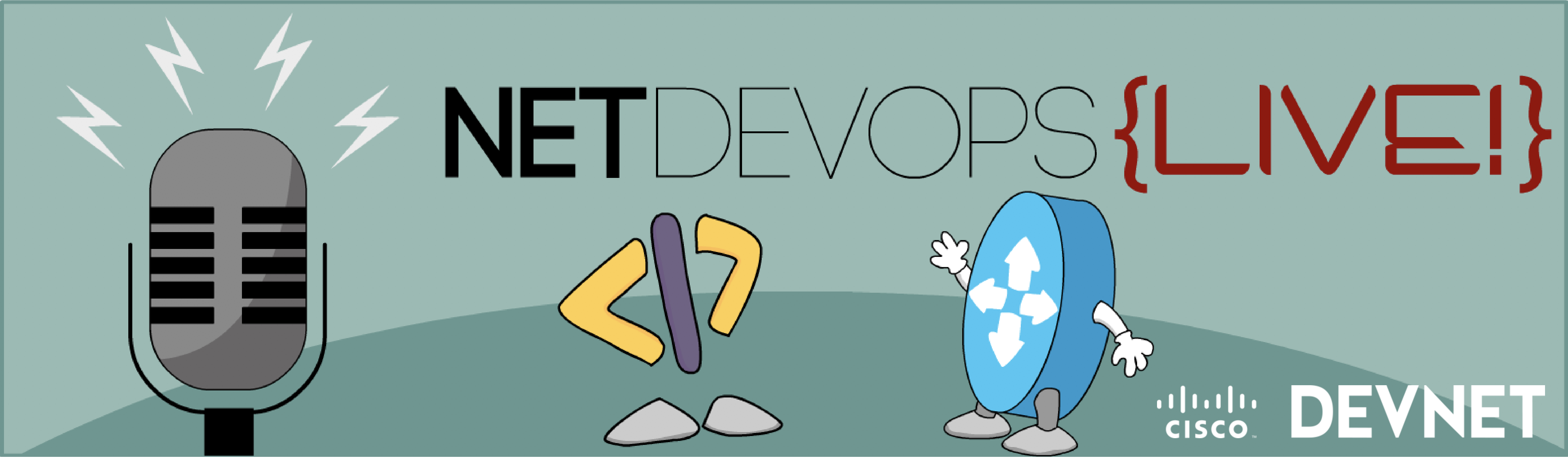


Useful Python Libraries for Network Engineers

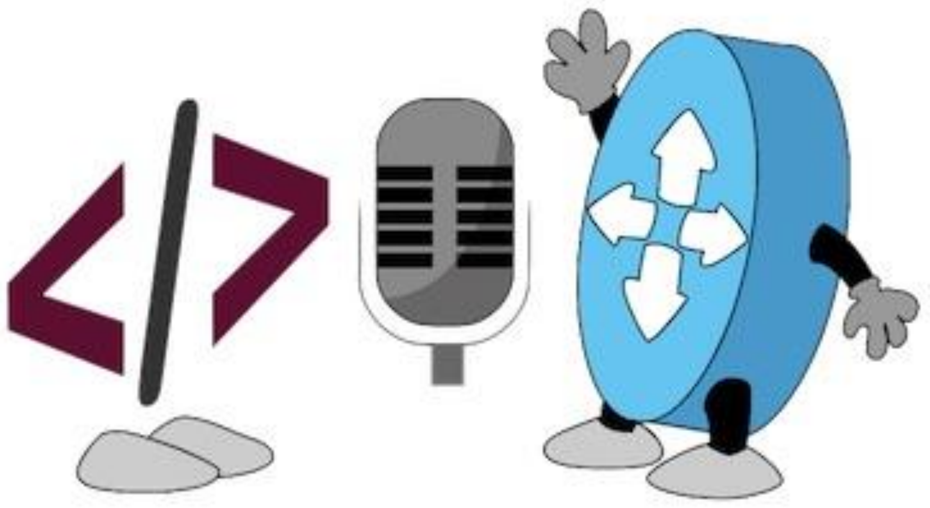
Hank Preston, ccie 38336 R/S
Developer Advocate, DevNet
Twitter: @hfpreston

Season 1, Talk 1

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

What are we going to talk about?

- Libraries to Work with Data
- API Libraries
- Configuration Management Tools and Libraries
- Some Other Cool Python Stuff



Libraries to Work with Data

Manipulating Data of All Formats

- XML - xmltodict
  - `pip install xmltodict`
    `import xmltodict`
- JSON
  - import json
- YAML - PyYAML
  - `pip install PyYAML`
    `import yaml`
- CSV
  - `import csv`
- YANG - pyang
  - `import pyang`

Treat XML like Python Dictionaries with xmltodict

- Easily work with XML data
- Convert from XML -> Dict* and back
  - xmltodict.parse(**xml_data**)
  - xmltodict.unparse(**dict**)
- Python includes a native Markup (html/xml) interfaces as well
  - More powerful, but also more complex

** Technically to an OrderedDict*

```python
# Import the xmltodict library
import xmltodict

# Open the sample xml file and read it into variable
with open("xml_example.xml") as f:
    xml_example = f.read()

# Print the raw XML data
print(xml_example)

# Parse the XML into a Python dictionary
xml_dict = xmltodict.parse(xml_example)

# Save the interface name into a variable using XML nodes as
keys
int_name = xml_dict["interface"]["name"]

# Print the interface name
print(int_name)

# Change the IP address of the interface
xml_dict["interface"]["ipv4"]["address"]["ip"] = "192.168.0.2"

# Revert to the XML string version of the dictionary
print(xmltodict.unparse(xml_dict))
```

data_manipulation/xml/xml_example.py

https://pypi.python.org/pypi/xmltodict

To JSON and back again with json

- JSON and Python go together like peanut butter and jelly
  - `json.loads(`**`json_data`**`)`
  - `json.dumps(`**`object`**`)`
- JSON Objects convert to Dictionaries
- JSON Arrays convert to Lists

```
# Import the jsontodict library
import json

# Open the sample json file and read it into variable
with open("json_example.json") as f:
    json_example = f.read()

# Print the raw json data
print(json_example)

# Parse the json into a Python dictionary
json_dict = json.loads(json_example)

# Save the interface name into a variable
int_name = json_dict["interface"]["name"]

# Print the interface name
print(int_name)

# Change the IP address of the interface
json_dict["interface"]["ipv4"]["address"][0]["ip"] = \
   "192.168.0.2"

# Revert to the json string version of the dictionary
print(json.dumps(json_dict))
```

https://docs.python.org/3/library/json.html

data_manipulation/json/json_example.py

YAML? Yep, Python Can Do That Too!

- Easily convert a YAML file to a Python Object
  - `yaml.load(`**`yaml_data`**`)`
  - `yaml.dump(`**`object`**`)`
- YAML Objects become Dictionaries
- YAML Lists become Lists

```python
# Import the yamltodict library
import yaml

# Open the sample yaml file and read it into variable
with open("yaml_example.yaml") as f:
    yaml_example = f.read()

# Print the raw yaml data
print(yaml_example)

# Parse the yaml into a Python dictionary
yaml_dict = yaml.load(yaml_example)

# Save the interface name into a variable
int_name = yaml_dict["interface"]["name"]

# Print the interface name
print(int_name)

# Change the IP address of the interface
yaml_dict["interface"]["ipv4"]["address"][0]["ip"] = \
   "192.168.0.2"

# Revert to the yaml string version of the dictionary
print(yaml.dump(yaml_dict, default_flow_style=False))
```

data_manipulation/yaml/yaml_example.py

https://pypi.python.org/pypi/PyYAML/3.12

Import Spreadsheets and Data with csv

- Treat CSV data as lists
  `csv.reader(`**`file_object`**`)`
- Efficiently processes large files without memory issues
- Options for header rows and different formats

```
# Import the csv library
import csv

# Open the sample csv file and print it to screen
with open("csv_example.csv") as f:
    print(f.read())

# Open the sample csv file, and create a csv.reader
object
with open("csv_example.csv") as f:
    csv_python = csv.reader(f)

    # Loop over each row in csv and leverage the data
    # in code
    for row in csv_python:
        print("{device} is in {location} " \
              "and has IP {ip}.".format(
                  device = row[0],
                  location = row[2],
                  ip = row[1]
                  )
             )
```

data_manipulation/csv/csv_example.py

https://docs.python.org/3/library/csv.html

YANG Data Modeling Language - IETF Standard

- Module that is a self-contained top-level hierarchy of nodes
- Uses containers to group related nodes
- Lists to identify nodes that are stored in sequence
- Each individual attribute of a node is represented by a leaf
- Every leaf must have an associated type

```
module ietf-interfaces {
  import ietf-yang-types {
    prefix yang;
  }
  container interfaces {
    list interface {
      key "name";
      leaf name {
        type string;
      }
      leaf enabled {
        type boolean;
        default "true";
    }
}
```

Example edited for simplicity and brevity

Investigate YANG Models with pyang

- Working in native YANG can be challenging
- pyang is a Python library for validating and working with YANG files
- Excellent for network developers working with NETCONF/RESTCONF/gRPC
- Quickly understand the key operational view of a model

```
echo "Print the YANG module in a simple text tree"
pyang -f tree ietf-interfaces.yang

echo "Print only part of the tree"
pyang -f tree --tree-path=/interfaces/interface \
  ietf-interfaces.yang

echo "Print an example XML skeleton (NETCONF)"
pyang -f sample-xml-skeleton ietf-interfaces.yang

echo "Create an HTTP/JS view of the YANG Model"
pyang -f jstree -o ietf-interfaces.html \
  ietf-interfaces.yang
open ietf-interfaces.html

echo 'Control the "nested depth" in trees'
pyang -f tree --tree-depth=2 ietf-ip.yang

echo "Include deviation models in the processing"
pyang -f tree \
  --deviation-module=cisco-xe-ietf-ip-deviation.yang \
  ietf-ip.yang
```

data_manipulation/yang/pyang-examples.sh

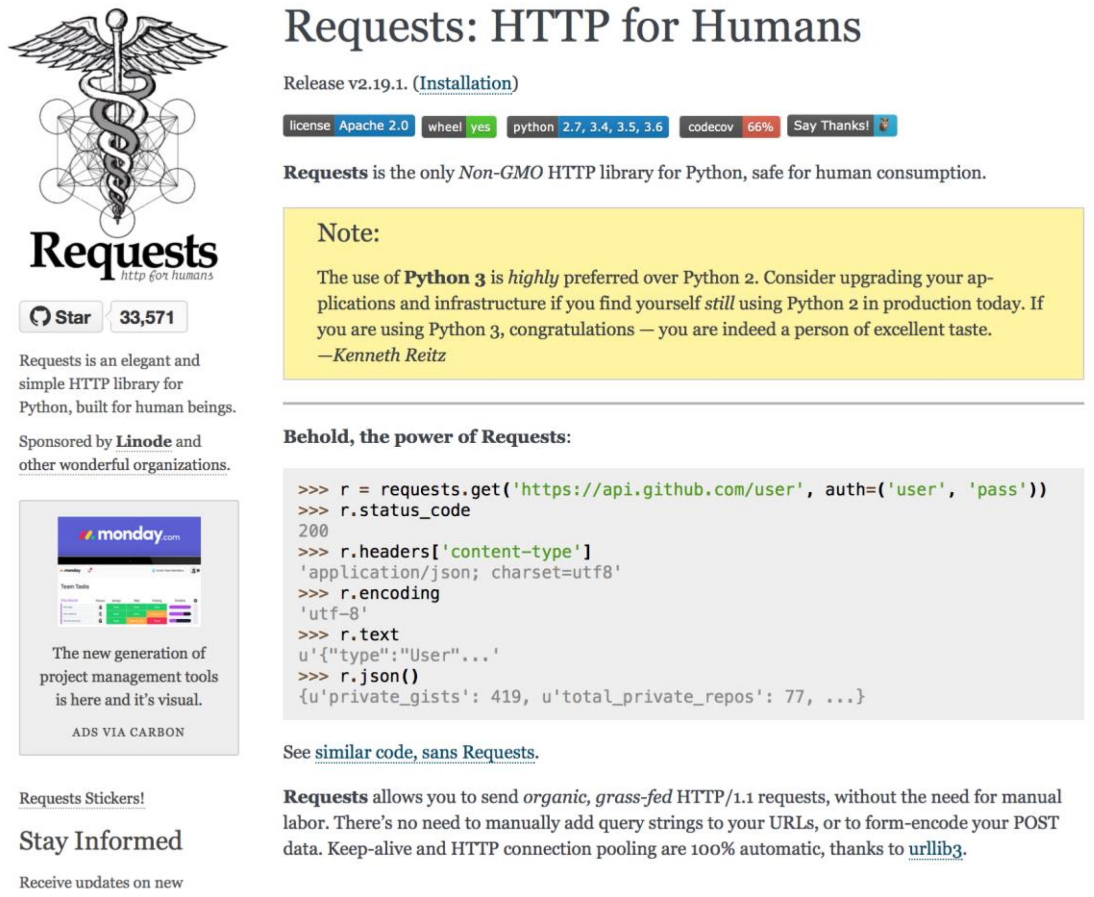
API Libraries

Access Different APIs Easily

- REST APIs – requests
  - `pip install requests`
    `import requests`
- NETCONF – ncclient
  - `pip install ncclient`
    `import ncclient`
- Network CLI – netmiko
  - `pip install netmiko`
    `import netmiko`
- SNMP – PySNMP
  - `pip install pysnmp`
    `import pysnmp`

Make HTTP Calls with Ease using “requests”

- Full HTTP Client
- Simplifies authentication, headers, and response tracking
- Great for REST API calls, or any HTTP request
- Network uses include RESTCONF, native REST APIs, JSON-RPC



Requests
http for humans

Star 33,571

Requests is an elegant and simple HTTP library for Python, built for human beings.

Sponsored by **Linode** and other wonderful organizations.

The new generation of project management tools is here and it's visual.

ADS VIA CARBON

Requests Stickers!

Stay Informed

Receive updates on new

Requests: HTTP for Humans

Release v2.19.1. (Installation)

license Apache 2.0 | wheel yes | python 2.7, 3.4, 3.5, 3.6 | codecov 66% | Say Thanks!

Requests is the only *Non-GMO* HTTP library for Python, safe for human consumption.

Note:

The use of **Python 3** is *highly* preferred over Python 2. Consider upgrading your applications and infrastructure if you find yourself *still* using Python 2 in production today. If you are using Python 3, congratulations — you are indeed a person of excellent taste.
—*Kenneth Reitz*

Behold, the power of Requests:

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
u'{"type":"User"...'
>>> r.json()
{u'private_gists': 419, u'total_private_repos': 77, ...}
```

See similar code, sans Requests.

Requests allows you to send *organic, grass-fed* HTTP/1.1 requests, without the need for manual labor. There's no need to manually add query strings to your URLs, or to form-encode your POST data. Keep-alive and HTTP connection pooling are 100% automatic, thanks to urllib3.

http://docs.python-requests.org

Example: Retrieving Configuration Details with RESTCONF

RESTCONF: Basic Request for Device Data 1/2

```
# Import libraries
import requests, urllib3
import sys

# Add parent directory to path to allow importing common vars
sys.path.append("..") # noqa
from device_info import ios_xe1 as device # noqa

# Disable Self-Signed Cert warning for demo
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# Setup base variable for request
restconf_headers = {"Accept": "application/yang-data+json"}
restconf_base = "https://{ip}:{port}/restconf/data"
interface_url = restconf_base + "/ietf-interfaces:interfaces/interface={int_name}"
```

Code edited for display on slide

device_apis/rest/restconf_example1.py

RESTCONF: Basic Request for Device Data 2/2

```
# Create URL and send RESTCONF request to core1 for GigE2 Config
url = interface_url.format(ip = device["address"], port = device["restconf_port"],
                           int_name = "GigabitEthernet2"
                          )
r = requests.get(url,
        headers = restconf_headers,
        auth=(device["username"], device["password"]),
        verify=False)

# Print returned data
print(r.text)

# Process JSON data into Python Dictionary and use
interface = r.json()["ietf-interfaces:interface"]
print("The interface {name} has ip address {ip}/{mask}".format(
      name = interface["name"],
      ip = interface["ietf-ip:ipv4"]["address"][0]["ip"],
      mask = interface["ietf-ip:ipv4"]["address"][0]["netmask"],
      )
)
```

Code edited for display on slide

device_apis/rest/restconf_example1.py

Example: Updating Configuration with RESTCONF

RESTCONF: Creating a New Loopback 1/2

```
# Setup base variable for request
restconf_headers["Content-Type"] = "application/yang-data+json"
# New Loopback Details
loopback = {"name": "Loopback101",
            "description": "Demo interface by RESTCONF",
            "ip": "192.168.101.1",
            "netmask": "255.255.255.0"}
# Setup data body to create new loopback interface
data = {
    "ietf-interfaces:interface": {
        "name": loopback["name"],
        "description": loopback["description"],
        "type": "iana-if-type:softwareLoopback",
        "enabled": True,
        "ietf-ip:ipv4": {
            "address": [
                {"ip": loopback["ip"],
                 "netmask": loopback["netmask"]}
            ] } } }
```

Only showing significant code changes

device_apis/rest/restconf_example2.py

RESTCONF: Creating a New Loopback 2/2

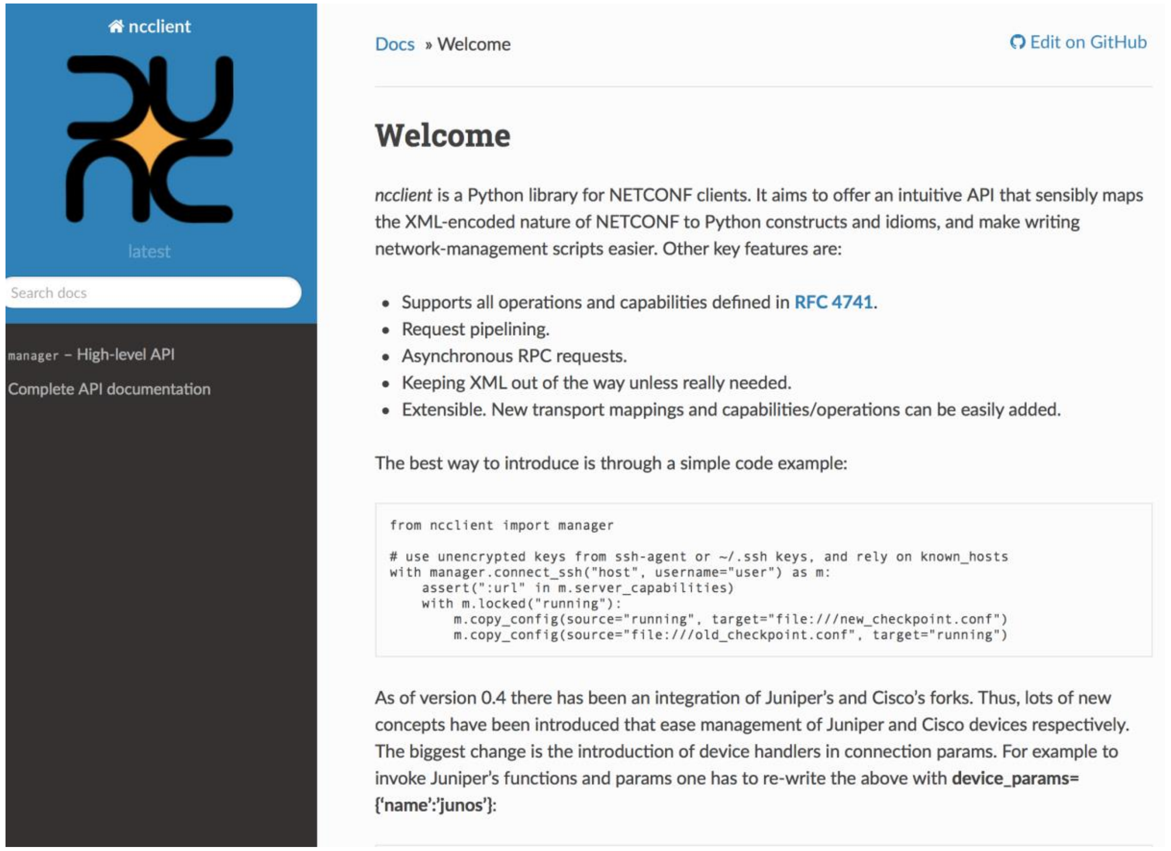
```
# Create URL and send RESTCONF request to core1 for GigE2 Config
url = interface_url.format(ip = core1_ip, int_name = loopback["name"])
r = requests.put(url,
        headers = restconf_headers,
        auth=(username, password),
        json = data,
        verify=False)


# Print returned data
print("Request Status Code: {}".format(r.status_code))
```

Only showing significant code changes

device_apis/rest/restconf_example2.py

Easily Interface with NETCONF and ncclient

- Full NETCONF Manager (ie client) implementation in Python
  - See later presentation on NETCONF details
- Handles all details including authentication, RPC, and operations
- Deals in raw XML



Welcome

ncclient is a Python library for NETCONF clients. It aims to offer an intuitive API that sensibly maps the XML-encoded nature of NETCONF to Python constructs and idioms, and make writing network-management scripts easier. Other key features are:

- Supports all operations and capabilities defined in RFC 4741.
- Request pipelining.
- Asynchronous RPC requests.
- Keeping XML out of the way unless really needed.
- Extensible. New transport mappings and capabilities/operations can be easily added.

The best way to introduce is through a simple code example:

```
from ncclient import manager

# use unencrypted keys from ssh-agent or ~/.ssh keys, and rely on known_hosts
with manager.connect_ssh("host", username="user") as m:
    assert(":url" in m.server_capabilities)
    with m.locked("running"):
        m.copy_config(source="running", target="file:///new_checkpoint.conf")
        m.copy_config(source="file:///old_checkpoint.conf", target="running")
```

As of version 0.4 there has been an integration of Juniper's and Cisco's forks. Thus, lots of new concepts have been introduced that ease management of Juniper and Cisco devices respectively. The biggest change is the introduction of device handlers in connection params. For example to invoke Juniper's functions and params one has to re-write the above with **device_params= {'name':'junos'}**:

https://ncclient.readthedocs.io

Example: Retrieving Configuration Details with NETCONF

NETCONF: Basic Request for Device Data 1/2

```
# Import libraries
from ncclient import manager
import xmltodict

# Create filter template for an interface
interface_filter = """
<filter>
  <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
    <interface>
      <name>{int_name}</name>
    </interface>
  </interfaces>
</filter>
"""
```

Code edited for display on slide

device_apis/netconf/netconf_example1.py

NETCONF: Basic Request for Device Data 2/2

```python
# Open NETCONF connection to device
with manager.connect(host=core1_ip, username=username, password=password,
                     hostkey_verify=False) as m:

    # Create desired NETCONF filter and <get-config>
    filter = interface_filter.format(int_name = "GigabitEthernet2")
    r = m.get_config("running", filter)


    # Process the XML data into Python Dictionary and use
    interface = xmltodict.parse(r.xml)
    interface = interface["rpc-reply"]["data"]["interfaces"]["interface"]

    print("The interface {name} has ip address {ip}/{mask}".format(
            name = interface["name"]["#text"],
            ip = interface["ipv4"]["address"]["ip"],
            mask = interface["ipv4"]["address"]["netmask"],
            )
        )
```

Code edited for display on slide

device_apis/netconf/netconf_example1.py

Example: Updating Configuration with NETCONF

NETCONF: Creating a New Loopback 1/2

```
# Create config template for an interface
config_data = """<config>
  <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      <interface>
        <name>{int_name}</name>
        <description>{description}</description>
        <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
          ianaift:softwareLoopback
        </type>
        <enabled>true</enabled>
        <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
          <address>
            <ip>{ip}</ip>
            <netmask>{netmask}</netmask>
          </address>
        </ipv4>
      </interface>
  </interfaces>
</config>
"""
```

Only showing significant code changes

device_apis/netconf/netconf_example2.py

NETCONF: Creating a New Loopback 2/2

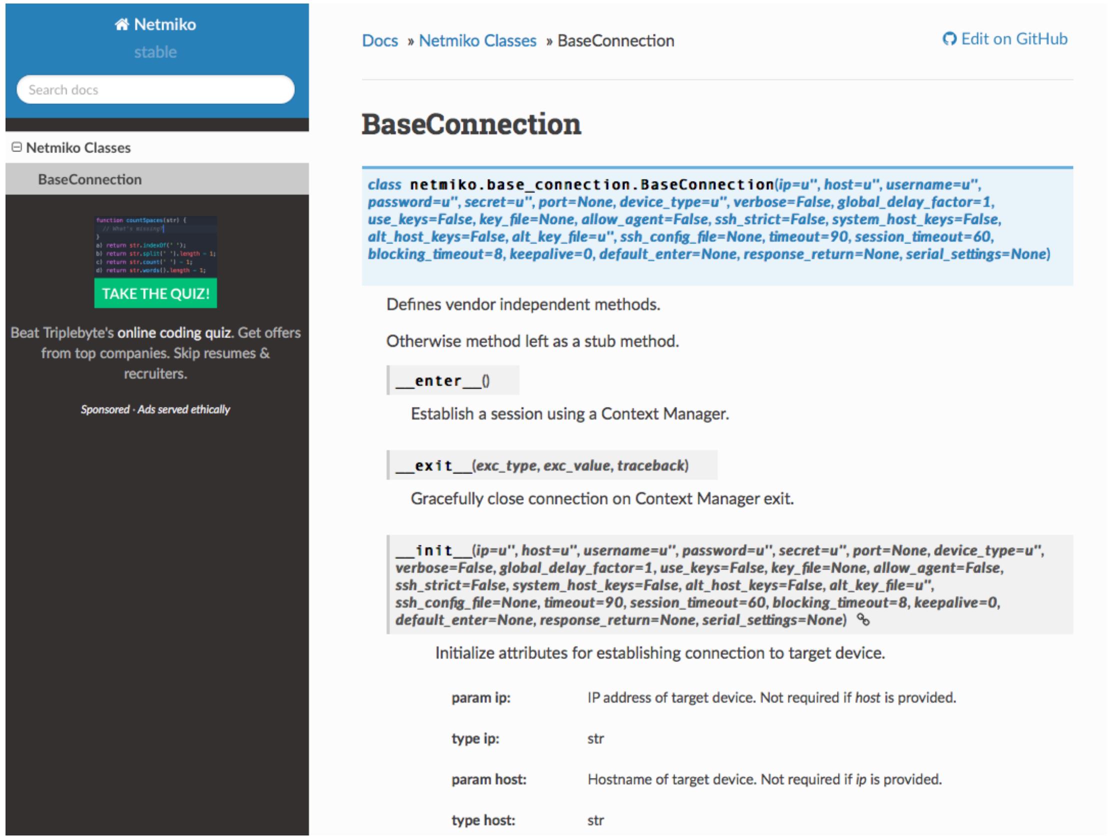
```
# New Loopback Details
loopback = {"int_name": "Loopback102",
            "description": "Demo interface by NETCONF",
            "ip": "192.168.102.1",
            "netmask": "255.255.255.0"}

# Open NETCONF connection to device
with manager.connect(host=core1_ip,
                     username=username,
                     password=password,
                     hostkey_verify=False) as m:

    # Create desired NETCONF config payload and <edit-config>
    config = config_data.format(**loopback)
    r = m.edit_config(target = "running", config = config)

    # Print OK status
    print("NETCONF RPC OK: {}".format(r.ok))
```

Only showing significant code changes

device_apis/netconf/netconf_example2.py

For When CLI is the Only Option – netmiko

- If no other API is available…
- Builds on paramiko library for SSH connectivity
- Support for a range of vendors network devices and operating systems
- Send and receive clear text
  - Post processing of data will be key



Netmiko
stable
Search docs
Netmiko Classes
BaseConnection

```
function countSpaces(str) {
  // What's missing?
}
a) return str.indexOf(' ');
b) return str.split(' ').length - 1;
c) return str.count(' ') - 1;
d) return str.words().length - 1;
```

TAKE THE QUIZ!

Beat Triplebyte's **online coding quiz**. Get offers from top companies. Skip resumes & recruiters.

Sponsored · Ads served ethically

Docs » Netmiko Classes » BaseConnection   Edit on GitHub

BaseConnection

```
class netmiko.base_connection.BaseConnection(ip=u'', host=u'', username=u'',
password=u'', secret=u'', port=None, device_type=u'', verbose=False, global_delay_factor=1,
use_keys=False, key_file=None, allow_agent=False, ssh_strict=False, system_host_keys=False,
alt_host_keys=False, alt_key_file=u'', ssh_config_file=None, timeout=90, session_timeout=60,
blocking_timeout=8, keepalive=0, default_enter=None, response_return=None, serial_settings=None)
```

Defines vendor independent methods.

Otherwise method left as a stub method.

```
__enter__()
```

Establish a session using a Context Manager.

```
__exit__(exc_type, exc_value, traceback)
```

Gracefully close connection on Context Manager exit.

```
__init__(ip=u'', host=u'', username=u'', password=u'', secret=u'', port=None, device_type=u'',
verbose=False, global_delay_factor=1, use_keys=False, key_file=None, allow_agent=False,
ssh_strict=False, system_host_keys=False, alt_host_keys=False, alt_key_file=u'',
ssh_config_file=None, timeout=90, session_timeout=60, blocking_timeout=8, keepalive=0,
default_enter=None, response_return=None, serial_settings=None)
```

Initialize attributes for establishing connection to target device.

param ip:	IP address of target device. Not required if *host* is provided.
type ip:	str
param host:	Hostname of target device. Not required if *ip* is provided.
type host:	str

https://github.com/ktbyers/netmiko

Example: Retrieving Configuration Details with CLI

CLI: Basic Request for Device Data 1/3

```python
# Import libraries
from netmiko import ConnectHandler
import re
import sys

# Add parent directory to path to allow importing common vars
sys.path.append("..") # noqa
from device_info import ios_xe1 as device # noqa

# Set device_type for netmiko
device["device_type"] = "cisco_ios"

# Create a CLI command template
show_interface_config_temp = "show running-config interface {}"
```

Code edited for display on slide

device_apis/cli/netmiko_example1.py

CLI: Basic Request for Device Data 2/3

```python
# Open CLI connection to device
with ConnectHandler(ip = device["address"],
                    port = device["ssh_port"],
                    username = device["username"],
                    password = device["password"],
                    device_type = device["device_type"]) as ch:


    # Create desired CLI command and send to device
    command = show_interface_config_temp.format("GigabitEthernet2")
    interface = ch.send_command(command)

    # Print the raw command output to the screen
    print(interface)
```

Code edited for display on slide

device_apis/cli/netmiko_example1.py

CLI: Basic Request for Device Data 3/3

```python
# Use regular expressions to parse the output for desired data
name = re.search(r'interface (.*)', interface).group(1)
description = re.search(r'description (.*)', interface).group(1)
ip_info = re.search(r'ip address (.*) (.*)', interface)
ip = ip_info.group(1)
netmask = ip_info.group(2)

# Print the info to the screen
print("The interface {name} has ip address {ip}/{mask}".format(
        name = name,
        ip = ip,
        mask = netmask,
        )
    )
```

Code edited for display on slide

device_apis/cli/netmiko_example1.py

Example: Updating Configuration with CLI

CLI: Creating a New Loopback

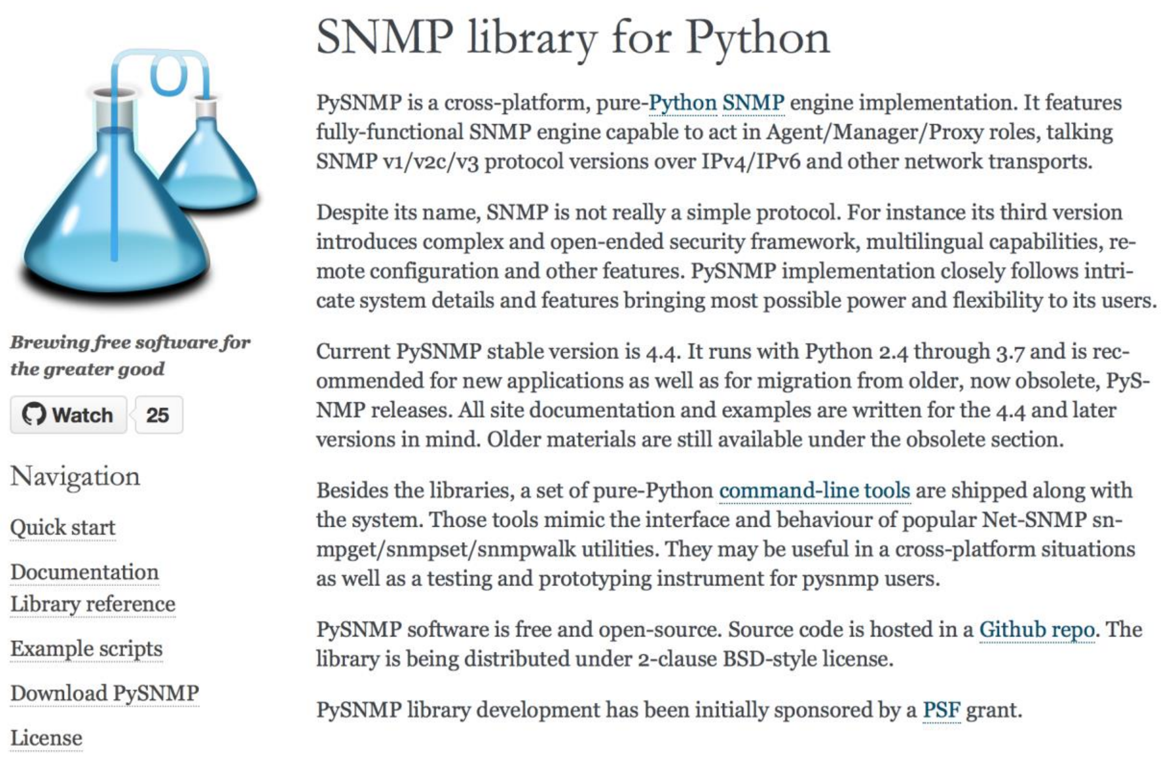
```python
# New Loopback Details
loopback = {"int_name": "Loopback103",
            "description": "Demo interface by CLI and netmiko",
            "ip": "192.168.103.1",
            "netmask": "255.255.255.0"}
# Create a CLI configuration
interface_config = [
    "interface {}".format(loopback["int_name"]),
    "description {}".format(loopback["description"]),
    "ip address {} {}".format(loopback["ip"], loopback["netmask"]),
    "no shut"]

# Open CLI connection to device
with ConnectHandler(ip=core1["ip"],
                    username=username,
                    password=password,
                    device_type=core1["device_type"]) as ch:

    # Send configuration to device
    output = ch.send_config_set(interface_config)
```

Only showing significant code changes

device_apis/cli/netmiko_example2.py

SNMP, a classic network interface with PySNMP

- Support for both GET and TRAP communications
- Can be a bit complex to write and leverage
  - Examples are available
- Data returned in custom objects



Brewing free software for the greater good

Watch 25

Navigation

Quick start

Documentation
Library reference

Example scripts

Download PySNMP

License

SNMP library for Python

PySNMP is a cross-platform, pure-Python SNMP engine implementation. It features fully-functional SNMP engine capable to act in Agent/Manager/Proxy roles, talking SNMP v1/v2c/v3 protocol versions over IPv4/IPv6 and other network transports.

Despite its name, SNMP is not really a simple protocol. For instance its third version introduces complex and open-ended security framework, multilingual capabilities, remote configuration and other features. PySNMP implementation closely follows intricate system details and features bringing most possible power and flexibility to its users.

Current PySNMP stable version is 4.4. It runs with Python 2.4 through 3.7 and is recommended for new applications as well as for migration from older, now obsolete, PySNMP releases. All site documentation and examples are written for the 4.4 and later versions in mind. Older materials are still available under the obsolete section.

Besides the libraries, a set of pure-Python command-line tools are shipped along with the system. Those tools mimic the interface and behaviour of popular Net-SNMP snmpget/snmpset/snmpwalk utilities. They may be useful in a cross-platform situations as well as a testing and prototyping instrument for pysnmp users.

PySNMP software is free and open-source. Source code is hosted in a Github repo. The library is being distributed under 2-clause BSD-style license.

PySNMP library development has been initially sponsored by a PSF grant.

http://snmplabs.com/pysnmp/
https://github.com/etingof/pysnmp

Example: Making an SNMP Query

SNMP: Basic SNMP Query

```python
# Setup SNMP connection and query a MIB
iterator = getCmd(SnmpEngine(),
                  CommunityData(ro_community),
                  UdpTransportTarget((device["address"], device["snmp_port"])),
                  ContextData(),
                  ObjectType(ObjectIdentity('SNMPv2-MIB', 'sysDescr', 0)))

# Process the query
errorIndication, errorStatus, errorIndex, varBinds = next(iterator)

# Check for errors, and if OK, print returned result
if errorIndication:  # SNMP engine errors
    print(errorIndication)
else:
    if errorStatus:  # SNMP agent errors
        print('%s at %s' % (errorStatus.prettyPrint(),
                            varBinds[int(errorIndex)-1] if errorIndex else '?'))
    else:
        for varBind in varBinds:  # SNMP response contents
            print(' = '.join([x.prettyPrint() for x in varBind]))
```

Code edited for display on slide

device_apis/snmp/pysnmp_example1.py

Configuration Management Tools and Libraries

Open Source Python projects for full network config management

Designed for Network Automation

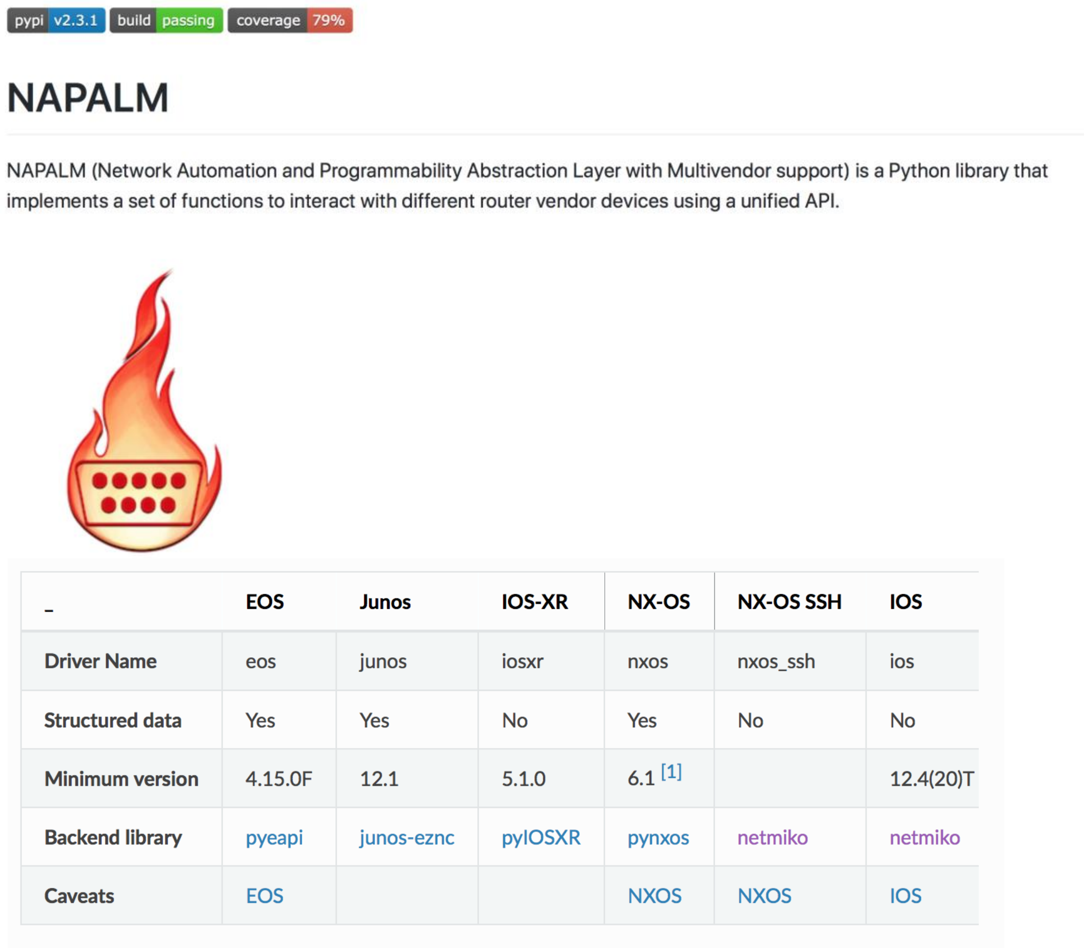
- NAPALM
  - Library providing a standard set of functions for working with different network OS's
- Nornir
  - New automation framework focused on being Python native. Can leverage other tools like NAPALM.

Designed for Server Automation

- Ansible
  - Declarative, agent-less automation framework for managing configuration. Robust support for network platforms
- Salt
  - Configuration management and remote code execution engine. Network automation options in development.

NAPALM – Mature Python Library for Multi-Vendor Interactions

- Robust configuration management options
  - Replace, Merge, Compare, Commit, Discard, Rollback
- Builds on available backend libraries and interfaces (CLI, NX-API, NETCONF, etc)
- Can be used and integrated into other tools (ie Ansible, Nornir)



NAPALM

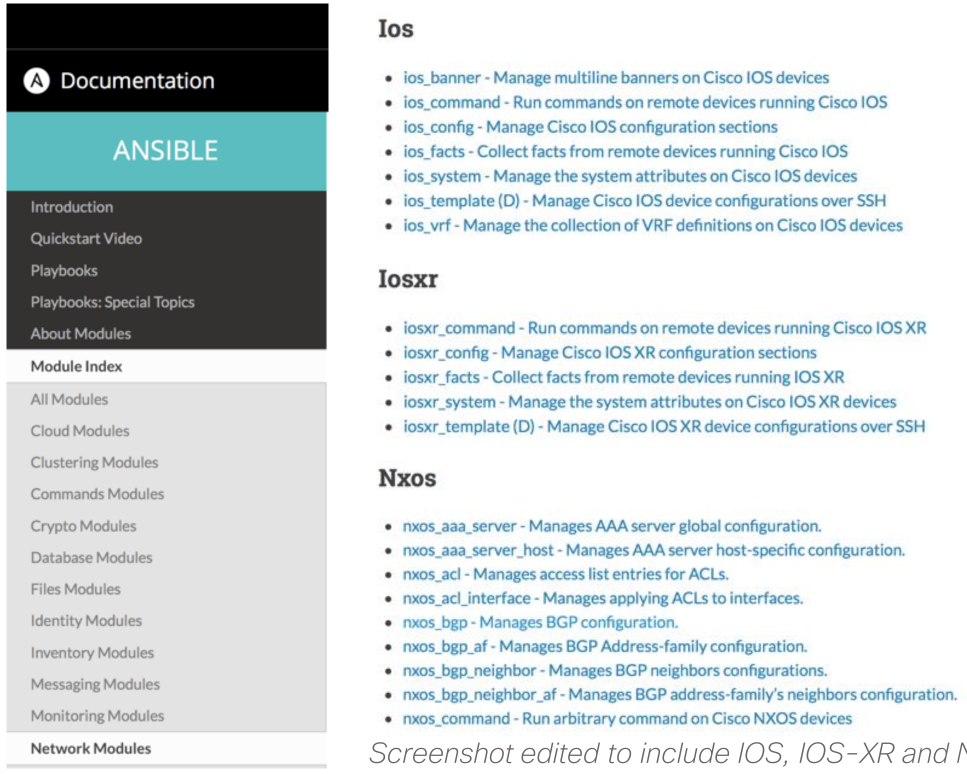
NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor support) is a Python library that implements a set of functions to interact with different router vendor devices using a unified API.

_	EOS	Junos	IOS-XR	NX-OS	NX-OS SSH	IOS
Driver Name	eos	junos	iosxr	nxos	nxos_ssh	ios
Structured data	Yes	Yes	No	Yes	No	No
Minimum version	4.15.0F	12.1	5.1.0	6.1 [1]		12.4(20)T
Backend library	pyeapi	junos-eznc	pyIOSXR	pynxos	netmiko	netmiko
Caveats	EOS			NXOS	NXOS	IOS

https://github.com/napalm-automation/napalm
https://napalm.readthedocs.io

Ansible – Leading DevOps Tool for Network Configuration Management

- Agentless – no edge software installation needed
- Support for both old and new platforms and interfaces (ie CLI & NETCONF)
- Robust and growing library of network modules



Ios

- ios_banner - Manage multiline banners on Cisco IOS devices
- ios_command - Run commands on remote devices running Cisco IOS
- ios_config - Manage Cisco IOS configuration sections
- ios_facts - Collect facts from remote devices running Cisco IOS
- ios_system - Manage the system attributes on Cisco IOS devices
- ios_template (D) - Manage Cisco IOS device configurations over SSH
- ios_vrf - Manage the collection of VRF definitions on Cisco IOS devices

Iosxr

- iosxr_command - Run commands on remote devices running Cisco IOS XR
- iosxr_config - Manage Cisco IOS XR configuration sections
- iosxr_facts - Collect facts from remote devices running IOS XR
- iosxr_system - Manage the system attributes on Cisco IOS XR devices
- iosxr_template (D) - Manage Cisco IOS XR device configurations over SSH

Nxos

- nxos_aaa_server - Manages AAA server global configuration.
- nxos_aaa_server_host - Manages AAA server host-specific configuration.
- nxos_acl - Manages access list entries for ACLs.
- nxos_acl_interface - Manages applying ACLs to interfaces.
- nxos_bgp - Manages BGP configuration.
- nxos_bgp_af - Manages BGP Address-family configuration.
- nxos_bgp_neighbor - Manages BGP neighbors configurations.
- nxos_bgp_neighbor_af - Manages BGP address-family's neighbors configuration.
- nxos_command - Run arbitrary command on Cisco NXOS devices

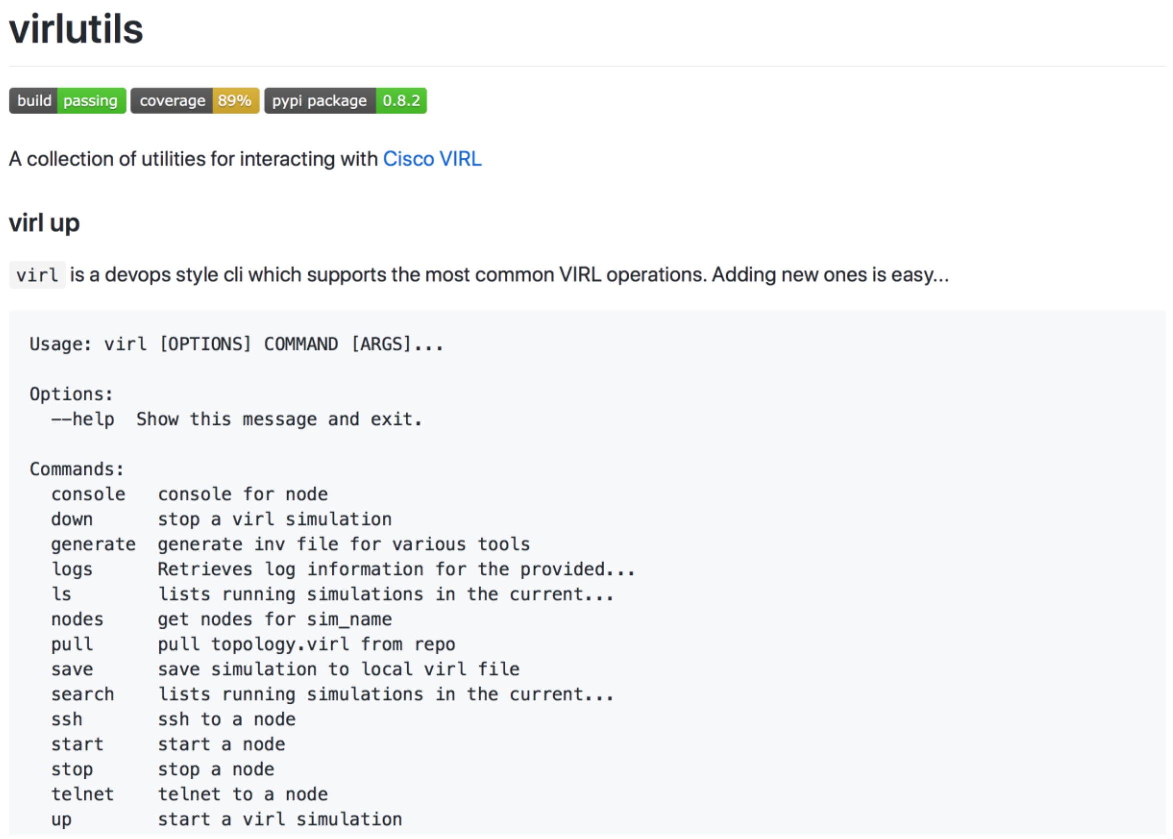
Screenshot edited to include IOS, IOS-XR and NX-OS Content

https://www.ansible.com/overview/networking
https://docs.ansible.com/ansible/latest/modules/list_of_network_modules.html

Some Other Cool Python Stuff

virlutils – It's like "vagrant up" but for the Network!

- Open Source command line utility for managing simulations with Cisco VIRL/CML
- Designed for NetDevOps workflows
  - Development environments
  - Test networks within CICD pipelines



virlutils

A collection of utilities for interacting with Cisco VIRL

virl up

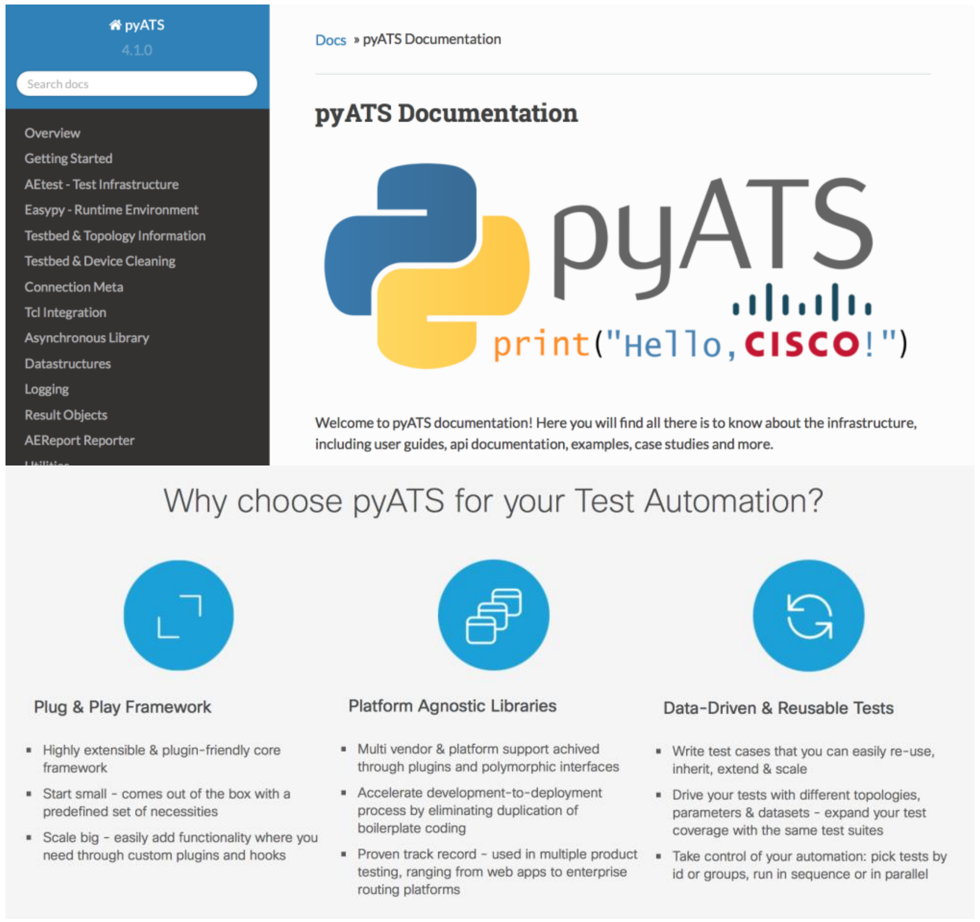
`virl` is a devops style cli which supports the most common VIRL operations. Adding new ones is easy...

```
Usage: virl [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  console   console for node
  down      stop a virl simulation
  generate  generate inv file for various tools
  logs      Retrieves log information for the provided...
  ls        lists running simulations in the current...
  nodes     get nodes for sim_name
  pull      pull topology.virl from repo
  save      save simulation to local virl file
  search    lists running simulations in the current...
  ssh       ssh to a node
  start     start a node
  stop      stop a node
  telnet    telnet to a node
  up        start a virl simulation
```

https://github.com/CiscoDevNet/virlutils
https://learningnetworkstore.cisco.com/virlfaq/aboutVirl

pyATS – Profile and Test Your Network Before, During, and After Changes

- No longer is "ping" the best network test tool available
- PyATS is built to work like software test suites, and uses common frameworks (ie robot)
- Profile the network to get a baseline for interfaces, routing protocols, and platform details – verify at anytime.



pyATS
4.1.0
Search docs

Overview
Getting Started
AEtest - Test Infrastructure
Easypy - Runtime Environment
Testbed & Topology Information
Testbed & Device Cleaning
Connection Meta
Tcl Integration
Asynchronous Library
Datastructures
Logging
Result Objects
AEReport Reporter

Docs » pyATS Documentation

pyATS Documentation

Welcome to pyATS documentation! Here you will find all there is to know about the infrastructure, including user guides, api documentation, examples, case studies and more.

Why choose pyATS for your Test Automation?

Plug & Play Framework

- Highly extensible & plugin-friendly core framework
- Start small - comes out of the box with a predefined set of necessities
- Scale big - easily add functionality where you need through custom plugins and hooks

Platform Agnostic Libraries

- Multi vendor & platform support achived through plugins and polymorphic interfaces
- Accelerate development-to-deployment process by eliminating duplication of boilerplate coding
- Proven track record - used in multiple product testing, ranging from web apps to enterprise routing platforms

Data-Driven & Reusable Tests

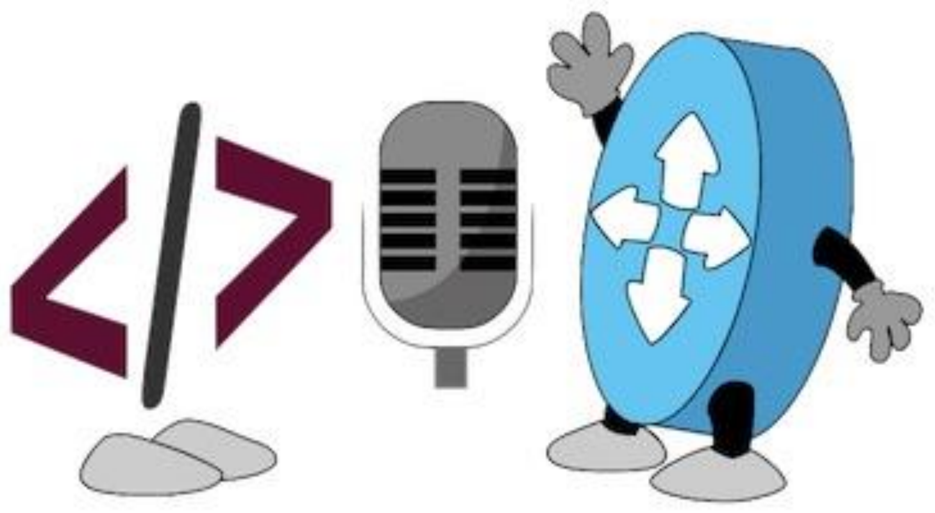
- Write test cases that you can easily re-use, inherit, extend & scale
- Drive your tests with different topologies, parameters & datasets - expand your test coverage with the same test suites
- Take control of your automation: pick tests by id or groups, run in sequence or in parallel

https://developer.cisco.com/site/pyats/
https://developer.cisco.com/docs/pyats/

Summing up

What did we talk about?

- Libraries to Work with Data
  - xmltodict, json, PyYAML, csv, pyang
- API Libraries
  - requests, ncclient, netmiko, pysnmp
- Configuration Management
  - NAPALM, Ansible, Salt, Nornir
- Some Other Cool Python Stuff
  - virlutils, pyATS



Webinar Resource List

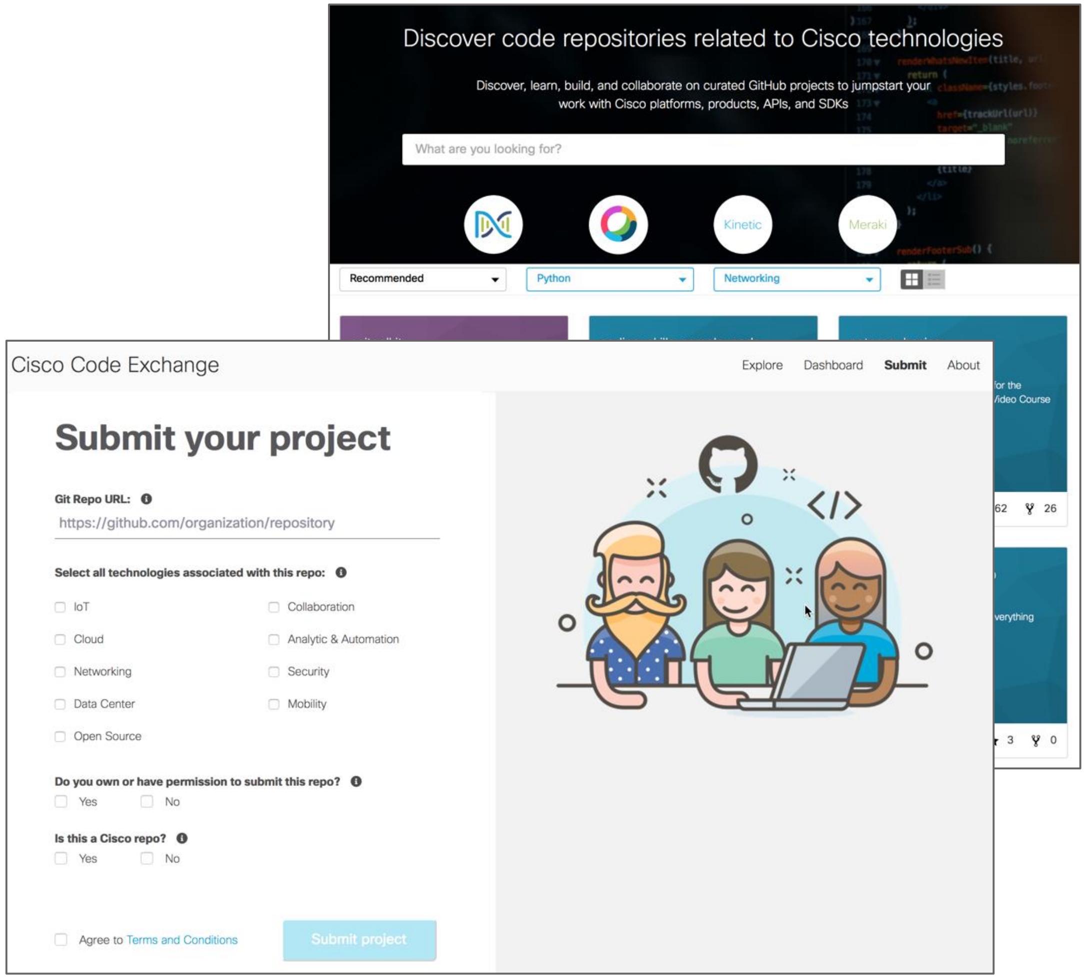
- Docs and Links
  - https://developer.cisco.com/python
- Learning Labs
  - Laptop Setup http://cs.co/lab-dev-setup
  - Coding Fundamentals http://cs.co/lab-coding-fundamentals
  - Model Driven Programmability http://cs.co/lab-mdp
- DevNet Sandboxes
  - IOS Always On http://cs.co/sbx-iosxe
  - NX-OS Always On http://cs.co/sbx-nxos
- Code Samples
  - http://cs.co/code-python-networking

NetDevOps Live! Code Exchange Challenge

developer.cisco.com/codeexchange

Use one or more of the libraries discussed to write a Python script to automate one common networking task.

Example: Compile the MAC and ARP tables from all devices on the network.



Discover code repositories related to Cisco technologies

Discover, learn, build, and collaborate on curated GitHub projects to jumpstart your work with Cisco platforms, products, APIs, and SDKs

What are you looking for?

Cisco Code Exchange

Submit your project

Git Repo URL: https://github.com/organization/repository

Select all technologies associated with this repo:

☐ IoT	☐ Collaboration
☐ Cloud	☐ Analytic & Automation
☐ Networking	☐ Security
☐ Data Center	☐ Mobility
☐ Open Source	

Do you own or have permission to submit this repo? ☐ Yes ☐ No

Is this a Cisco repo? ☐ Yes ☐ No

☐ Agree to Terms and Conditions

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!

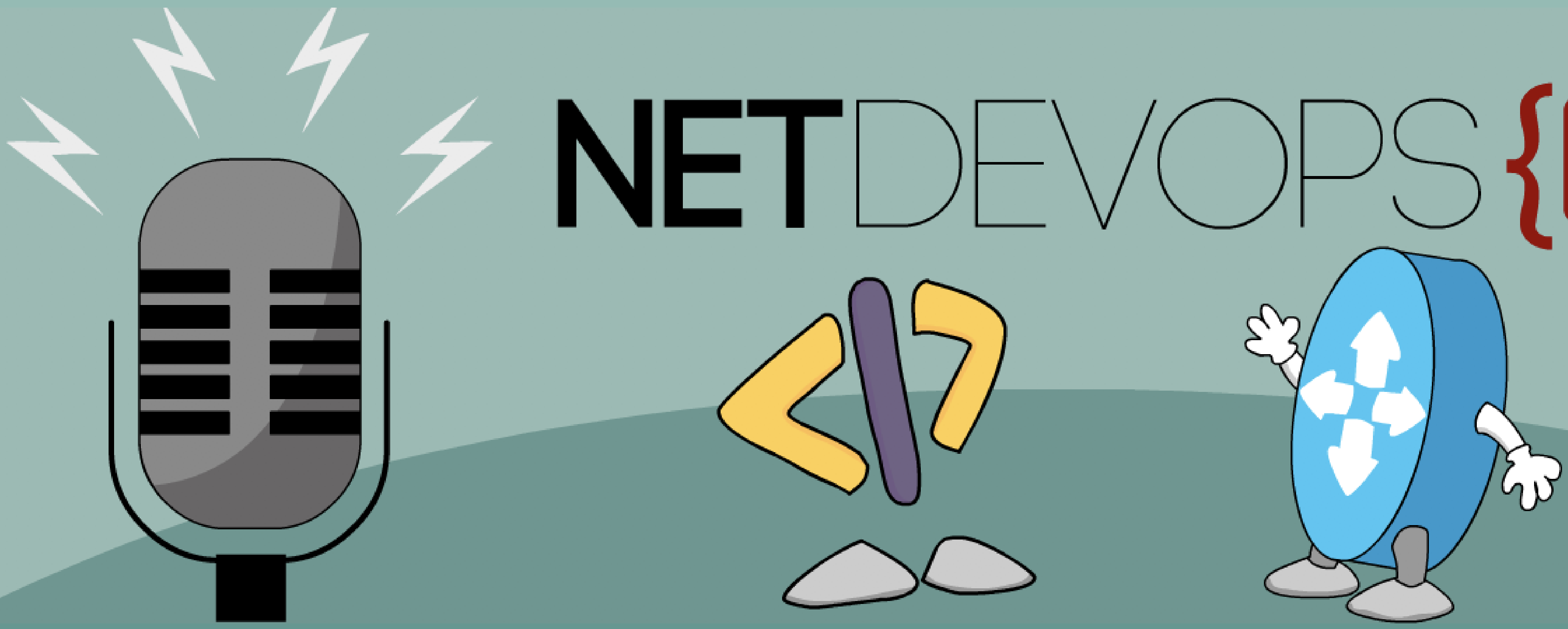
Hank Preston

hapresto@cisco.com
@hfpreston
http://github.com/hpreston

CISCO DEVNET
LEARN CODE INSPIRE CONNECT

developer.cisco.com

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



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

@netdevopslive