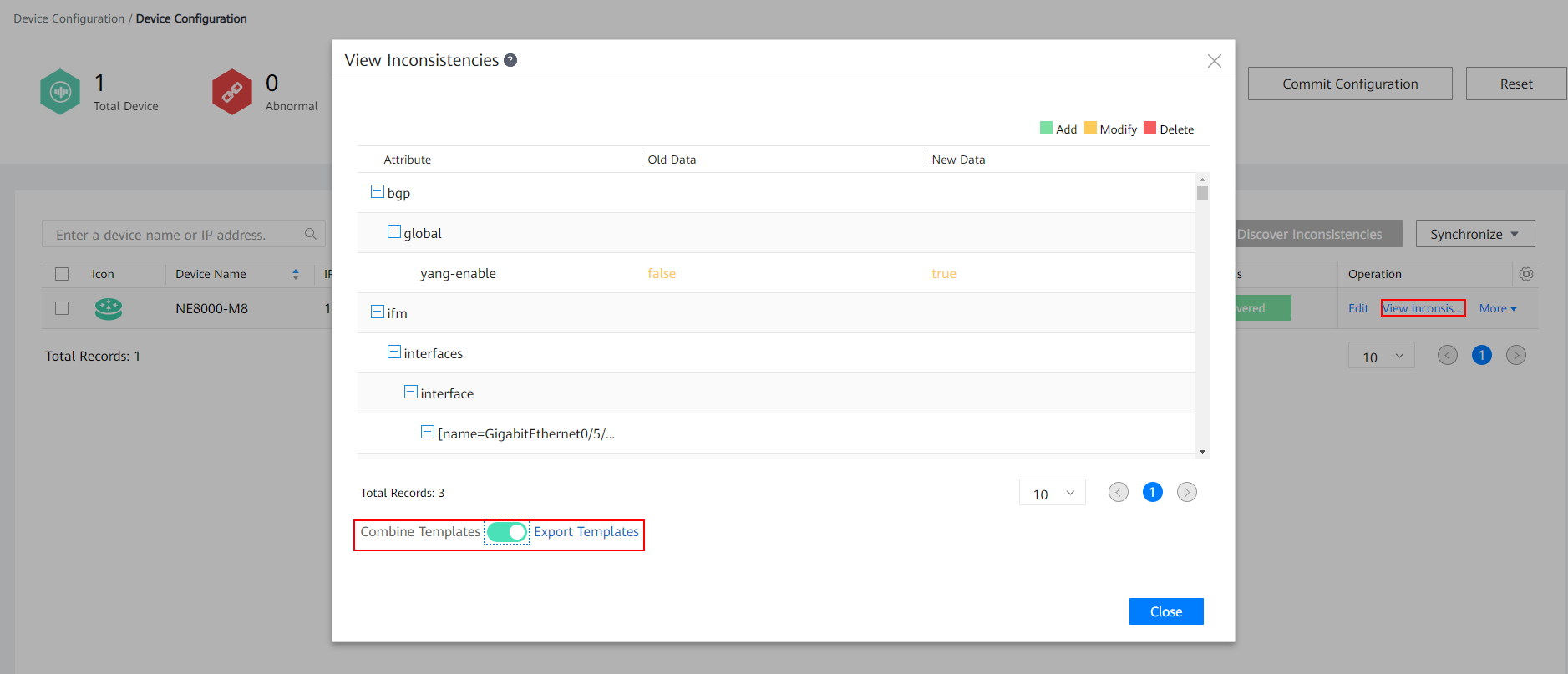
1. Configure VPN service-related commands on the device.
2. Discover differences.



1. Difference discovery completed



1. Export the template (XML data corresponding to the commands).



1. View the exported template and retain only the configuration to be delivered. The other configurations can be deleted.

<bgp xmlns="urn:huawei:yang:huawei-bgp" xmlns:ns0="urn:ietf:params:xml:ns:netconf:base:1.0" ns0:operation="merge">

<global>

<yang-enable>true</yang-enable>

</global>

</bgp>

<ifm xmlns="urn:huawei:yang:huawei-ifm" xmlns:ns0="urn:ietf:params:xml:ns:netconf:base:1.0" ns0:operation="merge">

<interfaces>

<interface>

<name>GigabitEthernet0/5/0.1</name>

...

<ipv4 xmlns="urn:huawei:yang:huawei-ip">

<addresses>

<address>

<ip>20.1.2.9</ip>

<mask>255.255.255.0</mask>

<type>main</type>

</address>

</addresses>

</ipv4>

...

<vrf-name>5G-RAN</vrf-name>

...

</interface>

</interfaces>

</ifm>

<network-instance xmlns="urn:huawei:yang:huawei-network-instance" xmlns:ns0="urn:ietf:params:xml:ns:netconf:base:1.0" ns0:operation="merge">

<instances>

<instance>

<name>5G-RAN</name>

<bgp xmlns="urn:huawei:yang:huawei-bgp">

<base-process>

<peers>

<peer>

<address>2.2.2.2</address>

...

<remote-as>100</remote-as>

...

</afs>

...

</peer>

</peers>

<afs>

<af>

<type>ipv4uni</type>

...

<import-routes>

<import-route>

<protocol>direct</protocol>

<process-id>0</process-id>

</import-route>

<import-route>

<protocol>static</protocol>

<process-id>0</process-id>

</import-route>

</import-routes>

</ipv4-unicast>

</af>

</afs>

...

</base-process>

</bgp>

...

<afs xmlns="urn:huawei:yang:huawei-l3vpn">

<af>

<type>ipv4-unicast</type>

...

<label-mode>per-instance</label-mode>

...

<vpn-targets>

<vpn-target>

<value>100:11</value>

<type>export-extcommunity</type>

</vpn-target>

<vpn-target>

<value>100:11</value>

<type>import-extcommunity</type>

</vpn-target>

</vpn-targets>

...

<route-distinguisher>100:1</route-distinguisher>

</af>

</afs>

</instance>

</instances>

</network-instance>

1. Sort out the input parameters for network configuration based on the network service.
2. Device name and VPN instance, which are used to specify the device to which a specific VPN service is to be delivered. The device name and VPN instance are combined into a key to identify the VPN service.

|  |  |
| --- | --- |
| Parameter | Value |
| deviceName | PE1 |
| vrfName | 5G-RAN |

1. RT and RD for the VPN, which are used to filter routing information

| Parameter | Value |
| --- | --- |
| rd | 100:1 |
| rt | 100:11 |

1. Sub-interface name, sub-interface description, sub-interface IP address, and mask

| Parameter | Value |
| --- | --- |
| ifName | GigabitEthernet0/5/0.1 |
| description | connect to pe2 |
| ip | 20.1.2.9 |
| subnet | 255.255.255.0 |

1. BGP peer: The peer IP address needs to be obtained. The AS number of a BGP peer is 100 by default and does not need to be obtained as a northbound input parameter.

| Parameter | Value |
| --- | --- |
| peerAddress | 2.2.2.2 |

After the preceding configurations, you only need to enter the corresponding parameter values, and VPN services are automatically provisioned on network devices.



Before performing this lab, ensure that the SND package has been loaded to and devices have been managed by iMaster NCE.

### Procedure

The procedure for this lab is as follows:

Environment preparation: Prepare the local environment and iMaster NCE lab environment.

Compile an SSP package locally and load it to iMaster NCE.

Provision network services on the iMaster NCE GUI.

Provision network services through iMaster NCE northbound APIs.

## Environment Preparation

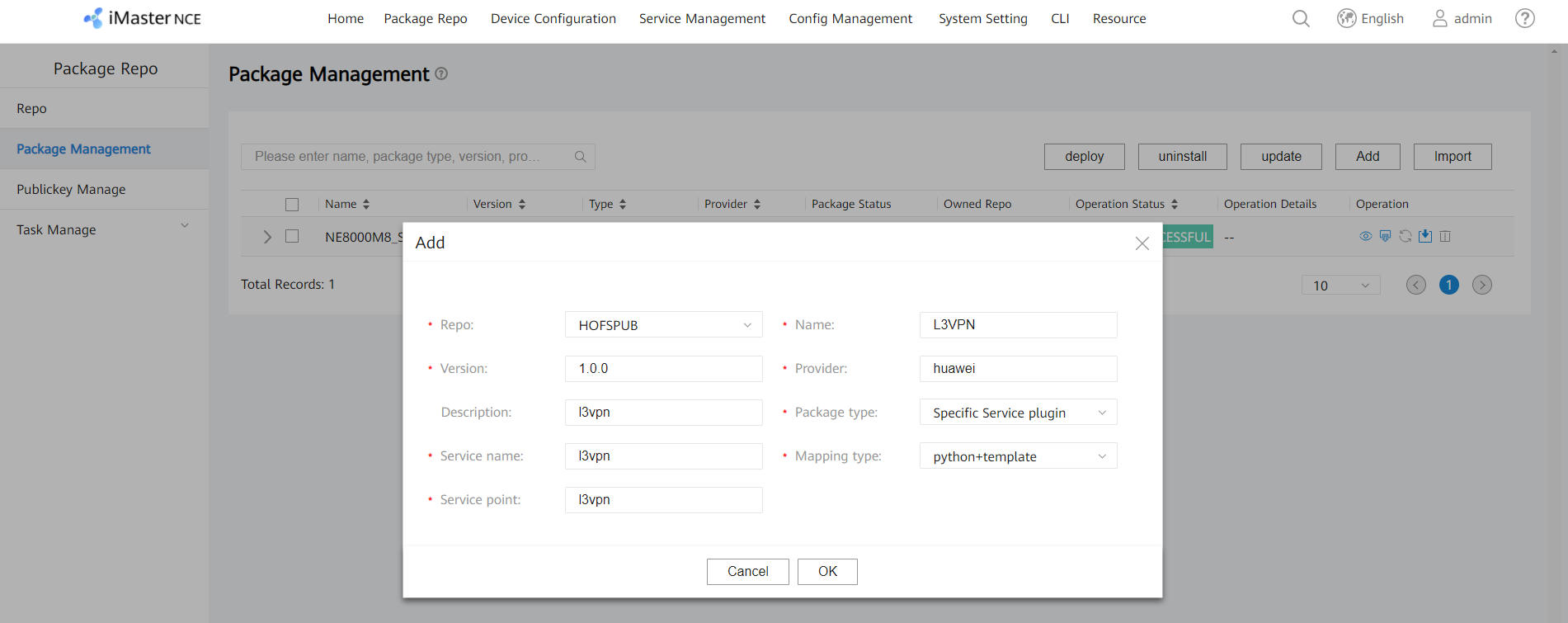
For details about how to prepare the local environment, see "About This Document > Lab Environment > Preparing the Local Compilation Environment."

## Compiling an SSP Package

This section describes how to compile an SSP package for the NE8000 M8 in the local environment and load it to iMaster NCE. For details about code samples and resource files related to this lab, see [https://devzone.huawei.com/apistudio/sample/aoc/apiSdk.html.](https://devzone.huawei.com/apistudio/sample/aoc/apiSdk.html)

### Creating an SSP Package Template

Log in to iMaster NCE. On the homepage, click **Service Programming**. On the **Package Management** page, click **Add**, set SSP package parameters, and click **OK**. An SSP package template is created.

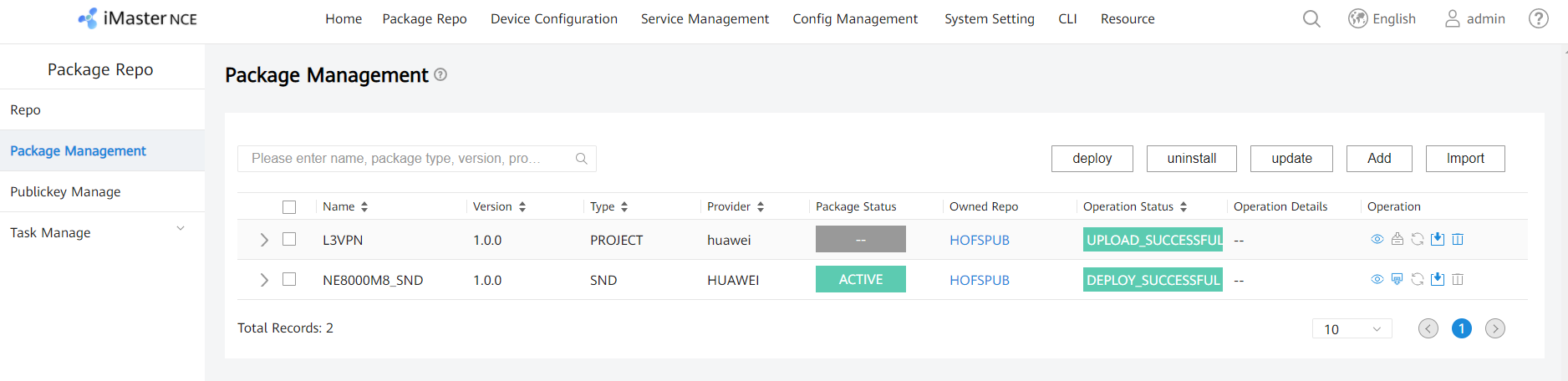


The parameters are as follows:

* **Name**: L3VPN
* **Version**: 1.0.0
* **Provider**: HUAWEI
* **Package type**: Specific Service plug-in
* **Mapping type**: python+template
* **Service name**: l3vpn
* **Service point**: l3vpn

### Exporting the SSP Package Template to the Local IDE

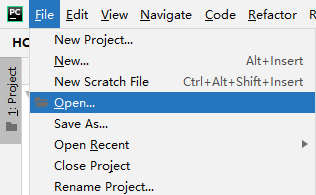
After the template is successfully created, download it to the local PC. Click  in the **Operation** column.



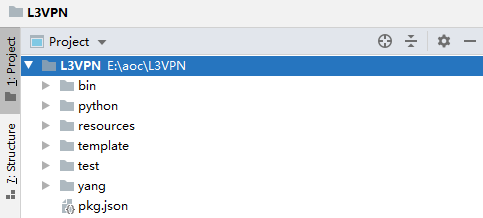
Decompress the SSP package to the local PC.



Use PyCharm to open this directory.



The directory structure is as follows:



* **bin**: stores executable scripts, including tools and scripts for packing.
* **key**: stores private keys.
* **python**: stores Python code, including Python scripts used to implement service callback logic.
* **template**: stores Jinja2 template files, which are used to map service models to device models.
* **test**: stores unit test code, including scripts used to perform local unit tests on software package functions.
* **yang**: YANG model of services.
* **pkg.json**: package configuration file, which is used to set basic attributes and callback hooks of the current software package.

### Compiling the Service YANG Model

The service YANG model defines the configurations to be delivered for VPN services. For details, see section 1.2.2 Objectives.

In the default YANG model of the SSP package template, the service name corresponds to the service YANG module name, and the service point corresponds to **app:application-definition** of the service YANG model.

module l3vpn {

namespace "http://example.com/ l3vpn ";

prefix " l3vpn ";

import huawei-ac-applications {

prefix app;

}

description

"The module for l3vpn example.";

revision 2018-12-09 {

description "Initial revision.";

}

augment "/app:applications"{

list user {

app:application-definition " l3vpn ";

key "name";

leaf name {

type string;

}

leaf full-name {

type string;

}

leaf class {

type string;

}

}

}

}

Define the service identifier in the YANG model. In this example, the combination of the device name and VPN instance is used as the service identifier.

| Parameter | Value | Description |
| --- | --- | --- |
| deviceName | PE1 | Device name. |
| vrfName | 5G-RAN | VPN name. |

module l3vpn{  
 ……  
 augment "/app:applications"{   
 list l3vpn {  
 app:application-definition "l3vpn";  
 key "deviceName vrfName";  
 leaf deviceName {  
 type string {  
 length "1..512";  
 }  
 }  
  
 leaf vrfName {  
 description "Defines a type of service component identifier.";  
 type string {  
 length 1..255 {  
 description "VPN Routing/Forwarding instance name, support 1-255 characters.";  
 }  
 }  
 mandatory true;  
 }

Define the RD and RT of VPN services in the YANG model.

| Parameter | Value | Description |
| --- | --- | --- |
| rd | 100:1 | RD. |
| rt | 100:11 | RT. |

module L3VPN {  
 ……  
 augment "/app:applications"{   
 list l3vpn {

……  
 leaf rd {  
 description "BGP route distinguisher";  
 type string;  
 }  
  
 leaf rt {  
 description "Route target extended community as per RFC4360";  
 type string;  
 }

Define the sub-interface configuration in the YANG model.

| Parameter | Value |
| --- | --- |
| ifName | GigabitEthernet0/5/0.1 |
| description | connect to pe2 |
| ip | 20.1.2.9 |
| subnet | 255.255.255.0 |

module l3vpn {

...

augment "/app:applications"{

list l3vpn {

...

leaf ifName {

type string {

length "1..63";

}

}

leaf description {

type string {

length "0..242";

}

}

leaf ip {

type inet:ipv4-address;

}

leaf subnet {

type string {

length "9..15";

pattern '((([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5]))';

}

description

"IPv4 mask support mask format.";

}

}

}

}

Define peer information in the YANG model.

| Parameter | Value | Description |
| --- | --- | --- |
| peerAddress | 2.2.2.2 | Peer IP address. |

module l3vpn {

...

augment "/app:applications"{

list l3vpn {

...

leaf peerAddress {

type inet:ipv4-address;

}

}

}

}

The complete service YANG model is as follows:

module l3vpn {

namespace "http://example.com/l3vpn";

prefix "l3vpn";

import huawei-ac-applications {

prefix app;

}

import ietf-inet-types {

prefix inet;

}

description

"The module for l3vpn example.";

revision 2020-11-05 {

description "Initial revision.";

}

augment "/app:applications"{

list l3vpn {

app:application-definition "l3vpn";

key "deviceName vrfName";

leaf deviceName {

type string {

length "1..512";

}

}

leaf vrfName {

description "Defines a type of service component identifier.";

type string {

length 1..255 {

description "VPN Routing/Forwarding instance name, support 1-255 characters.";

}

}

mandatory true;

}

leaf rd {

description "BGP route distinguisher";

type string;

}

leaf rt {

description "Route target extended community as per RFC4360";

type string;

}

leaf ifName {

type string {

length "1..63";

}

}

leaf description {

type string {

length "0..242";

}

}

leaf ip {

type inet:ipv4-address;

}

leaf subnet {

type string {

length "9..15";

pattern '((([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}([1-9]?[0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5]))';

}

description

"IPv4 mask support mask format.";

}

leaf peerAddress {

type inet:ipv4-address;

}

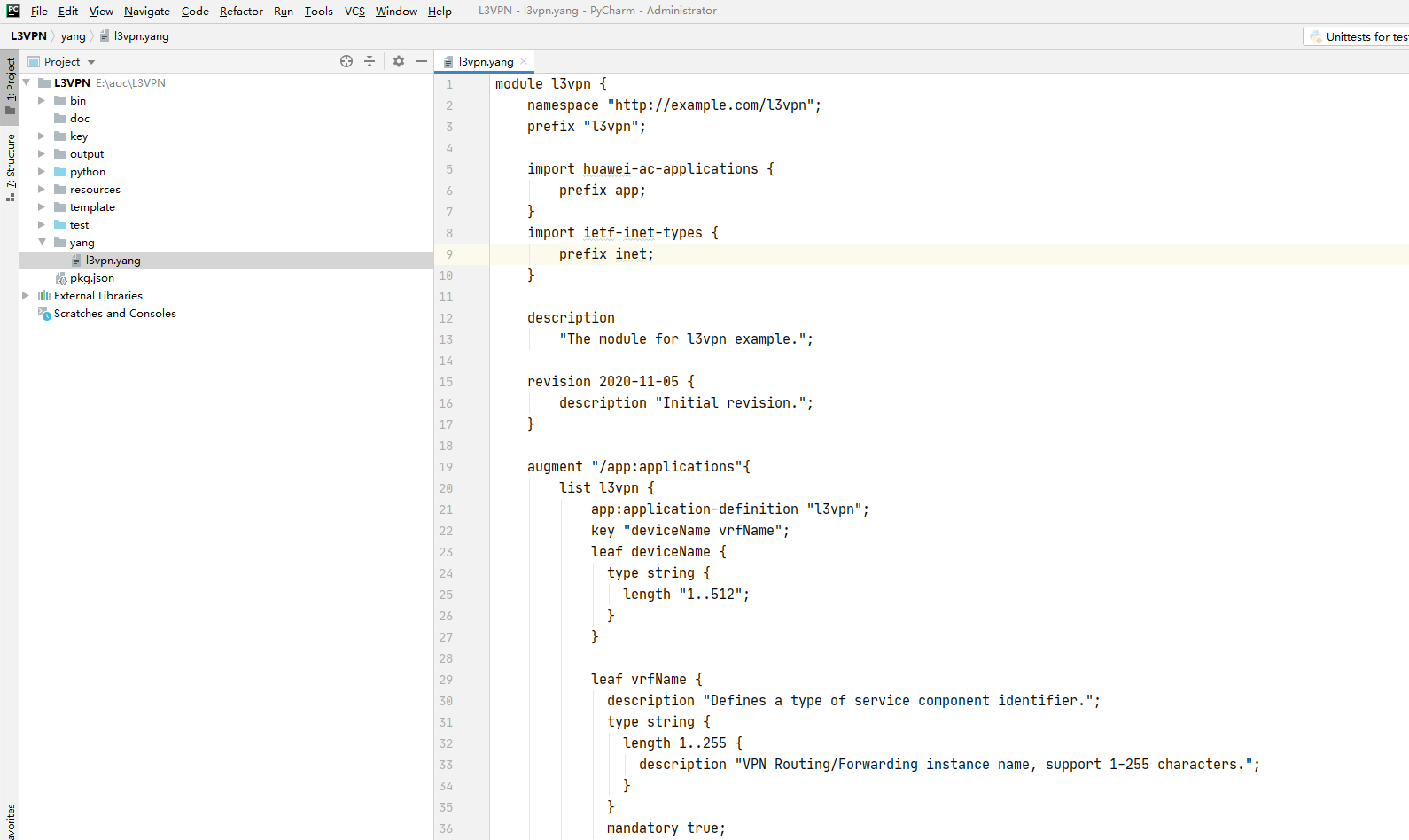
}

}

}

The preceding code can be directly invoked without modification.

The following figure shows the code in PyCharm.



In this example, the service YANG module is named **l3vpn**, and the **huawei-ac-applications** module is imported, and the **l3vpn** module is added to **/app:applications** of the app module.

The module contains a **list** node named **l3vpn**. The list defines parameters such as **deviceName**, **vrfName**, **rd**, **rt**, **ifName**, **description**, **ip**, **subnet**, and **peerAddress**, and defines the service parameters to be entered through the northbound API.

For details about YANG implementations, see RFC 7950.

### Verifying the YANG Model File

Save the service YANG model and its dependent YANG files to the directory where the YANG verification tool is stored. Open the **cmd** window, go to the directory, and run the **java -jar .\yang-offline-util.jar validate console path .** command.

If no command output is displayed, the verification is successful.

### Creating a Southbound Template

The southbound template is the XML format of the YANG model and is closely related to the device type and model.

The southbound template uses Jinja2 code. Jinja2 is a template system that provides simple control statements, expressions, and inheritance capabilities, making southbound template development more efficient. It has the following common syntax:

* {%...%}: control structure.
* {{...}}: expressions, which can be constants, variables, mathematical formulas, and logical statements.
* {# ... #}: comment.

For more information about the Jinja2 syntax and basic principles, visit <https://jinja.palletsprojects.com/en/2.11.x/>.

#### Exporting a Southbound Template

The following shows the default Jinja2 template in an SSP package template. **nes** indicates the devices to which the Jinja2 template can be delivered and can contain multiple **ne** labels. **ne** specifies the device to which services are to be delivered. **neid** indicates the device ID. The other is the XML data of the YANG model. XML data can be exported from the device configuration page based on service requirements and can be used after further processing.

<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">

<nes>

<ne>

<neid>{{"devicename"| to\_ne\_id}}</neid>

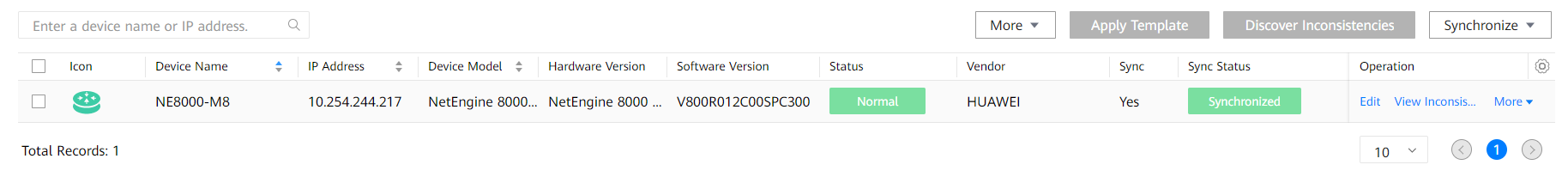
<user>{{user.name}}</user>

</ne>

</nes>

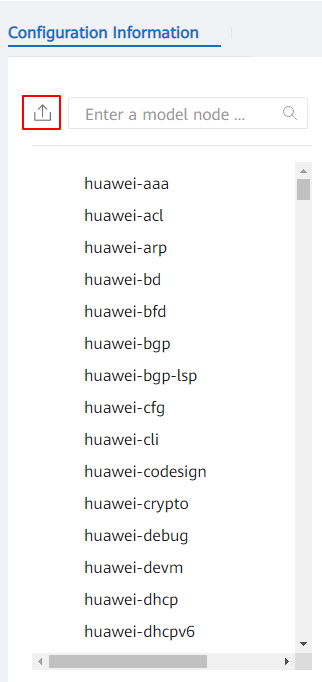
</inventory-cfg>

Choose **Device** **Configuration** > **Device Configuration** from the main menu. Click **Edit** in the **Operation** column. The configuration page of the current device is displayed.

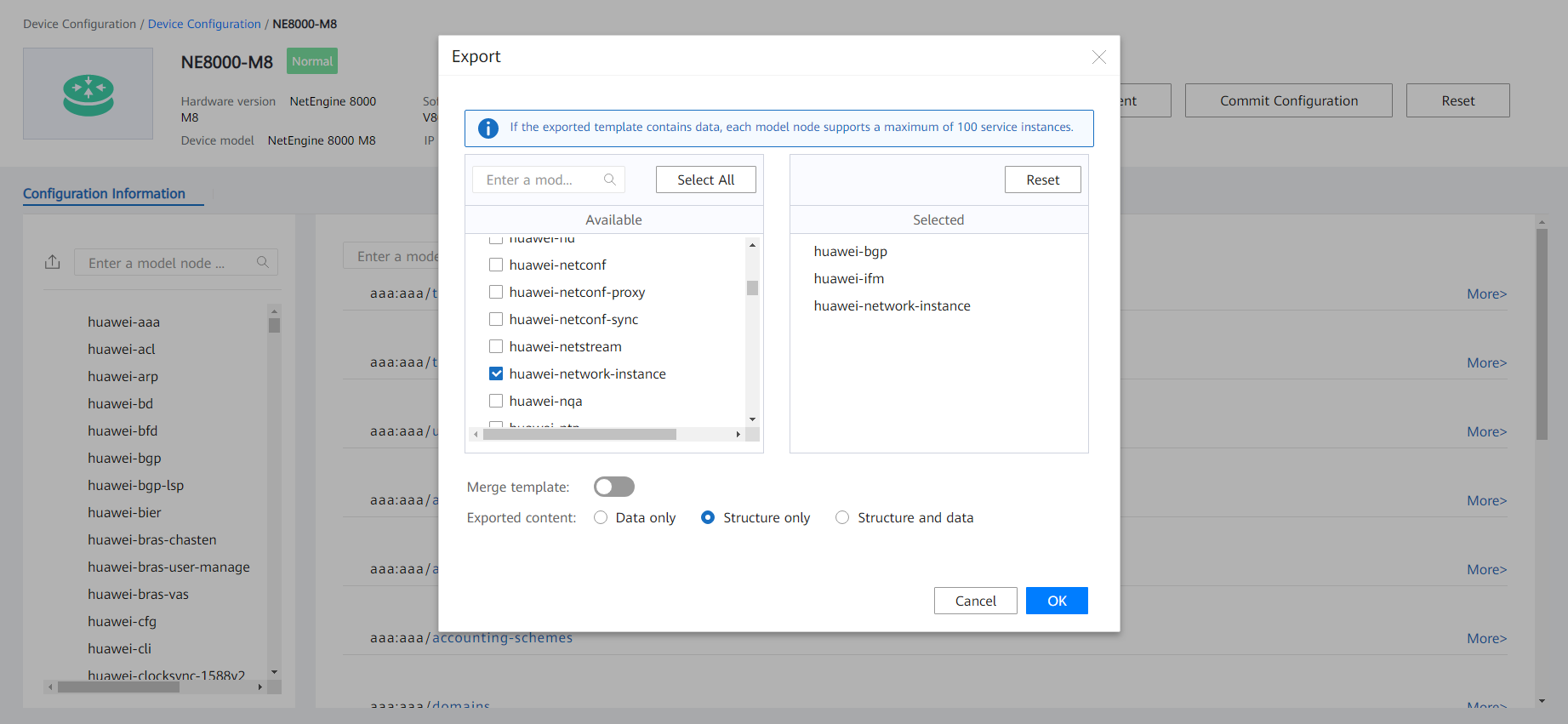


In this example, the YANG files **huawei-ifm**, **huawei-network-instance**, and **huawei-bgp** need to be invoked for the southbound configuration of the network service to orchestrate the sub-interface and L3VPN services.

Click .



Enter the module name (**ifm**, **network-instance**, and **bgp**) in the search box, and select **huawei-ifm**, **huawei-network-instance**, and **huawei-bgp**. In the **Export** dialog box that is displayed, enable **Merge template**, set the export mode to **Structure only**, and export the template.



The template package is exported into a compressed file, whose name contains the export time.

 combinedTemplate\_20210226044527160.j2

After decompressing the package, use an editor to open the .j2 file extracted from the compressed file. The .j2 file contains the structure of each feature, and you need to fill in data for each feature.

huawei-bgp:

<bgp xmlns="urn:huawei:yang:huawei-bgp">

<global>

<yang-enable/>

</global>

<base-process>

<enable/>

<as/>

</base-process>

</bgp>

huawei-ifm:

<ifm xmlns="urn:huawei:yang:huawei-ifm">

<interfaces>

<interface>

<name/>

...

<ipv4 xmlns="urn:huawei:yang:huawei-ip">

<addresses>

<address>

<ip/>

<mask/>

<type/>

</address>

</addresses>

</ipv4>

...

<vrf-name/>

...

</interface>

</ifm>

huawei-network-instance:

<network-instance xmlns="urn:huawei:yang:huawei-network-instance">

<instances>

<instance>

<name/>

<bgp xmlns="urn:huawei:yang:huawei-bgp">

<base-process>

<afs>

<af>

<type/>

<ipv4-unicast>

<import-routes>

<import-route>

<protocol/>

<process-id/>

</import-route>

<import-route>

<protocol/>

<process-id/>

</import-route>

</import-routes>

</ipv4-unicast>

</af>

</afs>

<peers>

<peer>

<address/>

<remote-as/>

</peer>

</peers>

</base-process>

</bgp>

<afs xmlns="urn:huawei:yang:huawei-l3vpn">

<af>

<type/>

<vpn-targets>

<vpn-target>

<value/>

<type/>

</vpn-target>

<vpn-target>

<value/>

<type/>

</vpn-target>

</vpn-targets>

...

<route-distinguisher/>

<tunnel-policy/>

</af>

</afs>

</instance>

...

</instances>

</network-instance>

Set related parameters based on the objectives in section 1.2.2 Objectives.

huawei-bgp:

Set yang-enable to true.

huawei-network-instance:

| Parameter | Value |
| --- | --- |
| name | 5G-RAN |
| description | vrf |
| rd | 100:1 |
| rt | 100:11 |

huawei-ifm:

| Parameter | Value |
| --- | --- |
| name | GigabitEthernet0/5/0.1 |
| description | connect to pe2 |
| ip | 20.1.2.9 |
| mask | 255.255.255.0 |

Compile the southbound template of the NE8000 M8, make the Jinja2 template based on the **ifm**, **network-instance**, and **bgp** configurations (extract variables based on the parameters to be delivered), and save the compiled template to the template file **L3VPN/template/l3vpn/NE8000M8.j2**.

huawei-bgp:

<bgp xmlns="urn:huawei:yang:huawei-bgp">

<global>

<yang-enable>true</yang-enable>

</global>

</bgp>

huawei-network-instance:

<network-instance xmlns="urn:huawei:yang:huawei-network-instance">

<instances>

<instance>

<name>{{l3vpn.vrfName}}</name>

{%- if l3vpn.vrfDes %}

<description>{{vrfDes}}</description>

{%- endif %}

<afs xmlns="urn:huawei:yang:huawei-l3vpn">

<af>

<type>ipv4-unicast</type>

<route-distinguisher>{{ l3vpn.rd }}</route-distinguisher>

<tunnel-policy>LDP</tunnel-policy>

<vpn-targets>

<vpn-target>

<value>{{ l3vpn.rt }}</value>

<type>export-extcommunity</type>

</vpn-target>

<vpn-target>

<value>{{ l3vpn.rt }}</value>

<type>import-extcommunity</type>

</vpn-target>

</vpn-targets>

</af>

</afs>

<bgp xmlns="urn:huawei:yang:huawei-bgp">

<base-process>

<afs>

<af>

<type>ipv4uni</type>

<ipv4-unicast>

<import-routes>

<import-route>

<protocol>direct</protocol>

<process-id>0</process-id>

</import-route>

<import-route>

<protocol>static</protocol>

<process-id>0</process-id>

</import-route>

</import-routes>

</ipv4-unicast>

</af>

</afs>

<peers>

<peer>

<address>{{ l3vpn.peerAddress }}</address>

<remote-as>100</remote-as>

</peer>

</peers>

</base-process>

</bgp>

</instance>

</instances>

</network-instance>

huawei-ifm:

<ifm xmlns="urn:huawei:yang:huawei-ifm">

<interfaces>

<interface>

<name>{{ l3vpn.ifName }}</name>

<type>GigabitEthernet</type>

<description>{{ l3vpn.description }}</description>

<vrf-name>{{ l3vpn.vrfName }}</vrf-name>

<ipv4 xmlns="urn:huawei:yang:huawei-ip">

<addresses>

<address>

<ip>{{ l3vpn.ip }}</ip>

<type>main</type>

<mask>{{ l3vpn.subnet }}</mask>

</address>

</addresses>

</ipv4>

</interface>

</interfaces>

</ifm>

#### Complete Code

<bgp xmlns="urn:huawei:yang:huawei-bgp">

<global>

<yang-enable>true</yang-enable>

</global>

</bgp>

<network-instance xmlns="urn:huawei:yang:huawei-network-instance">

<instances>

<instance>

<name>{{l3vpn.vrfName}}</name>

{%- if l3vpn.vrfDes %}

<description>{{vrfDes}}</description>

{%- endif %}

<afs xmlns="urn:huawei:yang:huawei-l3vpn">

<af>

<type>ipv4-unicast</type>

<route-distinguisher>{{ l3vpn.rd }}</route-distinguisher>

<tunnel-policy>LDP</tunnel-policy>

<vpn-targets>

<vpn-target>

<value>{{ l3vpn.rt }}</value>

<type>export-extcommunity</type>

</vpn-target>

<vpn-target>

<value>{{ l3vpn.rt }}</value>

<type>import-extcommunity</type>

</vpn-target>

</vpn-targets>

</af>

</afs>

<bgp xmlns="urn:huawei:yang:huawei-bgp">

<base-process>

<afs>

<af>

<type>ipv4uni</type>

<ipv4-unicast>

<import-routes>

<import-route>

<protocol>direct</protocol>

<process-id>0</process-id>

</import-route>

<import-route>

<protocol>static</protocol>

<process-id>0</process-id>

</import-route>

</import-routes>

</ipv4-unicast>

</af>

</afs>

<peers>

<peer>

<address>{{ l3vpn.peerAddress }}</address>

<remote-as>101</remote-as>

</peer>

</peers>

</base-process>

</bgp>

</instance>

</instances>

</network-instance>

<ifm xmlns="urn:huawei:yang:huawei-ifm">

<interfaces>

<interface>

<name>{{ l3vpn.ifName }}</name>

<type>GigabitEthernet</type>

<description>{{ l3vpn.description }}</description>

<vrf-name>{{ l3vpn.vrfName }}</vrf-name>

<ipv4 xmlns="urn:huawei:yang:huawei-ip">

<addresses>

<address>

<ip>{{ l3vpn.ip }}</ip>

<type>main</type>

<mask>{{ l3vpn.subnet }}</mask>

</address>

</addresses>

</ipv4>

</interface>

</interfaces>

</ifm>

#### Code Interpretation

Conditional control statement

{%- if l3vpn.vrfDes %}

<description>{{vrfDes}}</description>

{%- endif %}

**{%- if l3vpn.vrfDes %}** is a conditional control statement in Jinja2, and **{%- endif %}** indicates the end of the conditional control code block. If the **l3vpn.vrfDes** parameter is not empty in the rendered Jinja2 template, **<description>{{vrfDes}}</description>** is added to the final rendering result.

Jinja2 template variables

<name>{{ l3vpn.ifName }}</name>

The sub-interface name is extracted as a template variable. (**{{xxx}}** is the syntax of variables in the Jinja2 template.) Variables are input through the service YANG model. In the service YANG model, **l3vpn** is the name of the **list** object, and **ifName** is the name of the leaf node under **l3vpn**.

module L3VPN {  
 ……  
 augment "/app:applications"{   
 list l3vpn {

...

leaf ifName {

type string {

length "0..255";

}

}

An SSP package can map services to different device models using a Jinja2 template. As shown in the following code, **{%- if l3vpn.deviceName | is\_snd('NE8000M8SPC300\_SND') %}** enables the NE8000 M8 running SPC300 to use **l3vpn/NE8000M8.j2** as the southbound Jinja2 template. Other device models can be added as required. In this example, the NE8000 M8 running different versions use the same L3VPN configuration model. Therefore, the same Jinja2 template can be used.

<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">

<nes>

<ne>

<neid>{{l3vpn.deviceName| to\_ne\_id}}</neid>

{%- if l3vpn.deviceName | is\_snd('NE8000M8SPC300\_SND') %}

{% include 'l3vpn/NE8000M8.j2' %}

{% elif l3vpn.deviceName | is\_snd('NE8000M8\_SND') %}

{% include 'l3vpn/NE8000M8.j2' %}

</ne>

</nes>

</inventory-cfg>

Filter

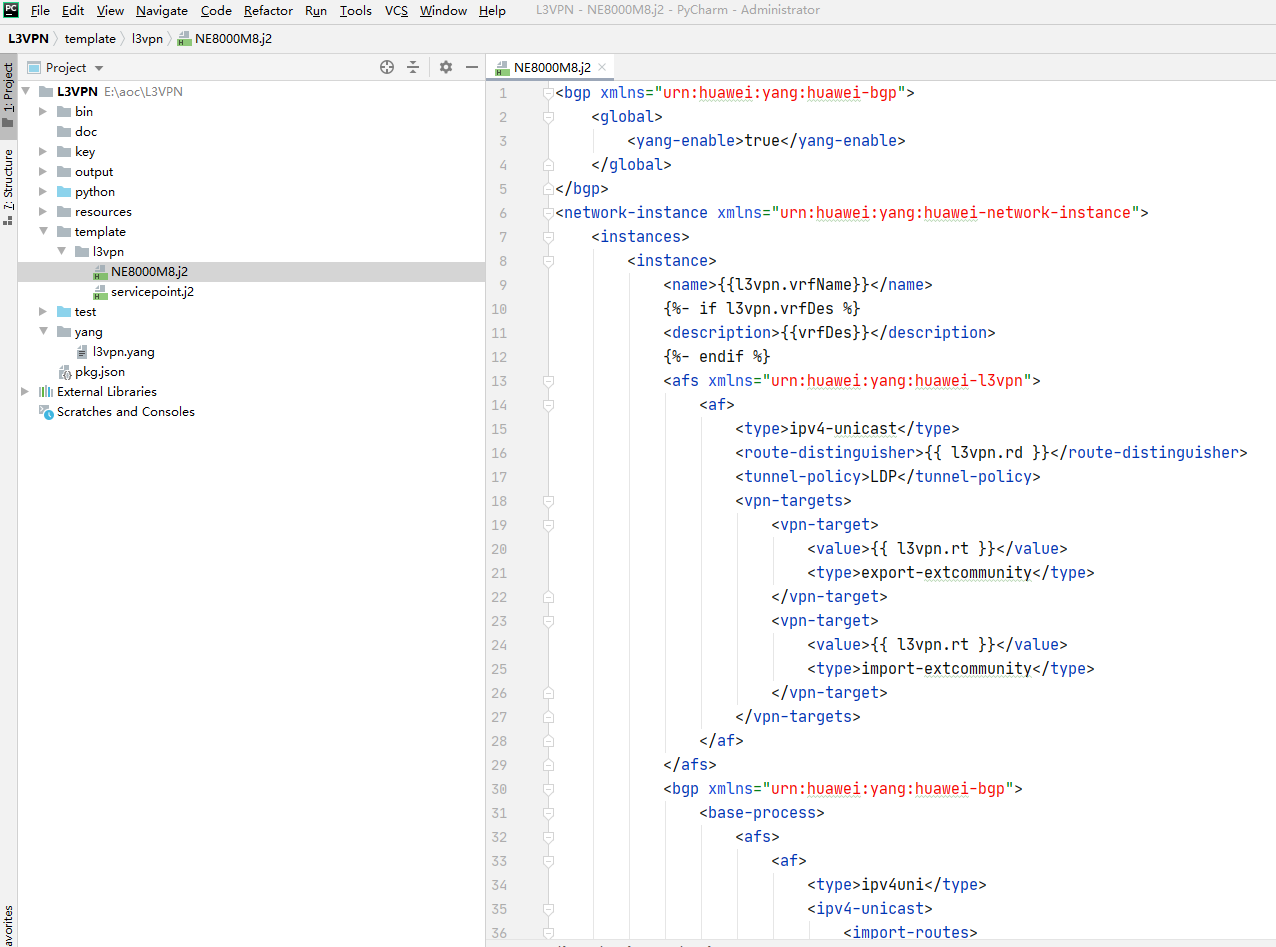
<neid>{{l3vpn.deviceName| to\_ne\_id}}</neid>

**to\_ne\_id** is a filter that converts a device name into a device ID that can be identified by the system. The device name is displayed externally, and the device ID is used to perform operations within the system.

{% include 'l3vpn/NE8000M8.j2' %}

This part of code is used to invoke the customized southbound template. This template is used to set parameters of **ifm**, **network-instance**, and **bgp**, such as **name** (interface name), **ip** (interface IPv4 address), and **mask** (interface mask). You can directly invoke this part of code without modification.

The following figure shows the code in PyCharm.



### Compiling Python Mapping Code

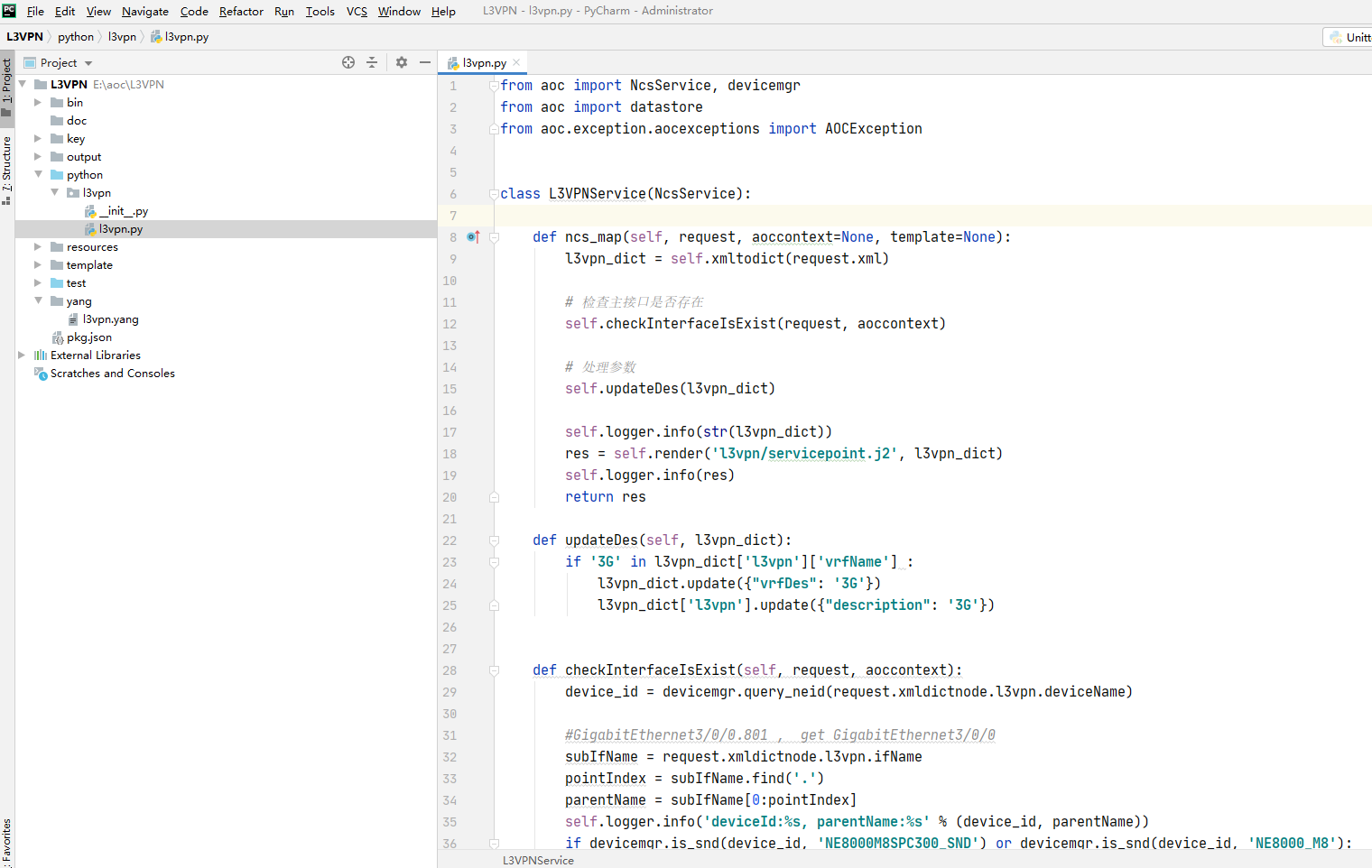
The Python mapping code is used to compile service processing logic and describes how to fill the service data defined in the service YANG model into the input required for Jinja2 template rendering and then map the service data to a southbound NE template through the framework mechanism. Before rendering a southbound template, you can further process rendering parameters based on the input of the service YANG model, perform verification before service delivery, and update rendering parameters.

#### Complete Code

Check whether there is a main interface and update the VRF description before sub-interfaces are delivered.

from aoc import NcsService, devicemgrfrom aoc import NcsService, devicemgr  
from aoc import datastore  
from aoc.exception.aocexceptions import AOCException  
  
  
class L3VPNService(NcsService):  
  
 def ncs\_map(self, request, aoccontext=None, template=None):  
 l3vpn\_dict = self.xmltodict(request.xml)  
  
 *# Check whether a main interface exists.* self.checkInterfaceIsExist(request, aoccontext)  
  
 *# Parameter settings* self.updateDes(l3vpn\_dict)  
  
 self.logger.info(str(l3vpn\_dict))  
 res = self.render(**'l3vpn/servicepoint.j2'**, l3vpn\_dict)  
 self.logger.info(res)  
 return res  
  
 def updateDes(self, l3vpn\_dict):  
 if **'3G'** in l3vpn\_dict[**'l3vpn'**][**'vrfName'**] :  
 l3vpn\_dict.update({**"vrfDes"**: **'3G'**})  
 l3vpn\_dict[**'l3vpn'**].update({**"description"**: **'3G'**})  
  
  
 def checkInterfaceIsExist(self, request, aoccontext):  
 device\_id = devicemgr.query\_neid(request.xmldictnode.l3vpn.deviceName)  
  
 *#GigabitEthernet3/0/0.801 , get GigabitEthernet3/0/0* subIfName = request.xmldictnode.l3vpn.ifName  
 pointIndex = subIfName.find(**'.'**)  
 parentName = subIfName[0:pointIndex]  
 self.logger.info(**'deviceId:%s, parentName:%s'** % (device\_id, parentName))  
 if devicemgr.is\_snd(device\_id, **'NE8000\_M8\_SPC300\_SND'**) or devicemgr.is\_snd(device\_id, **'NE8000\_M8\_SND'**):  
 path = **"huawei-ac-nes:inventory-cfg/nes/ne/"** + device\_id + **"/huawei-ifm:ifm/interfaces/"** output = datastore.read\_datastore\_rdb(aoccontext, path)  
 asnindex = str(output).find(parentName)  
 if asnindex == -1:   
 raise AOCException(parentName + **' not exist in device!'**)  
  
 def add\_filters(self):  
 result = {**"is\_snd"**: self.is\_snd}  
 return result  
  
 def is\_snd(self, nename, sndid):  
 neid = devicemgr.query\_neid(nename)  
 return devicemgr.is\_snd(neid, sndid)

The following figure shows the code in PyCharm.



#### Code Interpretation

Import the required header file and the parent class of the SSP package template.

from aoc import NcsService, devicemgr

from aoc import datastore

from aoc.exception.aocexceptions import AOCException

NcsService is a base class provided by iMaster NCE. You need to override its interfaces to map service YANG data to Jinja2 templates. The devicemgr SDK API provides some interfaces for querying device information. The datastore SDK API provides interfaces for reading and writing iMaster NCE database data. AOCException is an exception class defined by iMaster NCE, which displays user-defined exceptions on the GUI.

ncs\_map overrides the method in the parent class, checks whether the main interface exists, and updates the VRF description based on the VRF name.

def ncs\_map(self, request, aoccontext=None, template=None):  
 l3vpn\_dict = self.xmltodict(request.xml)  
  
 *# Check whether a main interface exists.* self.checkInterfaceIsExist(request, aoccontext)  
  
 *# Parameter settings* self.updateDes(l3vpn\_dict)  
  
 self.logger.info(str(l3vpn\_dict))  
 res = self.render(**'l3vpn/servicepoint.j2'**, l3vpn\_dict)  
 self.logger.info(res)  
 return res

Define the updateDes(self, l3vpn\_dict) method, which is used to update the VRF description. After the update, **l3vpn\_dict** renders the new value to the Jinja2 template.

def updateDes(self, l3vpn\_dict):

if '3G' in l3vpn\_dict['l3vpn']['vrfName'] :

l3vpn\_dict.update({"vrfDes": '3G'})

l3vpn\_dict['l3vpn'].update({"description": '3G'})

Define the checkInterfaceIsExist(self, request, aoccontext) method to check whether the main interface exists, the devicemgr.query\_neid() method to query the device ID based on the device name, the parentName = subIfName[0:pointIndex] method to obtain the main interface name, the devicemgr.is\_snd() method to determine the SND package used by the device, and the datastore.read\_datastore\_rdb() method to query data in the iMaster NCE database. The value of **path** is the combination of the obtained device ID and Xpath in the YANG model corresponding to the data to be queried.

def checkInterfaceIsExist(self, request, aoccontext):

device\_id = devicemgr.query\_neid(request.xmldictnode.l3vpn.deviceName)

# GigabitEthernet3/0/0.801 , get GigabitEthernet3/0/0

subIfName = request.xmldictnode.l3vpn.ifName

pointIndex = subIfName.find('.')

parentName = subIfName[0:pointIndex]

self.logger.info('deviceId:%s, parentName:%s' % (device\_id, parentName))

if devicemgr.is\_snd(device\_id, 'NE8000\_M8\_SPC300\_SND') or devicemgr.is\_snd(device\_id, 'NE8000\_M8\_SND'):

path = "huawei-ac-nes:inventory-cfg/nes/ne/" + device\_id + "/huawei-ifm:ifm/interfaces/"

output = datastore.read\_datastore\_rdb(aoccontext, path)

asnindex = str(output).find(parentName)

if asnindex == -1:

raise AOCException(parentName + ' not exist in device!')

Define the add\_filters(self) function to add a Jinja2 template filter. It inherits the parent class and returns the filter to be added. The following code is used to add a filter for determining the corresponding SND package of a device based on the device ID.

def add\_filters(self):

result = {"is\_snd": self.is\_snd}

return result

def is\_snd(self, nename, sndid):

neid = devicemgr.query\_neid(nename)

return devicemgr.is\_snd(neid, sndid)

This section briefly describes how to use SSP package APIs. For details, see the *Development Guide* at https://intl.devzone.huawei.com/community/en/aoc/developGuide.html?mdName=basic-knowledge.md.

### Compiling Test Cases

Compile test cases to test whether the generated configuration packets are correct. The northbound input only needs to be defined for the low level test (LLT) code framework based on the service YANG model and is displayed in XML format.

#### Complete Code

The complete LLT code for this lab is as follows:

Path: **test/l3vpn/test\_l3vpn\_service.py**

import unittest

import sys

from aoc.sys import devicemgr, datastore

from mock import Mock

sys.path.insert(0, "../../python")

from l3vpn.l3vpn import L3VPNService

class Test(unittest.TestCase):

xml = '''

<l3vpn xmlns="http://example.com/l3vpn">

<deviceName>NE8000</deviceName>

<vrfName>4G\_test</vrfName>

<rd>100:1</rd>

<rt>100:11</rt>

<ifName>GigabitEthernet0/5/0.1</ifName>

<description>wireless base</description>

<ip>20.1.2.9</ip>

<subnet>255.255.255.0</subnet>

<peerAddress>2.2.2.2</peerAddress>

</l3vpn>

'''

def mock\_get\_id(self, neName):

if neName == 'NE8000':

return 'mock\_neid:NE8000'

return 'mock\_neid:NE8000'

def mock\_sys\_func(self):

devicemgr.query\_neid = self.mock\_get\_id

# Ethernet0/1/3.101 GigabitEthernet0/5/4.101

mock\_read\_datastore\_rdb = Mock(return\_value='GigabitEthernet0/5/0')

datastore.read\_datastore\_rdb = mock\_read\_datastore\_rdb

mock\_write\_datastore = Mock(return\_value='success')

datastore.write\_datastore = mock\_write\_datastore

devicemgr.is\_snd = self.mock\_is\_snd

def mock\_is\_snd(self, nename, sndid):

if nename == 'mock\_neid:NE8000' and sndid == 'NE8000M8\_SND':

return True

if nename == 'mock\_neid:NE8000' and sndid == 'NE8000M8SPC300\_SND':

return True

return False

def setUp(self):

self.mock\_sys\_func()

def test\_l3vpn(self):

result = L3VPNService().ncs\_map\_test(self.xml)

print(result)

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

#### Code Interpretation

Simulate the input of the SSP package, which is generated based on the service YANG model.

Go to the directory where the service YANG model is located and run the **java -jar .\yang-offline-util.jar generateSubtree .** command.

If no command output is displayed, the execution is successful. Open the **subtree.xml** file generated in the same directory to obtain the input of the SSP package. The tool generates only the structure, and you need to fill in data.

xml = '''

<l3vpn xmlns="http://example.com/l3vpn">

<deviceName>NE8000</deviceName>

<vrfName>4G\_test</vrfName>

<rd>100:1</rd>

<rt>100:11</rt>

<ifName>GigabitEthernet0/5/0.1</ifName>

<description>wireless base</description>

<ip>20.1.2.9</ip>

<subnet>255.255.255.0</subnet>

<peerAddress>2.2.2.2</peerAddress>

</l3vpn>

'''

Method for starting the test code:

def test\_l3vpn(self):

result = L3VPNService().ncs\_map\_test(self.xml)

print(result)

Before the test code is executed, setUp() is invoked to perform some initialization actions.

def setUp(self):

self.mock\_sys\_func()

In a local test, methods that depend on the iMaster NCE environment cannot be invoked. Instead, some methods in the mapping code need to be mocked.

def mock\_sys\_func(self):

devicemgr.query\_neid = self.mock\_get\_id

# Ethernet0/1/3.101 GigabitEthernet0/5/4.101

mock\_read\_datastore\_rdb = Mock(return\_value='GigabitEthernet0/5/0')

datastore.read\_datastore\_rdb = mock\_read\_datastore\_rdb

mock\_write\_datastore = Mock(return\_value='success')

datastore.write\_datastore = mock\_write\_datastore

devicemgr.is\_snd = self.mock\_is\_snd

Replace the datastore.read\_datastore\_rdb() method by the mock\_read\_datastore\_rdb() method.

mock\_read\_datastore\_rdb = Mock(return\_value='GigabitEthernet0/5/0')

datastore.read\_datastore\_rdb = mock\_read\_datastore\_rdb

Replace the datastore.write\_datastore() method by the mock\_write\_datastore() method.

mock\_write\_datastore = Mock(return\_value='success')

datastore.write\_datastore = mock\_write\_datastore

Replace the devicemgr.is\_snd() method by the mock\_is\_snd() method.

def mock\_is\_snd(self, nename, sndid):

if nename == 'mock\_neid:NE8000' and sndid == 'NE8000M8\_SND':

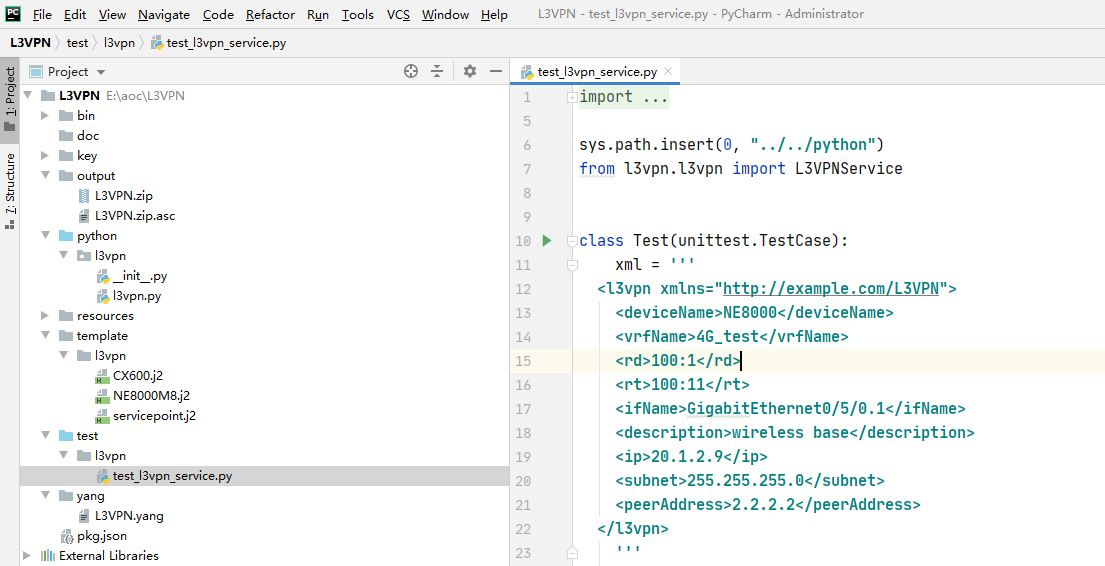
return True

if nename == 'mock\_neid:NE8000' and sndid == 'NE8000M8SPC300\_SND':

return True

return False

The following figure shows the code in the IDE.



Run the test script to check whether the output meets the expectation based on the generated packets.

<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">

<nes>

<ne>

<neid>mock\_neid:NE8000</neid>

<bgp xmlns="urn:huawei:yang:huawei-bgp">

<global>

<yang-enable>true</yang-enable>

</global>

</bgp>

<network-instance xmlns="urn:huawei:yang:huawei-network-instance">

<instances>

<instance>

<name>4G\_test</name>

<afs xmlns="urn:huawei:yang:huawei-l3vpn">

<af>

<type>ipv4-unicast</type>

<route-distinguisher>100:1</route-distinguisher>

<tunnel-policy>LDP</tunnel-policy>

<vpn-targets>

<vpn-target>

<value>100:11</value>

<type>export-extcommunity</type>

</vpn-target>

<vpn-target>

<value>100:11</value>

<type>import-extcommunity</type>

</vpn-target>

</vpn-targets>

</af>

</afs>

<bgp xmlns="urn:huawei:yang:huawei-bgp">

<base-process>

<afs>

<af>

<type>ipv4uni</type>

<ipv4-unicast>

<import-routes>

<import-route>

<protocol>direct</protocol>

<process-id>0</process-id>

</import-route>

<import-route>

<protocol>static</protocol>

<process-id>0</process-id>

</import-route>

</import-routes>

</ipv4-unicast>

</af>

</afs>

<peers>

<peer>

<address>2.2.2.2</address>

<remote-as>100</remote-as>

</peer>

</peers>

</base-process>

</bgp>

</instance>

</instances>

</network-instance>

<ifm xmlns="urn:huawei:yang:huawei-ifm">

<interfaces>

<interface>

<name>GigabitEthernet0/5/0.1</name>

<type>GigabitEthernet</type>

<description>wireless base</description>

<vrf-name>4G\_test</vrf-name>

<ipv4 xmlns="urn:huawei:yang:huawei-ip">

<addresses>

<address>

<ip>20.1.2.9</ip>

<type>main</type>

<mask>255.255.255.0</mask>

</address>

</addresses>

</ipv4>

</interface>

</interfaces>

</ifm>

</ne>

</nes>

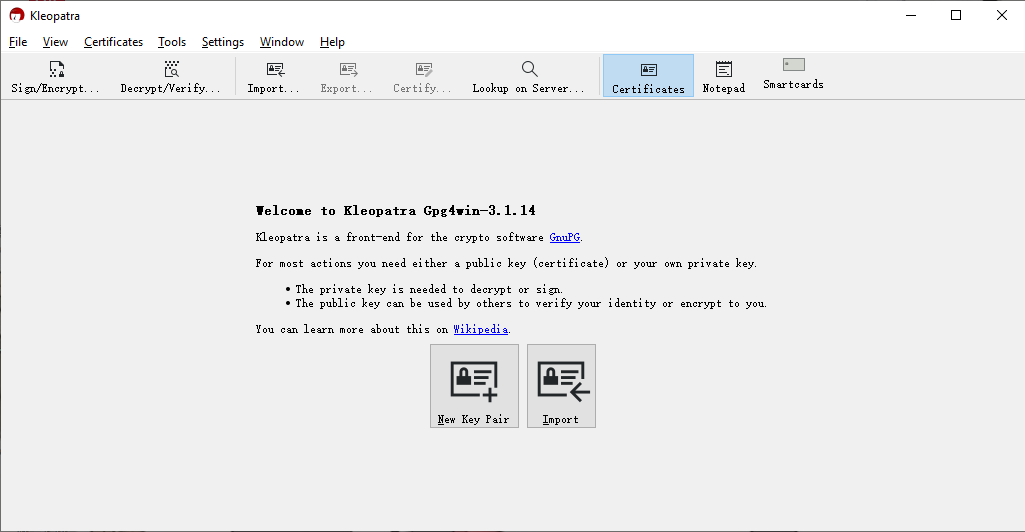
</inventory-cfg>

### Configuring Keys

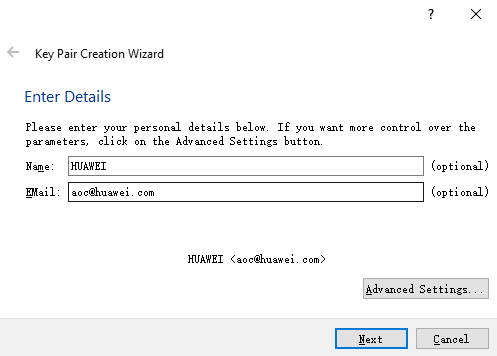
Use Gpg4win to generate a public key and a private key. Save the private key to the SSP package and upload the public key to iMaster NCE for encryption and authentication.

Generate the public key and private key.

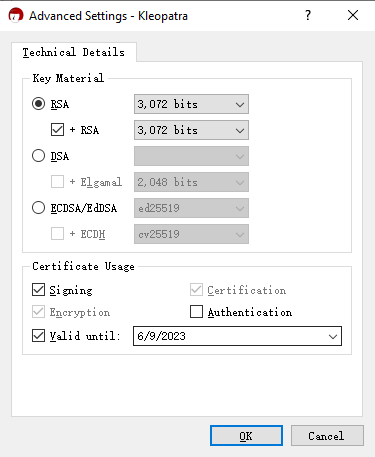
Start the Gpg4win key generation tool and click **New Key Pair**.



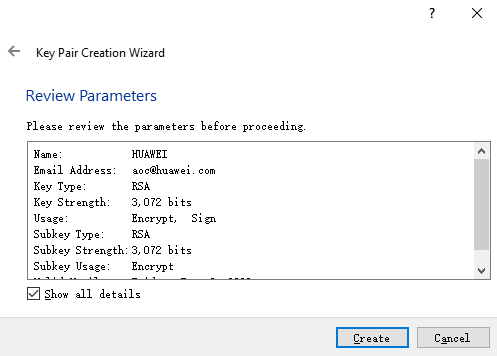
In the **Key Pair Creation Wizard** dialog box, enter the name and email address.



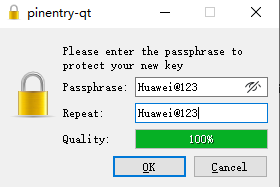
Click **Advanced Settings**. In the dialog box that is displayed, select **3072 bits (default)** and click **OK**.



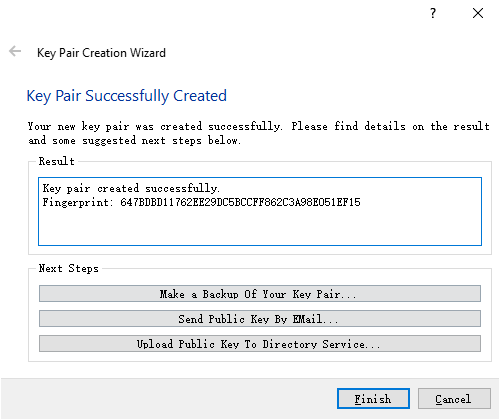
Click **Next**, select **Show all details**, and click **Create**.



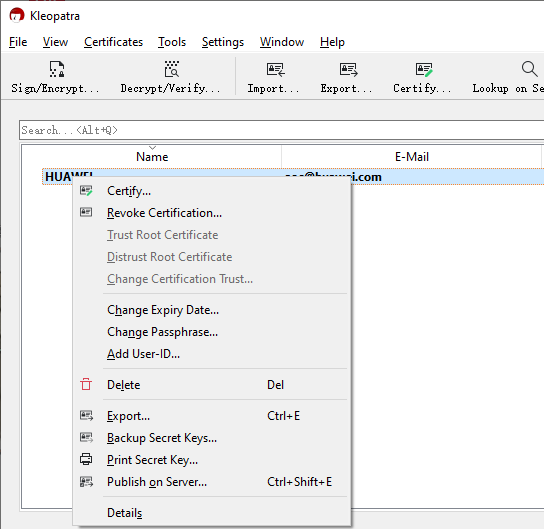
In the dialog box that is displayed, set the password, for example, **Huawei@123**, and click **OK**.



After the key is created, click **Make a Backup Of Your Key Pair** to export the key (named **private.asc**). When exporting the key, you need to enter the password for verification.

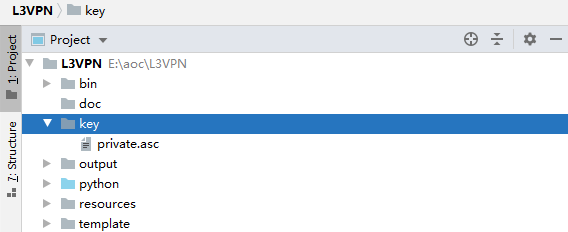


On the homepage, right-click the public key file and choose **Export** to export the public key file named **public.asc**. Then upload the file to the AOC for authentication.



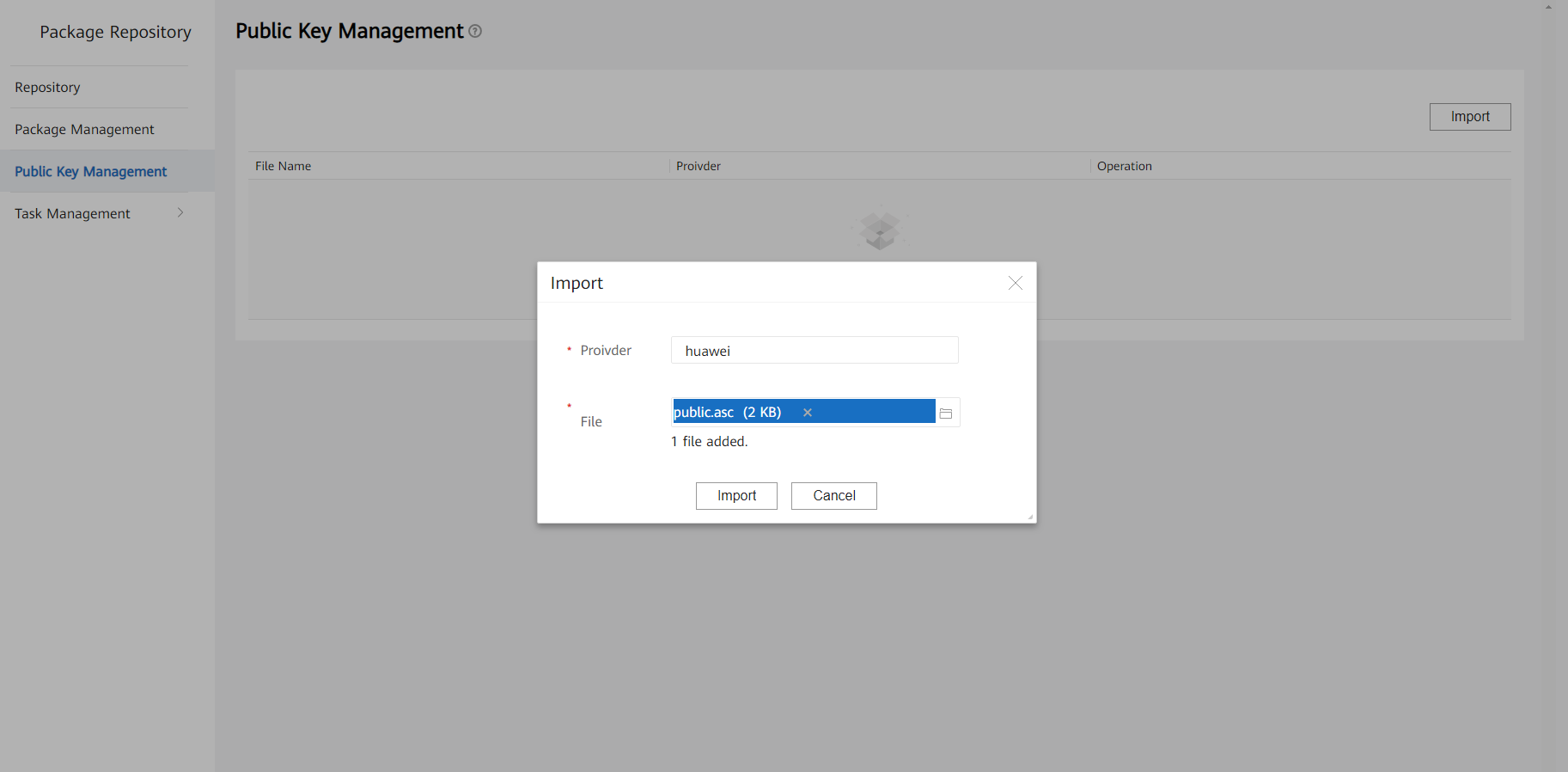
Save the private key to the local PC.

Copy the exported key file **private.asc** to the **key** directory of the L3VPN SSP package template.



Upload the public key file.

Choose **Package** **Repo** > **Publickey** **Manage** in the AOC and import the public key.



### Generating an SSP Package

Open the **cmd** window, go to the **bin** directory of the L3VPN SSP package template, run the **makeFile.bat** script, and enter the configured key password, for example, **Huawei@123**.

E:\aoc\L3VPN\bin>makeFile.bat

Please input password for private key:

...

2021-02-03 09:28:26,369 INFO [com.huawei.cloudsop.extended.pkg.mgr.tools.common.FileUtil] - [Sign]Generate signature file success.

2021-02-03 09:28:26,369 INFO [com.huawei.cloudsop.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign] Sign: Execute success

If the command output contains **success**, the SSP package is generated in the **output** directory.

Switch to the **output** directory of the L3VPN SSP package template and check the generated SSP package and signature file.

E:\aoc\L3VPN\output>dir

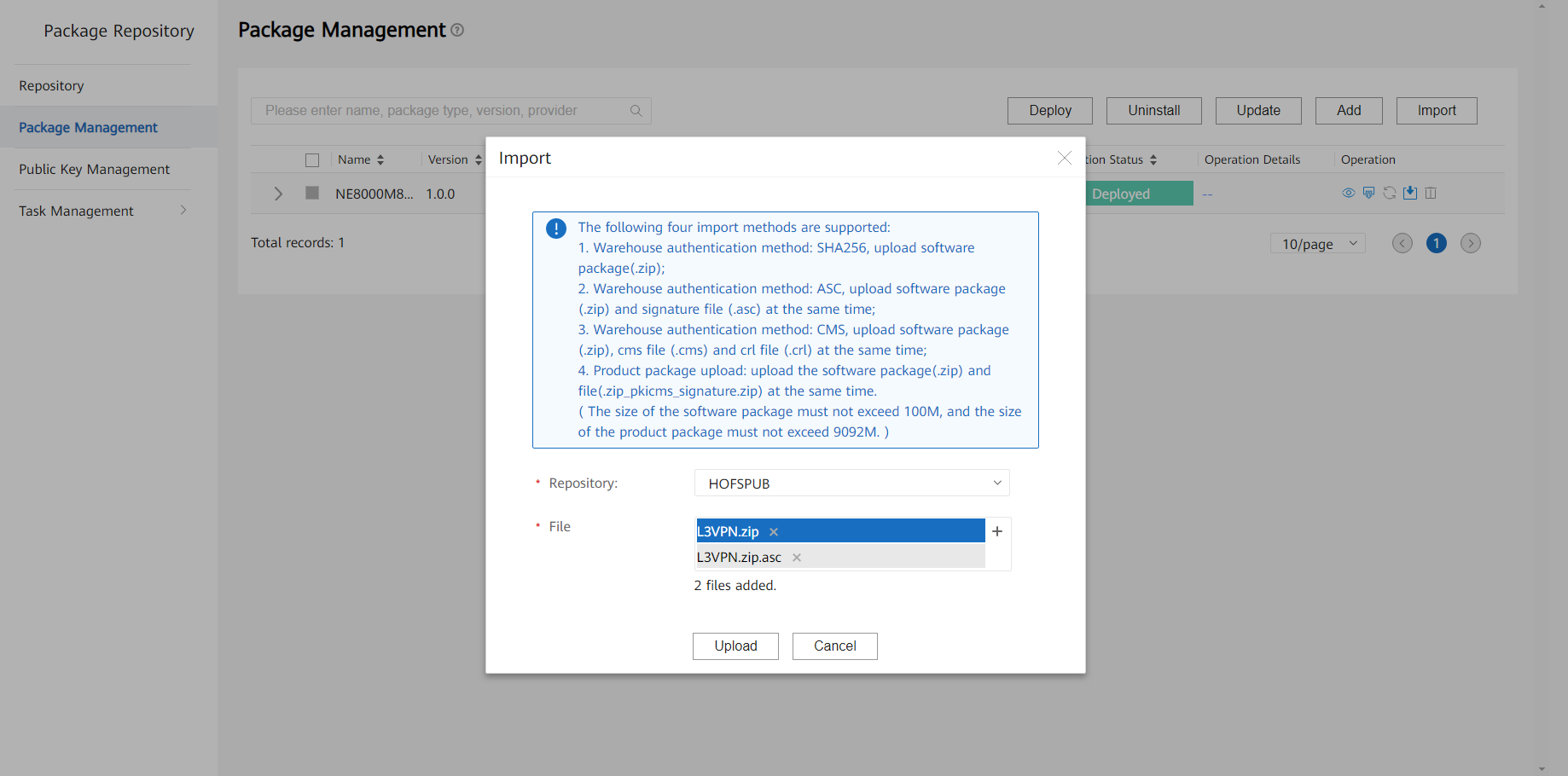
2021/02/04 11:22 <DIR> .

2021/02/04 11:22 <DIR> ..

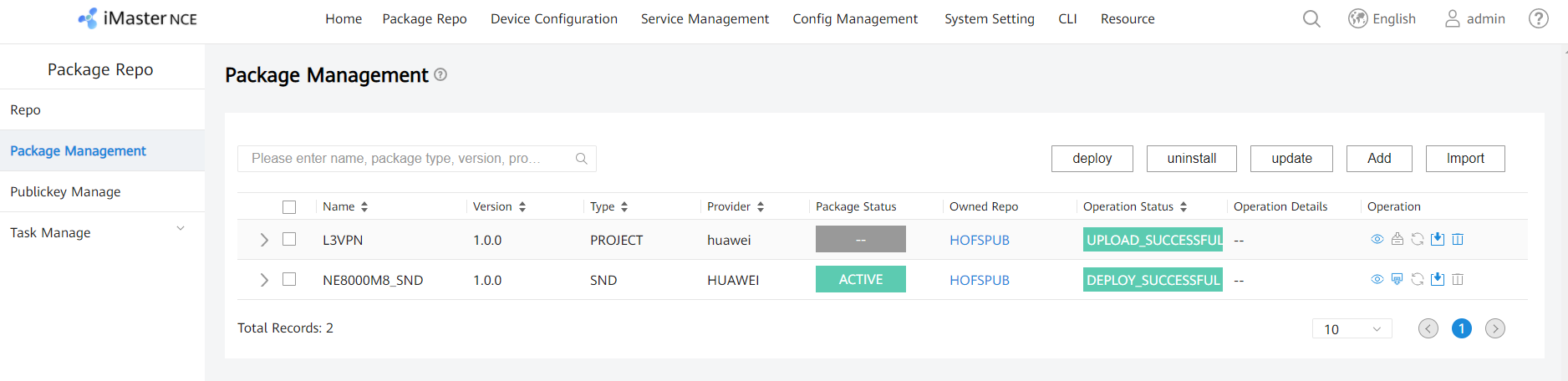
2021/02/04 11:22 4,714 L3VPN.zip

2021/02/04 11:22 664 L3VPN.zip.asc

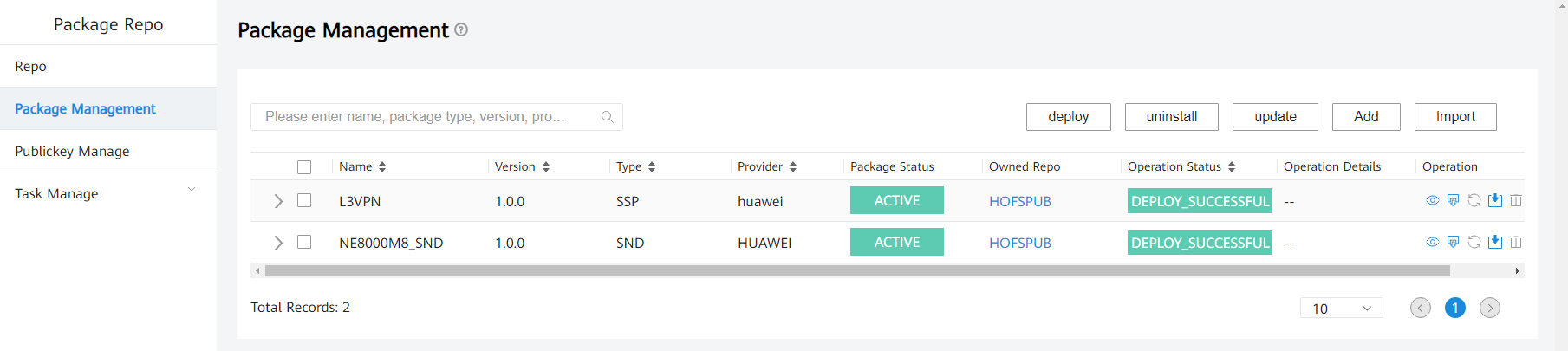
On iMaster NCE, choose **Package** **Repo** > **Package Management** to import the SSP package.



Click **Upload**. The SSP package is imported successfully.



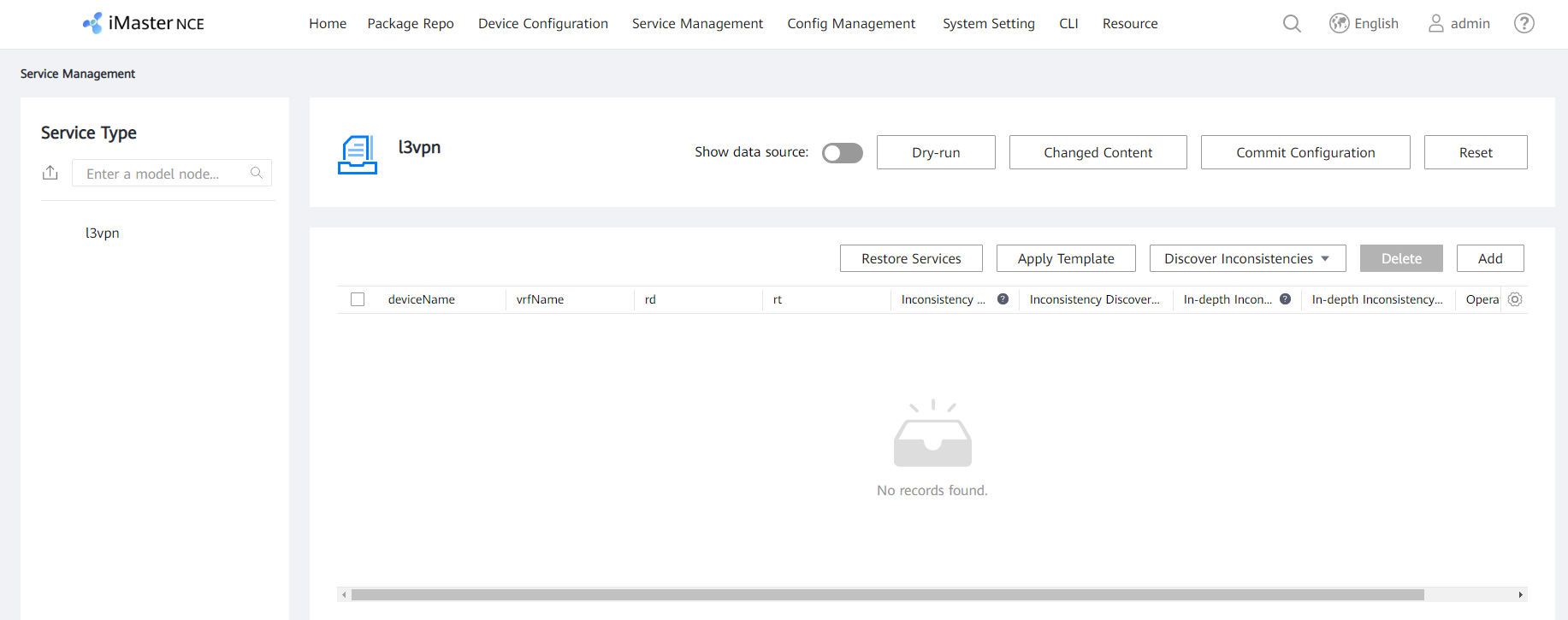
Click C:\Users\w00438901\AppData\Roaming\eSpace_Desktop\UserData\w00438901\imagefiles\045AB7EB-5957-4908-A804-8516B6F0AC5A.png in the **Operation** column to activate the SSP package.



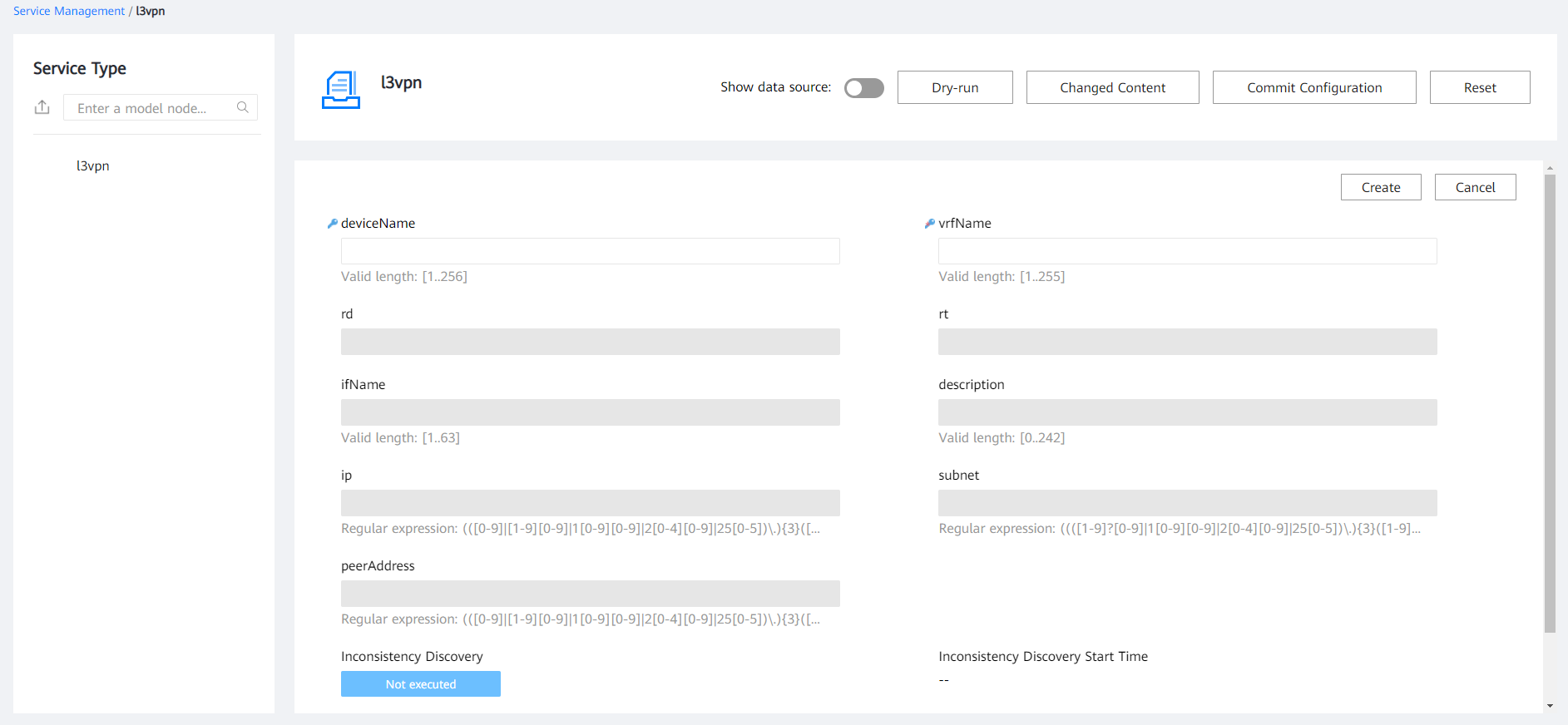
## Delivering Network Services on the GUI

### Delivering a Network Service

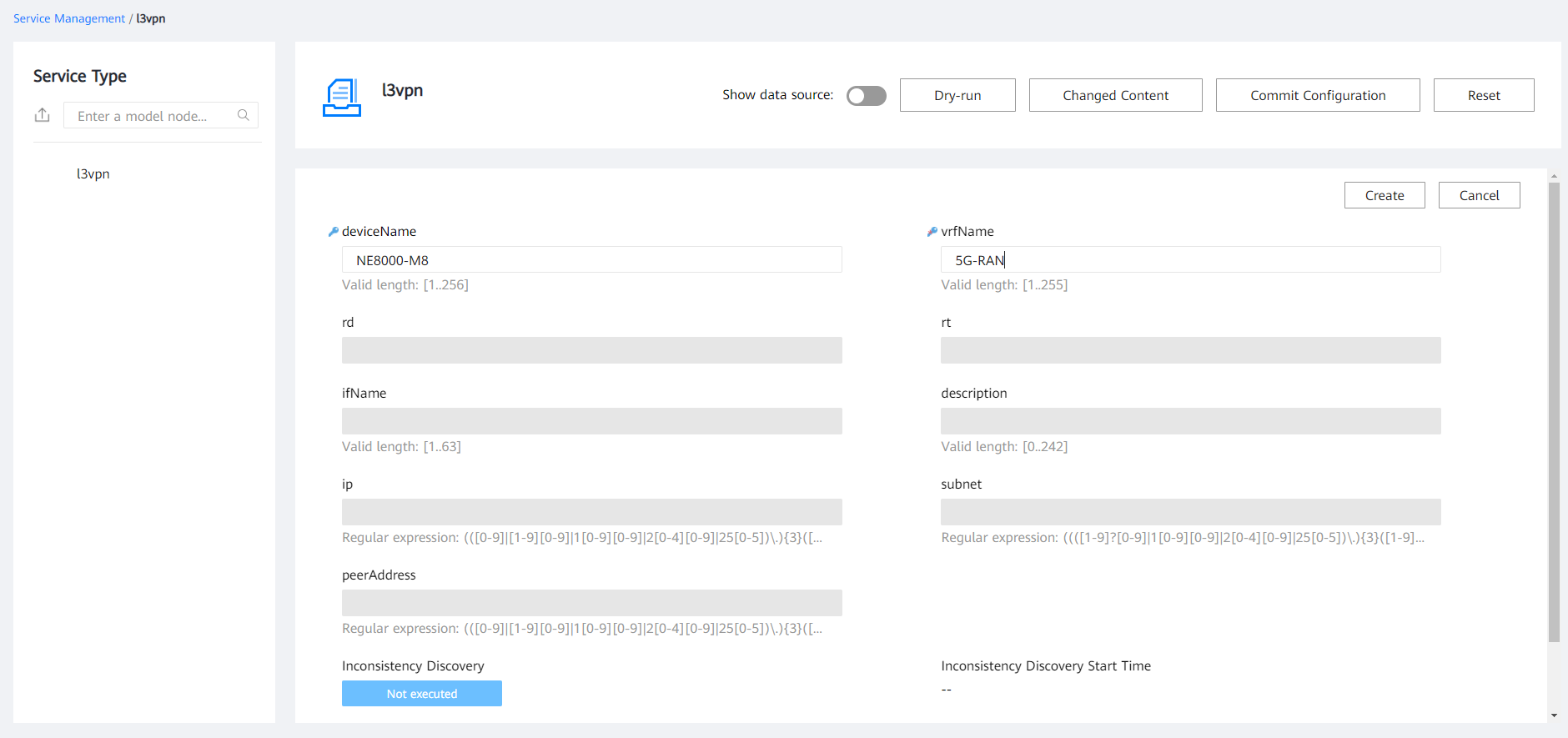
Log in to iMaster NCE and click **Service Management**.



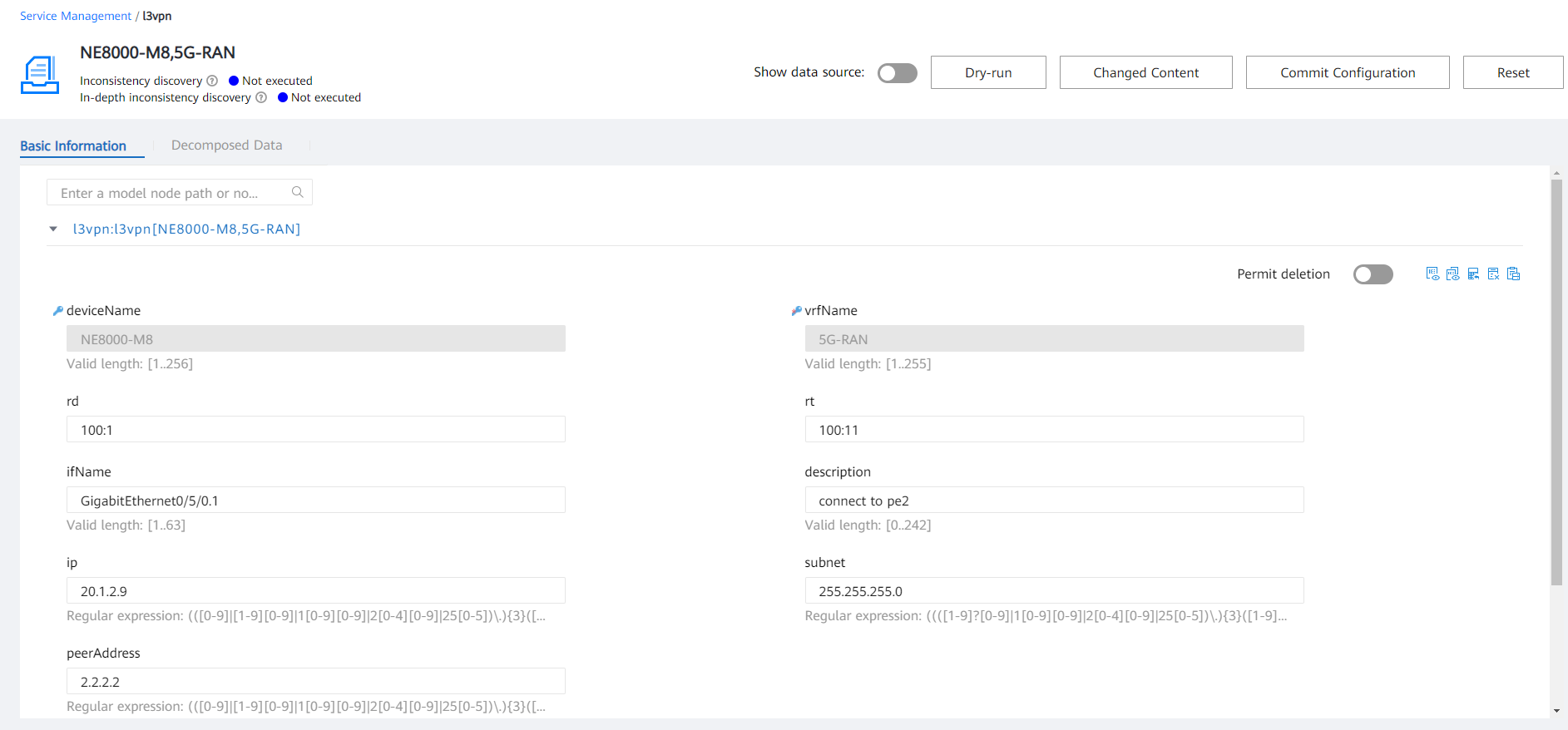
On the **Service Management** page, select the **l3vpn** service model and click **Add**.



Set **deviceName** to **NE8000-M8** and **vrfName** to **5G-RAN**, and click **Create**.



Set **rd** to **100:1**, **rt** to **100:11**, **ifName** to **GigabitEthernet0/5/0.1**, **description** to **connect to pe2**, **ip** to **20.1.2.9**, **subnet** to **255.255. 255.0**, and **peerAddress** to **2.2.2.2**.



Click **Dry-run** to check the configuration packets to be delivered.



The packet content is as follows:

<bgp xmlns="urn:huawei:yang:huawei-bgp">

    <global>

        <yang-enable>true</yang-enable>

    </global>

</bgp>

<network-instance xmlns="urn:huawei:yang:huawei-network-instance">

    <instances>

        <instance xmlns:ns0="urn:ietf:params:xml:ns:netconf:base:1.0" ns0:operation="merge">

            <name>5G-RAN</name>

            <bgp xmlns="urn:huawei:yang:huawei-bgp">

                <base-process>

                    <afs>

                        <af>

                            <type>ipv4uni</type>

                            <ipv4-unicast>

                                <import-routes>

                                    <import-route>

                                        <protocol>direct</protocol>

                                        <process-id>0</process-id>

                                    </import-route>

                                    <import-route>

                                        <protocol>static</protocol>

                                        <process-id>0</process-id>

                                    </import-route>

                                </import-routes>

                            </ipv4-unicast>

                        </af>

                    </afs>

                    <peers>

                        <peer>

                            <address>2.2.2.2</address>

                            <remote-as>100</remote-as>

                        </peer>

                    </peers>

                </base-process>

            </bgp>

            <afs xmlns="urn:huawei:yang:huawei-l3vpn">

                <af>

                    <type>ipv4-unicast</type>

                    <route-distinguisher>100:1</route-distinguisher>

                    <tunnel-policy>LDP</tunnel-policy>

                    <vpn-targets>

                        <vpn-target>

                            <value>100:11</value>

                            <type>export-extcommunity</type>

                        </vpn-target>

                        <vpn-target>

                            <value>100:11</value>

                            <type>import-extcommunity</type>

                        </vpn-target>

                    </vpn-targets>

                </af>

            </afs>

        </instance>

    </instances>

</network-instance>

<ifm xmlns="urn:huawei:yang:huawei-ifm">

    <interfaces>

        <interface>

            <name>GigabitEthernet0/5/0.1</name>

            <ipv4 xmlns="urn:huawei:yang:huawei-ip">

                <addresses>

                    <address xmlns:ns0="urn:ietf:params:xml:ns:netconf:base:1.0" ns0:operation="merge">

                        <ip>20.1.2.9</ip>

                        <mask>255.255.255.0</mask>

                        <type>main</type>

                    </address>

                </addresses>

            </ipv4>

        </interface>

    </interfaces>

</ifm>

<ifm xmlns="urn:huawei:yang:huawei-ifm">

    <interfaces>

        <interface xmlns:ns0="urn:ietf:params:xml:ns:netconf:base:1.0" ns0:operation="merge">

            <name>GigabitEthernet0/5/0.1</name>

            <type>GigabitEthernet</type>

            <description>connect to pe2</description>

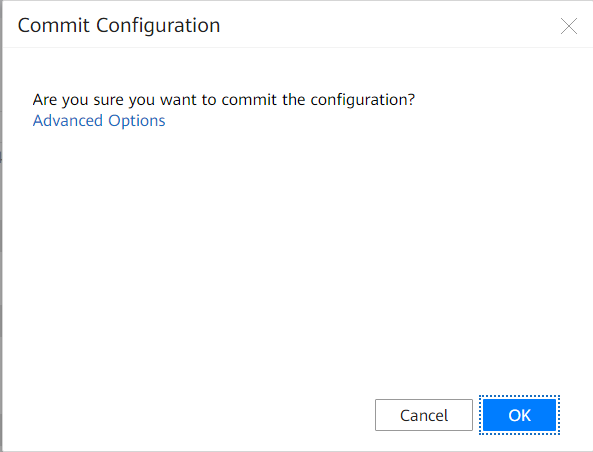
            <vrf-name>5G-RAN</vrf-name>

        </interface>

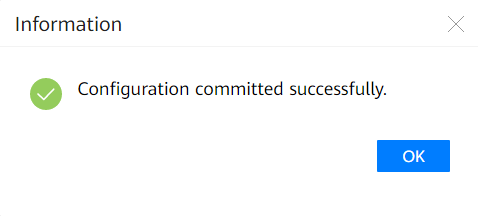
    </interfaces>

</ifm>

Click **Commit Configuration** to deliver the configuration to the device.



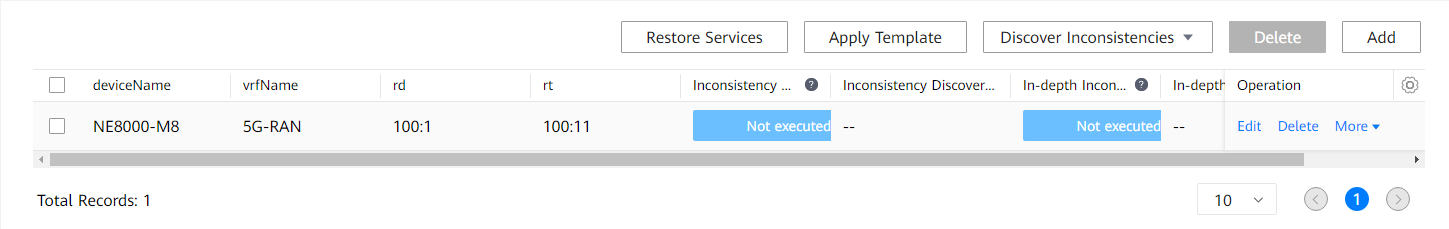
The configuration is delivered successfully.



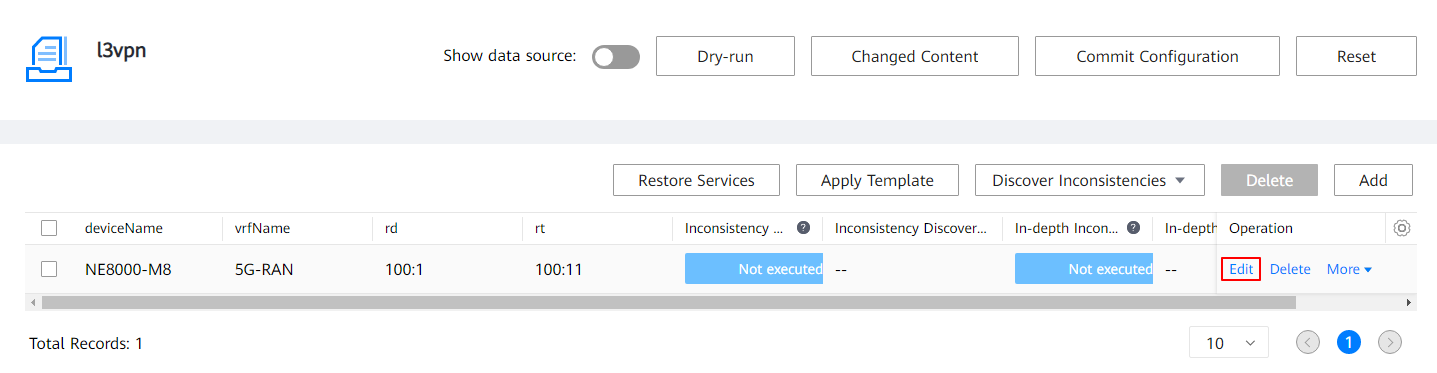
In this way, you successfully customize a network service and deliver the configuration.

### Modifying a Network Service

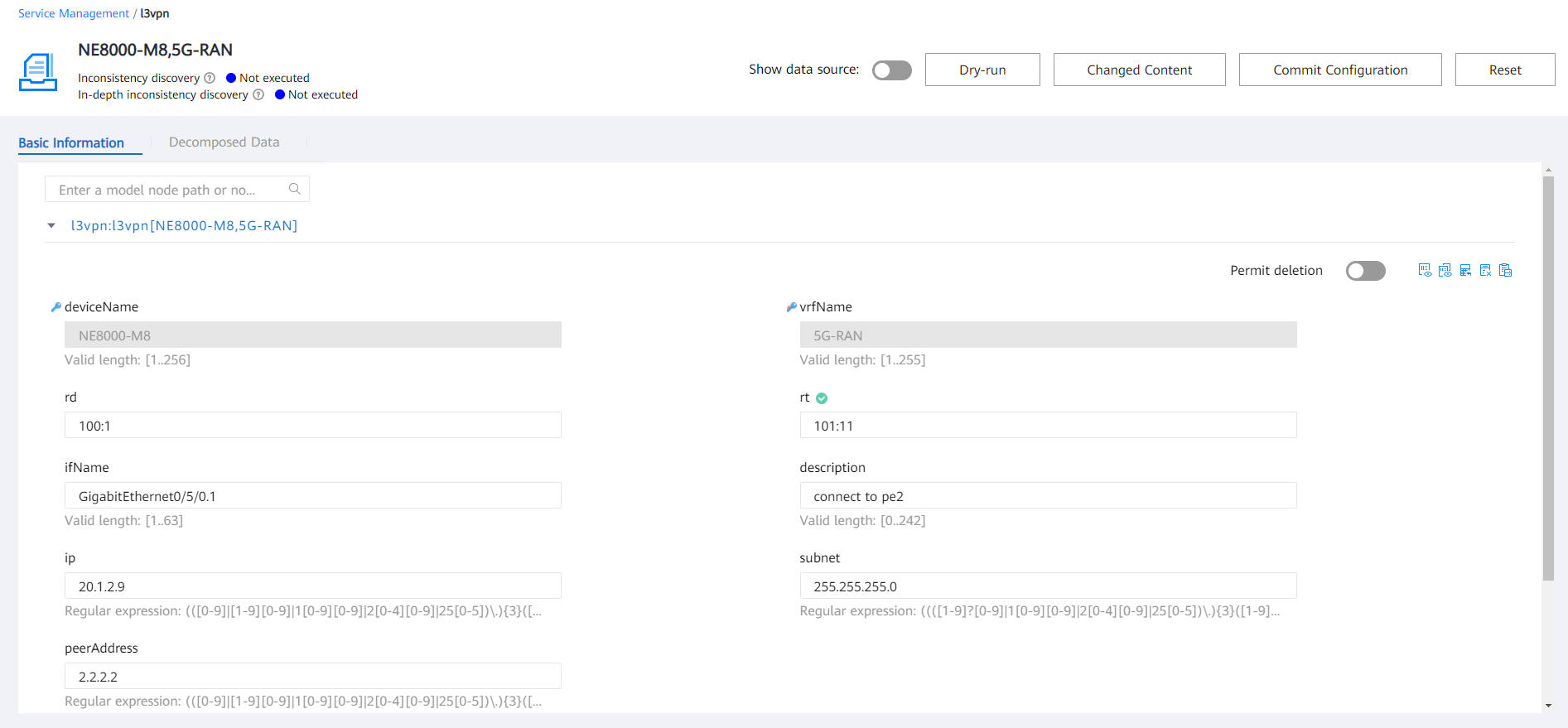
On the **Service Management** page, check the current network service.



Click **Edit**.



Change the RT to **101:11**.



Click **Dry-run** to check the configuration packets to be delivered.



The packet content is as follows:

<bgp xmlns="urn:huawei:yang:huawei-bgp">

<global>

<yang-enable>true</yang-enable>

</global>

</bgp>

<network-instance xmlns="urn:huawei:yang:huawei-network-instance">

<instances>

<instance xmlns:ns0="urn:ietf:params:xml:ns:netconf:base:1.0" ns0:operation="merge">

<name>5G-RAN</name>

<bgp xmlns="urn:huawei:yang:huawei-bgp">

<base-process>

<afs>

<af>

<type>ipv4uni</type>

<ipv4-unicast>

<import-routes>

<import-route>

<protocol>direct</protocol>

<process-id>0</process-id>

</import-route>

<import-route>

<protocol>static</protocol>

<process-id>0</process-id>

</import-route>

</import-routes>

</ipv4-unicast>

</af>

</afs>

<peers>

<peer>

<address>2.2.2.2</address>

<remote-as>100</remote-as>

</peer>

</peers>

</base-process>

</bgp>

<afs xmlns="urn:huawei:yang:huawei-l3vpn">

<af>

<type>ipv4-unicast</type>

<route-distinguisher>100:1</route-distinguisher>

<tunnel-policy>LDP</tunnel-policy>

<vpn-targets>

<vpn-target>

<value>101:11</value>

<type>export-extcommunity</type>

</vpn-target>

<vpn-target>

<value>101:11</value>

<type>import-extcommunity</type>

</vpn-target>

</vpn-targets>

</af>

</afs>

</instance>

</instances>

</network-instance>

<ifm xmlns="urn:huawei:yang:huawei-ifm">

<interfaces>

<interface>

<name>GigabitEthernet0/5/0.1</name>

<ipv4 xmlns="urn:huawei:yang:huawei-ip">

<addresses>

<address xmlns:ns0="urn:ietf:params:xml:ns:netconf:base:1.0" ns0:operation="merge">

<ip>20.1.2.9</ip>

<mask>255.255.255.0</mask>

<type>main</type>

</address>

</addresses>

</ipv4>

</interface>

</interfaces>

</ifm>

<ifm xmlns="urn:huawei:yang:huawei-ifm">

<interfaces>

<interface xmlns:ns0="urn:ietf:params:xml:ns:netconf:base:1.0" ns0:operation="merge">

<name>GigabitEthernet0/5/0.1</name>

<type>GigabitEthernet</type>

<description>connect to pe2</description>

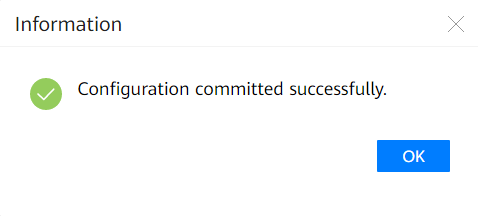
<vrf-name>5G-RAN</vrf-name>

</interface>

</interfaces>

</ifm>

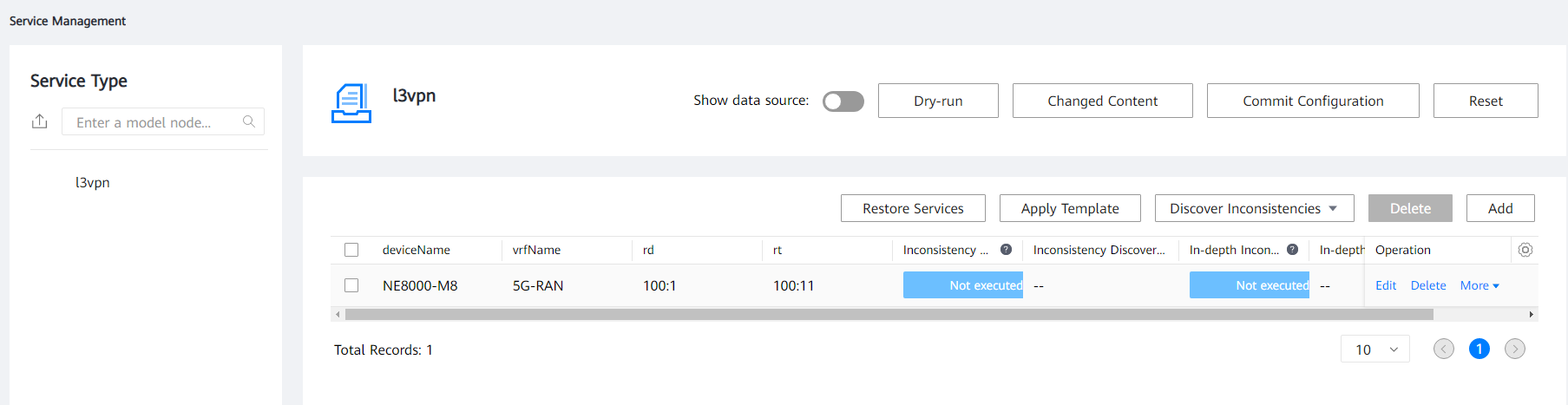
Commit the configuration.



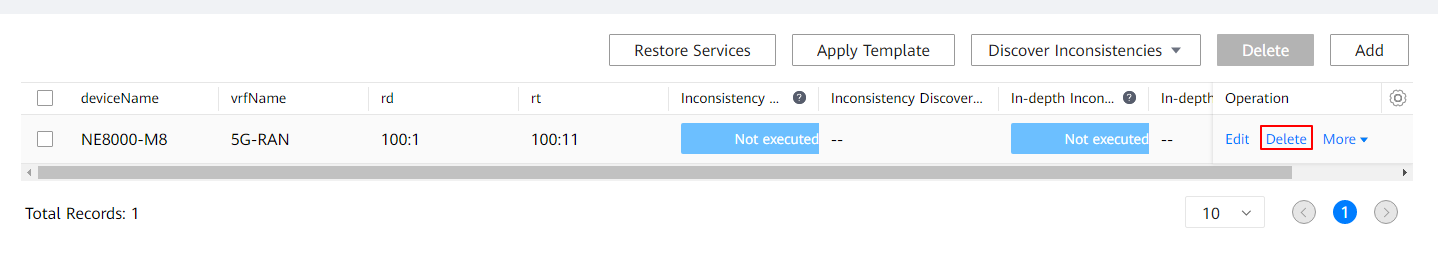
The network service is modified successfully.

### Deleting a Network Service

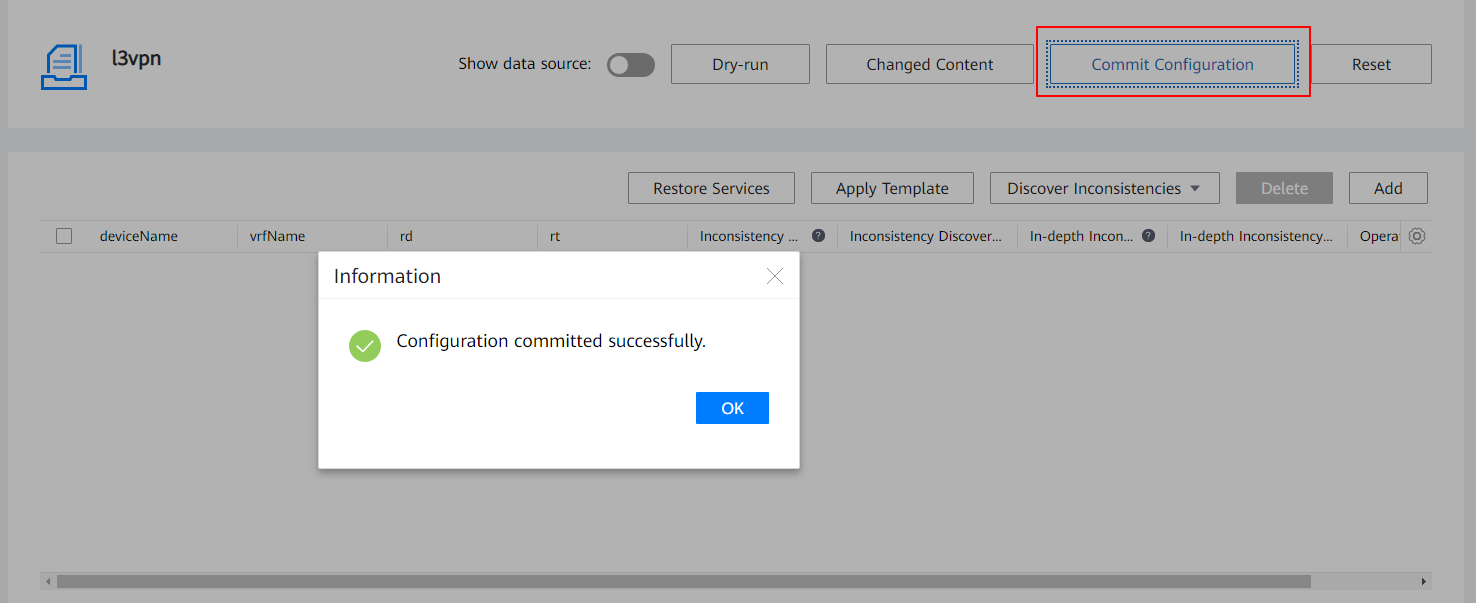
On the **Service Management** page, check the current network service.



Select the service to be deleted and click **Delete**.



Click **Commit Configuration**.



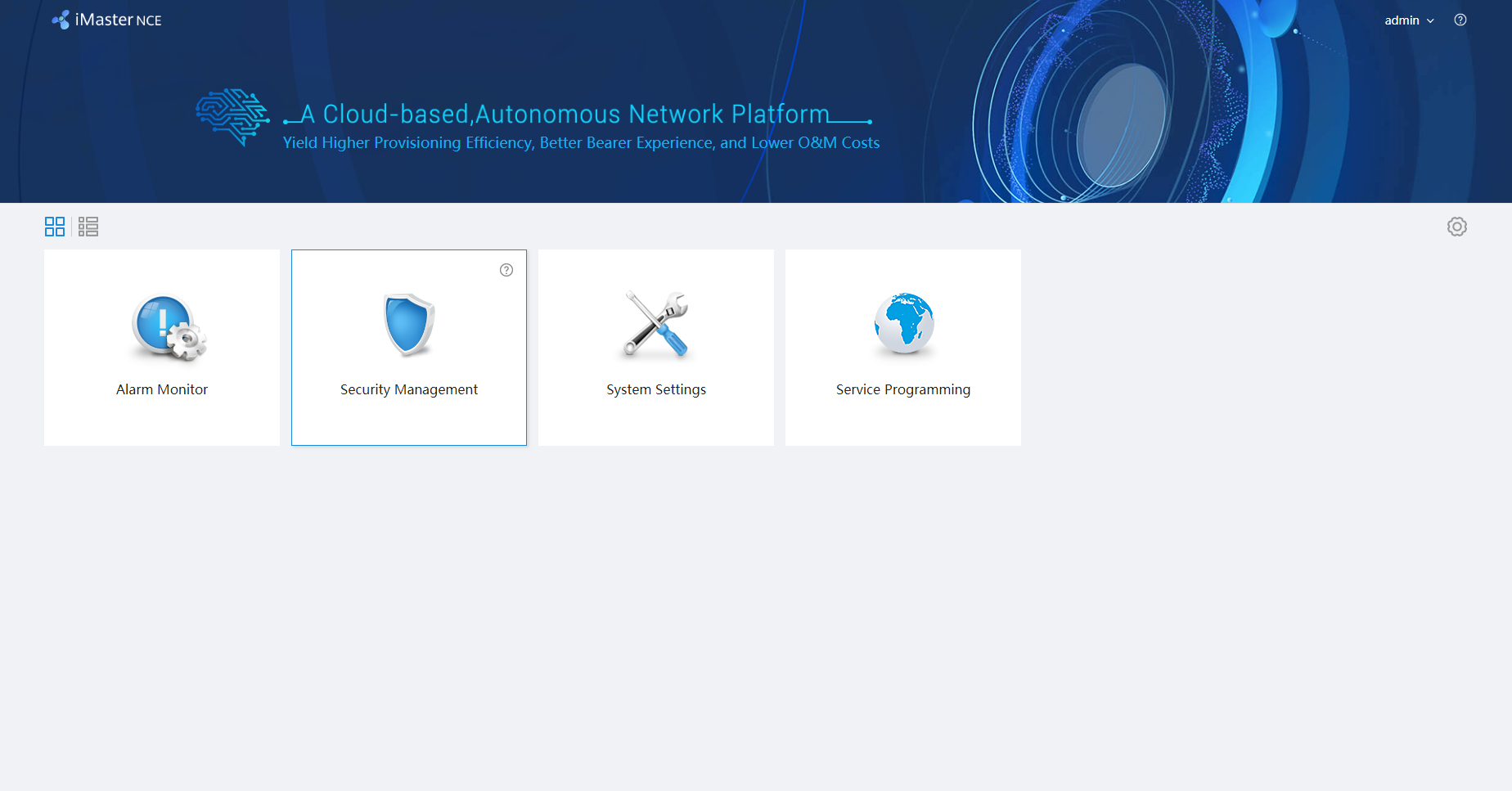
## Delivering Network Services Through the Northbound API

iMaster NCE provides open northbound capabilities based on customized SSP package capabilities. You need to create a northbound user and use the northbound account to implement the add, delete, modify, and query functions. For more iMaster NCE northbound capabilities, see **Document Center** > **API Explorer** > **Service Management API** in the Developer Community.

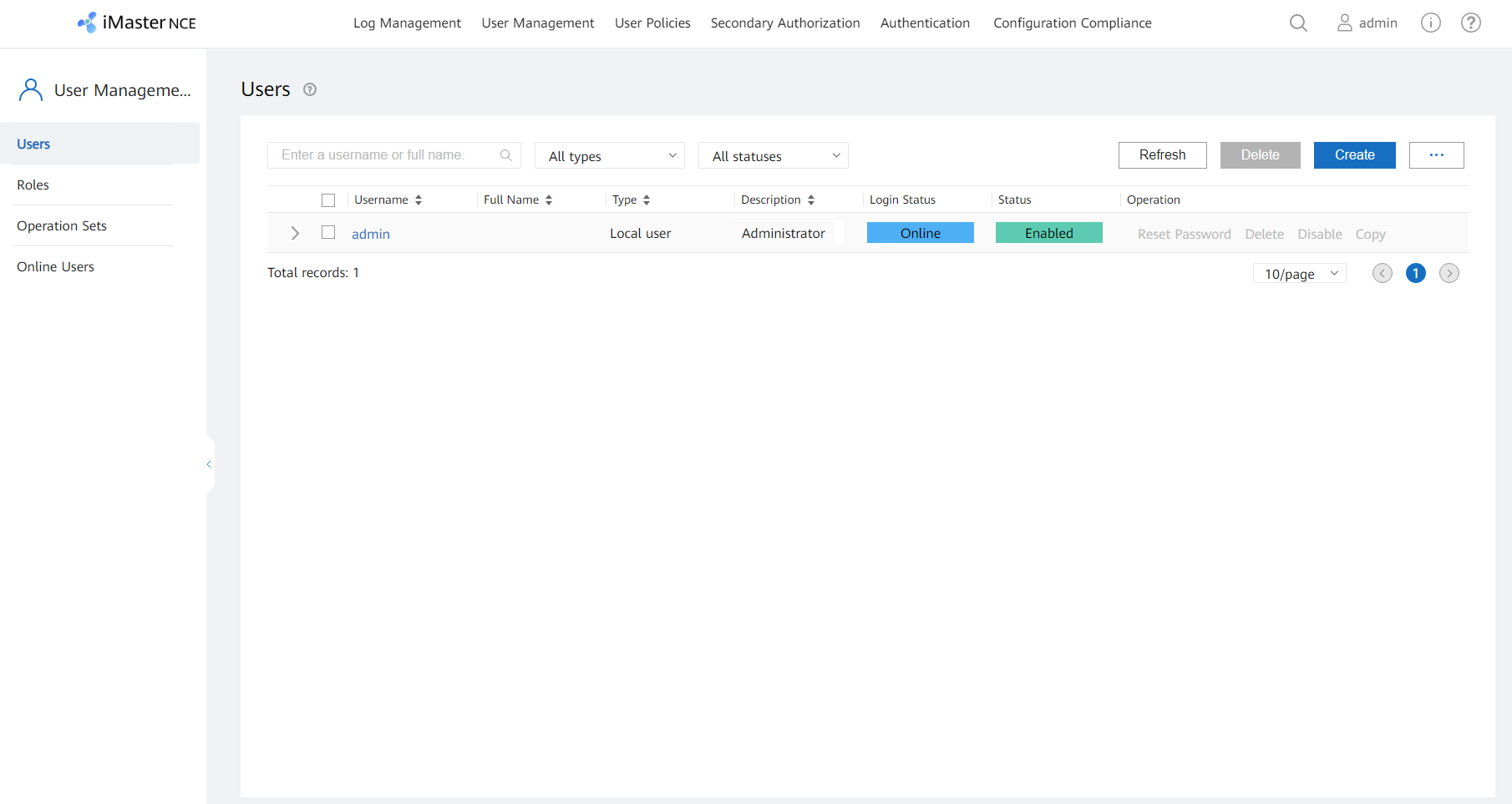
This section describes how to create a northbound user and deliver service configurations through the northbound API.

### Creating a Northbound User

On the iMaster NCE homepage, click **Security Management**.

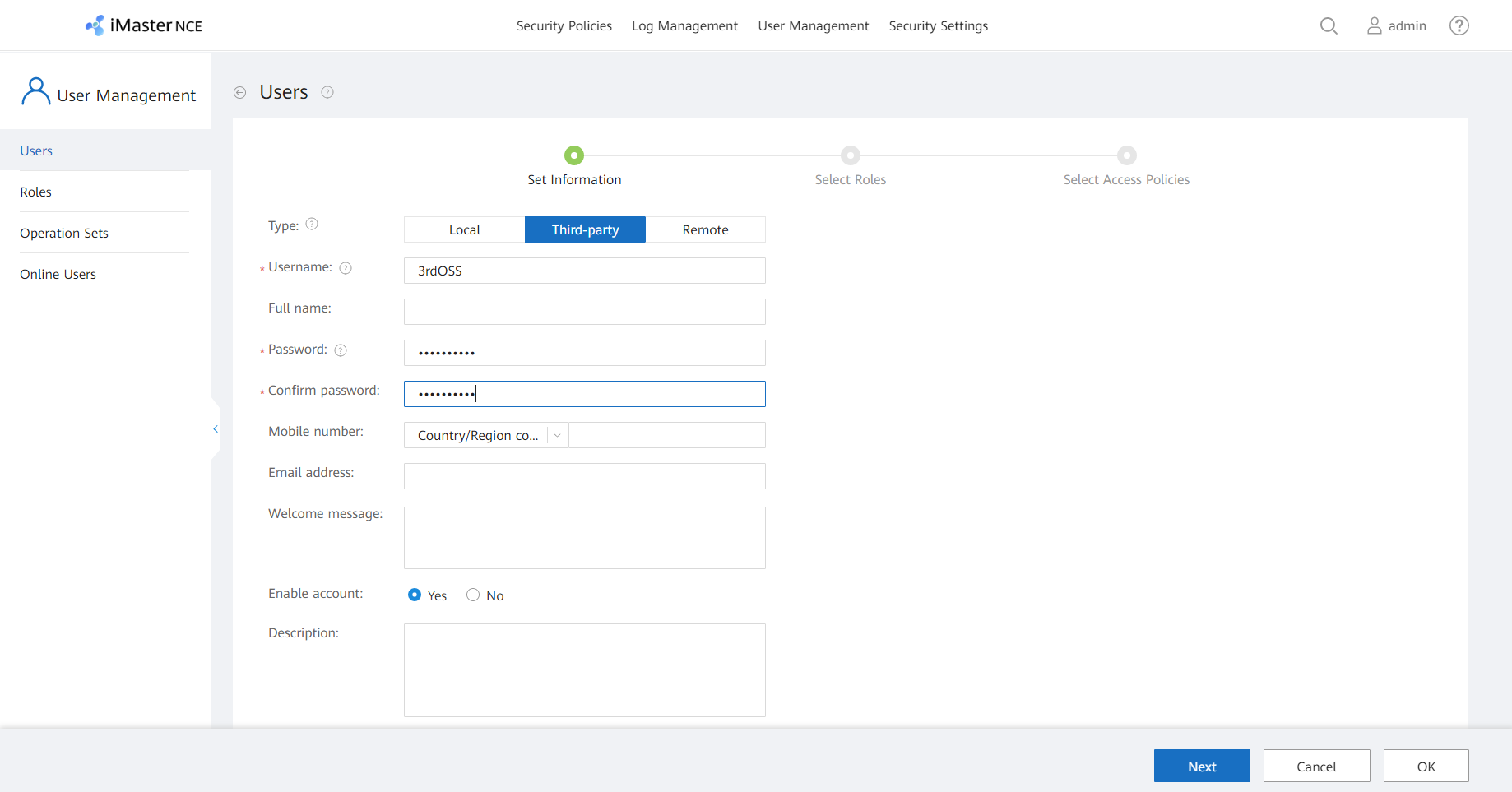


On the **Security Management** page, choose **User Management** and click **Create**.

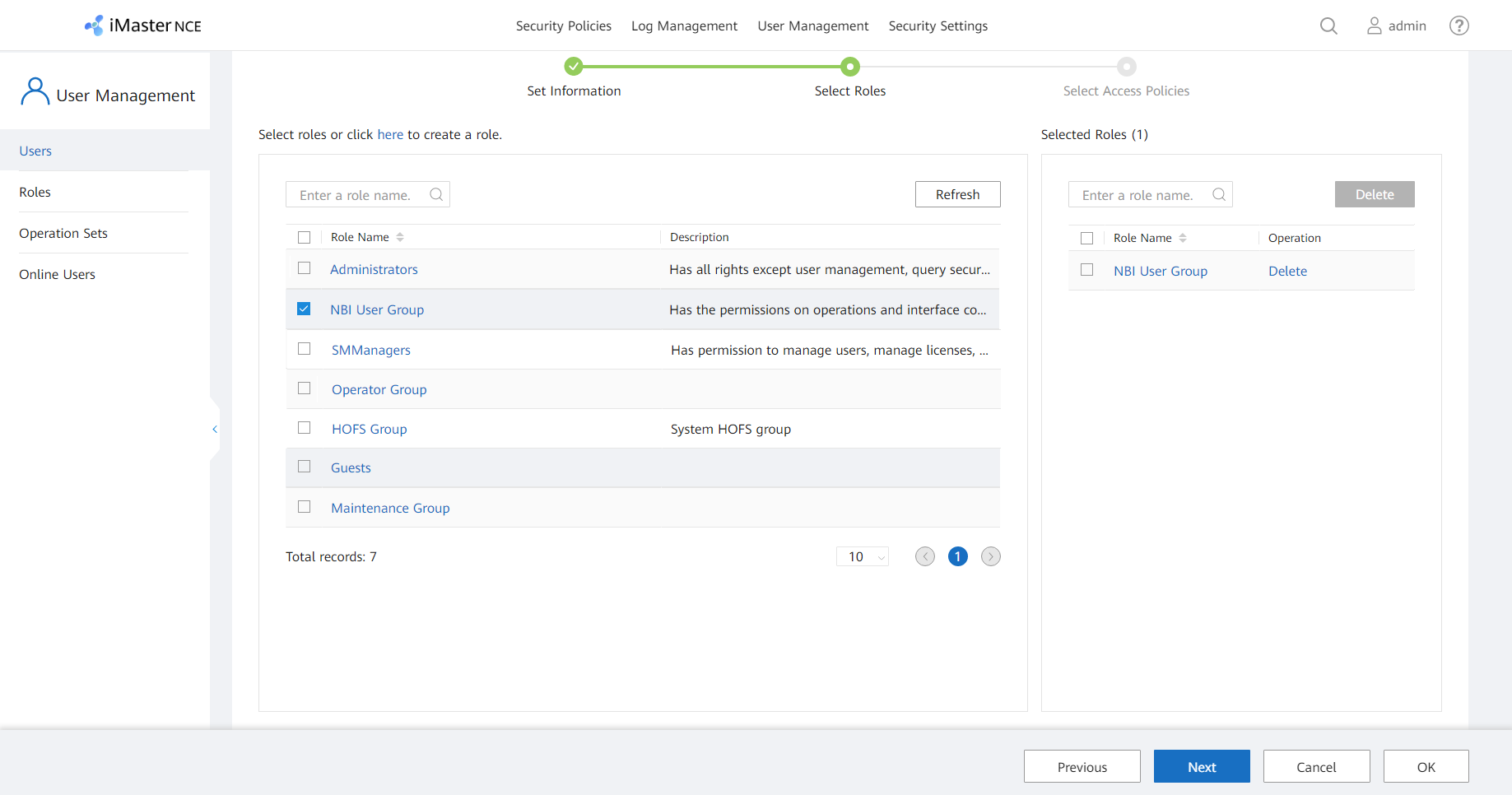


On the page for creating users, select **Third-party**, set the username and password, and click **Next**.

In this example, the username is **3rdOSS** and the password is **Huawei@123**.



Select **Northbound User Group**, click **Next**, and retain the default configuration in the subsequent steps in the wizard.



### Creating Configurations

Obtain the token. In the URL, **northIP** indicates the login IP address.

Method: PUT

https://{{northIP}}:26335/rest/plat/smapp/v1/oauth/token

Request header:

| Key | Value |
| --- | --- |
| Content-Type | application/json |

Request body:

{

"grantType":"password",

"userName":"3rdOSS",

"value":"Huawei@123"

}

Response body:

{

"accessSession": "x-1cpceq1j0bdhft2niqiotg087uak6ps82ptfuqnw2pteqm84ob04o77u3vo4rtlg6mpdle2ktgo9dfml9cmrbzljs5uk2k1f9ho7rw2pen6kqk49obdcamc8jzvs6llj",

"roaRand": "48fa2a40d3250e6beb6a16cd806034574cf4fd9969a5f096",

"expires": 1800,

"additionalInfo": null

}

Obtain the token. Headers of all sent packets must contain the token, which is stored in the **Cookie** field in the headers.

Apply for a transaction.

Method: POST

https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:create

Request header, which contains the token — **accessSession** — generated in step 1:

| Key | Value |
| --- | --- |
| Content-Type | application/json |
| Accept | application/json |
| Cookie | accessSession={{accessSession}} |

Response body:

{

"huawei-ac-restconf-transactions:output": {

"trans-id": "a4ce2a02-5e6f-41af-8824-852ba2155c2b"

}

}

A transaction ID needs to be applied for before a service is edited.

Edit the configuration to be delivered.

Method: POST

https://{{northIP}}:26335/restconf/v1/data/huawei-ac-applications:applications

Request header, which carries the token generated in step 1 and the transaction ID generated in step 2:

| Key | Value |
| --- | --- |
| restconf-transaction-id | {{transactionsID}} |
| Content-Type | application/json |
| Accept | application/json |
| Cookie | accessSession={{accessSession}} |

Request body:

{

    " l3vpn:l3vpn": {

        "deviceName": "NE8000-M8",

        "vrfName": "5G-RAN",

        "ifName": "GigabitEthernet0/5/0.1",

        "subnet": "255.255.255.0",

        "peerAddress": "2.2.2.2",

        "ip": "20.1.2.9",

        "description": "connect to pe2",

        "rt": "100:11",

        "rd": "100:1"

    }

}

Perform a dry run.

Method: POST

https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:dry-run

Request header:

| Key | Value |
| --- | --- |
| Content-Type | application/json |
| Accept | application/json |
| Cookie | accessSession={{accessSession}} |

Request body:

{

"huawei-ac-restconf-transactions:input":{

"trans-id":{{transactionsID}}

}

}

Response body:

{

"huawei-ac-restconf-transactions:output": {

"result": true

}

}

After the service is edited, you can perform a dry run to view the device configurations to be delivered, without delivering the configurations to the devices.

Preview configuration differences.

Method: POST

https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:diff

Request header:

| Key | Value |
| --- | --- |
| Content-Type | application/json |
| Accept | application/json |
| Cookie | accessSession={{accessSession}} |

Request body:

{

"huawei-ac-restconf-transactions:input":{

"trans-id": {{transactionsID}}

}

}

Response body:

{

"huawei-ac-restconf-transactions:output": {

"service-diff-infos": [

{

"feature": "/(urn:huawei:yang:huawei-ac-applications?revision=2018-07-06)applications/AugmentationIdentifier{childNames=[(http://example.com/l3vpn?revision=2020-11-05)l3vpn]}/(http://example.com/l3vpn?revision=2020-11-05)l3vpn/l3vpn[{(http://example.com/l3vpn?revision=2020-11-05)vrfName=5G-RAN, (http://example.com/l3vpn?revision=2020-11-05)deviceName=NE8000-M8}]",

"diff-info": "{\"l3vpn:l3vpn\": [{\"left\": null, \"right\": {\"[deviceName=NE8000-M8,vrfName=5G-RAN]\": {\"deviceName\": \"NE8000-M8\", \"vrfName\": \"5G-RAN\", \"rd\": \"100:1\", \"rt\": \"100:11\", \"ifName\": \"GigabitEthernet0/7/1\", \"description\": \"connect to ce\", \"ip\": \"20.1.2.9\", \"subnet\": \"255.255.255.0\", \"peerAddress\": \"2.2.2.2\"}}}]}"

}

]

}

}

After editing the configuration, you can preview the modified contents of the current transaction to check the network- or NE-layer data differences.

Check the NE configuration after the dry run.

Method: POST

https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:dry-run-query

Request header:

| Key | Value |
| --- | --- |
| Content-Type | application/json |
| Accept | application/json |
| Cookie | accessSession={{accessSession}} |

Request body:

{

"huawei-ac-restconf-applications:input": {

"trans-id": "{{trans-id}}"

}

}

Check the configuration and difference information generated after the dry run.

{

"huawei-ac-restconf-transactions:output": {

"native": {

"dry-run-ne-native-confs-item": [

{

"ne-id": "1fd5468c-8244-11eb-8e45-6e1fce390113",

"ne-name": "NE8000-M8",

"ne-cfg": [

"<bgp xmlns=\"urn:huawei:yang:huawei-bgp\">\n<global>\n<yang-enable>true</yang-enable>\n</global>\n</bgp>\n",

"<network-instance xmlns=\"urn:huawei:yang:huawei-network-instance\">\n<instances>\n<instance xmlns:ns0=\"urn:ietf:params:xml:ns:netconf:base:1.0\" ns0:operation=\"create\">\n<name>5G-RAN</name>\n<bgp xmlns=\"urn:huawei:yang:huawei-bgp\">\n<base-process>\n<afs>\n<af>\n<type>ipv4uni</type>\n<ipv4-unicast>\n<import-routes>\n<import-route>\n<protocol>direct</protocol>\n<process-id>0</process-id>\n</import-route>\n<import-route>\n<protocol>static</protocol>\n<process-id>0</process-id>\n</import-route>\n</import-routes>\n</ipv4-unicast>\n</af>\n</afs>\n<peers>\n<peer>\n<address>2.2.2.2</address>\n<remote-as>100</remote-as>\n</peer>\n</peers>\n</base-process>\n</bgp>\n<afs xmlns=\"urn:huawei:yang:huawei-l3vpn\">\n<af>\n<type>ipv4-unicast</type>\n<route-distinguisher>100:1</route-distinguisher>\n<tunnel-policy>LDP</tunnel-policy>\n<vpn-targets>\n<vpn-target>\n<value>100:11</value>\n<type>export-extcommunity</type>\n</vpn-target>\n<vpn-target>\n<value>100:11</value>\n<type>import-extcommunity</type>\n</vpn-target>\n</vpn-targets>\n</af>\n</afs>\n</instance>\n</instances>\n</network-instance>\n",

"<ifm xmlns=\"urn:huawei:yang:huawei-ifm\">\n<interfaces>\n<interface xmlns:ns0=\"urn:ietf:params:xml:ns:netconf:base:1.0\" ns0:operation=\"create\">\n<name>GigabitEthernet0/7/1</name>\n<type>GigabitEthernet</type>\n<description>connect to ce</description>\n<vrf-name>5G-RAN</vrf-name>\n<ipv4 xmlns=\"urn:huawei:yang:huawei-ip\">\n<addresses>\n<address>\n<ip>20.1.2.9</ip>\n<mask>255.255.255.0</mask>\n<type>main</type>\n</address>\n</addresses>\n</ipv4>\n</interface>\n</interfaces>\n</ifm>\n"

]

}

]

},

"diff": {

"dry-run-ne-diff-confs-item": [

{

"ne-id": "1fd5468c-8244-11eb-8e45-6e1fce390113",

"ne-name": "NE8000-M8",

"ne-cfg-diffs": [

{

"feature": "/(urn:huawei:yang:huawei-bgp?revision=2020-02-25)bgp",

"diff-info": "{\"bgp\": {\"global\": {\"yang-enable\": {\"left\": false, \"right\": true}}}}"

},

{

"feature": "/(urn:huawei:yang:huawei-ifm?revision=2020-02-14)ifm",

"diff-info": "{\"ifm\": {\"interfaces\": {\"interface\": [{\"left\": null, \"right\": {\"[name=GigabitEthernet0/7/1]\": {\"name\": \"GigabitEthernet0/7/1\", \"description\": \"connect to ce\", \"vrf-name\": \"5G-RAN\", \"type\": \"GigabitEthernet\", \"huawei-ip:ipv4\": {\"addresses\": {\"address\": [{\"[ip=20.1.2.9]\": {\"ip\": \"20.1.2.9\", \"mask\": \"255.255.255.0\", \"type\": \"main\"}}]}}}}}]}}}"

},

{

"feature": "/(urn:huawei:yang:huawei-network-instance?revision=2020-02-25)network-instance",

"diff-info": "{\"network-instance\": {\"instances\": {\"instance\": [{\"left\": null, \"right\": {\"[name=5G-RAN]\": {\"name\": \"5G-RAN\", \"huawei-bgp:bgp\": {\"base-process\": {\"afs\": {\"af\": [{\"[type=ipv4uni]\": {\"type\": \"ipv4uni\", \"ipv4-unicast\": {\"import-routes\": {\"import-route\": [{\"[protocol=direct,process-id=0]\": {\"protocol\": \"direct\", \"process-id\": 0}}, {\"[protocol=static,process-id=0]\": {\"protocol\": \"static\", \"process-id\": 0}}]}}}}]}, \"peers\": {\"peer\": [{\"[address=2.2.2.2]\": {\"address\": \"2.2.2.2\", \"remote-as\": \"100\"}}]}}}, \"huawei-l3vpn:afs\": {\"af\": [{\"[type=ipv4-unicast]\": {\"type\": \"ipv4-unicast\", \"vpn-targets\": {\"vpn-target\": [{\"[value=100:11,type=export-extcommunity]\": {\"value\": \"100:11\", \"type\": \"export-extcommunity\"}}, {\"[value=100:11,type=import-extcommunity]\": {\"value\": \"100:11\", \"type\": \"import-extcommunity\"}}]}, \"route-distinguisher\": \"100:1\", \"tunnel-policy\": \"LDP\"}}]}}}}]}}}"

}

]

}

]

},

"mapconf": {

"service\_map\_confs": [

{

"source": "/huawei-ac-applications:applications/l3vpn:l3vpn/NE8000-M8/5G-RAN",

"dry-run-map-confs-item": [

{

"ne-id": "1fd5468c-8244-11eb-8e45-6e1fce390113",

"ne-name": "NE8000-M8",

"ne-cfg": [

"<bgp xmlns=\"urn:huawei:yang:huawei-bgp\">\n<global>\n<yang-enable>true</yang-enable>\n</global>\n</bgp>\n",

"<network-instance xmlns=\"urn:huawei:yang:huawei-network-instance\">\n<instances>\n<instance>\n<name>5G-RAN</name>\n<bgp xmlns=\"urn:huawei:yang:huawei-bgp\">\n<base-process>\n<afs>\n<af>\n<type>ipv4uni</type>\n<ipv4-unicast>\n<import-routes>\n<import-route>\n<protocol>direct</protocol>\n<process-id>0</process-id>\n</import-route>\n<import-route>\n<protocol>static</protocol>\n<process-id>0</process-id>\n</import-route>\n</import-routes>\n</ipv4-unicast>\n</af>\n</afs>\n<peers>\n<peer>\n<address>2.2.2.2</address>\n<remote-as>100</remote-as>\n</peer>\n</peers>\n</base-process>\n</bgp>\n<afs xmlns=\"urn:huawei:yang:huawei-l3vpn\">\n<af>\n<type>ipv4-unicast</type>\n<vpn-targets>\n<vpn-target>\n<value>100:11</value>\n<type>export-extcommunity</type>\n</vpn-target>\n<vpn-target>\n<value>100:11</value>\n<type>import-extcommunity</type>\n</vpn-target>\n</vpn-targets>\n<route-distinguisher>100:1</route-distinguisher>\n<tunnel-policy>LDP</tunnel-policy>\n</af>\n</afs>\n</instance>\n</instances>\n</network-instance>\n",

"<ifm xmlns=\"urn:huawei:yang:huawei-ifm\">\n<interfaces>\n<interface>\n<name>GigabitEthernet0/7/1</name>\n<description>connect to ce</description>\n<vrf-name>5G-RAN</vrf-name>\n<type>GigabitEthernet</type>\n<ipv4 xmlns=\"urn:huawei:yang:huawei-ip\">\n<addresses>\n<address>\n<ip>20.1.2.9</ip>\n<mask>255.255.255.0</mask>\n<type>main</type>\n</address>\n</addresses>\n</ipv4>\n</interface>\n</interfaces>\n</ifm>\n"

]

}

]

}

]

}

}

}

Submit a transaction.

Method: POST

https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:commit

Request header:

| Key | Value |
| --- | --- |
| Content-Type | application/json |
| Accept | application/json |
| Cookie | accessSession={{accessSession}} |

Request body:

{

"huawei-ac-restconf-transactions:input": {

"trans-id": "{{trans-id}}",

“no-network”: true,

"force": false,

"service-parallel": true

}

}

Response body:

{

"huawei-ac-restconf-transactions:output": {

"result": true

}

}

The modified data needs to be submitted to the controller for the data to take effect after the service is modified.

### Checking Configurations

Obtain a token.

Method: PUT

https://{{northIP}}:26335/rest/plat/smapp/v1/oauth/token

Request header:

| Key | Value |
| --- | --- |
| Content-Type | application/json |
| Accept | application/json |

Request body:

{

"grantType":"password",

"userName":"3rdOSS",

"value":"Huawei@123"

}

Response body:

{ "accessSession": "x-1cpceq1j0bdhft2niqiotg087uak6ps82ptfuqnw2pteqm84ob04o77u3vo4rtlg6mpdle2ktgo9dfml9cmrbzljs5uk2k1f9ho7rw2pen6kqk49obdcamc8jzvs6llj",

"roaRand": "48fa2a40d3250e6beb6a16cd806034574cf4fd9969a5f096",

"expires": 1800,

"additionalInfo": null

}

**accessSession** in the response packet is the token. Headers of all sent packets need to carry the token for authentication.

Check the configurations.

Method: GET

https://{{northIP}}:26335/restconf/v1/data/huawei-ac-applications:applications/ L3VPN:l3vpn

Request header:

| Key | Value |
| --- | --- |
| Accept | application/json |
| Cookie | accessSession={{accessSession}} |
| Content-Type | application/json |

Response body:

{

" l3vpn:l3vpn": [

{

"deviceName": "NE8000-M8",

"vrfName": "5G-RAN",

"ifName": "GigabitEthernet0/5/0.1",

"subnet": "255.255.255.0",

"peerAddress": "2.2.2.2",

"ip": "20.1.2.9",

"description": "connect to pe2",

"rt": "100:11",

"rd": "100:1"

}

]

}

### Modifying Configurations

Obtain a token.

Method: PUT

https://{{northIP}}:26335/rest/plat/smapp/v1/oauth/token

Request header:

| Key | Value |
| --- | --- |
| Content-Type | application/json |
| Accept | application/json |

Request body:

{

"grantType":"password",

"userName":"3rdOSS",

"value":"Huawei@123"

}

Response body:

{ "accessSession": "x-1cpceq1j0bdhft2niqiotg087uak6ps82ptfuqnw2pteqm84ob04o77u3vo4rtlg6mpdle2ktgo9dfml9cmrbzljs5uk2k1f9ho7rw2pen6kqk49obdcamc8jzvs6llj",

"roaRand": "48fa2a40d3250e6beb6a16cd806034574cf4fd9969a5f096",

"expires": 1800,

"additionalInfo": null

}

**accessSession** in the response packet is the token. Headers of all sent packets need to carry the token for authentication.

Apply for a transaction.

Method: POST

https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:create

Request header, which contains the token generated in step 1:

| Key | Value |
| --- | --- |
| Content-Type | application/json |
| Accept | application/json |
| Cookie | accessSession={{accessSession}} |

Response body, in which **trans-id** indicates the ID of the new transaction:

{

"huawei-ac-restconf-transactions:output": {

"trans-id": "a4ce2a02-5e6f-41af-8824-852ba2155c2b"

}

}

Modify the configuration. Change the value of **description** to **test modify description**.

Method: PUT

https://{{northIP}}:26335/restconf/v1/data/huawei-ac-applications:applications/ l3vpn:l3vpn

Request header:

| Key | Value |
| --- | --- |
| restconf-transaction-id | {{transactionsID}} |
| Accept | application/json |
| Cookie | accessSession={{accessSession}} |
| Content-Type | application/json |

Request body:

{

" l3vpn:l3vpn": [

{

"deviceName": "NE8000-M8",

"vrfName": "5G-RAN",

"ifName": "GigabitEthernet0/5/0.1",

"subnet": "255.255.255.0",

"peerAddress": "2.2.2.2",

"ip": "20.1.2.9",

"description": "test modify description",

"rt": "100:11",

"rd": "100:1"

}

]

}

Perform a dry run.

Method: POST

https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:dry-run

Request header:

| Key | Value |
| --- | --- |
| restconf-transaction-id | {{transactionsID}} |
| Accept | application/json |
| Cookie | accessSession={{accessSession}} |
| Content-Type | application/json |

Request body:

{

"huawei-ac-restconf-transactions:input":{

"trans-id":{{transactionsID}}

}

}

Response body:

{

"huawei-ac-restconf-transactions:output": {

"result": true

}

}

Preview configuration differences.

Method: POST

https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:diff

Request header:

| Key | Value |
| --- | --- |
| Content-Type | application/json |
| Accept | application/json |
| Cookie | accessSession={{accessSession}} |

Request body:

{

"huawei-ac-restconf-transactions:input":{

"trans-id": {{transactionsID}}

}

}

Response body:

{

"huawei-ac-restconf-transactions:output": {

"service-diff-infos": [

{

"feature": "/(urn:huawei:yang:huawei-ac-applications?revision=2018-07-06)applications/AugmentationIdentifier{childNames=[(http://example.com/l3vpn?revision=2020-11-05)l3vpn]}/(http://example.com/l3vpn?revision=2020-11-05)l3vpn/l3vpn[{(http://example.com/l3vpn?revision=2020-11-05)vrfName=5G-RAN, (http://example.com/l3vpn?revision=2020-11-05)deviceName=NE8000-M8}]",

"diff-info": "{\"l3vpn\": [{\"[deviceName=NE8000-M8,vrfName=5G-RAN]\": {\"deviceName\": \"NE8000-M8\", \"vrfName\": \"5G-RAN\", \"ifName\": {\"left\": \"GigabitEthernet0/7/1\", \"right\": \"GigabitEthernet0/5/0.1\"}, \"description\": {\"left\": \"connect to ce\", \"right\": \"test modify description\"}}}]}"

}

]

}

}

After editing the configuration, you can preview the modified contents of the current transaction to check the network- or NE-layer data differences.

Check the NE configuration after the dry run.

Method: POST

https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:dry-run-query

Request header:

| Key | Value |
| --- | --- |
| Content-Type | application/json |
| Accept | application/json |
| Cookie | accessSession={{accessSession}} |

Request body:

{

"huawei-ac-restconf-transactions:input":{

"trans-id": {{transactionsID}}

}

}

Response body:

{

"huawei-ac-restconf-transactions:output": {

"native": {

"dry-run-ne-native-confs-item": [

{

"ne-id": "1fd5468c-8244-11eb-8e45-6e1fce390113",

"ne-name": "NE8000-M8",

"ne-cfg": [

"<ifm xmlns=\"urn:huawei:yang:huawei-ifm\">\n<interfaces>\n<interface xmlns:ns0=\"urn:ietf:params:xml:ns:netconf:base:1.0\" ns0:operation=\"delete\">\n<name>GigabitEthernet0/7/1</name>\n</interface>\n</interfaces>\n</ifm>\n",

"<ifm xmlns=\"urn:huawei:yang:huawei-ifm\">\n<interfaces>\n<interface xmlns:ns0=\"urn:ietf:params:xml:ns:netconf:base:1.0\" ns0:operation=\"create\">\n<name>GigabitEthernet0/5/0.1</name>\n<type>GigabitEthernet</type>\n<description>test modify description</description>\n<vrf-name>5G-RAN</vrf-name>\n<ipv4 xmlns=\"urn:huawei:yang:huawei-ip\">\n<addresses>\n<address>\n<ip>20.1.2.9</ip>\n<mask>255.255.255.0</mask>\n<type>main</type>\n</address>\n</addresses>\n</ipv4>\n</interface>\n</interfaces>\n</ifm>\n"

]

}

]

},

"diff": {

"dry-run-ne-diff-confs-item": [

{

"ne-id": "1fd5468c-8244-11eb-8e45-6e1fce390113",

"ne-name": "NE8000-M8",

"ne-cfg-diffs": [

{

"feature": "/(urn:huawei:yang:huawei-ifm?revision=2020-02-14)ifm",

"diff-info": "{\"ifm\": {\"interfaces\": {\"interface\": [{\"left\": {\"[name=GigabitEthernet0/7/1]\": {\"name\": \"GigabitEthernet0/7/1\", \"description\": \"connect to ce\", \"vrf-name\": \"5G-RAN\", \"type\": \"GigabitEthernet\", \"huawei-ip:ipv4\": {\"addresses\": {\"address\": [{\"[ip=20.1.2.9]\": {\"ip\": \"20.1.2.9\", \"mask\": \"255.255.255.0\", \"type\": \"main\"}}]}}}}, \"right\": null}, {\"left\": null, \"right\": {\"[name=GigabitEthernet0/5/0.1]\": {\"name\": \"GigabitEthernet0/5/0.1\", \"description\": \"test modify description\", \"vrf-name\": \"5G-RAN\", \"type\": \"GigabitEthernet\", \"huawei-ip:ipv4\": {\"addresses\": {\"address\": [{\"[ip=20.1.2.9]\": {\"ip\": \"20.1.2.9\", \"mask\": \"255.255.255.0\", \"type\": \"main\"}}]}}}}}]}}}"

}

]

}

]

},

"mapconf": {

"service\_map\_confs": [

{

"source": "/huawei-ac-applications:applications/l3vpn:l3vpn/NE8000-M8/5G-RAN",

"dry-run-map-confs-item": [

{

"ne-id": "1fd5468c-8244-11eb-8e45-6e1fce390113",

"ne-name": "NE8000-M8",

"ne-cfg": [

"<bgp xmlns=\"urn:huawei:yang:huawei-bgp\">\n<global>\n<yang-enable>true</yang-enable>\n</global>\n</bgp>\n",

"<network-instance xmlns=\"urn:huawei:yang:huawei-network-instance\">\n<instances>\n<instance>\n<name>5G-RAN</name>\n<bgp xmlns=\"urn:huawei:yang:huawei-bgp\">\n<base-process>\n<afs>\n<af>\n<type>ipv4uni</type>\n<ipv4-unicast>\n<import-routes>\n<import-route>\n<protocol>direct</protocol>\n<process-id>0</process-id>\n</import-route>\n<import-route>\n<protocol>static</protocol>\n<process-id>0</process-id>\n</import-route>\n</import-routes>\n</ipv4-unicast>\n</af>\n</afs>\n<peers>\n<peer>\n<address>2.2.2.2</address>\n<remote-as>100</remote-as>\n</peer>\n</peers>\n</base-process>\n</bgp>\n<afs xmlns=\"urn:huawei:yang:huawei-l3vpn\">\n<af>\n<type>ipv4-unicast</type>\n<vpn-targets>\n<vpn-target>\n<value>100:11</value>\n<type>export-extcommunity</type>\n</vpn-target>\n<vpn-target>\n<value>100:11</value>\n<type>import-extcommunity</type>\n</vpn-target>\n</vpn-targets>\n<route-distinguisher>100:1</route-distinguisher>\n<tunnel-policy>LDP</tunnel-policy>\n</af>\n</afs>\n</instance>\n</instances>\n</network-instance>\n",

"<ifm xmlns=\"urn:huawei:yang:huawei-ifm\">\n<interfaces>\n<interface>\n<name>GigabitEthernet0/5/0.1</name>\n<description>test modify description</description>\n<vrf-name>5G-RAN</vrf-name>\n<type>GigabitEthernet</type>\n<ipv4 xmlns=\"urn:huawei:yang:huawei-ip\">\n<addresses>\n<address>\n<ip>20.1.2.9</ip>\n<mask>255.255.255.0</mask>\n<type>main</type>\n</address>\n</addresses>\n</ipv4>\n</interface>\n</interfaces>\n</ifm>\n"

]

}

]

}

]

}

}

}

Check the configuration and difference information generated after the dry run.

Commit the configuration.

Method: POST

https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:commit

Request header:

| Key | Value |
| --- | --- |
| Content-Type | application/json |
| Accept | application/json |
| Cookie | accessSession={{accessSession}} |

Request body:

{

"huawei-ac-restconf-transactions:input": {

"trans-id": "{{trans-id}}",

“no-network”: true,

"force": false,

"service-parallel": true

}

}

Response body:

{

"huawei-ac-restconf-transactions:output": {

"result": true

}

}

### Deleting Configurations

Obtain a token.

Method: PUT

https://{{northIP}}:26335/rest/plat/smapp/v1/oauth/token

Request header:

| Key | Value |
| --- | --- |
| Content-Type | application/json |

Request body:

{

"grantType":"password",

"userName":"3rdOSS",

"value":"Huawei@123"

}

Response body:

{

"accessSession": "x-1cpceq1j0bdhft2niqiotg087uak6ps82ptfuqnw2pteqm84ob04o77u3vo4rtlg6mpdle2ktgo9dfml9cmrbzljs5uk2k1f9ho7rw2pen6kqk49obdcamc8jzvs6llj",

"roaRand": "48fa2a40d3250e6beb6a16cd806034574cf4fd9969a5f096",

"expires": 1800,

"additionalInfo": null

}

Obtain the token. Headers of all sent packets must contain the token.

Apply for a transaction.

Method: POST

https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:create

Request header, which contains the token — **accessSession** — generated in step 1:

| Key | Value |
| --- | --- |
| Content-Type | application/json |
| Accept | application/json |
| Cookie | accessSession={{accessSession}} |

Response body:

{

"huawei-ac-restconf-transactions:output": {

"trans-id": "a4ce2a02-5e6f-41af-8824-852ba2155c2b"

}

}

A transaction ID needs to be applied for before a service is edited.

Delete configurations.

Method: DELETE

https://{{northIP}}:26335/restconf/v1/data/huawei-ac-applications:applications/ l3vpn:l3vpn/NE8000-M8/5G-RAN

Request header:

| Key | Value |
| --- | --- |
| restconf-transaction-id | {{transactionsID}} |
| Accept | application/json |
| Cookie | accessSession={{accessSession}} |
| Content-Type | application/json |

Status code: 200

Commit the configuration.

Method: POST

https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:commit

Request header:

| Key | Value |
| --- | --- |
| Content-Type | application/json |
| Accept | application/json |
| Cookie | accessSession={{accessSession}} |

Request body:

{

"huawei-ac-restconf-transactions:input": {

"trans-id": "{{trans-id}}",

“no-network”: true,

"force": false,

"service-parallel": true

}

}

Response body:

{

"huawei-ac-restconf-transactions:output": {

"result": true

}

}

## Quiz

If a customer has its own OSS system, how do you develop a service model and complete interconnection based on the customer's requirements?

Reference Answers

Reference answer:

Sort out all configurations that need to be delivered for a network service.

Sort out the external input parameters required for these configurations. Then, the input parameters required for interconnecting with the northbound OSS system are determined.

Determine the service YANG model based on the northbound input parameters. Datacom Network Open Programmability automatically generates northbound APIs based on the service YANG model for interconnection with the upper-layer OSS.