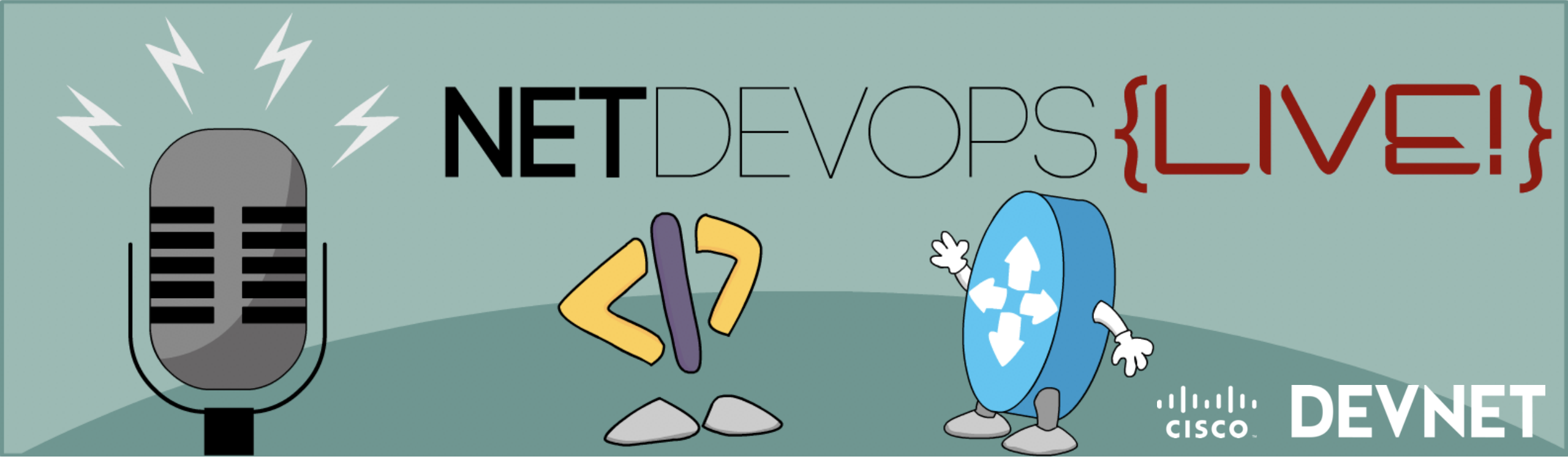


Profile, Test and Verify your Network is Running Smoothly with pyATS

Kevin Corbin, ccie 11577

Cisco Automation Dude

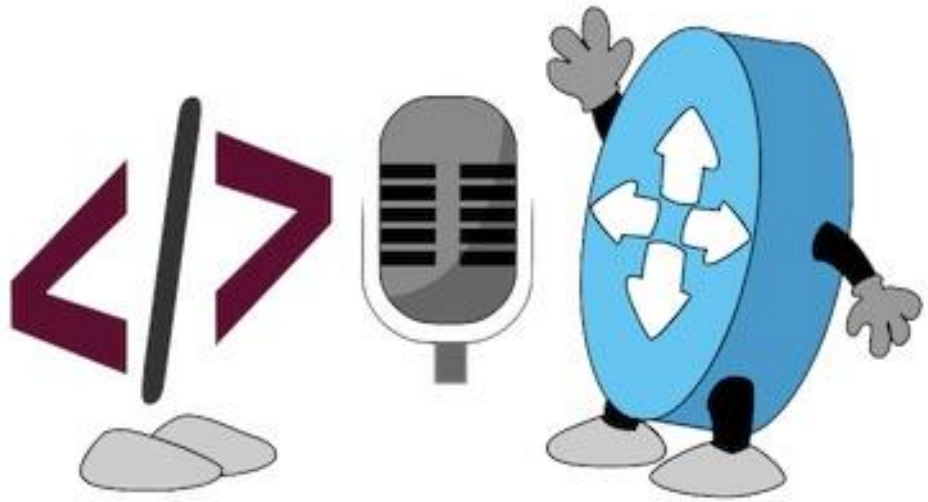
Twitter: @kecorbin

Season 1, Talk 8

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

What are we going to talk about?

- Continuous Network Verification Vison
- A brief history of pyATS, Genie
- pyATS– it does things
- Genie – it does stuff
- Demos



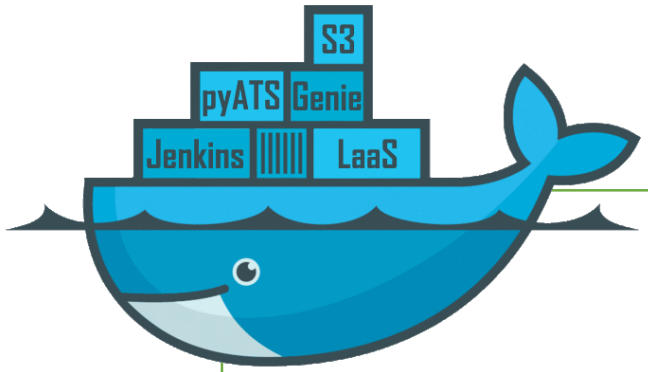
Device Health Checks

Configurational State

Operational Status

Device Health

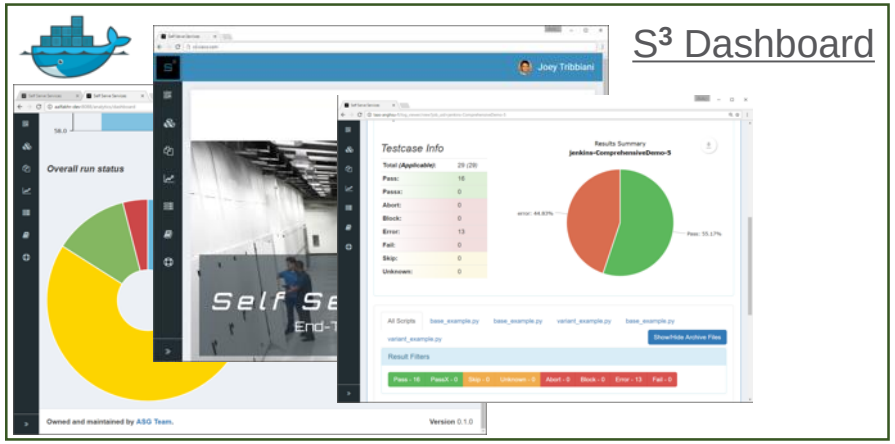
Continuous Network Verification Vision

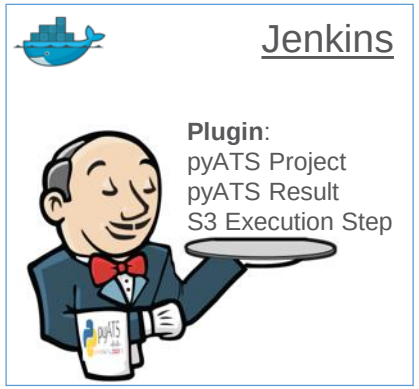


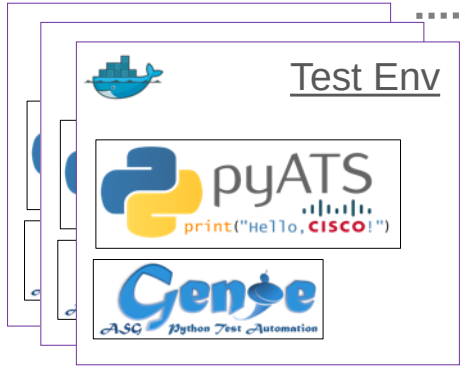
- Network Engineers
- Developers
- Customers

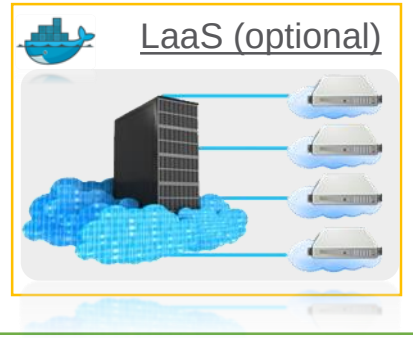
Virtual Machine
(Docker Server)

kafka









pyATS, 2014 – present

- Launched internally in Cisco engineering late 2014
- Quickly became the most adopted test framework within Cisco
- Running in sanity, regression, solution labs, etc.

3000+ Internal
Engineers/consumers

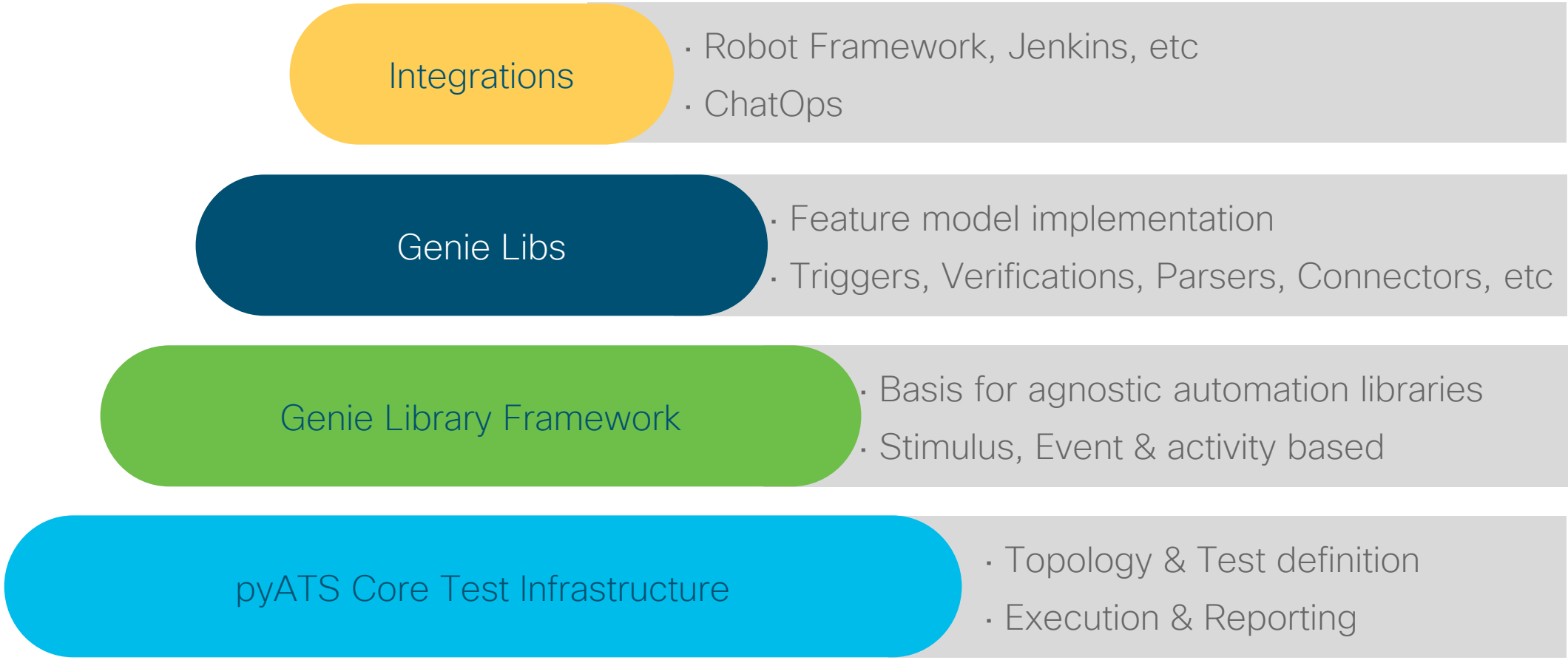
5,000,000+ LoC
Testcases/Scripts

1,000,000+ runs
month

Congratulations!!! (and thanks!)

- Sedy Yadollahi
- Siming Yuan
- Jean-Benoit Aubin
- Karim Mohamed
- Lubna Rasheed
- Takashi Higashimura

pyATS Framework



pyATS- it's pythonic, it does *things*

pyATS – The toolbox

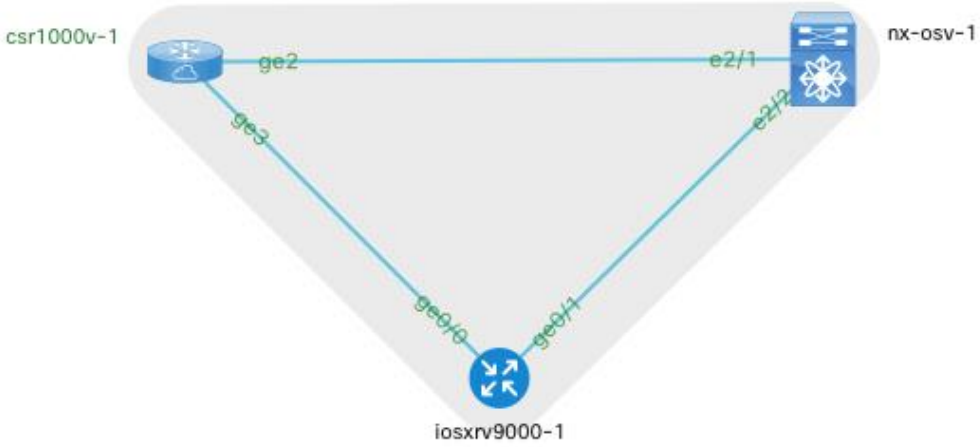
- pythonic infrastructure & object-oriented programming paradigm
- providing the most fundamental, common toolset needed by everyone
- Agile software development methodology
- Capable of leveraging existing tools and libraries

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

"There's a difference between knowing the path, and walking the path." - Morpheus

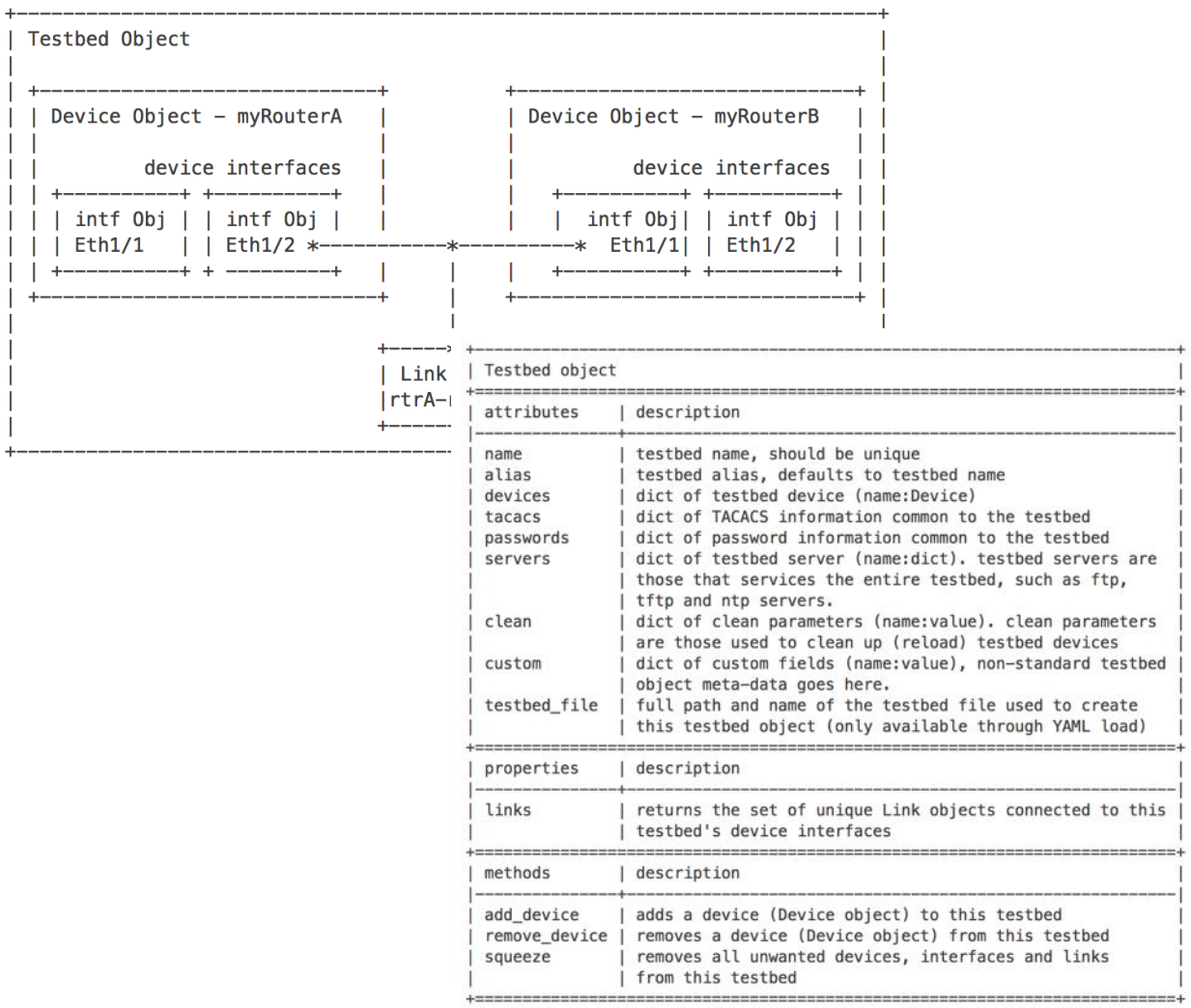
The Network is the Testbed

- **Everything is an Object!**
- using object attributes to store information and meta-data
- using object relationships (references/pointers to other objects) to represent topology interconnects



https://pubhub.devnetcloud.com/media/pyats/docs/topology/concept.html

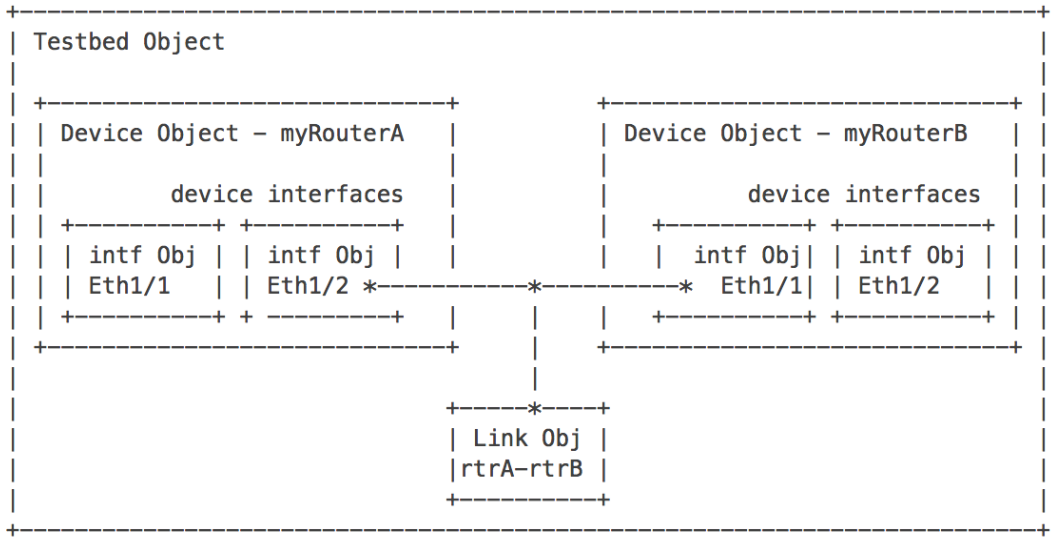
Testbeds are YAML




```yaml
testbed:
  name: 3-router-topo

devices:
  csr:
    connections:
      console:
        ip: 10.94.242.171
        protocol: telnet

  n9k:
    connections:
      console:
        ip: 10.94.242.172
        protocol: telnet

  asr9k:
    connections:
      console:
        ip: 10.94.242.173
        protocol: telnet
```

```yaml
topology:
  csr:
    interfaces:
      GigabitEthernet2:
        link: csr-to-n9k
      GigabitEthernet3:
        link: csr-to-asr9k

  n9k:
    interfaces:
      Ethernet2/1:
        link: csr-to-n9k
      Ethernet2/2:
        link: n9k-to-asr9k

  asr9k:
    interfaces:
      GigabitEthernet0/0:
        link: csr-to-asr9k
      GigabitEthernet0/1:
        link: n9k-to-asr9k
```

Testbed (Network) Object



Testbed object

attributes	description
name	testbed name, should be unique
alias	testbed alias, defaults to testbed name
devices	dict of testbed device (name:Device)
tacacs	dict of TACACS information common to the testbed
passwords	dict of password information common to the testbed
servers	dict of testbed server (name:dict). testbed servers are those that services the entire testbed, such as ftp, tftp and ntp servers.
clean	dict of clean parameters (name:value). clean parameters are those used to clean up (reload) testbed devices
custom	dict of custom fields (name:value), non-standard testbed object meta-data goes here.
testbed_file	full path and name of the testbed file used to create this testbed object (only available through YAML load)

properties	description
links	returns the set of unique Link objects connected to this testbed's device interfaces

methods	description
add_device	adds a device (Device object) to this testbed
remove_device	removes a device (Device object) from this testbed
squeeze	removes all unwanted devices, interfaces and links from this testbed

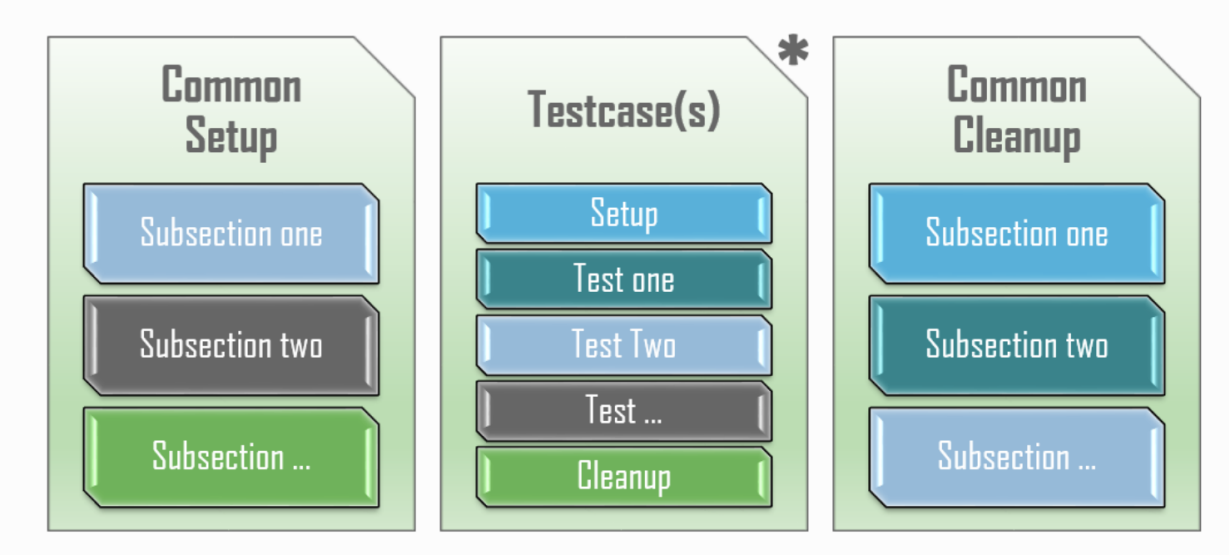
Devices Objects

- Devices represent *common operations as methods*
  - *connect()*
  - *ping(destination)*
  - *execute('show version')*
  - *configure('no ip domain-lookup')*
- *Output can be parsed directly, or using other libraries*
  - ***Genie**, TextFSM, regex*

Device object	
attributes	**description**
name	device name (a.k.a hostname)
alias	device alias, defaults to device name
type	device type (string)
testbed	parent testbed object. internally this is a weakref
interfaces	dict of device interfaces (name:Interface)
tacacs	dict of TACACS information unique to this device
passwords	dict of password information unique to the device
connections	dict of connection descriptions (name:dict). this is a description of connection methods to this device (eg: telnet, ssh, netconf & etc)
connectionmgr	connection manager (ConnectionManager obj), manages all the connections to this device
clean	dict of clean parameters (name:value). clean params are those used to clean up (reload) this device
custom	dict of custom fields (name:value), non-standard device object meta-data goes here.
properties	**description**
links	returns the set of unique Link objects connected to this device's interfaces
remote_devices	returns the set of unique devices connected to this device via its interface links
remote_interfaces	returns the set of unique interfaces connected to this device's interfaces via interface links
methods	**description**
add_interface	adds an interface (Interface object) to this device
remove_interface	removes an interface (Interface object) from this device
find_links	find and return a set of links connected to the provided destination object (Device/Interface)

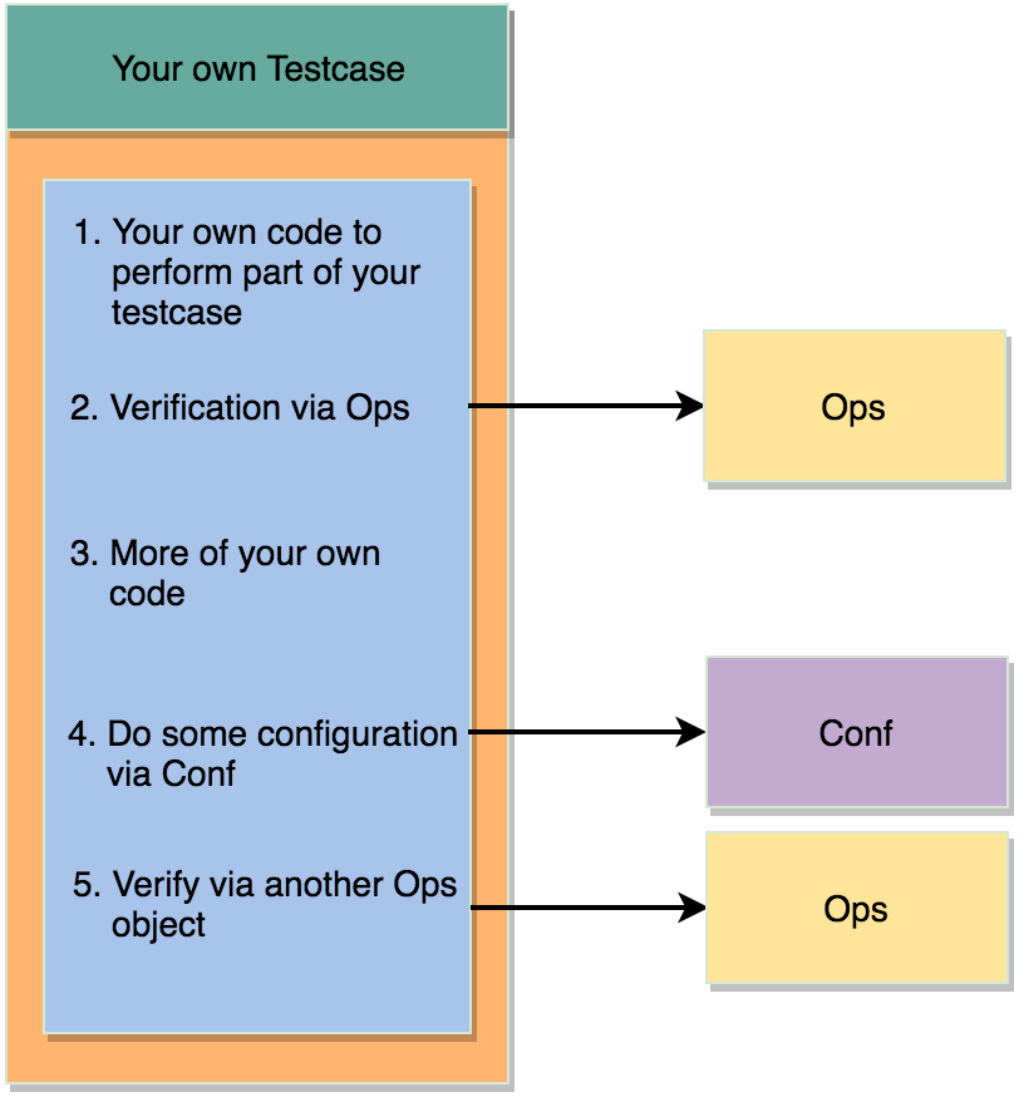
Test Cases

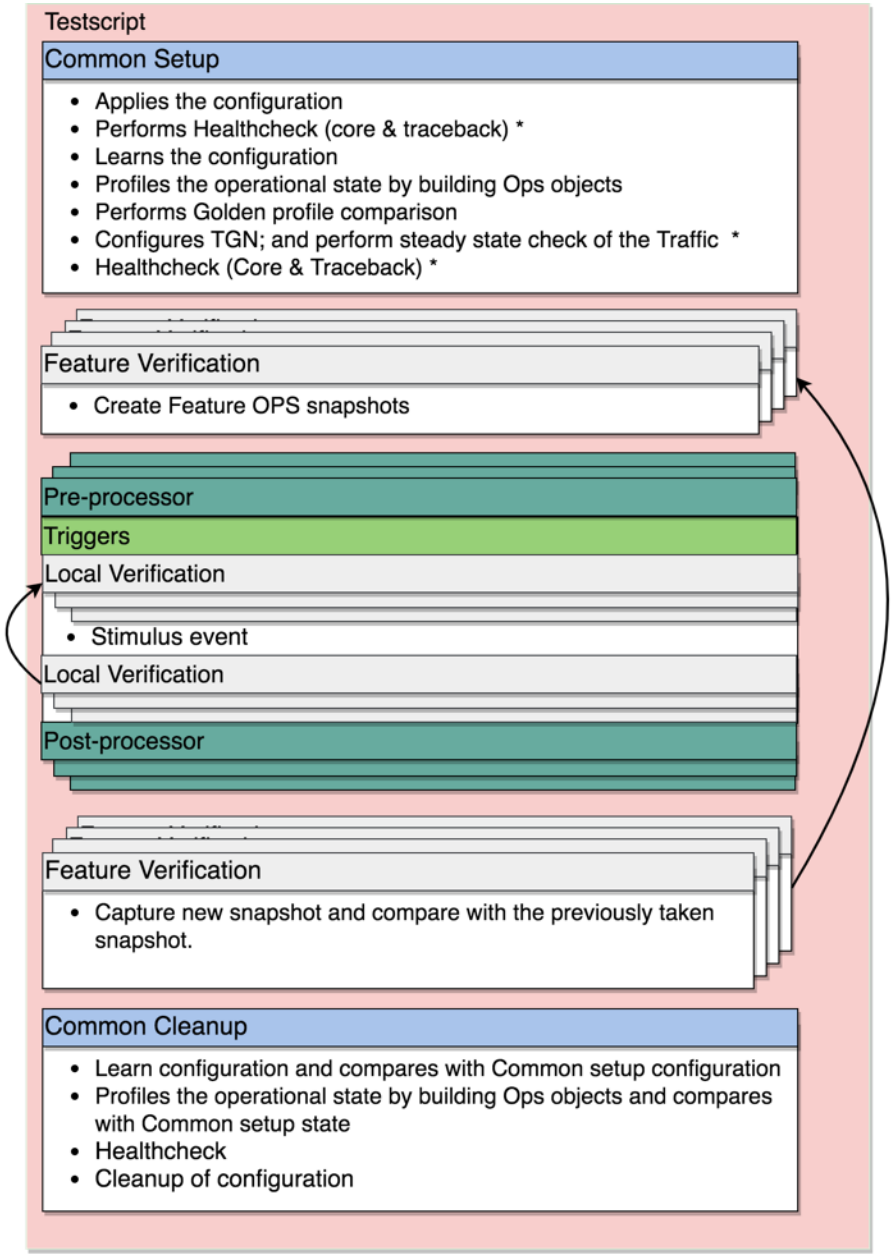
- Written in Python
- Framework Provides
  - logging
  - results tracking/reporting
  - common services (ping, execute, config)
- test control/flow (loops, etc)



https://pubhub.devnetcloud.com/media/pyats/docs/aetest/structure.html

Test Cases


Your own Testcase
1. Your own code to perform part of your testcase
2. Verification via Ops
3. More of your own code
4. Do some configuration via Conf
5. Verify via another Ops object
Ops
Conf
Ops


Testscript
Common Setup
• Applies the configuration
• Performs Healthcheck (core & traceback) *
• Learns the configuration
• Profiles the operational state by building Ops objects
• Performs Golden profile comparison
• Configures TGN; and perform steady state check of the Traffic *
• Healthcheck (Core & Traceback) *
Feature Verification
• Create Feature OPS snapshots
Pre-processor
Triggers
Local Verification
• Stimulus event
Local Verification
Post-processor
Feature Verification
• Capture new snapshot and compare with the previously taken snapshot.
Common Cleanup
• Learn configuration and compares with Common setup configuration
• Profiles the operational state by building Ops objects and compares with Common setup state
• Healthcheck
• Cleanup of configuration

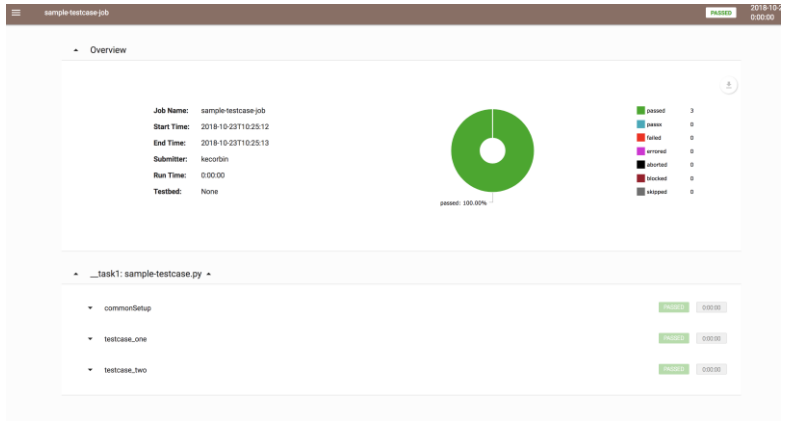
Test Results / Reports

```python
from ats import aetest


# defining a common setup section
# contains a subsection that is looped twice.
class CommonSetup(aetest.CommonSetup):

    # defining a subsection
    # this subsection is marked to be looped twice
    # the first time having a uid of "subsection_one", and
    # the second time having a uid of "subsection_two"
    @aetest.loop(uids=['subsection_one', 'subsection_two'])
    @aetest.subsection
    def looped_subsection(self):
        pass


# defining a testcase that loops
# this testcase also contains a test section that is looped twice
@aetest.loop(uids=['testcase_one', 'testcase_two'])
class Testcase(aetest.Testcase):

    # setup section of this testcase is run once
    # every time the testcase is looped.
    @aetest.setup
    def setup(self):
        pass

    # looped test section
    # both iterations are run per testcase iteration
    @aetest.loop(uids=['test_one', 'test_two'])
    @aetest.test
    def test(self):
        pass

    # cleanup section of this testcase is run once
    # every time the testcase is looped.
    @aetest.cleanup
    def cleanup(self):
        pass
```

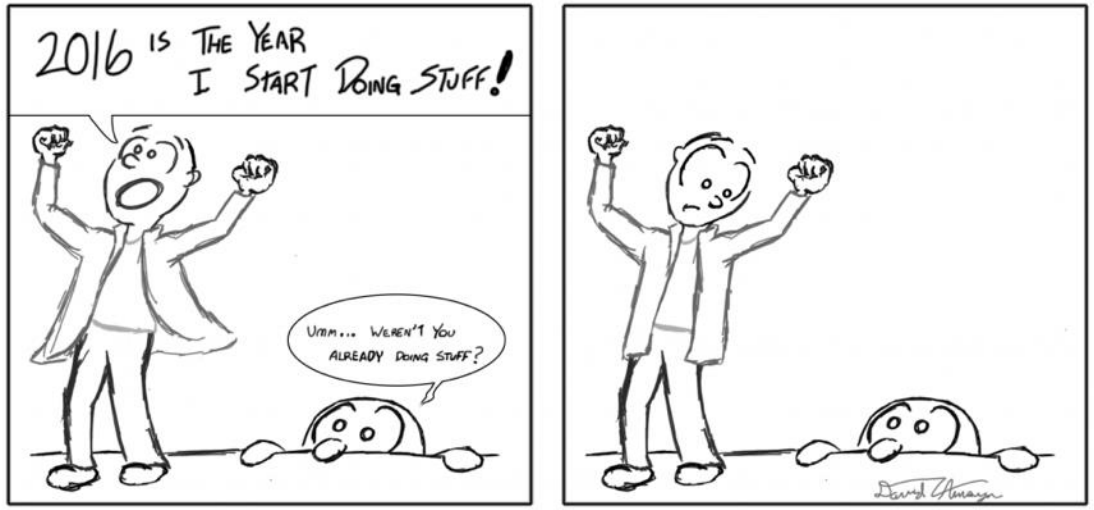
Console Test Report

```
2018-10-23T10:25:12: %EASYPY-INFO: +------------------------------------------------------------------------------+
2018-10-23T10:25:12: %EASYPY-INFO: |                            Task Result Summary                               |
2018-10-23T10:25:12: %EASYPY-INFO: +------------------------------------------------------------------------------+
2018-10-23T10:25:12: %EASYPY-INFO: common_setup                                                          PASSED
2018-10-23T10:25:12: %EASYPY-INFO: testcase_one                                                          PASSED
2018-10-23T10:25:12: %EASYPY-INFO: testcase_two                                                          PASSED
2018-10-23T10:25:12: %EASYPY-INFO:
2018-10-23T10:25:12: %EASYPY-INFO: +------------------------------------------------------------------------------+
2018-10-23T10:25:12: %EASYPY-INFO: |                            Task Result Details                               |
2018-10-23T10:25:12: %EASYPY-INFO: +------------------------------------------------------------------------------+
2018-10-23T10:25:12: %EASYPY-INFO: common_setup
2018-10-23T10:25:12: %EASYPY-INFO: |-- commonSetup                                                           PASSED
2018-10-23T10:25:12: %EASYPY-INFO: |   |-- subsection_one                                                   PASSED
2018-10-23T10:25:12: %EASYPY-INFO: |   `-- subsection_two                                                   PASSED
2018-10-23T10:25:12: %EASYPY-INFO: |-- testcase_one                                                         PASSED
2018-10-23T10:25:12: %EASYPY-INFO: |   |-- setup                                                            PASSED
2018-10-23T10:25:12: %EASYPY-INFO: |   |-- test_one                                                         PASSED
2018-10-23T10:25:12: %EASYPY-INFO: |   |-- test_two                                                         PASSED
2018-10-23T10:25:12: %EASYPY-INFO: |   `-- cleanup                                                          PASSED
2018-10-23T10:25:12: %EASYPY-INFO: `-- testcase_two                                                         PASSED
2018-10-23T10:25:12: %EASYPY-INFO:     |-- setup                                                            PASSED
2018-10-23T10:25:12: %EASYPY-INFO:     |-- test_one                                                         PASSED
2018-10-23T10:25:12: %EASYPY-INFO:     |-- test_two                                                         PASSED
2018-10-23T10:25:12: %EASYPY-INFO:     `-- cleanup                                                          PASSED
2018-10-23T10:25:12: %EASYPY-INFO: No SMTP server information configured, ignoring sending notification email.
2018-10-23T10:25:12: %EASYPY-INFO: Done!
```

HTMLTest Report



Genie
-it does *stuff*



http://davidamayahumanbeing.com/2016-the-year-i-start-doing-stuff/#sthash.ov80GZ5j.dpbs

Genie High Level Features

`genie.conf`

conf t

`genie.ops`

show

`genie.sdk`

clear ip bgp
show ip bgp

Provides ***feature-centric*** object models

- Focuses development effort on writing test cases & suites
- Shields the end scripter from explicit CLI/YANG-RPCs

Objects are ***agnostic***

- Works across management interfaces: CLI, YANG, XML, etc.
- Handles feature differences between images, releases, platforms, etc.

Genie is ***plug & play***

- Use only the classes you need
- SDK's triggers and verifications **plug directly into pyATS as test cases and sections**

Genie is ***extensible***

- Inherent & extend whenever needed
- Modify only what's required & accommodate for deltas between release/image/etc.

CLI Auto Parser

```
RP/0/0/CPU0:one#show arp location 0/0/CPU0
Thu May 13 11:59:18.909 EDT

Address         Age        Hardware Addr   State      Type  Interface
10.10.10.1      -          02b7.a23a.e076  Interface  ARPA  GigabitEthernet0/0/0/1
11.11.11.1      -          0292.77d4.d5ee  Interface  ARPA  GigabitEthernet0/0/0/0
10.10.10.2      00:00:31   02db.ebba.ecc4  Dynamic    ARPA  GigabitEthernet0/0/0/1
11.11.11.2      00:00:31   02e9.4522.5326  Dynamic    ARPA  GigabitEthernet0/0/0/0
RP/0/0/CPU0:one#
```

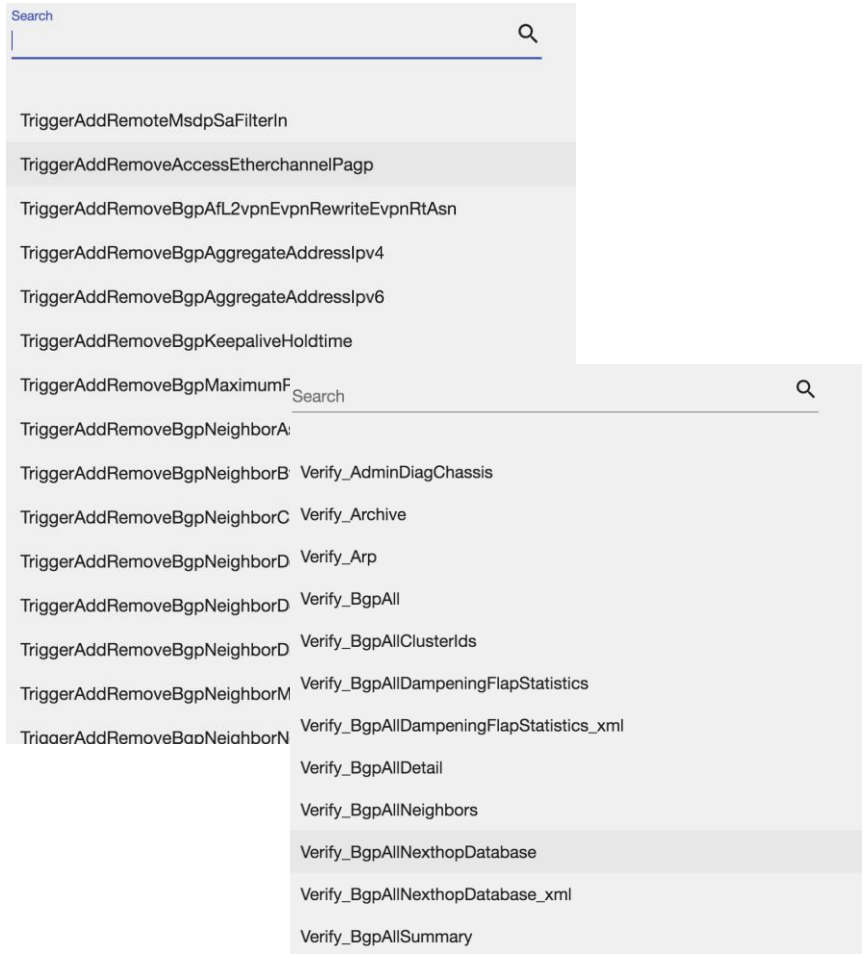
```
def test_log_arp_results_2 (self):
    res = parsergen.oper_fill_tabular(device=device1,
                                      show_command="show arp location 0/0/CPU0",
                                      header_fields =
                                        [ "Address",
                                          "Age",
                                          "Hardware Addr",
                                          "State",
                                          "Type",
                                          "Interface" ])
    log.info("Results:\n" + pprint.pformat(res.entries))
```

```
Results:
{'12.12.12.1': {'Address': '12.12.12.1',
                'Age': '-',
                'Hardware Addr': '0292.77d4.d5ee',
                'Interface': 'GigabitEthernet0/0/0/0',
                'State': 'Interface',
                'Type': 'ARPA'},
 '12.12.12.2': {'Address': '12.12.12.2',
                'Age': '00:47:27',
                'Hardware Addr': '02e9.4522.5326',
                'Interface': 'GigabitEthernet0/0/0/0',
                'State': 'Dynamic',
                'Type': 'ARPA'},
 '13.13.13.1': {'Address': '13.13.13.1',
                'Age': '-',
                'Hardware Addr': '02b7.a23a.e076',
                'Interface': 'GigabitEthernet0/0/0/1',
                'State': 'Interface',
                'Type': 'ARPA'},
 '13.13.13.2': {'Address': '13.13.13.2',
                'Age': '00:47:27',
                'Hardware Addr': '02db.ebba.ecc4',
                'Interface': 'GigabitEthernet0/0/0/1',
                'State': 'Dynamic',
                'Type': 'ARPA'}}
TEST 2010-01-22 16:26:41,113: PASS: example_suite_t.test_log_arp_results_2
```

https://pubhub.devnetcloud.com/media/pyats-packages/docs/parsergen/tabular/tabular.html#full-show-arp-example

Triggers and Verifications

- Triggers
  - In **testing** environments can be used for destructive tests
  - In **production** environments can be used for recovery actions
  - In **awesome** environments, triggers are chaos monkey
- Verifications
  - Before/After Snapshots



https://pubhub.devnetcloud.com/media/pyats-packages/docs/genie/genie_libs/#/triggers

Integrations – Robot Framework

Compare Before/After

```
*** Settings ***
Library         ats.robot.pyATSRobot
Library         genie.libs.robot.GenieRobot

*** Variables ***
${testbed}      tb.yaml
${datafile}     datafile.yaml
${profiles}     /profiles

Connect
    use testbed "${testbed}"
    connect to device "csr"
    connect to device "n9k"

Profile System
    profile system for "running-config,bgp,ospf,interface" on device "csr" and store as "current"
    profile system for "running-config,bgp,ospf,interface" on device "n9k" and store as "current"

Compare Profiles
    compare profile "${profiles}/csr.profile" with "current" from device"csr"
    compare profile "${profiles}/nxos.profile" with "current" from device"n9k"
```

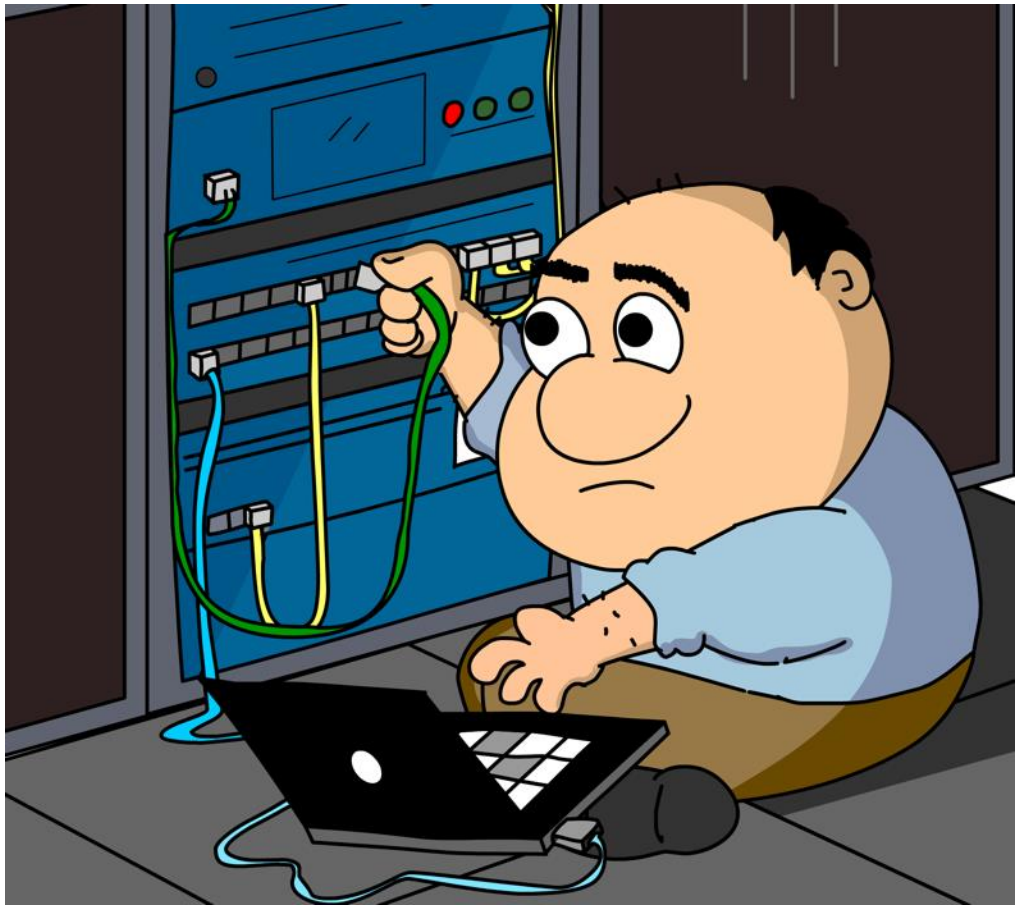
Test vs. Expected Values

```
*** Settings ***
Library         ats.robot.pyATSRobot
Library         genie.libs.robot.GenieRobot

*** Variables ***
${testbed}                  tb.yaml
${verification_datafile}    verification_datafile.yaml

Connect
    use testbed "${testbed}"
    connect to device "csr"
    connect to device "n9k"
    connect to device "asr9k"

Verify Interface and Ospf on n9k
    verify count "2" "interface up" on device "n9k"
    verify count "2" "ospf neighbors" on device "n9k"

Verify Redundancy Status
    run verification "Verify_RedundancyStatus" on device "asr9k"

Verify Bgp
    run verification "Verify_BgpVrfAllNeighbors_vrf_default" on device "csr"
```

Additional Available Keywords

```
use genie testbed "${testbed}"
learn "${feature:[^"]+}" on device "${device:[^"]+}"
learn "${feature:[^"]+}" on device "${device:[^"]+}" using alias "${alias:[^"]+}"
learn "${feature:[^"]+}" on device "${device:[^"]+}" using alias "${alias:[^"]+}" with context "${context:[^"]+}"
learn "${feature:[^"]+}" on device "${device:[^"]+}" with context "${context:[^"]+}"
parse "${parser:[^"]+}" on device "${device:[^"]+}"
parse "${parser:[^"]+}" on device "${device:[^"]+}" using alias "${alias:[^"]+}"
parse "${parser:[^"]+}" on device "${device:[^"]+}" using alias "${alias:[^"]+}" with context "${context}"
parse "${parser:[^"]+}" on device "${device:[^"]+}" with context "${context}"
run trigger "${name:[^"]+}" on device "${device:[^"]+}"
run trigger "${name:[^"]+}" on device "${device:[^"]+}" using alias "${alias:[^"]+}"
run trigger "${name:[^"]+}" on device "${device:[^"]+}" with context "${context:[^"]+}"
run trigger "${name}" on device "${device:[^"]+}" using alias "${alias:[^"]+}" with context "${context:[^"]+}"
run verification "${name:[^"]+}" on device "${device:[^"]+}"
run verification "${name:[^"]+}" on device "${device:[^"]+}" using alias "${alias:[^"]+}"
run verification "${name:[^"]+}" on device "${device:[^"]+}" using alias "${alias:[^"]+}" with context "${context:[^"]+}"
run verification "${name:[^"]+}" on device "${device:[^"]+}" with context "${context:[^"]+}"
verify count "${number:[^"]+}" "${structure:[^"]+}" on device "${device:[^"]+}"
verify count "${number:[^"]+}" "${structure:[^"]+}" on device "${device:[^"]+}" using alias "${alias:[^"]+}"
```

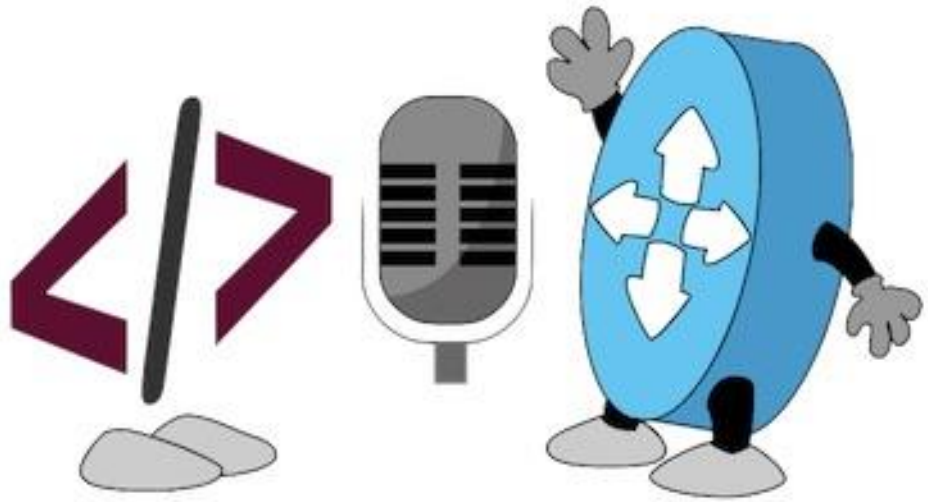
Demo Time!



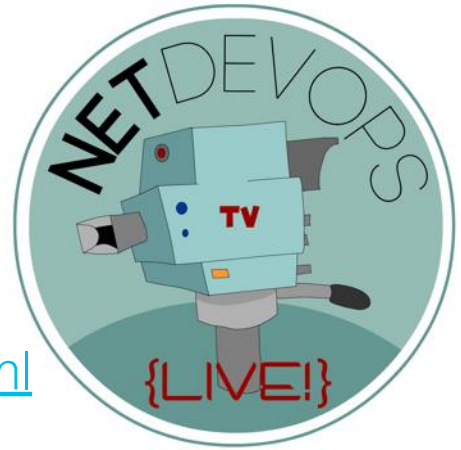
Summing up

What did we talk about?

- Continuous Network Verification Vison
- A brief history of pyATS, Genie
- pyATS– it does things
- Genie – it does stuff
- Demos



Webinar Resource List



- Documentation (it's really good)
  - https://pubhub.devnetcloud.com/media/pyats/docs/overview/index.html
- Code
  - https://github.com/CiscoDevNet/pyats-sample-scripts
  - https://github.com/kecorbin/pyats-network-checks
  - https://github.com/kecorbin/ipyats
- DevNet Sandboxes
  - Multi-IOS Sandbox with VIRL and NSO! http://cs.co/sbx-multi
  - Accompanying code samples http://cs.co/code-sbx-multi
- Learning Labs
  - https://katacoda.com/kecorbin

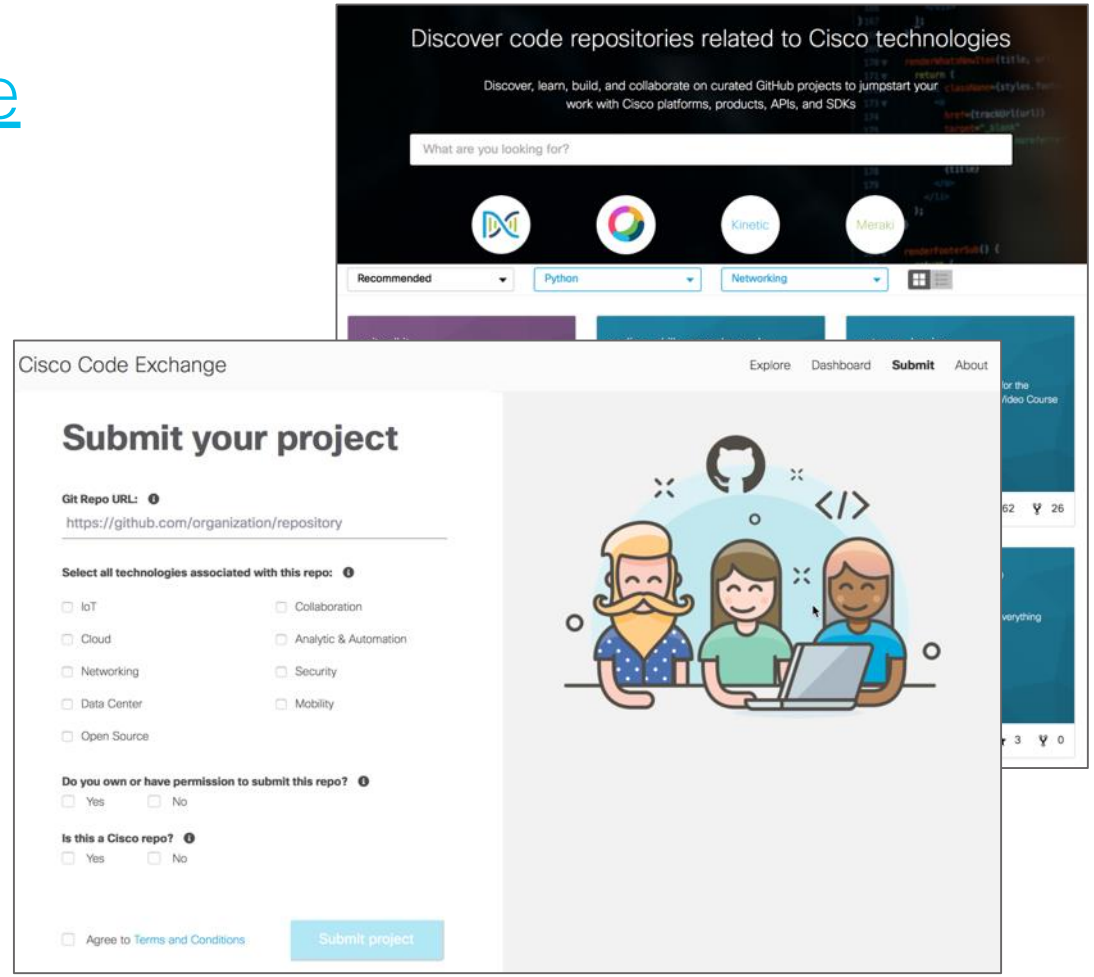
NetDevOps Live! Code Exchange Challenge

developer.cisco.com/codeexchange

Write your own Network Verifications

Examples:

- *Reachability Tests*
- *HSRP Status*
- *BGP/OSPF States*
- *Interface Counts/Counters*
- *Checks for critical routes*

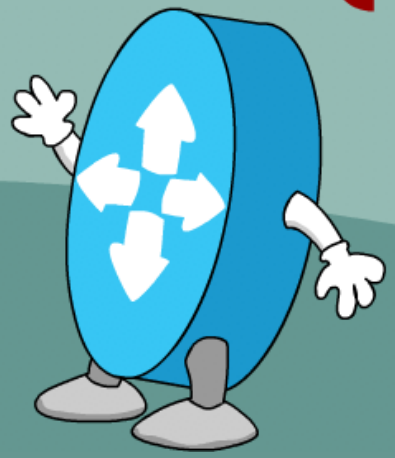


Looking for more about NetDevOps?

- 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/

NETDEVOPS {LIVE!}

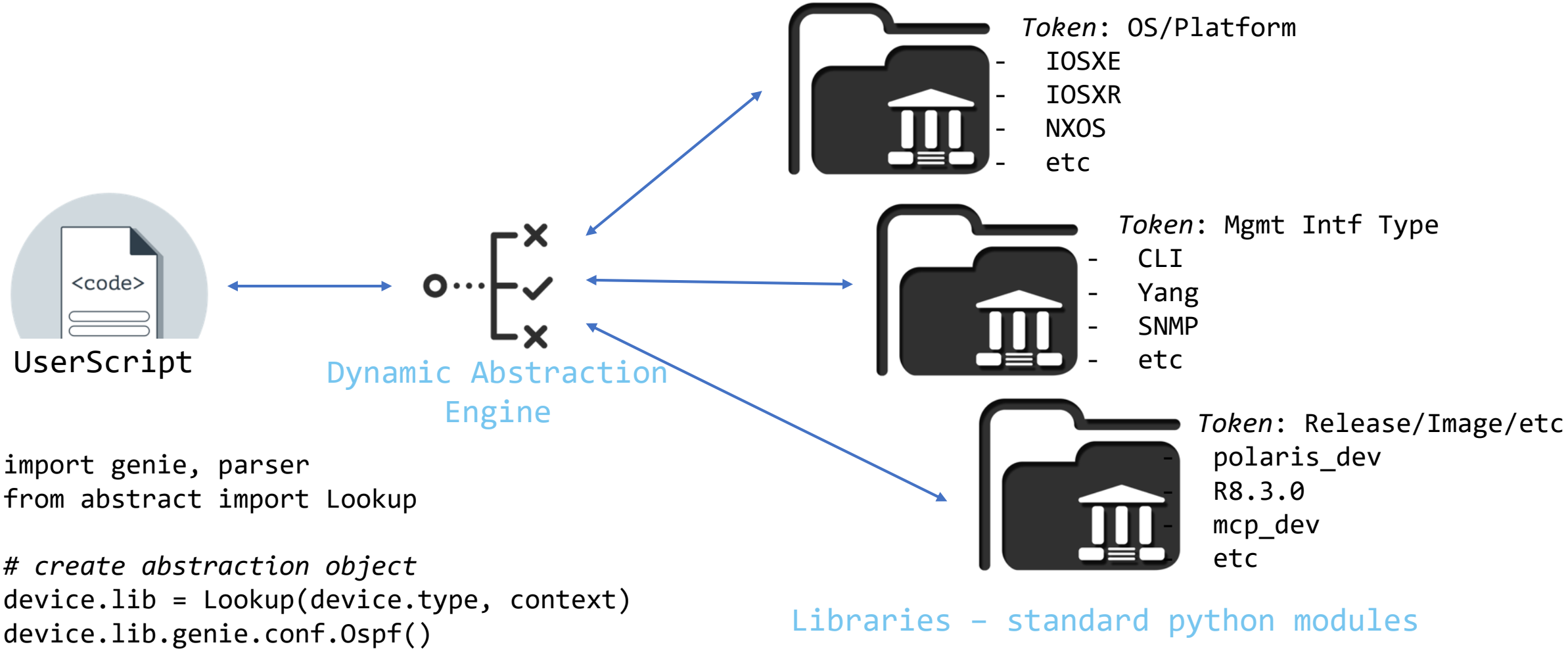
cisco DEVNET



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

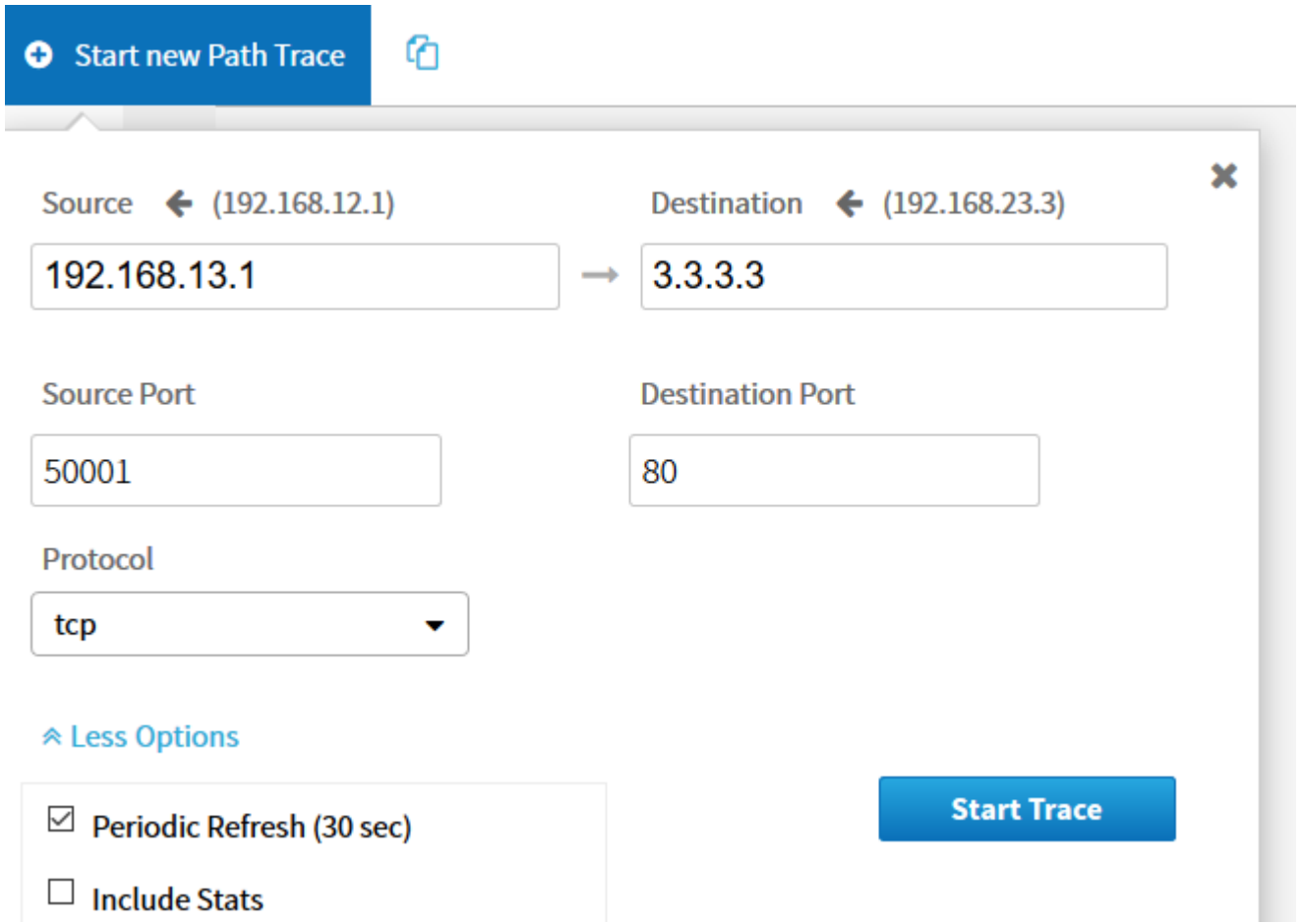
@netdevopslive

Dynamic Abstraction Design




```
import genie, parser
from abstract import Lookup

# create abstraction object
device.lib = Lookup(device.type, context)
device.lib.genie.conf.Ospf()
```

Genie in APP Testing




```
# top level object model
class PathTrace(genie.conf.Base):
    source = ManagedAttribute(
                type=ip_address, name=source,
                description="path trace source")
    destination = ManagedAttribute(
                type=ip_address, name=source,
                description="path trace dest")
    protocol = ManagedAttribute(
                 type=str, name="protocol",
                 description="protocol seletion")

    ...


    def build_config(self, device):
        raise NotImplementedError

# genie_libs/conf/path_trace/rest/
class PathTrace():
    def build_config(self, device):
        # call rest api library to do work

# genie_libs/conf/path_trace/ui/
Class PathTrace():
    def build_config(self, device):
        # invoke selenium driver
```