## Lab 18: Automated Testing Using pyATS and Genie

### Case Study

Great Western Bank has enhanced its IT infrastructure monitoring and testing by adopting automation, resulting in improved productivity and increased customer satisfaction. Before implementing this automation methodology, network engineers were performing all tasks manually within the data centers. Given the bank’s numerous data centers distributed across various regions, it was increasingly difficult for the IT team to manage operations manually and efficiently.

### Business Challenge

Great Western Bank initiated its technology overhaul by replacing its legacy network testing methodology with an automated solution. As the bank’s network spans multiple data centers, cloud platforms, and geographical locations, it became increasingly difficult for network engineers to manually configure, monitor, and troubleshoot such a complex environment—resulting in inconsistent outcomes and operational delays.

The bank's IT department sought a Python-based framework capable of automating the testing of network devices and systems. To address this need, the company engaged you—an experienced Cisco certified DevNet Associate—to design and implement a solution for automating network device configurations using a Python-based framework.

### Solution

As a certified Cisco DevNet Associate, you provide a solution for a company to use pyATS and Genie for the automation and testing of Cisco network devices in a data center. pyATS PythonATS is a powerful, data-driven testing framework for scalable and reusable test automation that is ideal for Agile development. Genie, built on top of pyATS, extends its capabilities for network environments by providing device connectivity, parsers, platform-agnostic feature models (e.g., OSPF, BGP), reusable test cases, and a YAML-driven test runner for efficient and structured testing.

1. Create a Python virtual environment.
2. Use the pyATS testing library.
3. Use Genie to parse IOS command output.
4. Use Genie to compare configurations.

|  |
| --- |
| **// Create a Python Virtual Environment**  1. Execute the following command: **mkdir labs/devnet-src/pyats && cd labs/devnet-src/pyats** to create a pyats directory and go inside that directory. The **&&** characters are used to combine two commands in one line.    2. Execute the following command: **python3 -m venv csr1kv** to create a new Python virtual environment. This command will create the directory **csr1kv** in the **pyats** directory.    3. Execute the **cd csr1kv** command to change a directory. Then execute the **ls -l** command to list all the files inside the csr1kv directory. Notice that the **bin** subdirectory and the **pyvenv.cfg** files that were created.    4. Execute the following command: **cat pyvenv.cfg** to view the contents of a file. Note that this file corresponds to the **/usr/bin** directory where your Python installation is located.    5. Execute the following command: **ls -1 /usr/bin/python\*.** A symbolic link (symlink) is a type of file that references another file or directory. To further understand the venv and how it employs symbolic links, list the Python files in the /usr/bin directory specified in the **pyvenv.cfg** file. Use the **ls** number one option (-1) to list each file on a single line.    6. Execute the **ls -l bin** command to review the contents of the venv-created bin. Notice that this subfolder contains two files, both of which are symlinks. Under this situation, it points to the Python binaries under **/usr/bin**. Symlinks are used to connect libraries and ensure that files have consistent access to these files without having to transfer or generate a copy of the original file.    7. Execute the following command: **source bin/activate** to activate the Python virtual environment. |

|  |
| --- |
| **// Use the pyATS Testing Library**  1. Execute the **pip3 install pyats[full]** command to install the pyATS Python library it will take a few minutes to complete the installation.    2. Execute the **pyats --help** command to verify that the pyATS Python library installed successfully.    3. Execute the **git clone https://github.com/CiscoTestAutomation/examples** command to clone the Github pyATS example scripts repository, **CiscoTestAutomation**.    4. Execute the **ls -l** command to verify that the copy of files was successful. Notice there is a new subdirectory **examples**.    5. Execute the **ls -l** **examples** command to list the files in the examples. Notice there is a subdirectory, **basic**, along with several other files.    6. Execute the **ls -l examples/basic** command to list the files in this **basic** subdirectory. This is where you will find the scripts you will need in the following stage.    A pyATS script is a Python file in which pyATS tests are specified. It may be launched as a standalone Python script file, with output solely to the terminal window. Alternatively, one or more pyATS scripts can be built into a "job" and executed as a batch using the pyATS EasyPy module. EasyPy allows many scripts to run in parallel, collects logs in one location, and serves as a central point for injecting changes into the topology under test.  7. Execute the following command: **cat examples/basic/basic\_example\_script.py | more** to view the contents of a Python script.    8. Execute the following command: **pyats run job examples/basic/basic\_example\_job.py** to launch the basic test case, run pyATS manually using the pyATS job and script files. This will ensure that the pyATS job and script files are working properly.    9. Execute the following command: **genie create testbed interactive --output yaml/testbed.yml --encode-password** to create testbed YAML file. Using the **--output** argument, a **testbed.yml** file will be created in the **yaml** directory. The directory will be immediately generated. The **--encode-password** argument encodes the passwords in the YAML file. The parameter **interactive** indicates that you will be given a series of questions. Answer "no" to the first three questions. To generate the **testbed.yaml** file, give the following answers.   * Match the device hostname (in this case, **CSR1kv**) * Enter your CSR1kv IPv4 address * The local username for SSH is **cisco** * The default password for SSH is **cisco123!** * Leave blank in enable password. There is no privileged password set up on the router * Use SSH protocol with the router's anticipated key exchange group * The router's operating system (OS)     10. Execute the **cat yaml/testbed.yml** command to view the configuration of **testbed.yml** inside the **yaml** directory. Take note of your entries in the YAML file. Your SSH password has been encrypted, and the enable password will "ASK" the user to enter the password if it is necessary.    11. In the Cisco CSR router, execute the command **show ip interface brief** from privileged exec mode. Your address may be changed to anything other than 192.168.56.101. Take note of the IPv4 address for your CSR1kv VM. You will utilize it later in the lab.    12. Execute the following command: **genie parse "show ip interface brief" --testbed-file yaml/testbed.yml --devices CSR1kv** torun Genie using testbed.yml file to convert the unstructured output of the **show ip interface** **brief** command to structured JSON. This command contains the IOS command to be processed, the YAML testbed file, and the selected device in the testbed file **CSR1kv**.    13. Execute the following command: **genie parse "show version" --testbed-file yaml/testbed.yml --devices CSR1kv** to parse unstructured output from the show version command into structured JSON. |

|  |  |
| --- | --- |
| **// Use Genie to Parse IOS Command Output**  1. Execute the following commands **interface gig 1** and **ipv6 address 2001:db8:acad:56::101/64** in the Cisco CSR router to assign the IPv6 address to an interface.    2. Execute the following command: **genie parse "show ipv6 interface gig 1" --testbed-file yaml/testbed.yml --devices CSR1kv --output verify-ipv6-1** to parse the unstructured output of the **show ipv6 interface** command into structured JSON. The **--output** argument directs the output to the directory **verify-ipv6-1**. Notice in the result that Genie indicates that two files were produced.    3. Execute the following command: **ls -l verify-ipv6-1** to list the files that Genie generated inside the directory **verify-ipv6-1**. Notice that two files have been generated with similar names, one ending in **\_console.txt** and the other in **\_parsed.txt**. Each file's name includes both the device name and the IOS command used by the Genie parse command.    4. Execute the **cat verify-ipv6-1/CSR1kv\_show-ipv6-interface-gig-1\_console.txt** command to view the contents of **the \_console.txt** file. Take note of both the IPv6 global unicast address that you specified and the automated EUI-64 link-local address.    5. Execute the **cat verify-ipv6-1/CSR1kv\_show-ipv6-interface-gig-1\_parsed.txt** command to view the contents of the **\_parsed.txt** file. This is the parsed JSON file of the **show ipv6 interface gig 1** command.    6. Execute the below-provided commands in the Cisco CSR router to modify the IPv6 link-local address.   |  | | --- | | enable  configure terminal  interface gigabitethernet 1  ipv6 address fe80::56:1 link-local |     7. Execute the following command: **genie parse "show ipv6 interface gig 1" --testbed-file yaml/testbed.yml --devices CSR1kv --output verify-ipv6-2** to parse the unstructured output of the **show ipv6 interface** command into structured JSON. The **--output** argument directs the output to the directory **verify-ipv6-2**.    8. Execute the **ls -l verify-ipv6-2** command to list the files that Genie generated in the directory **verify-ipv6-2**. These are comparable to the two files you prepared before setting the IPv6 link-local address.    9. Execute the following commands: **cat verify-ipv6-2/CSR1kv\_show-ipv6-interface-gig-1\_console.txt** and **cat verify-ipv6-2/CSR1kv\_show-ipv6-interface-gig-1\_parsed.txt** to view the contents of a file. The changes are highlighted in the output below. |

|  |
| --- |
| **// Use Genie to Compare Configurations**  1. Execute the following command: **genie diff verify-ipv6-1 verify-ipv6-2** to have Genie identify the differences between the two processed JSON files. Notice that the output indicates where you may locate Genie's comparisons. In this situation, the first filename refers to the prior configuration, while the second represents the current configuration.    2. Execute the following command: **cat ./diff\_CSR1kv\_show-ipv6-interface-gig-1\_parsed.txt** to show the contents of the file with the differences. The plus **+** sign indicates additions and the minus **-** sign indicates what was removed.    3. When you complete this lab execute the following command: **deactivate** to deactivate your Python virtual environment. |