pyhpecw7 Documentation

Author

Aug 21, 2020

Contents

CHAPTER 1

Introduction

This library was built to simplify working with HP Comware 7 switches that have a NETCONF API. Rather than to have network developers worry about the underlying NETCONF API, this library provides a means to manage HP Com7 switches through pre-built Python objects that make it extremely easy to get started programming in an HP environment.

Before getting started, it's important to understand the high level workflow required when using this library, which again, is meant to be fairly straight forward and simple.

1.1 Step 1 - Create HPCOM7 object for each HP switch being managed

```
>>> from pyhpecw7.comware import HPCOM7
>>>
>>> args = dict(host='hp1', username='hp', password='hp123')
>>>
>>> device = HPCOM7(**args)
```

1.2 Step 2 - Open a Connection to the Device

```
>>> device.open()
<ncclient.manager.Manager object at 0x7fa5088f98d0>
```

1.3 Step 3a - GET Config Data

Nearly all features supported such as `Vlan`, which is used in an example below, have a `get_config()` method.

The object is imported, it is instantiated, and the `get_config()` can be called. This method usually returns a Python dictionary with several key value pairs if the configuration resource exists and if it doesn't it's an empty dictionary.

Examples are below.

1.4 Step 3b - CONFIGURE the Device

Maybe you don't want to get any data, but rather want to make a configuration change.

Making a configuration change is a two-step process (for most configuration features).

Step 1 Build the configuration, which creates the appropriate configuration object and places it in a *staging* area, but does not push it.

Most methods of the feature class to *build* the configuration are called `build` or `remove`.

Step 2 Execute, or push, the configuration object(s) to the device.

While in most cases, users may just want or need to push a single object, it is possible to build, or stage, multiple configuration objects and push them all at once in which case they are executed in the order as they were built. Each time a `build` or `remove` is executed, a list is being appended to, which is storing the staged objects.

The final execution (config push) happens by using the `execute` method of the `HPCOM7` object.

Examples are below.

CHAPTER 2

Getting Started

```
>>> from pyhpecw7.comware import HPCOM7
>>>
>>> args = dict(host='hp1', username='hp', password='hp123')
>>>
>>> device = HPCOM7(**args)
>>>
>>> device.open()
<ncclient.manager.Manager object at 0x7fa5088f98d0>
```

2.1 GET VLAN INFO

```
>>> from pyhpecw7.features.vlan import Vlan
>>>
>>> vlan = Vlan(device, '20')
>>>
>>> vlan.get_config()
{}
>>>
>>>
>>> vlan = Vlan(device, '10')
>>>
>>> vlan.get_config()
{'name': 'VLAN10_WEB', 'vlanid': '10', 'descr': 'VLAN 0010'}
>>>
```

VLAN 20 did not exist on the switch, but VLAN 10 did.

A VLAN object for any VLAN can also return all VLAN IDs that exist on the switch.

```
>>> vlan.get_vlan_list()
['1', '10']
```

2.2 CONFIGURE A VLAN

Let's create a new VLAN object.

```
>>> vlan = Vlan(device, '20')
>>>
```

Now add VLAN 20. To do this we'll use the `build` method.

```
>>> vlan.build()
True
```

When `True` is returned, it means that the config object that will be pushed has successfully been staged.

The next step is to execute the change.

```
>>> response = device.execute()
>>>
>>> vlan.get_config()
{'name': 'VLAN 0020', 'vlanid': '20', 'descr': 'VLAN 0020'}
>>>
```

Setting the vlan name or description could easily be sent in as additional key value pairs (or with a dictionary) when using `build`.

```
>>> args = dict(name='NEWV20', descr='DESCR_20')
>>>
>>> vlan.build(**args)
True
>>>
>>> rsp = device.execute()
>>>
>>> vlan.get_config()
{'name': 'NEWV20', 'vlanid': '20', 'descr': 'DESCR_20'}
>>>
```

To remove the VLAN, the `remove` method is used.

```
>>> vlan.remove()
True
>>>
>>> rsp = device.execute()
>>>
>>> vlan.get_config()
{}
>>>
```

2.3 GET INTERFACE INFO

```
>>> from pyhpecw7.features.interface import Interface
>>>
>>> interface = Interface(device, 'FortyGigE1/0/4')
>>>
>>> interface.get_config()
```

(continues on next page)

(continued from previous page)

```
{'admin': 'up', 'duplex': 'auto', 'speed': 'auto', 'description': 'DESCR104', 'type':
↪'routed'}
>>>
```

2.4 DEFAULT AN INTERFACE

```
>>> interface.default()
True
>>>
>>> response = device.execute()
>>>
>>> interface.get_config()
{'admin': 'up', 'duplex': 'auto', 'speed': 'auto', 'description': 'FortyGigE1/0/4␣
↪Interface', 'type': 'switched'}
>>>
```

2.5 CONFIGURE AN INTERFACE

```
>>> interface.build(admin='down', description='TEST_DESCR')
True
>>>
>>> rsp = device.execute()
>>>
>>> interface.get_config()
{'admin': 'down', 'duplex': 'auto', 'speed': 'auto', 'description': 'TEST_DESCR',
↪'type': 'switched'}
>>>
```

Contents:

2.5.1 pyhpecw7 package

2.5.1.1 Subpackages

pyhpecw7.features package

Submodules

pyhpecw7.features.cleanerase module

pyhpecw7.features.config module

pyhpecw7.features.errors module

pyhpecw7.features.facts module

CHAPTER 3

Indices and tables

- genindex
- modindex
- search