# Discovery 8: Set Up Device Using Python Scripts

## Introduction

In this activity, you will create a python script to add new devices on NSO, fetch-host-keys, and sync-from the configuration.

After completing this activity, you will be able to:

- Create a python script to work with NSO.
- Implement a python function to initialize a new device on NSO.
- Implement a python function to perform fetch-host-keys and sync-from actions.

## Job Aids

The following job aid is available to help you complete the lab activities:

- This lab guide
- Student guide, for general explanations

The following table contains passwords that you might need.

| Device | User Name | Password |
|---|---|---|
| NSO server | student | 1234QWer |
| NSO application | admin | admin |

## Required Resources

The following resources and equipment are required for completing the activities in this lab guide:

- PC or laptop with a web browser
- Access to the internet

## Command Syntax Reference

This lab guide uses the following conventions for command syntax:

| **Formatting** | **Description and Examples** |
|---|---|
| **show running config** | Commands in steps use this formatting. |
| *Example* | Type **show running config** |
| *Example* | Use the **name** command. |
| `show running config` | Commands in CLI outputs and configurations use this formatting. |
| highlight | CLI output that is important is highlighted. |
| *Example* | `student@student-vm:~$ ncs -version` `5.8.2.1` |
| *Example* | Save your current configuration as the default **startup config**. `Router Name# copy running startup` |
| brackets ([ ]) | Indicates the optional element. You can choose one of the options. |
| *Example*: | `(config-if)# frame-relay lmi-type {ansi\|cisco\|q933a}` |
| *italics font* | Arguments for which you supply values. |
| *Example* | Open file **ip tcp window-size** *bytes* |
| angle brackets (<>) | In contexts that do not allow italics, arguments for which you supply values are enclosed in angle brackets [<>]. Do not type the brackets when entering the command. |
| *Example* | If the command syntax is **ping** <*ip_address*>, you enter ping *192.32.10.12* |
| string | A nonquoted set of characters. Type the characters as-is. |
| *Example* | (config)# **hostname MyRouter** |

| Formatting | Description and Examples |
|---|---|
| vertical line (\|) | Indicates that you enter one of the choices. The vertical line separates choices. Do not type the vertical line when entering the command. |
| *Example* | If the command syntax is **show ip route\|arp**, you enter either **show ip route** or **show ip arp**, but not both. |

## Command List

The following are the most common commands that you will need:

Linux Shell:

| Command | Comment |
|---|---|
| **source /home/student/ nso-5.8/ncsrc** | Source NSO environmental variables. |
| **ls\|ll** | Display contents of the current directory. |
| **cd** | Move directly to user home directory. |
| **cd ..** | Exit the current directory. |
| **cd test** | Move into folder »test,« which is a subfolder of the current directory. |
| **cd /home/student/test** | Move into folder »test« by specifying direct path to it, starting from the root of directory system. |
| **ncs_cli -Cu admin** | Log in to NSO CLI directly from local server. |

NSO CLI:

| Command | Comment |
|---|---|
| **switch cli** | Change CLI style. |
| **show ?** | Display all command options for current mode. |
| **configure** | Enter configuration mode. |
| **commit** | Commit new configuration (configuration mode only command). |
| **show configuration** | Display new configuration that has not yet been committed (configuration mode only command). |

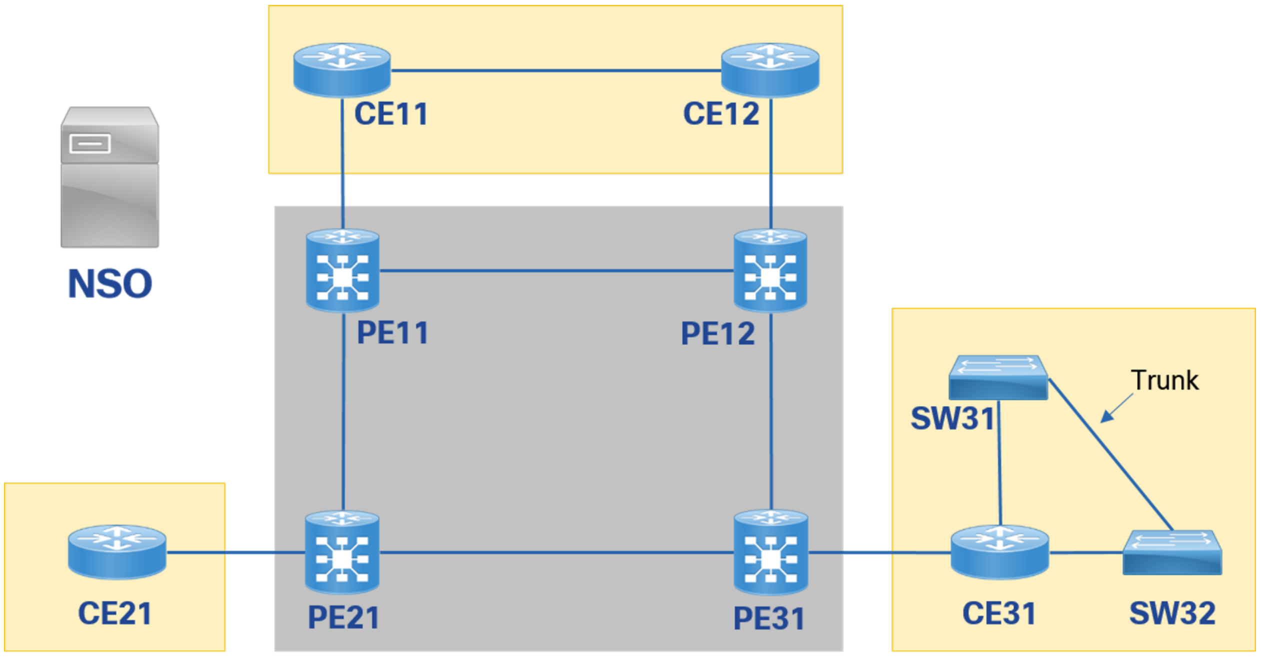
## Lab Topology Information

Your lab session is your own personal sandbox. Whatever you do in your session will not be reflected in anyone else's session. There are two topologies. The general one is your lab environment, and it has your workstation and a Linux server. On the Linux server within that topology is your second topology with your NSO installation, together with numerous netsim routers and switches that are logically grouped into a network topology. This network will be the one that you will orchestrate with your NSO.

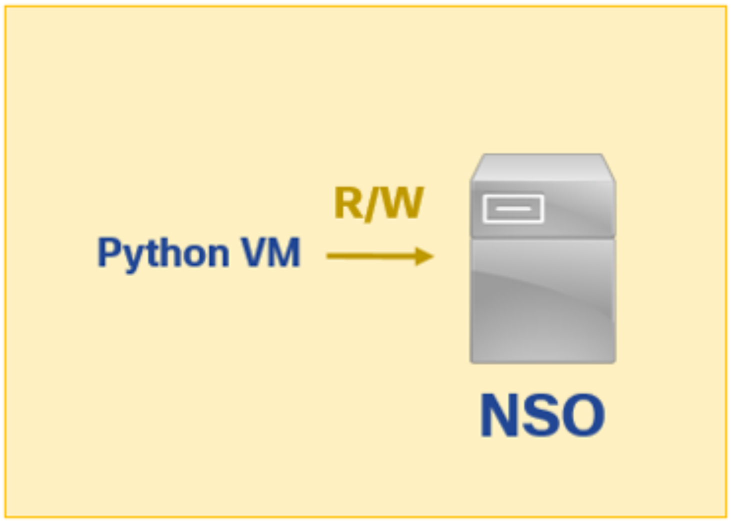
- The network topology is designed to cover both the service provider and enterprise use cases. It is a simulated netsim network; devices have no control or data plane.

Devices will, however, accept or reject a configuration sent by the NSO, just as real devices would.

## Topology



## Visual Objective



## Task 1: Create a Python Script

In this task, you will write a python script to create a new device using MAAGIC API.

The final solutions for all labs, including this one, are in the ~/solutions directory. You can use them for copy-pasting longer pieces of code and as a reference point for troubleshooting your packages.

## Activity

Complete these steps:

### *Step 1*

Connect to the NSO server by clicking the NSO icon.

### *Step 2*

Open the terminal window using the Terminal icon on the bottom bar.

### *Step 3*

Open the **maagic_create_device.py** python file in the home directory by using the Visual Studio Code.

```
student@student-vm:~$ code maagic_create_device.py
```

You can also use an IDE tool or text editor of your choice.

### *Step 4*

Observe the python code skeleton. It includes two functions. The first one is used to parse input arguments and uses the imported module **argparse**. The **main()** method includes the configuration logic.

```
# -*- mode: python; python-indent: 4 -*-

import argparse


def parse_args():
    parser = argparse.ArgumentParser()


def main(args):


if __name__ == '__main__':
    main(parse_args())
```

### *Step 5*

Implement the **parse_args()** function. The script accepts six different arguments, which specify device properties needed to add a device to NSO–name, address, NED ID, port, description, and authentication. The first three are mandatory, and you can specify some default values for the last three.

```
# -*- mode: python; python-indent: 4 -*-

import argparse
```

```
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--name', help='device name'', required=True)
    parser.add_argument('--address', help='device address',
required=True)
    parser.add_argument('--ned', help='device NED ID', required=True)
    parser.add_argument('--port', help='device port', type=int,
default=22)
    parser.add_argument('--desc', help='device description',
                        default='Device created by
maagic_create_device.py')
    parser.add_argument('--auth', help='device authgroup',
default='default')
    return parser.parse_args()


def main(args):


if __name__ == '__main__':
    main(parse_args())
```

***Step 6***

Import the **ncs** module. Inside the **main()** method, create a MAAPI object, establish a MAAPI session and start a write transaction. Using MAAGIC to get the root element, check if the device already exists in CDB, otherwise create a new device in the CDB devices list and set all device parameters as specified in script arguments. At the end, commit the changes using **t.apply()**.

```
# -*- mode: python; python-indent: 4 -*-

import argparse
import ncs


def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--name', help='device name', required=True)
    parser.add_argument('--address', help='device address',
required=True)
    parser.add_argument('--ned' help='device NED ID', required=True)
    parser.add_argument('--port', help='device port', type=int,
default=22)
    parser.add_argument('--desc', help='device description',
                        default='Device created by
maagic_create_device.py')
    parser.add_argument('--auth', help='device authgroup',
default='default')
    return parser.parse_args()


def main(args):
    with ncs.maapi.Maapi() as m:
        with ncs.maapi.Session(m, 'admin', 'python'):
            with m.start_write_trans() as t:
                print(f'Setting the device "{args.name}"
configuration...')

                # Get a reference to the device list
                root = ncs.maagic.get_root(t)
                device_list = root.devices.device

                if args.name not in device_list:
                    device = device_list.create(args.name)
                    device.address = args.address
                    device.port = args.port
                    device.description = args.desc
                    device.authgroup = args.auth
                    dev_type = device.device_type.cli
                    dev_type.ned_id = args.ned
                    device.state.admin_state = 'unlocked'
                    print('Committing the device configuration...')
                    t.apply()
                    pri't('Device committed!')
                else:
                    prin'(f'Device "{args.name}" configuration already
exists...')

            # This transaction is no longer valid - since we are moving
            # back under ncs.maapi.Session(', 'admin', 'python')


if __name__ == '__main__':
    main(parse_args())
```

In Python, a line starting with "#" represents a comment. In your lab, you may skip the comments, they are for your reference only.

You need to explicitly commit the manually opened transactions inside your scripts. However, this is not needed when creating a service callback class, since the commit implicitly happens at the end of the callback.

### *Step 7*

Continue using the MAAPI object and perform **fetch-host-keys** and **sync-from** actions. Both actions do not require any input data. You can call them directly and get the **output**.

```
# -*- mode: python; python-indent: 4 -*-

import argparse
import ncs


def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--name', help='device name', required=True)
    parser.add_argument('--address', help='device address',
required=True)
    parser.add_argument('--ned', help='device NED ID', required=True)
    parser.add_argument('--port', help='device port', type=int,
default=22)
    parser.add_argument('--desc', help='device description',
                        default='Device created by
maagic_create_device.py')
    parser.add_argument('--auth', help='device authgroup',
default='default')
    return parser.parse_args()


def main(args):
    with ncs.maapi.Maapi() as m:
        with ncs.maapi.Session(m, 'admin', 'python'):
            with m.start_write_trans() as t:
                prin'(f'Setting the device "{args.name}"
configuration...')

                # Get a reference to the device list
                root = ncs.maagic.get_root(t)
                device_list = root.devices.device

                if args.name not in device_list:
                    device = device_list.create(args.name)
                    device.address = args.address
                    device.port = args.port
                    device.description = args.desc
                    device.authgroup = args.auth
                    dev_type = device.device_type.cli
                    dev_type.ned_id = args.ned
                    device.state.admin_state = 'unlocked'
                    print('Committing the device configuration...')
                    t.apply()
                    print('Device committed!')
                else:
                    print(f'Device "{args.name}" configuration already
exists...')

            # This transaction is no longer valid - since we are moving
            # back under ncs.maapi.Session(m, 'admin', 'python')

            # fetch-host-keys and sync-from does not require a
            # transaction, continue using the Maapi object

            root = ncs.maagic.get_root(m)
            device = root.devices.device[args.name]
            print('Fetching SSH keys...')
            output = device.ssh.fetch_host_keys()
            print(f'Result: {output.result}')
            print('Syncing configuration...')
            output = device.sync_from()
            print(f'Result: {output.result}')
            if not output.result:
                print(f'Error: {output.info}')
```

```
if __name__ == '__main__':
    main(parse_args())
```

### *Step 8*

Save the file when finished.

### Activity Verification

You have completed this task when you attain the following result:

- You finished the **maagic_create_device.py** implementation.

## Task 2: Test the Script

In this task, you will test and verify the python script created in the previous task.

## Activity

Complete these steps:

### *Step 1*

Open the terminal window using the Terminal icon on the bottom bar.

### *Step 2*

Run the python script with the help option from the home directory.

By using the help option, the script will provide you with a list of parameters that you need to provide.

```
student@student-vm:~$ python maagic_create_device.py --help
usage: maagic_create_device.py [-- --name NAME --address ADDRESS --ned
NED [--port PORT] [--desc DESC] [--auth AUTH]

optional arguments:
  -h, --help         show this help message and exit
  --name NAME        device name
  --address ADDRESS  device address
  --ned NED          device NED ID
  --port PORT        device port
  --desc DESC        device description
  --auth AUTH        device authgroup
```

### *Step 3*

Add one device for each NED by supplying the parameters that match the ones from your netsim devices.

```
student@student-vm:~$ python maagic_create_device.py --name CE31 --
address 127.0.0.1 --ned cisco-ios-cli-6.85 --port 10028
Setting device 'CE31' configuration...
Committing the device configuration...
```

```
Committed!
Fetching SSH keys...
Result: updated
Syncing configuration...
Result: True
student@student-vm:~$ python maagic_create_device.py --name PE21 --
address 127.0.0.1 --ned cisco-iosxr-cli-7.41 --port 10030
Setting the device 'PE21' configuration...
Committing the device configuration...
Device committed!
Fetching SSH keys...
Result: updated
Syncing configuration...
Result: True
student@student-vm:~$ python maagic_create_device.py --name SW32 --
address 127.0.0.1 --ned cisco-nx-cli-5.23 --port 10031
Setting the device 'SW32' configuration...
Committing the device configuration...
Device committed!
Fetching SSH keys...
Result: updated
Syncing configuration...
Result: True
```

***Step 4***

To confirm that you have successfully imported devices using **maagic_create_device.py** script, first connect to the NSO CLI.

```
student@student-vm:~$ ncs_cli -Cu admin
```

***Step 5***

Check if NSO sees the newly imported devices.

```
admin@ncs# show devices brief
NAME  ADDRESS    DESCRIPTION                               NED ID
--------------------------------------------------------------------------------
CE31  127.0.0.1  Device created by maagic_create_device.py  cisco-ios-
cli-6.85
PE21  127.0.0.1  Device created by maagic_create_device.py  cisco-iosxr-
cli-7.42
SW32  127.0.0.1  Device created by maagic_create_device.py  cisco-nx-
cli-5.23
admin@ncs#
```

## Activity Verification

You have completed this task when you attain the following result:

- The devices have been successfully imported into NSO.