Lab - Automated Testing Using pyATS and Genie (Instructor Version)

Instructor Note: Red font color or gray highlights indicate text that appears in the instructor copy only.

Objectives

Part 1: Launch the DEVASC VM

Part 2: Create a Python Virtual Environment

Part 3: Use the pyATS Testing Library

Part 4: Use Genie to Parse IOS Command Output

Part 5: Use Genie to Compare Configurations

Part 6: Lab Cleanup and Further Investigation

Background / Scenario

In this lab, you will explore the fundamentals pyATS (pronounced "py" followed by each letter individually, "A", "T". "S") and Genie. The pyATS tool is an end-to-end testing ecosystem, specializing in data-driven and reusable testing, and engineered to be suitable for Agile, rapid development iterations. Extensible by design, pyATS enables developers start with small, simple, and linear test cases, and scale towards large, complex, and asynchronous test suites.

Genie extends and builds on pyATS to be used in a networking environment. Examples of features Genie provides include:

- device connectivity, parsers, and APIs
- platform-agnostic Python object models for features such as OSPF and BGP
- pool of reusable test cases
- YAML-driven test-runner engine

Required Resources

- 1 PC with operating system of your choice
- Virtual Box or VMWare
- DEVASC Virtual Machine
- CSR1kv Virtual Machine

Instructions

Part 1: Launch the DEVASC VM

If you have not already completed the **Lab - Install the Virtual Machine Lab Environment**, do so now. If you have already completed that lab, launch the DEVASC VM now.

Part 2: Create a Python Virtual Environment

In this part, you will create a Python virtual environment called a Python virtual environment or "venv".

Step 1: Open a terminal in the DEVASC-LABVM.

Double-click the Terminal Emulator icon on the desktop.

Step 2: Creating Python virtual environment (venv).

The pyATS tool is best installed for individual work within a venv. A venv environment is copied from your Python base environment but kept separate from it. This enables you to avoid installing software that might permanently change the overall state of your computer. The venv environment was covered in detail in the **Lab - Explore Python Development Tools** earlier in the course.

a. Create a **pyats** directory and change to that directory. You can use the characters **&&** to combine the two commands on one line.

```
devasc@labvm:~$ mkdir labs/devnet-src/pyats && cd labs/devnet-src/pyats
devasc@labvm:~/labs/devnet-src/pyats$
```

b. Create a new Python virtual environment that creates the directory **csr1kv** in the **pyats** directory.

```
devasc@labvm:~/labs/devnet-src/pyats$ python3 -m venv csr1kv
```

Note: You can also use a period "**.**" instead of a name of a directory if you want to create a venv environment in the current directory.

Step 3: Review your Python virtual environment (venv).

a. Change directories to your new "target" directory **csr1kv** and list the files. Venv creates a self-contained directory tree (test-project) that contains a Python installation for a particular version of Python, plus a number of additional packages. It also creates a bin subdirectory containing a copy of the Python binary.

Notice in particular the **bin** subdirectory and the **pyvenv.cfg** files that were created.

```
devasc@labvm:~/labs/devnet-src/pyats$ cd csr1kv
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l
total 20
drwxrwxr-x 2 devasc devasc 4096 May 31 16:07 bin
drwxrwxr-x 2 devasc devasc 4096 May 31 16:07 include
drwxrwxr-x 3 devasc devasc 4096 May 31 16:07 lib
lrwxrwxrwx 1 devasc devasc    3 May 31 16:07 lib64 -> lib
-rw-rw-r-- 1 devasc devasc   69 May 31 16:07 pyvenv.cfg
drwxrwxr-x 3 devasc devasc 4096 May 31 16:07 share
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

b. Examine the contents of the **pyvenv.cfg** file. Notice that this file points to the location of your Python installation in /usr/bin.

```
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat pyvenv.cfg
home = /usr/bin
include-system-site-packages = false
version = 3.8.2
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

c. A symbolic link (also known as a symlink) is a special type of file that serves as a reference to another file or directory. To get a better understanding of the venv and how it uses symbolic links, list Python files in the /usr/bin directory referenced in the **pyvenv.cfg** file. Use the **ls** number one option (-1) to list the files each on one line.

```
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -1 /usr/bin/python*
/usr/bin/python3
/usr/bin/python3.8
```

```
/usr/bin/python3.8-config
/usr/bin/python3-config
/usr/bin/python-argcomplete-check-easy-install-script3
/usr/bin/python-argcomplete-tcsh3
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

d. Now examine the contents of the venv-created **bin** subdirectory. Notice there are two files in this subdirectory, both of which are symlinks. In this case, it is a link to the Python binaries in **/usr/bin**. Symlinks are used to link libraries and make sure files there have consistent access to these files without having to move or create a copy of the original file. There is also a file, **activate**, that will be discussed next.

```
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l bin
total 44
-rw-r--r-- 1 devasc devasc 2225 May 31 16:07 activate
-rw-r--r-- 1 devasc devasc 1277 May 31 16:07 activate.csh
-rw-r--r-- 1 devasc devasc 2429 May 31 16:07 activate.fish
-rw-r--r-- 1 devasc devasc 8471 May 31 16:07 Activate.ps1
-rwxrwxr-x 1 devasc devasc  267 May 31 16:07 easy_install
-rwxrwxr-x 1 devasc devasc  267 May 31 16:07 easy_install-3.8
-rwxrwxr-x 1 devasc devasc  258 May 31 16:07 pip
-rwxrwxr-x 1 devasc devasc  258 May 31 16:07 pip3
-rwxrwxr-x 1 devasc devasc  258 May 31 16:07 pip3.8
lrwxrwxrwx 1 devasc devasc    7 May 31 16:07 python -> python3
lrwxrwxrwx 1 devasc devasc   16 May 31 16:07 python3 -> /usr/bin/python3
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

e. Launch the virtual environment using **bin/activate**. Notice your prompt is now preceded with **(csr1kv)**. All the commands done from this point on are within this venv.

```
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ source bin/activate
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Note: The **deactivate** command is used to exit the venv environment and return to the normal shell environment.

Part 3: Use the pyATS Testing Library

In this part, you will use pyATS, a python testing library.

Step 1: Installing pyATS.

Install pyATS using **pip3**. This will take a few minutes. During installation you may see some errors. These can usually be ignored as long as pyATS can be verified as shown in the next step.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ pip3 install
pyats[full]
Collecting pyats[full]
  Downloading pyats-20.4-cp38-cp38-manylinux1_x86_64.whl (2.0 MB)

<output omitted>
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Step 2: Verifying pyATS.

Verify that pyATS was successfully installed using the **pyats --help** command. Notice you can get additional help on any pyats command with the **pyats <command> --help** command.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ pyats --help
Usage:
  pyats <command> [options]

Commands:
    create              create scripts and libraries from template
    diff                Command to diff two snapshots saved to file or directory
    dnac                Command to learn DNAC features and save to file (Prototype)
    learn               Command to learn device features and save to file
    logs                command enabling log archive viewing in local browser
    parse               Command to parse show commands
    run                 runs the provided script and output corresponding results.
    secret              utilities for working with secret strings.
    shell               enter Python shell, loading a pyATS testbed file and/or
pickled data
    validate            utlities that helps to validate input files
    version             commands related to version display and manipulation

General Options:
  -h, --help            Show help

Run 'pyats <command> --help' for more information on a command.
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Step 3: Clone and examine the pyATS sample scripts from GitHub.

a. Clone the Github pyATS sample scripts repository **CiscoTestAutomation**.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ git clone
https://github.com/CiscoTestAutomation/examples
Cloning into 'examples'...
remote: Enumerating objects: 35, done.
remote: Counting objects: 100% (35/35), done.
remote: Compressing objects: 100% (31/31), done.
remote: Total 658 (delta 11), reused 18 (delta 4), pack-reused 623
Receiving objects: 100% (658/658), 1.00 MiB | 4.82 MiB/s, done.
Resolving deltas: 100% (338/338), done.
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

b. Verify the copy was successful by listing the files in the current directory. Notice there is a new subdirectory **example**.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l
total 24
drwxrwxr-x  2 devasc devasc 4096 May 31 16:07 bin
drwxrwxr-x 21 devasc devasc 4096 May 31 16:47 examples
drwxrwxr-x  2 devasc devasc 4096 May 31 16:07 include
drwxrwxr-x  3 devasc devasc 4096 May 31 16:07 lib
lrwxrwxrwx  1 devasc devasc    3 May 31 16:07 lib64 -> lib
```

```
-rw-rw-r--  1 devasc devasc   69 May 31 16:07 pyvenv.cfg
drwxrwxr-x  3 devasc devasc 4096 May 31 16:07 share
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

c. List the files in the **examples** subdirectory. Notice there is a subdirectory, **basic**, along with a several other files.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l examples
total 88
drwxrwxr-x  3 devasc devasc  4096 May 31 16:47 abstraction_example
drwxrwxr-x  2 devasc devasc  4096 May 31 16:47 basic
<output omitted>
drwxrwxr-x  2 devasc devasc  4096 May 31 16:47 uids
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

d. List the files in this **basic** subdirectory. This is the location of the scripts you will be using in the next step.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l examples/basic
total 12
-rw-rw-r-- 1 devasc devasc  510 May 31 16:47 basic_example_job.py
-rwxrwxr-x 1 devasc devasc 4475 May 31 16:47 basic_example_script.py
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Step 4: Examine the basic script files.

The test declaration syntax for pyATS is based on popular Python unit-testing frameworks like pytest. It supports basic testing statements, such as an assertion that a variable has a given value, and along with explicitly providing results via specific APIs.

a. The Python script you will use is **basic_example_script.py**. Display the content of the Python script using the **cat** command. Pipe it to **more** if you want to view it one screen or line at a time. Notice that this script contains the following sections as highlighted in the output below:

- A common setup block
- Multiple testing blocks
- A common Cleanup block

These blocks contain statements that prepare and/or determine readiness of the test topology (a process that can include problem injection), perform tests, and then return the topology to a known state.

The Testing blocks - often referred to in pyATS documentation as the Test Cases - can each contain multiple tests, with their own Setup and Cleanup code. Best practice suggests, though, that the common Cleanup section, at the end, be designed for idempotency, meaning it should check and restore all changes made by Setup and Test, and restore the topology to its original, desired state.

Note: Although it is not necessary to understand the code, you will find it helpful to read the comments within the Python script.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat
examples/basic/basic_example_script.py | more
#!/usr/bin/env python
######################################################################
# basic_example.py : A very simple test script example which include:
#     common_setup
#     Tescases
#     common_cleanup
# The purpose of this sample test script is to show the "hello world"
```

```
# of aetest.
##################################################################

# To get a logger for the script
import logging

# Needed for aetest script
from pyats import aetest

# Get your logger for your script
log = logging.getLogger(__name__)

##################################################################
###                  COMMON SETUP SECTION                      ###
##################################################################

# This is how to create a CommonSetup
# You can have one of no CommonSetup
# CommonSetup can be named whatever you want

class common_setup(aetest.CommonSetup):
    """ Common Setup section """

    # CommonSetup have subsection.
    # You can have 1 to as many subsection as wanted
    # here is an example of 2 subsections

    # First subsection
    @aetest.subsection
    def sample_subsection_1(self):
        """ Common Setup subsection """
        log.info("Aetest Common Setup ")

    # If you want to get the name of current section,
    # add section to the argument of the function.

    # Second subsection
    @aetest.subsection
    def sample_subsection_2(self, section):
        """ Common Setup subsection """
        log.info("Inside %s" % (section))

        # And how to access the class itself ?

        # self refers to the instance of that class, and remains consistent
        # throughout the execution of that container.
        log.info("Inside class %s" % (self.uid))

##################################################################
```

```
###                    TESTCASES SECTION                        ###
##################################################################

# This is how to create a testcase
# You can have 0 to as many testcase as wanted

# Testcase name : tc_one
class tc_one(aetest.Testcase):
    """ This is user Testcases section """

    # Testcases are divided into 3 sections
    # Setup, Test and Cleanup.

    # This is how to create a setup section
    @aetest.setup
    def prepare_testcase(self, section):
        """ Testcase Setup section """
        log.info("Preparing the test")
        log.info(section)

    # This is how to create a test section
    # You can have 0 to as many test section as wanted

    # First test section
    @ aetest.test
    def simple_test_1(self):
        """ Sample test section. Only print """
        log.info("First test section ")

    # Second test section
    @ aetest.test
    def simple_test_2(self):
        """ Sample test section. Only print """
        log.info("Second test section ")

    # This is how to create a cleanup section
    @aetest.cleanup
    def clean_testcase(self):
        """ Testcase cleanup section """
        log.info("Pass testcase cleanup")

# Testcase name : tc_two
class tc_two(aetest.Testcase):
    """ This is user Testcases section """

    @ aetest.test
    def simple_test_1(self):
        """ Sample test section. Only print """
        log.info("First test section ")
```

```
            self.failed('This is an intentional failure')

        # Second test section
        @ aetest.test
        def simple_test_2(self):
            """ Sample test section. Only print """
            log.info("Second test section ")

        # This is how to create a cleanup section
        @aetest.cleanup
        def clean_testcase(self):
            """ Testcase cleanup section """
            log.info("Pass testcase cleanup")

    ######################################################################
    ####                        COMMON CLEANUP SECTION                  ###
    ######################################################################

    # This is how to create a CommonCleanup
    # You can have 0 , or 1 CommonCleanup.
    # CommonCleanup can be named whatever you want :)
    class common_cleanup(aetest.CommonCleanup):
        """ Common Cleanup for Sample Test """

        # CommonCleanup follow exactly the same rule as CommonSetup regarding
        # subsection
        # You can have 1 to as many subsection as wanted
        # here is an example of 1 subsections

        @aetest.subsection
        def clean_everything(self):
            """ Common Cleanup Subsection """
            log.info("Aetest Common Cleanup ")

    if __name__ == '__main__': # pragma: no cover
        aetest.main()
    (csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

A pyATS script is a Python file where pyATS tests are declared. It can be run directly as a standalone Python script file, generating output only to your terminal window. Alternatively, one or more pyATS scripts can be compiled into a "job" and run together as a batch, through the pyATS EasyPy module. EasyPy enables parallel execution of multiple scripts, collects logs in one place, and provides a central point from which to inject changes to the topology under test.

b. Use **cat** to display your pyATS job file, **pyats_sample_job.py**. Notice the instructions on how to run this file, highlighted below.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat
examples/basic/basic_example_job.py
# To run the job:
# pyats run job basic_example_job.py
# Description: This example shows the basic functionality of pyats
```

```
#              with few passing tests

import os
from pyats.easypy import run

# All run() must be inside a main function
def main():
    # Find the location of the script in relation to the job file
    test_path = os.path.dirname(os.path.abspath(__file__))
    testscript = os.path.join(test_path, 'basic_example_script.py')

    # Execute the testscript
    run(testscript=testscript)
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Step 5: Run pyATS manually to invoke the basic test case.

a. Using the pyATS job and script files, run pyATS manually to invoke the basic test case. This will verify the pyATS job and script files work properly. The information in the output is beyond the scope of this lab, however you will notice that the job and script passed all required tasks.

Note: The output below was truncated. The Cisco Test Automation repository on GitHub is subject to change, which includes the pyATS job and scripts files. Your output is subject to change but should not affect your outcome. For example, an intentional failure was added to the **basic_example_script.py** file. This is an intentional failure and does not cause any problems. It is an example that repositories are dynamic. It is one of the highlighted lines below.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ pyats run job
examples/basic/basic_example_job.py
2020-05-31T17:10:17: %EASYPY-INFO: Starting job run: basic_example_job
2020-05-31T17:10:17: %EASYPY-INFO: Runinfo directory:
/home/devasc/.pyats/runinfo/basic_example_job.2020May31_17:10:16.735106
2020-05-31T17:10:17: %EASYPY-INFO: ----------------------------------------------
------------------------------
2020-05-31T17:10:18: %EASYPY-INFO: Starting task execution: Task-1
2020-05-31T17:10:18: %EASYPY-INFO:     test harness = pyats.aetest
2020-05-31T17:10:18: %EASYPY-INFO:     testscript   = /home/devasc/labs/devnet-
src/pyats/csr1kv/examples/basic/basic_example_script.py
2020-05-31T17:10:18: %AETEST-INFO: +-------------------------------------------
----------------------------+
2020-05-31T17:10:18: %AETEST-INFO: |                            Starting common setup
|
<output omitted>
-------+
2020-05-31T17:10:18: %SCRIPT-INFO: First test section
2020-05-31T17:10:18: %AETEST-ERROR: Failed reason: This is an intentional failure
2020-05-31T17:10:18: %AETEST-INFO: The result of section simple_test_1 is => FAILED
2020-05-31T17:10:18: %AETEST-INFO: +-------------------------------------------
----------------------------+
2020-05-31T17:10:18: %AETEST-INFO: |                         Starting section
simple_test_2                          |
<output omitted>
-------+
```

```
2020-05-31T17:10:20: %EASYPY-INFO: |                                    Easypy Report
|
2020-05-31T17:10:20: %EASYPY-INFO: +------------------------------------------------
----------------------------+
<output omitted>
2020-05-31T17:10:20: %EASYPY-INFO: Overall Stats
2020-05-31T17:10:20: %EASYPY-INFO:     Passed      : 3
2020-05-31T17:10:20: %EASYPY-INFO:     Passx       : 0
2020-05-31T17:10:20: %EASYPY-INFO:     Failed      : 1
2020-05-31T17:10:20: %EASYPY-INFO:     Aborted     : 0
2020-05-31T17:10:20: %EASYPY-INFO:     Blocked     : 0
2020-05-31T17:10:20: %EASYPY-INFO:     Skipped     : 0
2020-05-31T17:10:20: %EASYPY-INFO:     Errored     : 0
2020-05-31T17:10:20: %EASYPY-INFO:
2020-05-31T17:10:20: %EASYPY-INFO:     TOTAL       : 4
2020-05-31T17:10:20: %EASYPY-INFO:
2020-05-31T17:10:20: %EASYPY-INFO: Success Rate    : 75.00 %
2020-05-31T17:10:20: %EASYPY-INFO:
2020-05-31T17:10:20: %EASYPY-INFO: +------------------------------------------------
----------------------------+
2020-05-31T17:10:20: %EASYPY-INFO: |                               Task Result Summary
|
2020-05-31T17:10:20: %EASYPY-INFO: +------------------------------------------------
----------------------------+
2020-05-31T17:10:20: %EASYPY-INFO: Task-1: basic_example_script.common_setup
PASSED
2020-05-31T17:10:20: %EASYPY-INFO: Task-1: basic_example_script.tc_one
PASSED
2020-05-31T17:10:20: %EASYPY-INFO: Task-1: basic_example_script.tc_two
FAILED
2020-05-31T17:10:20: %EASYPY-INFO: Task-1: basic_example_script.common_cleanup
PASSED
2020-05-31T17:10:20: %EASYPY-INFO:
2020-05-31T17:10:20: %EASYPY-INFO: +------------------------------------------------
----------------------------+
2020-05-31T17:10:20: %EASYPY-INFO: |                               Task Result Details
|
2020-05-31T17:10:20: %EASYPY-INFO: +------------------------------------------------
----------------------------+
2020-05-31T17:10:20: %EASYPY-INFO: Task-1: basic_example_script
2020-05-31T17:10:20: %EASYPY-INFO: |-- common_setup
PASSED
2020-05-31T17:10:20: %EASYPY-INFO: |   |-- sample_subsection_1
PASSED
2020-05-31T17:10:20: %EASYPY-INFO: |   `-- sample_subsection_2
PASSED
2020-05-31T17:10:20: %EASYPY-INFO: |-- tc_one
PASSED
2020-05-31T17:10:20: %EASYPY-INFO: |   |-- prepare_testcase
PASSED
2020-05-31T17:10:20: %EASYPY-INFO: |   |-- simple_test_1
PASSED
```

```
2020-05-31T17:10:20: %EASYPY-INFO: |   |-- simple_test_2
PASSED
2020-05-31T17:10:20: %EASYPY-INFO: |   `-- clean_testcase
PASSED
2020-05-31T17:10:20: %EASYPY-INFO: |-- tc_two
FAILED
2020-05-31T17:10:20: %EASYPY-INFO: |   |-- simple_test_1
FAILED
2020-05-31T17:10:20: %EASYPY-INFO: |   |-- simple_test_2
PASSED
2020-05-31T17:10:20: %EASYPY-INFO: |   `-- clean_testcase
PASSED
2020-05-31T17:10:20: %EASYPY-INFO: `-- common_cleanup
PASSED
2020-05-31T17:10:20: %EASYPY-INFO:     `-- clean_everything
PASSED
2020-05-31T17:10:20: %EASYPY-INFO: Sending report email...
2020-05-31T17:10:20: %EASYPY-INFO: Missing SMTP server configuration, or failed to
reach/authenticate/send mail. Result notification email failed to send.
2020-05-31T17:10:20: %EASYPY-INFO: Done!

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

(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Part 4: Use Genie to Parse IOS Command Output

In this Part, you will use Genie to take unstructured IOS output and parse it into JSON output.

Note: Not all IOS commands are supported. Complete Genie documentation can be found at: https://developer.cisco.com/docs/genie-docs/

Step 1: Create a testbed YAML file.

The pyATS and Genie tools use a YAML file to know which devices to connect to and what the proper credentials are. This file is known as a testbed file. Genie includes built-in functionality to build the testbed file for you.

a. Enter the command **genie --help** to see all the available commands. For additional help on any command, use the **<command>** parameter, as shown below, for the **create** command. Notice that **testbed** is one of the options for the **create** command.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie --help
Usage:
  genie <command> [options]

Commands:
    create              Create Testbed, parser, triggers, ...
    diff                Command to diff two snapshots saved to file or directory
    dnac                Command to learn DNAC features and save to file (Prototype)
    learn               Command to learn device features and save to file
    parse               Command to parse show commands
```

```
    run                 Run Genie triggers & verifications in pyATS runtime
environment
    shell               enter Python shell, loading a pyATS testbed file and/or
pickled data

General Options:
  -h, --help            Show help

Run 'genie <command> --help' for more information on a command.
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie create --help
Usage:
  genie create <subcommand> [options]

Subcommands:
    parser              create a new Genie parser from template
    testbed             create a testbed file automatically
    trigger             create a new Genie trigger from template

General Options:
  -h, --help            Show help
  -v, --verbose         Give more output, additive up to 3 times.
  -q, --quiet           Give less output, additive up to 3 times, corresponding to
WARNING, ERROR,
                        and CRITICAL logging levels
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

b. To create your testbed YAML file, enter the command below. The **--output** parameter will create a **testbed.yml** file in a directory named **yaml**. The directory will be automatically created. The **--encode-password** parameter will encode the passwords in the YAML file. The parameter **interactive** means you will be asked a series of questions. Answer no to the first three questions. And then provide the following answers to create the **testbed.yaml** file.

- **Device hostname** - This must match the hostname of the device, which for this lab is **CSR1kv**.
- **IP address** - This must match your CSR1kv IPv4 address you discovered earlier in this lab. Shown here is **192.168.56.101**.
- **Username** - This is the local username used for ssh, which is **cisco**.
- **Default password** - This is the local password used for ssh, which is **cisco123!**.
- **Enable password** - Leave blank. There is no privileged password configured on the router.
- **Protocol** - SSH along with the key exchange group expected by the router.
- **OS** - The OS on the router.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie create testbed
interactive --output yaml/testbed.yml --encode-password
Start creating Testbed yaml file ...
Do all of the devices have the same username? [y/n] n
Do all of the devices have the same default password? [y/n] n
Do all of the devices have the same enable password? [y/n] n

Device hostname: CSR1kv
```

```
   IP (ip, or ip:port): 192.168.56.101
   Username: cisco
Default Password (leave blank if you want to enter on demand): cisco123!
Enable Password (leave blank if you want to enter on demand):
   Protocol (ssh, telnet, ...): ssh -o KexAlgorithms=diffie-hellman-group14-
sha1
   OS (iosxr, iosxe, ios, nxos, linux, ...): iosxe
More devices to add ? [y/n] n
Testbed file generated:
yaml/testbed.yml

(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

c. Use **cat** to view the **testbed.yml** file in the **yaml** directory. Notice your entries in the YAML file. Your SSH password is encrypted and the enable password will "ASK" the user to enter the password if one is required.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat yaml/testbed.yml
devices:
  CSR1kv:
    connections:
      cli:
        ip: 192.168.56.101
        protocol: ssh -o KexAlgorithms=diffie-hellman-group14-sha1
    credentials:
      default:
        password: '%ENC{w5PDosOUw5fDosKQwpbCmMKH}'
        username: cisco
      enable:
        password: '%ASK{}'
    os: iosxe
    type: iosxe
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Step 2: Use Genie to parse output from the show ip interface brief command into JSON.

a. If you have not already completed the **Lab - Install the CSR1kv VM**, do so now. If you have already completed that lab, launch the CSR1kv VM now.

b. In the CSR1kv VM, enter the command **show ip interface brief** from privileged exec mode. Your address may be incremented to some other address other than 192.168.56.101. Make note of the IPv4 address for your CSR1kv VM. You will use it later in the lab.

```
CSR1kv> en
CSR1kv# show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.101  YES DHCP   up                    up
CSR1kv#
```

c. Using your testbed YAML file, invoke Genie to parse unstructured output from the **show ip interface brief** command into structured JSON. This command includes the IOS command to be parsed (**show ip interface brief**), the YAML testbed file (**testbed.yml**), and the specified device in the testbed file (**CSR1kv**).

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie parse "show ip
interface brief" --testbed-file yaml/testbed.yml --devices CSR1kv
Enter enable password for device CSR1kv: <Enter>
2020-05-31T18:59:23: %UNICON-WARNING: Device 'CSR1kv' connection 'cli' does not have
IP and/or port specified, ignoring
Device 'CSR1kv' connection 'cli' does not have IP and/or port specified, ignoring
  0%|
| 0/1 [00:00<?, ?it/s]{
  "interface": {
    "GigabitEthernet1": {
      "interface_is_ok": "YES",
      "ip_address": "192.168.56.101",
      "method": "DHCP",
      "protocol": "up",
      "status": "up"
    }
  }
}
100%|████████████████████████████████████████████████████████████████████████████|
1/1 [00:00<00:00,  2.22it/s]

(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ^C
```

Step 3: Use Genie to parse output from the show version command into JSON.

For another example, parse unstructured output from the **show version** command into structured JSON.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie parse "show
version" --testbed-file yaml/testbed.yml --devices CSR1kv
Enter enable password for device CSR1kv: <Enter>
2020-05-31T18:41:32: %UNICON-WARNING: Device 'CSR1kv' connection 'cli' does not have
IP and/or port specified, ignoring
Device 'CSR1kv' connection 'cli' does not have IP and/or port specified, ignoring
  0%|
| 0/1 [00:00<?, ?it/s]{
  "version": {
    "chassis": "CSR1000V",
    "chassis_sn": "9K8P1OFYE3D",
    "compiled_by": "mcpre",
    "compiled_date": "Thu 30-Jan-20 18:48",
    "curr_config_register": "0x2102",
    "disks": {
      "bootflash:.": {
        "disk_size": "7774207",
        "type_of_disk": "virtual hard disk"
      },
      "webui:.": {
        "disk_size": "0",
        "type_of_disk": "WebUI ODM Files"
      }
    },
    "hostname": "CSR1kv",
```

```
      "image_id": "X86_64_LINUX_IOSD-UNIVERSALK9-M",
      "image_type": "production image",
      "last_reload_reason": "reload",
      "license_level": "ax",
      "license_type": "Default. No valid license found.",
      "main_mem": "2182252",
      "mem_size": {
        "non-volatile configuration": "32768",
        "physical": "3985032"
      },
      "next_reload_license_level": "ax",
      "number_of_intfs": {
        "Gigabit Ethernet": "1"
      },
      "os": "IOS-XE",
      "platform": "Virtual XE",
      "processor_type": "VXE",
      "returned_to_rom_by": "reload",
      "rom": "IOS-XE ROMMON",
      "rtr_type": "CSR1000V",
      "system_image": "bootflash:packages.conf",
      "uptime": "2 days, 6 hours, 26 minutes",
      "uptime_this_cp": "2 days, 6 hours, 27 minutes",
      "version": "16.9.5",
      "version_short": "16.9"
    }
}
100%|████████████████████████████████████████████████████████████████████████████████|
1/1 [00:00<00:00,  2.06it/s]

(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Part 5: Use Genie to Compare Configurations

As you have seen Genie can be used to parse show commands into structured json. Genie can also be used to:

- Take snapshots of configs annually and make comparisons between them
- Automate testing deployments against a virtual environment for testing before deployment in production
- To troubleshoot configurations by doing comparisons between devices

In parts 5 and 6, you will see how to do a comparison between two different outputs.

Step 1: Add an IPv6 address to CSR1kv.

a. On the CSR1kv VM add the following IPv6 address:

```
CSR1kv(config)# interface gig 1
CSR1kv(config-if)# ipv6 address 2001:db8:acad:56::101/64
```

Step 2: Use Genie to verify configuration and parse output in JSON.

a. Parse unstructured output from the **show ipv6 interface** command into structured JSON. Use the **--output** parameter to send the output to a directory **verify-ipv6-1**. Notice in the output that Genie tells you that two files were created.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie parse "show ipv6 interface gig 1" --testbed-file yaml/testbed.yml --devices CSR1kv --output verify-ipv6-1
Enter enable password for device CSR1kv: <Enter>
2020-05-31T19:36:19: %UNICON-WARNING: Device 'CSR1kv' connection 'cli' does not have IP and/or port specified, ignoring
Device 'CSR1kv' connection 'cli' does not have IP and/or port specified, ignoring
100%|████████████████████████████████████████████████████████████████████████████████|
1/1 [00:00<00:00,  2.08it/s]
+==============================================================================+
| Genie Parse Summary for CSR1kv                                               |
+==============================================================================+
|  Connected to CSR1kv                                                         |
|  -  Log: verify-ipv6-1/connection_CSR1kv.txt                                 |
|------------------------------------------------------------------------------|
|  Parsed command 'show ipv6 interface gig 1'                                  |
|  -  Parsed structure: verify-ipv6-1/CSR1kv_show-ipv6-interface-              |
| gig-1_parsed.txt                                                             |
|  -  Device Console:   verify-ipv6-1/CSR1kv_show-ipv6-interface-              |
| gig-1_console.txt                                                            |
|------------------------------------------------------------------------------|
csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

b. List the files created by Genie in the directory **verify-ipv6-1**. Notice there were two files created with the similar names but one ending in **_console.txt** and the other in **_parsed.txt**. The name of each file includes the device name and the IOS command used in the Genie parse command.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l verify-ipv6-1
total 16
-rw-rw-rw- 1 devasc devasc 9094 May 31 19:36 connection_CSR1kv.txt
-rw-rw-r-- 1 devasc devasc  745 May 31 19:36 CSR1kv_show-ipv6-interface-gig-1_console.txt
-rw-rw-r-- 1 devasc devasc  877 May 31 19:36 CSR1kv_show-ipv6-interface-gig-1_parsed.txt
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

c. Use **cat** to examine the contents of the **_console.txt** file. Notice both the IPv6 global unicast address that you configured and an automatic EUI-64 link-local address.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat verify-ipv6-1/CSR1kv_show-ipv6-interface-gig-1_console.txt
+++ CSR1kv: executing command 'show ipv6 interface gig 1' +++
show ipv6 interface gig 1
GigabitEthernet1 is up, line protocol is up
  IPv6 is enabled, link-local address is FE80::A00:27FF:FE73:D79F
  No Virtual link-local address(es):
  Description: VBox
  Global unicast address(es):
```

```
    2001:DB8:ACAD:56::101, subnet is 2001:DB8:ACAD:56::/64
  Joined group address(es):
    FF02::1
    FF02::1:FF00:101
    FF02::1:FF73:D79F
  MTU is 1500 bytes
  ICMP error messages limited to one every 100 milliseconds
  ICMP redirects are enabled
  ICMP unreachables are sent
  ND DAD is enabled, number of DAD attempts: 1
  ND reachable time is 30000 milliseconds (using 30000)
  ND NS retransmit interval is 1000 milliseconds
CSR1kv#
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

d. Use **cat** to examine the contents of the **_parsed.txt** file. This is the parsed JSON file of the **show ipv6 interface gig 1** command.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat verify-ipv6-
1/CSR1kv_show-ipv6-interface-gig-1_parsed.txt
{
  "GigabitEthernet1": {
    "enabled": true,
    "ipv6": {
      "2001:DB8:ACAD:56::101/64": {
        "ip": "2001:DB8:ACAD:56::101",
        "prefix_length": "64",
        "status": "valid"
      },
      "FE80::A00:27FF:FE73:D79F": {
        "ip": "FE80::A00:27FF:FE73:D79F",
        "origin": "link_layer",
        "status": "valid"
      },
      "enabled": true,
      "icmp": {
        "error_messages_limited": 100,
        "redirects": true,
        "unreachables": "sent"
      },
      "nd": {
        "dad_attempts": 1,
        "dad_enabled": true,
        "ns_retransmit_interval": 1000,
        "reachable_time": 30000,
        "suppress": false,
        "using_time": 30000
      }
    },
```

```
      "joined_group_addresses": [
        "FF02::1",
        "FF02::1:FF00:101",
        "FF02::1:FF73:D79F"
      ],
      "mtu": 1500,
      "oper_status": "up"
    },
    "_exclude": []
}
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Step 3: Modify the IPv6 Link-Local address.

On CSR1kv VM add the following IPv6 address:

```
CSR1kv> en
CSR1kv# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
CSR1kv(config)# interface gig 1
CSR1kv(config-if)# ipv6 address fe80::56:1 link-local
```

Step 4: Use Genie to verify configuration and parse output in JSON.

a. Parse unstructured output from the **show ipv6 interface** command into structured JSON. Use the **--output** parameter to send the output to a different directory **verify-ipv6-2**. You can use the command history to recall the previous command (up arrow). Just make sure you change the **1** to a **2** to create a new **verify-ipv6-2** directory.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie parse "show ipv6
interface gig 1" --testbed-file yaml/testbed.yml --devices CSR1kv --output
verify-ipv6-2
Enter enable password for device CSR1kv: <Enter>
2020-05-31T20:03:58: %UNICON-WARNING: Device 'CSR1kv' connection 'cli' does not have
IP and/or port specified, ignoring
Device 'CSR1kv' connection 'cli' does not have IP and/or port specified, ignoring
100%|████████████████████████████████████████████████████████████████████████████|
1/1 [00:00<00:00,  2.24it/s]
+==============================================================================+
| Genie Parse Summary for CSR1kv                                               |
+==============================================================================+
|  Connected to CSR1kv                                                         |
|  -  Log: verify-ipv6-2/connection_CSR1kv.txt                                 |
|------------------------------------------------------------------------------|
|  Parsed command 'show ipv6 interface gig 1'                                  |
|  -  Parsed structure: verify-ipv6-2/CSR1kv_show-ipv6-interface-              |
| gig-1_parsed.txt                                                             |
|  -  Device Console:   verify-ipv6-2/CSR1kv_show-ipv6-interface-              |
| gig-1_console.txt                                                            |
|------------------------------------------------------------------------------|

(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

b. List the files created by Genie in the directory **verify-ipv6-2**. These are similar to the two files you created before changing the IPv6 link-local address.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l verify-ipv6-2
total 16
-rw-rw-rw- 1 devasc devasc 4536 May 31 20:04 connection_CSR1kv.txt
-rw-rw-r-- 1 devasc devasc  728 May 31 20:04 CSR1kv_show-ipv6-interface-gig-1_console.txt
-rw-rw-r-- 1 devasc devasc  846 May 31 20:04 CSR1kv_show-ipv6-interface-gig-1_parsed.txt
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

c. Use the **cat** to examine the contents each file. The changes are highlighted in the output below.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat verify-ipv6-2/CSR1kv_show-ipv6-interface-gig-1_console.txt
+++ CSR1kv: executing command 'show ipv6 interface gig 1' +++
show ipv6 interface gig 1
GigabitEthernet1 is up, line protocol is up
  IPv6 is enabled, link-local address is FE80::56:1
  No Virtual link-local address(es):
  Description: VBox
  Global unicast address(es):
    2001:DB8:ACAD:56::101, subnet is 2001:DB8:ACAD:56::/64
  Joined group address(es):
    FF02::1
    FF02::1:FF00:101
    FF02::1:FF56:1
  MTU is 1500 bytes
  ICMP error messages limited to one every 100 milliseconds
  ICMP redirects are enabled
  ICMP unreachables are sent
  ND DAD is enabled, number of DAD attempts: 1
  ND reachable time is 30000 milliseconds (using 30000)
  ND NS retransmit interval is 1000 milliseconds
CSR1kv#
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat verify-ipv6-2/CSR1kv_show-ipv6-interface-gig-1_parsed.txt
{
  "GigabitEthernet1": {
    "enabled": true,
    "ipv6": {
      "2001:DB8:ACAD:56::101/64": {
        "ip": "2001:DB8:ACAD:56::101",
        "prefix_length": "64",
        "status": "valid"
      },
      "FE80::56:1": {
        "ip": "FE80::56:1",
        "origin": "link_layer",
        "status": "valid"
      },
```

```
      "enabled": true,
      "icmp": {
        "error_messages_limited": 100,
        "redirects": true,
        "unreachables": "sent"
      },
      "nd": {
        "dad_attempts": 1,
        "dad_enabled": true,
        "ns_retransmit_interval": 1000,
        "reachable_time": 30000,
        "suppress": false,
        "using_time": 30000
      }
    },
    "joined_group_addresses": [
      "FF02::1",
      "FF02::1:FF00:101",
      "FF02::1:FF56:1"
    ],
    "mtu": 1500,
    "oper_status": "up"
  },
  "_exclude": []
}
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Step 5: Use Genie to compare the difference between the configurations.

In the previous step, it is fairly easy to find the change to the IPv6 link-local address. But assume you were looking for a problem in a complex configuration. Perhaps, you are trying to find a difference between an OSPF configuration on a router that is receiving the proper routes and another router that is not, and you want to see the differences in their OSPF configurations. Or perhaps, you are trying to spot the difference in a long list of ACL statements between two routers that are supposed to have identical security policies. Genie can do the comparison for you and make it easy to find the differences.

a. Use the following command to have Genie find the differences between the two parsed JSON files. Notice that the output tells you where you can find Genie's comparisons. In this case, the first filename is the previous configuration and the second filename is the current configuration.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie diff verify-ipv6-1 verify-ipv6-2
1it [00:00, 579.32it/s]
+==============================================================================+
| Genie Diff Summary between directories verify-ipv6-1/ and verify-ipv6-2/     |
+==============================================================================+
|  File: CSR1kv_show-ipv6-interface-gig-1_parsed.txt                           |
|   - Diff can be found at ./diff_CSR1kv_show-ipv6-interface-gig-1_parsed.txt  |
|------------------------------------------------------------------------------|
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

b. Use **cat** to view the contents of the file with the differences. The plus "+" sign indicated additions and the minus "-" sign indicates what was removed.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat ./diff_CSR1kv_show-ipv6-interface-gig-1_parsed.txt
--- verify-ipv6-1/CSR1kv_show-ipv6-interface-gig-1_parsed.txt
+++ verify-ipv6-2/CSR1kv_show-ipv6-interface-gig-1_parsed.txt
 GigabitEthernet1:
   ipv6:
+    FE80::56:1:
+     ip: FE80::56:1
+     origin: link_layer
+     status: valid
-    FE80::A00:27FF:FE73:D79F:
-     ip: FE80::A00:27FF:FE73:D79F
-     origin: link_layer
-     status: valid
   joined_group_addresses:
-    index[2]: FF02::1:FF73:D79F
+    index[2]: FF02::1:FF56:1
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Part 6: Lab Cleanup and Further Investigation

In this Part, you will deactivate your Python venv and investigate other Genie use cases.

Step 1: Deactivate your Python virtual environment.

When you have completed this lab, you can deactivate your Python virtual environment using the **deactivate** command. Notice that your prompt is no longer preceded by "(csr1kv)".

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ deactivate
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Step 2: Explore more pyATS and Genie use cases.

Previously in this lab, you cloned the **examples** folder from the Cisco Test Automation with pyATS and Genie repository on GitHub.

There are many other use-cases and in this GitHub repository. You may wish to explore other folders and the various other use-cases. See the following web sites for more information:

- Search for: " NetDevOps validation using Cisco pyATS | Genie for network engineers: no coding necessary"
- Cisco GitHub: https://github.com/CiscoTestAutomation