Travaux pratiques - Tests automatisés utilisant PYATS et Genie

Objectifs

Partie 1 : Lancer la DEVASC VM

Partie 2 : Créer un environnement virtuel Python

Partie 3 : Utiliser la bibliothèque de tests PyATS

Partie 4 : Utiliser Genie pour analyser la sortie de la commande IOS

Partie 5 : Utiliser Genie pour comparer les configurations

Partie 6 : Nettoyage du Labo et enquêtes complémentaires

Contexte/scénario

Dans ce TP, vous explorerez les principes fondamentaux PYATS (prononcé "py" suivi de chaque lettre individuellement, "A", "T". "S") et Genie. L'outil PYATS est un écosystème de test de bout en bout, spécialisé dans les tests basés sur les données et réutilisables, et conçu pour être adapté aux itérations de développement agiles et rapides. Extensible par sa conception, PyATS permet aux développeurs de commencer par des cas de test petits, simples et linéaires, et d'évoluer vers des suites de test volumineuses, complexes et asynchrones.

Genie étend et construit sur PyATS pour être utilisé dans un environnement de réseau. Examples of features Genie provides include:

- connectivité des périphériques, analyseurs et API
- Modèles d'objets Python indépendants de la plate-forme pour des fonctionnalités telles que OSPF et BGP
- pool de cas de test réutilisables
- Moteur d'essai piloté par YAML

Ressources requises

- 1 PC avec système d'exploitation de votre choix
- Boîte virtuelle ou VMWare
- Machine virtuelle DEVASC
- Machine virtuelle CSR1kv

Instructions

Partie 1 : Lancer la machine virtuelle DEVASC

Si vous n'avez pas encore terminé le **TP - Installez l'environnement de laboratoire de la machine virtuelle** , faites-le maintenant. Si vous avez déjà terminé ce TP, lancez la machine virtuelle DEVASC maintenant.

Partie 2 : Créer un environnement virtuel Python

Dans cette partie, vous allez créer un environnement virtuel Python appelé environnement virtuel Python ou "venv".

Étape 1: Ouvrez un terminal dans le DEVASC-LABVM.

Double-cliquez sur l'icône de l'émulateur de terminal sur le bureau pour ouvrir une fenêtre de terminal.

Étape 2: Création d'environnement virtuel Python (venv).

L'outil PYATS est le mieux installé pour un travail individuel dans un venv. Un environnement venv est copié à partir de votre environnement de base Python mais gardé séparé de celui-ci. Cela vous permet d'éviter d'installer des logiciels susceptibles de modifier définitivement l'état général de votre ordinateur. L'environnement venv a été abordé en détail dans le **TP - Explorer Python Development Tools** plus tôt dans le cours.

a. Créez un répertoire **pyats** et changez vers ce répertoire. Vous pouvez utiliser les caractères **&&** pour combiner les deux commandes sur une seule ligne.

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

b. Créez un nouvel environnement virtuel Python qui crée le répertoire **csr1kv** dans le répertoire **pyats**.

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

Remarque : Vous pouvez également utiliser un point "**.**" au lieu d'un nom de répertoire si vous voulez créer un environnement venv dans le répertoire courant.

Étape 3: Passez en revue votre environnement virtuel Python (venv).

a. Changez les répertoires à votre nouveau répertoire "cible" **csr1kv** et listez les fichiers. Venv crée une arborescence de répertoires autonome (test-project) qui contient une installation Python pour une version particulière de Python, ainsi qu'un certain nombre de paquets supplémentaires. Il crée également un sous-répertoire bin contenant une copie du binaire Python.

Notez en particulier le sous-répertoire **bin** et les fichiers **pyvenv.cfg** qui ont été créés.

```
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. Examinez le contenu du fichier **pyvenv.cfg**. Notez que ce fichier pointe vers l'emplacement de votre installation Python dans /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. Un lien symbolique (également connu sous le nom de lien symbolique) est un type spécial de fichier qui sert de référence à un autre fichier ou répertoire. Pour mieux comprendre le venv et comment il utilise les liens symboliques, listez les fichiers Python dans le répertoire /usr/bin référencé dans le fichier **pyvenv.cfg**. Utilisez l'option **ls** numéro un (-1) pour répertorier les fichiers sur une ligne.

```
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. Examinez maintenant le contenu du sous-répertoire **bin** créé par venv. Notez qu'il y a deux fichiers dans ce sous-répertoire, qui sont tous deux des liens symboliques. Dans ce cas, il s'agit d'un lien vers les binaires Python dans **/usr/bin**. Les liens symboliques sont utilisés pour lier des bibliothèques et s'assurer que les fichiers y ont un accès cohérent à ces fichiers sans avoir à déplacer ou à créer une copie du fichier d'origine. Il y a aussi un fichier, **activate**, qui sera discuté ensuite.

```
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. Lancez l'environnement virtuel à l'aide de **bin/activate**. Notez que votre invite est maintenant précédée de **(csr1kv)**. Toutes les commandes effectuées à partir de ce point sont dans ce venv.

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

Remarque : La commande **deactivate** est utilisée pour quitter l'environnement venv et revenir à l'environnement shell normal.

Partie 3 : Utiliser la bibliothèque de tests PyATS

Dans cette partie, vous allez utiliser PyATS, une bibliothèque de tests python.

Étape 1: Installation de PYATS.

Installez PyATS à l'aide de **pip3**. This will take a few minutes. Lors de l'installation, vous pouvez voir quelques erreurs. Ceux-ci peuvent généralement être ignorés tant que PYATS peut être vérifié comme indiqué à l'étape suivante.

```
(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$
```

Étape 2: Verifying pyATS.

Vérifiez que PYATS a bien été installé à l'aide de la commande **pyats —help**. Notez que vous pouvez obtenir de l'aide supplémentaire sur n'importe quelle commande pyats avec la commande **pyats <command> --help**.

```
(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$
```

Étape 3: Cloner et examiner les exemples de scripts PyATS de GitHub.

a. Cloner le dépôt des exemples de scripts Github PyATS **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. Vérifiez que la copie a réussi en listant les fichiers dans le répertoire courant. Notez qu'il existe un nouvel **exemple**de sous-répertoire.

```
(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. Répertorier les fichiers dans le sous-répertoire des **exemples**. Notez qu'il y a un sous-répertoire, **basique**, avec plusieurs autres fichiers.

```
(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. Répertorier les fichiers dans ce sous-répertoire **de base**. Il s'agit de l'emplacement des scripts que vous utiliserez à l'étape suivante.

```
(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$
```

Étape 4: Examinez les fichiers de script de base.

La syntaxe de déclaration de test pour PyATS est basée sur des frameworks de test d'unité Python populaires comme pytest. Il prend en charge les instructions de test de base, telles que l'affirmation qu'une variable a une valeur donnée, et avec la fourniture explicite de résultats via des API spécifiques.

a. Le script Python que vous allez utiliser est **basic_example_script.py**. Affichez le contenu du script Python à l'aide de la commande **cat**. Cannez-le vers **more** si vous voulez l'afficher un écran ou une ligne à la fois. Notez que ce script contient les sections suivantes, telles que mises en évidence dans la sortie ci-dessous :

- Un bloc d'installation commun
- Blocs de test multiples
- Un bloc de nettoyage commun

Ces blocs contiennent des instructions qui préparent et/ou déterminent l'état de préparation de la topologie de test (un processus qui peut inclure l'injection de problème), exécutent des tests, puis renvoient la topologie à un état connu.

Les blocs de test - souvent appelés dans la documentation PyATS sous le nom de cas de test - peuvent contenir chacun plusieurs tests, avec leur propre code d'installation et de nettoyage. Les meilleures pratiques suggèrent, cependant, que la section Nettoyage commune, à la fin, soit conçue pour l'idempuissance, ce qui signifie qu'elle devrait vérifier et restaurer toutes les modifications apportées par Setup and Test, et restaurer la topologie à son état d'origine souhaité.

Remarque : Bien qu'il ne soit pas nécessaire de comprendre le code, vous trouverez utile de lire les commentaires dans le script Python.

```
(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 .sous-section
    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))

######################################################################
### SECTION DE TESTCAS ###
######################################################################

# 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 .sous-section
    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$
```

Un script PyATS est un fichier Python où les tests PyATS sont déclarés. Il peut être exécuté directement en tant que fichier de script Python autonome, générant une sortie uniquement vers votre fenêtre de terminal. Alternativement, un ou plusieurs scripts PyATS peuvent être compilés dans un "job" et exécutés ensemble sous forme de lot, via le module PyATS EasyPy. EasyPy permet l'exécution parallèle de plusieurs scripts, collecte les journaux à un seul endroit et fournit un point central à partir duquel injecter des modifications à la topologie testée.

b. Utilisez **cat** pour afficher votre fichier de travail PyATS, **pyats_sample_job.py**. Notez les instructions sur la façon d'exécuter ce fichier, mises en évidence ci-dessous.

```
(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$
```

Étape 5: Exécutez PyATS manuellement pour appeler le cas de test de base.

a. À l'aide des fichiers de travail et de script PyATS, exécutez PyATS manuellement pour appeler le cas de test de base. Cela vérifiera que le travail PyATS et les fichiers de script fonctionnent correctement. Les informations contenues dans la sortie dépassent le cadre de ce laboratoire, mais vous remarquerez que le travail et le script ont passé toutes les tâches requises.

Remarque : La sortie ci-dessous a été tronquée. Le référentiel Cisco Test Automation sur GitHub est sujet à modification, qui inclut les fichiers de travail et de scripts PyATS. Votre résultat est sujet à changement, mais ne devrait pas affecter votre résultat. Par exemple, un échec intentionnel a été ajouté au fichier **basic_example_script.py**. C'est un échec intentionnel et ne cause aucun problème. C'est un exemple que les dépôts sont dynamiques. Il s'agit de l'une des lignes surlignées ci-dessous.

```
(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: +------------------------------------------------
----------------------------+
2020-05-31T 17:10:20 :%EASYPY-INFO : | Détails du résultat de la tâche |
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!

Conseil d'expert
-------
   Utilisez la commande suivante pour afficher vos journaux localement :
       pyats logs view

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

Partie 4 : Utiliser Genie pour analyser la sortie de la commande IOS

Dans cette partie, vous allez utiliser Génie pour prendre une sortie IOS non structurée et l'analyser dans la sortie JSON.

Remarque : toutes les commandes IOS ne sont pas prises en charge. La documentation complète de Génie peut être consultée à l'adresse suivante: https://developer.cisco.com/docs/genie-docs/

Étape 1: Créez un fichier YAML testbed.

Les outils PyATS et Genie utilisent un fichier YAML pour savoir à quels périphériques se connecter et quelles sont les informations d'identification appropriées. Ce fichier est connu sous le nom de fichier testbed. Genie inclut des fonctionnalités intégrées pour construire le fichier de banc d'essai pour vous.

a. Entrez la commande **genie —help** pour voir toutes les commandes disponibles. Pour obtenir de l'aide supplémentaire sur n'importe quelle commande **<command>**, utilisez le paramètre, comme indiqué ci-dessous, pour la commande **create**. Notez que **testbed** est l'une des options de la commande **create**.

```
(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. Pour créer votre fichier YAML testbed, entrez la commande ci-dessous. Le paramètre **—output** va créer un fichier .yml **testbed.yml** dans un répertoire nommé **yaml**. Le répertoire sera automatiquement créé. Le paramètre **—encode-password** encodera les mots de passe dans le fichier YAML. Le paramètre **interactif** signifie que vous serez posé une série de questions. Répondez non aux trois premières questions. Et puis fournissez les réponses suivantes pour créer le fichier **testbed.yaml**.

- **Nom d'hôte du périphérique** - Ce nom doit correspondre au nom d'hôte du périphérique, qui est CSR1kv pour ce laboratoire .
- **Adresse IP** - Cette adresse doit correspondre à votre adresse IPv4 CSR1kv que vous avez découverte plus tôt dans ce laboratoire. Le montre ici est **192.168.56.101**.
- **Nom d'utilisateur** - Ceci est le nom d'utilisateur local utilisé pour ssh, qui est **cisco**.
- **Mot de passe par défaut** - Ceci est le mot de passe local utilisé pour ssh, qui est **cisco123!**.
- **Activer le mot de passe** - Laissez vide. Aucun mot de passe privilégié n'est configuré sur le routeur.
- **Protocole** - SSH avec le groupe d'échange de clés attendu par le routeur.
- **OS** - Le système d'exploitation sur le routeur.

```
(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. Utilisez **cat** pour afficher le fichier **testbed.yml** dans le répertoire **yaml**. Notez vos entrées dans le fichier YAML. Votre mot de passe SSH est crypté et le mot de passe d'activation "DEMANDE" à l'utilisateur d'entrer le mot de passe s'il en a besoin.

```
(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$
```

Étape 2: Utilisez Génie pour analyser la sortie de la commande show ip interface brief dans JSON.

a. Si vous n'avez pas encore terminé le **TP - Installez la machine virtuelle CSR1kV**, faites-le maintenant. Si vous avez déjà terminé ce laboratoire, lancez la machine virtuelle CSR1kv maintenant.

b. Dans la machine virtuelle CSR1kv, entrez la commande **show ip interface brief** à partir du mode exec privilégié. Votre adresse peut être incrémentée à une autre adresse que 192.168.56.101. Notez l'adresse IPv4 de votre machine virtuelle CSR1kv. Vous l'utiliserez plus tard dans le labo.

```
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. En utilisant votre fichier YAML testbed, appelez Génie pour analyser la sortie non structurée de la commande **show ip interface brief** en JSON structuré. Cette commande inclut la commande IOS à analyser (**show ip interface brief**), le fichier testbed YAML (**testbed.yml**) et le périphérique spécifié dans le fichier testbed (**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
```

Étape 3: Utilisez Genie pour analyser la sortie de la commande show version dans JSON.

Pour un autre exemple, analysez la sortie non structurée de la commande **show version** en JSON structuré.

```
(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$
```

Partie 5 : Utiliser Genie pour comparer les configurations

Comme vous l'avez vu, Génie peut être utilisé pour analyser les commandes show en json structuré. Genie peut également être utilisé pour:

- Prenez des instantanés de configs chaque année et faites des comparaisons entre eux
- Automatisez les déploiements de test sur un environnement virtuel pour les tester avant le déploiement en production
- Pour résoudre les problèmes de configuration en effectuant des comparaisons entre les périphériques

Dans les parties 5 et 6, vous verrez comment faire une comparaison entre deux sorties différentes.

Étape 1: Ajoutez une adresse IPv6 à CSR1kv.

a. Sur la machine virtuelle CSR1kv, ajoutez l'adresse IPv6 suivante:

```
CSR1kV (config) # interface gig 1
CSR1kv(config-if)# ipv6 address 2001:db8:acad:56::101/64
```

Étape 2: Utilisez Genie pour vérifier la configuration et analyser la sortie dans JSON.

a. Analyser la sortie non structurée de la commande **show ipv6 interface** en JSON structuré. Utilisez le paramètre **—output** pour envoyer la sortie vers un répertoire **verify-ipv6-1**. Notez dans la sortie que Genie vous dit que deux fichiers ont été créés.

```
(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. Répertorier les fichiers créés par Génie dans le répertoire **verify-ipv6-1**. Notez qu'il y a eu deux fichiers créés avec les noms similaires, mais l'un se terminant par **_console.txt** et l'autre dans **_parsed.txt**. Le nom de chaque fichier inclut le nom du périphérique et la commande IOS utilisée dans la commande Genie parse.

```
(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. Utilisez **cat** pour examiner le contenu du fichier **_console.txt**. Notez à la fois l'adresse de monodiffusion globale IPv6 que vous avez configurée et une adresse de liaison locale EUI-64 automatique.

```
(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. Utilisez **cat** pour examiner le contenu du fichier **_parsed.txt**. Il s'agit du fichier JSON analysé de la commande **show ipv6 interface gig 1**.

```
(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$
```

Étape 3: Modifiez l'adresse Lien-Local IPv6.

Sur CSR1kv VM, ajoutez l'adresse IPv6 suivante :

```
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
```

Étape 4: Utilisez Genie pour vérifier la configuration et analyser la sortie dans JSON.

a. Analyser la sortie non structurée de la commande **show ipv6 interface** en JSON structuré. Utilisez le paramètre **—output** pour envoyer la sortie vers un autre répertoire **verify-ipv6-2**. Vous pouvez utiliser l'historique des commandes pour rappeler la commande précédente (flèche vers le haut). Assurez-vous simplement de changer le **1** à un **2** pour créer un nouveau répertoire **verify-ipv6-2** .

```
(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. Répertorier les fichiers créés par Génie dans le répertoire **verify-ipv6-2**. Ces fichiers sont similaires aux deux fichiers que vous avez créés avant de modifier l'adresse lien-local IPv6.

```
(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 728 May 31 20:04 CSR1kv_show-ipv6-interface-gig-
1_console.txt
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

c. Utilisez le **cat** pour examiner le contenu de chaque fichier. Les modifications sont mises en surbrillance dans la sortie ci-dessous.

```
(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$
```

Étape 5: Utilisez Genie pour comparer la différence entre les configurations.

Dans l'étape précédente, il est assez facile de trouver le changement de l'adresse lien-local IPv6. Mais supposons que vous cherchiez un problème dans une configuration complexe. Peut-être, vous essayez de trouver une différence entre une configuration OSPF sur un routeur qui reçoit les routes appropriées et un autre routeur qui ne l'est pas, et vous voulez voir les différences dans leurs configurations OSPF. Ou peut-être, vous essayez de repérer la différence dans une longue liste d'instructions ACL entre deux routeurs qui sont censés avoir des stratégies de sécurité identiques. Genie peut faire la comparaison pour vous et faciliter la recherche des différences.

a. Utilisez la commande suivante pour que Genie trouve les différences entre les deux fichiers JSON analysés. Notez que la sortie vous indique où vous pouvez trouver les comparaisons de Genie. Dans ce cas, le premier nom de fichier est la configuration précédente et le second nom de fichier est la configuration actuelle.

```
(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. Utilisez **cat** pour afficher le contenu du fichier avec les différences. Le signe plus "+" indique les ajouts et le signe moins "-" indique ce qui a été supprimé.

```
(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$
```

Partie 6 : Nettoyage de labo et recherches approfondies

Dans cette partie, vous allez désactiver votre venv Python et enquêter sur d'autres cas d'utilisation de Génie.

Étape 1: Désactivez votre environnement virtuel Python.

Une fois ce labo terminé, vous pouvez désactiver votre environnement virtuel Python à l'aide de la commande **deactivate**. Notez que votre invite n'est plus précédée de "(csr1kv)".

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

Étape 2: Explorez d'autres cas d'utilisation de PyATS et de Genie.

Auparavant dans ce laboratoire, vous avez cloné le dossier **exemples** à partir du dépôt Cisco Test Automation avec PyATS et Genie sur GitHub.

Il y a beaucoup d'autres cas d'utilisation et dans ce dépôt GitHub. Vous pouvez explorer d'autres dossiers et les divers autres cas d'utilisation. Consultez les sites Web suivants pour plus d'informations :

- Rechercher : "Validation NetDevOps avec Cisco PyATS | Génie pour les ingénieurs réseau : pas de codage nécessaire »
- Cisco GitHub: https://github.com/CiscoTestAutomation