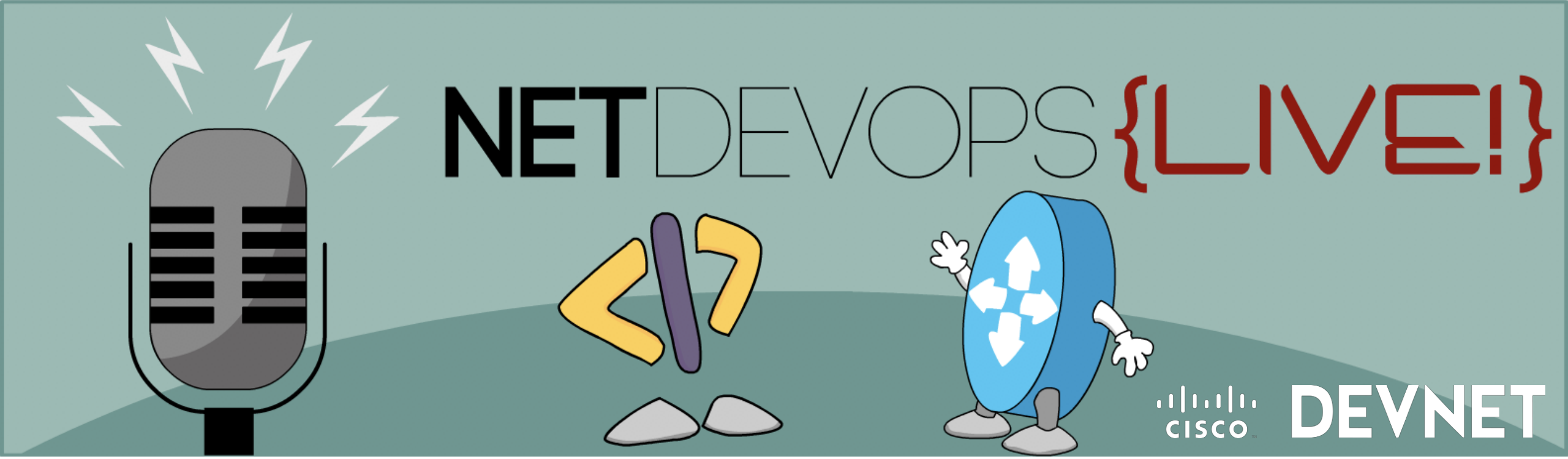


Embrace the DRY Principal with Network Configuration Templates

Bryan Byrne
Technical Solutions Architect
@bryan25607

Season 2, Talk 4

https://developer.cisco.com/netdevops/live

http://cs.co/ndl

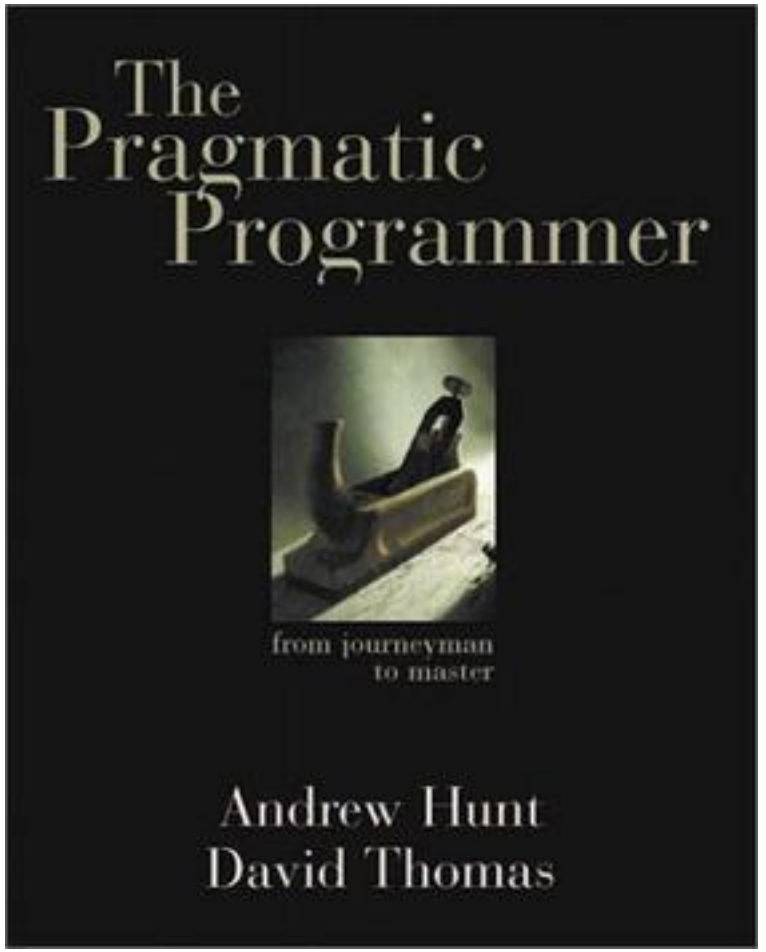
Help us track NetDevOps Live Interest!

Agenda

- What is the DRY Principal?
  - The Foundations for Reusable Code
- Using Jinja to Create CLI Templates
  - Understanding the Jinja Templating Language
  - Using YAML to Provide Structured Input for Variables
- Using TextFSM to put Structure into Show Outputs
  - Creating TextFSM Templates
  - Retrieving Command Outputs with Netmiko.

DRY – Don't Repeat Yourself

- The Dry Principle states
  - *"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system."*
- Aimed at reducing repetition of information
- Less Code is Good
- Divide code and logic into smaller reusable units.



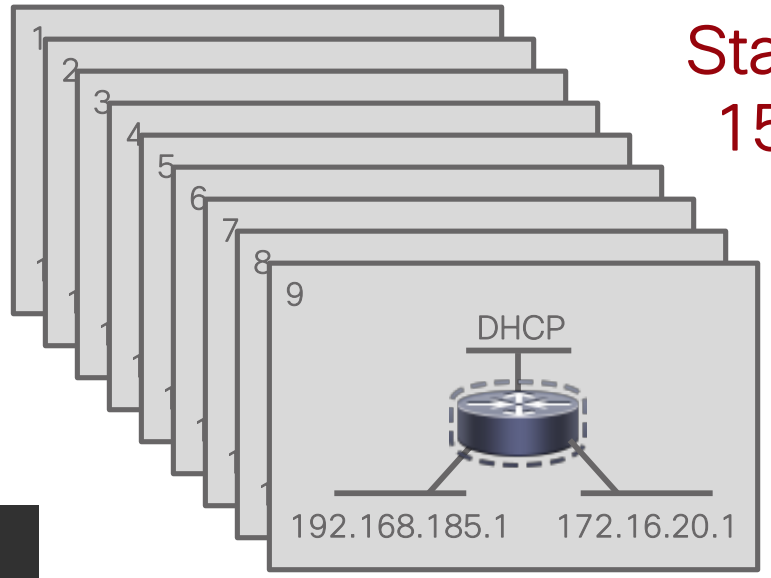
A LONG TIME AGO……

My First Code Was WET

Developer

How a Traditional Network Engineer Got Started with Network Programmability

Bryan Byrne
August 22, 2017 - 3 Comments



Start to Finish
15 Minutes!!

Interface Specific Script x2

```
#!/usr/bin/env python

import requests

url = "https://127.0.0.1:2125/restconf/data/ietf-interfaces:interfaces/interface=GigabitEthernet2"

payload = "{\"ietf-interfaces:interface\"" \
          ": {\"name\": \"GigabitEthernet2\",\"type\": \"iana-if-type:ethernetCsmacd\",\"enabled\": true,\"" \
          "ietf-ip:ipv4\": {\"address\": [{\"ip\": \"172.16.20.1\",\"netmask\": \"255.255.255.0\"}]}," \
          "\"ietf-ip:ipv6\": {}}}"
headers = {
    'Content-Type': "application/yang-data+json",
    'Accept': "application/yang-data+json",
    'Authorization': "Basic dmFncmFudDp2YWdyYW50",
    }

response = requests.request("PUT", url, data=payload, headers=headers, verify=False)

print(response.text)
```

Glued Together with Bash

```
#!/usr/bin/env bash

echo *Configuring Interface G2*
python g2.py
echo *Configuring Interface G3*
python g3.py
echo *Configuration Complete*
echo
```

So What Went Wrong?

"Humans are not good at managing complexity; they're good at finding creative solutions for problems of a specific scope."

-Chris Peters, 3 Key Software Principles

https://code.tutsplus.com

- What at first seemed a simple solution added significant complexity the first time I needed to modify my script.
- The code performed a specific task. Any other task required significant modifications.
- Required specific a development environment. (i.e. Widows doesn't have bash)

What is a Template?

In programming, a template is a generic class or other unit of source code that can be used as the basis for other units of code.

```
router ospf 100
 router-id <<replace with loopback0>>
 network <<gig1 ip>> 0.0.0.0 area 0
 network <<gig2 ip>> 0.0.0.0 area 100
```

The Foundation for Reusable Code

- Scripting Language
  - Ex: Python, Ansible, NSO
- Templates – Sets of re-usable configuration and operational command sets. Include logic for substitution of device specific parameters
  - Ex: Jinja2, TextFSM, Notepad ☺
- Structured Data – Any data that exists in a fixed field within a record or a file
  - Ex: Dictionaries, Lists, CSV, YAML

Using Jinja to Create CLI Templates

Jinja Templating

- Jinja2 is one of the most used template engines for Python.
- Extends capabilities through tools such as loops, conditionals and inheritance.
- Installed via pip
- http://jinja.pocoo.org

Jinja Templating – Loops and Conditionals

- Variables and/or logic are placed between delimiters
  - {% …. %} - Used for expressions or loops.
    - '-%}' – Removes additional white space.
  - {{ …. }} – Used for outputting the expression or variable.
- Loops
  - Opened with {% for 'x' in 'y' %} and typically fed in from some form of programmatic script.
  - Closed with {% endfor %}
- Conditionals
  - Can be nested in other operations
  - if, elif, else otpions (ex: {% if x == y %}
  - Closed with {% endif %}

```
{% for member in band_member -%}

        {% if member.instrument == "Singer" -%}
                {{ member.name }} Ramone is
                the{{ member.instrument }}
        {% else -%}
                {{ member.name }} Ramone plays
                the{{ member.instrument }}
        {% endif -%}

{% endfor -%}
```

Input:	Output:
Johnny, Guitar	Johnny Ramone plays the Guitar
Dee Dee, Bass	Dee Dee Ramone plays the Bass
Joey, Singer	Joey Ramone is the Singer
Tommy, Drums	Tommy Ramone plays the Drums

Jinja Examples

Example 1: Starting with the Basics

```
#!/usr/bin/env python

from jinja2 import Template

vlan_template = Template("""vlan {{id}}
 name VLAN_{{id}}""")

output = vlan_template.render(id=101)

print()
print(output)
print()
```

Define the Template

Render the template with variables

Process the output

Output:

```
(venv)$python ex1_hello_vlan.py

vlan 101
 name VLAN_101
```

jinja_example/ex1_hello_vlan.py

Example 2: Reusing Variables

```
#!/usr/bin/env python

from jinja2 import Template

vlan_var = 101
svi_ip = '192.168.1.1'
svi_mask = '255.255.255.0'

vlan_template = Template('vlan {{id}} \n'
                         ' name VLAN_{{id}}')

svi_template = Template('interface vlan {{id}} \n'
                        ' description This is the SVI for VLAN {{id}} \n'
                        ' ip address {{address}} {{mask}}')

vlan_output = vlan_template.render(id=vlan_var)

svi_output = svi_template.render(id=vlan_var, address=svi_ip, mask=svi_mask)

print()
print('!Output from vlan_template')
print(vlan_output)
print()
print('!Output from SVI'
      ' template')
print(svi_output)
print()
```

Variables substituted into the template

Define the Template

Render the template with variables

Process the output

jinja_example/ex2_vlan_svi_bad.py

Example 2: Reusing Variables

Output:

```
(venv)$python ex2_vlan_svi_bad.py


!Output from vlan_template
vlan 101
 name VLAN_101
!Output from SVI template
interface vlan 101
 description This is the SVI for VLAN 101
 ip address 192.168.1.1 255.255.255.0
```

Example 3: Reading In a Template

```
#!/usr/bin/env python

from jinja2 import Template

vlan_var = 101
svi_ip = '192.168.1.1'
svi_mask = '255.255.255.0'

with open("ex3_vlan_template.j2") as f:
    vlan_in_temp = Template(f.read())

with open("ex3_svi_template.j2") as f:
    svi_in_temp = Template(f.read())

vlan_output = vlan_in_temp.render(id=vlan_var)

svi_output = svi_in_temp.render(id=vlan_var, ip=svi_ip, mask=svi_mask)

print()
print('!Output from vlan_template')
print(vlan_output)
print()
print('!Output from SVI template')
print(svi_output)
print()
```

Read in the Template

Jinja_example/ex3_vlan_svi_good.py

Example 3: Reading In a Template

ex3_vlan_template.j2

```
vlan {{ id }}
 name VLAN_{{ id }}
```

ex3_svi_template.j2

```
interface vlan {{ id }}
 description This is the SVI for VLAN {{ id }}
 ip address {{ ip }} {{ mask }}
```

- Structure is similar to the embedded templates
- Notice that now we are calling external templates we need the proper variable formatting of {{ … }}
- This is a simple substitution of {{ … }} for the defined variable.
- Note: The file can be called anything. The .j2 extension is a personal preference for quickly identifying templates vs. code.

Jinja_example/ex3_vlan_template.j2 and jinja_example/ex3_vlan_template.j2

Example 3: Reading In a Template

```
#!/usr/bin/env python

from jinja2 import Template

vlan_var = 101
svi_ip = '192.168.1.1'
svi_mask = '255.255.255.0'

with open("ex3_vlan_template.j2") as f:
    vlan_in_temp = Template(f.read())

with open("ex3_svi_template.j2") as f:
    svi_in_temp = Template(f.read())

vlan_output = vlan_in_temp.render(id=vlan_var)

svi_output = svi_in_temp.render(id=vlan_var, ip=svi_ip, mask=svi_mask)

print()
print('!Output from vlan_template')
print(vlan_output)
print()
print('!Output from SVI template')
print(svi_output)
print()
```

Read in the Template

Render the template with variables

Process the output

jinja_example/ex3_vlan_svi_good.py

Example 3: Reusing Variables

Output:

```
(venv)$python ex3_vlan_svi_good.py
!Output from vlan_template
vlan 101
 name VLAN_101


!Output from SVI template
interface vlan 101
 description This is the SVI for VLAN 101
 ip address 192.168.1.1 255.255.255.0
```

jinja_example/ex3_vlan_svi_good.py

Example 4: Loops

```
#!/usr/bin/env python

from jinja2 import Template

vlans = [
    {'name:': 'VLAN_101',
     'vlan_var': 101,
     'ip_var': '192.168.1.1',
     'mask_var': '255.255.255.0'},
    {'name:': 'VLAN_201',
     'vlan_var': 201,
     'ip_var': '172.16.20.1',
     'mask_var': '255.255.0.0'},
    {'name:': 'VLAN_301',
     'vlan_var': 301,
     'ip_var': '10.0.0.0',
     'mask_var': '255.0.0.0'}
]

with open("ex4_svi_template_loop.j2") as f:
    config_in = Template(f.read())

config_out = config_in.render(vlans=vlans)

print ("!Generating Output for Multiple VLANs")
print(config_out)
```

jinja_example/ex4_svi_template_loop.py

Example 4: Loops

```
#!/usr/bin/env python


with open("ex4_svi_template_loop.j2") as f:
    config_in = Template(f.read())

config_out = config_in.render(vlans=vlans)

print ("!Generating Output for Multiple VLANs")
print(config_out)
```

jinja_example/ex4_svi_template_loop.py

Example 4: Loops

```
{% for vlan in vlans %}

# Generating Configuration for: {{ vlan.name }}

vlan {{ vlan.vlan_var }}
 name VLAN_{{ vlan.vlan_var }}
!
interface vlan {{ vlan.vlan_var }}
 description This is the SVI for VLAN {{ vlan.vlan_var }}
 ip address {{ vlan.ip_var }} {{ vlan.mask_var }}

{% endfor %}

# Config Complete
```

- In this template we will be substituting the key/value pairs from the list called ‘vlans’.
- {% for vlan in vlans %} – Iterate over each line in the list and apply the value from the matching key.
- Notice in this example I closed the operators with ’%}’. This will have an impact on the output.

jinja_example/ex4_svi_template_loop.j2

Example 4: Loops

```
#!/usr/bin/env python


with open("ex4_svi_template_loop.j2") as f:
    config_in = Template(f.read())

config_out = config_in.render(vlans=vlans)

print ("!Generating Output for Multiple VLANs")
print(config_out)
```

Read in the template

Render the template with the provided variables

Process the output

jinja_example/ex4_svi_template_loop.py

Example 4: Loops

Output (based on {% … %}):

```
(venv)$python ex4_svi_template_loop.py
!Generating Output for Multiple VLANs


# Generating Configuration for:

vlan 101
 name VLAN_101
!
interface vlan 101
 description This is the SVI for VLAN 101
 ip address 192.168.1.1 255.255.255.0



# Generating Configuration for:

vlan 201
 name VLAN_201
!
interface vlan 201
 description This is the SVI for VLAN 201
 ip address 172.16.20.1 255.255.0.0
```

- Operation lines in the template are rendered as blank lines unless it's closed with '-%}
- Blank lines in the template are rendered as blank lines in the output.

Output (based on {% … -%}:

```
(venv)$python ex4_svi_template_loop.py
!Generating Output for Multiple VLANs
# Generating Configuration for:
vlan 101
 name VLAN_101
!
interface vlan 101
 description This is the SVI for VLAN 101
 ip address 192.168.1.1 255.255.255.0
# Generating Configuration for:
vlan 201
 name VLAN_201
!
interface vlan 201
 description This is the SVI for VLAN 201
 ip address 172.16.20.1 255.255.0.0
```

Example 5: Conditionals

```
#!/usr/bin/env python

from jinja2 import Template

ports = [
    {'name': 'GigabitEthernet0/0/1',
     'mode': 'access',
     'vlan': 101,
     'state': 'enabled'},
    {'name': 'GigabitEthernet0/0/2',
     'mode': 'access',
     'vlan': 201,
     'state': 'shutdown'},
    {'name': 'GigabitEthernet0/0/3',
     'mode': 'access',
     'vlan': 301,
     'state': 'up'},
    {'name': 'GigabitEthernet0/0/4',
     'mode': 'routed',
     'vlan': ''},
    {'name': 'GigabitEthernet0/0/24',
     'mode': 'trunk',
     'allowed': '101,201,301'}
]

with open("ex5_port_template_conditional.j2") as f:
    config_in = Template(f.read())

config_out = config_in.render(ports=ports)

print(config_out)
```

```
{'name': 'GigabitEthernet0/0/1',
  'mode': 'access',
  'vlan': 101,
  'state': 'enabled'},
{'name': 'GigabitEthernet0/0/2',
  'mode': 'access',
  'vlan': 201,
  'state': 'shutdown'},
```

Jinja_example/ex5_port_template_conditional.py

Example 5: Conditionals

```
#!/usr/bin/env python


with open("ex5_port_template_conditional.j2") as f:
    config_in = Template(f.read())

config_out = config_in.render(ports=ports)

print ("!Generating Output for Multiple VLANs")
print(config_out)
```

Read in the template

jinja_example/ex5_port_template_conditional.py

Example 5: Conditionals

```
{% for port in ports -%}

# Generating Configuration for: {{ port.name }}
{% if port.mode == "access" -%}
interface {{ port.name }}
 switchport {{ port.mode }} vlan {{ port.vlan }}
 spanning-tree portfast
 {% if port.state == "shutdown" -%}
 description "Port Shutdown in Data Set"
 shutdown
 {% elif port.state == "enabled" -%}
 description "Configured by NetDevOps Live"
 no shutdown
 {% else -%}
 description "Interface Set as Shutdown Due to Invalid Input"
 shutdown
 {% endif -%}

{% elif port.mode == "trunk" -%}
interface {{ port.name }}
 switchport mode {{ port.mode }}
 spanning-tree portfast trunk
 switchport trunk allowed vlan {{ port.allowed }}

{% else -%}
interface {{ port.name }}
 description "Shut Down For Invalid Port Mode"
 shutdown
{% endif -%}

### Config Complete ###
{% endfor -%}
```

- Let's walk the logic – Configure the ports based on the details provided in the embedded list.
  - For access ports assign the VLAN and
    - If the port state is 'shutdown' issue a shutdown.
    - If the port state is 'enabled' issue a no shutdown.
    - If the port state is anything else shutdown the port
  - For trunk ports set the correct mode and configure the allowed vlans
  - For all other port types shutdown the interface

jinja_example/ex5_port_template_conditional.j2

Example 5: Conditionals

```
#!/usr/bin/env python


with open("ex5_port_template_conditional.j2") as f:
    config_in = Template(f.read())

config_out = config_in.render(ports=ports)

print ("!Generating Output for Multiple VLANs")
print(config_out)
```

Read in the template

Render the template with the provided variables

Process the output

jinja_example/ex5_port_template_conditional.py

Example 5: Conditionals – Full Output (Hidden)

Output:

```
# Generating Configuration for: GigabitEthernet0/0/1
interface GigabitEthernet0/0/1
 switchport access vlan 101
 spanning-tree portfast
 description "Configured by NetDevOps Live"
 no shutdown
 ### Config Complete ###
# Generating Configuration for: GigabitEthernet0/0/2
interface GigabitEthernet0/0/2
 switchport access vlan 201
 spanning-tree portfast
 description "Port Shutdown in Data Set"
 shutdown
 ### Config Complete ###
# Generating Configuration for: GigabitEthernet0/0/3
interface GigabitEthernet0/0/3
 switchport access vlan 301
 spanning-tree portfast
 description "Interface Set as Shutdown Due to Invalid Input"
access
 shutdown
 ### Config Complete ###
```

Output (continued):

```
# Generating Configuration for: GigabitEthernet0/0/4
interface GigabitEthernet0/0/4
 description "Shut Down For Invalid Port Mode"
 shutdown
### Config Complete ###
# Generating Configuration for: GigabitEthernet0/0/24
interface GigabitEthernet0/0/24
 switchport mode trunk
 spanning-tree portfast trunk
 switchport trunk allowed vlan 101,201,301

### Config Complete ###
```

jinja_example/ex5_port_template_conditional.py

Example 5: Conditionals – Let's Walk the Output

Input:

Input:

Input:

Input:

Input:

```
{'name': 'GigabitEthernet0/0/24',
 'mode': 'trunk',
 'allowed': '101,201,301'}
```

Output:

Output:

Output:

Output:

Output:

```
GigabitEthernet0/0/24
interface GigabitEthernet0/0/24
 switchport mode trunk
 spanning-tree portfast trunk
 switchport trunk allowed vlan 101,201,301
```

jinja_example/ex5_port_template_conditional.py

Example 5: Conditionals

```
#!/usr/bin/env python

from jinja2 import Template

ports = [
    {'name': 'GigabitEthernet0/0/1',
     'mode': 'access',
     'vlan': 101,
     'state': 'enabled'},
    {'name': 'GigabitEthernet0/0/2',
     'mode': 'access',
     'vlan': 201,
     'state': 'shutdown'},
    {'name': 'GigabitEthernet0/0/3',
     'mode': 'access',
     'vlan': 301,
     'state': 'up'},
    {'name': 'GigabitEthernet0/0/4',
     'mode': 'routed',
     'vlan': ''},
    {'name': 'GigabitEthernet0/0/24',
     'mode': 'trunk',
     'allowed': '101,201,301'}
]

with open("ex5_port_template_conditional.j2") as f:
    config_in = Template(f.read())

config_out = config_in.render(ports=ports)

print(config_out)
```

What about this? It looks pretty WET.

Jinja_example/ex5_port_template_conditional.py

YAML - "YAML Ain't Markup Language"

A human readable data structure that **applications use** to store, transfer, and read data.

```
---
ietf-interfaces:interface:
  name: GigabitEthernet2
  description: Wide Area Network
  enabled: true
  ietf-ip:ipv4:
    address:
    - ip: 172.16.0.2
      netmask: 255.255.255.0
```

Example 6: Generating Configurations from YAML

```
#!/usr/bin/env python

from jinja2 import Template
import yaml
from argparse import ArgumentParser
from pprint import pprint

parser = ArgumentParser("Specifying the YAML File")
parser.add_argument("-f", "--file",
                    help="Please Specify the YAML file.",
                    required=True)
args = parser.parse_args()
file_name = args.file

with open(file_name) as f:
    yaml_data = yaml.safe_load(f.read())

with open("ex6_yaml_data.j2") as f:
    config_in = Template(f.read())

for device in yaml_data["devices"]:
    config_out = config_in.render(interfaces=device["interfaces"],
                                  file=file_name)

print(config_out)
```

argparse allows the operator to pass in a variable. In this case it will be our device details in YAML format.

Example 1: Generating Configurations from YAML

```yaml
---

devices:
  - name: iosxe1
    mgmt_ip: 10.0.0.1
    mgmt_user: iosxe1_user
    mgmt_pass: iosxe1_pass
    interfaces:
      - name: GigabitEthernet2
        state: "enabled"
        ip: 192.168.1.1
        mask: 255.255.255.0
      - name: GigabitEthernet3
        state: "enabled"
        ip: 172.16.20.1
        mask: 255.255.0.0
      - name: GigabitEthernet4
        state: "shutdown"
        ip: 10.10.0.1
        mask: 255.255.255.0
```

jinja_example/ex6_yaml_data_1.yaml

```yaml
---

devices:
  - name: iosxe1
    mgmt_ip: 10.0.0.1
    mgmt_user: iosxe1_user
    mgmt_pass: iosxe1_pass
    interfaces:
      - name: GigabitEthernet2
        state: "shutdown"
        ip: 192.168.85.1
        mask: 255.255.255.0
      - name: GigabitEthernet3
        state: "enabled"
        ip: 172.20.100.1
        mask: 255.255.0.0
      - name: GigabitEthernet4
        state: "enabled"
        ip: 10.20.30.1
        mask: 255.255.255.0
```

jinja_example/ex6_yaml_data_2.yaml

Example 6: Generating Configurations from YAML

```
#!/usr/bin/env python

from jinja2 import Template
import yaml
from argparse import ArgumentParser
from pprint import pprint

parser = ArgumentParser("Specifying the YAML File")
parser.add_argument("-f", "--file",
                    help="Please Specify the YAML file.",
                    required=True)
args = parser.parse_args()
file_name = args.file

with open(file_name) as f:
    yaml_data = yaml.safe_load(f.read())

with open("ex6_yaml_data.j2") as f:
    config_in = Template(f.read())

for device in yaml_data["devices"]:
    config_out = config_in.render(interfaces=device["interfaces"],
                                  file=file_name)

print(config_out)
```

argparse allows the operator to pass in a variable. In this case it will be our device details in YAML format.

Open the YAML file and read in the contents.

Open the template, read it in and render the configuration

Process the output

Example 6: Generating Configurations from YAML

python ex1_yaml_data.py -f **ex1_yaml_data_1.yaml**

```
interface GigabitEthernet2
 description Generated with YAML file
         ex1_yaml_data_1.yaml
 ip address 192.168.1.1 255.255.255.0
 no shutdown

interface GigabitEthernet3
 description Generated with YAML file
         ex1_yaml_data_1.yaml
 ip address 172.16.20.1 255.255.0.0
 no shutdown

interface GigabitEthernet4
 description Generated with YAML file
         ex1_yaml_data_1.yaml
 ip address 10.10.0.1 255.255.255.0
 shutdown
```

python ex1_yaml_data.py -f **ex1_yaml_data_2.yaml**

```
interface GigabitEthernet2
 description Generated with YAML file
         ex1_yaml_data_2.yaml
 ip address 192.168.85.1 255.255.255.0
 shutdown

interface GigabitEthernet3
 description Generated with YAML file
         ex1_yaml_data_2.yaml
 ip address 172.20.100.1 255.255.0.0
 no shutdown

interface GigabitEthernet4
 description Generated with YAML file
         ex1_yaml_data_2.yaml
 ip address 10.20.30.1 255.255.255.0
 no shutdown
```


Note – The description line has word wrapped based on the text box.

THIS CODE IS DRY!!!!!!

```
#!/usr/bin/env python

from jinja2 import Template
import yaml
from argparse import ArgumentParser
from pprint import pprint

parser = ArgumentParser("Specifying the YAML File")
parser.add_argument("-f", "--file",
                    help="Please Specify the YAML file.",
                    required=True)
args = parser.parse_args()
file_name = args.file

with open(file_name) as f:
    yaml_data = yaml.safe_load(f.read())

with open("ex6_yaml_data.j2") as f:
    config_in = Template(f.read())

for device in yaml_data["devices"]:
    config_out = config_in.render(interfaces=device["interfaces"],
                                  file=file_name)

print(config_out)
```

In Summary

- Jinja is a templating language for Python
- Allows a simple structure for variable substitution
- Adds additional functionality through loops and conditionals

Creating Structured Data with TextFSM

The Challenge

- Based on the output of ‘show version’ generate a report that shows:
  - Hostname
  - Serial Number
  - IOS Version
  - Uptime

```
iosxe1#show version
Cisco IOS XE Software, Version 16.09.01
Cisco IOS Software [Fuji], Virtual XE Software (X86_64_LINUX_IOSD-UNIVERSALK9-M),
Version 16.9.1, RELEASE SOFTWARE (fc2)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2018 by Cisco Systems, Inc.
Compiled Tue 17-Jul-18 16:57 by mcpre

<< License Details Omitted>>

iosxe1 uptime is 1 day, 1 hour, 45 minutes
Uptime for this control processor is 1 day, 1 hour, 46 minutes
System returned to ROM by reload
System image file is "bootflash:packages.conf"
Last reload reason: reload



<< Crypto Statement Omitted >>

A summary of U.S. laws governing Cisco cryptographic products may be found at:
http://www.cisco.com/wwl/export/crypto/tool/stqrg.html

If you require further assistance please contact us by sending email to
export@cisco.com.

License Level: ax
License Type: Default. No valid license found.
Next reload license Level: ax

cisco CSR1000V (VXE) processor (revision VXE) with 2183060K/3075K bytes of memory.
Processor board ID 9XYR3U3MQGE
3 Gigabit Ethernet interfaces
32768K bytes of non-volatile configuration memory.
3984988K bytes of physical memory.
7774207K bytes of virtual hard disk at bootflash:.
0K bytes of WebUI ODM Files at webui:.

Configuration register is 0x2102
```

TextFSM

- Open source project created by Google:
- Specifically designed to work with CLI driven devices
- The template is a series of rules on how to parse the data

"TextFSM is a Python module that implements a template based state machine for parsing semi-formatted text"

https://github.com/google/textfsm/wiki/TextFSM

TextFSM – The Basics

- The engine takes two inputs:
  - Text input
    - Typically, but not limited to, command responses from CLI commands
  - Template file
    - Required for each uniquely structured text input. Each command would have a corresponding template.
    - Is a description of how the template should parse out the data
    - Parsed data is returned as a tabular representation
    - Rules are defined in regex

RegEX is HARD

It's dangerous to go alone! Take this:

https://pynet.twb-tech.com/blog/automation/netmiko.html*

*Additional resources at the end of the presentation.

TextFSM – The Template

- The template consists of two parts
  - The ‘Value’ definitions, which describe the columns of data to extract.
    - Formatted as ‘Value name regex’
  - One or more ‘State’ definitions, describing the various states of the engine while parsing data.
    - The first line is the state name followed by one or more rules
    - Rules are indented and start with a ‘^’
    - Formatted as ‘^regex -> action’
    - Actions are Next, Continue, NoRecord, Record, Clear, Clearall

```
Value person (\S+\s\S+)
Value instrument (Singer|Guitar|Bass|Drums)

Start
  ^${name}\s+\w+\s+\w+\s+${instrument} -> Record
```

Input:
Henry Rollins is the Singer
Greg Ginn plays the Guitar
Dez Cadena plays the Guitar
Chuck Dukowski plays the Bass
Robo Valverde plays the Drums

Output:
[['Henry Rollins', 'Singer'],
['Greg Ginn', 'Guitar'],
['Dez Cadena', 'Guitar'],
['Chuck Dukowski', 'Bass'],
['Robo Valverde', 'Drums']]

Breaking down a template

Text Input: Henry Rollins is the Singer

```
Value person (\S+\s\S+)
Value instrument (Singer|Guitar|Bass|Drums)

Start
  ^${person}\s+\w+\s+\w+\s+${instrument} -> Record
```

- Remember '\' starts a regex condition. Anything else will be matched exactly
- The regex key:
  - \s (Backslash, Lowercase s) – matches a white space
  - \S+ (Backslash, Uppercase S, Plus) – matches anything NOT matched by \s
  - \w+ (Backslash, Lowercase w, Plus) – one or more word character
- The first value
  - Name = person
  - Regex = Any non-white space, followed by a white space, followed by any non-white space.
- The second value
  - Name = instrument
  - Regex = must exactly match any of: Singer, Guitar, Bass Drums
- Record the matching values based on this rule
  - From the start of the line it expects
  - The first value (person)
  - Followed by white space, word, white space, word, white space
  - The second value (instrument)

Who Creates the Templates?

- You! Brush up on your regex.

https://regexr.com

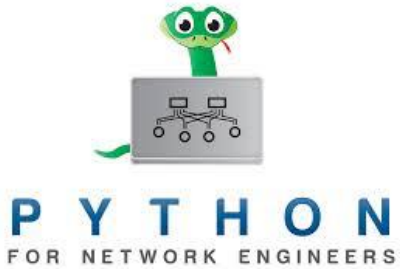
https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html

- The team at Network to Code has a large number of pre-defined templates available on github

https://github.com/networktocode/ntc-templates.git

- The NTC templates can be used directly with Kirk Byer's NetMiko.
  - NetMiko is a Python library for sending and receiving CLI commands with SSH.

https://pynet.twb-tech.com/



TextFSM Examples

Example 1: Starting with the Basics

```
#!/usr/bin/env python

import textfsm
from pprint import pprint

file = open ('ex1_ship_output.txt', 'r')
show_output = file.read()

template = open('ex1_ship_template.textfsm')
re_table = textfsm.TextFSM(template)
results = re_table.ParseText(show_output)

pprint(results)
```

Output:

```
[['GigabitEthernet1', '10.0.2.15', 'up', 'up'],
 ['GigabitEthernet2', '192.168.12.1', 'up', 'up'],
 ['GigabitEthernet3', '192.168.10.1', 'up', 'up']]
```

Read in the text to parse

Open the template and parse the text

Process the output

ex1_ship_output.txt

```
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       10.0.2.15       YES DHCP   up                    up
GigabitEthernet2       192.168.12.1    YES NVRAM  up                    up
GigabitEthernet3       192.168.10.1    YES NVRAM  up                    up
```

ex1_ship_template.textfsm

```
Value INTF (\S+)
Value IPADDR (\S+)
Value STATUS (up|down|administratively down)
Value PROTO (up|down)

Start
  ^${INTF}\s+${IPADDR}\s+\w+\s+\w+\s+${STATUS}\s+${PROTO} -> Record
```

textfsm_example/ex1_ship_output.py

Example 2: Using NetMiko to Generate Text Input

```
#!/usr/bin/env python

from netmiko import ConnectHandler
import textfsm
from pprint import pprint
from device_details import ios_xe1, ios_xe2

with open("ex2_ship_template.textfsm") as f:
    re_table = textfsm.TextFSM(f)

with ConnectHandler(device_type='cisco_ios',
                    ip=ios_xe1['address'],
                    username=ios_xe1['username'],
                    password=ios_xe1['password'],
                    port=ios_xe1['port']) as ch:

    ship_output = ch.send_command("show ip interface brief")

    results = re_table.ParseText(ship_output)

for interface in results:
    if interface[2]!="up":
        print("Warning: " + interface[0] + " is currently in the " + interface[2] + " state.")
        print("Someone should probably do something.")
        print()
    else:
        print("Good Job! " + interface[0] + " is up. Nothing to see here!")
        print()
```

Read in the template. It's the same as the previous example.

Generate the text to parse by sending a show command to the device.

Parse the show output against the template

Process the output

Example 2: Using NetMiko to Generate Text Input

Output:

```
(venv)$ python ex2_ship_netmiko.py
Good Job! GigabitEthernet1 is up. Nothing to see here!

Good Job! GigabitEthernet2 is up. Nothing to see here!

Warning: GigabitEthernet3 is currently in the administratively down state.
Someone should probably do something.
```

textfsm_example/ex2_ship_netmiko.py

So Let's Get Back to that Challenge

- Based on the output of 'show version' generate a report that shows:
  - Hostname
  - Serial Number
  - IOS Version
  - Uptime

```
#!/usr/bin/env python

from netmiko import ConnectHandler
import os
import yaml

device_outputs = []

with open("device_details.yaml") as f:
    config = yaml.full_load(f.read())

env = config["env_path"]

os.environ['NET_TEXTFSM'] = env

for device in config["devices"]:

    with ConnectHandler(device_type='cisco_ios',
                        ip=device["address"],
                        username=device["username"],
                        password=device["password"],
                        port=device["ssh_port"]) as ch:

        sh_ver_output = ch.send_command("show version", use_textfsm=True)

        for line in sh_ver_output:
            print()
            print("============================================================")
            print("For device: {} ".format(line["hostname"]))
            print("The serial number is: {}".format(line["serial"]))
            print("The IOS Version is: {}".format(line["version"]))
            print("The uptime is: {}".format(line["uptime"]))
            print("============================================================")
```

Textfsm_example/ex3_netmiko_sh_ver.py

This is a little hard to read. Let's break it up.

Example 3: Using NetMiko with TextFSM

- The NetMiko library has a built in function that will assign a TextFSM template based on the sent show command.
- The show command is mapped to a a template in an index file.
- Device details will be provided by YAML
- The location of the index file ('env') is specified with 'os.environ.'

```
#!/usr/bin/env python

from netmiko import ConnectHandler
import os
import yaml

with open("device_details.yaml") as f:
    config = yaml.full_load(f.read())

env = config["env_path"]

os.environ['NET_TEXTFSM'] = env
```

Textfsm_example/ex3_netmiko_sh_ver.py

Example 3: Using NetMiko with TextFSM

- For each device in the YAML file:
  - Connect to the device with SSH
  - Send the command “show version” and parse it against the appropriate template.
- Process the output and print the desired values.

```
for device in config["devices"]:

    with ConnectHandler(device_type='cisco_ios',
                        ip=device["address"],
                        username=device["username"],
                        password=device["password"],
                        port=device["ssh_port"]) as ch:

        sh_ver_output = ch.send_command("show version",
                                        use_textfsm=True)

        for line in sh_ver_output:
            print()
            print("=========================================")
            print("For device: {} ".format(line["hostname"]))
            print("The serial number is: {}".format(line["serial"]))
            print("The IOS Version is: {}".format(line["version"]))
            print("The uptime is: {}".format(line["uptime"]))
            print("=========================================")
```

Textfsm_example/ex3_netmiko_sh_ver.py

Example 3: Using NetMiko with TextFSM

Output

```
==========================================================
For device: iosxe1
The serial number is: ['9XYR3U3MQGE']
The IOS Version is: 16.9.1
The uptime is: 1 day, 18 hours, 31 minutes
==========================================================

==========================================================
For device: iosxe2
The serial number is: ['99L5P0LFIVU']
The IOS Version is: 16.9.1
The uptime is: 22 hours, 19 minutes
==========================================================
```

Textfsm_example/ex3_netmiko_sh_ver.py

THIS CODE IS DRY!!!!!!!!!

```
#!/usr/bin/env python

from netmiko import ConnectHandler
import os
import yaml

device_outputs = []

with open("device_details.yaml") as f:
    config = yaml.full_load(f.read())

env = config["env_path"]

os.environ['NET_TEXTFSM'] = env

for device in config["devices"]:

    with ConnectHandler(device_type='cisco_ios', ip=device["address"], username=device["username"],
                        password=device["password"], port=device["ssh_port"]) as ch:

        sh_ver_output = ch.send_command("show version",
                                        use_textfsm=True)

        for line in sh_ver_output:
            print()
            print("=========================================")
            print("For device: {} ".format(line["hostname"]))
            print("The serial number is: {}".format(line["serial"]))
            print("The IOS Version is: {}".format(line["version"]))
            print("The uptime is: {}".format(line["uptime"]))
            print("=========================================")
```

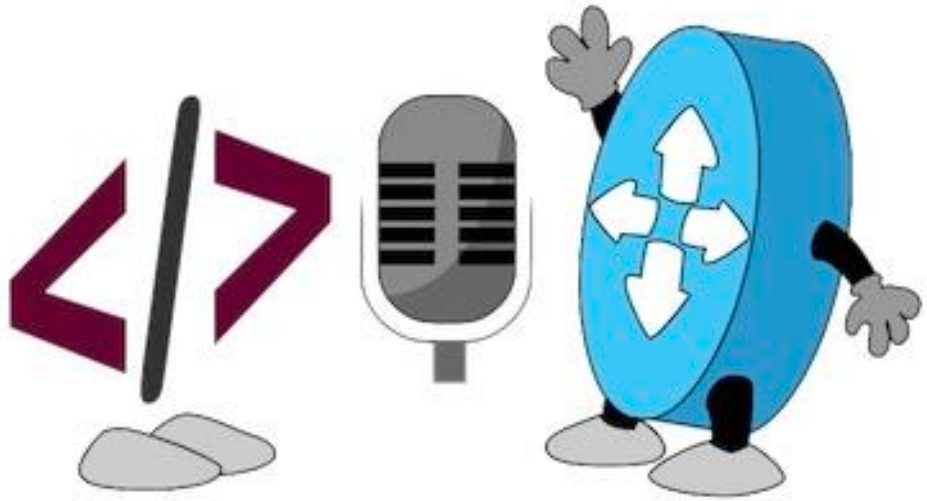
Summary

- TextFSM is an open source project created by Google to parse semi-structured data (CLI output)
- Templates are defined using a series of rules, written with regex, to process text.
- The result is structured data.

Wrap Up!

What did we talk about?

- What is the DRY Principal?
  - The Foundations for Reusable Code
- Using Jinja to Create CLI Templates
  - Understanding the Jinja Templating Language
  - Using YAML to Provide Structured Input for Variables
- Using TextFSM to put Structure into Show Outputs
  - Creating TextFSM Templates
  - Retrieving Command Outputs with Netmiko



Webinar Resource List

- Docs and Links
  - https://developer.cisco.com/python
  - http://jinja.pocoo.org – Jinja2
  - https://github.com/google/textfsm/wiki/TextFSM – TextFSM
  - https://github.com/networktocode/ntc-templates – Network to Code TextFSM Templates
  - https://pynet.twb-tech.com/ – Netmiko
- RegEx Tools
  - https://regexr.com
  - https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html

Webinar Resource List

- Learning Labs
  - Laptop Setup http://cs.co/lab-dev-setup
  - Coding Fundamentals http://cs.co/lab-coding-fundamentals
- DevNet Sandboxes
  - IOS Always On http://cs.co/sbx-iosxe
- Code Samples
  - http://cs.co/jinfsm-guide

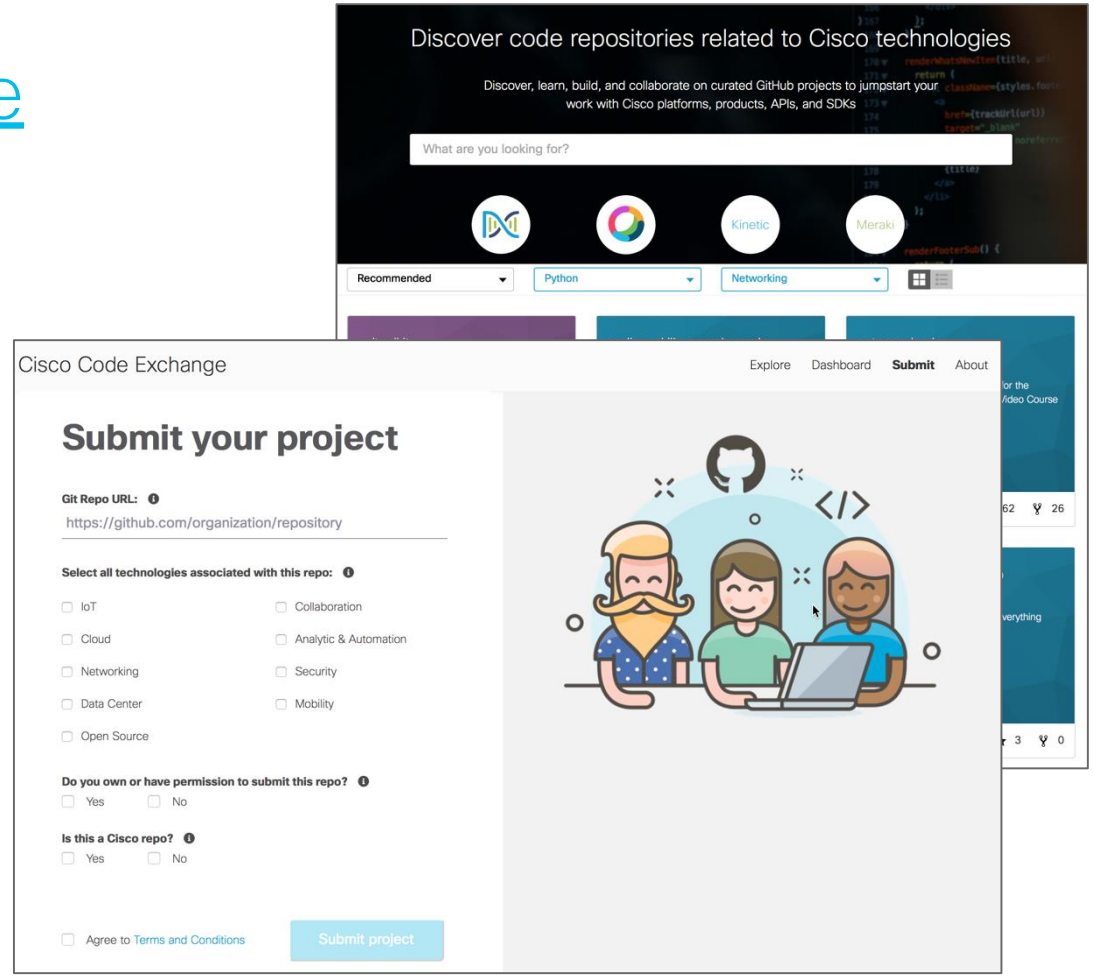
NetDevOps Live! Code Exchange Challenge

developer.cisco.com/codeexchange

Leverage one or more of the suggestions shown in an active network automation project of yours and submit to Code Exchange!

Challenge: Take a common configuration task and create a reusable CLI template.

Bonus Challenge: Use TextFSM to validate the change!



Looking for more about NetDevOps?

- NetDevOps on DevNet
  developer.cisco.com/netdevops
- NetDevOps Live!
  developer.cisco.com/netdevops/live
- NetDevOps Blogs
  blogs.cisco.com/tag/netdevops
- Network Programmability Basics Video Course
  developer.cisco.com/video/net-prog-basics/

Got more questions? Stay in touch!

Bryan Byrne

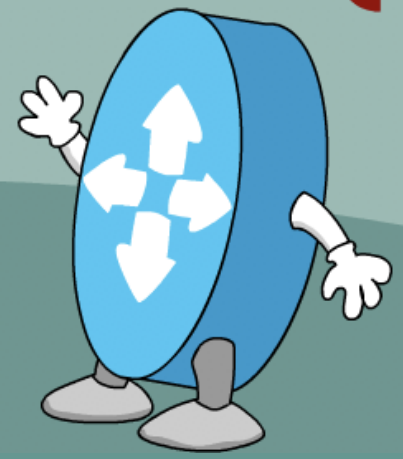
brybyrne@cisco.com
@bryan25607
http://github.com/brybyrne

CISCO DEVNET
LEARN CODE INSPIRE CONNECT

developer.cisco.com

@CiscoDevNet
facebook.com/ciscodevnet/
http://github.com/CiscoDevNet

NETDEVOPS {LIVE!}



cisco DEVNET

https://developer.cisco.com/netdevops/live

@netdevopslive