cisco Live!

Let's go

#CiscoLive

The bridge to possible

Test Driven Automation with pyATS

A simplified approach to network automation

John Capobianco – Developer Advocate
@john_capobianco

BRKCRT-2013

Cisco Webex App

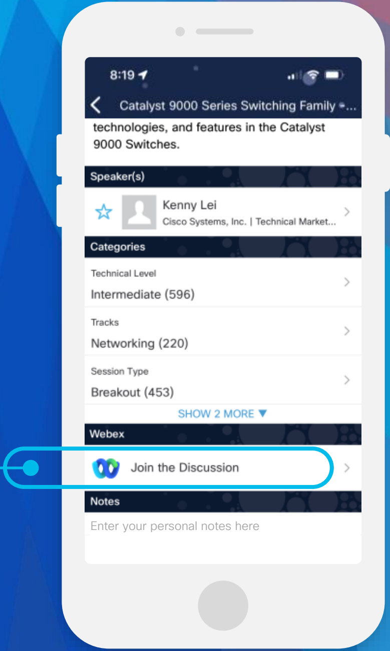
Questions?

Use Cisco Webex App to chat with the speaker after the session

How

1. Find this session in the Cisco Live Mobile App
2. Click "Join the Discussion"
3. Install the Webex App or go directly to the Webex space
4. Enter messages/questions in the Webex space

Webex spaces will be moderated by the speaker until June 9, 2023.


8:19

Catalyst 9000 Series Switching Family ...

technologies, and features in the Catalyst 9000 Switches.

Speaker(s)

Kenny Lei
Cisco Systems, Inc. | Technical Market...

Categories

Technical Level
Intermediate (596)

Tracks
Networking (220)

Session Type
Breakout (453)

SHOW 2 MORE ▼

Webex

Join the Discussion

Notes

Enter your personal notes here

https://ciscolive.ciscoevents.com/ciscolivebot/#BRKCRT-2013

Agenda

- Introduction
- Test Driven Development
- Cisco pyATS
- State Testing
- Configuration Testing
- A Sample TDA Workflow
- Conclusion

Introduction

John Capobianco

Cisco Learning and Certifications – Training Bootcamps

- 20+ years in IT
- Introduced to network automation at Cisco Live 2015
- Ansible (2015 – 2019)
  - Self-published “Automate Your Network” on Amazon (2019)
  - Please contact me on socials to get your PDF copy after this session
- pyATS / Python (2019 – present)

The “old way”

Some assumptions

- Design – Build – Test
- Manual processes
- Testing occurs at the end of the delivery cycle
- Testing is done against very large complex topologies
- Little to no instrumentation
- Reactive

Test Driven Development

Test Driven Development

Encouraging simple designs and inspiring confidence

- Software development process
- Convert requirements into test cases
- Three rules to TDD – You Are NOT Allowed To:
  1. Write any production code unless it is to make a failing unit test pass
  2. Write any more of a unit test than is sufficient to fail (and compilation failures are failures)
  3. Write any more production code than is sufficient to pass the one failing unit test.

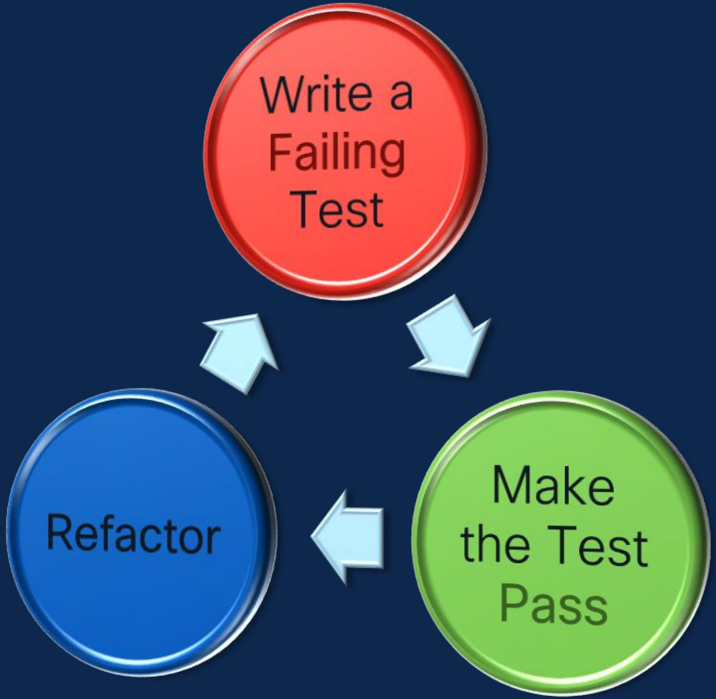
Test Driven Development

Important figures in TDD

- Kent Beck – credited with “rediscovering” the technique
  - Creator of “extreme programming”
  - One of the original 17 signatories on the “Agile Manifesto”

- Robert C. Martin (“Uncle Bob”)
  - Programmer, speaker, teacher from cleancoder.com
  - Provided the concise set of TDD rules

Test Driven Development



Test Driven Development

Important figures in TDD

1. Add a Test
2. Run all tests. The new test should fail.
3. Write the simplest code that passes the new test.
4. All tests should now pass.
5. Refactor as needed; repeating the testing cycle after each refactor to ensure refactoring quality.
6. Add a Test...

Test Driven Development

Best practices

- Keep the unit small
- General test structure
  - Setup, execution, validation, cleanup
- Always test a known state
- Limit, or eliminate, dependencies between tests
- Complex is fine, complicated is not
- “All-knowing” tests

Test Driven Development

Studies

- Engineering teams at Microsoft and IBM concluded
  - “pre-release defect density of four projects *decreased* between 40% and 90% relative to similar projects that did not use TDD practice”
    - Realizing-Quality-Improvement-Through-Test-Driven-Development-Results-and-Experiences-of-Four-Industrial-Teams-nagappan_tdd.pdf (microsoft.com)

Test Driven Development

Studies

- Department of Computer Science at North Carolina State University experiment
  - “92% of developers believed that TDD yields higher quality code, 79% thought it promoted simpler design, and 71% thought the approach was noticeably effective”
  - “56% of the professional developers believed it was difficult to get into a TDD mindset, while 23% claim the lack of upfront design phase is the reason for this difficulty. 40% believed the adoption of TDD was difficult.”
    - An Initial Investigation of Test Driven Development in Industry

Test Driven Development

Applying to Network Automation

- Direct mapping of business requirements of the network to test cases
- Easily applied in practice to network automation
- Network state and network configurations can be tested
- Intent can be enforced
- CI/CD pipelines can be established

Cisco pyATS

pyATS
print(network.profile)

Business Logic

Integration
- XPRESSO, Ansible, RobotFramework
- Jenkins, CI/CD pipelines, CLI, other tooling, etc

SDK & Library

Genie Libs
- Parsers, Feature/Protocol Models
- Reusable Testcases: Triggers, Verifications

Genie Library Framework
- Basis for agnostic automation libraries
- Boilerplate library foundation & engine

Toolbox

pyATS Core Test Infrastructure
- Topology & Test definition
- Execution & Reporting

“Purpose gives meaning to action in the same way that structure gives meaning to data

David Amerland

Intentional: How to Live, Love, Work and Play Meaningfully

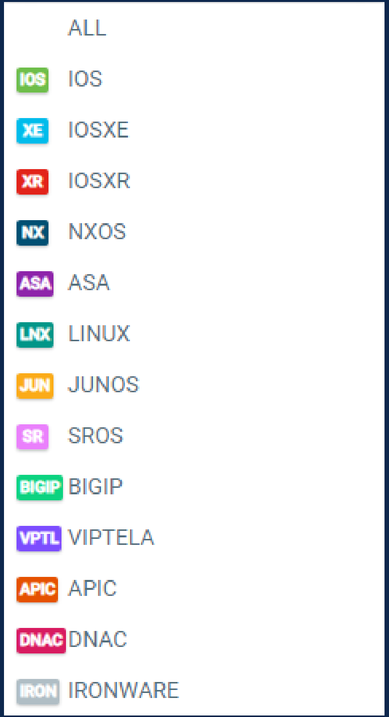
Cisco pyATS

A brief history

- Lesson One: How to pronounce pyATS
- Python Automated Test Systems (pyATS)
- Cisco runs over 3 million tests internally monthly with pyATS
- General public availability in 2017
- Officially endorsed by the Cisco executive team
- Suitable for Agile, rapid development iterations, and NetDevOps

pyATS Supported Platforms

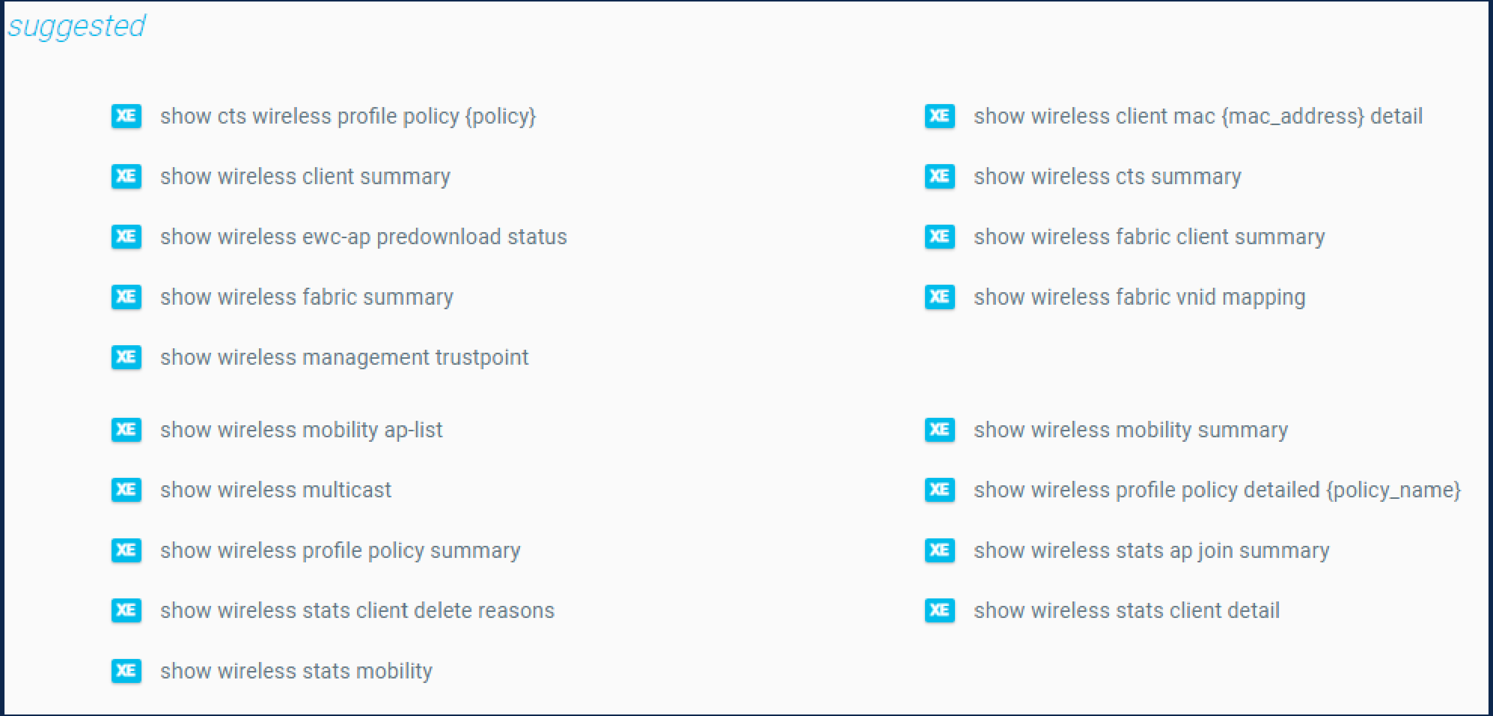
Not limited to Cisco



ALL
IOS IOS
XE IOSXE
XR IOSXR
NX NXOS
ASA ASA
LNX LINUX
JUN JUNOS
SR SROS
BIGIP BIGIP
VPTL VIPTELA
APIC APIC
DNAC DNAC
IRON IRONWARE

Wireless – WLC running IOS-XE

All standard IOS-XE parsers + new wireless parsers

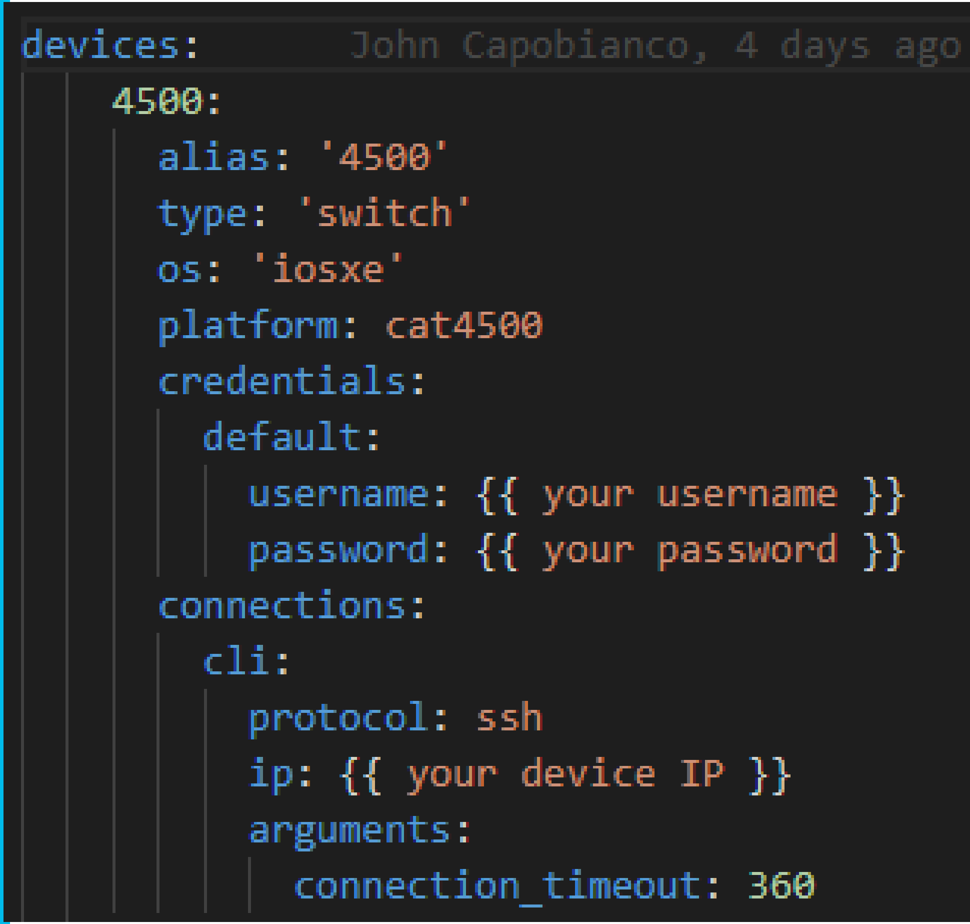


suggested

- XE show cts wireless profile policy {policy}
- XE show wireless client mac {mac_address} detail
- XE show wireless client summary
- XE show wireless cts summary
- XE show wireless ewc-ap predownload status
- XE show wireless fabric client summary
- XE show wireless fabric summary
- XE show wireless fabric vnid mapping
- XE show wireless management trustpoint
- XE show wireless mobility ap-list
- XE show wireless mobility summary
- XE show wireless multicast
- XE show wireless profile policy detailed {policy_name}
- XE show wireless profile policy summary
- XE show wireless stats ap join summary
- XE show wireless stats client delete reasons
- XE show wireless stats client detail
- XE show wireless stats mobility

Testbed

- Used to establish connectivity to network devices
- Extensible for intent-based configurations
- Static
  - YAML file
- Dynamic



```
devices:        John Capobianco, 4 days ago
  4500:
    alias: '4500'
    type: 'switch'
    os: 'iosxe'
    platform: cat4500
    credentials:
      default:
        username: {{ your username }}
        password: {{ your password }}
    connections:
      cli:
        protocol: ssh
        ip: {{ your device IP }}
        arguments:
          connection_timeout: 360
```

Testbed
Secret Strings

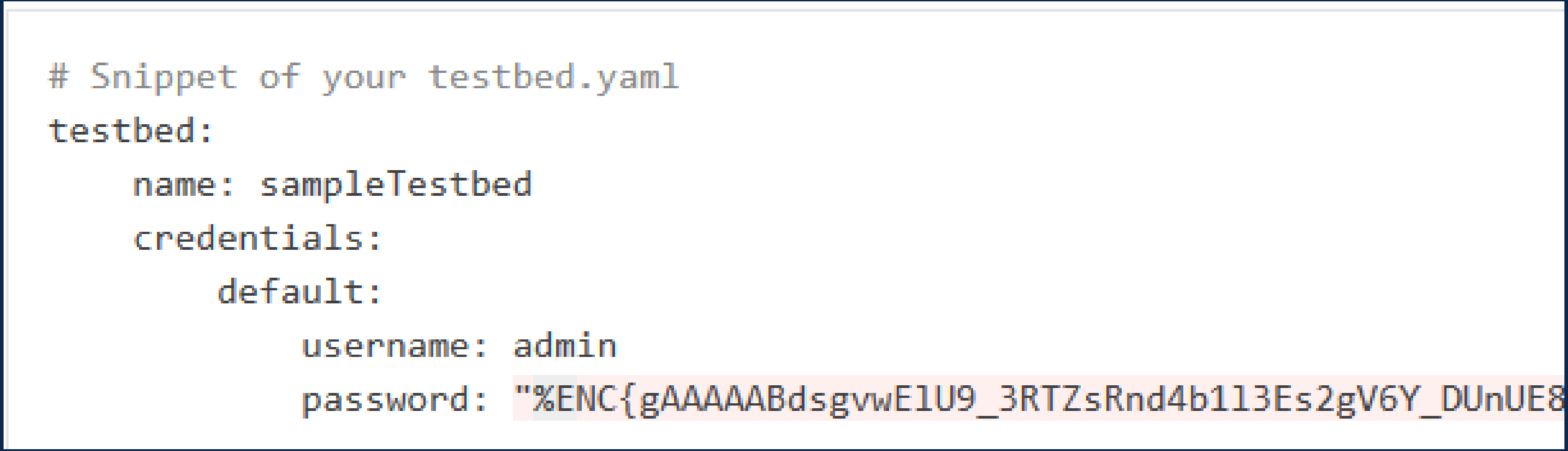
- Secret Strings can be used to encrypt entire testbed files or, more commonly, individual values such as the password.
  - Secret Strings – pyATS Documentation (devnetcloud.com)

- Follow the 8 steps to encrypt your password

- Represent your password as "%ENC{}" inside your testbed placing the encrypted string inside the curly braces

Testbed Example

Secret String used to encrypt password



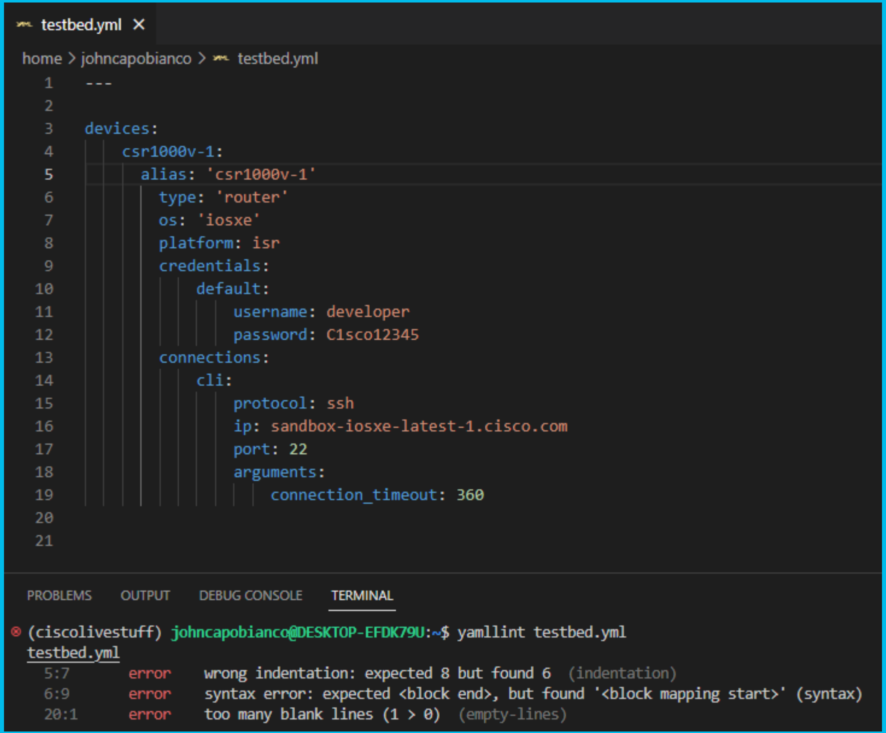
```
# Snippet of your testbed.yaml
testbed:
    name: sampleTestbed
    credentials:
        default:
            username: admin
            password: "%ENC{gAAAAABdsgvwElU9_3RTZsRnd4b1l3Es2gV6Y_DUnUE8
```

Testbed

Validation with linting

- yamllint – Python package used to validate YAML files
- pyATS validate testbed – built-it testbed validation command

yamllint



testbed.yml ×

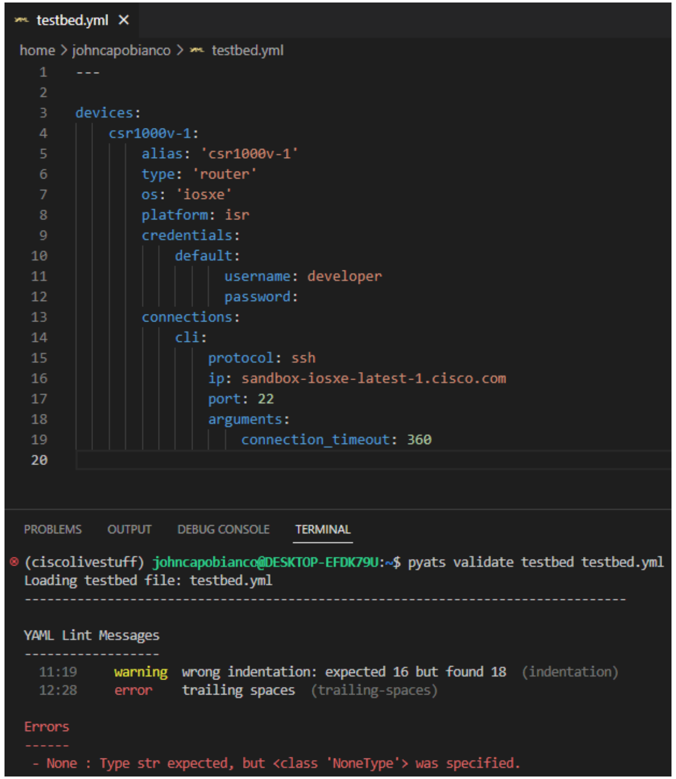
home > johncapobianco > testbed.yml

```yaml
---

devices:
    csr1000v-1:
      alias: 'csr1000v-1'
        type: 'router'
        os: 'iosxe'
        platform: isr
        credentials:
            default:
                username: developer
                password: C1sco12345
        connections:
            cli:
                protocol: ssh
                ip: sandbox-iosxe-latest-1.cisco.com
                port: 22
                arguments:
                    connection_timeout: 360


```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
(ciscolivestuff) johncapobianco@DESKTOP-EFDK79U:~$ yamllint testbed.yml
testbed.yml
  5:7       error    wrong indentation: expected 8 but found 6  (indentation)
  6:9       error    syntax error: expected <block end>, but found '<block mapping start>' (syntax)
  20:1      error    too many blank lines (1 > 0)  (empty-lines)
```

pyats validate



testbed.yml ×

home > johncapobianco > testbed.yml

```yaml
---

devices:
    csr1000v-1:
        alias: 'csr1000v-1'
        type: 'router'
        os: 'iosxe'
        platform: isr
        credentials:
            default:
                  username: developer
                  password: 
        connections:
            cli:
                protocol: ssh
                ip: sandbox-iosxe-latest-1.cisco.com
                port: 22
                arguments:
                    connection_timeout: 360

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
(ciscolivestuff) johncapobianco@DESKTOP-EFDK79U:~$ pyats validate testbed testbed.yml
Loading testbed file: testbed.yml
--------------------------------------------------------------------------------

YAML Lint Messages
------------------
  11:19     warning  wrong indentation: expected 16 but found 18  (indentation)
  12:28     error    trailing spaces  (trailing-spaces)

Errors
------
 - None : Type str expected, but <class 'NoneType'> was specified.
```

pyATS Command Line Interface (CLI)

Start using pyATS immeditely

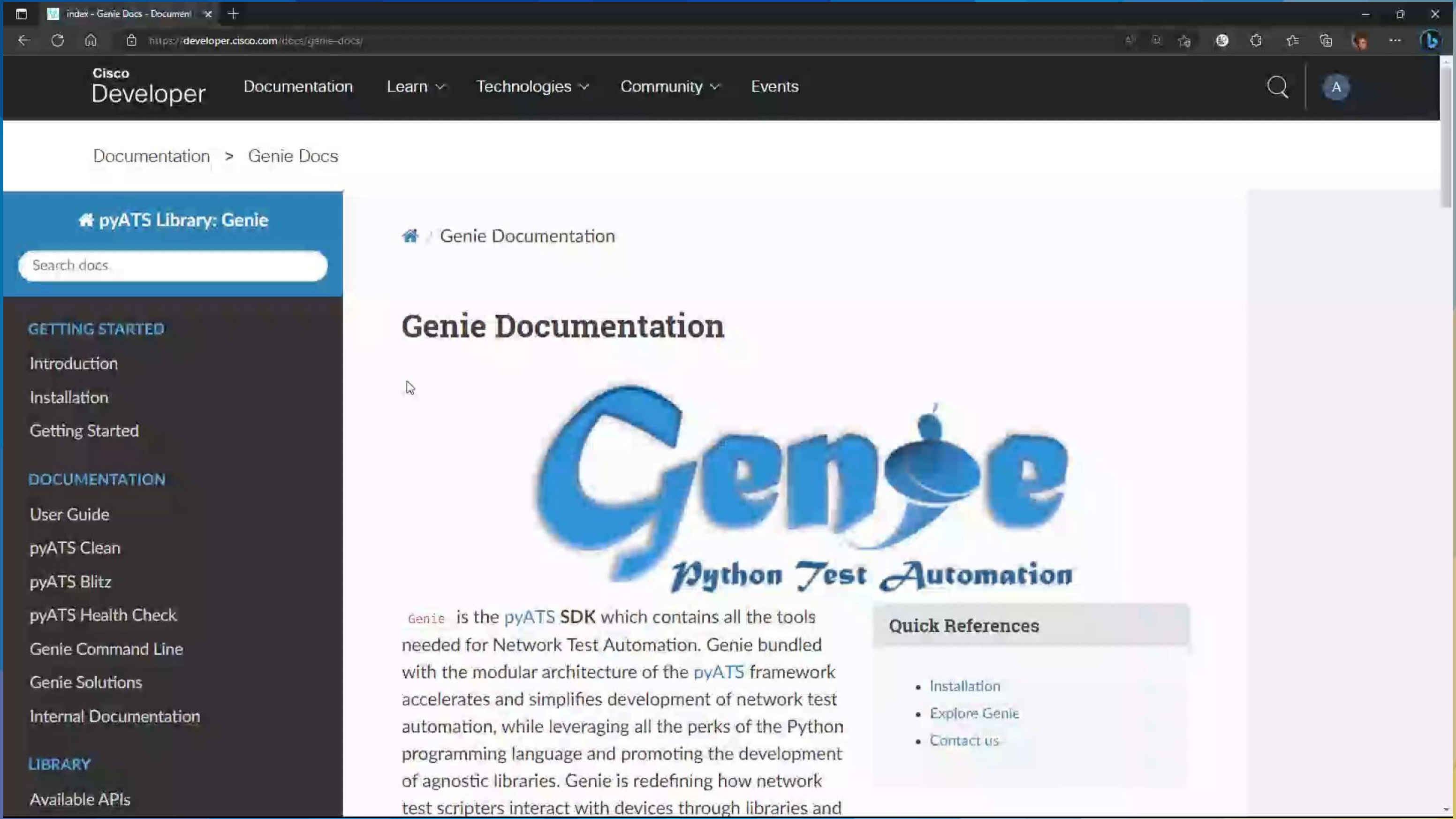
- pyATS has a CLI !
- All you need is valid testbed file and either the parser or model you want to transform into structure JavaScript Object Notation (JSON)
- Python-free
- Faster and more efficient than logging into a device and running show commands
- Directly from IDE like VS Code

Parsers

- Parse show commands into JSON
- Platform specific
- Thousands available
- Online searchable library of parsers

Models

- Learn command
- Platform agnostic
- 32 available commands
- “Learn” everything about a specific networking object



index - Genie Docs - Document

https://developer.cisco.com/docs/genie-docs/

Cisco Developer Documentation Learn Technologies Community Events

Documentation > Genie Docs

pyATS Library: Genie

Search docs

GETTING STARTED
- Introduction
- Installation
- Getting Started

DOCUMENTATION
- User Guide
- pyATS Clean
- pyATS Blitz
- pyATS Health Check
- Genie Command Line
- Genie Solutions
- Internal Documentation

LIBRARY
- Available APIs

Genie Documentation

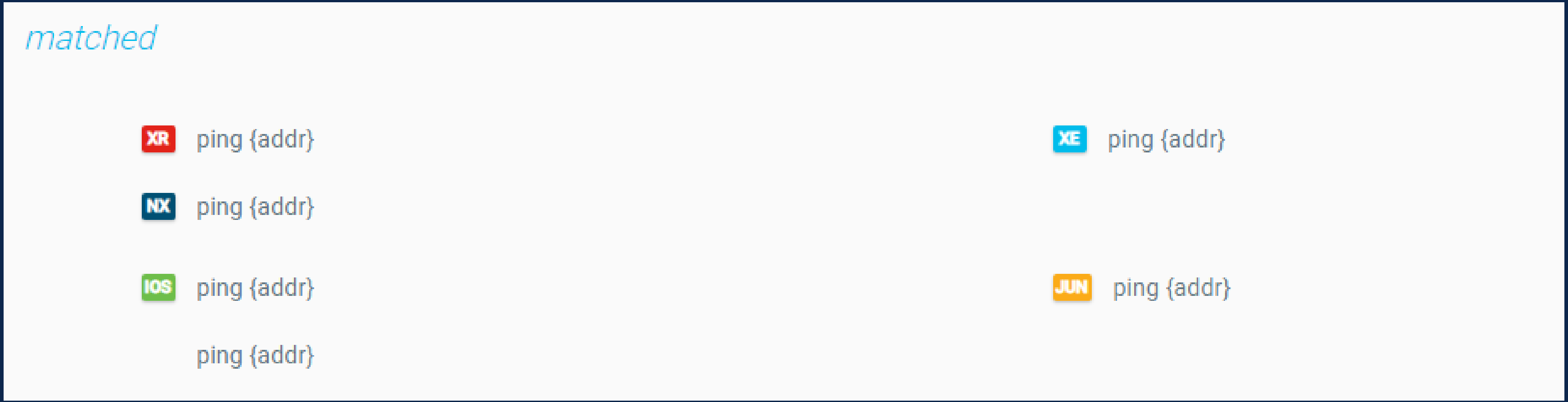
Genie Documentation

`Genie` is the pyATS **SDK** which contains all the tools needed for Network Test Automation. Genie bundled with the modular architecture of the pyATS framework accelerates and simplifies development of network test automation, while leveraging all the perks of the Python programming language and promoting the development of agnostic libraries. Genie is redefining how network test scripters interact with devices through libraries and

Quick References
- Installation
- Explore Genie
- Contact us

pyATS Ping

Test reachability and more



matched

XR ping {addr}

NX ping {addr}

IOS ping {addr}

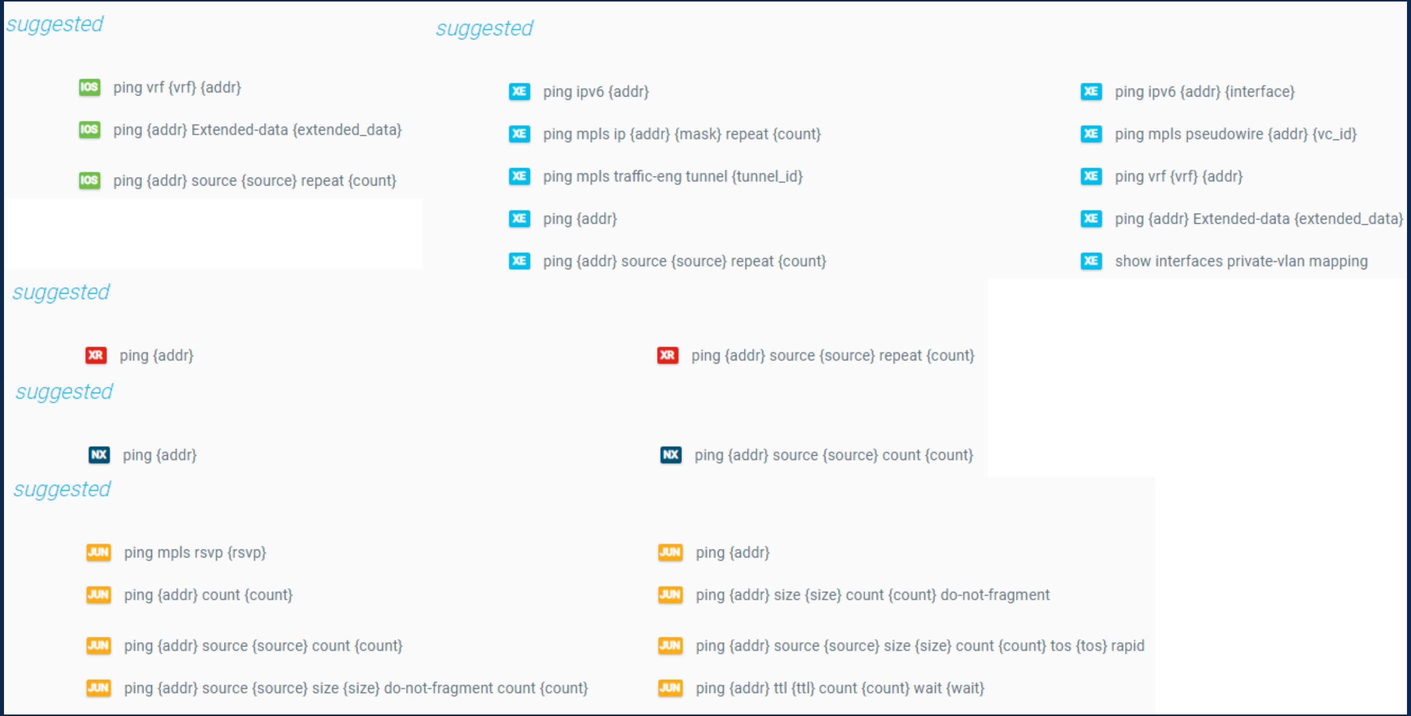
ping {addr}

XE ping {addr}

JUN ping {addr}

pyATS Advanced Ping

Unlimited potential



suggested

IOS ping vrf {vrf} {addr}

IOS ping {addr} Extended-data {extended_data}

IOS ping {addr} source {source} repeat {count}

suggested

XE ping ipv6 {addr}

XE ping mpls ip {addr} {mask} repeat {count}

XE ping mpls traffic-eng tunnel {tunnel_id}

XE ping {addr}

XE ping {addr} source {source} repeat {count}

XE ping ipv6 {addr} {interface}

XE ping mpls pseudowire {addr} {vc_id}

XE ping vrf {vrf} {addr}

XE ping {addr} Extended-data {extended_data}

XE show interfaces private-vlan mapping

suggested

XR ping {addr}

XR ping {addr} source {source} repeat {count}

suggested

NX ping {addr}

NX ping {addr} source {source} count {count}

suggested

JUN ping mpls rsvp {rsvp}

JUN ping {addr} count {count}

JUN ping {addr} source {source} count {count}

JUN ping {addr} source {source} size {size} do-not-fragment count {count}

JUN ping {addr}

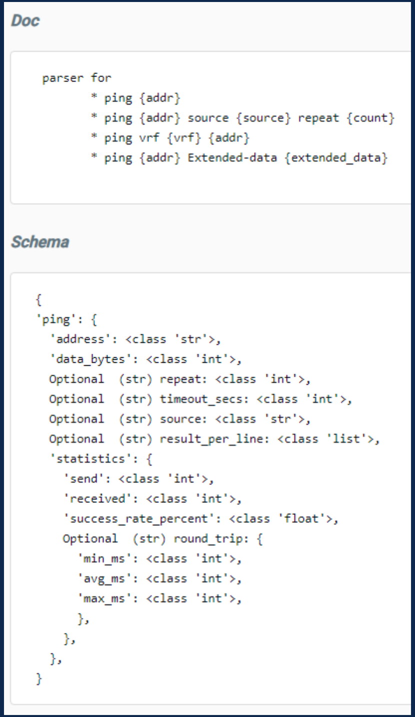
JUN ping {addr} size {size} count {count} do-not-fragment

JUN ping {addr} source {source} size {size} count {count} tos {tos} rapid

JUN ping {addr} ttl {ttl} count {count} wait {wait}

pyATS Ping Schema (IOS-XE)

Testable keys and values



Doc

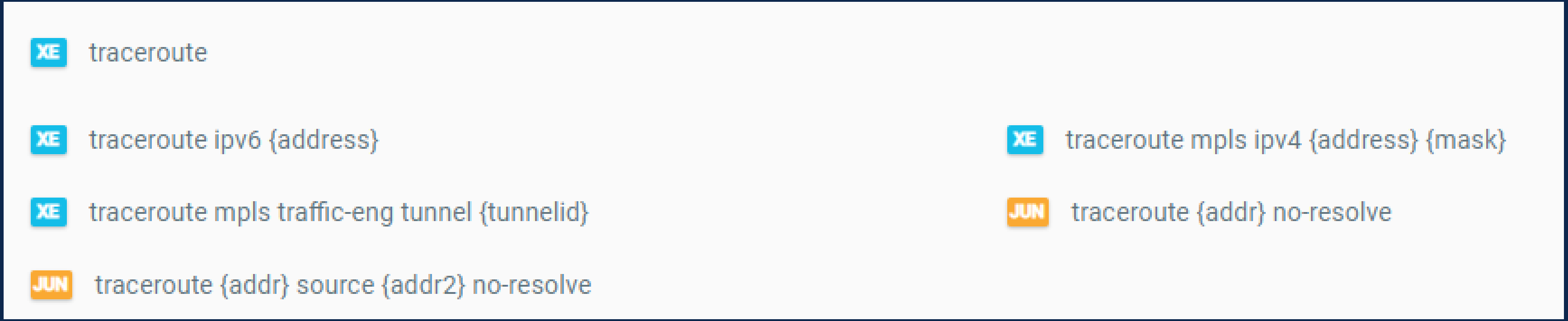
```
  parser for
         * ping {addr}
         * ping {addr} source {source} repeat {count}
         * ping vrf {vrf} {addr}
         * ping {addr} Extended-data {extended_data}
```

Schema

```
{
'ping': {
  'address': <class 'str'>,
  'data_bytes': <class 'int'>,
  Optional  (str) repeat: <class 'int'>,
  Optional  (str) timeout_secs: <class 'int'>,
  Optional  (str) source: <class 'str'>,
  Optional  (str) result_per_line: <class 'list'>,
  'statistics': {
    'send': <class 'int'>,
    'received': <class 'int'>,
    'success_rate_percent': <class 'float'>,
    Optional  (str) round_trip: {
      'min_ms': <class 'int'>,
      'avg_ms': <class 'int'>,
      'max_ms': <class 'int'>,
      },
    },
  },
}
```

pyATS Traceroute

Test traffic paths and flows



XE traceroute

XE traceroute ipv6 {address}

XE traceroute mpls traffic-eng tunnel {tunnelid}

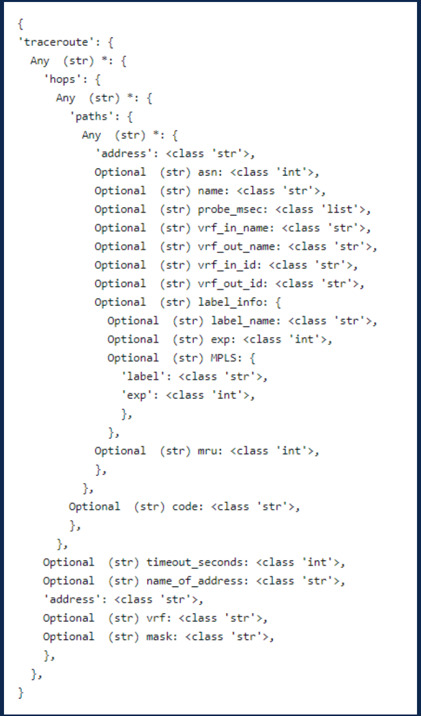
JUN traceroute {addr} source {addr2} no-resolve

XE traceroute mpls ipv4 {address} {mask}

JUN traceroute {addr} no-resolve

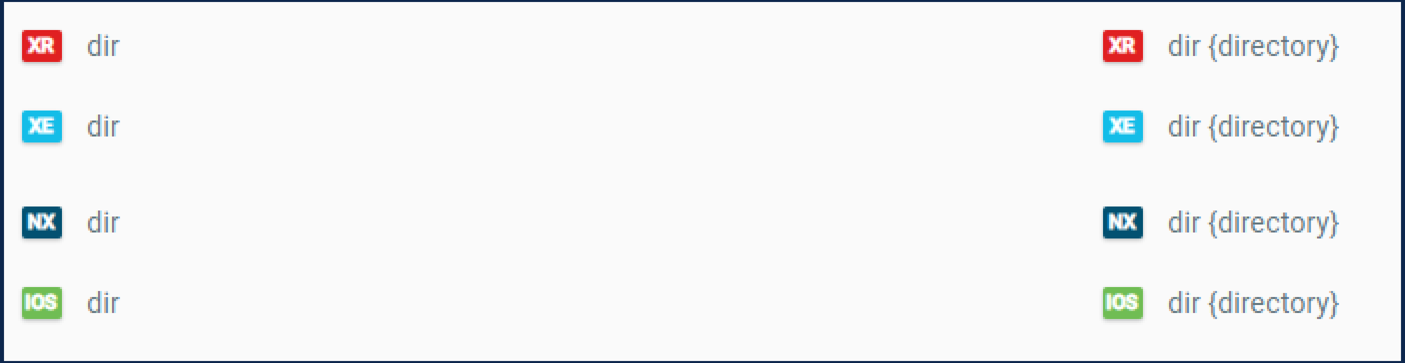
pyATS Traceroute

Test traffic paths and flows



```
{
'traceroute': {
  Any  (str) *: {
    'hops': {
      Any  (str) *: {
        'paths': {
          Any  (str) *: {
            'address': <class 'str'>,
            Optional  (str) asn: <class 'int'>,
            Optional  (str) name: <class 'str'>,
            Optional  (str) probe_msec: <class 'list'>,
            Optional  (str) vrf_in_name: <class 'str'>,
            Optional  (str) vrf_out_name: <class 'str'>,
            Optional  (str) vrf_in_id: <class 'str'>,
            Optional  (str) vrf_out_id: <class 'str'>,
            Optional  (str) label_info: {
              Optional  (str) label_name: <class 'str'>,
              Optional  (str) exp: <class 'int'>,
              Optional  (str) MPLS: {
                'label': <class 'str'>,
                'exp': <class 'int'>,
                },
              },
            Optional  (str) mru: <class 'int'>,
            },
          },
        Optional  (str) code: <class 'str'>,
        },
      },
    Optional  (str) timeout_seconds: <class 'int'>,
    Optional  (str) name_of_address: <class 'str'>,
    'address': <class 'str'>,
    Optional  (str) vrf: <class 'str'>,
    Optional  (str) mask: <class 'str'>,
    },
  },
}
```

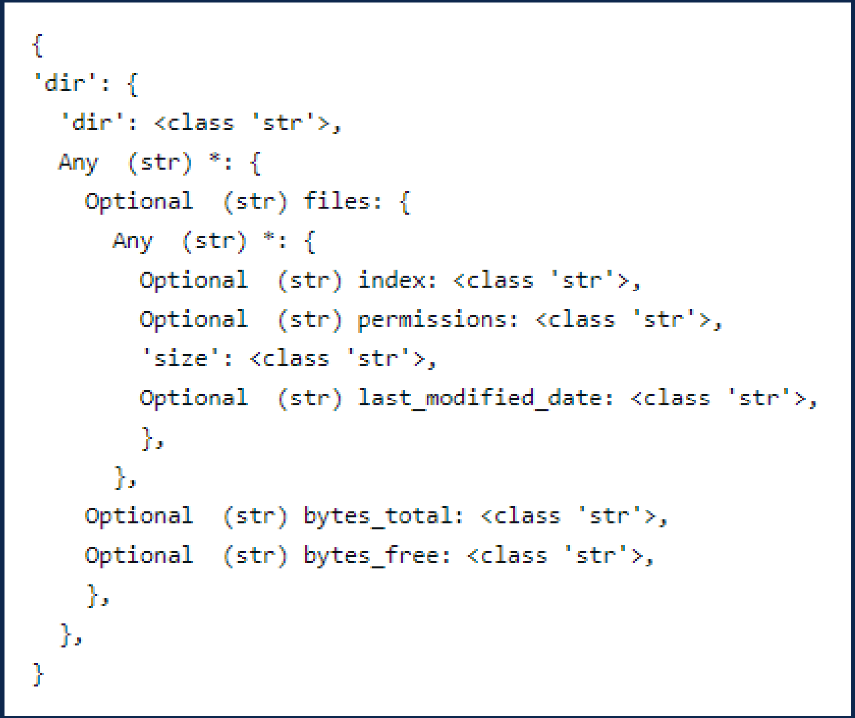
pyATS Dir
Test directories



XR	dir	XR	dir {directory}
XE	dir	XE	dir {directory}
NX	dir	NX	dir {directory}
IOS	dir	IOS	dir {directory}

pyATS Dir Schema (IOS-XE

Testable keys and values



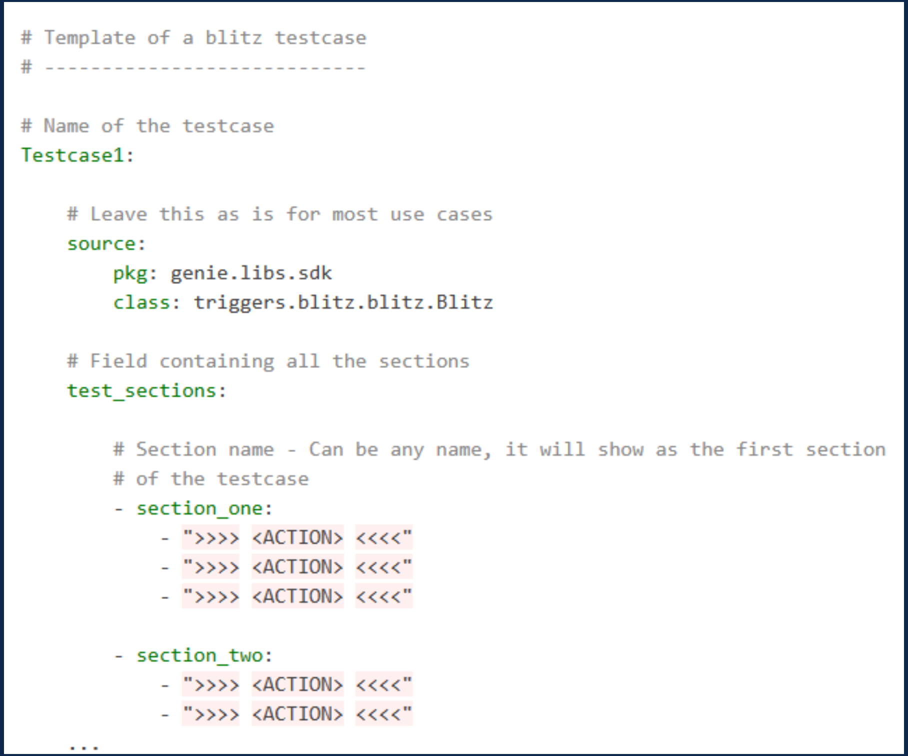
```
{
'dir': {
  'dir': <class 'str'>,
  Any  (str) *: {
    Optional  (str) files: {
      Any  (str) *: {
        Optional  (str) index: <class 'str'>,
        Optional  (str) permissions: <class 'str'>,
        'size': <class 'str'>,
        Optional  (str) last_modified_date: <class 'str'>,
        },
      },
    Optional  (str) bytes_total: <class 'str'>,
    Optional  (str) bytes_free: <class 'str'>,
    },
  },
}
```

pyATS Blitz

Python-free, YAML-based, quick triggers

- pyATS has a Python-free implementation known as Quick Triggers or Blitz
- Structured text (YAML) defines configuration and verification tasks
- Rapid adoption of automation
- No programming experience or expertise required
- Nice on-ramp from Ansible

pyATS Blitz



```
# Template of a blitz testcase
# ----------------------------

# Name of the testcase
Testcase1:

    # Leave this as is for most use cases
    source:
        pkg: genie.libs.sdk
        class: triggers.blitz.blitz.Blitz

    # Field containing all the sections
    test_sections:

        # Section name - Can be any name, it will show as the first section
        # of the testcase
        - section_one:
            - ">>>> <ACTION> <<<<"
            - ">>>> <ACTION> <<<<"
            - ">>>> <ACTION> <<<<"

        - section_two:
            - ">>>> <ACTION> <<<<"
            - ">>>> <ACTION> <<<<"
    ...
```

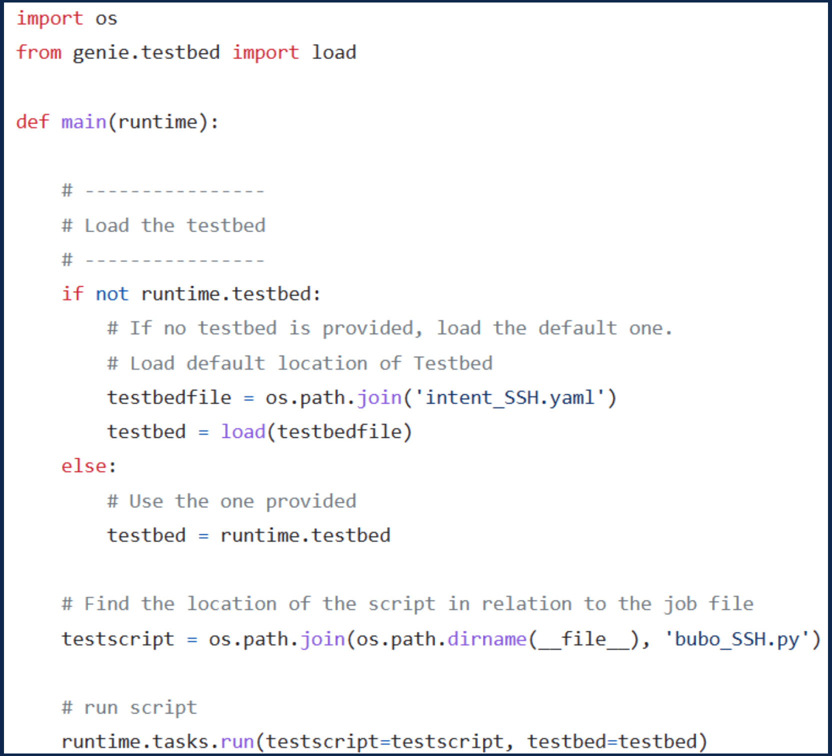
pyATS jobs

Python implementation

- pyATS jobs are made up of two files:
  - A python script with the pyATS logic
  - A job file (also Python)

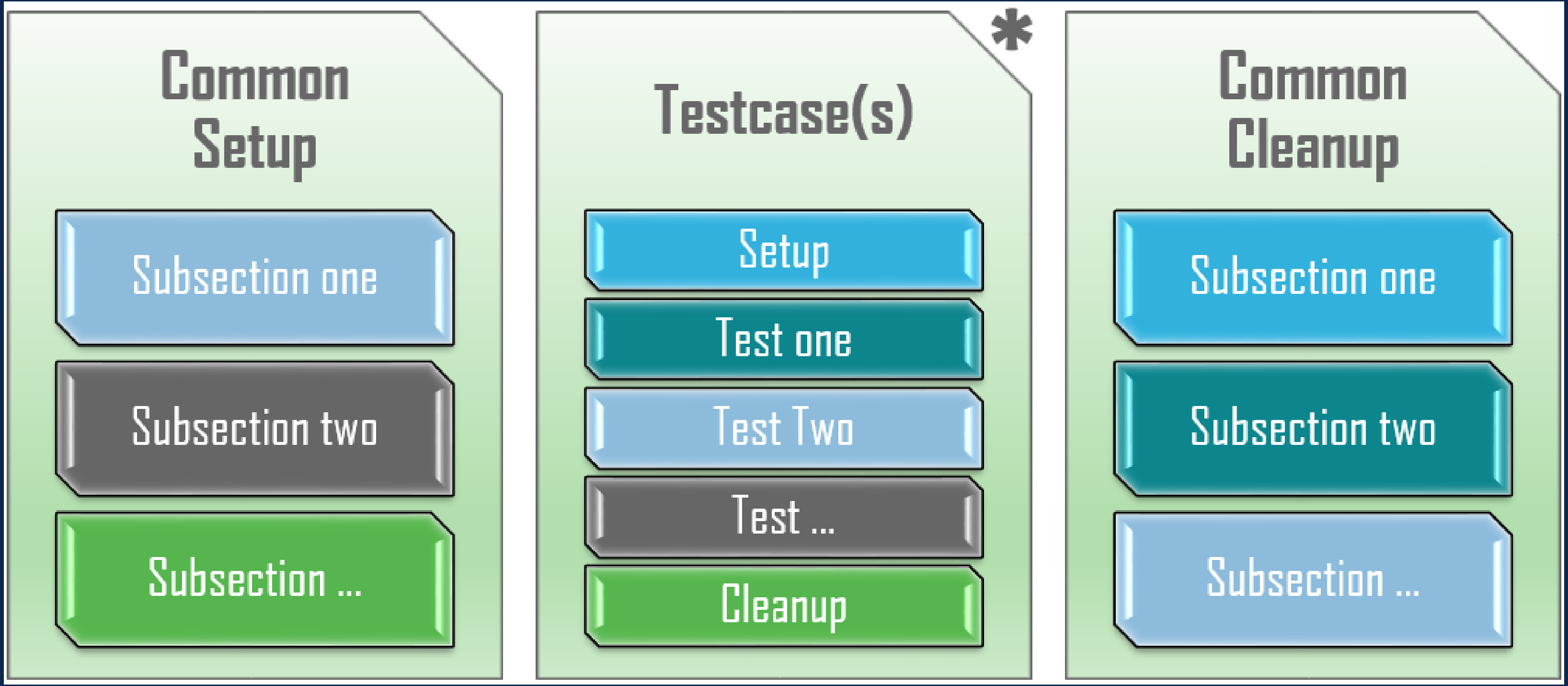
- The job file abstracts and assists in loading the testbed file as well as establishing a connection between the Python script and the pyATS framework

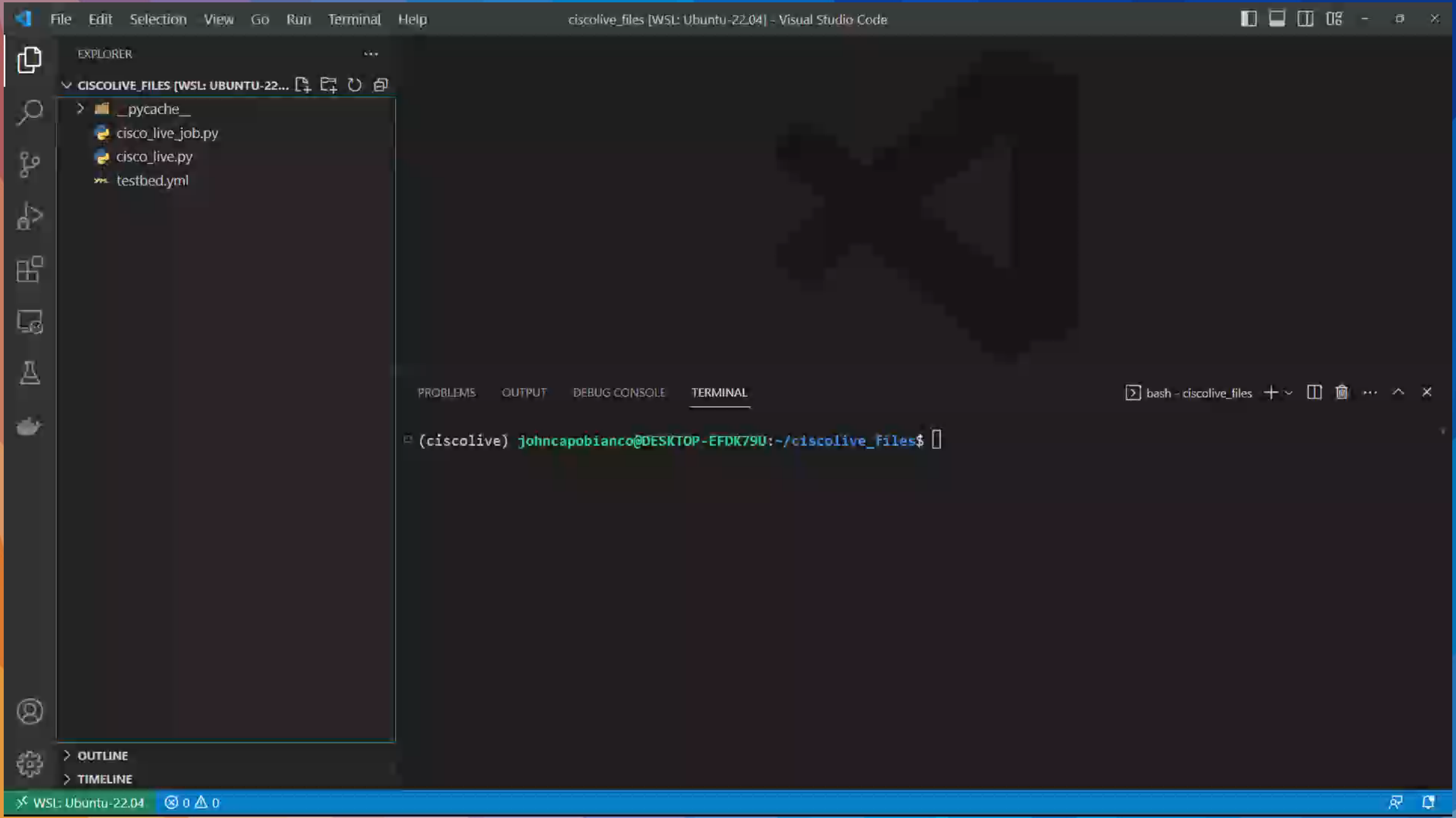
pyATS job file



```python
import os
from genie.testbed import load

def main(runtime):

    # ----------------
    # Load the testbed
    # ----------------
    if not runtime.testbed:
        # If no testbed is provided, load the default one.
        # Load default location of Testbed
        testbedfile = os.path.join('intent_SSH.yaml')
        testbed = load(testbedfile)
    else:
        # Use the one provided
        testbed = runtime.testbed

    # Find the location of the script in relation to the job file
    testscript = os.path.join(os.path.dirname(__file__), 'bubo_SSH.py')

    # run script
    runtime.tasks.run(testscript=testscript, testbed=testbed)
```

pyATS scripts



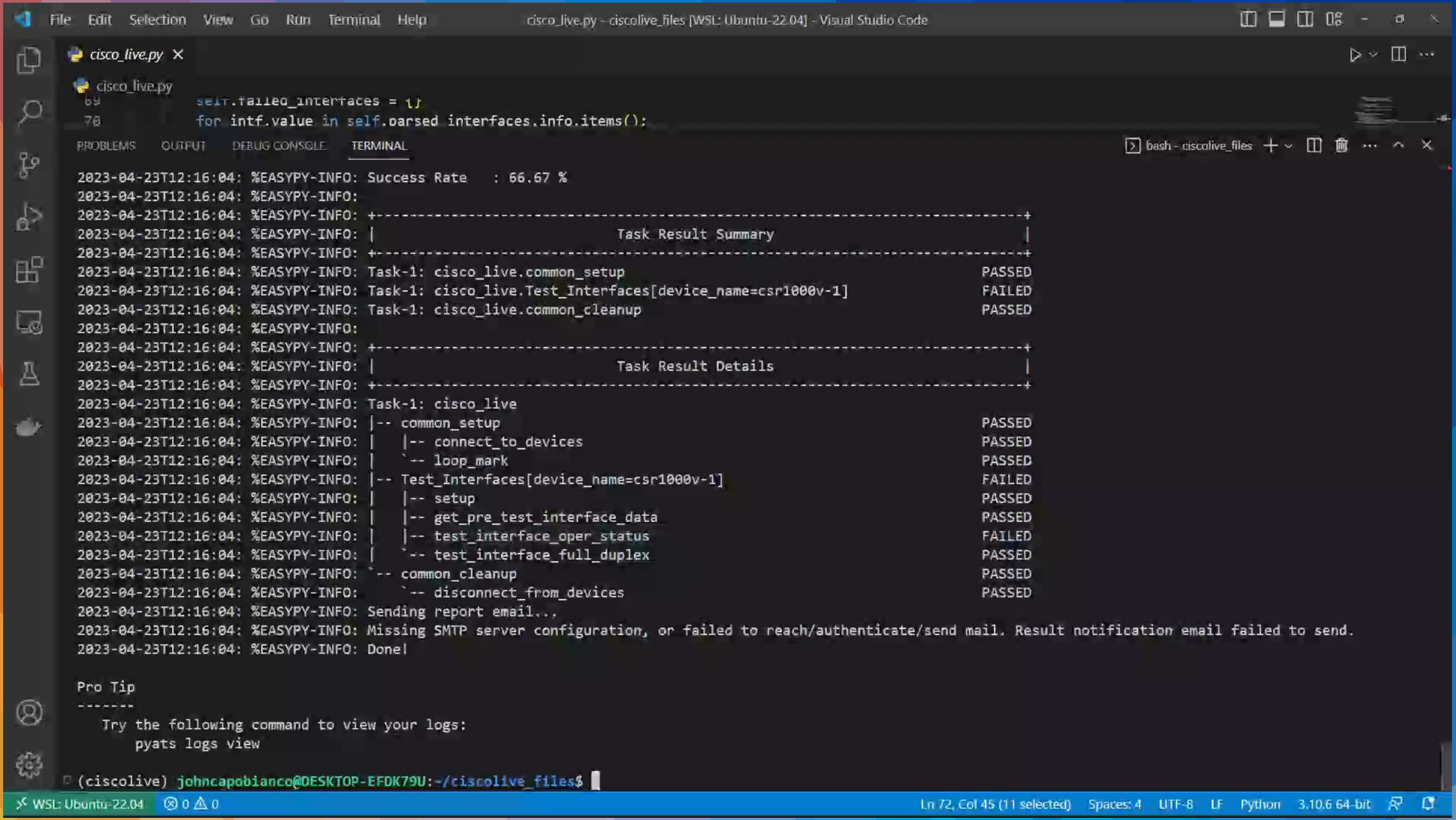


```
(ciscolive) johncapobianco@DESKTOP-EFDK79U:~/ciscolive_files$
```

pyATS Log Viewer

Built-in Enriched HTML Log Viewer

- At the end of a pyATS job you can launch an interactive enriched HTML page to review your logs beyond the CLI recap
- Big advantage over other network automation frameworks
- Historical view of local jobs


File Edit Selection View Go Run Terminal Help
cisco_live.py - ciscolive_files [WSL: Ubuntu-22.04] - Visual Studio Code
cisco_live.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
bash - ciscolive_files
WSL: Ubuntu-22.04
Ln 72, Col 45 (11 selected) Spaces: 4 UTF-8 LF Python 3.10.6 64-bit

```
        self.failed_interfaces = {}
        for intf.value in self.parsed interfaces.info.items():
```

```
2023-04-23T12:16:04: %EASYPY-INFO: Success Rate   : 66.67 %
2023-04-23T12:16:04: %EASYPY-INFO:
2023-04-23T12:16:04: %EASYPY-INFO: +------------------------------------------------------------------------------+
2023-04-23T12:16:04: %EASYPY-INFO: |                             Task Result Summary                              |
2023-04-23T12:16:04: %EASYPY-INFO: +------------------------------------------------------------------------------+
2023-04-23T12:16:04: %EASYPY-INFO: Task-1: cisco_live.common_setup                                        PASSED
2023-04-23T12:16:04: %EASYPY-INFO: Task-1: cisco_live.Test_Interfaces[device_name=csr1000v-1]             FAILED
2023-04-23T12:16:04: %EASYPY-INFO: Task-1: cisco_live.common_cleanup                                      PASSED
2023-04-23T12:16:04: %EASYPY-INFO:
2023-04-23T12:16:04: %EASYPY-INFO: +------------------------------------------------------------------------------+
2023-04-23T12:16:04: %EASYPY-INFO: |                             Task Result Details                              |
2023-04-23T12:16:04: %EASYPY-INFO: +------------------------------------------------------------------------------+
2023-04-23T12:16:04: %EASYPY-INFO: Task-1: cisco_live
2023-04-23T12:16:04: %EASYPY-INFO: |-- common_setup                                                         PASSED
2023-04-23T12:16:04: %EASYPY-INFO: |   |-- connect_to_devices                                             PASSED
2023-04-23T12:16:04: %EASYPY-INFO: |   `-- loop_mark                                                      PASSED
2023-04-23T12:16:04: %EASYPY-INFO: |-- Test_Interfaces[device_name=csr1000v-1]                             FAILED
2023-04-23T12:16:04: %EASYPY-INFO: |   |-- setup                                                          PASSED
2023-04-23T12:16:04: %EASYPY-INFO: |   |-- get_pre_test_interface_data                                    PASSED
2023-04-23T12:16:04: %EASYPY-INFO: |   |-- test_interface_oper_status                                     FAILED
2023-04-23T12:16:04: %EASYPY-INFO: |   `-- test_interface_full_duplex                                     PASSED
2023-04-23T12:16:04: %EASYPY-INFO: `-- common_cleanup                                                     PASSED
2023-04-23T12:16:04: %EASYPY-INFO:     `-- disconnect_from_devices                                        PASSED
2023-04-23T12:16:04: %EASYPY-INFO: Sending report email...
2023-04-23T12:16:04: %EASYPY-INFO: Missing SMTP server configuration, or failed to reach/authenticate/send mail. Result notification email failed to send.
2023-04-23T12:16:04: %EASYPY-INFO: Done!

Pro Tip
-------
   Try the following command to view your logs:
       pyats logs view

(ciscolive) johncapobianco@DESKTOP-EFDK79U:~/ciscolive_files$
```

pyATS WebEx Integration

Built-in WebEx communication

- pyATS can send the log summary to WebEx dynamically

- Options that can be appended to pyATS job commands
  --webex-token – WebEx Bot token
  --webex-space – WebEx Space ID to send notification to
  --webex-email – Email of specific user to send notification to

pyATS – CLI vs API

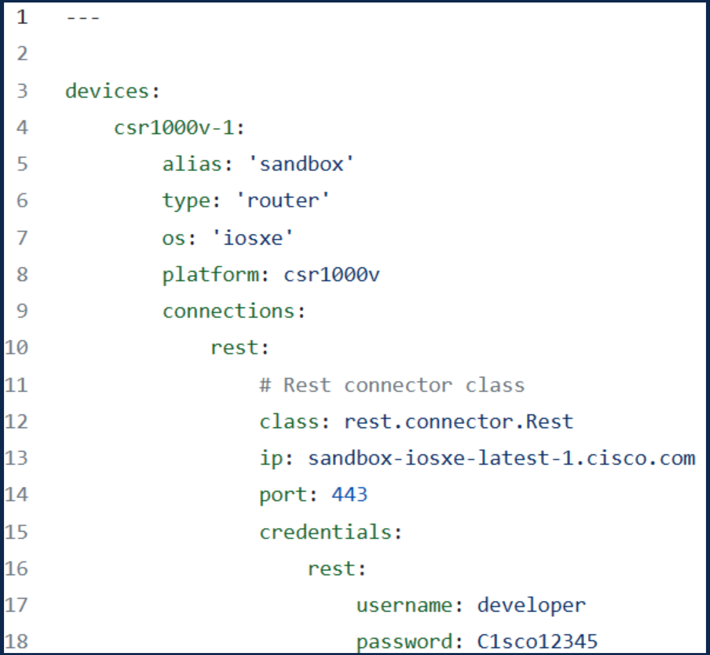
Other ways to connect

- In addition to SSH CLI based operation pyATS has several connectors that extend it’s capability to various APIs
  - REST Connector
    - NXOS
    - NSO
    - DNAC
    - IOS-XE RESTCONF
    - APIC
    - CML
    - BigIP
    - vManage
    - DCNM
    - Nexus Dashboard

pyATS – REST Connector

Testbed

- pip install rest.connector

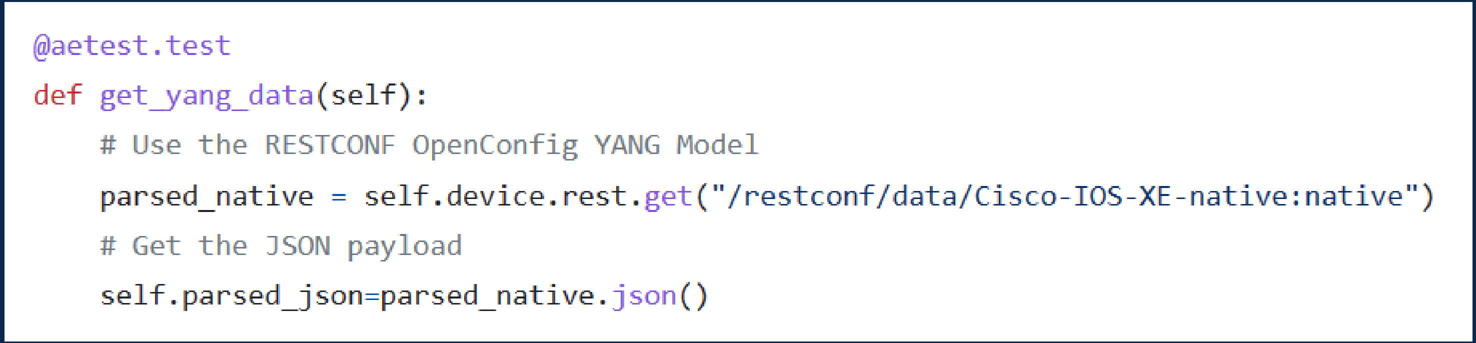


```yaml
---

devices:
    csr1000v-1:
        alias: 'sandbox'
        type: 'router'
        os: 'iosxe'
        platform: csr1000v
        connections:
            rest:
                # Rest connector class
                class: rest.connector.Rest
                ip: sandbox-iosxe-latest-1.cisco.com
                port: 443
                credentials:
                    rest:
                        username: developer
                        password: C1sco12345
```

pyATS – REST Connector

Job example

- pyATS job file is unchanged
- Use rest.<method>() to interact with the device's API



```
@aetest.test
def get_yang_data(self):
    # Use the RESTCONF OpenConfig YANG Model
    parsed_native = self.device.rest.get("/restconf/data/Cisco-IOS-XE-native:native")
    # Get the JSON payload
    self.parsed_json=parsed_native.json()
```

pyATS – Recording jobs

Simulate your testbed offline

- We can add --record <name of recording> to our job
  - Capture the state / config of the device during this job
- Use the following command to playback the recorded data
  - Python3 –m unicon.playback.mock --recorded-data <name of recording>

pyATS – Mock Devices

Simulate your testbed offline

- Add --output <path/mock_device.yml> to the recording to generate mock device from the output
- Inspect the mock device – it will have the entire state as YAML
- From the command-line interface you can connect to the "CLI" of the mock device
  - mock_device_cli -os <os> --mock_data_dir <dir> --state connect
- From there you can run CLI commands !

Enhancing your pyATS experience

Optional Python libraries

- Rich
  - Add tables to your pyATS logging output
  - Includes colours
  - Available in pyATS log viewer output
- Tabulate
  - Add tables to your pyATS logging output
  - Alternative to Rich

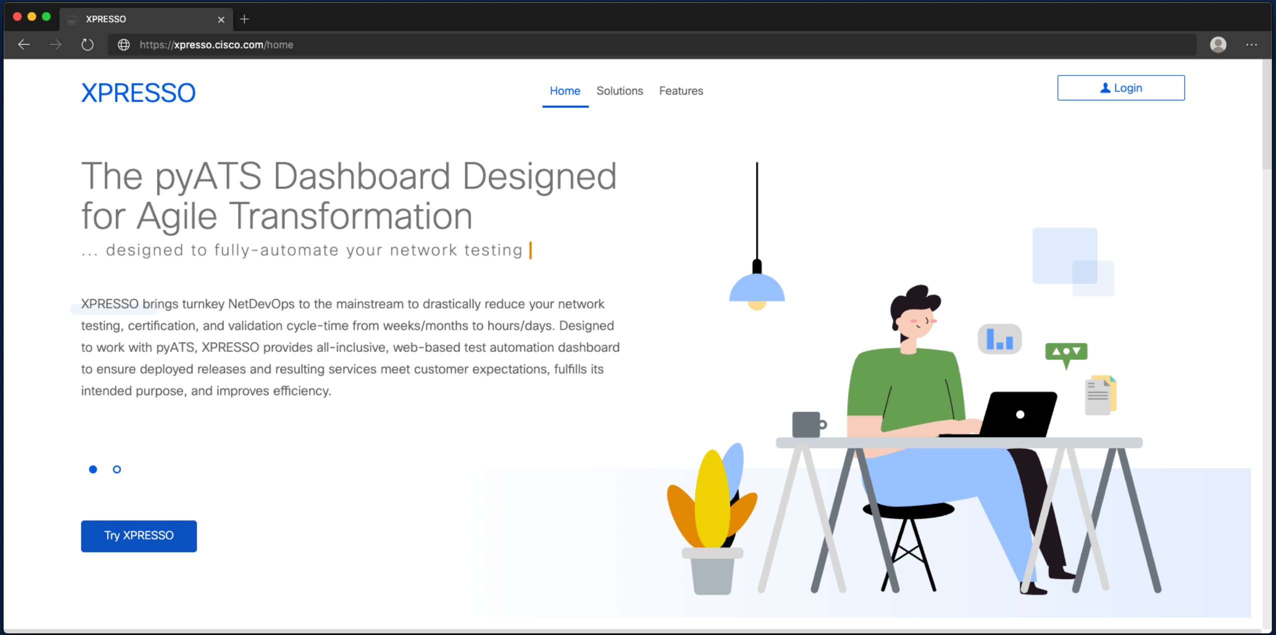
Enhancing your pyATS experience

Optional Python libraries

- Requests
  - Make arbitrary API calls to outside APIs when a test passes / fails
    - ServiceNow
    - WebEx
- Various Python SDKs
  - COBRA for ACI
  - Meraki SDK
  - DNA Center Python SDK

xPresso

A Dashboard for orchestrating and scheduling pyATS jobs



The pyATS Dashboard Designed for Agile Transformation

... designed to fully-automate your network testing |

XPRESSO brings turnkey NetDevOps to the mainstream to drastically reduce your network testing, certification, and validation cycle-time from weeks/months to hours/days. Designed to work with pyATS, XPRESSO provides all-inclusive, web-based test automation dashboard to ensure deployed releases and resulting services meet customer expectations, fulfills its intended purpose, and improves efficiency.

xPresso

A Dashboard for orchestrating and scheduling pyATS jobs

- Docker-based
- pyATS job-to-container conversion tool
- Schedule, orchestrate, monitor, reporting from pyATS jobs
- Integrates with Jenkins and other CI/CD
- DevNet Sandbox available
  - https://devnetsandbox.cisco.com/RM/Diagram/Index/756b58ba-15aa-4228-8a41-f94f684330e7?diagramType=Topology

State Testing

State Testing

Testing the operational state of the network

- Every test results in a Boolean pass or fail
- Using either parsed show commands or REST API JSON gather the state of the network
  - Interfaces
  - Routing
  - Neighbors
  - ACLs
  - You get the idea

State Testing

Testing the operational state of the network

- Primarily integers but could be strings
- Personal preferences:
  - I like to setup a threshold and test against it
  - == != => =< > <
  - Mathematical formulas (+ - / *)
  - Strings can also use “contains” or “in”, for example

More personal preferences:

- I like to setup an empty flag
  - test_failed = {}
- When a test fails I set this to a value
  - test_failed = “Failed”
- At the end of my testing I evaluate this flag to ultimately pass or fail the test

  -
    ```
    if test_failed:
      self.failed(“This test failed”)
    else
      self.passed(“This test passed!”
    ```
- I always use Rich tables and log.info() to include the tables in my pyATS log view logs.

Configuration Testing

Configuration Testing

Testing the configuration of devices

- Similar testing but using:
  - device.learn(“config”) (JSON)
  - device.parse(“show running-config”) (JSON)
  - RESTCONF API root (JSON)
  - device.execute(“show run”) (Raw CLI)

pyATS .configure()

Using pyATS to push configurations to a device

- pyATS .configure() can be used several ways to push configurations to a device
  - Static configurations
    - Single line – device.configure(“ntp server 192.168.1.1”)
    - Multiline – device.configure(```interface GigabitEthernet2
      ip address 172.16.100.100 255.255.255.0
      no shutdown ```)
  - Jinja2 templated configurations
    - Jinja2 included in the pyATS framework and easily incorporated

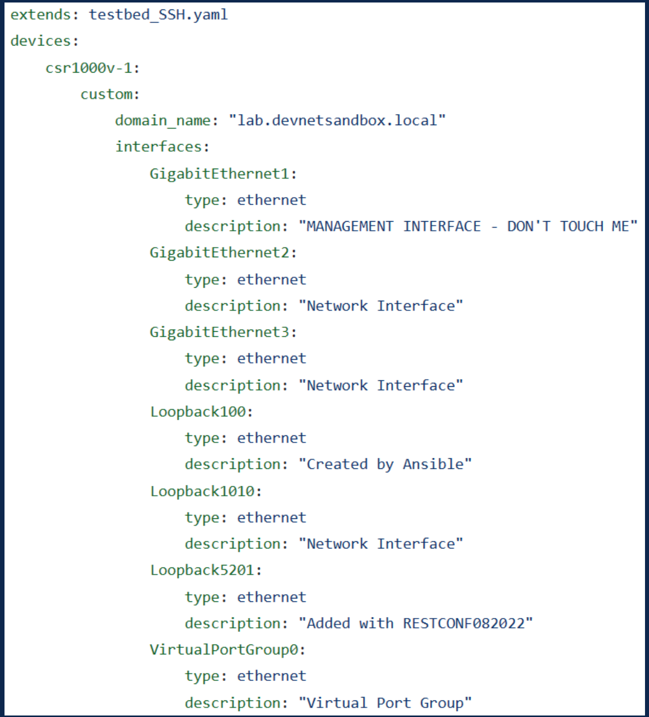
Intent-based Configuration Management

Extending our Testbed to represent Intent

- Our testbed YAML files can be extended to include intent
- Intent can then be tested
  - Compare intent-values in testbed against actual configuration or state
- Intent can be enforced
  - .configure() to push intent

Intent-based Configuration Management

Extending our Testbed to represent Intent

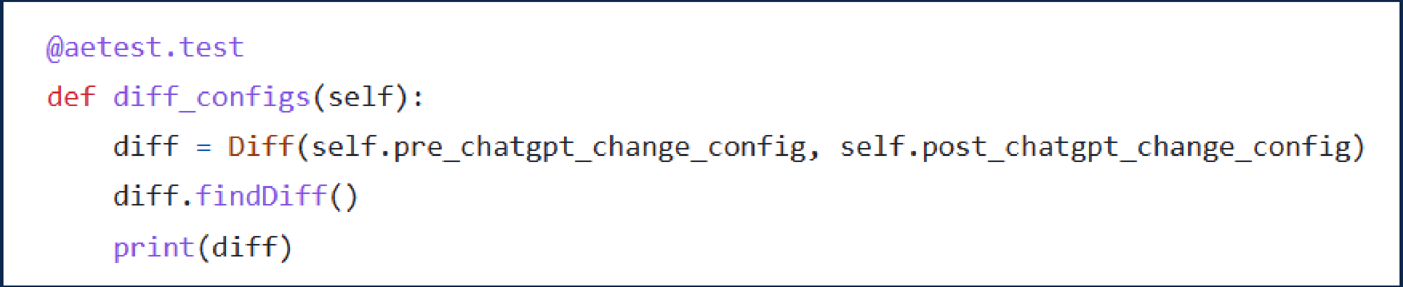


```yaml
extends: testbed_SSH.yaml
devices:
    csr1000v-1:
        custom:
            domain_name: "lab.devnetsandbox.local"
            interfaces:
                GigabitEthernet1:
                    type: ethernet
                    description: "MANAGEMENT INTERFACE - DON'T TOUCH ME"
                GigabitEthernet2:
                    type: ethernet
                    description: "Network Interface"
                GigabitEthernet3:
                    type: ethernet
                    description: "Network Interface"
                Loopback100:
                    type: ethernet
                    description: "Created by Ansible"
                Loopback1010:
                    type: ethernet
                    description: "Network Interface"
                Loopback5201:
                    type: ethernet
                    description: "Added with RESTCONF082022"
                VirtualPortGroup0:
                    type: ethernet
                    description: "Virtual Port Group"
```

Differentials

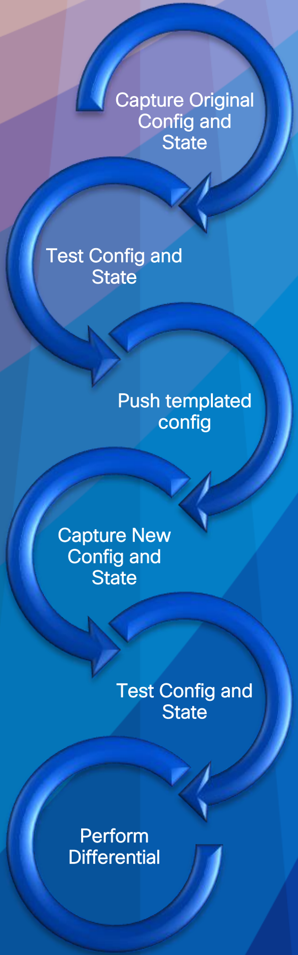
.diff()

- pyATS can perform differentials with Linux-style +/- additions and removals between datasets
- Where we capture with device.parse or device.learn pre and post change into variables then perform a diff against them



```
@aetest.test
def diff_configs(self):
    diff = Diff(self.pre_chatgpt_change_config, self.post_chatgpt_change_config)
    diff.findDiff()
    print(diff)
```

A Sample Test Driven Automation Workflow


Capture Original Config and State
Test Config and State
Push templated config
Capture New Config and State
Test Config and State
Perform Differential

A Sample Test Driven Automation Workflow

1. Capture current configuration and state
2. Test configuration and state
3. Push templated configurations incorporating intent
4. Capture new configuration and state
5. Test new configuration and state
6. Perform differentials
7. Optionally send reports or other 3rd party tools

Fill out your session surveys!

Attendees who fill out a minimum of four session surveys and the overall event survey will get **Cisco Live-branded socks** (while supplies last)!

Attendees will also earn 100 points in the **Cisco Live Challenge** for every survey completed.

These points help you get on the leaderboard and increase your chances of winning daily and grand prizes

Continue your education

- Visit the Cisco Showcase for related demos
- Book your one-on-one Meet the Engineer meeting
- Attend the interactive education with DevNet, Capture the Flag, and Walk-in Labs
- Visit the On-Demand Library for more sessions at www.CiscoLive.com/on-demand

The bridge to possible

Thank you

cisco Live!

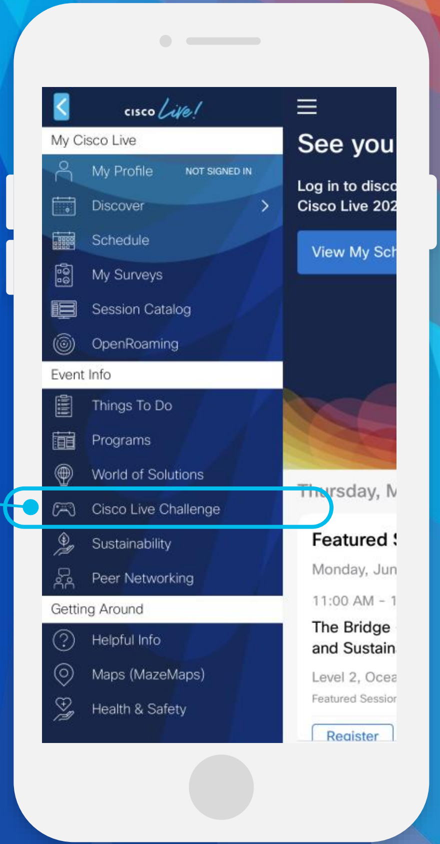
Let's go

Cisco Live **Challenge**

Gamify your Cisco Live experience!
Get points for attending this session!

How:

1. Open the Cisco Events App.
2. Click on 'Cisco Live Challenge' in the side menu.
3. Click on View Your Badges at the top.
4. Click the + at the bottom of the screen and scan the QR code:


cisco Live!
My Cisco Live
My Profile
NOT SIGNED IN
Discover
Schedule
My Surveys
Session Catalog
OpenRoaming
Event Info
Things To Do
Programs
World of Solutions
Cisco Live Challenge
Sustainability
Peer Networking
Getting Around
Helpful Info
Maps (MazeMaps)
Health & Safety
See you
Log in to disco
Cisco Live 202
View My Sch
Thursday, M
Featured
Monday, Jun
11:00 AM - 1
The Bridge
and Sustain
Level 2, Ocea
Featured Session
Register