Travaux pratiques - Utiliser NETCONF pour accéder à un périphérique IOS XE

Objectifs

Partie 1 : Création du réseau et vérification de la connectivité

Partie 2 : Utiliser une session NETCONF pour recueillir des informations

Partie 3 : Utiliser ncclient pour se connecter à NETCONF

Partie 4 : Utiliser ncclient pour récupérer la configuration

Partie 5 : Utiliser ncclient pour configurer un périphérique

Partie 6 : Défi : Modifier le programme utilisé dans ce laboratoire

Contexte/scénario

Le protocole NETCONF (Network Configuration Protocol), défini dans les RFC 4741 et 6241, utilise des modèles de données YANG pour communiquer avec divers périphériques du réseau. YANG (encore une nouvelle génération) est un langage de modélisation de données. Ce langage définit les données qui sont envoyées via des protocoles de gestion réseau, comme NETCONF. Lorsque vous utilisez NETCONF pour accéder à un périphérique IOS XE, les données sont renvoyées au format XML.

Dans ce TP, vous utiliserez un client NETCONF, ncclient, qui est un module Python pour les scripts côté client. Vous utiliserez ncclient pour vérifier que NETCONF est configuré, récupérer une configuration de périphérique et modifier une configuration de périphérique.

Ressources requises

- 1 PC avec système d'exploitation de votre choix
- Boîte virtuelle ou VMWare
- Machine virtuelle DEVASC
- Machine virtuelle CSR1kv

Instructions

Partie 1 : Lancez les machines virtuelles et vérifiez la connectivité

Dans cette partie, vous lancez les deux machines virtuelles et vérifiez la connectivité. Vous établirez ensuite une connexion SSH (Secure Shell).

Étape 1: Lancez les machines virtuelles

Si vous n'avez pas encore terminé le **travail pratique - Installer l'environnement de laboratoire de machine virtuelle** et le **travail pratique - Installer la machine virtuelle CSR1kv**, faites-le maintenant. Si vous avez déjà terminé ces laboratoires, lancez dès maintenant la machine virtuelle DEVASC et la machine virtuelle CSR1000v.

Étape 2: Vérifiez la connectivité entre les machines virtuelles.

a. Dans la machine virtuelle CSR1kv, appuyez sur Entrée pour obtenir une invite de commande, puis utilisez **show ip interface brief** pour vérifier que l'adresse IPv4 est 192.168.56.101. Si l'adresse est différente, notez la.

b. Ouvrez un terminal en code VS dans le DEVASC VM.

c. Ping du CSR1kv pour vérifier la connectivité. Vous auriez déjà dû le faire précédemment dans les laboratoires d'installation. Si vous n'êtes pas en mesure de ping, consultez les laboratoires énumérés ci-dessus à la partie 1a.

```
devasc@labvm:~$ ping -c 5 192.168.56.101
PING 192.168.56.101 (192.168.56.101) 56(84) bytes of data.
64 bytes from 192.168.56.101: icmp_seq=1 ttl=254 time=1.37 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=254 time=1.15 ms
64 bytes from 192.168.56.101: icmp_seq=3 ttl=254 time=0.981 ms
64 bytes from 192.168.56.101: icmp_seq=4 ttl=254 time=1.01 ms
64 bytes from 192.168.56.101: icmp_seq=5 ttl=254 time=1.14 ms

--- 192.168.56.101 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 0.981/1.130/1.365/0.135 ms
devasc@labvm:~$
```

Étape 3: Vérifiez la connectivité SSH à la machine virtuelle CSR1kV.

a. Dans le terminal de la machine virtuelle DEVASC, SSH à la machine virtuelle CSR1kV avec la commande suivante :

```
devasc@labvm:~$ ssh cisco@192.168.56.101
```

Remarque : La première fois que vous SSH vers le CSR1kV, votre VM DEVASC vous avertit de l'authenticité du CSR1kV. Parce que vous faites confiance au CSR1kV, répondez oui à l'invite.

```
The authenticity of host '192.168.56.101 (192.168.56.101)' can't be
established.
RSA key fingerprint is SHA256:HYv9K5Biw7PFiXeoCDO/LTqs3EfZKBuJdiPo34VXDUY.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.56.101' (RSA) to the list of known hosts.
```

b. Entrez **cisco123!** comme mot de passe et vous devriez maintenant être à l'invite de commande EXEC privilégiée pour le CSR1kv.

```
Password:


CSR1kv#
```

c. Laissez la session SSH ouverte pour la partie suivante.

Partie 2 : Utiliser une session NETCONF pour recueillir des informations

Dans cette partie, vous allez vérifier que NETCONF est en cours d'exécution, activer NETCONF si ce n'est pas le cas et vérifier que NETCONF est prêt pour une connexion SSH. YSOU se connecte ensuite au processus NETCONF, démarre une session NETCONF, recueille des informations d'interface et ferme la session.

Étape 1: Vérifiez si NETCONF est en cours d'exécution sur le CSR1kV.

a. NETCONF est peut-être déjà en cours d'exécution si un autre étudiant l'a activé, ou si une version IOS ultérieure l'active par défaut. À partir de votre session SSH avec CSR1kv, utilisez la commande **show platform software yang-management process** pour voir si le démon SSH NETCONF (**ncsshd**) est en cours d'exécution.

```
CSR1kv# show platform software yang-management process
confd : Running
nesd : Running
syncfd : Running
ncsshd : Running
dmiauthd : Running
nginx : Running
ndbmand : Running
pubd : Running

CSR1kv#
```

b. Si NETCONF n'est pas en cours d'exécution, comme indiqué dans la sortie ci-dessus, entrez la commande de configuration globale **netconf-yang**.

```
CSR1kv# config t
CSR1kV (config) # netconf-yang
```

c. Tapez **exit** pour fermer la session SSH.

Étape 2: Accédez au processus NETCONF via un terminal SSH.

Dans cette étape, vous allez rétablir une session SSH avec le CSR1kV. Mais cette fois, vous allez spécifier le port NETCONF 830 et envoyer **netconf** en tant que commande de sous-système.

Remarque : Pour plus d'informations sur ces options, consultez les pages de manuel de SSH (**man ssh**).

a. Tapez exit pour fermer la session SSH. Vous pouvez utiliser la flèche vers le haut pour rappeler la dernière commande SSH et simplement ajouter les options **-p** et **-s** comme indiqué. Ensuite, entrez **cisco123!** comme mot de passe.

```
devasc@labvm:~$ ssh cisco@192.168.56.101 -p 830 -s netconf
cisco@192.168.56.101's password:
```

b. Le CSR1kv répondra par un message de **hello** comprenant plus de 400 lignes de sortie répertoriant toutes ses fonctionnalités NETCONF. La fin des messages NETCONF est identifiée par**]]>]]>**.

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0 </capability>
<capability>urn:ietf:params:netconf:base:1.1 </capability>
<capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
<capability>urn:ietf:params:netconf:capability:xpath:1.0 </capability>
<capability>urn:ietf:params:netconf:capability:validate:1.0 </capability>
<capability>urn:ietf:params:netconf:capability:validate:1.1 </capability>
(output omitted)
      </capability>
</capabilities>
<session-id>20</session-id></hello>]]>]]>
```

Étape 3: Démarrez une session NETCONF en envoyant un message de bonjour du client.

Pour démarrer une session NETCONF, le client doit envoyer son propre message de bonjour. Le message hello doit inclure la version des fonctionnalités de base NETCONF que le client souhaite utiliser.

a. Copiez et collez le code XML suivant dans la session SSH. Notez que la fin du message de hello client est identifiée par un**]]>]]>**.

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
 <capabilities>
   <capability>urn:ietf:params:netconf:base:1.0 </capability>
 </capabilities>
</hello>
]]>]]>
```

b. Passez à la machine virtuelle CSR1kv et utilisez la commande **show netconf-yang sessions** pour vérifier qu'une session NETCONF a été démarrée. Si l'écran de la VM CSR1kv est sombre, appuyez sur la touche **Enter** pour le réveiller.

```
CSR1kv> en
CSRk1v# show netconf-yang sessions
R: Global-lock on running datastore
C: Global-lock on candidate datastore
S: Global-lock on startup datastore

Number of sessions : 1

session-id transport username source-host global-lock
--------------------------------------------------------------------------------
20 netconf-ssh cisco 192.168.56.1 None

CSR1kv#
```

Étape 4: Envoyer des messages RPC à un périphérique IOS XE.

Au cours d'une session SSH, un client NETCONF peut utiliser des messages RPC (Remote Procedure Call) pour envoyer des opérations NETCONF au périphérique IOS XE. Le tableau répertorie certaines des opérations NETCONF les plus courantes.

Opération	Description
<get>	Récupérer les informations de configuration et d'état des périphériques en cours
<get-config>	Récupérer tout ou partie d'un magasin de données de configuration spécifié
<edit-config>	Charge tout ou partie d'une configuration dans le magasin de données de configuration spécifié
<copy-config>	Remplacer un magasin de données de configuration entier par un autre
<delete-config>	Supprimer un magasin de données de configuration
<commit>	Copier le magasin de données candidat vers le magasin de données en cours

Opération	Description
`<lock>/ <unlock>`	Verrouiller ou déverrouiller le système de stockage des données de configuration
`<close-session>`	Fin gracieuse de la session NETCONF
`<kill-session>`	Fin forcée de la session NETCONF

a. Copiez et collez le code XML RPC **get** message suivant dans la session SSH du terminal pour récupérer des informations sur les interfaces sur R1.

```
<rpc message-id="103" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <get>
  <filter>
    <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"/>
   </filter>
 </get>
</rpc>
]]>]]>
```

b. Rappelons que XML ne nécessite pas d'indention ou d'espace blanc. Par conséquent, le CSR1kv retournera une longue chaîne de données XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-
id="103"><data><interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-
interfaces"><interface><name>GigabiteThernet1</name><description>vBox</description><ty
pe xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-
type">ianaift:ethernetCSMACD</type><enabled>true</enabled><ipv4
xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"></ipv4><ipv6
xmlns="urn:ietf:params:xml:ns:yang:ietf-
ip"></ipv6></interface></interfaces></data></rpc-reply>]]>]]>
```

c. Copiez le XML renvoyé, mais n'incluez pas les caractères “]]>]]>”finaux. Ces caractères ne font pas partie du XML renvoyé par le routeur.

d. Recherchez sur Internet “prettify XML”. Trouvez un site approprié et utilisez son outil pour transformer votre XML en un format plus lisible, tel que :

```
<?xml version="1.0"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="103">
  <data>
    <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      <interface>
        <name>GigabiteThernet1 </name>
        <description>VBox </description>
        <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-
type">IANAIFT:EthernetCSMACD </type>
        <enabled>true</enabled>
        <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"/>
        <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"/>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```

Étape 5: Fermez la session NETCONF.

a. Pour fermer la session NETCONF, le client doit envoyer le message RPC suivant :

```
<rpc message-id="9999999" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <close-session />
</rpc>
```

b. Après quelques secondes, vous serez retourné à l'invite du terminal. Revenez à l'invite CSR1kv et affichez les sessions netconf ouvertes. Vous verrez que la session a été fermée.

```
CSR1kv# montrer les sessions netconf-yang
There are no active sessions

CSR1kv#
```

Partie 3 : Utiliser ncclient pour se connecter à NETCONF

Travailler avec NETCONF ne nécessite pas de travailler avec des messages RPC NETCONF bruts et XML. Dans cette partie, vous apprendrez à utiliser le module Python **ncclient** pour interagir facilement avec les périphériques réseau à l'aide de NETCONF. En outre, vous apprendrez à identifier les modèles YANG pris en charge par l'appareil. Ces informations sont utiles lors de la construction d'un système d'automatisation de réseau de production qui nécessite la prise en charge de modèles YANG spécifiques par le périphérique réseau donné.

Étape 1: Vérifiez que ncclient est installé et prêt à l'emploi.

Dans un terminal DEVASC-VM, entrez la commande **pip3 list --format=columns** pour voir tous les modules Python actuellement installés. Canalisation de la sortie vers **more**. Votre sortie peut différer de ce qui suit. Mais vous devriez voir **ncclient** listé, comme indiqué. Sinon, utilisez la commande **pip3 install ncclient** pour l'installer.

```
devasc@labvm:~$ pip3 list --format=columns | more
Package Version
---------------------- -----------
ansible 2.9.6
apache-libcloud 2.8.0
appdirs 1.4.3
argcomplete 1.8.1
astroid 2.3.3
(output omitted)
ncclient 0.6.7
netaddr 0.7.19
netifaces 0.10.4
netmiko 3.1.0
ntlm-auth 1.1.0
oauthlib 3.1.0
(output omitted)
xmltodict 0.12.0
zipp 1.0.0
devasc@labvm:~$
```

Étape 2: Créez un script pour utiliser ncclient pour se connecter au service NETCONF.

Le module ncclient fournit une classe de **gestionnaire** avec une méthode **connect ()** pour configurer les connexions NETCONF distantes. Après une connexion réussie, l'objet renvoyé représente la connexion NETCONF au périphérique distant.

a. Dans le code VS, cliquez sur **File > Open Folder...** and navigate to the et accédez au répertoire **devnet-src**. Cliquez sur **OK**.

b. Ouvrez une fenêtre de terminal en VS Code : **Terminal > New Terminal**.

c. Créez un sous-répertoire appelé **netconf** dans le répertoire **/devnet-src**.

```
devasc@labvm:~/labs/devnet-src$ mkdir netconf
devasc@labvm:~/labs/devnet-src$
```

d. Dans le volet **EXPLORER** sous **DEVNET-SRC**, cliquez avec le bouton droit de la souris sur le répertoire **netconf** et choisissez **New File**.

e. Nommez le fichier **ncclient-netconf.py**.

f. Dans votre fichier de script, importez la classe manager à partir du module **ncclient**. Créez ensuite une variable **m** pour représenter la méthode **connect ()**. La méthode **connect ()** inclut toutes les informations nécessaires pour se connecter au service NETCONF s'exécutant sur CSR1kv. Notez que le port est 830 pour NETCONF.

```
from ncclient import manager

m = manager.connect (
    host="192.168.56.101",
    port=830,
    username="cisco",
    password="cisco123!",
    hostkey_verify=False
    )
```

Si **hostkey_verify** est défini sur True, le CSR1kv vous demandera de vérifier l'empreinte SSH. Dans un environnement de laboratoire, il est sûr de définir cette valeur sur False, comme nous l'avons fait ici.

g. Enregistrez et exécutez le programme pour vérifier qu'il n'y a pas d'erreurs. Vous ne verrez pas encore de sortie.

```
devasc@labvm:~/labs/devnet-src$ cd netconf/
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
devasc@labvm:~/labs/devnet-src/netconf$
```

h. Vous pouvez vérifier que le CSR1kv a accepté la demande de session NETCONF. Il devrait y avoir un message syslog **%DMI-5-AUTH_PASSED** dans la VM CSR1kv. Si l'écran est noir, appuyez sur **Enter** pour réactiver le routeur. Le message syslog peut être vu au-dessus de la bannière.

Étape 3: Ajoutez une fonction d'impression au script afin que les fonctionnalités NETCONF du CSR1kv soient répertoriées.

L'objet **m** renvoyé par la fonction **manager.connect ()** représente la session distante NETCONF. Comme vous l'avez vu précédemment, dans chaque session NETCONF, le serveur envoie d'abord ses capacités, qui est une liste, au format XML, des modèles YANG pris en charge. Avec le module **ncclient**, la liste des capacités reçues est stockée dans la liste **m.server_capabilities**.

a. Utilisez une boucle **for** et une fonction **print** pour afficher les capacités de l'appareil :

```
print("#Supported Capabilities (YANG models):")
```

```
for capability in m.server_capabilities:
    print(capability)
```

b. Sauvegardez et exécutez le programme. La sortie est la même sortie que vous avez obtenue en envoyant le message **Hello** complexe précédemment, mais sans la <capability> balise XML d'ouverture et de fermeture sur chaque ligne.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
#Supported Capabilities (YANG models):
urn:ietf:params:netconf:base:1.0
urn:ietf:params:netconf:base:1.1
urn:ietf:params:netconf:capability:writable-running:1.0
urn:ietf:params:netconf:capability:xpath:1.0
<output omitted>
urn:ietf:params:xml:ns:netconf:base:1.0?module=ietf-netconf&revision=2011-06-01
urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults?module=ietf-netconf-with-
defaults&revision=2011-06-01

        urn:ietf:params:netconf:capability:notification:1.1

devasc@labvm:~/labs/devnet-src/netconf$
```

Partie 4 : Utiliser ncclient pour récupérer la configuration

Dans cette partie, vous allez utiliser le **ncclient** NETCONF pour récupérer la configuration du CSR1kv, utiliser le module **xml.dom.minidom** pour formater la configuration et utiliser un filtre avec get_config () pour récupérer une partie de la configuration en cours d'exécution.

Étape 1: Utilisez la fonction get_config () pour récupérer la configuration en cours d'exécution pour R1.

a. Si vous souhaitez ignorer l'affichage de la sortie des capacités (plus de 400 lignes), commentez le bloc d'instructions qui impriment les capacités, comme indiqué dans la section suivante :

```
'''
print("#Supported Capabilities (YANG models):")
for capability in m.server_capabilities:
    print(capability)
'''
```

b. Vous pouvez utiliser la méthode **get_config ()** de l'objet de session **m** NETCONF pour récupérer la configuration du CSR1kv. La méthode **get_config ()** attend un paramètre de chaîne source qui spécifie la banque de données NETCONF source. Utilisez une fonction d'impression pour afficher les résultats. La seule banque de données NETCONF actuellement sur le CSR1kv est la banque de données **en cours d'exécution**. Vous pouvez vérifier cela avec la commande **show netconf-yang datastores**.

```
netconf_reply = m.get_config(source="running")
print (netconf_reply)
```

c. Enregistrez et exécutez votre programme. La sortie sera bien supérieure à 100 lignes, donc IDLE peut les compresser. Double-cliquez sur le message **texte pressé** dans la fenêtre du shell IDLE pour développer la sortie.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-
id="urn:uuid:3f31bedc-5671-47ca-9781-4d3d7aadae24"
```

```
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"><data><native
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native"><version>16.9</version><boot-
start-marker/><boot-end-marker/><banner><motd><banner>
(output omitted)
devasc@labvm:~/labs/devnet-src/netconf$
```

d. Notez que le XML renvoyé n'est pas formaté. Vous pouvez le copier sur le même site que vous avez trouvé précédemment pour préttifier le XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:3f31bedc-5671-
47ca-9781-4d3d7aadae24">
  <data>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
      <version>16.9 </version>
      <boot-start-marker/>
      <boot-end-marker/>
      <banner>
        <motd>
          <banner>^C </banner>
        </motd>
      </banner>
      <service>
        <timestamps>
          <debug>
            <datetime>
              <msec/>
            </datetime>
          </debug>
          <log>
            <datetime>
              <msec/>
            </datetime>
          </log>
        </timestamps>
      </service>
      <platform>
        <console xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-platform">
          <output>Virtuelle</output>
        </console>
      </platform>
      <hostname>CSR1kV </hostname>
(output omitted)
```

Étape 2: Utilisez Python pour préttifier le XML.

Python a intégré le support pour travailler avec des fichiers XML. Le module **xml.dom.minidom** peut être utilisé pour préttifier la sortie avec la fonction **toprettyxml ()**.

a. Au début de votre script, ajoutez une instruction pour importer le module **xml.dom.minidom**.

```
import xml.dom.minidom
```

b. Remplacez la fonction d'impression simple **print(netconf_reply)** par une version qui imprime une sortie XML préttifiée.

```
print (xml.dom.minidom.parseString (netconf_reply.xml) .toprettyxml ())
```

c. Enregistrez et exécutez votre programme. XML est affiché dans un format plus lisible.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
<?xml version="1.0" ?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:3a5f6abc-76b4-
436d-9e9a-7758091c28b7">
        <data>
                <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
                        <version>16.9 </version>
                        <boot-start-marker/>
                        <boot-end-marker/>
                        <banner>
                                <motd>
                                        <banner>^C </banner>
                                </motd>
                        </banner>
(output omitted)
        </data>
</rpc-reply>

devasc@labvm:~/labs/devnet-src/netconf$
```

Étape 3: Utilisez un filtre avec get_config () pour récupérer uniquement un modèle YANG spécifique.

Un administrateur réseau peut ne vouloir récupérer qu'une partie de la configuration en cours d'exécution sur un périphérique. NETCONF prend en charge le renvoi uniquement des données définies dans un paramètre de filtre de la fonction **get_conf ()**.

a. Créez une variable appelée **netconf_filter** qui récupère uniquement les données définies par le modèle Cisco IOS XE Native YANG.

```
netconf_filter = """
<filter>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native" />
</filter>
"""
netconf_reply = m.get_config(source="running", filter=netconf_filter)
print (xml.dom.minidom.parseString (netconf_reply.xml) .toprettyxml ())
```

b. Enregistrez et exécutez votre programme. Le début de la sortie est le même, comme indiqué ci-dessous. Cependant, seul l'élément XML est affiché cette fois. Auparavant, tous les modèles YANG disponibles sur le CSR1kV étaient affichés. Le filtrage des données récupérées pour afficher uniquement le module YANG natif réduit considérablement votre sortie. En effet, le module YANG natif ne comprend qu'un sous-ensemble de tous les modèles Cisco IOX XE YANG.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
<?xml version="1.0" ?>
```

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:4da5b736-1d33-
47c3-8e3c-349414be0958">
        <data>
                <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
                        <version>16.9 </version>
                        <boot-start-marker/>
                        <boot-end-marker/>
                        <banner>
                                <motd>
                                        <banner>^C </banner>
                                </motd>
                        </banner>
                        <service>
                                <timestamps>
                                        <debug>
                                                <datetime>
                                                        <msec/>
                                                </datetime>
                                        </debug>
                                        <log>
                                                <datetime>
                                                        <msec/>
                                                </datetime>
                                        </log>
                                </timestamps>
                        </service>
                        <platform>
                                <console xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-
platform">
                                        <output>Virtuelle</output>
                                </console>
                        </platform>
                        <hostname>CSR1kV </hostname>
(output omitted)
                </native>
        </data>
</rpc-reply>

devasc@labvm:~/labs/devnet-src/netconf$
```

Partie 5 : Utiliser ncclient pour configurer un périphérique

Dans cette partie, vous utiliserez **ncclient** pour configurer le CSR1kv en utilisant la méthode **edit_config ()** du module **gestionnaire**.

Étape 1: Utilisez ncclient pour modifier le nom d'hôte du CSR1kv.

Pour mettre à jour un paramètre existant dans la configuration du CSR1kv, vous pouvez extraire l'emplacement du paramètre de la configuration récupérée précédemment. Pour cette étape, vous allez définir une variable pour modifier la **<hostname>** valeur.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
<?xml version="1.0" ?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:4da5b736-1d33-
47c3-8e3c-349414be0958">
        <data>
                <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
(output omitted)
                        <hostname>CSR1kV </hostname>
(output omitted)
```

a. Auparavant, vous avez défini une variable <filter>. Pour modifier la configuration d'un périphérique, vous définissez une variable <config> . Ajoutez la variable suivante à votre script **ncclient_netconf.py**. Vous pouvez utiliser NEWHOSTNAME ou n'importe quel nom d'hôte que vous souhaitez.

```
netconf_hostname = """
<config>
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
     <hostname>NEWHOSTNOM </hostname>
  </native>
</config>
"""
```

b. Utilisez la fonction **edit_config ()** de l'objet de session **m** NETCONF pour envoyer la configuration et stocker les résultats dans la variable **netconf_reply** afin qu'ils puissent être imprimés. Les paramètres de la fonction **edit_config ()** sont les suivants :

- **target** - la banque de données NETCONF ciblée à mettre à jour
- **config** - la modification de configuration qui doit être envoyée

```
netconf_reply = m.edit_config(target="running", config=netconf_hostname)
```

c. La fonction **edit_config ()** renvoie un message de réponse XML RPC **<ok/>** indiquant que la modification a été appliquée avec succès. Répétez l'instruction d'impression précédente pour afficher les résultats.

```
print (xml.dom.minidom.parseString (netconf_reply.xml) .toprettyxml ())
```

d. Enregistrez et exécutez votre programme. Vous devriez obtenir une sortie similaire à la sortie affichée ci-dessous. Vous pouvez également vérifier que le nom d'hôte a changé en basculant vers la machine virtuelle CSR1kv.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
(output omitted)
<?xml version="1.0" ?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:e304b225-7951-
4029-afd5-59e8e7edbaa0">
        <ok/>
</rpc-reply>

devasc@labvm:~/labs/devnet-src/netconf$
```

e. Modifiez votre script pour changer le nom d'hôte en CSR1kv. Enregistrez et exécutez votre programme. Vous pouvez également simplement commenter le code de l'étape précédente si vous voulez éviter de changer à nouveau le nom d'hôte.

Étape 2: Use ncclient to create a new loopback interface on R1.

a. Créez une nouvelle <config> variable pour contenir la configuration d'une nouvelle interface de bouclage. Ajoutez ce qui suit à votre script **ncclient_netconf.py**.

Remarque: Vous pouvez utiliser la **description** que vous voulez. Cependant, utilisez uniquement des caractères alphanumériques ou vous devrez les échapper avec la barre oblique inverse (\ ).

```
netconf_loopback = """
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
   <Loopback>
    <name>1 </name>
    <description>Mon premier bouclage NETCONF </description>
    <ip>
     <address>
      <primary>
       <address>10.1.1.1</address>
       <mask>255.255.255.0 </mask>
      </primary>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
"""
```

b. Ajoutez la fonction **edit_config ()** suivante à votre **ncclient_netconf.py** pour envoyer la nouvelle configuration de bouclage à R1, puis imprimer les résultats.

```
netconf_reply = m.edit_config(target="running", config=netconf_loopback)
print (xml.dom.minidom.parseString (netconf_reply.xml) .toprettyxml ())
```

c. Enregistrez et exécutez votre programme. Vous devriez obtenir un résultat similaire à ce qui suit :

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
(output omitted)
<?xml version="1.0" ?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:98437f47-7a93-
4cac-9b9e-9bc8afc9dfa1">
        <ok/>
</rpc-reply>

devasc@labvm:~/labs/devnet-src/netconf$
```

d. Sur le CSR1kv, vérifiez que la nouvelle interface de bouclage a été créée.

```
CSR1kv>fr
CSR1kv# show ip interface brief
```

```
Interface IP-Address OK? Method Status Protocol
GigabitEthernet1 192.168.56.101 YES DHCP up up
Loopback1 10.1.1.1 YES other up up
CSR1kv# show run | section interface Loopback1
interface Loopback1
 description My first NETCONF loopback
 ip address 10.1.1.1 255.255.255.0
CSR1kv#
```

Étape 3: Essayez de créer une nouvelle interface de bouclage avec la même adresse IPv4.

a. Créez une nouvelle variable appelée **netconf_newloop**. Il contiendra une configuration qui crée une nouvelle interface de bouclage 2 mais avec la même adresse IPv4 que sur le bouclage 1: 10.1.1.1 /24. Au niveau de l'interface de ligne de commande du routeur, cela créerait une erreur en raison de la tentative d'assigner une adresse IP dupliquée à une interface.

```
netconf_newloop = """
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
   <Loopback>
    <name>2 </name>
    <description>My second NETCONF loopback</description>
    <ip>
     <address>
      <primary>
       <address>10.1.1.1</address>
       <mask>255.255.255.0 </mask>
      </primary>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
"""
```

b. Ajoutez la fonction **edit_config ()** suivante à votre **fichier ncclient_netconf.py** pour envoyer la nouvelle configuration de bouclage au CSR1kv. Vous n'avez pas besoin d'une instruction d'impression pour cette étape.

```
netconf_reply = m.edit_config(target="running", config=netconf_newloop)
```

c. Sauvegardez et exécutez le programme. Vous devriez obtenir une sortie d'erreur similaire à la suivante avec le message RpceError **Device refused one or more commands**.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
Traceback (most recent call last):
  File "ncclient-netconf.py", line 80, in <module>
    netconf_reply = m.edit_config(target="running", config=netconf_newloop)
  File "/home/devasc/.local/lib/python3.8/site-packages/ncclient/manager.py", line
231, in execute
```

```
    retour cls (self._session,
  File "/home/devasc/.local/lib/python3.8/site-packages/ncclient/operations/edit.py",
line 69, in request
    return self._request(node)
  File "/home/devasc/.local/lib/python3.8/site-packages/ncclient/operations/rpc.py",
line 348, in _request
    raise self._reply.error
ncclient.operations.rpc.RPCError: inconsistent value: Device refused one or more
commands
devasc@labvm:~/labs/devnet-src/netconf$
```

d. NETCONF n'appliquera aucune configuration envoyée si une ou plusieurs commandes sont rejetées. Pour le vérifier, entrez la commande **show ip interface brief** sur R1. Notez que votre nouvelle interface n'a pas été créée.

```
CSR1kv# show ip interface brief
Interface IP-Address OK? Method Status Protocol
GigabitEthernet1 192.168.56.101 YES DHCP up up
Loopback1 10.1.1.1 YES other up up
```

Partie 6 : Défi : Modifier le programme utilisé dans ce laboratoire

Ce qui suit est le programme complet qui a été créé dans ce laboratoire sans aucun code commenté afin que vous puissiez exécuter le script sans erreur. Votre script peut sembler différent. Pratiquez vos compétences Python en modifiant le programme pour envoyer différentes commandes de vérification et de configuration.

```
from ncclient import manager
import xml.dom.minidom

m = manager.connect (
    host="192.168.56.101",
    port=830,
    username="cisco",
    password="cisco123!",
    hostkey_verify=False
    )

print("#Supported Capabilities (YANG models):")
for capability in m.server_capabilities:
    print(capability)

netconf_reply = m.get_config(source="running")
print (xml.dom.minidom.parseString (netconf_reply.xml) .toprettyxml ())

netconf_filter = """
<filter>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native" />
</filter>
"""
netconf_reply = m.get_config(source="running", filter=netconf_filter)
```

```
print (xml.dom.minidom.parseString (netconf_reply.xml) .toprettyxml ())

netconf_hostname = """
<config>
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
     <hostname>CSR1kV </hostname>
  </native>
</config>
"""
netconf_reply = m.edit_config(target="running", config=netconf_hostname)
print (xml.dom.minidom.parseString (netconf_reply.xml) .toprettyxml ())

netconf_loopback = """
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
   <Loopback>
    <name>1 </name>
    <description>My NETCONF loopback</description>
    <ip>
     <address>
      <primary>
       <address>10.1.1.1</address>
       <mask>255.255.255.0 </mask>
      </primary>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
"""
netconf_reply = m.edit_config(target="running", config=netconf_loopback)
print (xml.dom.minidom.parseString (netconf_reply.xml) .toprettyxml ())

netconf_newloop = """
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
   <Loopback>
    <name>2</name>
    <description>My second NETCONF loopback</description>
    <ip>
     <address>
      <primary>
```

```
        <address>10.1.1.1</address>
        <mask>255.255.255.0 </mask>
       </primary>
      </address>
     </ip>
    </Loopback>
   </interface>
  </native>
</config>
"""
netconf_reply = m.edit_config(target="running", config=netconf_newloop)
```