Huawei Certification Training

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

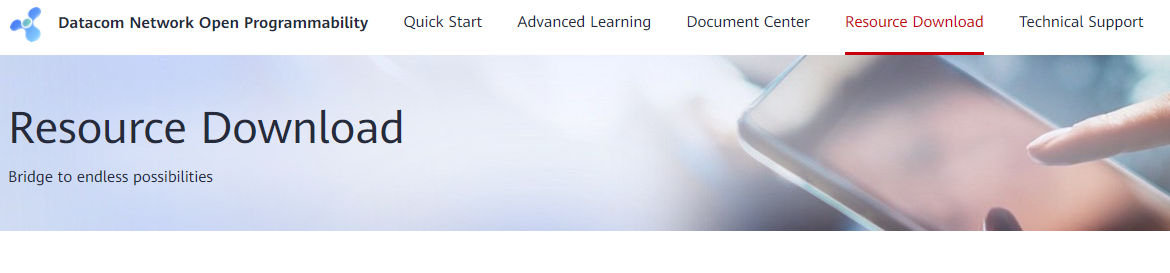
Download Java 1.8 from the official website (<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>), select the default configuration, and complete the installation as prompted.

* 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](#_Toc73715790)

[1 Interconnecting with New Devices – SND Package Practice 2](#_Toc73715791)

[1.1 Background 2](#_Toc73715792)

[1.1.1 SND Package 2](#_Toc73715793)

[1.1.2 Open Device Capabilities 3](#_Toc73715794)

[1.2 Introduction 3](#_Toc73715795)

[1.2.1 Networking 3](#_Toc73715796)

[1.2.2 Procedure 4](#_Toc73715797)

[1.3 Environment Preparation 4](#_Toc73715798)

[1.4 Compiling an SND Package 4](#_Toc73715799)

[1.4.1 Creating an SND Package Template 4](#_Toc73715800)

[1.4.2 Modifying the Package Configuration File 8](#_Toc73715801)

[1.4.3 Adding a YANG Model File 10](#_Toc73715802)

[1.4.4 Verifying the YANG Model File 10](#_Toc73715803)

[1.4.5 Compiling Device Driver Configurations 11](#_Toc73715804)

[1.4.6 Configuring Keys 15](#_Toc73715805)

[1.4.7 Generating an SND Package 20](#_Toc73715806)

[1.4.8 Uploading and Activating the SND Package 21](#_Toc73715807)

[1.5 Using iMaster NCE to Manage Devices 23](#_Toc73715808)

[1.5.1 Obtaining Device Interconnection Information 23](#_Toc73715809)

[1.5.2 Managing a Device 23](#_Toc73715810)

[1.5.3 Synchronizing Device Information 24](#_Toc73715811)

[1.6 Delivering Device Configurations on the GUI 25](#_Toc73715812)

[1.7 Delivering Device Configurations Through the Northbound API 30](#_Toc73715813)

[1.7.1 Creating a Northbound User 30](#_Toc73715814)

[1.7.2 Creating Configurations 32](#_Toc73715815)

[1.7.3 Checking Configurations 39](#_Toc73715816)

[1.7.4 Modifying Configurations 40](#_Toc73715817)

[1.7.5 Deleting Configurations 46](#_Toc73715818)

# Interconnecting with New Devices – SND Package Practice

## Background

Based on the YANG model-driven open architecture, Huawei Datacom Network Open 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 and development of new network services.

This document provides guidance for you to compile an SND package to open native device capabilities, that is, use iMaster NCE to manage network devices, deliver basic device configurations, and generate northbound APIs. Upon completion of this course, you will be able to:

* Compile an SND package.
* Deliver basic configurations based on native device capabilities.

### SND Package

An SND package is a type of iMaster NCE software package, which provides a data model for interconnection between the open programmability system and NEs. The data model typically contains a .py file and YANG data models of several features. The .py file is used to define NE information, such as the device type, vendor, and connection information. The YANG data models describe the data structure of NE-related features.



### Open Device Capabilities

iMaster NCE can open native device capabilities. Based on device YANG models, Datacom Network Open Programmability automatically generates northbound APIs and configuration pages to quickly manage Huawei and third-party devices. It supports difference discovery, consistency verification, and synchronization for device data.



This lab mainly introduces the process and operations of opening device capabilities on iMaster NCE. For more information about capabilities, visit **Document Center** in the Developer Community.

## Introduction

### Networking



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

### Procedure

The procedure for this lab is as follows:

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

Compile an SND package locally.

Use iMaster NCE to manage devices.

Use iMaster NCE to deliver basic device configurations (creating device sub-interfaces is used as an example).

Use the northbound API to deliver device configurations.

## Environment Preparation

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

## Compiling an SND Package

This section describes how to compile an SND package for the NE8000 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>.

### Creating an SND Package Template

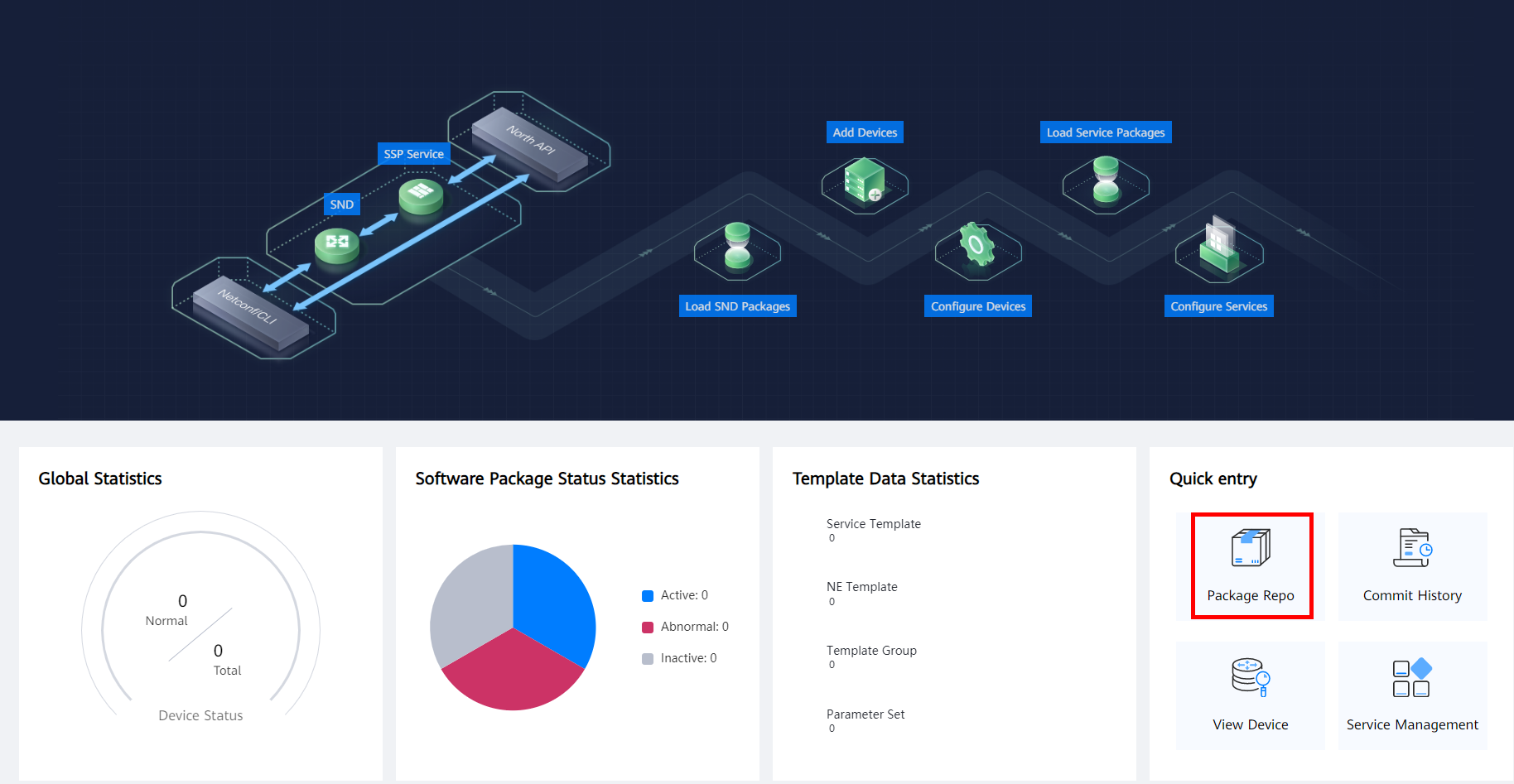
Log in to iMaster NCE.

On the homepage, click **Service Programming**.

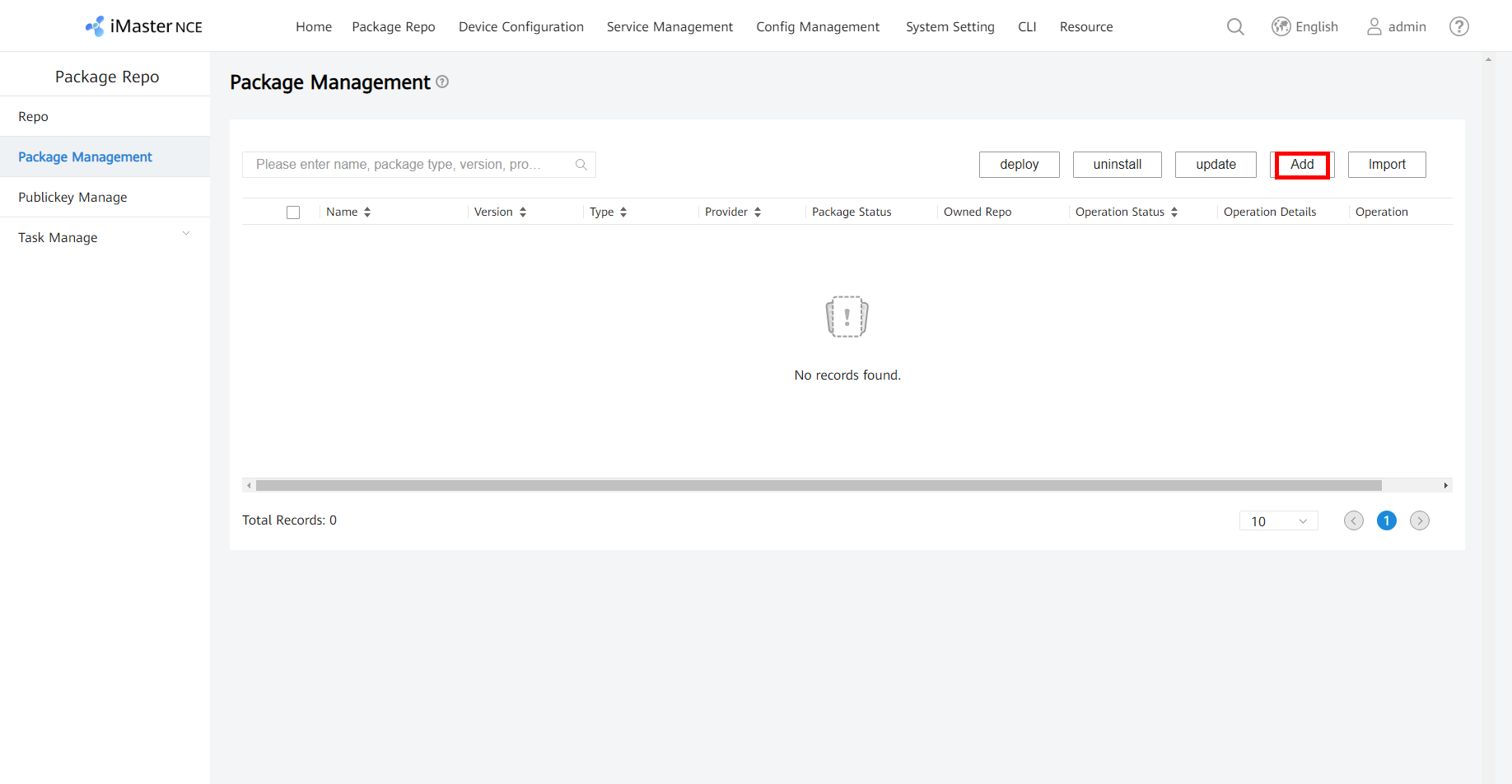


Create an SND package template.

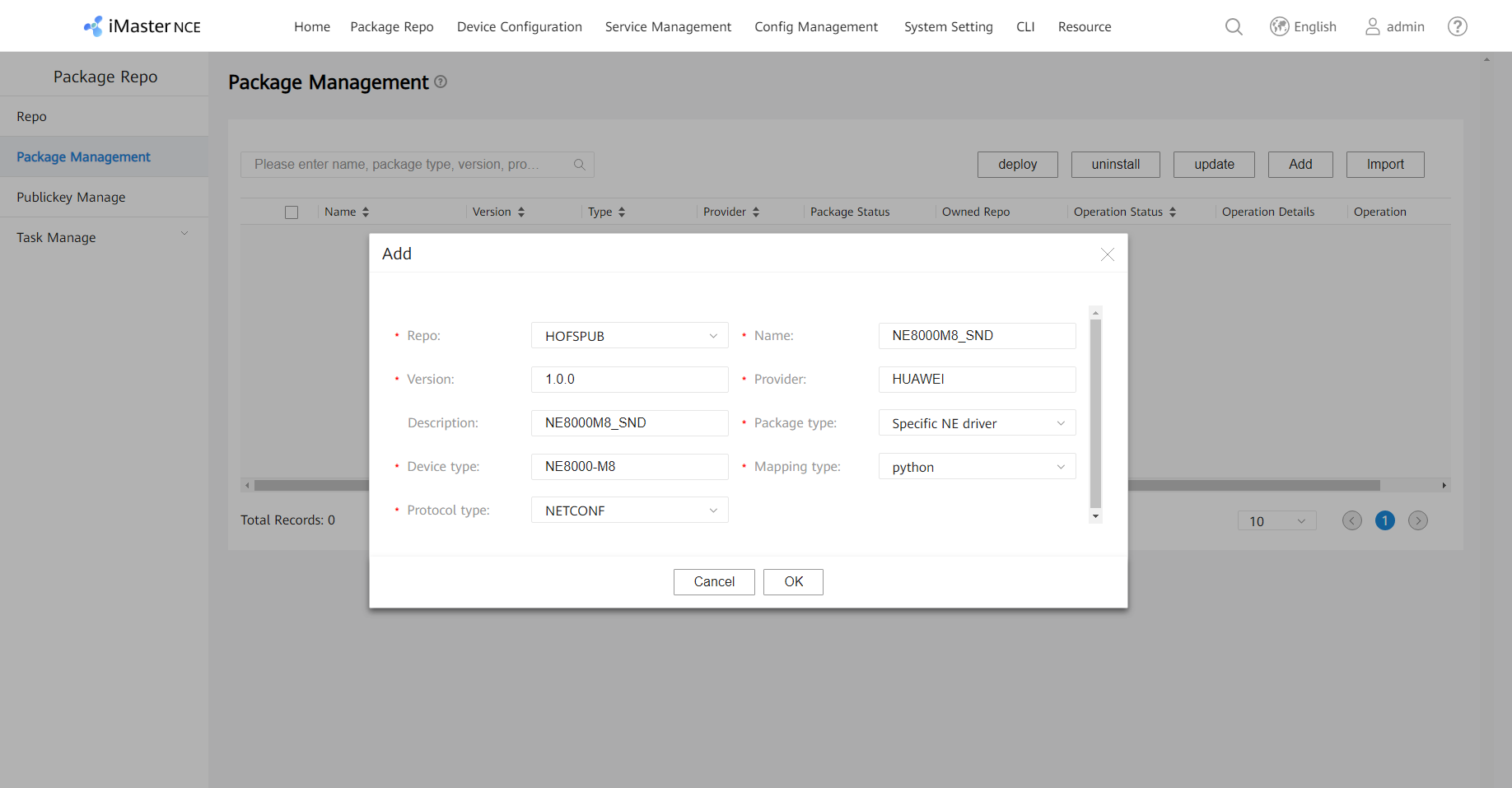
Click **Package Repo**. The page for creating an SND package is displayed.



Choose **Package Management** and click **Add**.



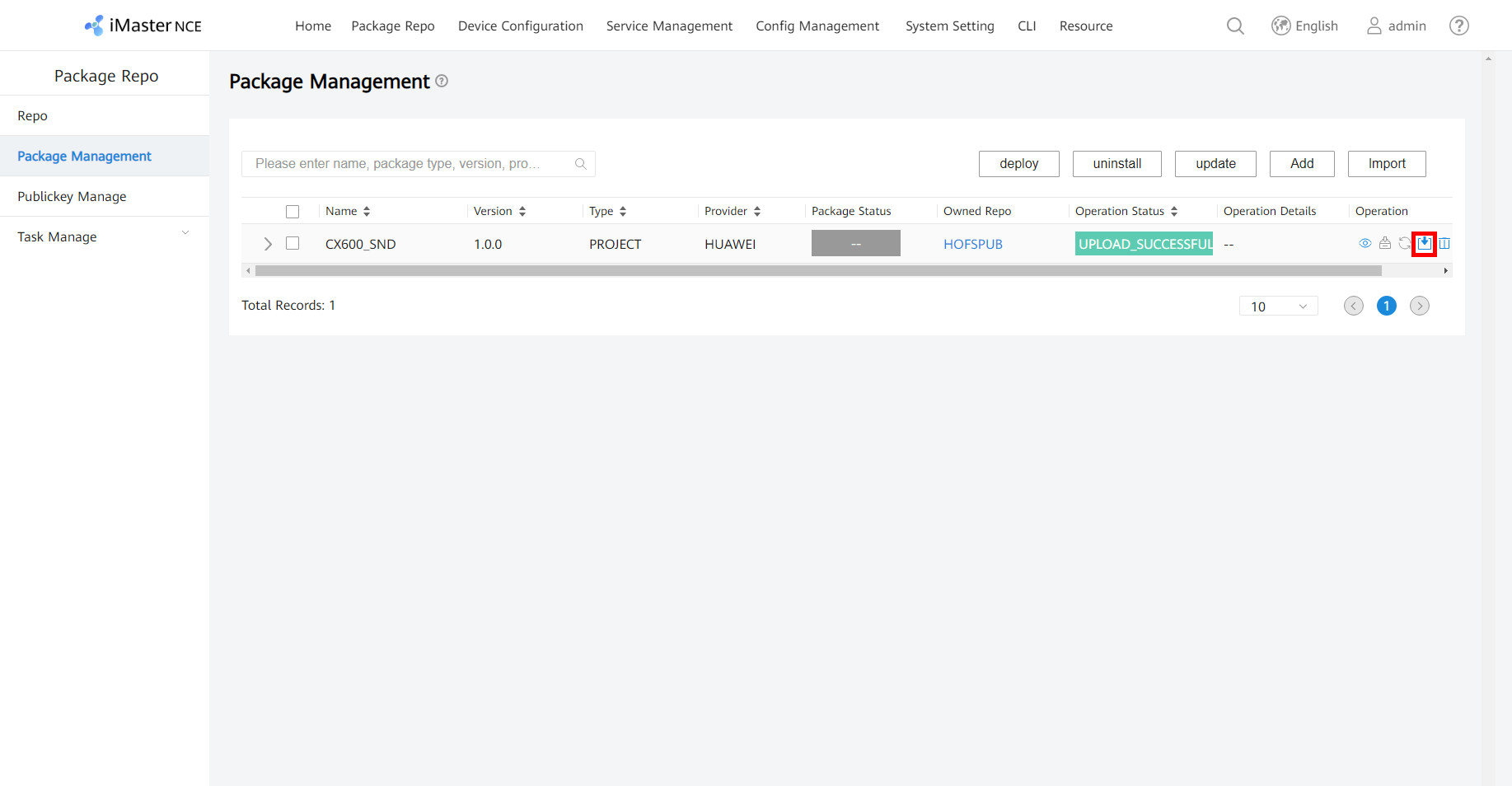
Set SND package parameters and click **OK**.



* **Name**: NE8000M8\_SND
* **Version**: 1.0.0
* **Provider**: HUAWEI
* **Package type**: Specific NE driver
* **Mapping type**: python
* **Device type**: NE8000-M8
* **Protocol type**: NETCONF

Download the template and import it to the local IDE PyCharm.

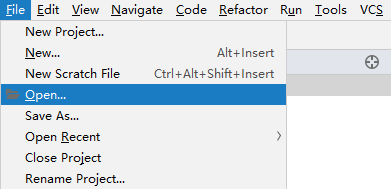
Click C:\Users\w00438901\AppData\Roaming\eSpace_Desktop\UserData\w00438901\imagefiles\FE65EEDE-9E62-4A10-A318-691B7B75584F.png in the **Operation** column to download the template to the local PC.



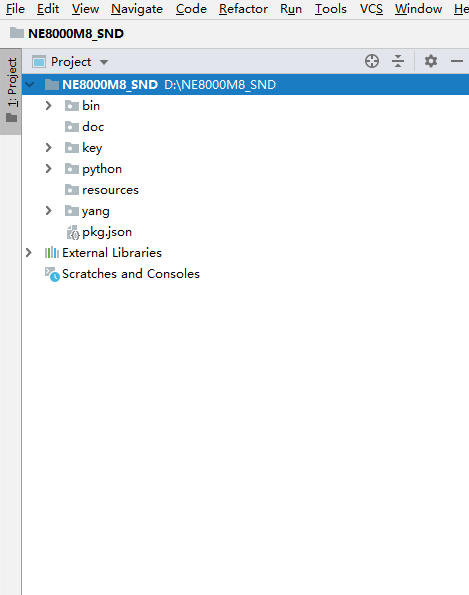
Obtain the **NE8000M8\_SND.zip** package.



Decompress the package to a specified local directory and open the directory in PyCharm.



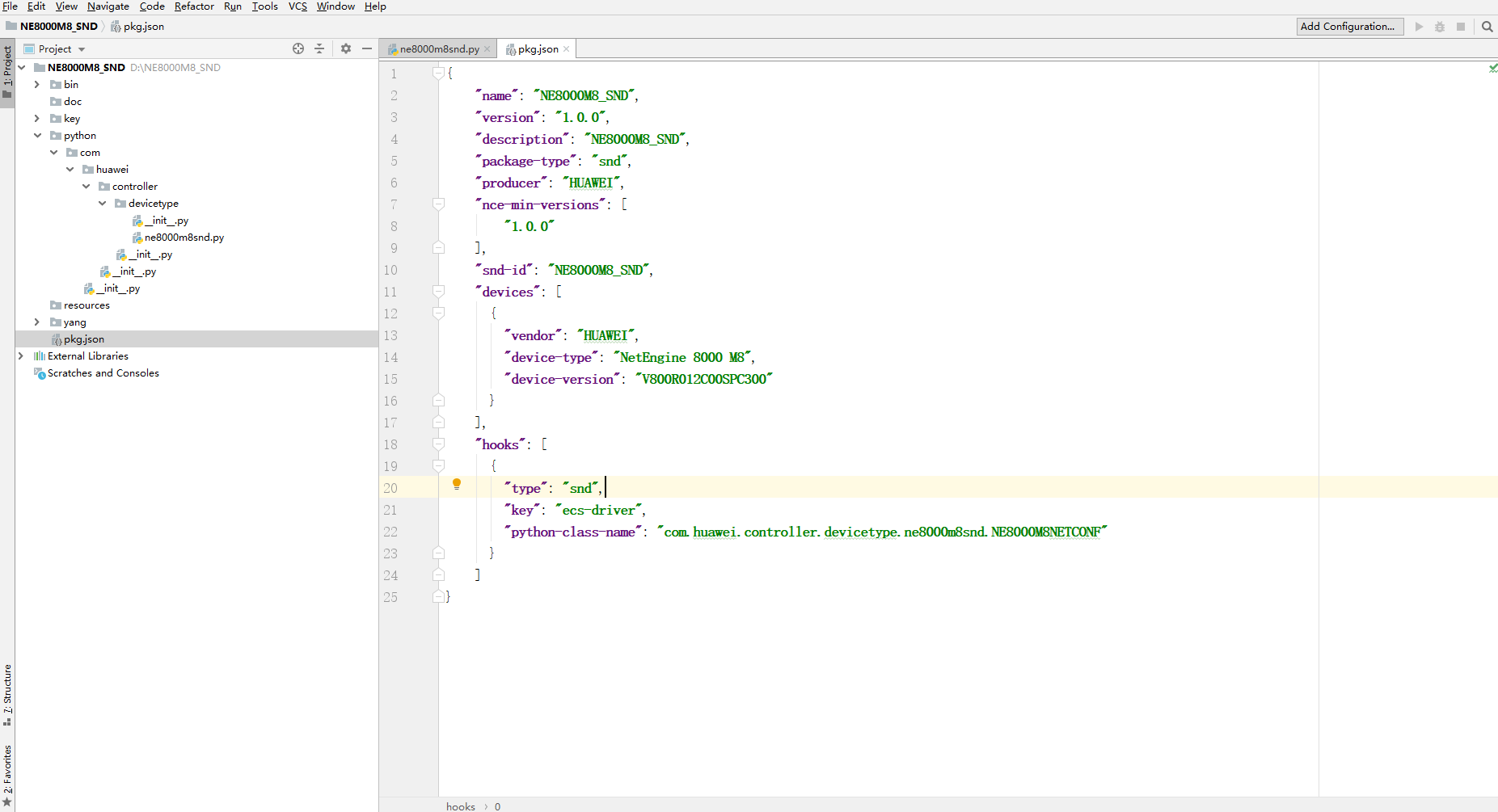
Open the template in PyCharm. The detailed directory structure is as follows:



1. **bin**: stores executable scripts, including tools and scripts for packing.
2. **key**: stores private keys.
3. **python**: stores Python code, including device information, connection parameters, and driver information.
4. **yang**: YANG module of the device. Each module corresponds to a function on the device. Together, they form the device YANG model.
5. **pkg.json**: package configuration file, which is used to set basic attributes and callback hooks of the current software package.

### Modifying the Package Configuration File

Modify the specified device parameters in the **pkg.json** file.



Open the **pkg.json** file, change the device type and device version in lines 14 and 15 to the correct values. In this lab, NetEngine 8000 M8 running V800R012C00SPC300 is used. Retain the default values in the actual lab. Additionally, change the .py file to **ne8000m8snd** and **class** to **NE8000M8NETCONF** in line 22.

{

"name": "NE8000M8\_SND",

"version": "1.0.0",

"description": "NE8000M8\_SND",

"package-type": "snd",

"producer": "HUAWEI",

"nce-min-versions": [

"1.0.0"

],

"snd-id": "NE8000M8\_SND",

"devices": [

{

"vendor": "HUAWEI",

"device-type": "NetEngine 8000 M8",

"device-version": "V800R012C00SPC300"

}

],

"hooks": [

{

"type": "snd",

"key": "ecs-driver",

"python-class-name": "com.huawei.controller.devicetype.ne8000m8snd.NE8000M8NETCONF"

}

]

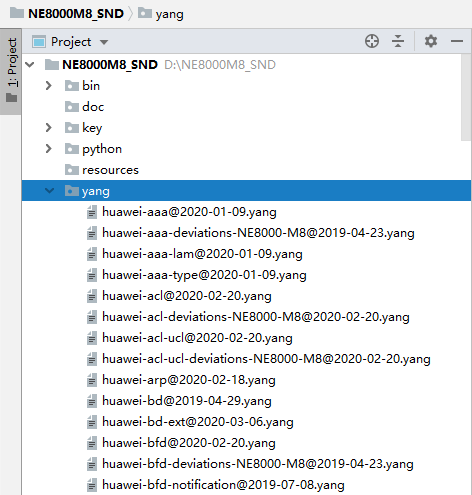
}

### Adding a YANG Model File

Add the YANG model file of the corresponding version for the NE8000 M8 to the **yang** directory of the SND package template.

Delete the YANG model file from the SND package template.

Copy the YANG model file (obtained from the website of the device vendor) of the NE8000 M8 to the **yang** directory of the SND package template.



### Verifying the YANG Model File

Copy the YANG model file in the **yang** directory of the software package to the directory where **yang-offline-util.zip** is decompressed.

Run the **java -jar .\yang-offline-util.jar validate console path .** command to verify the YANG model file. If the command output is empty, the YANG model file format is correct.

D:\yang-offline-util> java -jar .\yang-offline-util.jar validate console path .

### Compiling Device Driver Configurations

#### Complete Code

Compile the Python file **ne8000m8snd.py**. (Change the name to that in line 22 in **pkg.json**.)

Expand the **python** directory of the SND package, and modify the **ne8000m8snd.py** file to compile the information required for device interconnection, such as connection information and driver information. For details, see section "SND Package" in the *Open Programmability Development Guide*. The complete code for managing the NE8000 M8 is as follows:

from aoc.snd.netconfsnd import NetconfSND

from aoc.snd.snd\_model\_pb2.sysoidinfo\_pb2 import SysoidInfo

from aoc.snd.snd\_model\_pb2.connectinfo\_pb2 import ConnectInfos, ProtocolEntity, DEFAULT\_CONNECT, PRIMARY\_CONNECTION, HelloEntity

from aoc.snd.snd\_model\_pb2.channelInfo\_pb2 import SINGLE\_CHANNEL, PROTECTED\_MODE

from aoc.snd.snd\_model\_pb2.ecsdriver\_pb2 import CommonDriverInfo

from aoc.snd.snd\_model\_pb2.ecsdriver\_pb2 import NetconfDriverInfo

class NE8000M8NETCONF (NetconfSND):

def getSysoidInfo(self, aoccontext, request=None):

sysoidInfo = SysoidInfo()

sysoidEntity = sysoidInfo.sysoidEntity.add()

sysoidEntity.sysoid = "1.3.6.1.4.1.2011.2.360.1.10"

sysoidEntity.deviceType = "ROUTER"

sysoidEntity.deviceModel = "NetEngine 8000 M8"

sysoidEntity.deviceVendor = "HUAWEI"

return sysoidInfo

def getConnectInfo(self, aoccontext, request=None):

self.logger.info('getConnectInfo start.')

connectInfos = ConnectInfos()

primaryConnectInfo = connectInfos.connectInfo.add()

primaryConnectInfo.connectPolicy = DEFAULT\_CONNECT

primaryConnectInfo.channelInfo.readChannel = SINGLE\_CHANNEL

primaryConnectInfo.channelInfo.is\_read\_share\_write = True

primaryConnectInfo.protocolEntity.protocolType = ProtocolEntity.netconf

primaryConnectInfo.protocolEntity.helloEntity.helloType = HelloEntity.extendType

primaryConnectInfo.connectionPriority = PRIMARY\_CONNECTION

return primaryConnectInfo

def getCommonDriverInfo(self, aoccontext, request=None):

common\_driver = CommonDriverInfo()

common\_driver.deleteStrategy = 1

syncToDel = common\_driver.para.add()

syncToDel.key = "sync-to-del-enable"

syncToDel.value = "true"

return common\_driver

def getNetconfDriverInfo(self, aoccontext, request=None):

netconf\_driver = NetconfDriverInfo()

netconf\_driver.phase = "two"

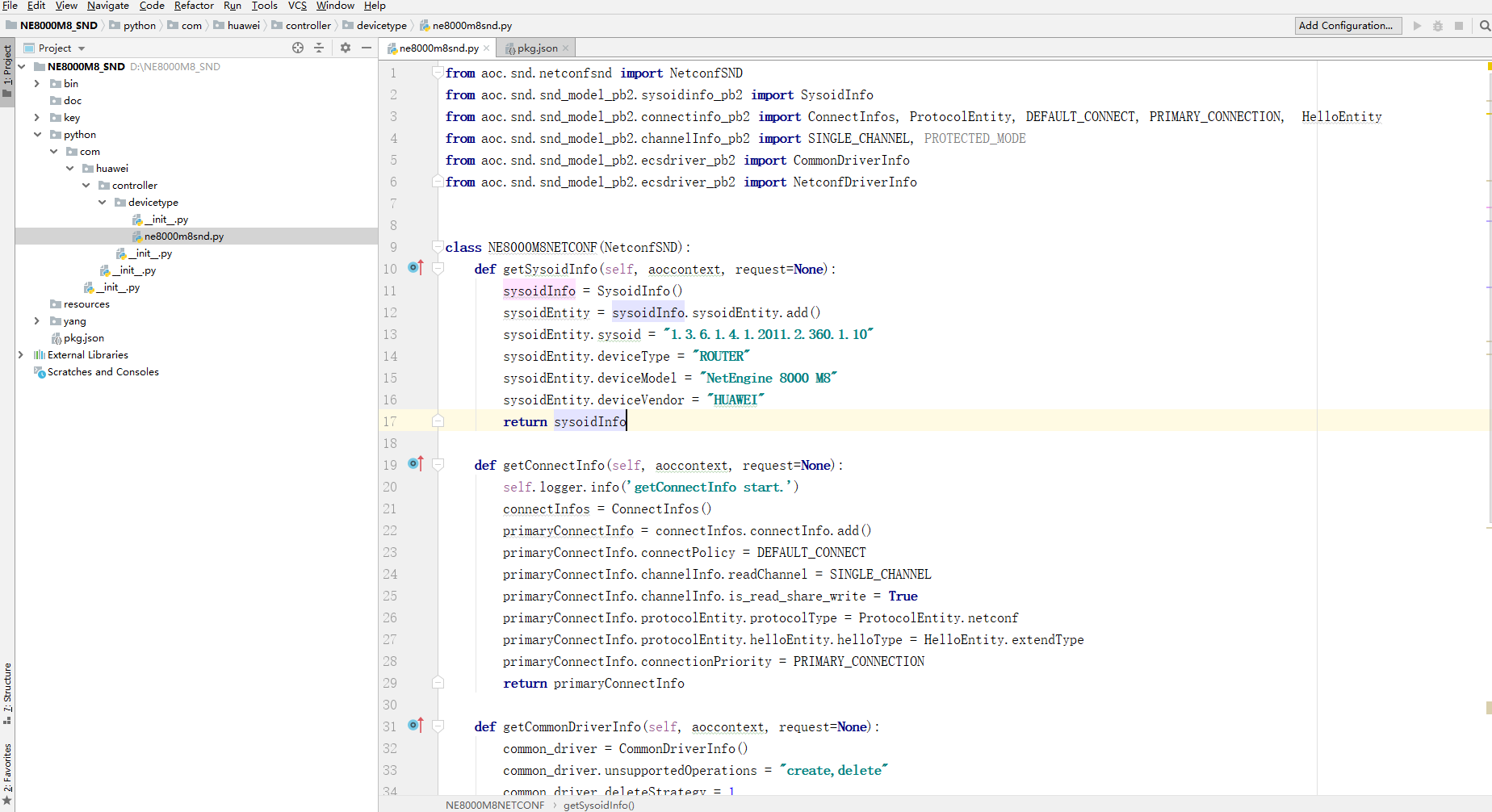
netconf\_driver.classification = "huawei-v5"

netconf\_driver.modelDiff = "match"

netconf\_driver.testOption = "set"

return netconf\_driver

Code after the modification:



#### Code Interpretation

Import the necessary module.

from aoc.snd.netconfsnd import NetconfSND

from aoc.snd.snd\_model\_pb2.sysoidinfo\_pb2 import SysoidInfo

from aoc.snd.snd\_model\_pb2.connectinfo\_pb2 import ConnectInfos, ProtocolEntity, DEFAULT\_CONNECT, PRIMARY\_CONNECTION, HelloEntity

from aoc.snd.snd\_model\_pb2.channelInfo\_pb2 import SINGLE\_CHANNEL, PROTECTED\_MODE

from aoc.snd.snd\_model\_pb2.ecsdriver\_pb2 import CommonDriverInfo

from aoc.snd.snd\_model\_pb2.ecsdriver\_pb2 import NetconfDriverInfo

Import the module required by the code. If the module has not been installed, install it by following the instructions in "Lab Environment".

**NetconfSND**: parent class of all NETCONF SND packages. The default configuration has been implemented in the parent class. For customized configurations, inherit NetconfSND in the SND package and override methods in the parent class.

**SysoidInfo**: configures the device type, vendor, and model corresponding to the SND package. After a device is managed, the corresponding SND package is matched based on the information.

**ConnectInfos**: configures connection information, such as the protocol for establishing a connection with the device, number of channels to be established, and Hello packet for the handshake.

**CommonDriverInfo**: customizes operation types in device configuration packets, for example, whether the device supports the create operation (if the create operation is not supported, merge needs to be used) and whether the device is deleted and then added for synchronization.

**NetconfDriverInfo**: configures NETCONF protocol parameters, such as whether the two-phase delivery and set mode are supported.

Register the sysoid information of the device. (You only need to modify the device sysoid, type, model, and vendor.)

def getSysoidInfo(self, aoccontext, request=None):

sysoidInfo = SysoidInfo()

sysoidEntity = sysoidInfo.sysoidEntity.add()

sysoidEntity.sysoid = "1.3.6.1.4.1.2011.2.360.1.10"

sysoidEntity.deviceType = "ROUTER"

sysoidEntity.deviceModel = "NetEngine 8000 M8"

sysoidEntity.deviceVendor = "HUAWEI"

return sysoidInfo

Define the getSysoidInfo(self, aoccontext, request=None) method. This method defines the sysoid in the SND package, such as device type (deviceType), model (deviceModel), and vendor (deviceVendor).

Methods for obtaining related information for Huawei devices:

* Sysoid

<NE8000M8>system-view

[~NE8000M8]diagnose

[~NE8000M8-diagnose]display system information

The system object ID:

1.3.6.1.4.1.2011.2.360.1.10

* **DeviceType**: The default value is **ROUTER**.
* Device model

[~NE8000M8]display version

Huawei Versatile Routing Platform Software

VRP (R) software, Version 8.191 (NetEngine 8000 M8 V800R012C00SPC300)

Copyright (C) 2012-2020 Huawei Technologies Co., Ltd.

HUAWEI NetEngine 8000 M8 uptime is 90 days, 16 hours, 21 minutes

Configure the connection capability information of the device. (This part of code can be directly invoked and does not need to be modified.)

def getConnectInfo(self, aoccontext, request=None):

self.logger.info('getConnectInfo start.')

connectInfos = ConnectInfos()

primaryConnectInfo = connectInfos.connectInfo.add()

primaryConnectInfo.connectPolicy = DEFAULT\_CONNECT

primaryConnectInfo.channelInfo.readChannel = SINGLE\_CHANNEL

primaryConnectInfo.channelInfo.is\_read\_share\_write = True

primaryConnectInfo.protocolEntity.protocolType = ProtocolEntity.netconf

primaryConnectInfo.protocolEntity.helloEntity.helloType = HelloEntity.extendType

primaryConnectInfo.connectionPriority = PRIMARY\_CONNECTION

return primaryConnectInfo

Define the getConnectInfo(self, aoccontext, request=None) method, which defines connection information.

* **primaryConnectInfo.connectPolicy = DEFAULT\_CONNECT**: sets the connection policy.
* **primaryConnectInfo.channelInfo.readChannel = SINGLE\_CHANNEL**: specifies the channel configuration. Different channel types need to be configured for the NETCONF protocol. **Protected\_MODE** indicates the active/standby mode for protection, and **SINGLE\_CHANNEL** indicates a single channel with no protection.
* **primaryConnectInfo.channelInfo.is\_read\_share\_write = True**: sets whether the read and write channels are shared. If **read\_share\_write** is set to **true**, the read and write channels are shared. In this case, you do not need to set **readChannel**.
* **primaryConnectInfo.protocolEntity.protocolType = ProtocolEntity.netconf**: specifies the protocol type.
* **primaryConnectInfo.protocolEntity.helloEntity.helloType = HelloEntity.extendType**: specifies the packet type. The value **defaultType** is used to establish a NETCONF channel for schema interconnection and leave the capability set empty. The value **standardType** is used to establish a NETCONF channel of the standard YANG capability set and leave the capability set empty. The value **extendType** is used to establish a NETCONF channel of the extended YANG capability set while a capability set list needs to be added.
* **primaryConnectInfo.connectionPriority = PRIMARY\_CONNECTION**: specifies the connection invoking priority.

Configure the device type. (This part of code can be directly invoked and does not need to be modified.)

def getCommonDriverInfo(self, aoccontext, request=None):

common\_driver = CommonDriverInfo()

common\_driver.deleteStrategy = 1

syncToDel = common\_driver.para.add()

syncToDel.key = "sync-to-del-enable"

syncToDel.value = "true"

return common\_driver

def getNetconfDriverInfo(self, aoccontext, request=None):

netconf\_driver = NetconfDriverInfo()

netconf\_driver.phase = "two"

netconf\_driver.classification = "huawei-v5"

netconf\_driver.modelDiff = "match"

netconf\_driver.testOption = "set"

return netconf\_driver

Define the getCommonDriverInfo(self, aoccontext, request=None) method, which is used to define the interaction policy with the device.

* **common\_driver.deleteStrategy = 1**: specifies whether to delete specific information from the delete or remove packets. If the value is **1**, specific information is deleted. If the value is **0** or empty, the entire container is deleted, which may pose security risks. If the configuration of the controller is inconsistent with that of the device, extra configurations may be deleted. The default value is 1.
* **syncToDel.key = "sync-to-del-enable"**: defines the key value.
* **syncToDel.value = "true"**: specifies whether southbound device configuration instances can be deleted during data consistency verification. The value **true** indicates that southbound device configuration instances can be deleted.

Define the getNetconfDriverInfo(self, aoccontext, request=None) method, which is used to customize NETCONF parameters.

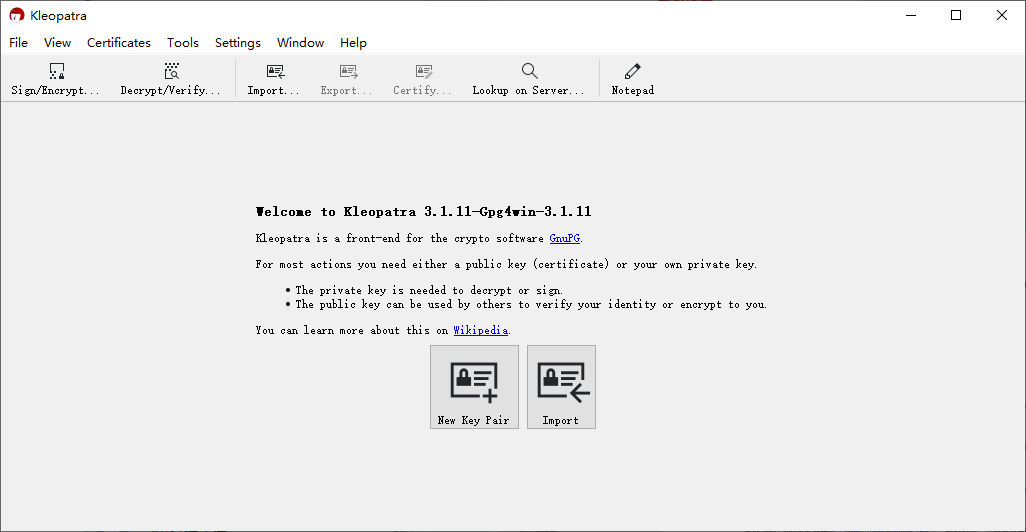
* **netconf\_driver.phase = "two"**: configures two-phase delivery for NETCONF.
* **netconf\_driver.classification = "huawei-v5"**: specifies the YANG model channel to be used. In this example, the channel is **huawei-v5**.
* **netconf\_driver.modelDiff = "match"**: defines the packet conversion type. If the value is **same**, a standard schema packet is delivered. If the value is **match**, a standard YANG packet is delivered.
* **netconf\_driver.testOption = "set"**: specifies the packet verification policy. If the value is **set**, the packet is directly processed without verification.

### Configuring Keys

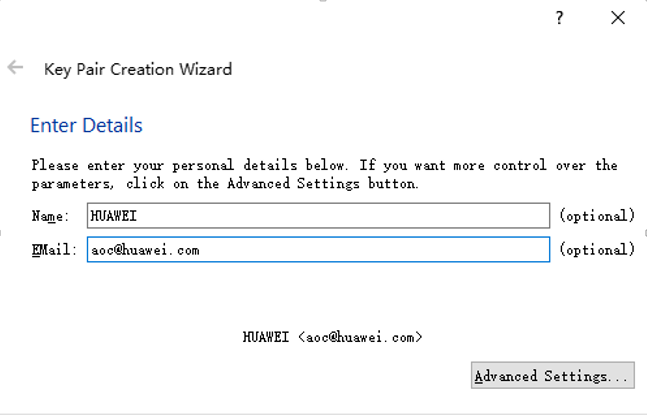
Use Gpg4win to generate a public key and a private key. Save the private key to the SND package and upload the public key to iMaster NCE for encryption and authentication.

Generate the public key and private key.

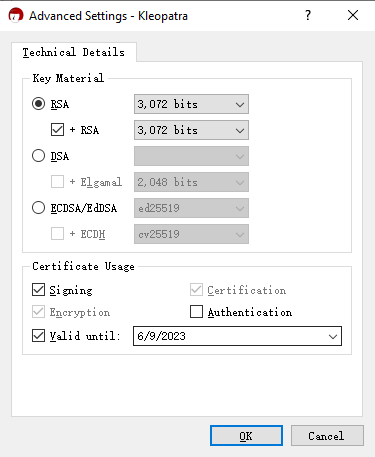
Start the Gpg4win key generation tool and click **New Key Pair**.



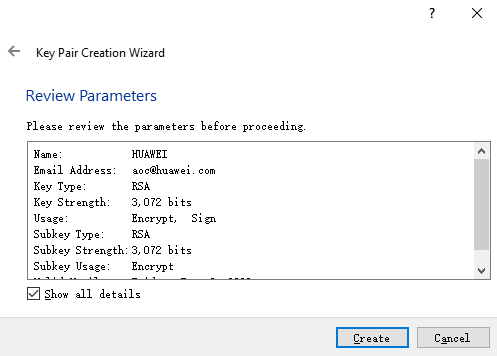
In the **Key Pair Creation Wizard** dialog box, enter the name and email address.



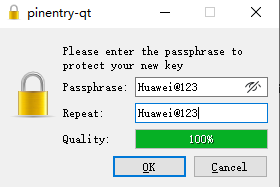
Click **Advanced Settings**. In the dialog box that is displayed, select **3072 bits (default)** and click **OK**.



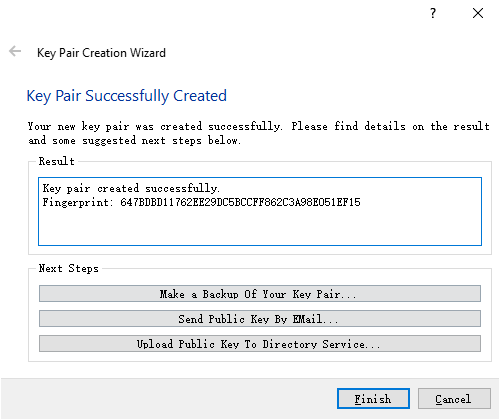
Click **Next**, select **Show all details**, and click **Create**.



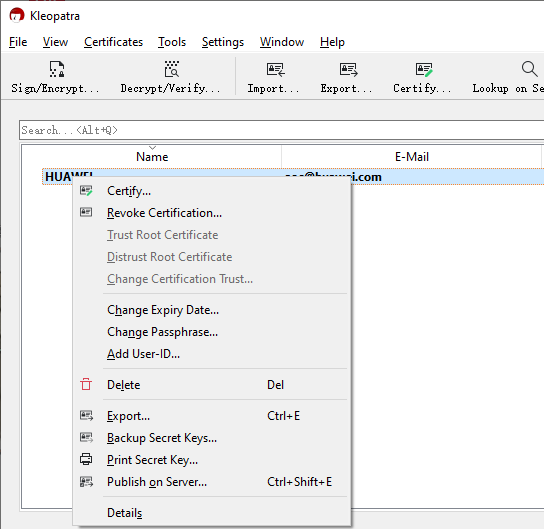
In the dialog box that is displayed, set the password, for example, **Huawei@123**, and click **OK**.



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.

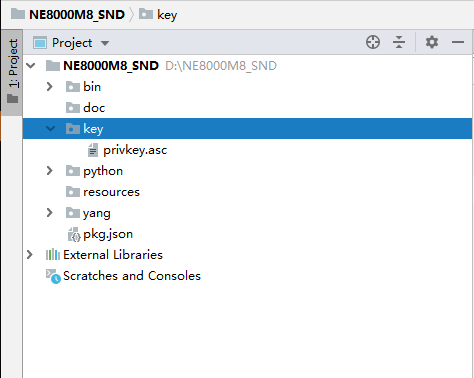


On the homepage, right-click the public key file and choose **Export** to export the public key file named **public.asc**. Then upload the file to the AOC for authentication.



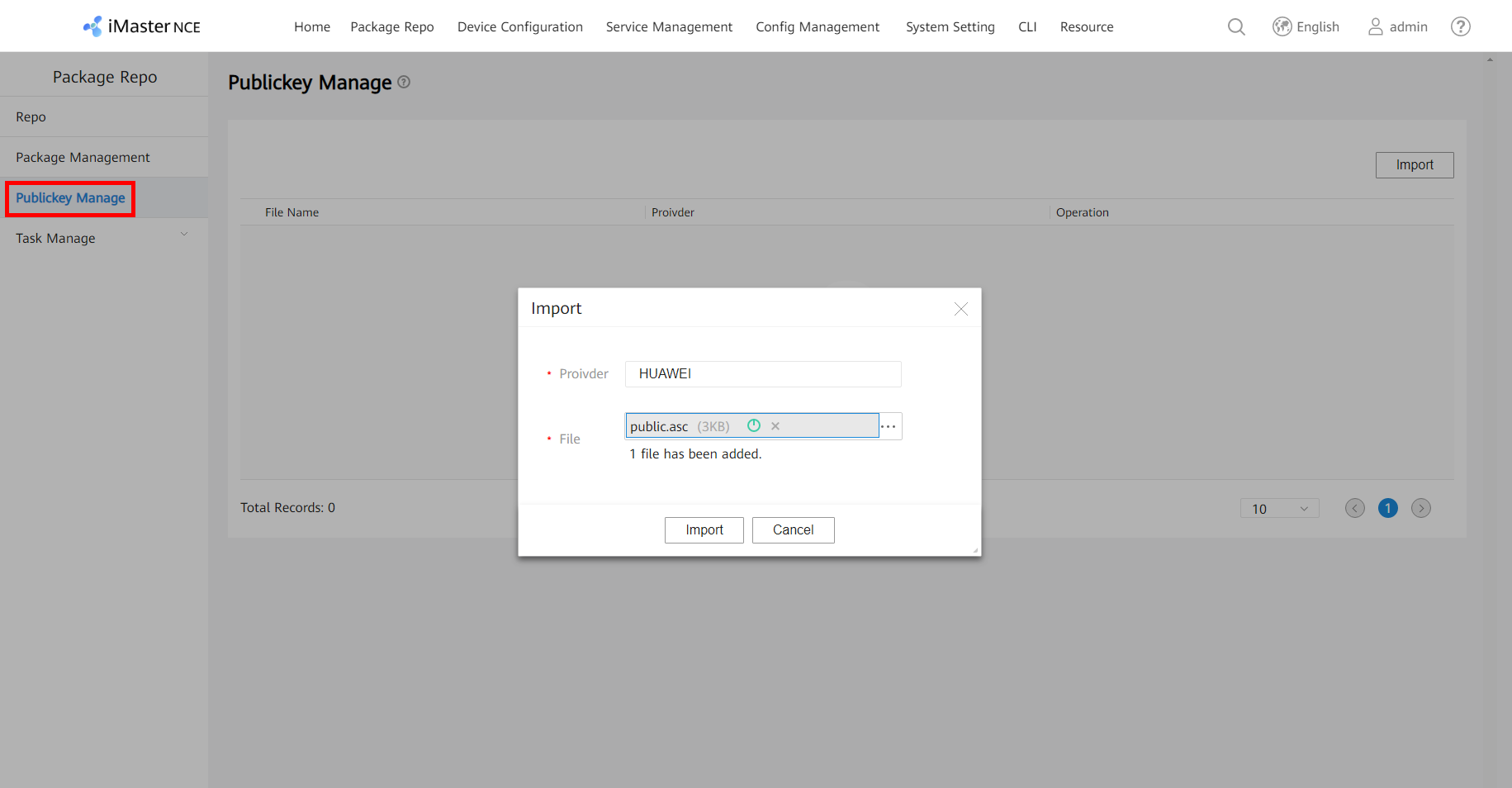
Save the private key to the local PC.

Copy the exported key file **private.asc** to the **key** directory of the NE8000 M8 SND package template.



Upload the public key file.

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



### Generating an SND Package

Open the **cmd** window, go to the **bin** directory of **NE8000M8\_SND**, run the **makeFile.bat** script, and enter the configured key password, for example, **Huawei@123**.

D:\NE8000M8\_SND\bin>makeFile.bat

Please input password for private key:

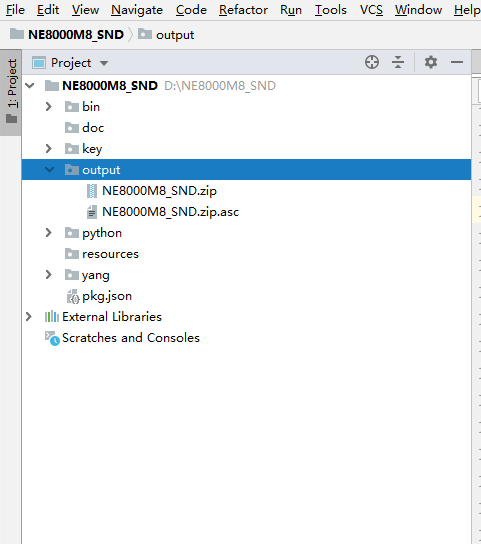
Huawei@123

…..

2021-02-04 11:38:38,588 INFO [com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] - [Sign]Generate signature file success.

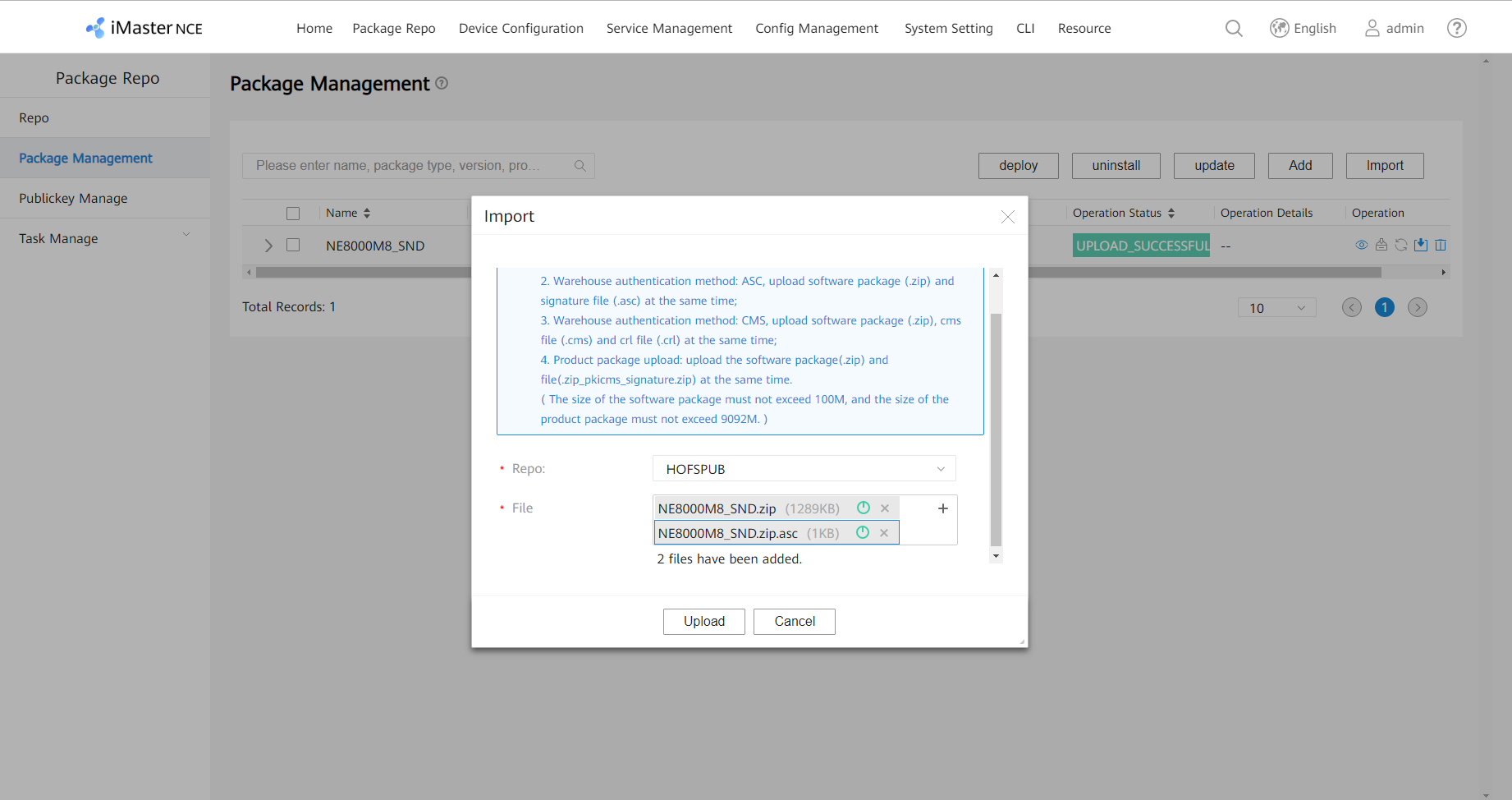
2021-02-04 11:38:38,589 INFO [com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign] Sign: Execute success

If the command output contains **success**, the SND package is generated in the **output** directory of the NE8000 M8 SND package template.

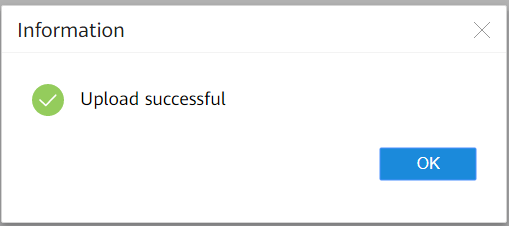


### Uploading and Activating the SND Package

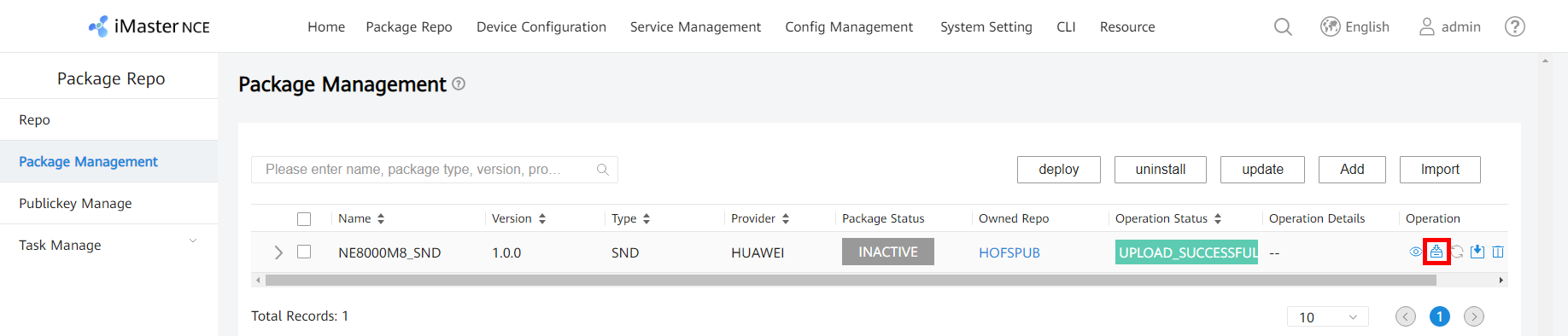
On iMaster NCE, choose **Package Repo** > **Package Management** to import the SND package.



Click **OK** after the SND package is successfully uploaded.



Click C:\Users\w00438901\AppData\Roaming\eSpace_Desktop\UserData\w00438901\imagefiles\2720D52A-C176-4418-B6B2-98DCEA5363AC.png in the **Operation** column to activate the SND package.



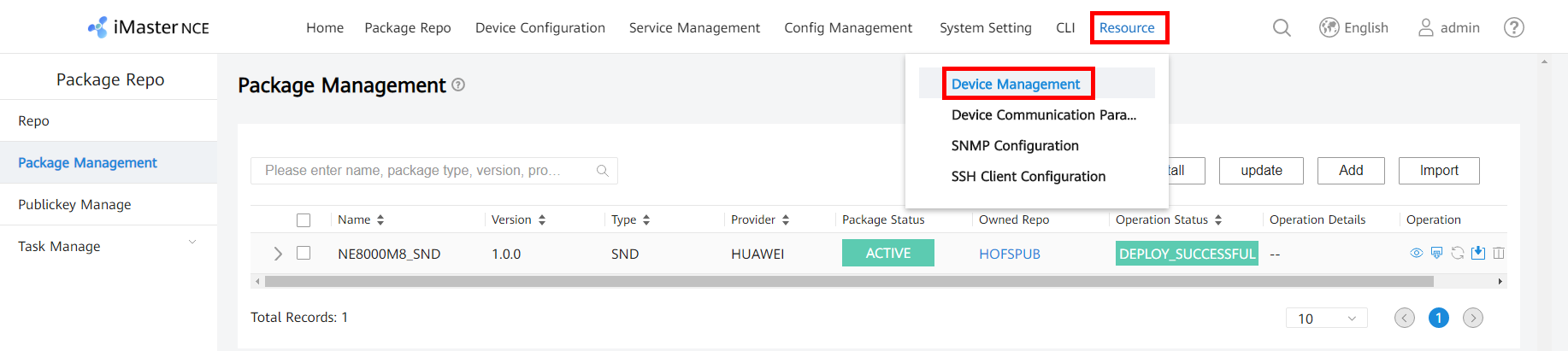
## Using iMaster NCE to Manage Devices

### Obtaining Device Interconnection Information

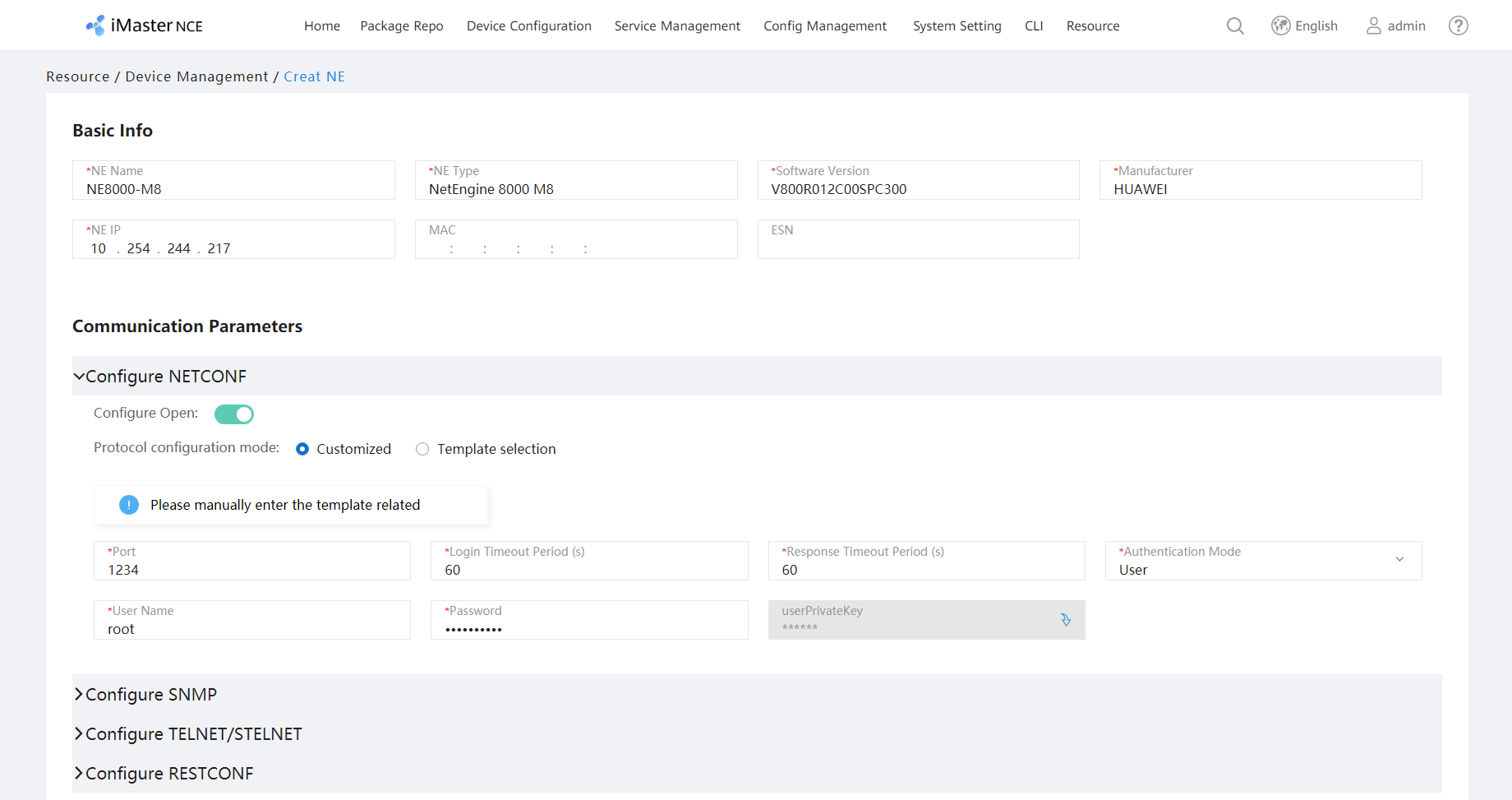
In this lab, the NE8000 M8 is used, whose IP address, NETCONF user name, and password are **10.254.244.217**, **root**, and **Huawei@123**, respectively.

### Managing a Device

On iMaster NCE, choose **Resource** > **Device Management**.

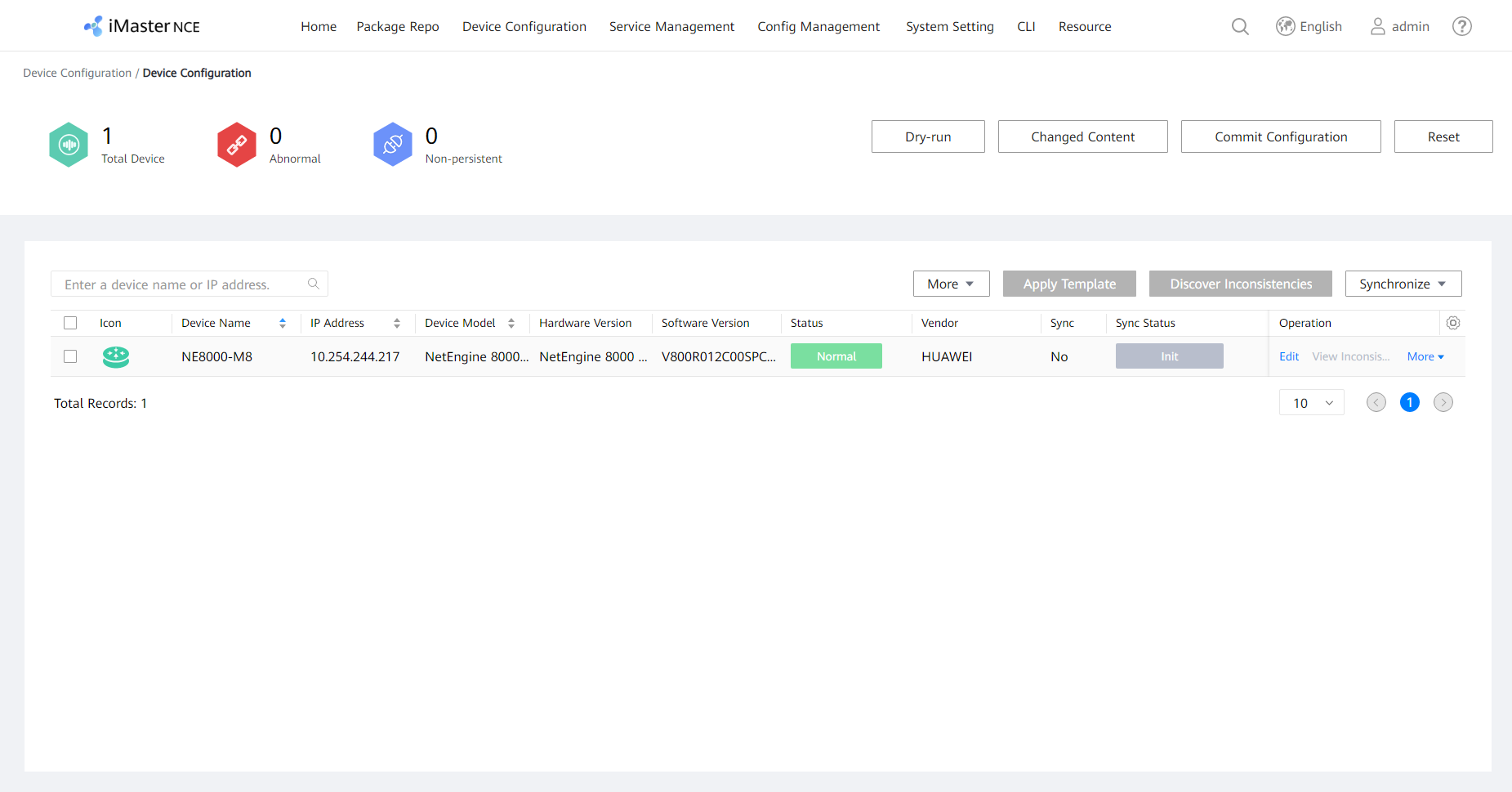


Click **Create**. On the **Create NE** page that is displayed, enter basic information. The NE type, software version, and vendor must be the same as those in the SND package. The device finds the SND package based on the NE type, software version, and vendor.



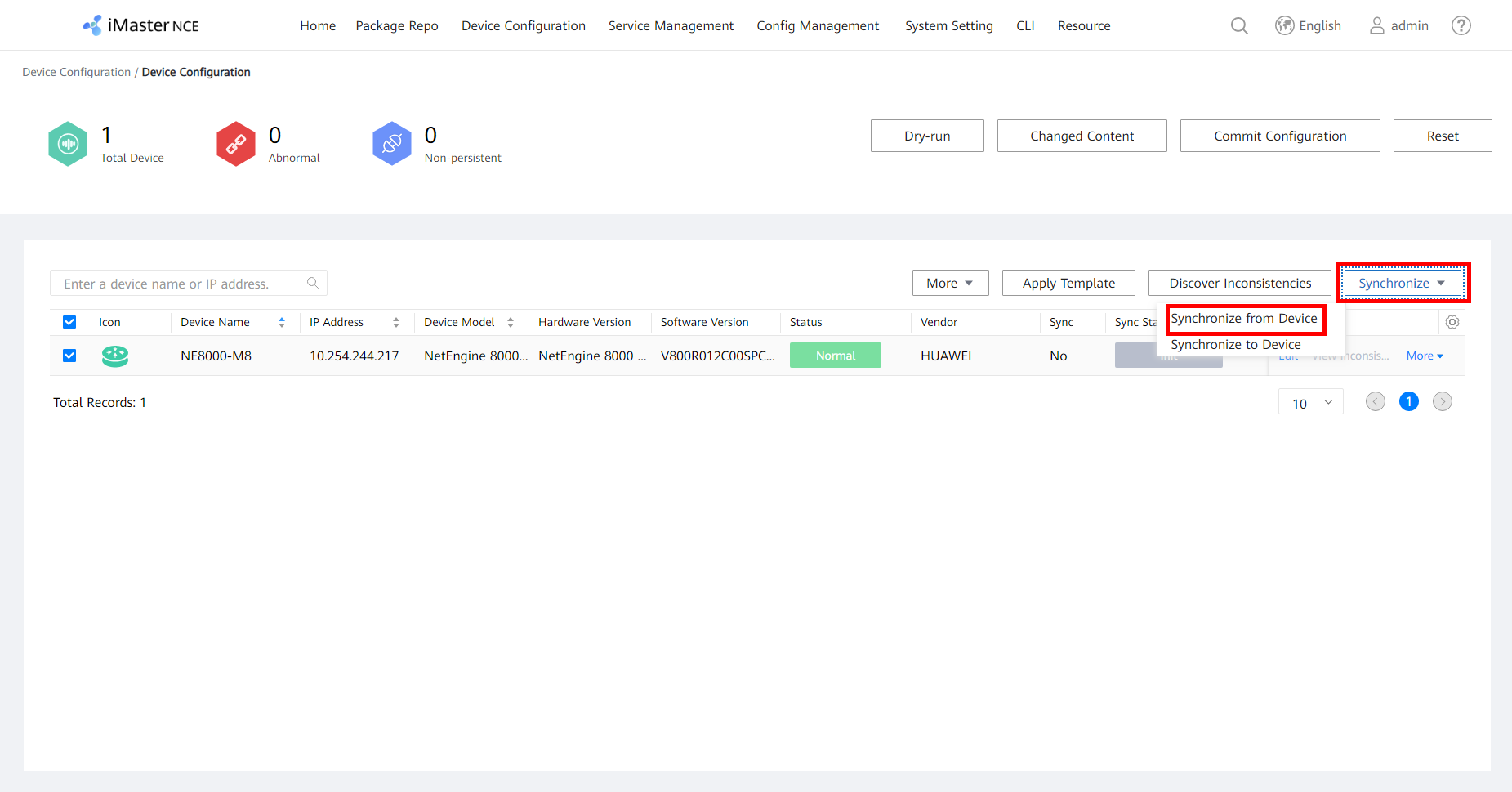
In this example, set the configuration file parameters according to section 1.4.2 Modifying the Package Configuration File.

Choose **Device Configuration** > **Device Configuration** to view the newly added device.

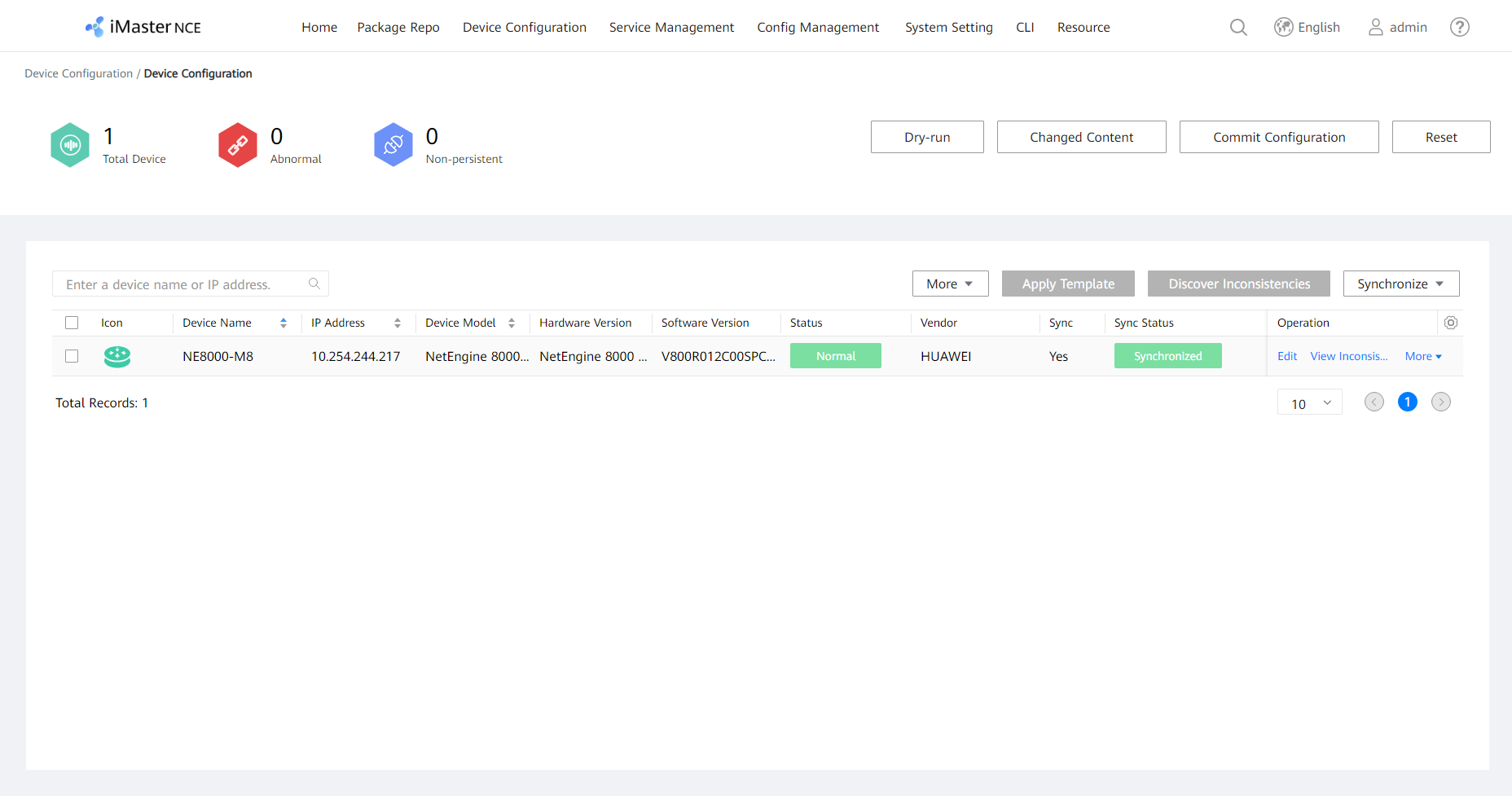


### Synchronizing Device Information

Select the device and click **Synchronize**.



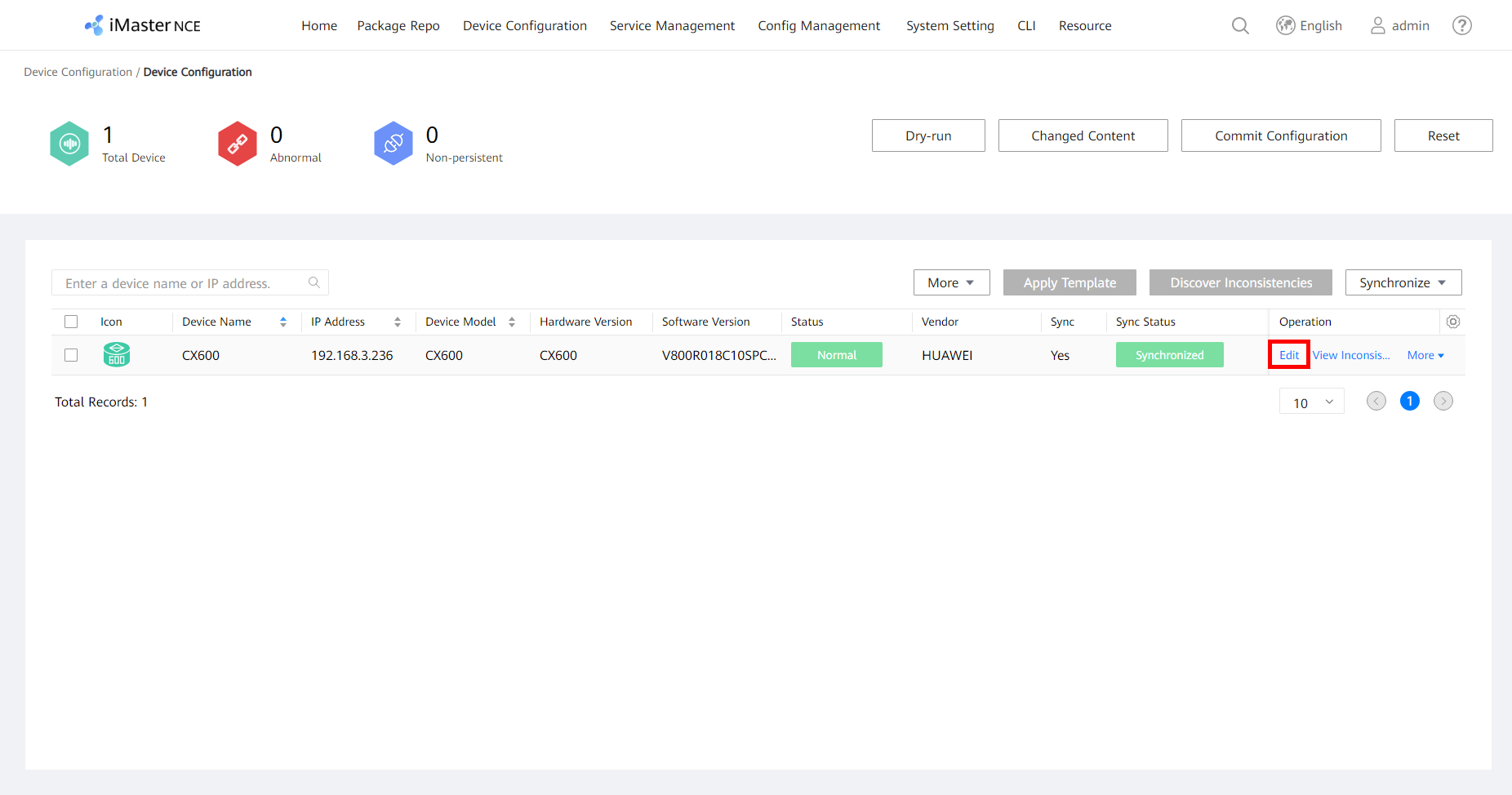
If the value of **Sync Status** is **Synchronized**, the synchronization is successful.

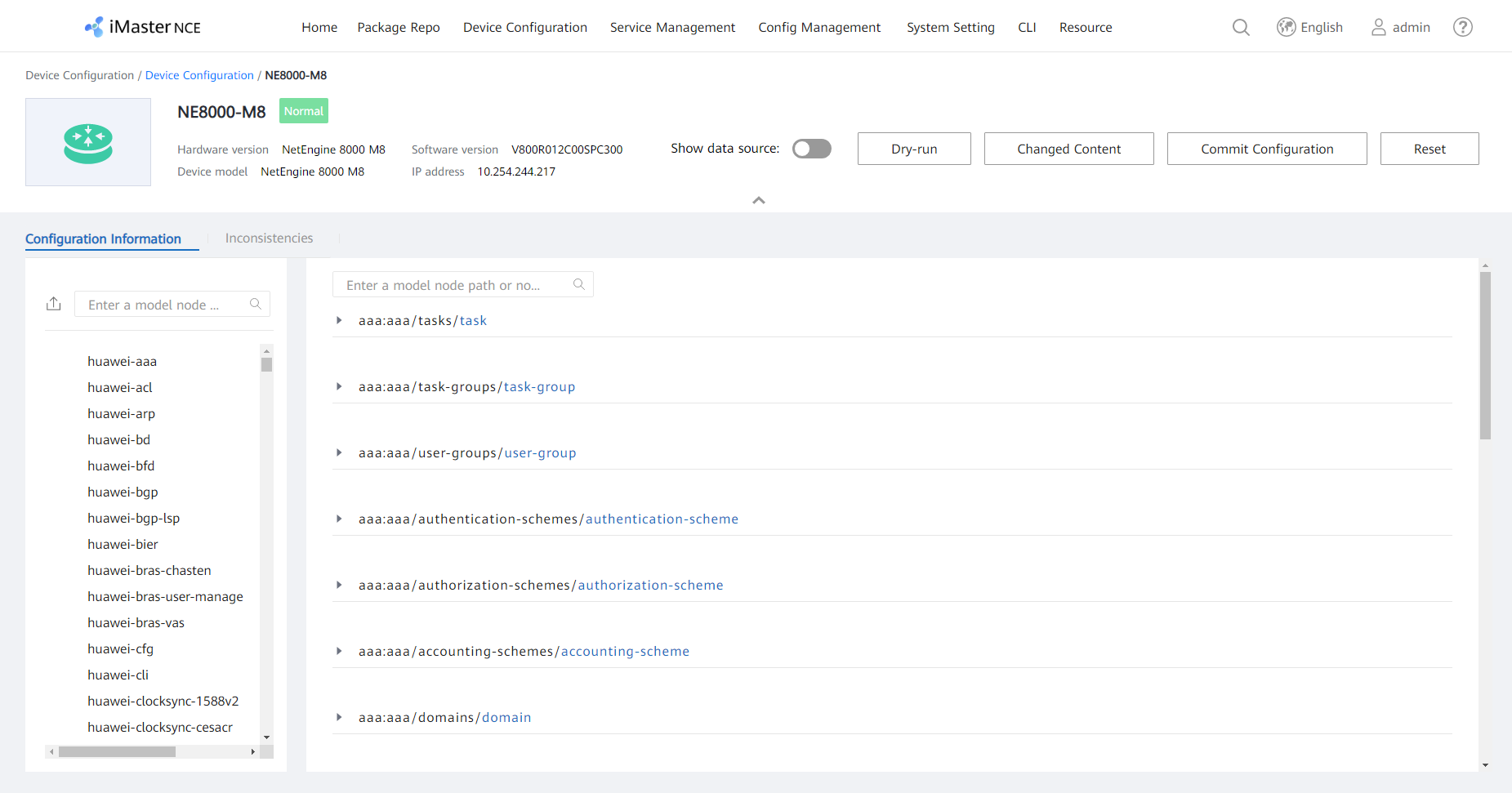


## Delivering Device Configurations on the GUI

After the device is successfully managed, you can deliver configurations of native device capabilities on iMaster NCE. This section uses the procedure for creating a sub-interface on the NE8000 M8 as an example.

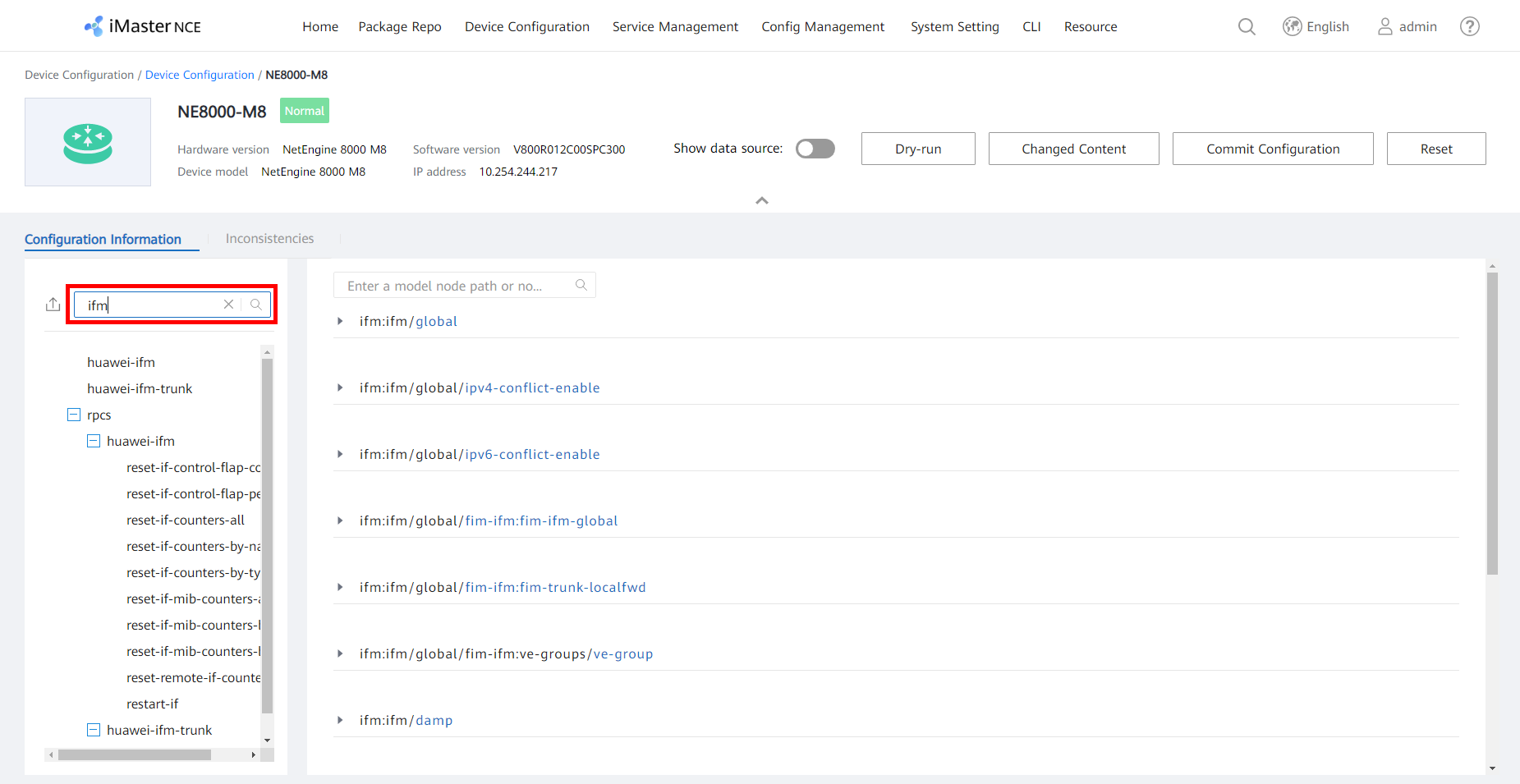
In the device list, click **Edit** in the **Operation** column of the device.



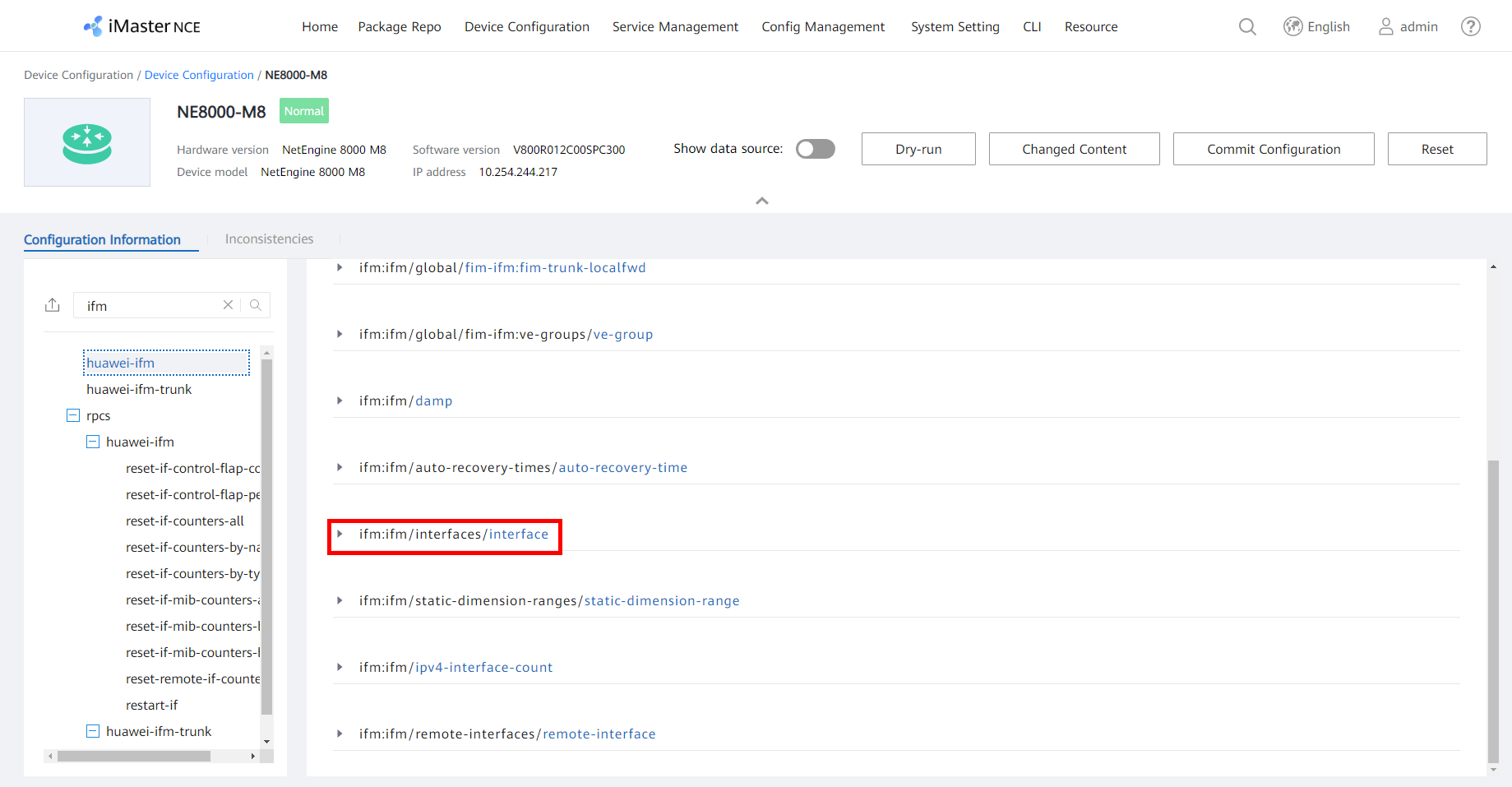


Create a sub-interface.

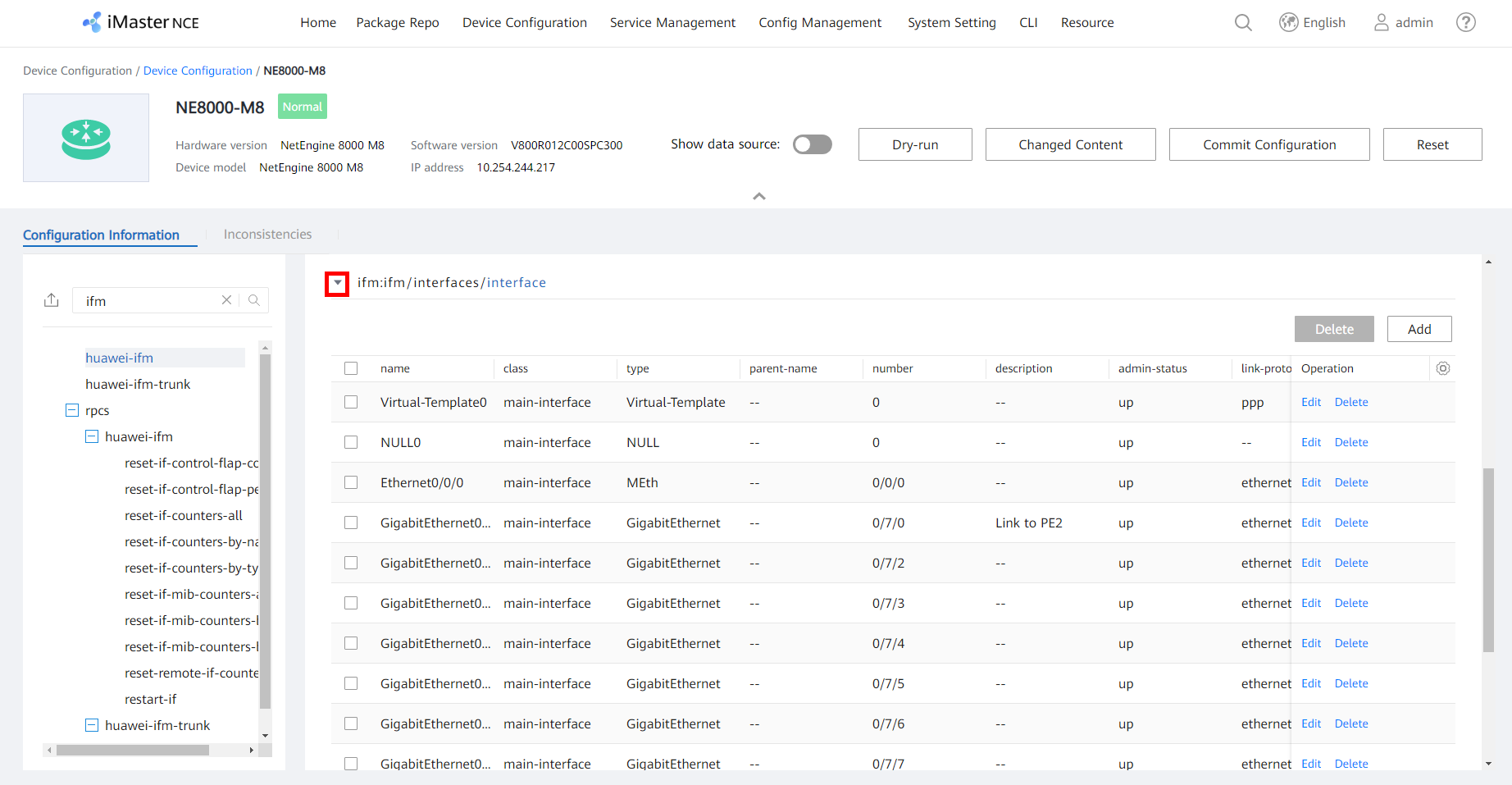
The YANG model file for creating a sub-interface is **huawei-ifm**. Search for **ifm** on the left.



Scroll down to **ifm:ifm/interfaces/interface**.

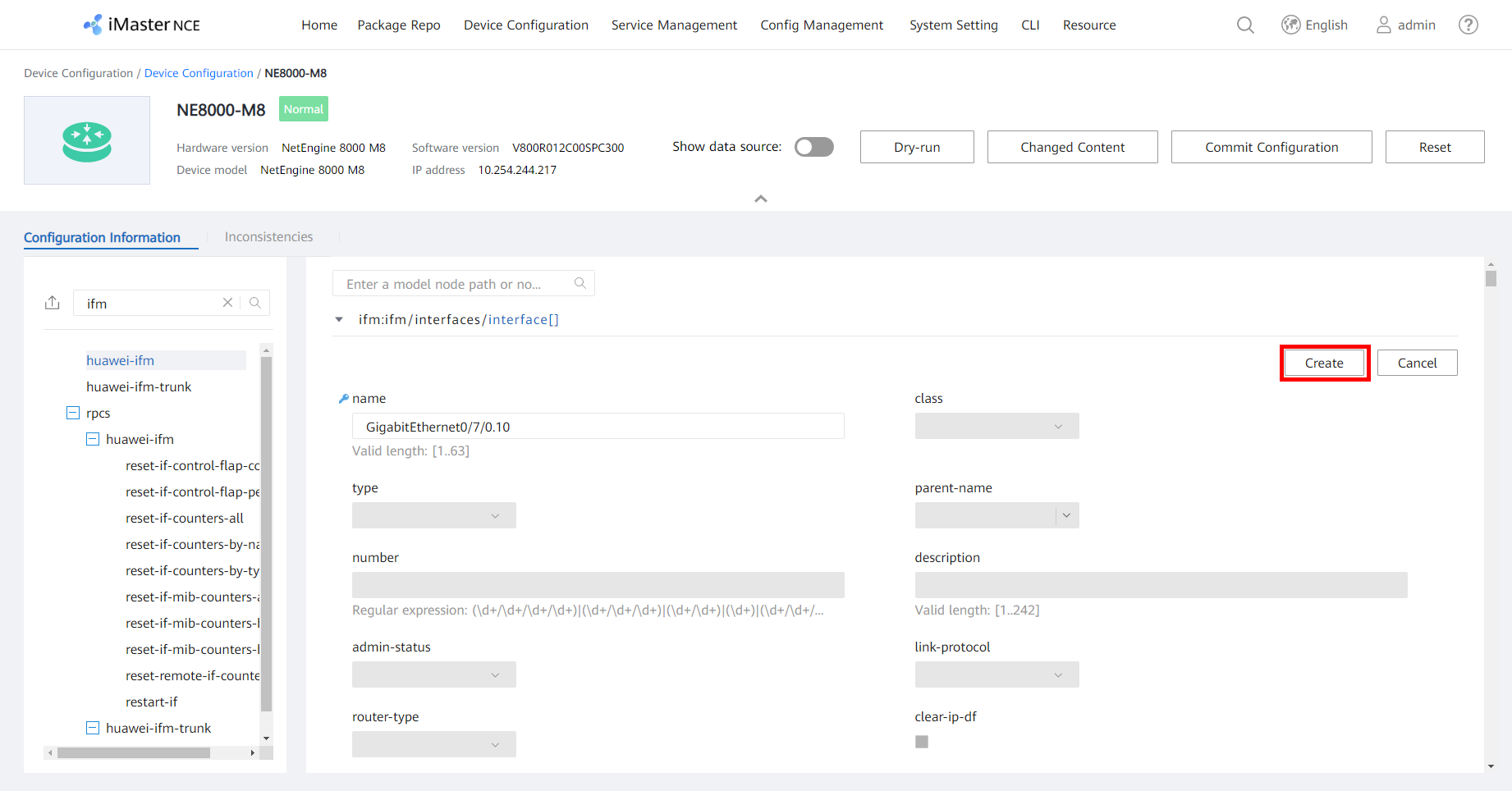


Click the drop-down button on the left. If device information has been synchronized in section 1.5.3 Synchronizing Device Information, the information about all interfaces on the device will be displayed, as shown in the following figure.

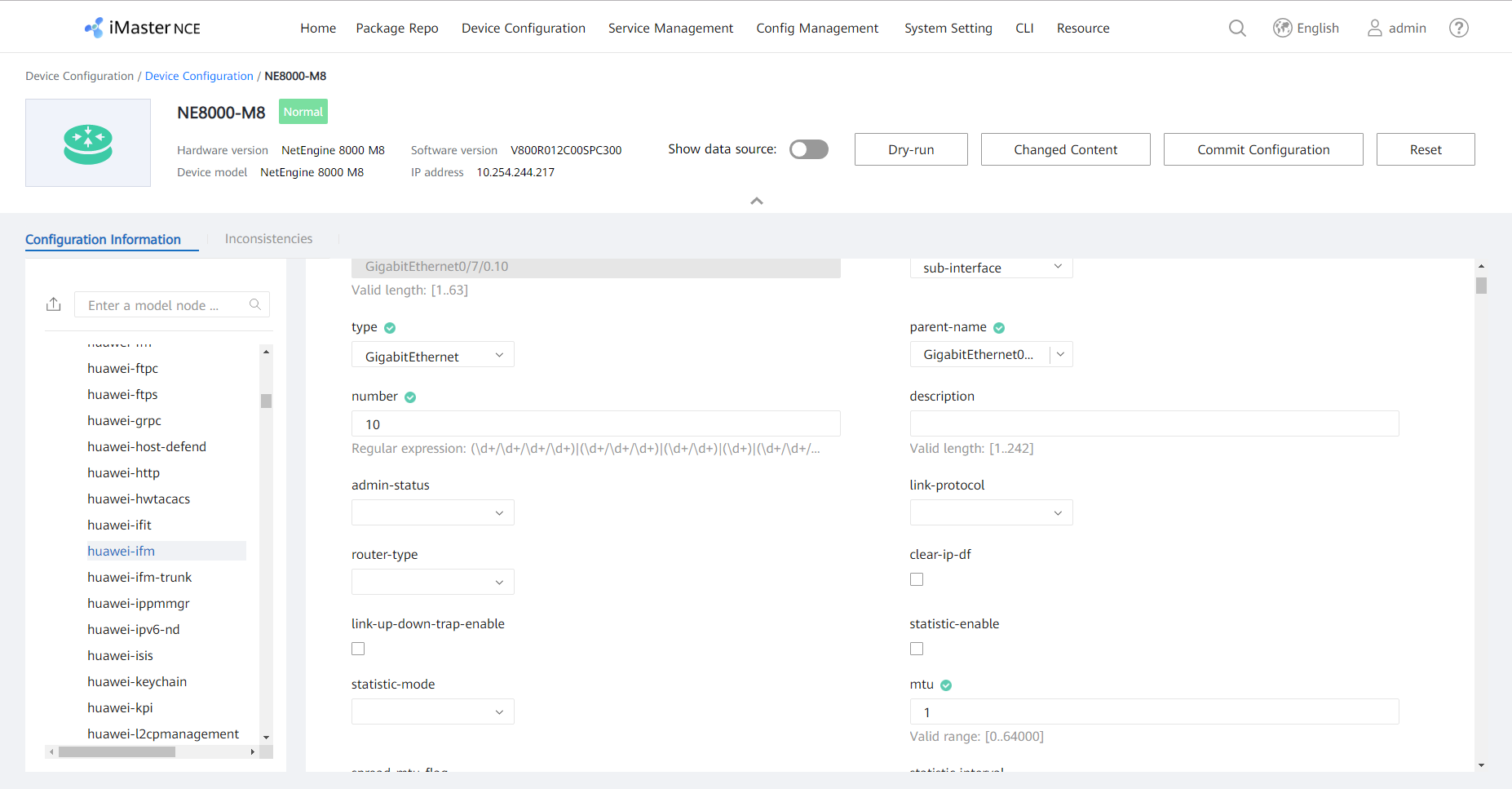


Create a sub-interface, for example, GigabitEthernet0/7/0.10. Click **Add**.

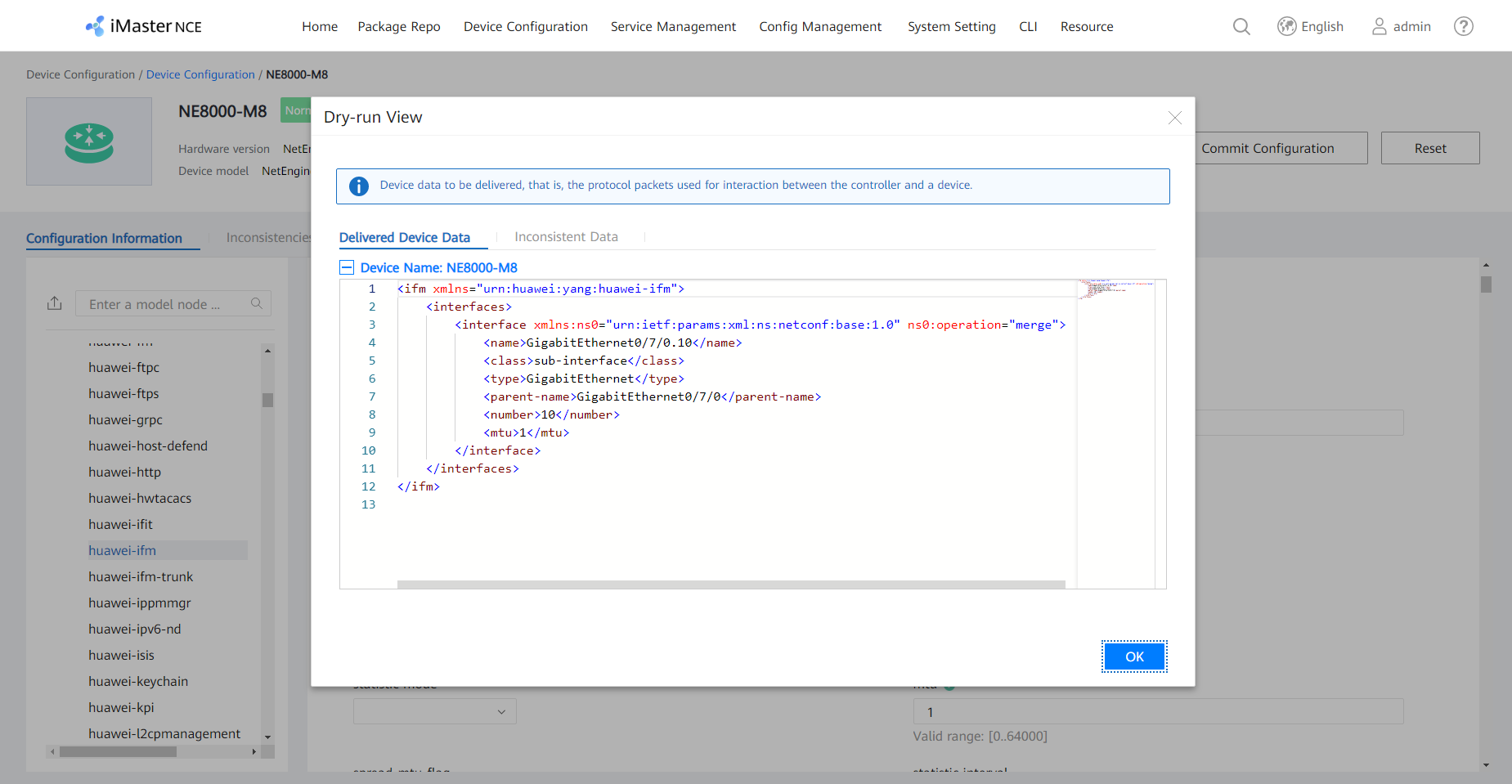
Set **name** to **GigabitEthernet0/7/0.10** and click **Create**.



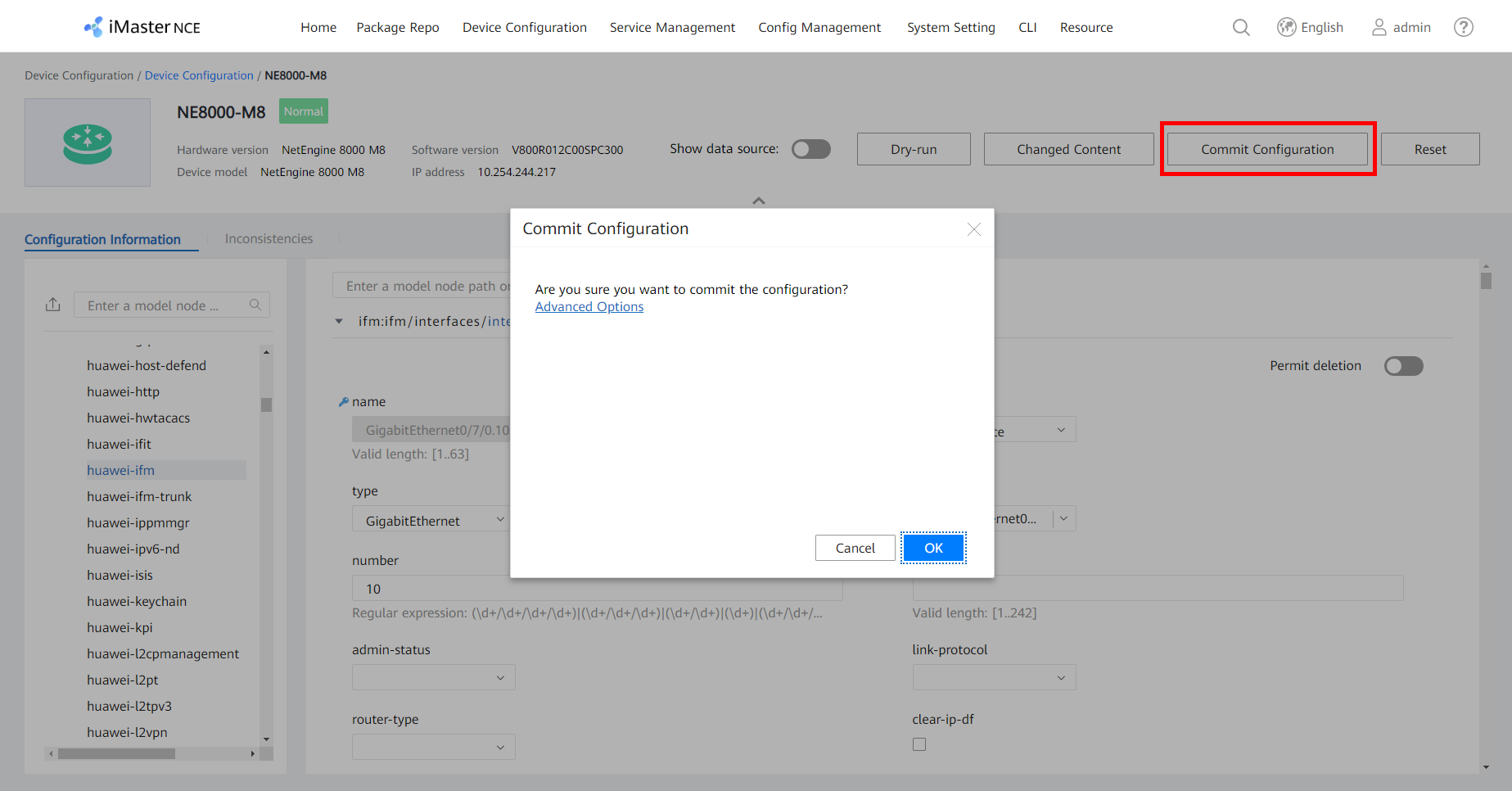
Select **class** to **subinterface**, **type** to **GigabitEthernet**, **parent-name** to **GigabitEthernet0/7/0**, **number** to **10**, and **mtu** to **1**.

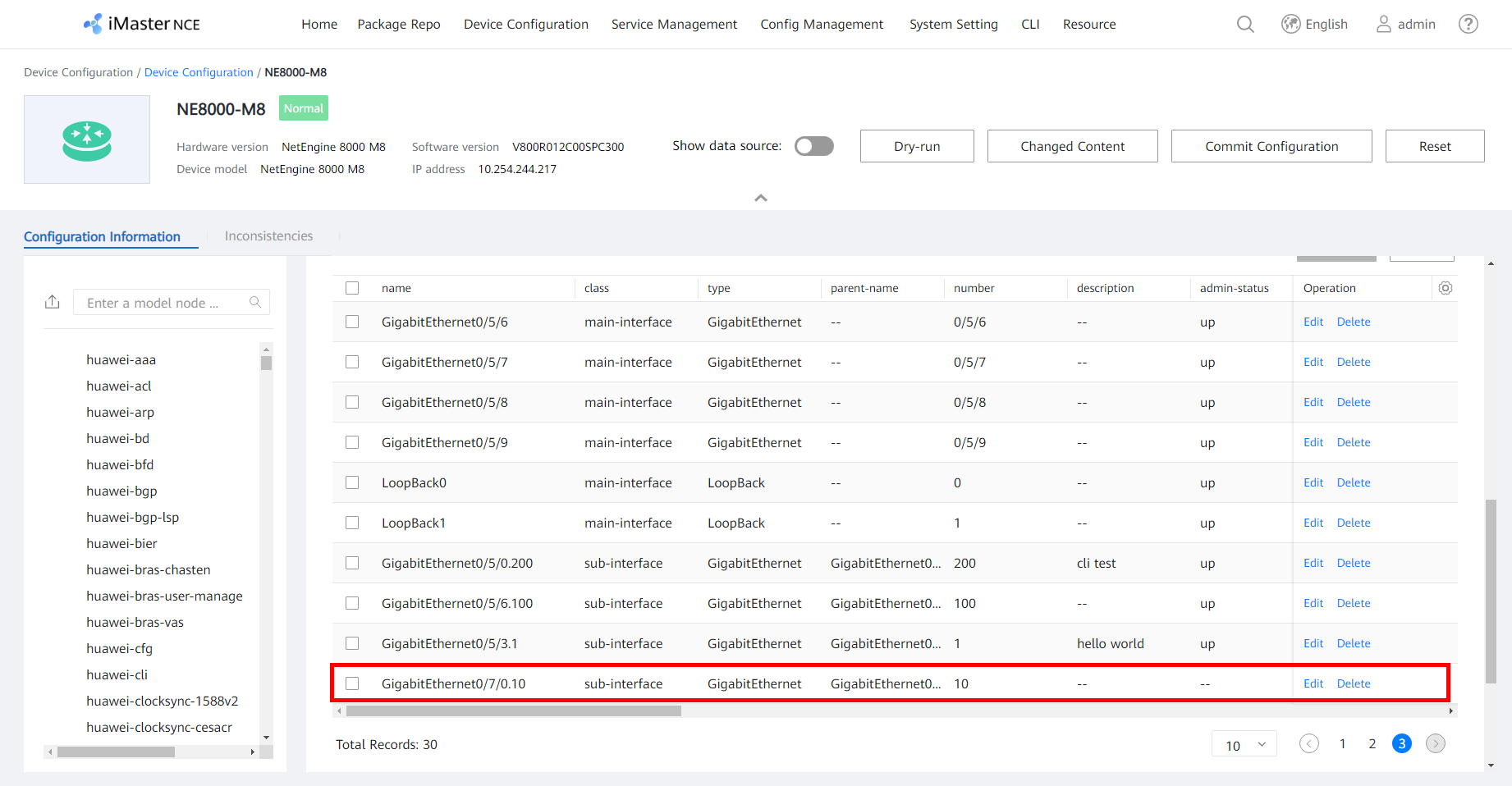


Click **Dry-run** to check the configuration packets to be delivered.



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



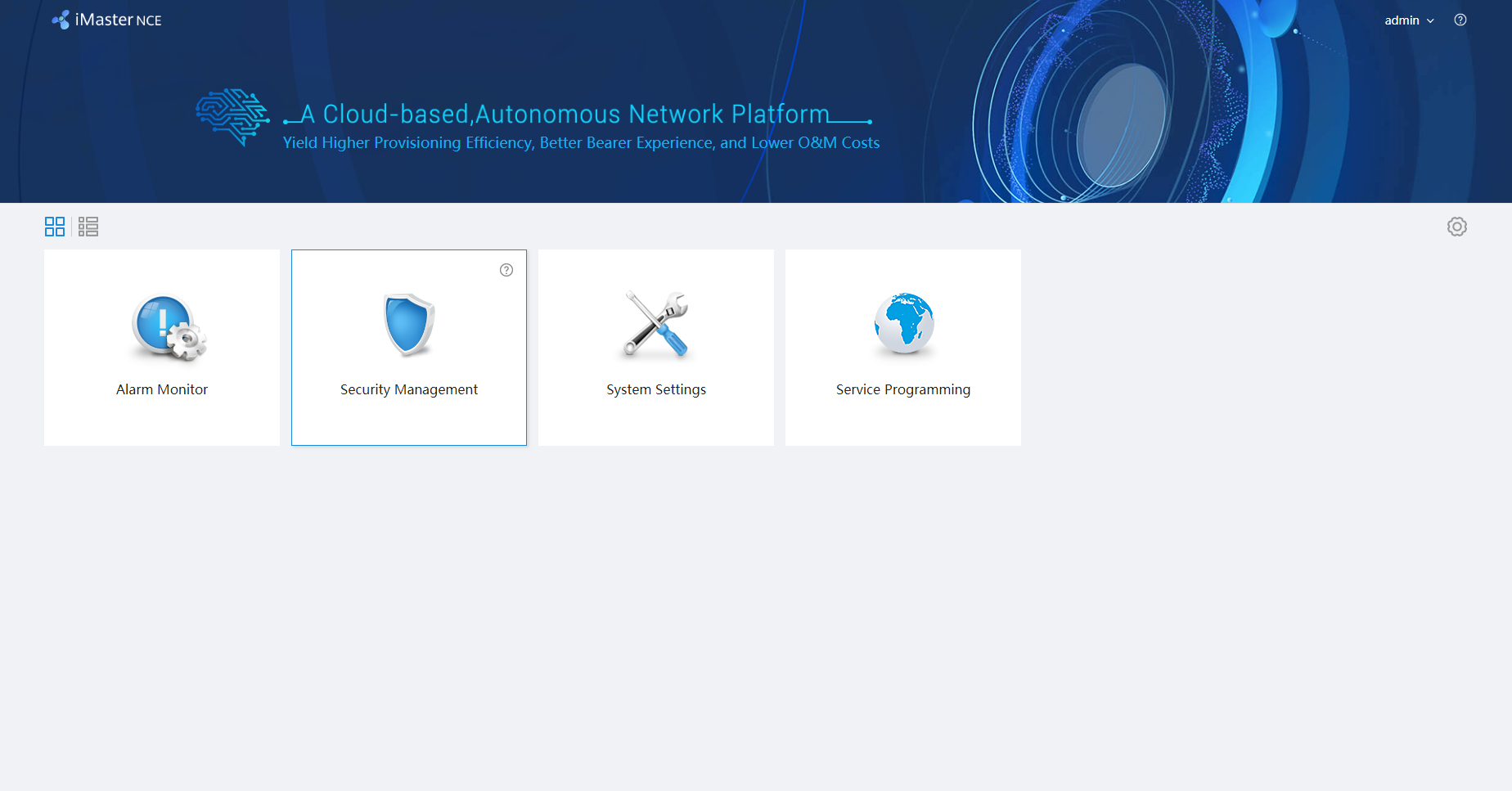


## Delivering Device Configurations Through the Northbound API

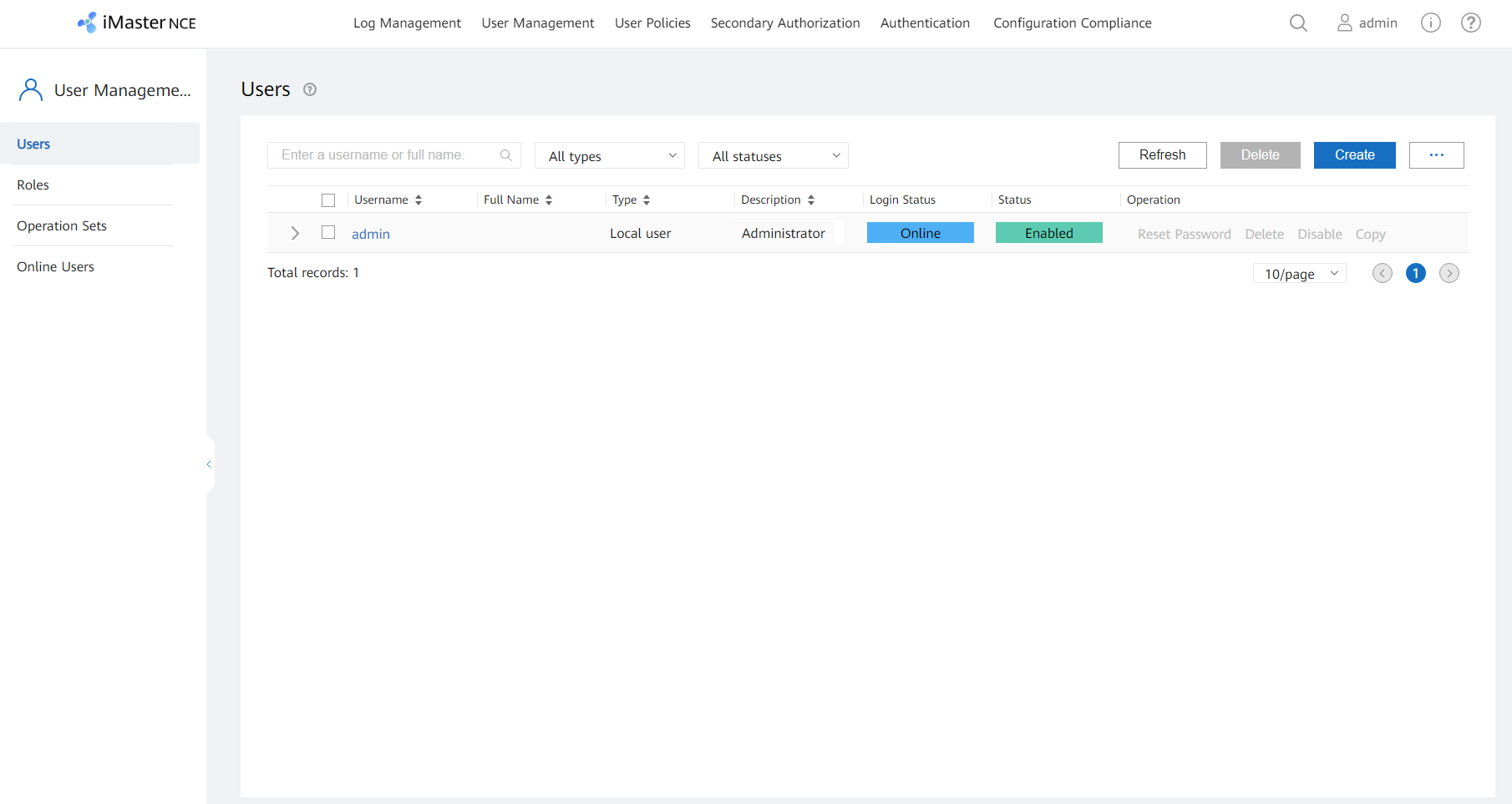
iMaster NCE provides northbound openness capabilities based on native device capabilities. You need to create a northbound user and use the northbound account to implement the add, delete, modify, and query functions. For more iMaster NCE northbound capabilities, see **Document Center** > **API Explorer**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

### Creating a Northbound User

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

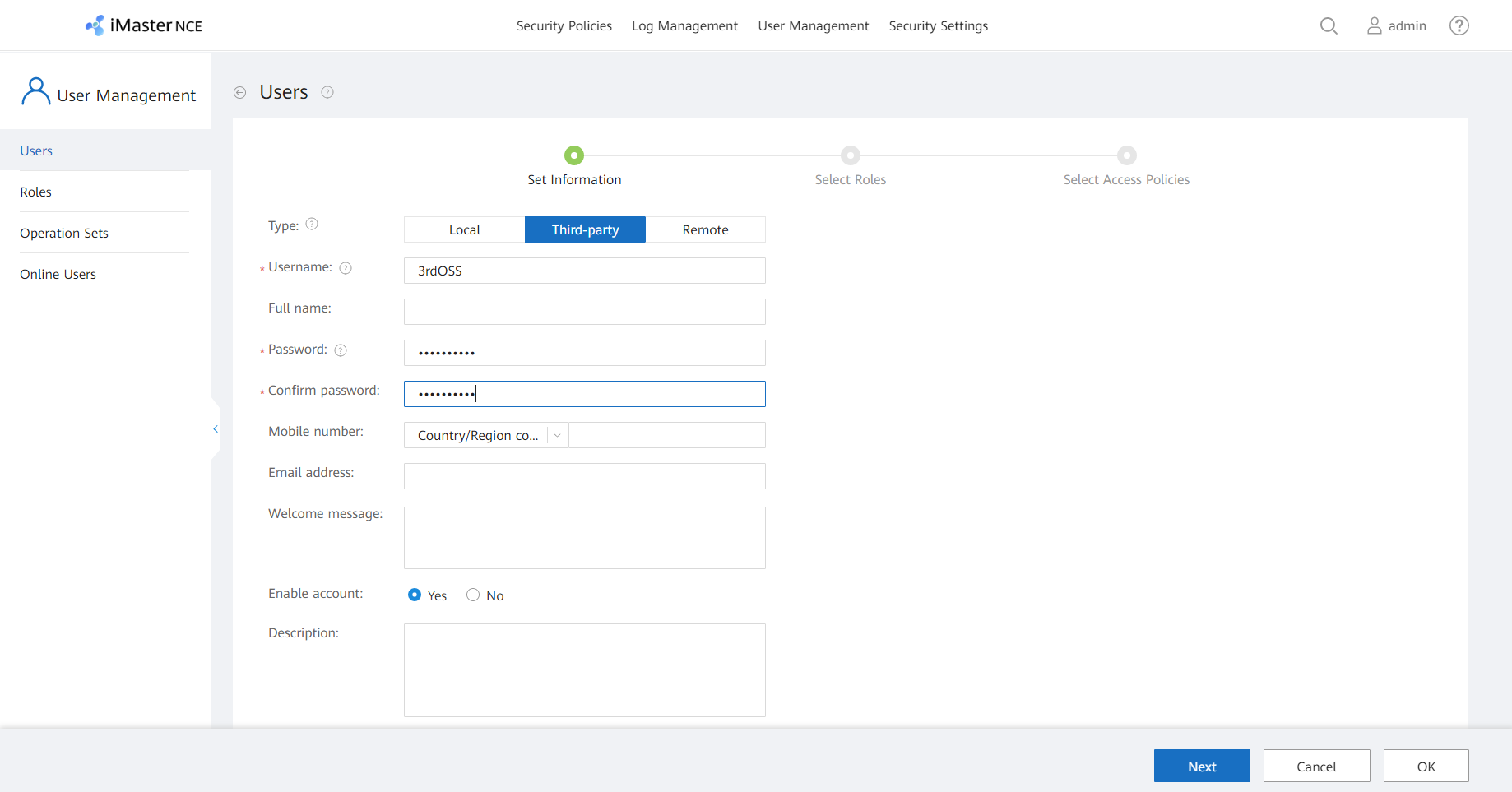


On the **Security Management** page, choose **User Management** and click **Create**.

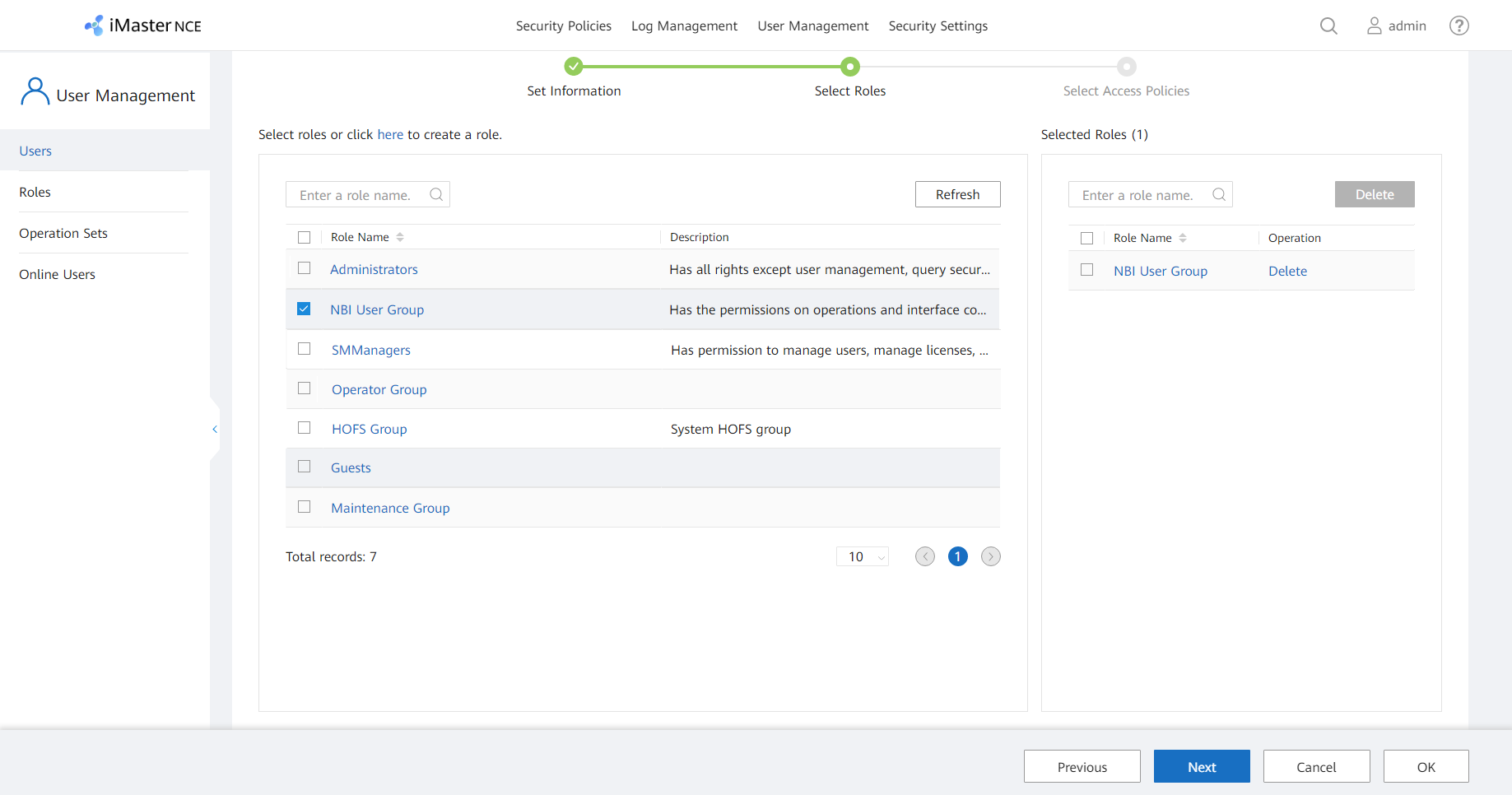


On the page for creating users, select **Third-party**, set the username and password, and click **Next**.

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 |
| 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. A transaction ID needs to be applied for before a service is edited. Transaction management allows committing configuration changes in one atomic transaction so that the iMaster NCE data is consistent with the forwarder data.

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 |
| 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"

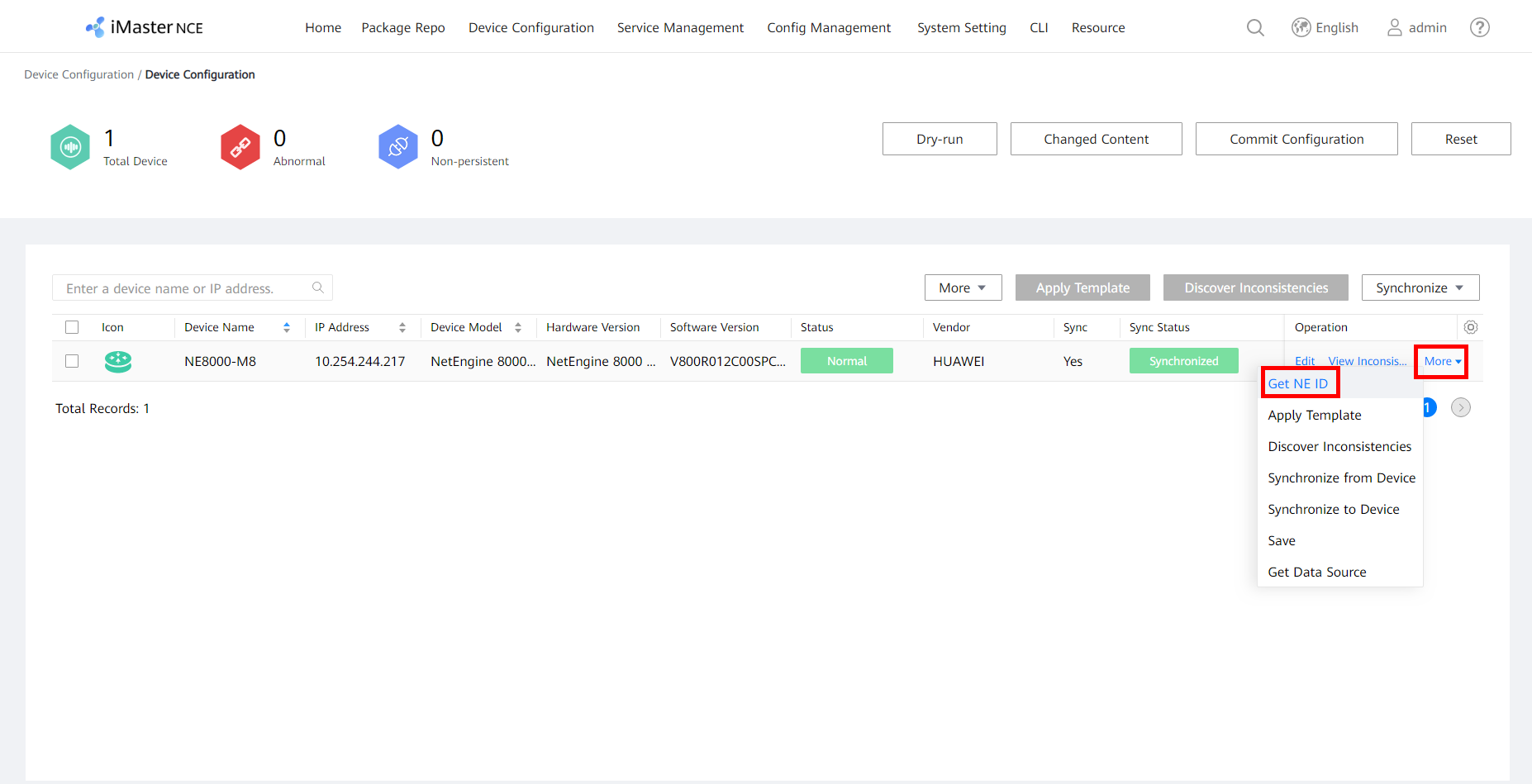
}

}

Edit the configuration to be delivered. (Creating a sub-interface is used as an example.)

Method: POST

In the URL, *{{neId}}* indicates the NE ID. To query the NE ID, locate the NE on the **Device Configuration** page, and choose **More**.



https://{{northIP}}:26335/restconf/v1/data/huawei-ac-nes:inventory-cfg/nes/ne/{{neId}}/huawei-ifm:ifm/interfaces

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 |
| accessSession | {{accessSession}} |

Request body:

{

"interface":[

{

"name":"GigabitEthernet0/7/0.10",

"class":"sub-interface",

"type":"GigabitEthernet",

"parent-name":"GigabitEthernet0/7/0",

"number":"10",

“mtu”:”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 |
| accessSession | {{accessSession}} |

Request body:

{

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

"trans-id":{{transactionsID}}

}

}

Response body:

{

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

"result": true

}

}

Remarks: 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 |
| accessSession | {{accessSession}} |

Request body:

{

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

"trans-id": {{transactionsID}}

}

}

Response body:

{

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

"ne-diff-info": [

{

"ne-id": "678ad4d5-7643-11eb-adae-1e2eb0f53cb9",

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

"diff-infos": [

{

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

"diff-info": "{\"ifm\": {\"interfaces\": {\"interface\": [{\"left\": null, \"right\": {\"[name=GigabitEthernet0/7/0.10]\": {\"name\": \"GigabitEthernet0/7/0.10\", \"class\": \"sub-interface\", \"number\": \"10\", \"type\": \"GigabitEthernet\", \"mtu\": 1, \"parent-name\": \"GigabitEthernet0/7/0\"}}}]}}}"

}

]

}

],

"service-diff-infos": [

{}

]

}

}

After editing the configuration, you can preview the modified contents of the current transaction to check the 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 |
| 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": "678ad4d5-7643-11eb-adae-1e2eb0f53cb9",

"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=\"merge\">\n<name>GigabitEthernet0/7/0.10</name>\n<class>sub-interface</class>\n<type>GigabitEthernet</type>\n<parent-name>GigabitEthernet0/7/0</parent-name>\n<number>10</number>\n<mtu>1</mtu>\n</interface>\n</interfaces>\n</ifm>\n"

]

}

]

},

"diff": {

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

{

"ne-id": "678ad4d5-7643-11eb-adae-1e2eb0f53cb9",

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

"ne-cfg-diffs": [

{

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

"diff-info": "{\"ifm\": {\"interfaces\": {\"interface\": [{\"left\": null, \"right\": {\"[name=GigabitEthernet0/7/0.10]\": {\"name\": \"GigabitEthernet0/7/0.10\", \"class\": \"sub-interface\", \"number\": \"10\", \"type\": \"GigabitEthernet\", \"mtu\": 1, \"parent-name\": \"GigabitEthernet0/7/0\"}}}]}}}"

}

]

}

]

},

"mapconf": {}

}

}

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 |
| accessSession | {{accessSession}} |

Request body:

{

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

"force":false,

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

"no-network":true

}

}

Response body:

{

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

"result": true,

"reason": ""

}

}

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 configuration. (The interface contains special characters, which need to be escaped. Change GigabitEthernet0/7/0.10 to GigabitEthernet0%2f 7%2f 0.10).

Method: GET

https://{{northIP}}:26335/restconf/v1/data/huawei-ac-nes:inventory-cfg/nes/ne/{{neId}}/huawei-ifm:ifm/interfaces/interface/GigabitEthernet0%2f7%2f0.10

Request header:

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

Response body:

{

"interface": [

{

"name": "GigabitEthernet0/7/0.10",

"class": "sub-interface",

"number": "10",

"type": "GigabitEthernet",

"mtu": 1,

"parent-name": "GigabitEthernet0/7/0"

}

]

}

### 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 |
| 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. (The interface contains special characters, which need to be escaped. Change GigabitEthernet0/7/0.10 to GigabitEthernet0%2f 7%2f 0.10). Change the MTU to **2**.

Method: PUT

https://{{northIP}}:26335/restconf/v1/data/huawei-ac-nes:inventory-cfg/nes/ne/{{neId}}/huawei-ifm:ifm/interfaces/interface/GigabitEthernet0%2f7%2f0.10

Request header:

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

Request body:

{

"interface":[

{

"name":"GigabitEthernet0/7/0.10",

"class":"sub-interface",

"type":"GigabitEthernet",

"parent-name":"GigabitEthernet0/7/0",

"number":"10",

"mtu":"2"

}

]

}

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 |
| 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 |
| accessSession | {{accessSession}} |

Request body:

{

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

"trans-id": {{transactionsID}}

}

}

Response body:

{

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

"ne-diff-info": [

{

"ne-id": "678ad4d5-7643-11eb-adae-1e2eb0f53cb9",

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

"diff-infos": [

{

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

"diff-info": "{\"ifm\": {\"interfaces\": {\"interface\": [{\"[name=GigabitEthernet0/7/0.10]\": {\"name\": \"GigabitEthernet0/7/0.10\", \"mtu\": {\"left\": 1, \"right\": 2}}}]}}}"

}

]

}

],

"service-diff-infos": [

{}

]

}

}

After editing the configuration, you can preview the modified contents of the current transaction to check the 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 |
| 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": "678ad4d5-7643-11eb-adae-1e2eb0f53cb9",

"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=\"merge\">\n<name>GigabitEthernet0/7/0.10</name>\n<mtu>2</mtu>\n</interface>\n</interfaces>\n</ifm>\n"

]

}

]

},

"diff": {

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

{

"ne-id": "678ad4d5-7643-11eb-adae-1e2eb0f53cb9",

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

"ne-cfg-diffs": [

{

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

"diff-info": "{\"ifm\": {\"interfaces\": {\"interface\": [{\"[name=GigabitEthernet0/7/0.10]\": {\"name\": \"GigabitEthernet0/7/0.10\", \"mtu\": {\"left\": 1, \"right\": 2}}}]}}}"

}

]

}

]

},

"mapconf": {}

}

}

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 |
| accessSession | {{accessSession}} |

Request body:

{

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

"force":false,

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

"no-network":true

}

}

Response body:

{

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

"result": true,

"reason": ""

}

}

### 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 generated in step 1:

| Key | Value |
| --- | --- |
| Content-Type | application/json |
| Accept | application/json |
| 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 the configuration. (The interface contains special characters, which need to be escaped. Change GigabitEthernet0/7/0.10 to GigabitEthernet0%2f 7%2f 0.10).

Method: DELETE

https://{{northIP}}:26335/restconf/v1/data/huawei-ac-nes:inventory-cfg/nes/ne/{{neId}}/huawei-ifm:ifm/interfaces/interface/GigabitEthernet0%2f7%2f0.10

Request header:

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

Status code: 200