TECOPS-2201

From Zero to Hero:

Cisco Network Services Orchestrator (NSO)

Intermediate

Sofia Athanasiou - Customer Success Specialist

Hector Oses – Customer Delivery Software Architect

Table of Contents

[Overview 3](#_Toc157866926)

[Learning Objectives 3](#_Toc157866927)

[Disclaimer 4](#_Toc157866928)

[NSO Overview 4](#_Toc157866929)

[Scenario 5](#_Toc157866930)

[Network Diagram 5](#_Toc157866931)

[Lab environment 6](#_Toc157866932)

[Lab Introduction and Verification 6](#_Toc157866933)

[Task 1: Configure build-in High Availability 8](#_Toc157866934)

[Step 1: Verifications before configuring High Availability 9](#_Toc157866935)

[Step 2: Enable High Availability 9](#_Toc157866936)

[Step 3: Verify High-Availability 9](#_Toc157866937)

[Task 2: Layered Services Architecture 10](#_Toc157866938)

[Task 3: Kickers and Subscribers 11](#_Toc157866939)

[Task 4: Nano-services 11](#_Toc157866940)

[Summary 11](#_Toc157866941)

[Appendix A 12](#_Toc157866942)

[Use Makefile to configure netsim devices 12](#_Toc157866943)

[Appendix B 12](#_Toc157866944)

[Start NSO and verify status 12](#_Toc157866945)

[Related Sessions at Ciscolive 14](#_Toc157866946)

Overview

Network operators and service providers today are struggling to control the difference between the growth of their operating costs and their revenue. Introduction and deployment of new services is much slower compared to service demand and availability on the market. It is because of inadequate provisioning processes where services are either configured manually or hard coded inside the Operations Support Systems (OSS). Cisco Network Service Orchestrator (NSO) is the answer to the above challenge. NSO architecture decouples network services from specific components, while automatically configuring the network according to the service specifications. NSO enabled by NETCONF and YANG models, enables operators to dynamically adopt the service configuration solution according to changes in the offered service portfolio.

This session is intended to familiarize the novice NSO user with the architecture and capabilities of the platform, touching standards utilized by NSO, such as NETCONF and YANG. The session will further discuss NSO components, service and device abstraction, integration with northbound systems via Application Programming Interfaces (APIs), communication procedure with southbound devices via Network Element Drivers (NEDs), configuration compliance, and configuration data collection.

Learning Objectives

Upon completion of this lab, you will be able to:

* Configure NSO build-in High Availability,
* Scale your services by using Layered Services Architecture,
* Configure kickers and use them inside your service,
* Use nano-services to create l3mplsvpn service.

The lab has 4 main excercises:

* **Exercise 1:** Configure build-in High Availability. It will help you get familiar with the concept of High-Availability and its benefits.
* **Exercise 2:** Create a service using Layered Service Architecture.
* **Exercise 3:** Configure a service taking advantage of FASTMAP using kickers.
* **Exercise 4:** L3MPLSVPN service creation usingnano-services.

Disclaimer

This training document is to familiarize with Cisco NSO intermediate topics for Automating your network. Although the lab design and configuration examples could be used as a reference, it’s not a real design, thus not all recommended features are used, or enabled optimally. For the design related questions please contact your representative at Cisco, or a Cisco partner.

NSO Overview

Cisco® Network Services Orchestrator (NSO) enabled by Tail-f® is an industry-leading orchestration platform for hybrid networks. It provides comprehensive lifecycle service automation to enable you to design and deliver high-quality services faster and more easily.

Figure : NSO high-level architecture



OSS

Service Order

Minimal Device Reconfigurations

**NSO**

The network is a foundation for revenue generation. Therefore, service providers must implement network orchestration to simplify the entire lifecycle management for services. For today’s virtualized networks, this means transparent orchestration that spans multiple domains in your network and includes network functions virtualization (NFV) and software-defined networking (SDN) as well as your traditional physical network and all its components.

NSO is a model driven (YANG) platform for automating your network orchestration. It supports multi-vendor networks through a rich variety of Network Element Drivers (NEDs).

We support the process of validating, implementing and abstracting your network config and network services, providing support for the entire transformation into intent-based networking.

Scenario

There are two servers with NSO version 6.1.5 installed that are part of the same subnet. NSO is running 4 simulated devices taking the roles of Provider Edge in the network. Two of them run Cisco IOS, and the other two Cisco IOS XR.

Network Diagram

A diagram of a cloud computing system

Description automatically generated

Figure : Network Diagram

**Table 1:** Lab first aid

|  |  |  |  |
| --- | --- | --- | --- |
| Host name | IP address | Username | Password |
| NSO-01 server | 198.18.134.28 | cisco | C1sco12345 |
| NSO-02 server | 198.18.134.29 | cisco | C1sco12345 |
| PE\_00 | 127.0.0.1 | admin | admin |
| PE\_01 | 127.0.0.1 | admin | admin |
| PE\_11 | 127.0.0.1 | admin | admin |
| PE\_10 | 127.0.0.1 | admin | admin |

Lab environment

The lab runs inside dCloud in a Windows machine. The 2 available NSOs are installed in Linux hosts and can be reached through SSH, GUI and some APIs (RESTCONF will be used) from the windows machine. Ways of development possible for this lab.

1. (Preferred) Visual Studio Code. You can find a shortcut in the desktop to this application. When you start it, you will be connected to NSO 1 server, and you will be able to view and edit files from your local Windows. A Terminal is available as well. See Appendix A for more information.
2. A computer screen with a lightning bolt

   Description automatically generatedA computer screen with a lightning bolt

   Description automatically generatedConnect to NSO host through putty and edit the files directly there by ‘vim’. Desktop shortcut available.

Lab Introduction and Verification

To make sure you are using the latest code and guides, with the least bugs, make sure you go a git pull from Windows Git Bash and from both NSOs through Putty session.

Windows Git Pull

Open Git Bash application from the desktop, execute the below commands and if you see any errors reach to the proctor:

cd /c/dcloud/CLEMEA24\_AMS\_TECOPS-2201/CLEMEA23\_TECNMS-4175

git pull

NSO 1 and NSO 2 Git Pull

**NOTE:** password for nsoadmin is nsoadmin

Only in NSO2. Open Putty session "NSO 2 Host"

sudo mkhomedir\_helper nsoadmin

sudo usermod -aG cisco nsoadmin

su nsoadmin

echo 'source /opt/ncs/current/ncsrc' >> ~/.bashrc

source /opt/ncs/current/ncsrc

In both NSO1 and NSO2. Open Putty session "NSO 2 Host"

cd CLEMEAR20\_TECNMS-4175/

sudo git config --global --add safe.directory /home/cisco/CLEMEAR20\_TECNMS-4175

sudo git pull

sudo chown -R nsoadmin:ncsadmin /home/cisco/CLEMEAR20\_TECNMS-4175

su nsoadmin

git pull

rm /var/opt/ncs/Makefile

cp /home/cisco/CLEMEAR20\_TECNMS-4175/Makefile /var/opt/ncs/

cp -r /home/cisco/CLEMEAR20\_TECNMS-4175/config\_examples /var/opt/ncs/

cp -r /home/cisco/CLEMEAR20\_TECNMS-4175/TECOPS-2201\* /var/opt/ncs

Close the Putty session and open now "NSO 2 Host" and execute the same commands

NSO and Virtual Devices Verification

The NSO version 6.1.5 is already installed and the required Network Element Drivers (NEDS) are loaded.

Desktop shortcut ‘NSO 1 Host’ and ‘NSO 2 Host’ allows you to connect to the Linux host where NSO is running as user ‘cisco’.

1. Change user to both NSOs be using nsoadmin user:

cisco@nso1:~$su nsoadmin

Password: nsoadmin

nsoadmin@nso1:/home/cisco$

1. Verify that the 4 netsim devices are up and running:

nsoadmin@nso1:/home/cisco$ cd /var/opt/ncs

nsoadmin@nso1:/var/opt/ncs$ ncs-netsim is-alive

DEVICE PE\_00 OK

DEVICE PE\_01 OK

DEVICE PE\_10 OK

DEVICE PE\_11 OK

If you are facing issues, follow Appendix A on how to bring back up the netsim devices. From same directory (/var/opt/ncs) use Makefile:

nsoadmin@nso1:/var/opt/ncs$ make rebuild-netsim

1. Access to NSO CLI. There are different ways to connect to NSO-01 CLI and to NSO-02 CLI.
   1. Double click on the icon shown here to access: A computer screen with a lightning bolt

      Description automatically generated
   2. Run from the previous terminal ‘ncs\_cli -u nsoadmin’ (default password for user admin is admin)

nsoadmin@nso1:/var/opt/ncs$ ncs\_cli –u nsoadmin

nsoadmin connected from 127.0.0.1 using console on ubuntu

nsoadmin@ncs>

* 1. When enabled in ncs.conf file, NSO allows direct access to CLI through SSH connection. Example:

nsoadmin@nso1:/var/opt/ncs$ ssh -l nsoadmin -p 2024 localhost

The authenticity of host '[localhost]:2024 ([127.0.0.1]:2024)' can't be established.

RSA key fingerprint is SHA256:nzmXDxz2gP7F8r5OYnz2d6OI20uwoHTRw+sstvftHI8.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added '[localhost]:2024' (RSA) to the list of known hosts.

nsoadmin@localhost's password:

nsoadmin connected from 127.0.0.1 using ssh on ubuntu

nsoadmin@nso1>

1. Cisco NSO allows 2 types of CLI to interact with it. During this workbook we will use Cisco based CLI. To use Cisco based CLI you can do it in 2 ways:
   1. You can run the command above and then ‘switch cli’.

nsoadmin@nso1:/var/opt/ncs$ **ncs\_cli –u nsoadmin**

admin connected from 127.0.0.1 using console on ubuntu

nsoadmin@nso1> **switch cli**

nsoadmin@nso1#

* 1. Run the above command with the additional option -C for cisco

nsoadmin@nso1:/var/opt/ncs $ **ncs\_cli –Cu nsoadmin**

nsoadmin connected from 127.0.0.1 using console on ubuntu

nsoadmin@ncs#

1. Let’s verify the 4 netsim devices are loaded into NSO

nsoadmin@nso1# **show devices list**

NAME ADDRESS DESCRIPTION NED ID ADMIN STATE

--------------------------------------------------------

PE\_00 127.0.0.1 - cisco-ios unlocked

PE\_01 127.0.0.1 - cisco-ios unlocked

PE\_10 127.0.0.1 - cisco-ios-xr unlocked

PE\_11 127.0.0.1 - cisco-ios-xr unlocked

1. Verify the required packages are loaded

nsoadmin@nso1# **show packages package package-version**

PACKAGE

NAME VERSION

------------------------------

cisco-ios-cli-6.100 6.100.9

cisco-iosxr-cli-7.53 7.53

juniper-junos-nc-4.14 4.14.3

l3mplsvpn 1.0

loopbackbasic 1.0

nsoadmin@nso1# **show packages package oper-status**

PACKAGE

PROGRAM META FILE

CODE JAVA BAD NCS PACKAGE PACKAGE CIRCULAR DATA LOAD ERROR

NAME UP ERROR UNINITIALIZED VERSION NAME VERSION DEPENDENCY ERROR ERROR INFO

---------------------------------------------------------------------------------------------------------

cisco-ios-cli-6.100 **X** - - - - - - - - -

cisco-iosxr-cli-7.53 **X** - - - - - - - - -

- -

Juniper-junos-nc- 4.14 X - - - - - - - - -

l3mplsvpn X - - - - - - - - -

loopbackbasic X - - - - - - - - -

1. Visual Studio code will be connected already with NSO-01, the window will be open and corresponding to the nso running directory. That way you do not need to change or create files in the terminal of the NSO-01 though through Visual Studio Code.
2. So that you obtain the latest code from lab, you need to pull the repository that is under /home/cisco directory.

nsoadmin@nso1:$ cd /home/cisco/CLEMEAR20\_TECNMS-4175

nsoadmin@nso1:/home/cisco/CLEMEAR20\_TECNMS-4175$ git pull

Task 1: Configure build-in High Availability

There are three different ways that High-Availability can be used:

1. Build-in HA
2. Using Tailf-hcc package and BGP capabilities
3. HA-RAFT

In this lab, you will configure the build-in HA. This type of High-Availability is used when NSO-Primary and NSO-Secondary lay in the same subnet, in other words, when they belong to the same network.

On the contrary, using High-Availability with Tailf-hcc package is used when you want to succeed High-Availability with NSOs that lay either on different networks in the same area either on different networks in different areas (Geo HA). In order to use it, you need to enable and configure BGP and a Virtual IP (VIP) that will be the bash of communication between the two NSO instances.

Lastly, HA-RAFT was introduced in 6.x.x version of NSO and is supporting further with auto-recovery and fully resolving the issues of the Split Brain. In order to configure it, you need more that 2 NSO instances, so far the recommended Cisco approach refers to using 3 NSO instances.

During High-Availability, a NSO node can be in three different modes:

1. Primary
2. Secondary
3. None

None is supporting so that there are no additional commits permitted while Primary is Down, that way we are not facing any longer problems of Split Brain.

Another important note is that while configuring the high-availability token, in local install the token between Primary and Secondary might be different. In system install they should be the **same**, otherwise Primary and Secondary will not be able to communicate and form a cluster.

More information regarding configuration of High-Availability of NSO 6.1.5: <https://developer.cisco.com/docs/nso-guides-6.1/#!high-availability> .

Step 1: Verifications before configuring High Availability

Let’s start exploring how to configure build-in high-availability in NSO 6.1.5. Before starting with high-availability configuration. There are some verifications needed:

1. Connect to NSO-01 CLI and verify that high-availability is not enable:
2. Verify the same for NSO-01
3. Verify that both NSOs are having same packages:
4. Verify netsim devices are up and running in NSO-01
5. Verify there are no services configured

Step 2: Enable High Availability

1. Connect to NSO Host 1 and navigate to /etc/ncs directory. There you will find the configuration files of NSO.

cd /etc/ncs

ls

1. Use vi to navigate in ncs.conf file and find ha field:

vi ncs.conf

/<ha>

1. Enable High-Availability by changing value in true
2. Replicate the same for NSO-02
3. Restart NSO-01 and NSO-02
4. Configure High-Availability to NSO-01, that NSO will act as Primary
5. Configure High-Availability to NSO-02, that NSO will act as Secondary
6. Enable High-Availability in NSO-01
7. Enable High-Availability in NSO-02
8. Verify High-Availability is up and running and Primary with Secondary has formed a cluster.

Step 3: Verify High-Availability

1. Enter configuration mode in NSO-02

You should not be able to enter since right now NSO-02 is working as Secondary and has read only abilities.

1. Verify in NSO-02 that there is no loopback interface configured in PE\_00
2. Go back to NSO-01 and configure device PE\_00 with a loopback interface:
3. Verify that loopback interface is configured for PE\_00 in both NSOs:
4. Now configure service loopback through NSO-01 and verify the same will be visible in NSO-02:
5. It is time to bring down NSO-01 and see NSO-02 taking over, stop NSO-01
6. Verify that NSO-02 has taken over
7. Bring back NSO-01 and revert roles :
8. Verify High-Availability is working between the two nodes.
9. To conclude disable HA again from both by changing ncs.conf file and restart both NSOs:

Task 2: Layered Services Architecture

Stacked services design conceptually consists of several services that are divided into customer-facing and resource-facing services. The approach to service development is almost the same as with traditional service development. Now, however, you need to consider multiple services being stacked and integrated together, to present themselves as one large service.

There are two types of services for working with stacked services design. CFSs and resource-facing services (RFSs) are identified as two types of services that are implemented together to achieve service stacking. The customer-facing service is assigned to a customer and can consist of several RFSs, as shown in the following figure.

A diagram of a diagram of a company

Description automatically generated

Figure : Layered Service Architecture – CFS & RFS

Following the figure above, NSO-01 with be responsible for the CFS to RFS information mapping and later on RFS will retrieve that information and will be responsible for the RFS to device configuration mapping.

More information regarding Layered Service Architecture : <https://developer.cisco.com/docs/nso-guides-6.1/#!lsa-overview>

The aim of the section is to develop a stacked service called l2vpn-qos.

In this task, the goal is from network service perspective:

* Configure a QoS policy with one class – default – and defined average bit rate shaping. Apply the QoS policy to a GigabitEthernet interface.
* Create an L2VPN VPWS tunnel, for this we will reuse the previously created l2vpn package.

A screenshot of a computer

Description automatically generated

Figure : Task 2 Service Architecture

Step 1: Create QoS Template Service Package

1. Get to packages directory in NSO-01:

nsoadmin@nso1:$ cd /var/opt/ncs/packages

nsoadmin@nso1:/var/opt/ncs/packages$

1. Generate a package with skeleton template-based only:

nsoadmin@nso1:/var/opt/ncs/packages$ ncs-make-package –-service-skeleton template qos

1. Start creating the YANG model by using the following instructions bellow, the YANG model will be in:

nsoadmin@nso1:$cd /var/opt/ncs/packages/qos/src

nsoadmin@nso1: /var/opt/ncs/packages/qos/src$ ls

Makefile yang

nrsoadmin@nso1:/var/opt/ncs/packages/qos/src$ cd yang

**Table 1:** Lab first aid

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter name | Parameter type | Restrictions | mandatory |
| qos | List | No | - |
| Service-id | String | (key to qos list) | - |
| Ce-devices | List | 2 devices always | - |
| Lan-ge-interface | String | - | - |
| Policy-name | String | - | True |
| Average-bit-rate | Uint32 | Multiple of 8000 | True |

module qos {

namespace "http://com/example/qos";

prefix qos;

import tailf-common {

prefix tailf;

}

import ietf-inet-types {

prefix inet;

}

import tailf-ncs {

prefix ncs;

}

description

"TECOPS-2201: QoS service";

revision 2024-02-05 {

description

"Initial revision.";

}

list qos {

uses ncs:service-data;

// the keyword ncs:servicepoint is used to link a yang model to a configuration template or service code

// the service is template-based only. Thus, you will find inside qos-template.xml a link to servicepoint "qos"

ncs:servicepoint "qos";

// service-id identifies the qos service instance

key service-id;

leaf service-id {

// thanks to tailf:info, "Unique service id" description will be printed in the NSO CLI

tailf:info "Unique service id";

tailf:cli-allow-range;

type string;

}

// Exactly 2 devices can be configured by one service instance.

// The minimum number of devices to configure is defined by "min-elements".

// The maximum number of devices to configure is defined by "max-elements".

list ce-devices {

min-elements 2;

max-elements 2;

key ce-device;

tailf:info "device name";

leaf ce-device {

type leafref {

// reference to the list of registered devices in NSO

// in the path the usage of prefix "ncs" is mandatory because the elements "ncs:devices", "ncs:device" and "ncs:name" are not part of the local yang module

path "/ncs:devices/ncs:device/ncs:name";

}

}

// The LAN facing interface that the policy-map will be applied to

leaf lan-ge-interface {

type string;

}

}

// the policy-name to be configured on the devices in the leaf-list

leaf policy-name {

tailf:info "Policy name";

// "mandatory true" means that this leaf must be filled. Otherwise NSO won't accept the service instantiation

mandatory true;

type string;

}

leaf average-bit-rate {

tailf:info "Target Bit Rate (bits per second)";

mandatory true;

type uint32 {

// the predefined type unint32 is locally refined

range "8000..max";

}

}

}

}

1. Define the qos configuration template. Template will be located:

nsoadmin@nso1:$ cd /var/opt/ncs/packages/qos/templates

nsoadmin@nso1: var/opt/ncs/packages/qos/templates$ ls

qos-template.xml

<!-- Here we find the same servicepoint like in the qos yang file-->

<config-template xmlns="http://tail-f.com/ns/config/1.0" servicepoint="qos">

<devices xmlns="http://tail-f.com/ns/ncs">

<?foreach {/ce-devices}?>

<device>

<!--

Select the devices from some data structure in the service

model. In this skeleton the devices are specified in a leaf-list.

Select all devices in that leaf-list:

-->

<name>{ce-device}</name>

<config>

<!-- With namespace "urn:ios" we explicitly define which YANG model NED to be used to configure the equipment -->

<policy-map xmlns="urn:ios">

<!-- Here we specify a variable which will be filled directly from the YANG model -->

<name>{/policy-name}</name>

<class-default>

<class>

<!-- A static parameter -->

<name>class-default</name>

<shape>

<average>

<bit-rate>{/average-bit-rate}</bit-rate>

</average>

</shape>

</class>

</class-default>

</policy-map>

<interface xmlns="urn:ios">

<GigabitEthernet>

<name>{lan-ge-interface}</name>

<service-policy>

<output>{/policy-name}</output>

</service-policy>

</GigabitEthernet>

</interface>

</config>

</device>

<?end?>

</devices>

</config-template>

1. Compile the package:

nsoadmin@nso1:$ cd /var/opt/ncs/packages/qos/src

nsoadmin@nso1: /var/opt/ncs/packages/qos/src$ make clean all

rm -rf ../load-dir  
mkdir -p ../load-dir  
/opt/ncs/ncs-5.7.11/bin/ncsc `ls qos-ann.yang > /dev/null 2>&1 && echo "-a qos-ann.yang"` \

--fail-on-warnings \ \

-c -o ../load-dir/qos.fxs yang/qos.yang

1. Perform packages reload to onboard package to NSO:

nsoadmin@nso1: /var/opt/ncs/packages/qos/src$ ncs\_cli -Cu nsoadmin

nsoadmin@nso1# packages reload

>>> System upgrade is starting.

>>> SessdiColnosud:iTnhecCoinscfoiDguemreo mode must exit to operational mode.

>>> No configuration changes can be performed until upgrade has completed.

>>> System upgrade has completed successfully.

1. Configure a qos service:

admin@ncs#

admin@ncs# **config**

Entering configuration mode terminal

admin@ncs(config)# qos test average-bit-rate 500000 policy-name test-policy ce-devices PE\_00 lan-ge-interface 0/3

admin@ncs(config-ce-devices-CE-A\_Site\_1)# exit admin@ncs(config-qos-CUST-PMAP-S-L2ETH)# ce-devices PE\_10 lan-ge-interface 0/3

nsoadmin@nso1(config-ce-devices-CE-A\_Site\_2)# top  
nsoadmin@nso1 (config)# show configuration  
qos test

ce-devices PE\_00 lan-ge-interface 0/3

!  
ce-devices PE\_10

lan-ge-interface 0/3 !

policy-name test-policy

average-bit-rate 500000

!

nsoadmin@nso1 (config)# **commit**

Commit complete.

1. Delete the created QoS service instance

nsoadmin@nso1(config)# no qos test

nsoadmin@nso1(config)# commit dry-run outformat native native {

device {  
name PE\_00

data interface GigabitEthernet0/3  
no service-policy output test-policy

exit  
no policy-map test-policy

}  
device {

name PE\_10  
data interface GigabitEthernet0/3

no service-policy output test

exit

no policy-map test-policy

nsoadmin@nso1(config)# commit

Step 2: Create l2vpn Template Service Package

Step 3: Create l2vpn-qos Python and Template Service Package

Task 3: Practicing Kickers

NOTE: The kickers exercise is present as a Markdown file under /var/opt/ncs in NSO1. If you are using VS Code we recommend you to open it there.

A screenshot of a computer screen

Description automatically generated

Then you can click on this icon  on the top right of the open file. This will open the viewer, which later you can drag and drop or resize to full the full screen. This way you will have the guide on top of your terminal and copy/pasting or writing the commands will be faster and less prone to errors than copying from the Word document.

A screenshot of a computer

Description automatically generated

To make sure we start the exercise with the required packages run the following under /var/opt/ncs directory

make load-initial-packages

The command might take a while to complete as executes a packages reload inside NSO. When it finishes it will show the output of `show packages package oper-status` showing if the packages were properly loaded.

echo "show packages package oper-status | tab" | ncs\_cli -C

PACKAGE

PROGRAM META FILE

CODE JAVA PYTHON BAD NCS PACKAGE PACKAGE CIRCULAR DATA LOAD ERROR

NAME UP ERROR UNINITIALIZED UNINITIALIZED VERSION NAME VERSION DEPENDENCY ERROR ERROR INFO WARNINGS

------------------------------------------------------------------------------------------------------------------------------------------

cisco-ios-cli-6.100 X - - - - - - - - - - -

cisco-iosxr-cli-7.53 X - - - - - - - - - - -

juniper-junos-nc-4.14 X - - - - - - - - - - -

l3mplsvpn X - - - - - - - - - - -

loopbackbasic X - - - - - - - - - - -

## Harsh Service Redeploy on Device Offline Changes

We don't expect anybody to touch the configuration of our service offline, on the devices, but... we know someone might "forget" about it and mess around with the validated configuration. So we decide that if we detect any changes in the device we'll re-deploy our service, just in case.

We are using already NSO scheduler to do a sync-from of the devices every night, but we don't want to write an action that does check-sync and if any changes do a sync-from and re-deploy. So we'll just monitor on changes on the devices and kick a re-deploy of the services.

First we'll configure a simple l3mpls service. As user nsoadmin from var/opt/ncs directory go into NSO CLI ncs\_cli -C

nsoadmin@ncs# config

Entering configuration mode terminal

nsoadmin@ncs(config)# load merge load\_payloads/l3mplsvpn\_basic\_service.cfg

Loading.

136 bytes parsed in 0.03 sec (3.32 KiB/sec)

nsoadmin@ncs(config)# show config

l3mplsvpn vpn3

vpn-id 128

customer ACME

link 1

device PE\_00

interface 0/0

!

link 2

device PE\_10

interface 0/1

!

!

nsoadmin@ncs(config)#

Let's check what we are going to configure on which devices with commit dry-run and then commit the configuration. Focus in the interface that is created on each PE\_00 and PE\_10, which we will later modify.

nsoadmin@ncs(config)# commit dry-run

cli {

local-node {

data devices {

device PE\_00 {

config {

vrf {

+ definition vpn128 {

+ rd 1:128;

+ route-target {

+ export 1:128;

+ import 1:128;

+ }

+ address-family {

+ ipv4 {

+ }

+ }

+ }

}

interface {

GigabitEthernet 0/0 {

vrf {

+ forwarding vpn128;

}

ip {

no-address {

- address false;

}

address {

primary {

+ address 172.17.1.17;

+ mask 255.255.255.252;

}

}

}

}

}

router {

+ bgp 1 {

+ address-family {

+ with-vrf {

+ ipv4 unicast {

+ vrf vpn128 {

+ redistribute {

+ connected {

+ }

+ static {

+ }

+ }

+ neighbor 172.17.1.18 {

+ remote-as 65001;

+ }

+ }

+ }

+ }

+ }

+ }

}

}

}

device PE\_10 {

config {

vrf {

+ vrf-list vpn128 {

+ address-family {

+ ipv4 {

+ unicast {

+ import {

+ route-target {

+ address-list 1:128;

+ }

+ }

+ export {

+ route-target {

+ address-list 1:128;

+ }

+ }

+ }

+ }

+ }

+ }

}

interface {

+ GigabitEthernet 0/1 {

+ vrf vpn128;

+ ipv4 {

+ address {

+ ip 172.17.1.33;

+ mask 255.255.255.252;

+ }

+ }

+ }

}

+ route-policy pass {

+ value pass;

+ }

router {

bgp {

+ bgp-no-instance 1 {

+ vrf vpn128 {

+ rd 1:128;

+ address-family {

+ ipv4 {

+ unicast {

+ redistribute {

+ connected {

+ }

+ static {

+ }

+ }

+ }

+ }

+ }

+ neighbor 172.17.1.34 {

+ remote-as 65001;

+ address-family {

+ ipv4 {

+ unicast {

+ route-policy in {

+ name pass;

+ }

+ route-policy out {

+ name pass;

+ }

+ as-override {

+ }

+ default-originate {

+ }

+ }

+ }

+ }

+ }

+ }

+ }

}

}

}

}

}

+l3mplsvpn vpn3 {

+ vpn-id 128;

+ customer ACME;

+ link 1 {

+ device PE\_00;

+ interface 0/0;

+ }

+ link 2 {

+ device PE\_10;

+ interface 0/1;

+ }

+}

}

}

nsoadmin@ncs(config)# commit

Commit complete.

nsoadmin@ncs(config)# end

nsoadmin@ncs#

**OPTIONAL:** If you want to look more in detail how kickers act you can leave open in another terminal in VS Code, or in a separate Putty session the devel.log file

nsoadmin@cisco-virtual-machine:/var/log/ncs$ tail -f devel.log

<DEBUG> 1-Feb-2024::22:08:46.072 cisco-virtual-machine ncs[189782][<0.12665.5>]: ncs progress usid=171 tid=5058 datastore=operational context=system trace-id=6cdd0d4c-bbbf-480e-bb35-88ea363bff60 check data kickers

<DEBUG> 1-Feb-2024::22:08:46.072 cisco-virtual-machine ncs[189782][<0.12665.5>]: ncs progress usid=171 tid=5058 datastore=operational context=system trace-id=6cdd0d4c-bbbf-480e-bb35-88ea363bff60 check data kickers: ok (0.000 s)

...

As well it might be interesting to leave open the l3mplsvpn service python vm file ncs-python-vm-l3mplsvpn.log.

The ncs.conf file has already been updated to show kickers, but before we can see them or configure them on the cli we need to unhide debug.

nsoadmin@ncs# unhide debug

nsoadmin@ncs#

### Task 1 - Creating your first kicker

Let's start simple, where for any changes on any devices we will re-deploy our L3VPN service.

Create a kicker that monitors any changes on all devices and triggers the redeploy of our l3mplsvpn vpn3 service. Remember that you can use the | display xpath or including prefixes | display xpath | display prefixes with any show running-config command to find the xpath you want to monitor.

For example

nsoadmin@ncs# show running-config devices device PE\_00 address | display xpath

/devices/device[name='PE\_00']/address 127.0.0.1

nsoadmin@ncs# show running-config devices device PE\_00 address | display xpath | display prefixes

/ncs:devices/ncs:device[ncs:name='PE\_00']/ncs:address 127.0.0.1

nsoadmin@ncs#

nsoadmin@ncs(config)# kickers data-kicker redeploy-l3mplsvpn-on-device-change

Value for 'monitor' (<string>): /ncs:devices/ncs:device

Value for 'kick-node' : /l3mplsvpn:l3mplsvpn

Value for 'action-name' (<string, min: 1 chars>): re-deploy

nsoadmin@ncs(config-data-kicker-redeploy-l3mplsvpn-on-device-change)# show config

kickers data-kicker redeploy-l3mplsvpn-on-device-change

monitor /ncs:devices/ncs:device

kick-node /l3mplsvpn:l3mplsvpn

action-name re-deploy

!

nsoadmin@ncs(config-data-kicker-redeploy-l3mplsvpn-on-device-change)# commit

Commit complete.

nsoadmin@ncs(config-data-kicker-redeploy-l3mplsvpn-on-device-change)#

Remember that a kicker becomes active as soon as you commit its confiuration.

If you have devel.log open you can find the following lines indicating the kicker has been applied

<DEBUG> 30-Jan-2024::12:45:35.874 cisco-virtual-machine ncs[189782][<0.2079.0>]: ncs progress usid=49 tid=4129 datastore=running context=cli trace-id=2801a9ea-dc67-4db5-a58e-f7aab5507721 subsystem=kicker internal at /kicker:kickers changed; invoking kicker\_server:kicker\_data\_changed/4

<INFO> 30-Jan-2024::12:45:35.874 cisco-virtual-machine ncs[189782][<0.2079.0>]: ncs progress usid=49 tid=4129 datastore=running context=cli trace-id=2801a9ea-dc67-4db5-a58e-f7aab5507721 check data kickers: ok (0.000 s)

...

<INFO> 30-Jan-2024::12:45:35.879 cisco-virtual-machine ncs[189782][<0.2079.0>]: ncs progress usid=49 tid=4129 datastore=running context=cli trace-id=2801a9ea-dc67-4db5-a58e-f7aab5507721 invoking data kickers

<INFO> 30-Jan-2024::12:45:35.879 cisco-virtual-machine ncs[189782][<0.2079.0>]: ncs progress usid=49 tid=4129 datastore=running context=cli trace-id=2801a9ea-dc67-4db5-a58e-f7aab5507721 invoking data kickers: ok (0.000 s)

We do sync-from but nothing has changed on the devices

nsoadmin@ncs# devices device PE\_00 sync-from

result true

nsoadmin@ncs#

If you have the devel.log open you can see there that actually no kicker was triggered

Let's now make an offline change on PE\_00, but without touching the L3VPN service. The re-deploy will happen, but there won't be any changes to push to the device

**NOTE:** We are using simulated NSO NETSIM devices to use ncs-netsim cli-c <device-name> to connect to them

nsoadmin@cisco-virtual-machine:~/nso/ncs-run\_CLtest$ ncs-netsim cli-c PE\_00

User admin last logged in 2024-01-30T11:47:40.282086+00:00, to cisco-virtual-machine, from 127.0.0.1 using cli-ssh

admin connected from 10.227.64.241 using ssh on cisco-virtual-machine

PE\_00# config

Entering configuration mode terminal

PE\_00(config)# interface GigabitEthernet 0/2

PE\_00(config-if)# description I was here

PE\_00(config-if)# commit

Commit complete.

PE\_00(config-if)#

To avoid having to schedule the sync-from and wait for it we will trigger it manually

nsoadmin@ncs# devices device PE\_00 sync-from

result true

nsoadmin@ncs#

devel.log shows the kicker was triggered

<INFO> 30-Jan-2024::12:51:17.154 cisco-virtual-machine ncs[189782][<0.2877.0>]: ncs progress usid=49 tid=4165 datastore=running context=cli trace-id=bbc15b25-635f-4af0-8641-535338af918e check data kickers

<DEBUG> 30-Jan-2024::12:51:17.155 cisco-virtual-machine ncs[189782][<0.2877.0>]: ncs progress usid=49 tid=4165 datastore=running context=cli trace-id=bbc15b25-635f-4af0-8641-535338af918e subsystem=kicker redeploy-l3mplsvpn-on-device-change at /ncs:devices/ncs:device[ncs:name='PE\_00'] changed; invoking 're-deploy'

...

And if you have the service log open (ncs-python-vm-l3mplsvpn.log) you can see the create was applied again

<INFO> 30-Jan-2024::12:51:17.198 l3mplsvpn ncs-dp-189876-l3mplsvpn:l3mplsvpn:0-3-th-4188: - Service create(service=/l3mplsvpn:l3mplsvpn{vpn3})

<INFO> 30-Jan-2024::12:51:17.203 l3mplsvpn ncs-dp-189876-l3mplsvpn:l3mplsvpn:0-3-th-4188: - Service create(applying template for device PE\_00)

<INFO> 30-Jan-2024::12:51:17.221 l3mplsvpn ncs-dp-189876-l3mplsvpn:l3mplsvpn:0-3-th-4188: - Service create(applying template for device PE\_10)

Now, the device configuration has not changed as interface GigabitEthernet 0/2 is not owned by our service

nsoadmin@ncs# show running-config devices device PE\_00 config interface GigabitEthernet 0/2 description

devices device PE\_00

config

interface GigabitEthernet0/2

description I was here...

exit

!

!

nsoadmin@ncs#

Now we will modify the configured IP in GigabitEthernet 0/0, which is owned by our L3VPN service.

Current configuration is the following

nsoadmin@ncs# show running-config devices device PE\_00 config interface GigabitEthernet 0/0

devices device PE\_00

config

interface GigabitEthernet0/0

no switchport

vrf forwarding vpn128

ip address 172.17.1.17 255.255.255.252

no shutdown

exit

!

!

nsoadmin@ncs#

As a reminder we can check that the configuration is owned by our service and only by our service (refcount is 1).

nsoadmin@ncs# show running-config devices device PE\_00 config interface GigabitEthernet 0/0 | display service-meta-data

devices device PE\_00

config

! Refcount: 1

! Backpointer: [ /l3mplsvpn:l3mplsvpn[l3mplsvpn:name='vpn3'] ]

interface GigabitEthernet0/0

no switchport

! Refcount: 1

vrf forwarding vpn128

! Refcount: 1

ip address 172.17.1.17 255.255.255.252

no shutdown

exit

!

!

nsoadmin@ncs#

Directly on the device

nsoadmin@cisco-virtual-machine:/var/opt/ncs$ ncs-netsim cli-c PE\_00

User admin last logged in 2024-02-01T21:43:28.406265+00:00, to cisco-virtual-machine, from 127.0.0.1 using cli-ssh

admin connected from 10.227.64.241 using ssh on cisco-virtual-machine

PE\_00# config

Entering configuration mode terminal

PE\_00(config)# interface GigabitEthernet 0/0 ip address 172.27.1.17 255.255.255.255

PE\_00(config-if)# commit

Commit complete.

PE\_00(config-if)#

Now we will do sync-from again, but this time the service re-deploy will find is not in sync as the IP has changed and will push back the original IP configuration.

nsoadmin@ncs# devices device PE\_00 sync-from

result true

nsoadmin@ncs# show running-config devices device PE\_00 config interface GigabitEthernet 0/0

devices device PE\_00

config

interface GigabitEthernet0/0

no switchport

vrf forwarding vpn128

ip address 172.17.1.17 255.255.255.252

no shutdown

exit

!

!

nsoadmin@ncs#

In devel.log we can see again the kicker was triggered

<INFO> 30-Jan-2024::13:00:21.779 cisco-virtual-machine ncs[189782][<0.3572.0>]: ncs progress usid=49 tid=4208 datastore=running context=cli trace-id=a90a3477-065e-4736-bd77-e3fbd7b5c753 check data kickers

<DEBUG> 30-Jan-2024::13:00:21.780 cisco-virtual-machine ncs[189782][<0.3572.0>]: ncs progress usid=49 tid=4208 datastore=running context=cli trace-id=a90a3477-065e-4736-bd77-e3fbd7b5c753 subsystem=kicker redeploy-l3mplsvpn-on-device-change at /ncs:devices/ncs:device[ncs:name='PE\_00'] changed; invoking 're-deploy'

...

And in the l3mplsvpn python vm log we see the create called again

<INFO> 30-Jan-2024::13:00:21.822 l3mplsvpn ncs-dp-189876-l3mplsvpn:l3mplsvpn:0-4-th-4230: - Service create(service=/l3mplsvpn:l3mplsvpn{vpn3})

<INFO> 30-Jan-2024::13:00:21.827 l3mplsvpn ncs-dp-189876-l3mplsvpn:l3mplsvpn:0-4-th-4230: - Service create(applying template for device PE\_00)

<INFO> 30-Jan-2024::13:00:21.849 l3mplsvpn ncs-dp-189876-l3mplsvpn:l3mplsvpn:0-4-th-4230: - Service create(applying template for device PE\_10)

<INFO> 30-Jan-2024::13:00:25.537 l3mplsvpn ncs-dp-189876-l3mplsvpn:l3mplsvpn:0-4-th-4270: - Service create(service=/l3mplsvpn:l3mplsvpn{vpn3})

<INFO> 30-Jan-2024::13:00:25.543 l3mplsvpn ncs-dp-189876-l3mplsvpn:l3mplsvpn:0-4-th-4270: - Service create(applying template for device PE\_00)

<INFO> 30-Jan-2024::13:00:25.565 l3mplsvpn ncs-dp-189876-l3mplsvpn:l3mplsvpn:0-4-th-4270: - Service create(applying template for device PE\_10)

### Task 2 - Adding constrains with trigger-expr

But of course if we made the change in device not used by our L3VPN service we don't want to redeploy, so we could better refine it.

We could just add a starts-with(name,'PE\_') to only apply to PE devices, but we know which devices are configured by our service, so let's be precise and refine it to only trigger if PE\_00 or PE\_10 is modified, using the Kickers trigger-expr option

**NOTE:** Consider that we could include in our service to provision as well a kicker for each service instance, specifically indicating the devices it manages.

To make it easier let's first remember how the xpath to the devices configuration looks like.

nsoadmin@ncs# show running-config devices device PE\_00 config interface GigabitEthernet 0/0 | display xpath

/devices/device[name='PE\_00']/config/ios:interface/GigabitEthernet[name='0/0']/vrf/forwarding vpn128

/devices/device[name='PE\_00']/config/ios:interface/GigabitEthernet[name='0/0']/ip/address/primary/address 172.17.1.17

/devices/device[name='PE\_00']/config/ios:interface/GigabitEthernet[name='0/0']/ip/address/primary/mask 255.255.255.252

nsoadmin@ncs#

And update our kicker with a trigger expression restricting to only trigger kicker if devices are PE\_00 or PE\_10

nsoadmin@ncs(config)# do show running-config kickers data-kicker redeploy-l3mplsvpn-on-device-change

kickers data-kicker redeploy-l3mplsvpn-on-device-change

monitor /ncs:devices/ncs:device

kick-node /l3mplsvpn:l3mplsvpn

action-name re-deploy

!

nsoadmin@ncs(config)# kickers data-kicker redeploy-l3mplsvpn-on-device-change

nsoadmin@ncs(config-data-kicker-redeploy-l3mplsvpn-on-device-change)# trigger-expr "(name = 'PE\_00') or (name = 'PE\_10')"

nsoadmin@ncs(config-data-kicker-redeploy-l3mplsvpn-on-device-change)# commit

Commit complete.

nsoadmin@ncs(config-data-kicker-redeploy-l3mplsvpn-on-device-change)# end

nsoadmin@ncs#

Now we can verify it works as expected. First by changing the configuration of a different device, for example PE\_01

nsoadmin@cisco-virtual-machine:~/nso/ncs-run\_CLtest$ ncs-netsim cli-c PE\_01

User admin last logged in 2024-01-29T12:28:14.199963+00:00, to cisco-virtual-machine, from 127.0.0.1 using cli-ssh

admin connected from 10.227.64.241 using ssh on cisco-virtual-machine

PE\_01# config

Entering configuration mode terminal

PE\_01(config)# interface GigabitEthernet 0/0 description I was here as well...

PE\_01(config-if)# commit

Commit complete.

PE\_01(config-if)#

Then we do a sync-from

nsoadmin@ncs# devices sync-from

sync-result {

device PE\_00

result true

}

sync-result {

device PE\_01

result true

}

sync-result {

device PE\_10

result true

}

sync-result {

device PE\_11

result true

}

nsoadmin@ncs#

In devel.log we can see the kicker was not triggered

<INFO> 30-Jan-2024::17:07:34.310 cisco-virtual-machine ncs[189782][<0.16546.0>]: ncs progress usid=82 tid=4325 datastore=running context=cli trace-id=070d33bb-c3b0-4e7e-860f-0eceb8c77fae check data kickers

<INFO> 30-Jan-2024::17:07:34.311 cisco-virtual-machine ncs[189782][<0.16546.0>]: ncs progress usid=82 tid=4325 datastore=running context=cli trace-id=070d33bb-c3b0-4e7e-860f-0eceb8c77fae check data kickers: ok (0.000 s)

Now we can modify the configuration in PE\_10 that is managed by our service

nsoadmin@cisco-virtual-machine:~/nso/ncs-run\_CLtest$ ncs-netsim cli-c PE\_10

User admin last logged in 2024-01-30T16:07:33.088977+00:00, to cisco-virtual-machine, from 127.0.0.1 using cli-ssh

admin connected from 10.227.64.241 using ssh on cisco-virtual-machine

PE\_10# config

Entering configuration mode terminal

PE\_10(config)# interface GigabitEthernet 0/1 ipv4 address 172.27.1.33 255.255.255.252

PE\_10(config-if)# commit

Commit complete.

PE\_10(config-if)# end

PE\_10#

Then sync-from

nsoadmin@ncs# devices device PE\_10 sync-from

result true

nsoadmin@ncs#

As before, the l3mplsvpn python vm log shows the re-deploy occurred.

<INFO> 02-Feb-2024::00:35:47.182 l3mplsvpn ncs-dp-211277-l3mplsvpn:l3mplsvpn:0-11-th-5896: - Service create(service=/l3mplsvpn:l3mplsvpn{vpn3})

<INFO> 02-Feb-2024::00:35:47.186 l3mplsvpn ncs-dp-211277-l3mplsvpn:l3mplsvpn:0-11-th-5896: - Service create(applying template for device PE\_00)

<INFO> 02-Feb-2024::00:35:47.201 l3mplsvpn ncs-dp-211277-l3mplsvpn:l3mplsvpn:0-11-th-5896: - Service create(applying template for device PE\_10)

Task 4: Implementing Nano-services

As for Task 3 you can open this exercise inside dCloud session, in the VS Code, for a better working and copy/pasting of the commands. Follow the same instructions as at the top of Task 3 but for file “TECOPS-2201 - Lab Guide - Implementing Nano Services.md”.

In this task we will modify the simple l3mplsvpn service into a nano service.

Let's imagine that in order to manage the used VPN IDs on L3VPN services there are some operators that are in charge to select and assign that value, and that in the future an external resource manager will be implemented.

However, we decide we don't want to have to ask them for an ID and wait for it, and later to go back and provision the service with it. We want that to be part of the process, so we trigger the service request with all the parameters, and later when they select the ID the service will be automatically deployed.

For this our service now will need at least 2 states, where in the first the request for allocating a VPN ID is triggered, by adding it to a list the operators monitor (we could as well implement in code sending an email or notification) and stop the create there. and the second so when the ID is allocated it will re-deploy our service and configure the devices.

We will need a kicker as well to monitor when the allocation occurs and trigger the re-deploy, but NSO nano-services will create it automatically and delete once it has been satisfied.

After completing this activity, you will be able to:

* Modify an existing service to use ID allocation.
* Transform a service into a nano service.
* Deploy and track the progress of a nano service.

## Modify Service Model

In this task, you will modify the existing service model in two steps. First, you will create a separate list for the vpn-id allocations for the network operator. Next, you will delete the existing vpn-id leaf.

**NOTE:** The final solution for this lab is located in the /home/cisco/CLEMEAR20\_TECNMS-4175/TECOPS2201\_solutions/nano-services-manual-solution/ directory. You can use it for copying and pasting longer pieces of code and as a reference point for troubleshooting your packages.

To make sure we start the exercise with the required packages run the following under /var/opt/ncs directory

make load-initial-packages

The command might take a while to complete as executes a packages reload inside NSO. When it finishes it will show the output of `show packages package oper-status` showing if the packages were properly loaded.

echo "show packages package oper-status | tab" | ncs\_cli -C

PACKAGE

PROGRAM META FILE

CODE JAVA PYTHON BAD NCS PACKAGE PACKAGE CIRCULAR DATA LOAD ERROR

NAME UP ERROR UNINITIALIZED UNINITIALIZED VERSION NAME VERSION DEPENDENCY ERROR ERROR INFO WARNINGS

------------------------------------------------------------------------------------------------------------------------------------------

cisco-ios-cli-6.100 X - - - - - - - - - - -

cisco-iosxr-cli-7.53 X - - - - - - - - - - -

juniper-junos-nc-4.14 X - - - - - - - - - - -

l3mplsvpn X - - - - - - - - - - -

loopbackbasic X - - - - - - -

### Task 1 Define the Nano Service in YANG

Complete these steps:

1. Use VS Code to open the l3mplsvpn.yang file under packages/l3mplsvpn/src/yang/.
2. Delete the existing vpn-id leaf.

< ... Output Omitted ... >

leaf name {

tailf:info "Service Instance Name";

type string;

}

leaf vpn-id {

type uint16;

tailf:info "Unique VPN ID";

}

leaf customer {

tailf:info "VPN Customer";

< ... Output Omitted ... >

1. Create a list vpn-allocations with two leaves vpn-name and vpn-id. Declare vpn-name to be used as a key leaf and use a unique constraint for the vpn-id. Make sure that vpn-name refers to the name leaf located in the l3mplsvpn list.

< ... Output Omitted ... >

revision 2020-06-04 {

description

"Initial revision.";

}

list vpn-allocations {

key "vpn-name";

unique "vpn-id";

leaf vpn-name {

type leafref {

path /l3mplsvpn/name;

}

}

leaf vpn-id {

type uint16;

tailf:info "Unique VPN ID";

}

}

< ... Output Omitted ... >

1. Save the file and exit the text editor.

#### Task 1 Verification

You have completed this task when you attain these results:

* You have successfully copied a service package
* You have deleted the existing vpn-id leaf
* You have declared a list to store VPN allocations.

## Transform the service into a Nano Service

We will now add the required blocks in YANG to define our nano-service, which will consist of three key steps.

First, the service must include the `ncs:nano-plan-data`` grouping which will change the execution model of the service.

Second, the service must define a nano plan declared with the ncs:plan-outline statement. Nano plan definitions consist of plan components and plan state definitions.

A diagram of a computer

Description automatically generated with medium confidence

Finally, we must define a service behavior tree declared with the ncs:service-behavior-tree statement. The behavior tree is responsible for creating and removing plan components from instances of your service plan.

### Task 2 Transform into a Nano Service

Complete these steps:

1. Open the service model l3mplsvpn.yang using VS code. You can find it under packages/l3mplsvpn/src/yang/l3mplsvpn.yang
2. Find the l3mplsvpn list definition and declare that uses nano-plan-data

< ... Output Omitted ... >

list l3mplsvpn {

uses ncs:service-data;

uses ncs:nano-plan-data;

ncs:servicepoint l3mplsvpn-servicepoint;

key name;

< ... Output Omitted ... >

1. Declare the plan component and plan state for later use in the plan outline.

< ... Output Omitted ... >

revision 2020-06-04 {

description

"Initial revision.";

}

identity l3mplsvpn {

base ncs:plan-component-type;

}

identity l3mplsvpn-configured {

base ncs:plan-state;

}

list vpn-allocations {

key "vpn-name";

unique "vpn-id";

< ... Output Omitted ... >

1. Define your service plan-outline and add the mandatory ncs:self component with two mandatory states ncs:init and ncs:ready.

< ... Output Omitted ... >

identity l3mplsvpn-configured {

base ncs:plan-state;

}

ncs:plan-outline l3mplsvpn-plan {

description "L3 MPLS VPN Plan";

ncs:component-type "ncs:self" {

ncs:state "ncs:init";

ncs:state "ncs:ready";

}

}

list vpn-allocations {

key "vpn-name";

unique "vpn-id";

< ... Output Omitted ... >

**Note:** init and ready states are primarily used as placeholders for timestamps.

1. Define the l3mplsvpn component-type and define two possible states. However, this time, use the l3mplsvpn:l3mplsvpn-configured for the second state declaration.

You have formally declared these identities previously.

< ... Output Omitted ... >

ncs:plan-outline l3mplsvpn-plan {

description "L3 MPLS VPN Plan";

ncs:component-type "ncs:self" {

ncs:state "ncs:init";

ncs:state "ncs:ready";

}

ncs:component-type "l3mplsvpn:l3mplsvpn" {

ncs:state "ncs:init";

ncs:state "l3mplsvpn:l3mplsvpn-configured" {

}

}

}

< ... Output Omitted ... >

**NOTE:** Consider that our service will trigger frist the ncs:init and with it it will add the request of a VPN ID to the list. Then our second state l3mplsvpn-configured will wait until the VPN ID is allocated and then trigger a callback to configure the devices

1. Add a create nano-callback and define a pre-condition inside the block. Write an XPath expression and pass it as an argument to the monitor statement. This will make sure that l3mplsvpn-configured state can be reached only if the monitor condition is satisfied. Make sure you add a nano-callback declaration right after the pre-condition block.

< ... Output Omitted ... >

ncs:plan-outline l3mplsvpn-plan {

description "L3 MPLS VPN Plan";

ncs:component-type "ncs:self" {

ncs:state "ncs:init";

ncs:state "ncs:ready";

}

ncs:component-type "l3mplsvpn:l3mplsvpn" {

ncs:state "ncs:init";

ncs:state "l3mplsvpn:l3mplsvpn-configured" {

ncs:create {

ncs:pre-condition {

ncs:monitor "/vpn-allocations[vpn-name=$SERVICE/name]/vpn-id";

}

ncs:nano-callback;

}

}

}

}

< ... Output Omitted ... >

**INFO:** Our plan outline is now defined, but is the behavior-tree the one responsible for creating components instantiated from your service plan. You could even use the component-type in a different order or skip one in the behavior-tree.

1. Write a behavior-tree for your service by providing the name of your service-point and the name of your plan-outline.

< ... Output Omitted ... >

ncs:service-behavior-tree l3mplsvpn-servicepoint {

description "L3 MPLS VPN behavior tree";

ncs:plan-outline-ref "l3mplsvpn:l3mplsvpn-plan";

ncs:selector {

ncs:create-component "'self'" {

ncs:component-type-ref "ncs:self";

}

ncs:create-component "'l3mplsvpn'" {

ncs:component-type-ref "l3mplsvpn:l3mplsvpn";

}

}

}

ncs:plan-outline l3mplsvpn-plan {

description "L3 MPLS VPN Plan";

< ... Output Omitted ... >

1. Save changes and close the file.
2. Open the l3mplsvpn.py file in VS code. You can find it under packages/l3mplsvpn/python/l3mplsvpn/l3mplsvpn.py.

student@student-vm:~/nso300$ vim packages/l3mplsvpn/python/l3mplsvpn/l3mplsvpn.py

1. Delete the highlighted ServiceCallbacks class declaration, @Service.create decorator, and cb\_create method declaration along with the call to the self.log.debug method.
2. Nano services require class inheritance from a different class than standard services.

import ncs

import math

class ServiceCallbacks(ncs.application.Service):

@Service.create

def cb\_create(self, tctx, root, service, proplist):

self.log.info('Service create(service=', service.\_path, ')')

< ... Output Omitted ... >

1. Declare a class L3MPLSNanoService, which inherits from the ncs.application.NanoService class. Use the @ncs.application.NanoService.create decorator and define the cb\_nano\_create method with a call to the self.log.debug method.

import ncs

import math

class L3MPLSNanoService(ncs.application.NanoService):

@ncs.application.NanoService.create

def cb\_nano\_create(self, tctx, root, service, plan, component, state, proplist, component\_proplist):

self.log.debug("NanoService create ", state)

< ... Output Omitted ... >

**Note:** cb\_nano\_create accepts additional parameters—plan, component, state, and component\_proplist.

1. Delete the line with the vpn\_id declaration. Remember that your service model no longer provides the vpn-id.

< ... Output Omitted ... >

vpn\_id = service.vpn\_id

< ... Output Omitted ... >

1. Add lines to obtain vpn-id from vpn-allocations list, and a call to self.log.info method. Make sure that you use **underscores** and not hyphen when declaring variables.

< ... Output Omitted ... >

self.log.debug("NanoService create ", state)

vpn\_id = root.vpn\_allocations[service.name].vpn\_id

self.log.info(f'Vpn ID read: {vpn\_id}')

< ... Output Omitted ... >

1. Scroll down to near the end of the file and add a return proplist statement. Also, add a call to self.register\_nano\_service with your service parameters.

< ... Output Omitted ... >

template.apply('l3mplsvpn-template', tvars)

return proplist

# ---------------------------------------------

# COMPONENT THREAD THAT WILL BE STARTED BY NCS.

# ---------------------------------------------

class L3MplsVpn(ncs.application.Application):

def setup(self):

self.log.info('L3MplsVpn RUNNING')

self.register\_nano\_service('l3mplsvpn-servicepoint', 'l3mplsvpn:l3mplsvpn', 'l3mplsvpn:l3mplsvpn-configured', L3MPLSNanoService)

def teardown(self):

self.log.info('L3MplsVpn FINISHED')

< ... Output Omitted ... >

1. Save changes.
2. Compile and deploy your package.

< ... Output Omitted ... >

-- Reloading packages for NSO testenv-nso300-5.3.2-student-nso

reload-result {

package cisco-asa-cli-6.10

result true

}

reload-result {

package cisco-ios-cli-6.54

result true

}

reload-result {

package cisco-iosxr-cli-7.26

result true

}

reload-result {

package cisco-nx-cli-5.15

result true

}

reload-result {

package l3mplsvpn

result true

}

< ... Output Omitted ... >

#### Task 2 Verification

You have completed this task when you attain these results:

* You have successfully transformed the service into a nano service.
* You have successfully built the nano service package.

### Task 3 Deploy Nano Services

In this task, you will deploy the newly created nano service. Activity

Complete these steps:

1. Enter the NSO CLI and switch to the Cisco mode.

student@student-vm:~/nso300$ make testenv-cli

docker exec -it testenv-nso300-5.3.2-student-nso bash -lc 'ncs\_cli -u admin'

admin connected from 127.0.0.1 using console on ebdb0d8aa9a2

nsoadmin@ncs> switch cli

nsoadmin@ncs#

1. Enter the configuration mode and create a new customer entry called ACME (if not existing already).

nsoadmin@ncs# config

Entering configuration mode terminal

nsoadmin@ncs(config)# customers customer ACME

nsoadmin@ncs(config-customer-ACME)#

1. Commit the configuration and return to the top.

nsoadmin@ncs(config-customer-ACME)# commit

Commit complete.

nsoadmin@ncs(config-customer-ACME)# top

nsoadmin@ncs(config)#

1. Configure a new VPN service instance for the new customer and name it vpn512.

nsoadmin@ncs(config)# l3mplsvpn vpn512 customer ACME

nsoadmin@ncs(config-l3mplsvpn-vpn512)#

1. Configure two links, each on a different device.

nsoadmin@ncs(config-l3mplsvpn-vpn512)# link 1 device PE\_00 interface 0/1

nsoadmin@ncs(config-link-1)# exit

nsoadmin@ncs(config-l3mplsvpn-vpn512)# link 2 device PE\_10 interface 0/2

nsoadmin@ncs(config-link-2)# exit

nsoadmin@ncs(config-l3mplsvpn-vpn512)#

1. Commit the configuration and exit configuration mode.

nsoadmin@ncs(config-l3mplsvpn-vpn512)# commit

Commit complete.

nsoadmin@ncs(config-l3mplsvpn-vpn512)# end

nsoadmin@ncs#

1. Inspect the status of the newly configured service instance.
2. Plan component l3mplsvpn has not reached the l3mplsvpn-configured state because a pre-condition exists, which requires the presence of a vpn-id value in the vpn-allocations list.

nsoadmin@ncs# show l3mplsvpn vpn512

POST

BACK ACTION

TYPE NAME TRACK GOAL STATE STATUS WHEN ref STATUS

--------------------------------------------------------------------------------------------------------

self self false - init reached 2020-09-15T18:10:11 - -

ready reached 2020-09-15T18:10:11 - -

l3mplsvpn l3mplsvpn false - init reached 2020-09-15T18:10:11 - -

l3mplsvpn-configured not-reached - - -

nsoadmin@ncs#

**Note:** The NSO CLI adapts the output to the width of your terminal window. In case your window is too narrow, you can use the | tab format, to force the NSO CLI to display the tabulated output.

1. Enter the configuration mode.

nsoadmin@ncs# config

Entering configuration mode terminal

nsoadmin@ncs(config)#

1. Use the vpn-allocations command to allocate a new vpn-id to the service instance.

You are taking the role of a network operator that is acting upon a new VPN service request.

nsoadmin@ncs(config)# vpn-allocations ?

Possible completions:

vpn512

nsoadmin@ncs(config)# vpn-allocations vpn512 vpn-id 512

nsoadmin@ncs(config-vpn-allocations-vpn512)#

Commit the configuration and exit the configuration mode.

nsoadmin@ncs(config-vpn-allocations-vpn512)# commit

Commit complete.

nsoadmin@ncs(config-vpn-allocations-vpn512)# end

nsoadmin@ncs#

1. Display the status of the service instance vpn512 one more time.

Notice that the plan component l3mplsvpn has now reached the l3mplsvpn-configured state after the pre-condition was satisfied.

nsoadmin@ncs# show l3mplsvpn vpn512

l3mplsvpn vpn512

modified devices [ PE\_10 ]

directly-modified devices [ PE\_10 ]

device-list [ PE\_10 ]

POST

BACK ACTION

TYPE NAME TRACK GOAL STATE STATUS WHEN ref STATUS

----------------------------------------------------------------------------------------------------

self self false - init reached 2020-09-15T18:10:11 - -

ready reached 2020-09-15T18:10:11 - -

l3mplsvpn l3mplsvpn false - init reached 2020-09-15T18:10:11 - -

l3mplsvpn-configured reached 2020-09-15T18:11:53 - -

nsoadmin@ncs#

#### Task 3 Verification

You have completed this task when you attain these results:

* You have successful configured service instance.
* Your plan component l3mplsvpn has reached l3mplsvpn-configured state.

Summary

Deliver high-quality services faster and more easily through network automation. Cisco Network Services Orchestrator (NSO) is industry-leading software for automating services across traditional and virtualized networks. Use NSO to add, change, and delete services without disrupting overall service, and help ensure that services are delivered in real time.

NSO is now [free to download](https://developer.cisco.com/site/nso/?utm_campaign=nso19&utm_source=website&utm_medium=nso-try) for non-production use! Download NSO to evaluate and learn how to automate your network and orchestrate your services using NETCONF and YANG today. Also do not miss the opportunity to practice more through : <https://developer.cisco.com/site/nso/> , there you can find lab guides, sandbox labs and learning tracks to support you in your automation journey.

[Graphical user interface, text, application, chat or text message

Description automatically generated](https://www.cisco.com/c/en/us/solutions/service-provider/solutions-cloud-providers/network-services-orchestrator-solutions/business-benefits-of-automation-and-orchestration.html)

Ref.: <https://www.cisco.com/c/en/us/solutions/service-provider/solutions-cloud-providers/network-services-orchestrator-solutions.html>

Appendix A

Use Makefile to configure netsim devices

The following command can be used to check the status of the NETSIM devices. If you see them failing like in the below output, we can recreate them.

nsoadmin@nso1:/var/opt/ncs$ ncs-netsim is-alive

DEVICE PE\_00 FAIL

DEVICE PE\_01 FAIL

DEVICE PE\_10 FAIL

DEVICE PE\_11 FAIL

nsoadmin@nso1:/var/opt/ncs$

If netsim devices are not up and running, there is a Makefile create that will support you with netsim creation, netsim loading and netsim deletion.

cd /var/opt/ncs

ls

cat Makefile

make rebuild-netsim

nsoadmin@nso1:/var/opt/ncs$ make rebuild-netsim

ncs-netsim stop

DEVICE PE\_00 already STOPPED

DEVICE PE\_01 already STOPPED

DEVICE PE\_10 already STOPPED

DEVICE PE\_11 already STOPPED

ncs-netsim delete-network

DEVICE PE\_00 already STOPPED

DEVICE PE\_01 already STOPPED

DEVICE PE\_10 already STOPPED

DEVICE PE\_11 already STOPPED

ncs-netsim create-device packages/cisco-ios-cli-6.100 PE\_00

DEVICE PE\_00 CREATED

ncs-netsim add-device packages/cisco-ios-cli-6.100 PE\_01

DEVICE PE\_01 CREATED

ncs-netsim add-device packages/cisco-iosxr-cli-7.53 PE\_10

DEVICE PE\_10 CREATED

ncs-netsim add-device packages/cisco-iosxr-cli-7.53 PE\_11

DEVICE PE\_11 CREATED

ncs-netsim ncs-xml-init > devices.xml

ncs\_load -l -m devices.xml

ncs-netsim start

DEVICE PE\_00 OK STARTED

DEVICE PE\_01 OK STARTED

DEVICE PE\_10 OK STARTED

DEVICE PE\_11 OK STARTED

nsoadmin@nso1:/var/opt/ncs$

Appendix B

Start NSO and verify status

1. Start NSO in system install:

nsoadmin@nso1:$/etc/init.d/ncs start

1. Verify NSO status:

nsoadmin@nso1:~/var/opt/ncs$ **ncs --status**

vsn: 6.1.5

SMP support: yes, using 4 threads

Using epoll: yes

available modules: backplane,netconf,cdb,cli,snmp,webui

running modules: backplane,netconf,cdb,cli,snmp,webui

status: started

…

Or

/etc/init.d/ncs status

1. Check the ncs.conf file for default NSO configuration parameters:

nsoadmin@nso1:$ cd /etc/ncs

nsoadmin@nso1:$ **cat ncs.conf**

<!-- -\*- nxml -\*- -->

<!-- Example configuration file for ncs. -->

<ncs-config xmlns="http://tail-f.com/yang/tailf-ncs-config">

<!-- NCS can be configured to restrict access for incoming connections -->

<!-- to the IPC listener sockets. The access check requires that -->

<!-- connecting clients prove possession of a shared secret. -->

<ncs-ipc-access-check>

<enabled>false</enabled>

<filename>${NCS\_DIR}/etc/ncs/ipc\_access</filename>

</ncs-ipc-access-check>

<!-- Where to look for .fxs and snmp .bin files to load -->

…

1. Login to NSO CLI Juniper mode (default password for user admin is admin):

nsoadmin@nso1:~/var/opt/ncs$ **ncs\_cli -u nsoadmin**

admin connected from 192.168.234.3 using ssh on nso-host

nsoadmin@nso1>

OR

nsoadmin@nso1:/var/opt/ncs$ **ssh -l nsoadmin -p 2024 localhost**

The authenticity of host '[localhost]:2024 ([127.0.0.1]:2024)' can't be established.

RSA key fingerprint is SHA256:ZLWvfBSWDj4yqS1a68ZpTT4nTVsrrCC8CVTBlDPJuO0.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added '[localhost]:2024' (RSA) to the list of known hosts.

Nsoadmin@nso1 password:

nsoadmin connected from 127.0.0.1 using ssh on nso-host

nsoadmin@nso1>

1. Switch to Cisco style CLI:

nsoadmin@nso1> **switch cli**

nsoadmin@nso1# << Cisco style CLI>>

nsoadmin@nso1#

nsoadmin@nso1#

nsoadmin@nso1# **switch cli**

[ok][2019-11-27 05:38:02]

nsoadmin@nso1> << Juniper Style>>

nsoadmin@nso1>

nsoadmin@nso1>

OR

## Cisco Style CLI

nsoadmin@cisco:/var/opt/ncs$ ncs\_cli -u nsoadmin -C

nsoadmin connected from 192.168.234.3 using ssh on nso-host

nsoadmin@ncs#

## Juniper Style CLI

nsoadmin@cisco:/var/opt/ncs$ ncs\_cli -u nsoadmin -J

nsoadmin connected from 192.168.234.3 using ssh on nso-host

nsoadmin@ncs>

1. Explore different show commands from NSO CLI

# show packages packages package-version

# show devices list

# show running-config

1. Observe NSO startup process through different logs: ncs-java-vm.log

nsoadmin@cisco:/var/opt/ncs$

nsoadmin@cisco:/var/opt/ncs$ cd /var/log/ncs

nsoadmin@cisco:/var/log/ncs$ ls -lrt

total 240

-rw-rw-r-- 1 developer developer 0 Nov 27 05:30 netconf.log

-rw-rw-r-- 1 developer developer 0 Nov 27 05:30 snmp.log

-rw-rw-r-- 1 developer developer 13 Nov 27 05:30 ncserr.log.siz

-rw-rw-r-- 1 developer developer 18 Nov 27 05:30 ncserr.log.idx

-rw-rw-r-- 1 developer developer 8 Nov 27 05:30 ncserr.log.1

-rw-rw-r-- 1 developer developer 0 Nov 27 05:31 **ncs-python-vm.log**

-rw-rw-r-- 1 developer developer 826 Nov 27 05:31 rollback10001

-rw-rw-r-- 1 developer developer 0 Nov 27 05:31 localhost:8080.access

-rw-rw-r-- 1 developer developer 7274 Nov 27 05:31 **ncs-java-vm.log**

-rw-rw-r-- 1 developer developer 21401 Nov 27 05:32 ncs.log

-rw-rw-r-- 1 developer developer 53451 Nov 27 05:40 xpath.trace

-rw-rw-r-- 1 developer developer 127736 Nov 27 05:40 devel.log

-rw-rw-r-- 1 developer developer 6097 Nov 27 05:40 audit.log

developer@nso-host:~/nso521/ncs-run-521/logs$

For example if you want to observe the steps of High-Availability and any issues in the establishment of it, you can use devel.log , it will provide with information regarding communication between Primary and Secondary.

Related Sessions at Ciscolive

You can search CiscoLive Amsterdam content catalog with specific keyword and recommend sessions that are relevant to your lab.

[Content Catalog Link](https://www.ciscolive.com/emea/learn/session-catalog.html?search=nso#/)

1. Embracing DevOps for my NSO Use Cases lifecycle – [DEVNET-2224], Alfonso Sandoval Rosas, Software Consulting Engineer, Cisco.
2. Automating Services with NSO – [LABOPS-1507], Spyridon Spyriadis, Software Consulting Engineer, Sofia Athanasiou, Customer Success Specialist, Cisco.
3. Advanced YANG Data Modeling for Cisco NSO – [DEVNET-3014], Bartosz Luraniec, Customer Delivery Software Architect, Cisco.
4. Automatic Services with Cisco NSO and Model-Driven Telemetry – [DEVWKS-3230] – Bartosz Luraniec, Customer Delivery Software Architect, Cisco.
5. Real-time Services Automation with NSO and Model-Driven Telemetry – [LABOPS-1305], Spyridon Spyriadis, Software Consulting Engineer, Sofia Athanasiou, Customer Success Specialist, Cisco.