The bridge to possible

Building your Certification Test Suites using pyATS

Dave Wapstra
Software Engineering Technical Leader

cisco Live!

DEVNET-2744

Cisco Webex App

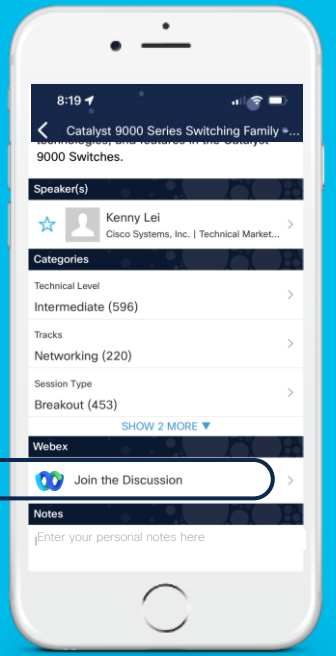
Questions?

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 until February 24, 2023.



Agenda

- Introduction
- Test Suite Creation
- Test Suite Execution
- CI/CD pipeline integration
- Conclusion

Introduction

Introduction

Create a set of automated test cases ('test suites') using pyATS

- Different types of test cases:
  - pyATS Python Testcases
  - pyATS Genie Triggers
  - pyATS Blitz Triggers
- Execution and control
  - Filter tests
  - Execute on failure
  - Pause execution on log message
- Verifications and Health

Introduction

- Prepare your test plan in Word/Excel/Wiki/etc.
- Identify TestCases, TestSections, Steps

- Create CSV file for use with pyats create project command

```
Testcase;Testsection;Testsection Description;Step
TC1;section1;This is the first section;first step
TC1;section1;This is the first section;second step
TC2;section1;This is the first section;first step
TC2;section1;This is the first section;second step
```

Test Suite Creation

Create Test Suite using pyats create project CLI

- You can use the pyats create project CLI command* with the CSV input file to create the initial code from a template project

```
$ pyats create project --csv-file data1.csv --name testPyATS

$ pyats create project --csv-file data1.csv --name testGenie \
--project-type genie

$ pyats create project --csv-file data1.csv --name testBlitz \
--project-type blitz
```

* the pyats create project CLI enhancements will be released soon

PyATS Test Suite

```
testPyATS                          <- Project Name
├── testPyATS.py                   <- Testcase module
├── testPyATS_data.yaml            <- Datafile
├── testPyATS_job.py               <- Job file
├── testPyATS_job.tem              <- Manifest file
└── testbed.yaml                   <- Testbed file
```

PyATS Test Suite

```
class CommonSetup(aetest.CommonSetup):
..
class TC1(aetest.Testcase):

    @aetest.setup
..
    @aetest.test
    def section1(self, steps):
        with steps.start('first step') as step:
            step.passed()
        with steps.start('second step') as step:
            step.passed()

    @aetest.cleanup
..
class CommonCleanup(aetest.CommonCleanup):
```

Genie Test Suite

```
testGenie                          <- Project Name
├── data
│   ├── subsection-data.yaml       Subsection datafile
│   └── trigger-data.yaml          Trigger datafile
├── etc
│   └── testbed.yaml               Testbed file
├── local_libs
│   ├── __init__.py
│   └── commons.py                 Local library
├── testGenie_job.py               Job file
├── testGenie_job.tem              Manifest file
└── testcases
    └── testGenie.py               Testcase module
```

Genie Test Suite

```
# testcases/testGenie.py
class TC1(Trigger):

    @aetest.setup
..
    @aetest.test
    def section1(self, steps, uut):
        with steps.start('first step') as step:
            step.passed()
        with steps.start('second step') as step:
            step.passed()

    @aetest.cleanup
..
```

Genie Test Suite

```
# data/trigger-data.yaml
TC1:
    source:
        pkg: testcases.testGenie
        class: TC1
    devices: [uut]
..

order:
    - TC1
    - TC2
```

Genie Test Suite

```
# data/subsection-data.yaml
setup:
    sections:
        connect:
            method: local_libs.commons.connect
    order: ['connect']


cleanup:
    sections:
        disconnect:
            method: local_libs.commons.disconnect
    order: ['disconnect']
```

Blitz Test Suite

```
testBlitz                          <- Project Name
├── testBlitz_job.py               Job file (for manifest)
├── testBlitz_job.tem              Manifest file
├── testbed.yaml                   Testbed file
└── trigger-data.yaml              Trigger datafile
```

Blitz Test Suite

```
TC1:
    source:
        pkg: genie.libs.sdk
        class: triggers.blitz.blitz.Blitz

    test_sections:
        - section1: {}  # Sections contain ”actions”
            # Example action "execute":
            # - execute:
            #     device: PE1
            #     command: show version
```

https://pubhub.devnetcloud.com/media/genie-docs/docs/blitz/design/actions/actions.html

Test suite development

https://pubhub.devnetcloud.com/media/genie-feature-browser/docs/#/apis

https://pubhub.devnetcloud.com/media/genie-feature-browser/docs/#/parsers

https://pubhub.devnetcloud.com/media/genie-feature-browser/docs/#/models

Test Execution

Test Execution using pyats run command

- You can use the pyats run CLI command to execute test suites

```
# pyATS job
$ pyats run job example_job.py --testbed-file testbed.yaml

# Genie and Blitz jobs
$ pyats run genie --testbed-file testbed.yaml --trigger-
datafile trigger-data.yaml --trigger-uids "Or('TC1|TC2')"

# All jobs with manifest file
$ pyats run manifest example_job.tem
```

PyATS Test Suite Execution Summary

```
%EASYPY-INFO: Task-1: testPyATS
%EASYPY-INFO: |-- common_setup                          PASSED
%EASYPY-INFO: |   `-- connect                           PASSED
%EASYPY-INFO: |-- TC1                                   PASSED
%EASYPY-INFO: |   |-- setup                            PASSED
%EASYPY-INFO: |   |-- section1                         PASSED
%EASYPY-INFO: |   |   |-- STEP 1: first step           PASSED
%EASYPY-INFO: |   |   `-- STEP 2: second step          PASSED
%EASYPY-INFO: |   `-- cleanup                          PASSED
%EASYPY-INFO: |-- TC2                                   PASSED
%EASYPY-INFO: |   |-- setup                            PASSED
%EASYPY-INFO: |   |-- section1                         PASSED
%EASYPY-INFO: |   |   |-- STEP 1: first step           PASSED
%EASYPY-INFO: |   |   `-- STEP 2: second step          PASSED
%EASYPY-INFO: |   `-- cleanup                          PASSED
%EASYPY-INFO: `-- common_cleanup                       PASSED
%EASYPY-INFO:     `-- disconnect                       PASSED
```

Genie Test Suite Execution Summary

```
%EASYPY-INFO: Task-1: genie_testscript
%EASYPY-INFO: |-- common_setup                          PASSED
%EASYPY-INFO: |   `-- connect                           PASSED
%EASYPY-INFO: |-- TC1.uut                               PASSED
%EASYPY-INFO: |   |-- setup                             PASSED
%EASYPY-INFO: |   |-- section1                          PASSED
%EASYPY-INFO: |   |   |-- STEP 1: first step            PASSED
%EASYPY-INFO: |   |   `-- STEP 2: second step           PASSED
%EASYPY-INFO: |   `-- cleanup                           PASSED
%EASYPY-INFO: |-- TC2.uut                               PASSED
%EASYPY-INFO: |   |-- setup                             PASSED
%EASYPY-INFO: |   |-- section1                          PASSED
%EASYPY-INFO: |   |   |-- STEP 1: first step            PASSED
%EASYPY-INFO: |   |   `-- STEP 2: second step           PASSED
%EASYPY-INFO: |   `-- cleanup                           PASSED
%EASYPY-INFO: `-- common_cleanup                        PASSED
%EASYPY-INFO:     `-- disconnect                        PASSED
```

Blitz Test Suite Execution Summary

```
%EASYPY-INFO: Task-1: genie_testscript
%EASYPY-INFO: |-- common_setup                                PASSED
%EASYPY-INFO: |   |-- connect                                 PASSED
%EASYPY-INFO: |   |-- configure                               SKIPPED
%EASYPY-INFO: |   |-- configuration_snapshot                  PASSED
%EASYPY-INFO: |   |-- save_bootvar                            PASSED
%EASYPY-INFO: |   |-- learn_system_defaults                   PASSED
%EASYPY-INFO: |   |-- initialize_traffic                      SKIPPED
%EASYPY-INFO: |   `-- PostProcessor-2                         PASSED
%EASYPY-INFO: |-- TC1.R2                                      PASSED
%EASYPY-INFO: |   `-- section1                                PASSED
%EASYPY-INFO: |-- TC2.R2                                      PASSED
%EASYPY-INFO: |   `-- section1                                PASSED
%EASYPY-INFO: `-- common_cleanup                              PASSED
%EASYPY-INFO:     |-- verify_configuration_snapshot           PASSED
%EASYPY-INFO:     |-- stop_traffic                            SKIPPED
%EASYPY-INFO:     `-- PostProcessor-2                         PASSED
```

Test Execution Control

- You can use the uids / trigger-uids CLI argument to filter testcases

```
# pyATS job
$ pyats run job … --uids "Or('TC1|TC2')"

# Genie and Blitz jobs
$ pyats run genie … --trigger-uids "Or('TC1|TC2')"
```

Test Execution Control – Execute on failure

- You can use the datafile with post processors to execute APIs or commands on failed testcases

```
# pyATS datafile.yaml
processors:
    post: processors.after_test
```

```
# trigger-datafile.yaml
global_processors:
      post:
          post_test:
            method: processors.after_test
```

Test Execution Control – Execute on failure

Processor module with post failure action implementation (pyATS)

```
# processors.py
import logging
from pyats.results import Failed

logger = logging.getLogger(__name__)

def after_test(section):
    if section.result == Failed and \
            section.__context_type__ == 'Testcase':
        logger.warning('Running checks after testcase failure')
```

Test Execution Control – Execute on failure (Genie)

```
# trigger-datafile.yaml
global_processors:
  post:
    collect_show_tech:
      method:
genie.libs.sdk.libs.abstracted_libs.processors.post_execute_command
      parameters:
        valid_section_results:
          - failed
          - passx
          - errored
        devices:
          R1:
            apis:
            - api: get_show_tech
```

Pause on phrase

- You can use the Pause on Phrase feature to pause execution

```
$ pyats run job … --pause-on [regex string|yaml file]

$ pyats run job … --pause-on ".*TC1.*"
```

```
# pause.yaml
timeout: 600          # pause a maximum of 10 minutes
patterns:
    - pattern: '.*pass.*'  # pause on all log messages
                           # with .*pass.* in them
      section: '^common_setup\..*$' # for common_setup
```

Verifications and Health

PyATS Verification and pyATS Health

- Use the Verifications to verify device state in between test cases for specific features

```
$ pyats run job job.py --verification-uids "Or('Verf','Verf2')"
```

- Use pyATS Health to verify generic device status such as CPU, Memory, Error messages, core files

```
$ pyats run job job.py --health-checks core cpu logging memory
```

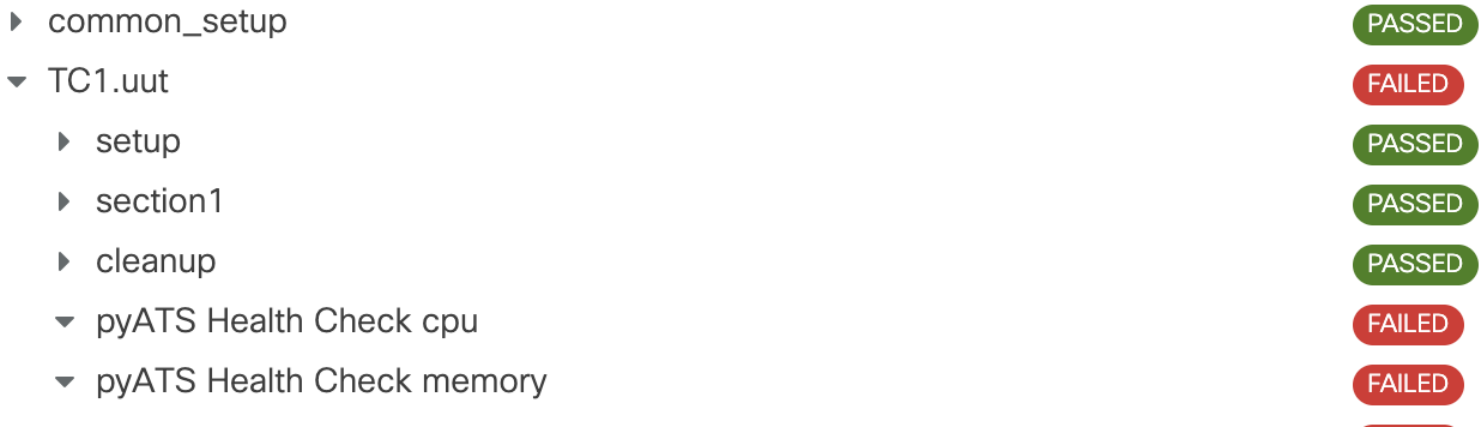
PyATS Verifications

```
$ pyats run job job.py --verification-uids "Or('Verify_Module')"
```

```
%EASYPY-INFO: |-- Verifications.TC1.uut                PASSED
%EASYPY-INFO: |   `-- Verify_Module.uut.1               PASSED
%EASYPY-INFO: |       `-- verify                       PASSED
%EASYPY-INFO: |-- TC1.uut                              PASSED
%EASYPY-INFO: |   |-- setup                            PASSED
%EASYPY-INFO: |   |-- section1                         PASSED
%EASYPY-INFO: |   |   |-- STEP 1: first step           PASSED
%EASYPY-INFO: |   |   `-- STEP 2: second step          PASSED
%EASYPY-INFO: |   `-- cleanup                          PASSED
%EASYPY-INFO: |-- Verifications.TC2.uut                PASSED
%EASYPY-INFO: |   `-- Verify_Module.uut.2              PASSED
%EASYPY-INFO: |       `-- verify                       PASSED
```

PyATS Health checks

```
$ pyats run job job.py --health-checks core cpu logging memory
```



CI/CD pipeline integration

CI/CD pipeline integration

- Exit codes
- XUNIT reports

CI/CD pipeline integration – Exit codes

- pyats run exits with system exit code 0 (Success) if all the following conditions are met:

Condition	Description
Some tests were ran	At least one test script/testcase was run and reported
Test results are available	No exception seen while trying to access test results.
success rate is 100%	Total # of tests with Passed, Passx or Skipped results / total # of tests.
All pre-job plugins ran ok	No exception seen in any pre-job plugin.
All pre-task plugins ran ok	No exception seen in any pre-task plugin for all scripts in the jobfile.
All post-task plugins ran ok	No exception seen in any post-task plugin for all scripts in the jobfile.
All post-job plugins ran ok	No exception seen in any post-job plugin.

CI/CD pipeline integration – Exit codes

pyATS Return Codes

Code	Description
1	Non-100% testcase success rate
2	Incorrect command-line arguments
3	Some errors/exceptions are seen while running plugins
4	All other exceptions unhandled by the infrastructure (unknown)

CI/CD pipeline integration – XUNIT reports

--xunit CLI option Enables the generation of an extra x-unit/j-unit result report XML.

```
# enable generation

$ pyats run job /path/to/job/file.py --xunit

# enable and also copy report to specified location

$ pyats run job /path/to/job/file.py --xunit /path/to/dir
```

Conclusion

Developing pyATS Test Suites

- Understanding of the different test suite types
- Understanding how to create a test suite and develop test cases
- Understanding how to control execution of test suites
- Understanding of additional verification and checks
- Understanding of integration with CI/CD pipelines

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

Complete your Session Survey

- Please complete your session survey after each session. Your feedback is important.
- All surveys can be taken in the Cisco Events Mobile App or by logging in to the Session Catalog and clicking the "Attendee Dashboard" at https://www.ciscolive.com/emea/learn/sessions/session-catalog.html

Continue Your Education

Visit the Cisco Showcase for related demos.

Book your one-on-one Meet the Engineer meeting.

Attend any of the related sessions at the DevNet, Capture the Flag, and Walk-in Labs zones.

Visit the On-Demand Library for more sessions at ciscolive.com/on-demand.


cisco
The bridge to possible
Thank you
cisco Live!


CISCO Live!
ALL IN