The bridge to possible

Enforcing Interface Configuration Standards through Automation

A DevNet Associate Prep Program Webinar

Hank Preston, ccie r/s, DevNet Associate & Professional
Principal Engineer – Learning and Certifications
Twitter: @hfpreston

June 8, 2021

Frequently Asked Questions

- Q: Is this session being recorded?
  - A: YES! It will be available on the DevNet Associate Prep Program as soon as possible
- Q: Is the code from the script/demo available?
  - A: YES! It is available on GitHub and CodeExchange
- Q: Will there be Q/A during the session?
  - A: YES! We have panelists answering questions, and we'll do live Q/A for 10-15 minutes after the presentation.
  - A2: You can also ask questions on this topic in this forum.
- Q: Will the slides be available for download?
  - A: YES! They will be available as resources on the DevNet Associate Prep Program and in the forum

You are walking out of root cause analysis readout meeting from last weeks network down situation when the network architect walks up. Like so many before, this outage was partially caused by inconsistent configuration in the network. It'll be a big project to fully resolve, but she asks if you can come up with a way to update interface descriptions across the network based on the CSV "Source of Truth".

So what do we really need to do

- Configure interface descriptions on IOS, IOS XE, NX-OS, and IOS XR devices
- Description should be of format:
  Connected to {DEVICE} {INTERFACE} – {PURPOSE}
- Data for device, interface, and purpose stored in CSV file
- Stretch Goals:
  - Save current description on interfaces for audit/change control
  - Check if interfaces are actually connected to interfaces listed in CSV file

A common manual approach to this task

1. Log into first device
2. Run: show interface description
3. Save output to file
4. Create text file with new interface descriptions for device
    1. Maybe use a bit of ”spreadsheet automation” to build config
5. Copy/Paste config into device
6. Run: show cdp neighbor
7. Compare output to spreadsheet, note discrepancies
8. Log out of device
9. Move to next device
10. Repeat

A common manual approach to this task

1. Log into first device
2. ~~Run: show interface description~~
3. ~~Save output to file~~
4. Create text file with new interface descriptions for device
    1. Maybe use a bit of ”spreadsheet automation” to build config
5. Copy/Paste config into device
6. ~~Run: show cdp neighbor~~
7. ~~Compare output to spreadsheet, note discrepancies~~
8. Log out of device
9. Move to next device
10. Repeat

How a DevNet Associate Candidate Can Tackle It

- Use programmability skills to gather info and build configs
- The goal is to automate what you would manually do anyway
- Test and build your solution in a lab
  - Or DevNet Sandbox if you don't have a Lab
- Result in a solution that can be used again and again

Questions to ask as we start

Questions from last session

How can we “talk” to the devices? (ie CLI, API, NETCONF, RESTCONF, etc)

We’ll stick with CLI for this task. It’s an interface that is understood by our team and works everywhere.

What tool/language will we use?

Python is still a great choice (and often will be). pyATS worked very well last time, no need to change.

How do we create the list of devices to work with?

It’s great when we’ve already solved a problem in one use case, means we don’t need to solve it again.

How will we share our code for others to use?

It’s great when we’ve already solved a problem in one use case, means we don’t need to solve it again.

How do we protect any “secrets”? (ie usernames/passwords)

It’s great when we’ve already solved a problem in one use case, means we don’t need to solve it again.

Questions to ask as we start

How do we read and update CSV “Source of Truth”?

How will we build the device configurations?

How will we apply the configurations to the devices?

How do we learn the current descriptions?

How can we verify what the interface is connected to?

How do we read and update CSV “Source of Truth”?

- Python’s CSV library supports reading and writing data
  - csv.DictReader / csv.DictWriter
    Interact with CSV like a dictionary
- “Updating” a file can be tricky
- Python tempfile and shutil libraries provides useful utilities here

The `csv` module defines the following classes:

class csv.**DictReader**(*f*, *fieldnames=None*, *restkey=None*, *restval=None*, *dialect='excel'*, **args*, ***kwds*)

Create an object that operates like a regular reader but maps the information in each row to a `dict` whose keys are given by the optional *fieldnames* parameter.

The *fieldnames* parameter is a sequence. If *fieldnames* is omitted, the values in the first row of file *f* will be used as the fieldnames. Regardless of how the fieldnames are determined, the dictionary preserves their original ordering.

If a row has more fields than fieldnames, the remaining data is put in a list and stored with the fieldname specified by *restkey* (which defaults to `None`). If a non-blank row has fewer fields than fieldnames, the missing values are filled-in with the value of *restval* (which defaults to `None`).

All other optional or keyword arguments are passed to the underlying `reader` instance.

Changed in version 3.6: Returned rows are now of type `OrderedDict`.

Changed in version 3.8: Returned rows are now of type `dict`.

A short usage example:

```
>>> import csv
>>> with open('names.csv', newline='') as csvfile:
...     reader = csv.DictReader(csvfile)
...     for row in reader:
...         print(row['first_name'], row['last_name'])
...
Eric Idle
John Cleese

>>> print(row)
{'first_name': 'John', 'last_name': 'Cleese'}
```

class csv.**DictWriter**(*f*, *fieldnames*, *restval=''*, *extrasaction='raise'*, *dialect='excel'*, **args*, ***kwds*)

How will we build the device configurations?

- First thing, what format of configuration and interaction?
  - CLI, NETCONF, RESTCONF, API?
- CLI configurations are just “strings”
- Many string formatting possibilities in Python
  - “f-strings”
    description = f”description {interface_description}”
  - String “addition”
    description = “description “ + interface_description
  - Jinja templates
    template = jinja2.Template(
    "description {{interface_description}}”
)
    template.render(interface_description="blah")

How will we apply the configurations to the devices?

- Changing configurations requires more caution reading
- Some options/ideas
  - Create CLI configurations, but manually apply (copy/paste)
  - Create 2 scripts – one to create configs and another to apply
  - Script options/arguments for applying configurations (“Are you sure?”)

How do we learn the current descriptions?

- Reading current config/state from device pretty straight forward
- Python + pyATS provides options
  - Parse “show interface(s)” output
    - Command and output platform differences
  - Parse “show running-config”
    - Lots of extra information
  - Learn “interface”
    - Consistent data model across devices

- pyATS Parser
  - Convert output from single command to a Python dictionary
  - OS/Platform specific
    Same command on different OS’s can result in different output
- pyATS Model
  - Convert output from several show commands to a Python dictionary
  - Data model returned consistent across OS and Platform
    Output from “learn interface” same for all OS’s

How can we verify what the interface is connected to?

- If running and supported, CDP or LLDP can provide this data
- pyATS parsers available for both CDP and LLDP
  - show neighbor detail
- pyATS has model (in development) for LLDP
- pyATS also has an API to configure (un) CDP and LLDP
  - Makes it simple to enable a feature in code

How can we verify what the interface is connected to?

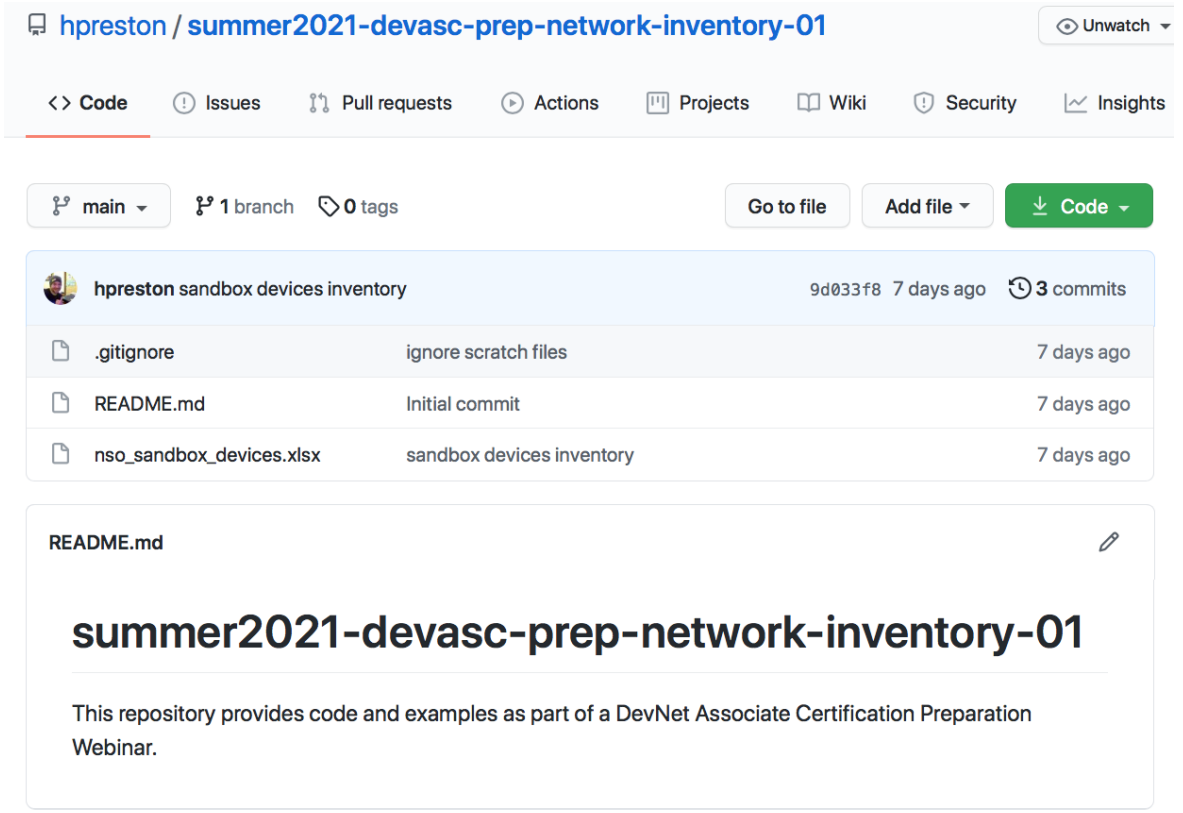
```
# Example enabling and using LLDP info
for device in testbed.devices:
    # Enable LLDP
    testbed.devices[device].api.configure_lldp()

    # Parse LLDP Neighbor Details
    lldp_info = testbed.devices[device].parse("show lldp neighbors detail")

    # Print neighbors
    for interface, details in lldp_info["interfaces"].items():
        local_interface = interface

        for neighbor_interface, neighbors in details["port_id"].items():

            for neighbor, neighbor_details in neighbors["neighbors"].items():
                print(f"Local  {local_interface} is connected Neighbor {neighbor} Interface {neighbor_interface}")
```

#OUTPUT

Local Interface GigabitEthernet0/0/0/1 is connected to Neighbor edge-sw01 Interface GigabitEthernet0/2
Local Interface GigabitEthernet0/0/0/2 is connected to Neighbor dist-rtr01.virl.info Interface GigabitEthernet2 Local Interface GigabitEthernet0/0/0/3 is connected to Neighbor dist-rtr02.virl.info Interface GigabitEthernet3 Local Interface GigabitEthernet2 is connected to Neighbor core-rtr01.virl.info Interface GigabitEthernet0/0/0/2
Local Interface GigabitEthernet6 is connected to Neighbor dist-rtr02.virl.info Interface GigabitEthernet6
Local Interface GigabitEthernet5 is connected to Neighbor dist-sw02 Interface Ethernet1/3
Local Interface GigabitEthernet4 is connected to Neighbor dist-sw01 Interface Ethernet1/3
Local Interface GigabitEthernet3 is connected to Neighbor core-rtr02.virl.info Interface GigabitEthernet0/0/0/2

Start out right!

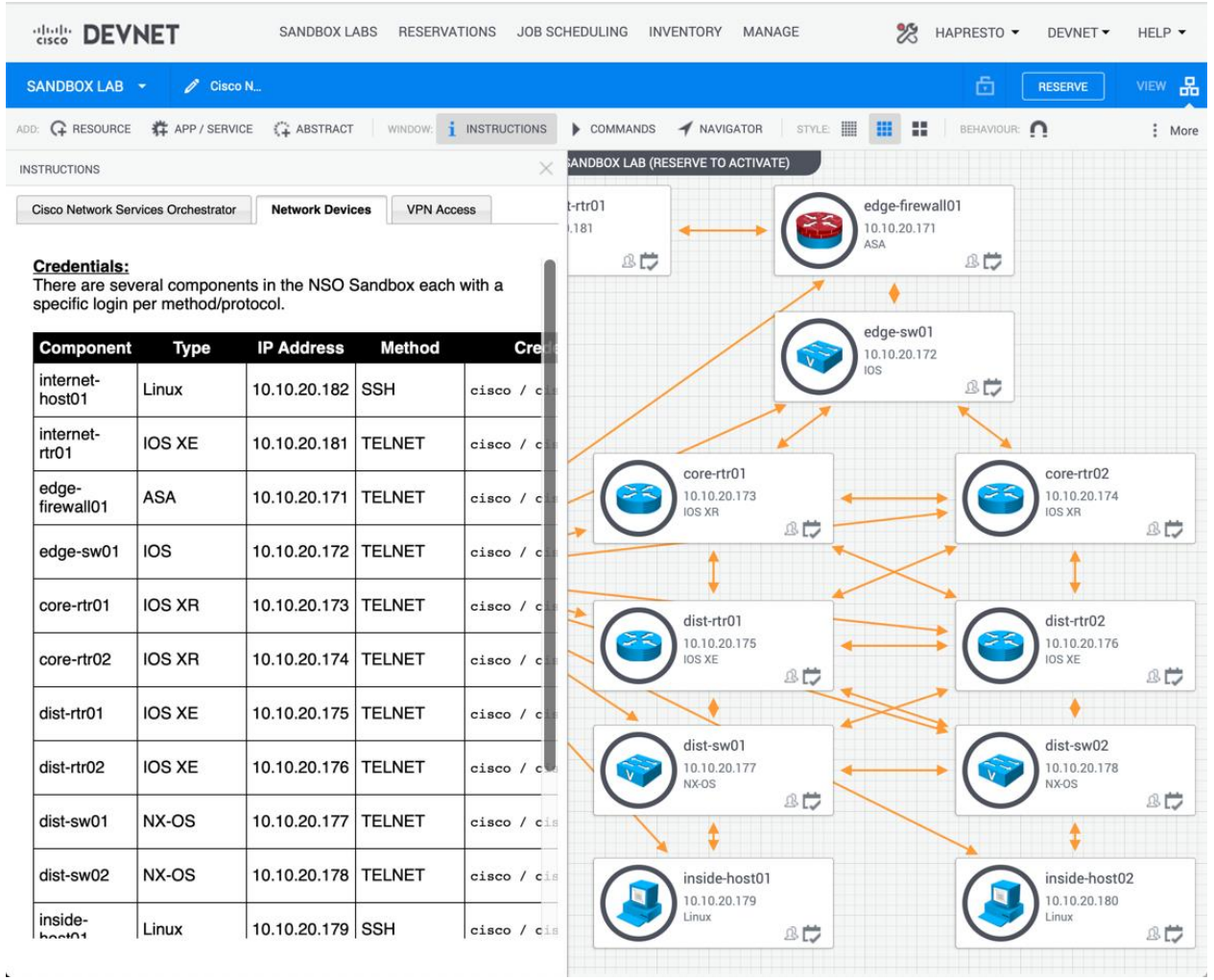
- Create a new Git repo for the project and clone to your workstation
- What lab will you develop with?
  - We'll use the Cisco NSO Reservable Sandbox from DevNet




```
git clone git@github.com:hpreston/summer2021-devasc-prep-interface-config.git

cd summer2021-devasc-prep-interface-config
```

Start out right!

- Create a new Git repo for the project and clone to your workstation
- What lab will you develop with?
  - We'll use the [Cisco NSO Reservable Sandbox] from DevNet



Python Virtual Environment

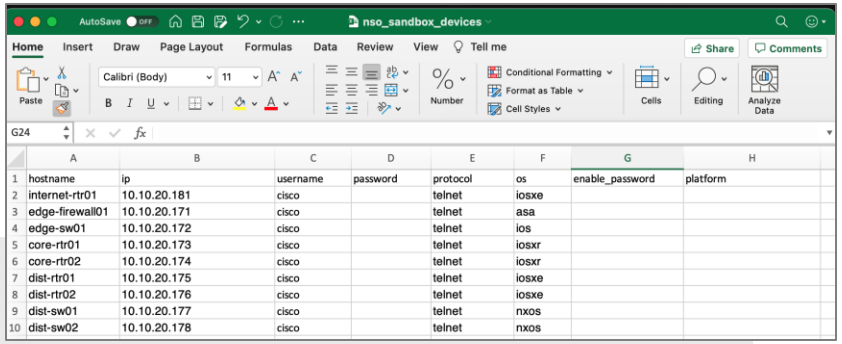
- Get into the habit of creating a new virtual environment for every Python project
- Python Version to use?
  - Python 3.7 or 3.8 great choices today (Summer 2021)
  - Python 3.9 okay too, but some tools and libraries don’t support yet
- Install requirements
  - pip install pyats pyats.contrib genie
  - pip install jinja2

```
# Create your Virtual Env
python3 -m venv venv
source venv/bin/activate

# Install the entire pyATS set of tools
pip install pyats[full] jinja2

# Install just the basics for this exercise
pip install pyats pyats.contrib genie
pip install jinja2
```

Review

Our “inventory” / testbed file for the project

- We create the pyATS testbed file from spreadsheet
- Suggested improvements to generated testbed file
  - Move credentials from each device to testbed level
  - ’ASK{}’ for Username as well as passwords



	A	B	C	D	E	F	G	H
1	hostname	ip	username	password	protocol	os	enable_password	platform
2	internet-rtr01	10.10.20.181	cisco		telnet	iosxe		
3	edge-firewall01	10.10.20.171	cisco		telnet	asa		
4	edge-sw01	10.10.20.172	cisco		telnet	ios		
5	core-rtr01	10.10.20.173	cisco		telnet	iosxr		
6	core-rtr02	10.10.20.174	cisco		telnet	iosxr		
7	dist-rtr01	10.10.20.175	cisco		telnet	iosxe		
8	dist-rtr02	10.10.20.176	cisco		telnet	iosxe		
9	dist-sw01	10.10.20.177	cisco		telnet	nxos		
10	dist-sw02	10.10.20.178	cisco		telnet	nxos		

```
testbed:
  credentials:
    default:
      password: '%ASK{}'
      username: '%ASK{}'
    enable:
      password: '%ASK{}'


devices:
 core-rtr01:
  connections:
    cli:
      ip: 10.10.20.173
      protocol: telnet
  os: iosxr
  type: iosxr
 core-rtr02:
  connections:
    cli:
      ip: 10.10.20.174
      protocol: telnet
  os: iosxr
  type: iosxr
```

```
pyats create testbed file \
  --path nso_sandbox_devices.xlsx \
  --output nso_sandbox_testbed.yaml

# Handy command to verify testbed file
pyats validate testbed \
  --testbed improved_nso_sandbox_testbed.yaml
```

Reminder 😀

Good Development Habits - commit early, commit often

- Even for small, solo projects it is a great idea to make regular commits
  - Makes it easy to track development progress on a project
  - Rolling back a change easier than “Undo”
  - Great practice for later when working on collaborative projects

```
# Stage your changes
#  Entire directory
git add .

#  Specific files
git add improved_nso_sandbox_testbed.yaml


# Commit your changes with good message
git commit –m “pyATS Testbed File for Sandbox”


# Push your changes upstream
#   Does NOT need to happen every commit
git push
```

Planning our Script

- Having a plan before starting lets us break the problem into small chunks
- As you develop the solution, the plan might change
  - New steps needed, different order, etc
- Your commit log tracks this evolution nicely

```
#! /usr/bin/env python
"""
A script to create and apply interface descriptions from CSV
file based source of truth.

Goal:
 - Create interface description config from CSV file
 - Apply configurations to devices with confirmation
 - Record initial/old interface description back to CSV
 - Verify if interfaces are connected as documented in CSV
"""

# Script entry point
if __name__ == "__main__":
    print("Deploying standard interface descriptions to network.")


    # Read data from CSV source of truth
    # Generate desired interface description configurations
    # Load pyATS testbed and connect to devices
    # Lookup current interface descriptions
    # Apply new interface description configuration (with confirmation)
    # Gather CDP/LLDP neighbor details from devices
    # Check if neighbor details match Source of Truth
    # Disconnect from devices
    # Update Source of Truth with Results
```

Run the script

- Important to verify every step along the way

```
chmod +x config_interface_descriptions.py

./config_interface_descriptions.py

# OUTPUT
Deploying standard interface descriptions to network.
```

Reminder 😀

Good Development Habits - commit early, commit often

- Even for small, solo projects it is a great idea to make regular commits
  - Makes it easy to track development progress on a project
  - Rolling back a change easier than “Undo”
  - Great practice for later when working on collaborative projects

Last reminder of this for this presentation, BUT that doesn’t mean we don’t commit at each step along the way.

```
# Stage your changes
#  Entire directory
git add .

#  Specific files
git add improved_nso_sandbox_testbed.yaml


# Commit your changes with good message
git commit –m “Script plan and outline”


# Push your changes upstream
#   Does NOT need to happen every commit
git push
```

Setting up script arguments

- Argparse again used to let users provide input
- Use flagged arguments rather than position based
- Note use of
  - required=True
  - action='store_true'

```
# Use argparse retrieve script options
import argparse
parser = argparse.ArgumentParser(
  description='Deploying standard interface descriptions to network.')

parser.add_argument('--testbed', required=True,
  type=str, help='pyATS Testbed File')
parser.add_argument('--sot', required=True, type=str,
  help='Interface Connection Source of Truth Spreadsheet')
parser.add_argument('--apply', action='store_true',
  help="Should configurations be applied to network. If not set, config not applied.")

args = parser.parse_args()


print(f"Generating interface descriptions from file {args.sot} for testbed
{args.testbed}.")

if args.apply:
    print("Configurations will be applied to devices.")
else:
    print("Configurations will NOT be applied to devices. They will be output to the
screen only.")
```

Run the script

- Argparse provides help messages

```
./config_interface_descriptions.py --help

# OUTPUT
Deploying standard interface descriptions to network.
usage: 02_config_interface_descriptions.py [-h] --testbed TESTBED --sot SOT [--apply]

Deploying standard interface descriptions to network.

optional arguments:
  -h, --help        show this help message and exit
  --testbed TESTBED  pyATS Testbed File
  --sot SOT         Interface Connection Source of Truth Spreadsheet
  --apply           Should configurations be applied to network. If not set, config not applied.
```

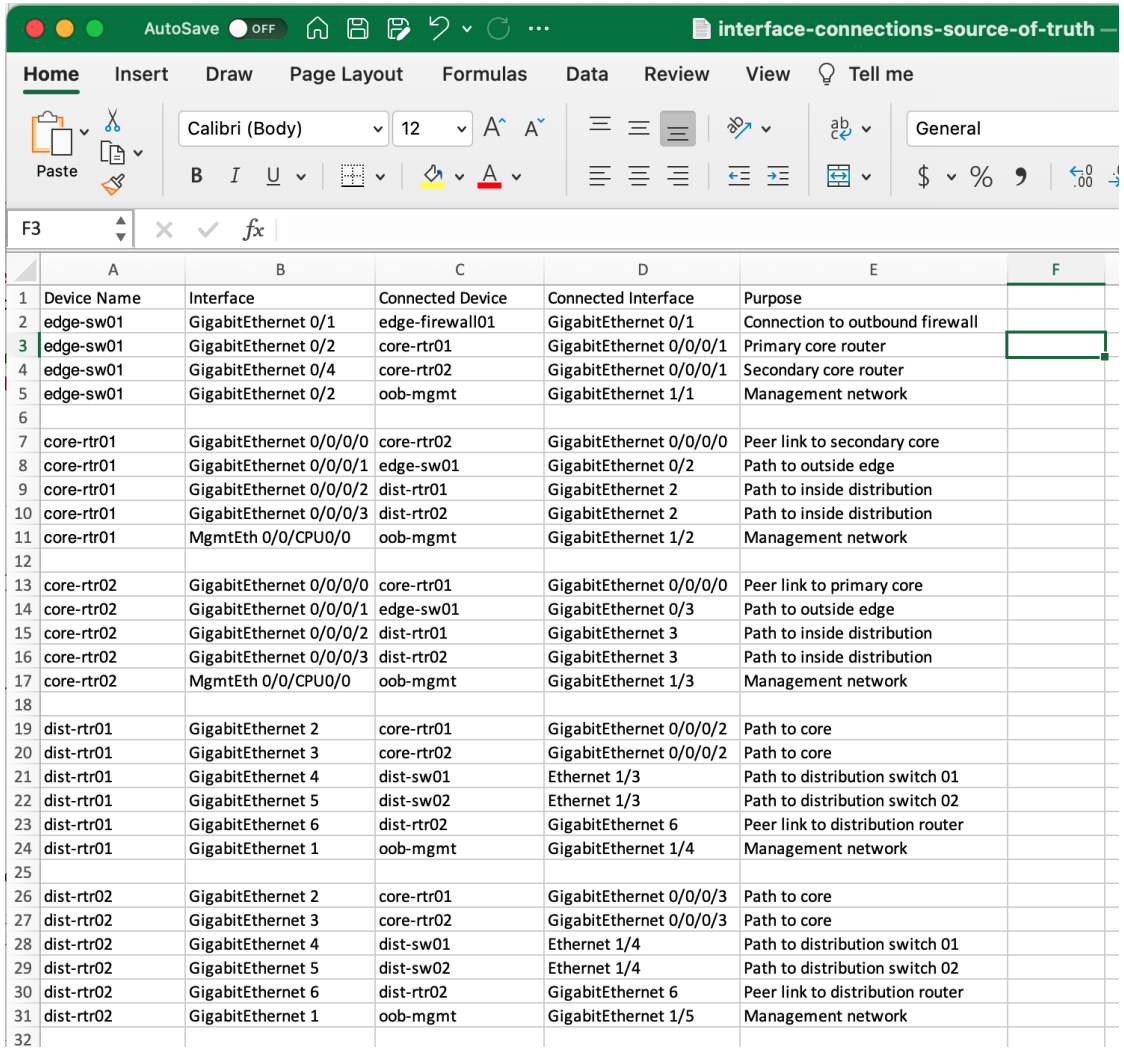
Run the script

- Verifying the inputs work as expected important before using the script

```
./config_interface_descriptions.py --testbed improved_nso_sandbox_testbed.yaml \
   --sot interface-connections-source-of-truth.csv --apply

# OUTPUT
Deploying standard interface descriptions to network.
Generating interface descriptions from file interface-connections-source-of-truth.xlsx for testbed improved_nso_sandbox_testbed.yaml.
Configurations will be applied to network devices.


################################################################################

./config_interface_descriptions.py --testbed improved_nso_sandbox_testbed.yaml \
   --sot interface-connections-source-of-truth.csv

# OUTPUT
Deploying standard interface descriptions to network.
Generating interface descriptions from file interface-connections-source-of-truth.xlsx for testbed improved_nso_sandbox_testbed.yaml.
Configurations will NOT be applied to network devices. They will be output to the screen only.
```

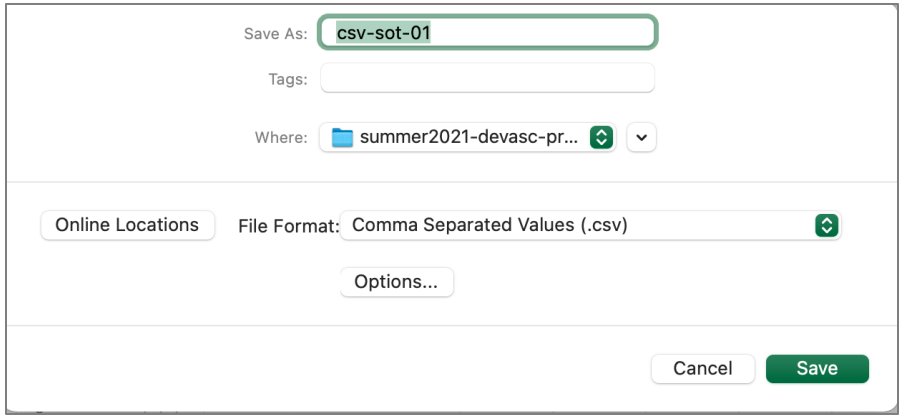
Reading in the Source of Truth Data

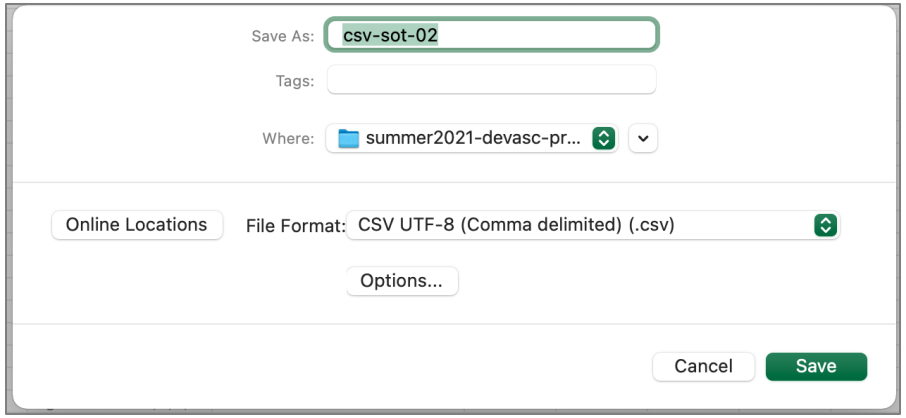
- Once again, spreadsheets are key to network engineering
- Understand the difference between CSV styles
  - Delimiters, quoting, spacing, etc
  - Encoding options – ASCII, UTF8, UTF16



	A	B	C	D	E	F
1	Device Name	Interface	Connected Device	Connected Interface	Purpose	
2	edge-sw01	GigabitEthernet 0/1	edge-firewall01	GigabitEthernet 0/1	Connection to outbound firewall	
3	edge-sw01	GigabitEthernet 0/2	core-rtr01	GigabitEthernet 0/0/0/1	Primary core router	
4	edge-sw01	GigabitEthernet 0/4	core-rtr02	GigabitEthernet 0/0/0/1	Secondary core router	
5	edge-sw01	GigabitEthernet 0/2	oob-mgmt	GigabitEthernet 1/1	Management network	
6						
7	core-rtr01	GigabitEthernet 0/0/0/0	core-rtr02	GigabitEthernet 0/0/0/0	Peer link to secondary core	
8	core-rtr01	GigabitEthernet 0/0/0/1	edge-sw01	GigabitEthernet 0/2	Path to outside edge	
9	core-rtr01	GigabitEthernet 0/0/0/2	dist-rtr01	GigabitEthernet 2	Path to inside distribution	
10	core-rtr01	GigabitEthernet 0/0/0/3	dist-rtr02	GigabitEthernet 2	Path to inside distribution	
11	core-rtr01	MgmtEth 0/0/CPU0/0	oob-mgmt	GigabitEthernet 1/2	Management network	
12						
13	core-rtr02	GigabitEthernet 0/0/0/0	core-rtr01	GigabitEthernet 0/0/0/0	Peer link to primary core	
14	core-rtr02	GigabitEthernet 0/0/0/1	edge-sw01	GigabitEthernet 0/3	Path to outside edge	
15	core-rtr02	GigabitEthernet 0/0/0/2	dist-rtr01	GigabitEthernet 3	Path to inside distribution	
16	core-rtr02	GigabitEthernet 0/0/0/3	dist-rtr02	GigabitEthernet 3	Path to inside distribution	
17	core-rtr02	MgmtEth 0/0/CPU0/0	oob-mgmt	GigabitEthernet 1/3	Management network	
18						
19	dist-rtr01	GigabitEthernet 2	core-rtr01	GigabitEthernet 0/0/0/2	Path to core	
20	dist-rtr01	GigabitEthernet 3	core-rtr02	GigabitEthernet 0/0/0/2	Path to core	
21	dist-rtr01	GigabitEthernet 4	dist-sw01	Ethernet 1/3	Path to distribution switch 01	
22	dist-rtr01	GigabitEthernet 5	dist-sw02	Ethernet 1/3	Path to distribution switch 02	
23	dist-rtr01	GigabitEthernet 6	dist-rtr02	GigabitEthernet 6	Peer link to distribution router	
24	dist-rtr01	GigabitEthernet 1	oob-mgmt	GigabitEthernet 1/4	Management network	
25						
26	dist-rtr02	GigabitEthernet 2	core-rtr01	GigabitEthernet 0/0/0/3	Path to core	
27	dist-rtr02	GigabitEthernet 3	core-rtr02	GigabitEthernet 0/0/0/3	Path to core	
28	dist-rtr02	GigabitEthernet 4	dist-sw01	Ethernet 1/4	Path to distribution switch 01	
29	dist-rtr02	GigabitEthernet 5	dist-sw02	Ethernet 1/4	Path to distribution switch 02	
30	dist-rtr02	GigabitEthernet 6	dist-rtr02	GigabitEthernet 6	Peer link to distribution router	
31	dist-rtr02	GigabitEthernet 1	oob-mgmt	GigabitEthernet 1/5	Management network	
32						

The Impact of Encoding Differences with Python





file csv-sot-01.csv
csv-sot-01.csv: ASCII text, with CRLF line terminators

config_interface_descriptions.py --sot csv-sot-01.csv

{'Device Name': 'edge-sw01', 'Interface': 'GigabitEthernet0/1', 'Connected Device': 'edge-firewall01', 'Connected Interface': 'GigabitEthernet0/1', 'Purpose': 'Connection to outbound firewall'}
{'Device Name': 'edge-sw01', 'Interface': 'GigabitEthernet0/2', 'Connected Device': 'core-rtr01', 'Connected Interface': 'GigabitEthernet0/0/0/1', 'Purpose': 'Primary core router'}

file csv-sot-02.csv
csv-sot-02.csv: UTF-8 Unicode (with BOM) text, with CRLF line terminators

config_interface_descriptions.py --sot csv-sot-02.csv

{'\ufeffDevice Name': 'edge-sw01', 'Interface': 'GigabitEthernet0/1', 'Connected Device': 'edge-firewall01', 'Connected Interface': 'GigabitEthernet0/1', 'Purpose': 'Connection to outbound firewall'}
{'\ufeffDevice Name': 'edge-sw01', 'Interface': 'GigabitEthernet0/2', 'Connected Device': 'core-rtr01', 'Connected Interface': 'GigabitEthernet0/0/0/1', 'Purpose': 'Primary core router'}

Some info on \ufeff and encoding from StackOverflow

Reading in CSV Data, part 1

- Tip: Adding a “\n” to a print statement will add an extra empty line
- The DictReader will allow accessing row data by key name
  - Default is to read keys from first row
  - This can be overridden if first row does NOT contain headers

```
# Read data from CSV source of truth
print("Opening and readying Source of Truth File.\n")

with open(args.sot, "r") as sot_file:
    sot = csv.DictReader(sot_file)


    # Loop over each row in the Source of Truth
    for row in sot:
        # For debugging, print out the raw data
        print(row)
```

Run the script

- Reading data looks good
- But the empty rows in the file show up as rows in the reader, will need to check for good data

```
./config_interface_descriptions.py --testbed improved_nso_sandbox_testbed.yaml \
   --sot interface-connections-source-of-truth.csv

# OUTPUT
Deploying standard interface descriptions to network.
Generating interface descriptions from file interface-connections-source-of-truth.csv for testbed improved_nso_sandbox_testbed.yaml.
Configurations will NOT be applied to network devices. They will be output to the screen only.

Opening and readying Source of Truth File.

{'Device Name': 'edge-sw01', 'Interface': 'GigabitEthernet0/1', 'Connected Device': 'edge-firewall01', 'Connected Interface': 'GigabitEthernet0/1', 'Purpose': 'Connection to outbound firewall'}
{'Device Name': 'edge-sw01', 'Interface': 'GigabitEthernet0/2', 'Connected Device': 'core-rtr01', 'Connected Interface': 'GigabitEthernet0/0/0/1', 'Purpose': 'Primary core router'}
{'Device Name': '', 'Interface': '', 'Connected Device': '', 'Connected Interface': '', 'Purpose': ''}
{'Device Name': 'core-rtr01', 'Interface': 'GigabitEthernet0/0/0/0', 'Connected Device': 'core-rtr02', 'Connected Interface': 'GigabitEthernet0/0/0/0', 'Purpose': 'Peer link to secondary core'}
{'Device Name': 'core-rtr01', 'Interface': 'GigabitEthernet0/0/0/1', 'Connected Device': 'edge-sw01', 'Connected Interface': 'GigabitEthernet0/2', 'Purpose': 'Path to outside edge'}
{'Device Name': '', 'Interface': '', 'Connected Device': '', 'Connected Interface': '', 'Purpose': ''}
```

Reading in CSV Data, part 2

- Comment out the debug print statement
- Provide “padding” to f-strings to help line up output

```
# Read data from CSV source of truth
print("Opening and readying Source of Truth File.\n")

with open(args.sot, "r") as sot_file:
    sot = csv.DictReader(sot_file)

    # Loop over each row in the Source of Truth
    for row in sot:
        # For debugging, print out the raw data of the row
        # print(row)

        # Status message on interface being processed
        if row["Device Name"]:
            print(f'Device {row["Device Name"]:15} Interface {row["Interface"]:25} SOT connection: {row["Connected Device"]} {row["Connected Interface"]}')
```

Run the script

- We can read the data, and pull the information from the table for use

```
./config_interface_descriptions.py --testbed improved_nso_sandbox_testbed.yaml \
   --sot interface-connections-source-of-truth.csv

# OUTPUT
Deploying standard interface descriptions to network.
Generating interface descriptions from file interface-connections-source-of-truth.csv for testbed improved_nso_sandbox_testbed.yaml.
Configurations will NOT be applied to network devices. They will be output to the screen only.

Opening and readying Source of Truth File.

Device edge-sw01      Interface GigabitEthernet0/1      SOT connection: edge-firewall01 GigabitEthernet0/1
Device edge-sw01      Interface GigabitEthernet0/2      SOT connection: core-rtr01 GigabitEthernet0/0/0/1
Device core-rtr01     Interface GigabitEthernet0/0/0/0  SOT connection: core-rtr02 GigabitEthernet0/0/0/0
Device core-rtr01     Interface GigabitEthernet0/0/0/1  SOT connection: edge-sw01 GigabitEthernet0/2
Device core-rtr01     Interface MgmtEth 0/0/CPU0/0      SOT connection: oob-mgmt GigabitEthernet1/2
Device dist-rtr01     Interface GigabitEthernet2        SOT connection: core-rtr01 GigabitEthernet0/0/0/2
Device dist-rtr01     Interface GigabitEthernet3        SOT connection: core-rtr02 GigabitEthernet0/0/0/2
Device dist-rtr01     Interface GigabitEthernet4        SOT connection: dist-sw01 Ethernet 1/3
Device dist-sw01      Interface Ethernet 1/1            SOT connection: dist-sw02 Ethernet 1/1
Device dist-sw01      Interface Ethernet 1/2            SOT connection: dist-sw02 Ethernet 1/2
```

Creating the Interface Configuration Template

- Jinja2 is a widely used string formatting/templating framework
  - Popular in Python and Ansible
- Common to create template files that are read into code
- Jinja supports programming logic like loops and conditionals within templates
  - Alternative to logic in Python script
- Our use case is a VERY simple template

```
# Contents of interface_config_template.j2

interface {{interface_name}}
  description Connected to {{connected_device}} {{connected_interface}} – {{purpose}}}
```

Creating new configurations from the template

- If you have multiple templates, a Jinja Environment is more commonly used to load them
- The defaultdict object is a variation on a standard dictionary
  - When a non-existing key is accessed, it is initialized with the default value
  - This allows us to easily add new devices and interface configs to the new_config variable

```
from jinja2 import Template
from collections import defaultdict

# Create Jinja Template for Interface configuration
with open("interface_config_template.j2") as f:
    interface_template = Template(f.read())

# defaultdict variable for holding configurations
new_config = defaultdict(dict)

######

# Loop over each row in the Source of Truth
for row in sot:
    if row["Device Name"]:
        # Generate desired configurations
        new_config[row["Device Name"]][row["Interface"]] =
          interface_template.render(
            interface_name=row["Interface"],
            connected_device=row["Connected Device"],
            connected_interface=row["Connected Interface"],
            purpose=row["Purpose"]
          )

# For debugging, print out the new_configurations data
print("Jinja Template rendered configuration data.")
print(new_configurations)
```

Run the script

- We can see the rendered device configurations in the dictionary
- When printed this way, the config looks like a single line, but the \n indicates carriage returns

```
./config_interface_descriptions.py --testbed improved_nso_sandbox_testbed.yaml \
   --sot interface-connections-source-of-truth.csv

# OUTPUT
.
Opening and readying Source of Truth File.

Device edge-sw01      Interface GigabitEthernet0/1      SOT connection: edge-firewall01 GigabitEthernet0/1
Device edge-sw01      Interface GigabitEthernet0/2      SOT connection: core-rtr01 GigabitEthernet0/0/0/1
Device dist-sw01      Interface Ethernet 1/2            SOT connection: dist-sw02 Ethernet ½

Jinja Template rendered configuration data.
defaultdict(<class 'dict'>, {'edge-sw01': {'GigabitEthernet0/1': 'interface GigabitEthernet0/1\n  description Connected to edge-firewall01 GigabitEthernet0/1 – Connection to outbound firewall}', 'GigabitEthernet0/2': 'interface GigabitEthernet0/2\n  description Connected to oob-mgmt GigabitEthernet1/1 – Management network}', 'GigabitEthernet0/4': 'interface GigabitEthernet0/4\n  description Connected to core-rtr02 GigabitEthernet0/0/0/1 – Secondary core router}'}, 'core-rtr01': {'GigabitEthernet0/0/0/0': 'interface GigabitEthernet0/0/0/0\n  description Connected to core-rtr02 GigabitEthernet0/0/0/0 – Peer link to secondary core}',
'GigabitEthernet0/0/0/1': 'interface GigabitEthernet0/0/0/1\n  description Connected to edge-sw01 GigabitEthernet0/2 – Path to outside edge}',
'GigabitEthernet0/0/0/2': 'interface GigabitEthernet0/0/0/2\n  description Connected to dist-rtr01 GigabitEthernet2 – Path to inside distribution }',...
```

Printing out rendered configuration for verification

- When no longer needed, comment out debugs
- We want to print out the configs for users to review, and possibly manually apply (copy/paste)
- for loops are very useful in Python
  - Looping over .items() of a dict is a useful practice

```
# For debugging, print out the new_configurations data
# print("Jinja Template rendered configuration data.")
# print(new_configurations)


# Display the new configurations for the devices to the user for verifications.
print("New Device Configurations for Descriptions")
print("----------------------------------------")

for device, interfaces in new_configurations.items():
    print(f"! Device {device}")
    for interface_name, interface_config in interfaces.items():
        print(interface_config)
    print("!\n")
```

Run the script

- We are nearly done with the core asks from this use case assignment!
- Before we move to pushing the configuration to the network, let's gather the current descriptions

```
./config_interface_descriptions.py --testbed improved_nso_sandbox_testbed.yaml \
   --sot interface-connections-source-of-truth.csv

# OUTPUT
.
.

New Device Configurations for Interface Descriptions
--------------------------------------------------
! Device edge-sw01
interface GigabitEthernet0/1
 description Connected to edge-firewall01 GigabitEthernet0/1 – Connection to outbound firewall
interface GigabitEthernet0/2
 description Connected to core-rtr01 GigabitEthernet0/0/0/1 – Primary core router
!

! Device core-rtr01
interface GigabitEthernet0/0/0/0
 description Connected to core-rtr02 GigabitEthernet0/0/0/0 – Peer link to secondary core
interface GigabitEthernet0/0/0/1
 description Connected to edge-sw01 GigabitEthernet0/2 – Path to outside edge
```

Review

Connecting and Disconnecting to the devices

- As we did in our previous use case
  - Load the testbed file given in arguments
  - Connect to all devices in the testbed
  - Loop over devices to disconnect when done with them

```
from pyats.topology.loader import load

# Load pyATS testbed and connect to devices
print(f"Loading testbed file {args.testbed}")
testbed = load(args.testbed)
print(f"Connecting to all devices in testbed {testbed.name}")
testbed.connect(log_stdout=False)


# Lookup current interface descriptions
# Apply new interface description configuration (with confirmation)
# Gather CDP/LLDP neighbor details from devices
# Check if neighbor details match Source of Truth


# Disconnect from devices
for device in testbed.devices:
    print(f"Disconnecting from device {device}.")
    testbed.devices[device].disconnect()
```

Run the script

- And ready to start gathering details from the network

```
./config_interface_descriptions.py --testbed improved_nso_sandbox_testbed.yaml \
   --sot interface-connections-source-of-truth.csv

# OUTPUT
.

New Device Configurations for Interface Descriptions
----------------------------------------------------
! Device edge-sw01
interface GigabitEthernet0/1
  description Connected to edge-firewall01 GigabitEthernet0/1 – Connection to outbound firewall
.
Loading testbed file improved_nso_sandbox_testbed.yaml
Enter default password for testbed:

Enter value for testbed.credentials.default.username: cisco
Enter enable password for testbed:
Connecting to all devices in testbed improved_nso_sandbox_testbed
Disconnecting from device edge-firewall01.
Disconnecting from device dist-rtr01.
Disconnecting from device internet-rtr01.
```

Learning Interface State with pyATS

- device.learn() with pyATS
  - Runs and parses several commands on the device
  - Returns the data in a model common across platforms (XE, XR, NX)
  - Removes need for OS specific data access
- Dealing with 2 different source of data
  - We only care about ***devices in the SoT***
  - But we need to prepare for a device in SoT but ***not*** in the Testbed
    - Try/Expect with KeyError ✔

```
# Lookup current interface descriptions - But only for devices in SoT
current_interface_details = {}
for device in new_configurations.keys():
    try:
        print(f'Learning current state on device {device}')
        current_interface_details[device] =\
          testbed.devices[device].learn("interface")
    except KeyError:
        print(f" ⚠ Error: Device {device} from Source of Truth is NOT in the testbed")


# For debugging, print the current_interface_details
print("Output from learn interface operation")
print(current_interface_details)
```

Run the script

- Okay, we've learned the Interface state
- But the data is an "Interface object" - Let's check the docs

```
./config_interface_descriptions.py --testbed improved_nso_sandbox_testbed.yaml \
   --sot interface-connections-source-of-truth.csv

# OUTPUT
.

Connecting to all devices in testbed improved_nso_sandbox_testbed
Learning current interface state for device edge-sw01
Learning current interface state for device core-rtr01
Learning current interface state for device dist-sw01
Learning current interface state for device dist-sw02
Output from learn interface operation
{'edge-sw01': <Interface object at 0x400581c400>, 'core-rtr01': <Interface object at 0x405bf07a30>, 'core-rtr02': <Interface object at 0x405bf23eb0>, 'dist-rtr01':
<Interface object at 0x40599ddd60>, 'dist-rtr02': <Interface object at 0x40591f18e0>, 'dist-sw01': <Interface object at 0x405c82bd00>, 'dist-sw02': <Interface object at
0x405cac7dc0>}

Disconnecting from device edge-firewall01.
Disconnecting from device dist-rtr01.
Disconnecting from device internet-rtr01.
```

Exploring the Interface Model Docs for pyATS

- An important skill for a DevNet Associate is ability to read developer docs and guides
- pyATS provides documentation for on DevNet for every model, parser, trigger, etc
  - Interface Model Doc
- Interface Description should be available at:
  data.info[”GigabitEthernet1”][“description”]

key

value

Interface Ops structure

show commands

IOS-XE	IOS-XR	NX-OS
show interfaces show vrf detail show ip interface show ipv6 interface show interface switchport show etherchannel summary show interfaces [intf] accounting	show interfaces detail show vlan interface show vrf all detail show ipv4 vrf all interface show ipv6 vrf all interface show bundle show interfaces [intf] accounting	show interface show vrf all interface show ip interface vrf all show ipv6 interface vrf all show interface switchport show routing ipv6 vrf all show routing vrf all

Ops structure

NOTE: where self represents the instance object of the ops object

```
self.info {
  interface: {                          # Ops/Conf 'Ethernet1/1'|'Vlan10'|'Ethernet1/1.10'
    'description': description,         # Ops/Conf 'To R2'
    'type': type,                       # Ops/Conf '10/100/1000 Ethernet'
    'oper_status': oper_status,         # Ops 'up'|'down'
    'last_change': last_change,         # Ops '00:00:04'|'never'
    'phys_address': phys_address,       # Ops '5254.009c.f2e6'
    'mtu': mtu,                         # Ops/Conf '1500 bytes'
    'enabled': enabled,                 # Ops/Conf True(no shut)|False(shut)
    'vlan_id': vlan_id,                 # Ops '10'
```

Learning Interface State with pyATS – Better Debug of Data

- By reviewing the data model for learn("interface") we can access the description data
- Two loops over .items()
  - device (name) , interfaces (model)
  - interface (name) , details (for interface)
- Always be prepared to catch KeyErrors when accessing a dictionary (like) object

```
# Lookup current interface descriptions - But only for devices in SoT
current_interface_details = {}
for device in new_configurations.keys():
    try:
        print(f'Learning current interface state for device {device}')
        current_interface_details[device] = testbed.devices[device].learn("interface")
    except KeyError:
        print(f" ⚠ Error: Device {device} from Source of Truth is NOT in the testbed")


# For debugging, print the current_interface_details
print("Output from learn interface operation")

for device, interfaces in current_interface_details.items():
    print(f'Device {device} Current Interface Descriptions are: ')

    for interface, details in interfaces.info.items():
        try:
            print(f'  {interface} : {details["description"]}')
        # Interfaces without descriptions won't have the key
        except KeyError:
            print(f'  {interface} : ')
```

Run the script

- We can successfully retrieve the current interface descriptions

```
./config_interface_descriptions.py --testbed improved_nso_sandbox_testbed.yaml \
    --sot interface-connections-source-of-truth.csv

# OUTPUT
.

Device edge-sw01 Current Interface Descriptions are:
  Loopback0 : to
  GigabitEthernet0/3 : to GigabitEthernet0/0/0/1.core-rtr02
  GigabitEthernet0/2 : to GigabitEthernet0/0/0/1.core-rtr01
  GigabitEthernet0/1 : to GigabitEthernet0/1.edge-firewall01
  GigabitEthernet0/0 : to port3.sandbox-backend
Device core-rtr01 Current Interface Descriptions are:
  GigabitEthernet0/0/0/3 : L3 Link to dist-rtr02
  Loopback0 : test
  Null0 :
Device dist-rtr01 Current Interface Descriptions are:
  Loopback0 : to
  GigabitEthernet6 : L3 Link to dist-rtr02
  GigabitEthernet5 : L3 Link to dist-sw02
.
```

Recording the Current Descriptions into the SoT - planning

- Before we start changing configurations, let's save the capture current configuration
- First… Decision Time
  - Do we create a "report" with the updated data?
  - Or update the source SoT itself
- Considerations
  - Avoid destroying data
  - Avoid replicating data

```
# Update Source of Truth with Results
# Open the SoT again to process read the data for the report
with open(args.sot, "r") as sot_file:
    sot = csv.DictReader(sot_file)

    # Create a report file based on date/time
    now = datetime.now()
    report_name = f'{now.strftime("%Y-%m-%d-%H-%M-
%S")}_interface_config_report.csv'


    print(f'Writing config report to file {report_name}.')
    # Open the new report file

    # Create field names list for report by adding new
    # fields to the SoT fields

    # Create the DictWriter object for the report output

    # Loop over each row in the Source of Truth

    # Retrieve the current description from the learned
    # data

    # Write report row to file
```

Run the script

- No report data yet, but we've got our file name and a plan

```
./config_interface_descriptions.py --testbed improved_nso_sandbox_testbed.yaml \
    --sot interface-connections-source-of-truth.csv

# OUTPUT
.

! Device dist-sw02
interface Ethernet1/1
  description Connected to dist-sw01 Ethernet1/1 – Peer link to distribution switch 01
interface Ethernet1/2
  description Connected to dist-sw01 Ethernet1/2 – Peer link to distribution switch 01
interface Ethernet1/3
!

Loading testbed file improved_nso_sandbox_testbed.yaml
Connecting to all devices in testbed improved_nso_sandbox_testbed
Learning current interface state for device edge-sw01
Learning current interface state for device dist-sw02
Disconnecting from devices.
Writing config report to file 2021-06-03-20-22-41_interface_config_report.csv.
```

Recording the Current Descriptions into the SoT – coding it up

```
# Update Source of Truth with Results
# Open the SoT again to process read the data for the report
with open(args.sot, "r") as sot_file:
    sot = csv.DictReader(sot_file)

    # Create a report file based on date/time
    now = datetime.now()
    report_name = f'{now.strftime("%Y-%m-%d-%H-%M-%S")}_interface_config_report.csv'

    print(f'Writing config report to file {report_name}.')
    with open(report_name, 'w', newline='') as report_file:
        # Create field names list for report by adding new fields to the SoT fields
        report_fields = sot.fieldnames + ["Previous Interface Description"]

        report = csv.DictWriter(report_file, fieldnames=report_fields)
        report.writeheader()

        # Loop over each row in the Source of Truth
        for row in sot:
            # Retrieve the current description from the learned data
            try:
                row["Previous Interface Description"] = \
                    current_interface_details[row["Device Name"]].info[row["Interface"]]["description"]
            # If a KeyError found indicating no current description, or device not in testbed skip data
            except KeyError:
                row["Previous Interface Description"] = ""

            # Write report row to file
            report.writerow(row)
```

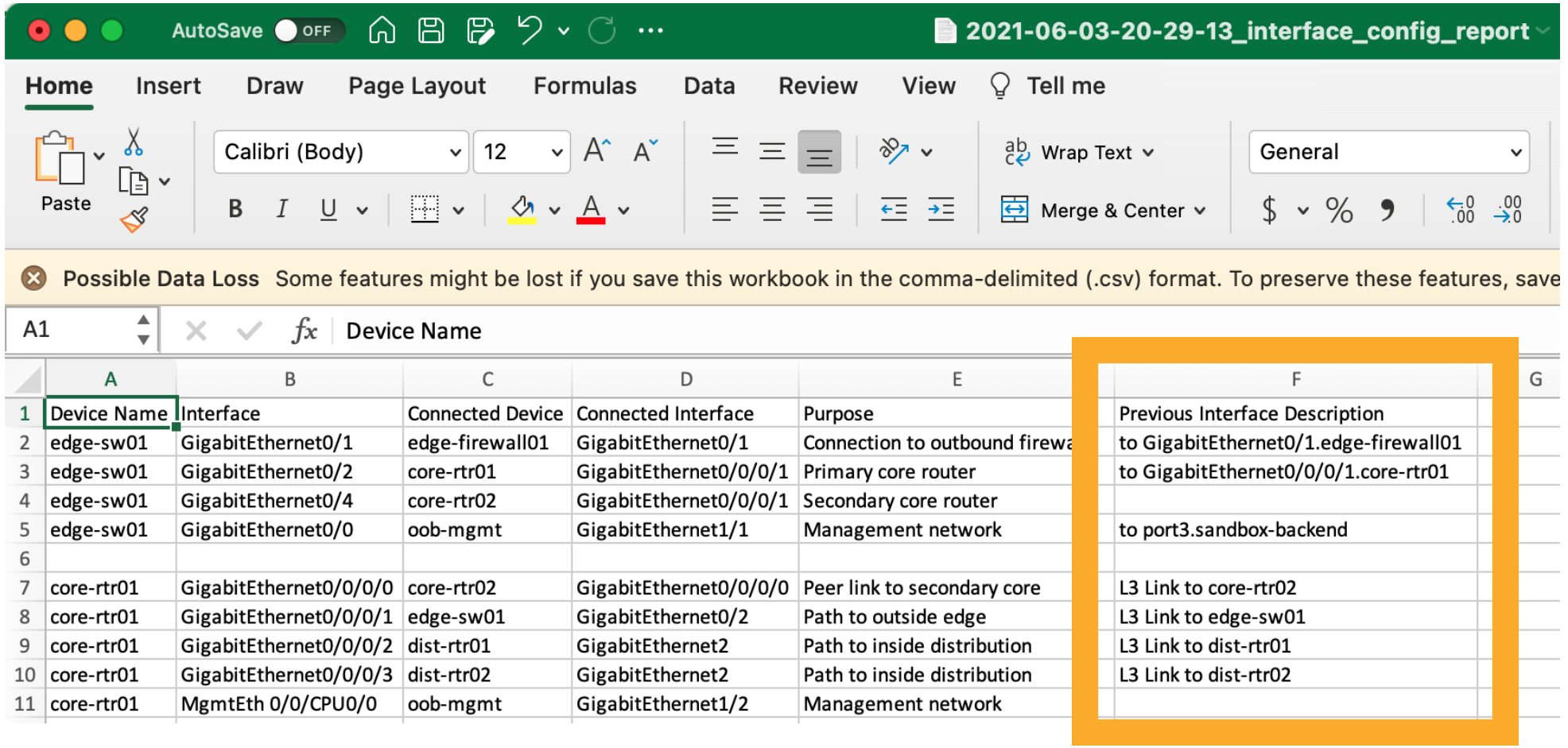
Run the script

- Do we have data

```
./config_interface_descriptions.py --testbed improved_nso_sandbox_testbed.yaml \
   --sot interface-connections-source-of-truth.csv

# OUTPUT
.

! Device dist-sw02
interface Ethernet1/1
  description Connected to dist-sw01 Ethernet1/1 – Peer link to distribution switch 01
interface Ethernet1/2
  description Connected to dist-sw01 Ethernet1/2 – Peer link to distribution switch 01
interface Ethernet1/3
!

Loading testbed file improved_nso_sandbox_testbed.yaml
Connecting to all devices in testbed improved_nso_sandbox_testbed
Learning current interface state for device edge-sw01
Learning current interface state for device dist-sw02
Disconnecting from devices.
Writing config report to file 2021-06-03-20-29-13_interface_config_report.csv.
```

Run the script

- Yes we do!



	A	B	C	D	E	F	G
1	Device Name	Interface	Connected Device	Connected Interface	Purpose	Previous Interface Description	
2	edge-sw01	GigabitEthernet0/1	edge-firewall01	GigabitEthernet0/1	Connection to outbound firewa	to GigabitEthernet0/1.edge-firewall01	
3	edge-sw01	GigabitEthernet0/2	core-rtr01	GigabitEthernet0/0/0/1	Primary core router	to GigabitEthernet0/0/0/1.core-rtr01	
4	edge-sw01	GigabitEthernet0/4	core-rtr02	GigabitEthernet0/0/0/1	Secondary core router		
5	edge-sw01	GigabitEthernet0/0	oob-mgmt	GigabitEthernet1/1	Management network	to port3.sandbox-backend	
6							
7	core-rtr01	GigabitEthernet0/0/0/0	core-rtr02	GigabitEthernet0/0/0/0	Peer link to secondary core	L3 Link to core-rtr02	
8	core-rtr01	GigabitEthernet0/0/0/1	edge-sw01	GigabitEthernet0/2	Path to outside edge	L3 Link to edge-sw01	
9	core-rtr01	GigabitEthernet0/0/0/2	dist-rtr01	GigabitEthernet2	Path to inside distribution	L3 Link to dist-rtr01	
10	core-rtr01	GigabitEthernet0/0/0/3	dist-rtr02	GigabitEthernet2	Path to inside distribution	L3 Link to dist-rtr02	
11	core-rtr01	MgmtEth 0/0/CPU0/0	oob-mgmt	GigabitEthernet1/2	Management network		

Applying the Interface Configurations - planning

- For better efficiency and logic, changing the order of steps
  - Was
    - Generate new interface configurations
    - Print new configs to screen
    - Connect to network testbed
    - Lookup current configuration
    - Apply new configs to devices
  - New
    - Generate new interface configurations
    - Connect to network testbed
    - Lookup current configuration
    - Print new configs to screen
    - Apply new configs to devices


```
# Loop over each device from SoT.
#  1. Display the new configuration to user
#  2. If --apply was set, ask user if wish to deploy config to device
print("New Device Configurations ")
print("-------------------------")
for device, interfaces in new_configurations.items():
    # Display the new configurations for verifications.
    print(f"! Device {device}")
    for interface_name, interface_config in interfaces.items():
        print(interface_config)
    print("!\n")


    # Apply new interface description configuration
    # Check if --apply was set


    # Ask user if wish to deploy change to device


    # If yes, apply configuration to device
```

Run the script

- No new code, so the output and results shouldn't change
- But even a reorder/refactor of a script needs to be tested

```
./config_interface_descriptions.py --testbed improved_nso_sandbox_testbed.yaml \
   --sot interface-connections-source-of-truth.csv

# OUTPUT
.

! Device dist-sw02
interface Ethernet1/1
  description Connected to dist-sw01 Ethernet1/1 – Peer link to distribution switch 01
interface Ethernet1/2
  description Connected to dist-sw01 Ethernet1/2 – Peer link to distribution switch 01
interface Ethernet1/3
!

Loading testbed file improved_nso_sandbox_testbed.yaml
Connecting to all devices in testbed improved_nso_sandbox_testbed
Learning current interface state for device edge-sw01
Learning current interface state for device dist-sw02
Disconnecting from devices.
Writing config report to file 2021-06-04-15-05-08_interface_config_report.csv..
```

Applying the Interface Configurations – Apply Logic

- The input() function asks user to provide text input an stores in variable
  - Be explicit about what input for a “confirm”.
  - Anything else means NO
- Alternative to try/except is to use use “if” to see if device is in testbed
- Adding blank lines and “dividers” can make script output easier to read for users
  - There are libraries like Rich that provide robust formatting options

```
# Loop over each device from SoT.
#  1. Display the new configuration to user
#  2. If --apply was set, ask user if wish to deploy config
print("New Device Configurations ")
print("-------------------------")
for device, interfaces in new_configurations.items():
    # Display the new configurations for verifications.
    print(f"! Device {device}")
    for interface_name, interface_config in interfaces.items():
        print(interface_config)
    print("!\n")


    # Apply new interface description configuration
    # Check if --apply was set
    if args.apply:
        # Ask user if wish to deploy change to device
        confirm = input(f"Would you like to apply this configuration to device {device} (y/n)? ")
        # If yes, apply configuration to device
        if confirm == "y":
            # Verify the device from the SoT is in the testbed
            if device in testbed.devices:
                print(f"Applying config to device {device}.")
            else:
                print(f" ⚠ Error: {device} NOT in the testbed")


    # Print a divider between devices
    print("\n-----------------------------------------\n")
```

Run the script

- Notice adding the --apply argument to the script is needed
- And difference with answers to the question

```
./config_interface_descriptions.py --testbed improved_nso_sandbox_testbed.yaml \
   --sot interface-connections-source-of-truth.csv --apply

# OUTPUT
.

New Device Configurations for Interface Descriptions
--------------------------------------------------
! Device edge-sw01
interface GigabitEthernet0/1
  description Connected to edge-firewall01 GigabitEthernet0/1 – Connection to outbound firewall
interface GigabitEthernet0/2
  description Connected to core-rtr01 GigabitEthernet0/0/0/1 – Primary core router
interface GigabitEthernet0/4
  description Connected to core-rtr02 GigabitEthernet0/0/0/1 – Secondary core router
interface GigabitEthernet0/0
  description Connected to oob-mgmt GigabitEthernet1/1 – Management network
!

Would you like to apply this configuration to device edge-sw01 (y/n)? y
Applying configuration to device edge-sw01.
```

Run the script

- Notice adding the --apply argument to the script is needed
- And difference with answers to the question

```
.

Applying configuration to device edge-sw01.
! Device core-rtr01
interface GigabitEthernet0/0/0/0
  description Connected to core-rtr02 GigabitEthernet0/0/0/0 – Peer link to secondary core
interface GigabitEthernet0/0/0/1
  description Connected to edge-sw01 GigabitEthernet0/2 – Path to outside edge
interface GigabitEthernet0/0/0/2
  description Connected to dist-rtr01 GigabitEthernet2 – Path to inside distribution
interface GigabitEthernet0/0/0/3
  description Connected to dist-rtr02 GigabitEthernet2 – Path to inside distribution
interface MgmtEth 0/0/CPU0/0
  description Connected to oob-mgmt GigabitEthernet1/2 – Management network
!

Would you like to apply this configuration to device core-rtr01 (y/n)? n

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

! Device core-rtr02
```

Run the script

- Notice adding the --apply argument to the script is needed
- And difference with answers to the question

```
.

----------------------------------------------------
! Device edge-sw01
interface GigabitEthernet0/1
 description Connected to edge-firewall01 GigabitEthernet0/1 – Connection to outbound firewall
interface GigabitEthernet0/2
 description Connected to core-rtr01 GigabitEthernet0/0/0/1 – Primary core router
interface GigabitEthernet0/4
 description Connected to core-rtr02 GigabitEthernet0/0/0/1 – Secondary core router
interface GigabitEthernet0/0
 description Connected to oob-mgmt GigabitEthernet1/1 – Management network
!

Would you like to apply this configuration to device edge-sw01 (y/n)? y
Applying configuration to device edge-sw01.
⚠ Error: Device edge-sw01 from Source of Truth is NOT in the testbed - unable to apply configuration.

----------------------------------------------------
```

Applying the Interface Configurations – Deploy Configuration

- Rather than send a separate “config” for each interface, we combine them to a single string and send once
- pyATS device interaction methods
  - .parse() – Run and process show command
  - .execute() – Return raw output of command
  - .learn() – Leverage a standard model
  - .configure() – Send configuration
- A try/except block important when sending configuration
  - Prevent one device error stopping entire deployment
  - The generic “except Exception” catches any error

```
for interface_name, interface_config in interfaces.items():
    print(interface_config)

####

if confirm == "y":
    # Do a check to verify the device is in the testbed
    if device in testbed.devices:
        print(f"Applying configuration to device {device}.")

        # Try sending configuration to device
        try:
            result = testbed.devices[device].configure(
                # Combine all configs into a single string
                "\n".join(interfaces.values())
                )

            # For debugging, print result
            print(result)

        except Exception as e:
            print(f" ⚠ Error: Applying config to {device}")

            # For debugging, print error to screen
            print(e)


    else:
        print(f" ⚠ Error: {device} is NOT in the testbed")
```

Run the script

- The debug “print(result)” shows the interaction with the device
  - Comment this out once testing is done and things working

```
./config_interface_descriptions.py --testbed improved_nso_sandbox_testbed.yaml \
   --sot interface-connections-source-of-truth.csv --apply

# OUTPUT
.

Would you like to apply this configuration to device core-rtr01 (y/n)? y
Applying configuration to device core-rtr01.
interface GigabitEthernet0/0/0/0
RP/0/0/CPU0:  description Connected to core-rtr02 GigabitEthernet0/0/0/0 - Peer link to secondary core
RP/0/0/CPU0:interface GigabitEthernet0/0/0/1
RP/0/0/CPU0:  description Connected to edge-sw01 GigabitEthernet0/2 - Path to outside edge
RP/0/0/CPU0:interface GigabitEthernet0/0/0/2
RP/0/0/CPU0:  description Connected to dist-rtr01 GigabitEthernet2 - Path to inside distribution
RP/0/0/CPU0:interface GigabitEthernet0/0/0/3
RP/0/0/CPU0:  description Connected to dist-rtr02 GigabitEthernet2 - Path to inside distribution
RP/0/0/CPU0:interface MgmtEth 0/0/CPU0/0
RP/0/0/CPU0:  description Connected to oob-mgmt GigabitEthernet1/2 - Management network
RP/0/0/CPU0:commit
Fri Jun  4 16:52:40.417 UTC
RP/0/0/CPU0:
```

Checking if Interface Connections Match SoT - Planning

- Add additional argument flag to run verification
- Layout the plan of attack in comments
- Configure LLDP to insure running
  - Probably ***NOT*** a step to take without approval in production

```
parser.add_argument('--check-neighbors’,
action='store_true', help="Should we try to use LLDP to verify interface neighbors.
Default is NO.")


###

# Gather CDP/LLDP neighbor details from devices
if args.check_neighbors:
    print(f"Will attempt to check neighbors with LLDP.")


    # Enable LLDP on devices (wait 30s to learn neighbors)
    # Learn neighbor details
    # Check if neighbor details match Source of Truth
    #   Possibilities: Confirmed - LLDP Data Matches SoT
    #                  Incorrect - LLDP Data Diff from SoT
    #                  Unknown   - LLDP Data Not Available
```

Stretch Goal Bonus!

See slides and code on GitHub for breakdown of this development!

Start with [development-steps/09_config_interface_descriptions.py]

Adding Test Results to the Report

- Fairly easy to update the report code to include the test data if ran

```
print(f'Writing config report to file {report_name}.')
with open(report_name, 'w', newline='') as report_file:
    # Create field names list for report by adding new fields to the SoT fields
    report_fields = sot.fieldnames + ["Previous Interface Description"]

    # If neighbor check was completed, add field to report
    if args.check_neighbors:
        report_fields.append("LLDP Neighbor Check Test")

    # Loop over each row in the Source of Truth
    for row in sot:

        # If neighbor check was completed, add data to report
        if args.check_neighbors:
            try:
                row["LLDP Neighbor Check Test"] = test_results[row["Device Name"]][row["Interface"]]
            except KeyError:
                row["LLDP Neighbor Check Test"] = "LLDP Test Not Run."

        # Write report row to file
        report.writerow(row)
```

Run the script

- Execution of final script with both --check-neighbors and --apply set
- Each phase of the script completing with good information for user

```
./config_interface_descriptions.py --testbed improved_nso_sandbox_testbed.yaml \
   --sot interface-connections-source-of-truth.csv --check-neighbors --apply

# OUTPUT
Deploying standard interface descriptions to network.
Generating interface descriptions from file interface-connections-source-of-truth.csv for testbed improved_nso_sandbox_testbed.yaml.
Configurations will be applied to network devices.

Opening and readying Source of Truth File.

Device edge-sw01      Interface GigabitEthernet0/1      SOT connection: edge-firewall01 GigabitEthernet0/1
Device edge-sw01      Interface GigabitEthernet0/2      SOT connection: core-rtr01 GigabitEthernet0/0/0/1
.
Device dist-sw02      Interface Mgmt 0                  SOT connection: oob-mgmt GigabitEthernet1/7

Loading testbed file improved_nso_sandbox_testbed.yaml
Connecting to all devices in testbed improved_nso_sandbox_testbed
Learning current interface state for device edge-sw01
.
Learning current interface state for device dist-sw02
```

1 of 3

Run the script

- Execution of final script with both --check-neighbors and --apply set
- Each phase of the script completing with good information for user

```
.
New Device Configurations for Interface Descriptions
--------------------------------------------------
! Device edge-sw01
interface GigabitEthernet0/1
 description Connected to edge-firewall01 GigabitEthernet0/1 - Connection to outbound firewall
interface GigabitEthernet0/2
 description Connected to core-rtr01 GigabitEthernet0/0/0/1 - Primary core router
interface GigabitEthernet0/3
 description Connected to core-rtr02 GigabitEthernet0/0/0/1 - Secondary core router
interface GigabitEthernet0/0
 description Connected to oob-mgmt GigabitEthernet1/1 - Management network
!


Would you like to apply this configuration to device edge-sw01 (y/n)? y
Applying configuration to device edge-sw01.


--------------------------------------------------
.
```

2 of 3

Run the script

- Execution of final script with both --check-neighbors and --apply set
- Each phase of the script completing with good information for user

```
.
.
Will attempt to check interface neighbors with LLDP.
Configuring LLDP on edge-sw01
Learning LLDP Neighbor Details from on edge-sw01
Configuring LLDP on core-rtr01
Learning LLDP Neighbor Details from on core-rtr01
.
Checking if edge-sw01 GigabitEthernet0/1 is connected to edge-firewall01 GigabitEthernet0/1
  ⚠ No LLDP Info for interface GigabitEthernet0/1 for device edge-sw01
Checking if edge-sw01 GigabitEthernet0/2 is connected to core-rtr01 GigabitEthernet0/0/0/1
  Interface GigabitEthernet0/2 on device edge-sw01 is connected as expected.
.
Checking if dist-rtr02 GigabitEthernet6 is connected to dist-rtr02 GigabitEthernet6
  ⚠ Interface GigabitEthernet6 on device dist-rtr02 is NOT connected as expected.
.
Checking if dist-sw02 Mgmt 0 is connected to oob-mgmt GigabitEthernet1/7
  ⚠ No LLDP Info for interface Mgmt 0 for device dist-sw02
Disconnecting from devices.
Writing config report to file 2021-06-07-00-39-31_interface_config_report.csv.
```

3 of 3

A look at the final report

Look at that, an error was discovered!

	A	B	C	D	E	F	G
1	Device Name	Interface	Connected Device	Connected Interface	Purpose	Previous Interface Description	LLDP Neighbor Check Test
2	edge-sw01	GigabitEthernet0/1	edge-firewall01	GigabitEthernet0/1	Connection to outbound firewall	Connected to edge-firewall01 GigabitEthernet0/1 - Connection to outbound firewall	Unknown - No LLDP Info for Interface
3	edge-sw01	GigabitEthernet0/2	core-rtr01	GigabitEthernet0/0/0/1	Primary core router	Connected to core-rtr01 GigabitEthernet0/0/0/1 - Primary core router	Confirmed
4	edge-sw01	GigabitEthernet0/3	core-rtr02	GigabitEthernet0/0/0/1	Secondary core router	to GigabitEthernet0/0/0/1.core-rtr02	Confirmed
5	edge-sw01	GigabitEthernet0/0	oob-mgmt	GigabitEthernet1/1	Management network	to port3.sandbox-backend	Unknown - No LLDP Info for Interface
6							LLDP Test Not Run.
7	core-rtr01	GigabitEthernet0/0/0/0	core-rtr02	GigabitEthernet0/0/0/0	Peer link to secondary core	Connected to core-rtr02 GigabitEthernet0/0/0/0 - Peer link to secondary core	Confirmed
8	core-rtr01	GigabitEthernet0/0/0/1	edge-sw01	GigabitEthernet0/2	Path to outside edge	Connected to edge-sw01 GigabitEthernet0/2 - Path to outside edge	Confirmed
9	core-rtr01	GigabitEthernet0/0/0/2	dist-rtr01	GigabitEthernet2	Path to inside distribution	Connected to dist-rtr01 GigabitEthernet2 - Path to inside distribution	Confirmed
10	core-rtr01	GigabitEthernet0/0/0/3	dist-rtr02	GigabitEthernet2	Path to inside distribution	Connected to dist-rtr02 GigabitEthernet2 - Path to inside distribution	Confirmed
11	core-rtr01	MgmtEth 0/0/CPU0/0	oob-mgmt	GigabitEthernet1/2	Management network		Unknown - No LLDP Info for Interface
12							LLDP Test Not Run.
13	core-rtr02	GigabitEthernet0/0/0/0	core-rtr01	GigabitEthernet0/0/0/0	Peer link to primary core	L3 Link to core-rtr01	Confirmed
14	core-rtr02	GigabitEthernet0/0/0/1	edge-sw01	GigabitEthernet0/3	Path to outside edge	L3 Link to edge-sw01	Confirmed
15	core-rtr02	GigabitEthernet0/0/0/2	dist-rtr01	GigabitEthernet3	Path to inside distribution	L3 Link to dist-rtr01	Confirmed
16	core-rtr02	GigabitEthernet0/0/0/3	dist-rtr02	GigabitEthernet3	Path to inside distribution	L3 Link to dist-rtr02	Confirmed
17	core-rtr02	MgmtEth 0/0/CPU0/0	oob-mgmt	GigabitEthernet1/3	Management network		Unknown - No LLDP Info for Interface
18							LLDP Test Not Run.
19	dist-rtr01	GigabitEthernet2	core-rtr01	GigabitEthernet0/0/0/2	Path to core	Connected to core-rtr01 GigabitEthernet0/0/0/2 - Path to core	Confirmed
20	dist-rtr01	GigabitEthernet3	core-rtr02	GigabitEthernet0/0/0/2	Path to core	Connected to core-rtr02 GigabitEthernet0/0/0/2 - Path to core	Confirmed
21	dist-rtr01	GigabitEthernet4	dist-sw01	Ethernet1/3	Path to distribution switch 01	Connected to dist-sw01 Ethernet1/3 - Path to distribution switch 01	Confirmed
22	dist-rtr01	GigabitEthernet5	dist-sw02	Ethernet1/3	Path to distribution switch 02	Connected to dist-sw02 Ethernet1/3 - Path to distribution switch 02	Confirmed
23	dist-rtr01	GigabitEthernet6	dist-rtr02	GigabitEthernet6	Peer link to distribution router	Connected to dist-rtr02 GigabitEthernet6 - Peer link to distribution router	Confirmed
24	dist-rtr01	GigabitEthernet1	oob-mgmt	GigabitEthernet1/4	Management network	Connected to oob-mgmt GigabitEthernet1/4 - Management network	Unknown - No LLDP Info for Interface
25							LLDP Test Not Run.
26	dist-rtr02	GigabitEthernet2	core-rtr01	GigabitEthernet0/0/0/3	Path to core	L3 Link to core-rtr01	Confirmed
27	dist-rtr02	GigabitEthernet3	core-rtr02	GigabitEthernet0/0/0/3	Path to core	L3 Link to core-rtr02	Confirmed
28	dist-rtr02	GigabitEthernet4	dist-sw01	Ethernet1/4	Path to distribution switch 01	L3 Link to dist-sw01	Confirmed
29	dist-rtr02	GigabitEthernet5	dist-sw02	Ethernet1/4	Path to distribution switch 02	L3 Link to dist-sw02	Confirmed
30	dist-rtr02	GigabitEthernet6	dist-rtr02	GigabitEthernet6	Peer link to distribution router	L3 Link to dist-rtr01	Incorrect
31	dist-rtr02	GigabitEthernet1	oob-mgmt	GigabitEthernet1/5	Management network	to port7.sandbox-backend	Unknown - No LLDP Info for Interface
32							LLDP Test Not Run.
33	dist-sw01	Ethernet1/1	dist-sw02	Ethernet1/1	Peer link to distribution switch 02	Connected to dist-sw02 Ethernet1/1 - Peer link to distribution switch 02	Confirmed

Considerations on this Use Case and Development

- With planning and breaking down a use case, even complex tasks can be successfully automated
- We didn't use ***any*** functions in this script
  - When you use common tools/libraries across use cases you will often find yourself repeating code/functions
  - Great opportunity to create common "utility" libraries to reuse across projects
  - Examples: Loading, Connecting and Disconnecting a Testbed, Parsing Commands
- Scope Creep is a potential problem for any project
  - Assignment – Deploy the correct Interface Descriptions based on SoT
  - "Creep"
    - "Confirmation Dialogs"
    - Logging previous descriptions
    - Testing Connectivity
  - Important to know when to "stop"

DevNet Associate Blueprint Topics To Be Used

1.5 Explain the benefits of organizing code into methods / functions, classes and modules

3.1 Construct a Python script that uses a Cisco SDK given SDK documentation

3.7 Identify the appropriate DevNet resource for a given scenario (Sandbox, Code Exchange, support, forums, Learning Labs, and API documentation)

3.9 Construct code to perform a specific operation based on a set of requirements and given API reference

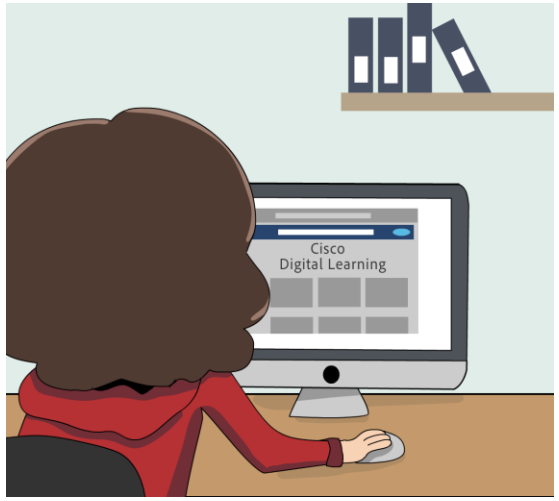
4.8 Identify application security issues related to secret protection, encryption (storage and transport), and data handling

5.5 Describe principles of infrastructure as code

Closing Down

Webinar Resources

- Code for this use case on GitHub and Code Exchange
  - Cisco NSO Reservable Sandbox
- pyATS Getting Started Guide
  - pyATS Genie APIs, Models, and Parsers
- DevNet Associate Free Training Plan
- DevNet Associate Exam Topics List
- Python CSV Library
- Python Collection Datatypes
- Jinja Templates Documentation



Next Webinar - June 16th

Software Defined Network Inventory – Adding ACI and SD-WAN

Your work on an automated network inventory report for your switches and routers was such a big hit that the manager of the SDN team would like you to add the hardware from the ACI and SD-WAN networks to the report. This will be your first chance to explore the REST APIs for these tools!

DevNet Associate Prep Program

cisco

DevNet Prep: Your ultimate FREE self-study resource.

Join now and access:

- Exclusive webinars
- Self-study learning map
- Practice quiz

Register now

DevNet Associate Essentials Webinar Series

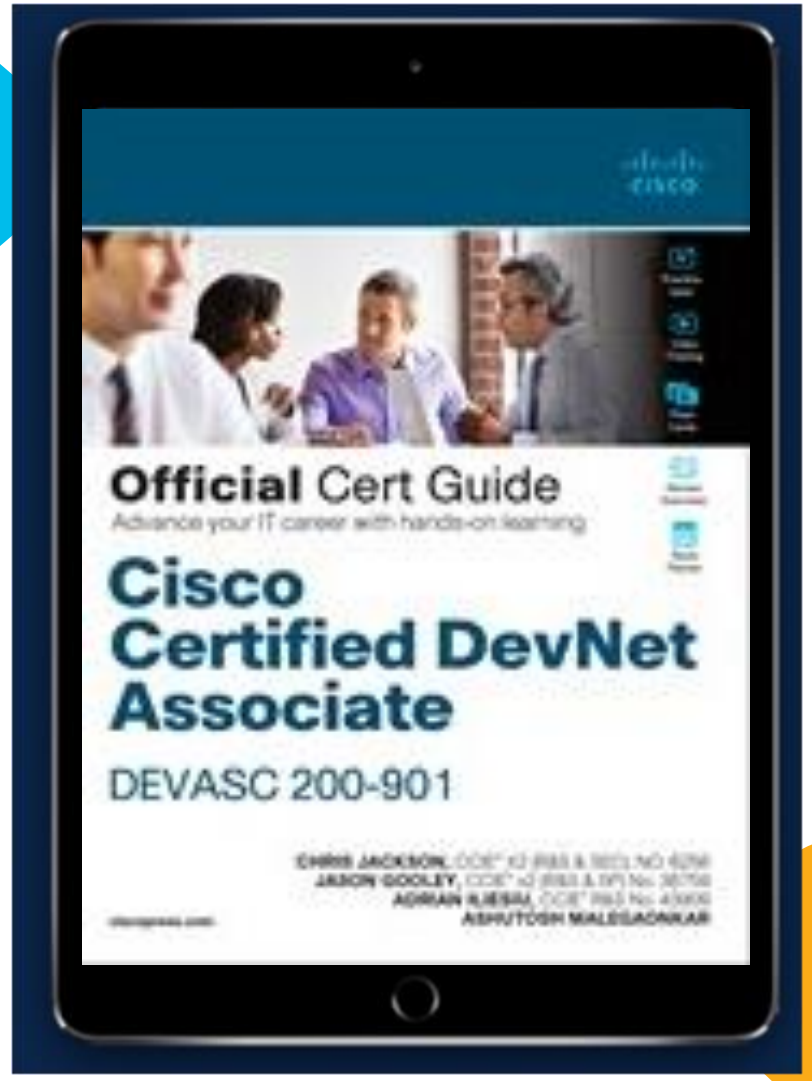
Get a deep dive of the DevNet Associate in this three-part webinar series.

DevNet Associate Bundle

Purchase E-learning and exam bundles and save 15%.

Cisco Press Offer

Special Offer:
Save 40% off with code DEVNET



Post-Webinar Discussion

If you're interested in continuing the conversation, head over to the post-webinar discussion forum!

Karlo - Community Manager asked a question.
Edited 19h ago

DevNet Associate Prep - Enforcing Interface Configuration Standards through Automation

Hello All,

Please consider this an open discussion thread for additional Q&A from the DevNet Associate Prep - Enforcing Interface Configuration Standards through Automation webinar on Tuesday, June 8, 2021 at 1:00 pm Pacific Time.

To sign-up for this webinar and other upcoming webinars in our DevNet Associate Exam Prep Program series, please visit: DevNet Associate Prep Program

Please Note: The on-demand recording of this webinar will be available within 5 business days after the live event. An update will be posted on this thread when it becomes available.

Thank you very much!
Karlo Bobiles
Cisco Learning Network

DEVNET CERTIFICATIONS COMMUNITY

The bridge to possible