UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE CIENCIAS

ÁREA DE CIENCIA DE LA COMPUTACIÓN

9° LABORATORIO - CC312

- **TÍTULO:** Pruebas automatizadas usando pyATS y Genie

- **ALUMNO:**

JHONATAN POMA MARTINEZ 20182729F

- **PROFESORES:** YURI JAVIER.,CCOICCA PACASI

2023

Objetivos
Parte 1: Iniciar la Máquina Virtual DEVASC.
Parte 2: Crear un entorno virtual de Python.
Parte 3: Utilizar la biblioteca de pruebas pyATS.
Parte 4: Usar Genie para analizar la salida del comando IOS.
Parte 5: Utilizar Genie para comparar configuraciones.
Parte 6: Limpieza de laboratorio e investigación adicional.

Introduccion:

pyATS, abreviatura de "Python Automation Framework (Framework de Automatización en Python)", es una biblioteca de código abierto desarrollada por Cisco Systems. Proporciona un conjunto de herramientas y API para automatizar tareas de prueba y verificación en dispositivos de red.
pyATS se basa en el lenguaje de programación Python y proporciona una amplia gama de funcionalidades para facilitar la automatización de pruebas en entornos de red. Permite interactuar con dispositivos de red, como enrutadores y conmutadores, para realizar pruebas de conectividad, recopilación de datos, verificación de configuraciones, pruebas de rendimiento y más.
Esta biblioteca proporciona una interfaz de programación fácil de usar, lo que permite a los usuarios crear scripts y casos de prueba personalizados para automatizar sus flujos de trabajo de pruebas de red. pyATS también ofrece características como la administración de topología de red, ejecución de pruebas en paralelo, generación de informes y análisis de resultados.

Genie es framework de automatización de pruebas desarrollado por Cisco Systems. Es una parte integral de la biblioteca pyATS y está diseñado específicamente para la automatización de pruebas en dispositivos de red.
Genie nos permite crear scripts y casos de prueba para realizar pruebas de verificación, validación y solución de problemas en dispositivos de red.
Una de las características clave de Genie es su capacidad para abstraer la complejidad de los comandos y configuraciones específicas del dispositivo. Proporciona una capa de abstracción que permite a los usuarios interactuar con dispositivos de diferentes fabricantes y modelos de manera uniforme, lo que facilita la escritura de scripts de prueba portables y reutilizables.
Genie también ofrece una amplia gama de capacidades de prueba, incluyendo verificación de configuración, comprobación de estado de interfaces, recopilación de datos de rendimiento, simulación de tráfico de red y mucho más. Además, Genie facilita la generación de informes detallados y el análisis de resultados de pruebas.

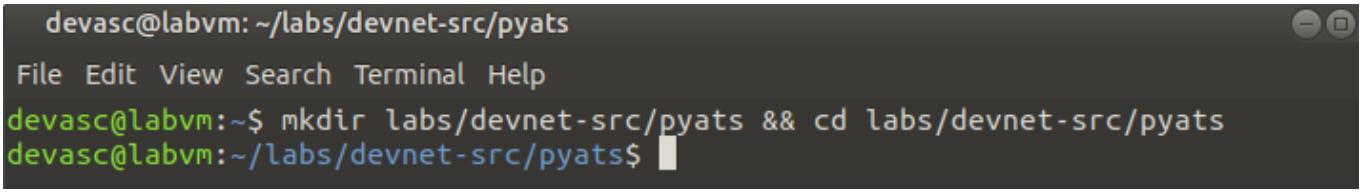
venv (un entorno virtual) es un entorno aislado y autónomo que nos permite instalar bibliotecas y dependencias específicas para un proyecto de software, evitando conflictos con otros proyectos y el sistema operativo subyacente.

Parte 1: Iniciar la Máquina Virtual DEVASC.

Iniciado.

Parte 2: Crear un entorno virtual de Python.

Creamos un directorio **pyats** y cambiamos a ese directorio. Puede utilizar los caracteres && para combinar los dos comandos en una línea.



Cree un nuevo entorno virtual (venv) de Python que cree el directorio csr1kv en el directorio pyats.

```
devasc@labvm:~/labs/devnet-src/pyats$ python3 -m venv csr1k
devasc@labvm:~/labs/devnet-src/pyats$
```

Revisamos el entorno virtual de Python (venv) que hemos creado, veremos que se creo un subdirectorio **bin** que contiene una copia del binario de Python.

```
devasc@labvm:~/labs/devnet-src/pyats$ python3 -m venv csr1kv
devasc@labvm:~/labs/devnet-src/pyats$ cd csr1kv
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls
bin  include  lib  lib64  pyvenv.cfg  share
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l
total 20
drwxrwxr-x 2 devasc devasc 4096 Jun 15 03:10 bin
drwxrwxr-x 2 devasc devasc 4096 Jun 15 03:10 include
drwxrwxr-x 3 devasc devasc 4096 Jun 15 03:10 lib
lrwxrwxrwx 1 devasc devasc    3 Jun 15 03:10 lib64 -> lib
-rw-rw-r-- 1 devasc devasc   69 Jun 15 03:10 pyvenv.cfg
drwxrwxr-x 3 devasc devasc 4096 Jun 15 03:10 share
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

El archivo **pyvenv.cfg.** Observamos que este archivo apunta a la ubicación de su instalación de Python en /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$
```

Un enlace simbólico (también conocido como "symlink") es un tipo especial de archivo que sirve como referencia a otro archivo o directorio. Enliste los archivos de Python en el directorio /usr/bin al que se hace referencia pyvenv.cfg.

```
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$
```

Ahora examine el contenido del subdirectorio bin creado por venv. Observe que hay **dos** archivos en este subdirectorio, los cuales son **enlaces simbólicos.** En este caso, es un enlace a los binarios de Python en /usr/bin. Los enlaces simbólicos se utilizan para vincular bibliotecas y asegurarse de que los archivos tienen acceso constante a estos archivos sin tener que mover o crear una copia del archivo original. También hay un archivo, activate, que se discutirá a continuación.

```
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l bin
total 44
-rw-r--r-- 1 devasc devasc 2225 Jun 15 03:10 activate
-rw-r--r-- 1 devasc devasc 1277 Jun 15 03:10 activate.csh
-rw-r--r-- 1 devasc devasc 2429 Jun 15 03:10 activate.fish
-rw-r--r-- 1 devasc devasc 8471 Jun 15 03:10 Activate.ps1
-rwxrwxr-x 1 devasc devasc  267 Jun 15 03:10 easy_install
-rwxrwxr-x 1 devasc devasc  267 Jun 15 03:10 easy_install-3.8
-rwxrwxr-x 1 devasc devasc  258 Jun 15 03:10 pip
-rwxrwxr-x 1 devasc devasc  258 Jun 15 03:10 pip3
-rwxrwxr-x 1 devasc devasc  258 Jun 15 03:10 pip3.8
lrwxrwxrwx 1 devasc devasc    7 Jun 15 03:10 python -> python3
lrwxrwxrwx 1 devasc devasc   16 Jun 15 03:10 python3 -> /usr/bin/python3
devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Iniciamos el entorno virtual usando **bin/activate**. Observe que su mensaje está precedido por **(csr1kv)**.

Todos los comandos hechos a partir de este punto están dentro de este venv "csr1kv".

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

Para desactivar el venv "cs1kv" usamos el comando "deactivate".

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

Parte 3: Utilizar la biblioteca de pruebas pyATS.

En esta parte, usaremos **pyATS**, una biblioteca de pruebas de Python.

Instalamos PyATs usando pip3.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ pip3 install pyats[full]
Collecting pyats[full]
  Downloading pyats-23.5-cp38-cp38-manylinux2014_x86_64.whl (4.4 MB)
     |████████████████████████████████| 4.4 MB 3.0 MB/s
Collecting pyats.reporter<23.6.0,>=23.5.0
  Downloading pyats.reporter-23.5-cp38-cp38-manylinux2014_x86_64.whl (3.7 MB)
     |████████████████████████████████| 3.7 MB 217 kB/s
```

Verificamos que este instalado pyats.

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

Commands:
    clean               runs the provided clean file
    create              create scripts and libraries from template
    develop             Puts desired pyATS packages into development mode
    diff                Command to diff two snapshots saved to file or directory
    dnac                Command to learn DNAC features and save to file (Prototy
pe)
    learn               Command to learn device features and save to file
    logs                command enabling log archive viewing in local browser
    migrate             utilities for migrating to future versions of pyATS
    parse               Command to parse show commands
    run                 runs the provided script and output corresponding result
s.
    secret              utilities for working with secret strings.
    shell               enter Python shell, loading a pyATS testbed file and/or
pickled data
    undevelop           Removes desired pyATS packages from development mode
    validate            utilities that help 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$
```

Paso 3: Clonaremos y examinaremos los scripts de ejemplo pyATS de GitHub.

Clonar el repositorio de scripts de ejemplo Github pyATS CiscoTestAutomation.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ git clone https://github.c
om/CiscoTestAutomation/examples
Cloning into 'examples'...
remote: Enumerating objects: 1414, done.
remote: Counting objects: 100% (262/262), done.
remote: Compressing objects: 100% (156/156), done.
remote: Total 1414 (delta 137), reused 150 (delta 87), pack-reused 1152
Receiving objects: 100% (1414/1414), 1.17 MiB | 1.93 MiB/s, done.
Resolving deltas: 100% (709/709), done.
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Compruebe que la copia se ha realizado correctamente listando los archivos en el directorio actual. Observamos que hay un nuevo subdirectorio **example**.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l
total 24
drwxrwxr-x  3 devasc devasc 4096 Jun 15 03:39 bin
drwxrwxr-x 24 devasc devasc 4096 Jun 15 04:52 examples
drwxrwxr-x  2 devasc devasc 4096 Jun 15 03:10 include
drwxrwxr-x  3 devasc devasc 4096 Jun 15 03:10 lib
lrwxrwxrwx  1 devasc devasc    3 Jun 15 03:10 lib64 -> lib
-rw-rw-r--  1 devasc devasc   69 Jun 15 03:10 pyvenv.cfg
drwxrwxr-x  3 devasc devasc 4096 Jun 15 03:10 share
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Enumeramos los archivos del subdirectorio **example.**

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l examples
total 96
drwxrwxr-x  3 devasc devasc  4096 Jun 15 04:52 abstraction_example
drwxrwxr-x  3 devasc devasc  4096 Jun 15 04:52 basic
drwxrwxr-x  3 devasc devasc  4096 Jun 15 04:52 blitz
drwxrwxr-x 31 devasc devasc  4096 Jun 15 04:52 clean
drwxrwxr-x  7 devasc devasc  4096 Jun 15 04:52 comprehensive
drwxrwxr-x  4 devasc devasc  4096 Jun 15 04:52 connection
drwxrwxr-x  5 devasc devasc  4096 Jun 15 04:52 datafiles
drwxrwxr-x  3 devasc devasc  4096 Jun 15 04:52 feature
drwxrwxr-x  2 devasc devasc  4096 Jun 15 04:52 GenieHarnessHelloWorld
drwxrwxr-x  2 devasc devasc  4096 Jun 15 04:52 groups
drwxrwxr-x  4 devasc devasc  4096 Jun 15 04:52 health
drwxrwxr-x 15 devasc devasc  4096 Jun 15 04:52 libraries
-rw-rw-r--  1 devasc devasc 11357 Jun 15 04:52 LICENSE
drwxrwxr-x  2 devasc devasc  4096 Jun 15 04:52 loop
drwxrwxr-x  2 devasc devasc  4096 Jun 15 04:52 metadata
drwxrwxr-x  2 devasc devasc  4096 Jun 15 04:52 metaparser
drwxrwxr-x  5 devasc devasc  4096 Jun 15 04:52 parsergen
-rw-rw-r--  1 devasc devasc   916 Jun 15 04:52 README.md
drwxrwxr-x  4 devasc devasc  4096 Jun 15 04:52 script_parameters
drwxrwxr-x  4 devasc devasc  4096 Jun 15 04:52 steps
drwxrwxr-x  3 devasc devasc  4096 Jun 15 04:52 tasks
drwxrwxr-x  2 devasc devasc  4096 Jun 15 04:52 uids
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Enumere los archivos de este subdirectorio **basic**. Esta es la ubicación de los scripts que usará en el siguiente paso.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l examples/basic
total 16
-rw-rw-r-- 1 devasc devasc  534 Jun 15 04:52 basic_example_job.py
-rwxrwxr-x 1 devasc devasc 4510 Jun 15 04:52 basic_example_script.py
drwxrwxr-x 2 devasc devasc 4096 Jun 15 04:52 results
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Paso 4: Examine los archivos de script básicos.

El script de Python que usará es **basic_example_script.py**. Observamos que este script contiene las siguientes secciones, tal y como se destaca en la siguiente salida:

- Un bloque de configuración común
- Bloques de prueba múltiples
- Un bloque de limpieza común

Estos bloques contienen instrucciones que preparan y/o determinan la preparación de la topología de prueba (un proceso que puede incluir la inyección de problemas), realizan pruebas y, a continuación, devuelven la topología a un estado conocido.

Los bloques de prueba, a menudo referidos en la documentación de pyATS como los casos de prueba, pueden contener varias pruebas, con su propio código de configuración y limpieza. Las mejores prácticas sugieren, sin embargo, que la sección de limpieza común, al final, esté diseñada para idempotencia, lo que significa que debe verificar y restaurar todos los cambios realizados por Configuración y Prueba, además de restaurar la topología a su estado original y deseado.

Abriendo el script **basic_example_script.py**.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat examples/basic/basic_e
xample_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
```

Utilice cat para mostrar su archivo de trabajo pyATS, pyats_sample_job.py. Observe las instrucciones sobre cómo ejecutar este archivo, resaltadas a continuación.

Abrimos el script **basic_example_job.py**

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat examples/basic/basic_e
xample_job.py
"""
<PYATS_JOBFILE>
"""
# 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$
```

Paso 5: Ejecutar pyATS manualmente para invocar el caso de prueba básico.

Utilizando el trabajo PyATS y los archivos de script, ejecute PyATS manualmente para invocar el caso de prueba básico. Esto verificará que el trabajo pyATS y los archivos de script funcionan correctamente. La información en el resultado está fuera del alcance de este laboratorio, sin embargo, notará que el trabajo y el script pasaron todas las tareas requeridas. A excepción de algunos que fueron hechos agrede.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ pyats run job examples/bas
ic/basic_example_job.py
2023-06-15T05:16:41: %EASYPY-INFO: Starting job run: basic_example_job
2023-06-15T05:16:41: %EASYPY-INFO: Runinfo directory: /home/devasc/.pyats/runinf
o/basic_example_job.2023Jun15_05:16:40.424758
2023-06-15T05:16:41: %EASYPY-INFO: ----------------------------------------
----------------------------------
2023-06-15T05:16:46: %EASYPY-INFO: +----------------------------------------
----------------------------------+
2023-06-15T05:16:46: %EASYPY-INFO: |                                Clean Informat
ion                                |
2023-06-15T05:16:46: %EASYPY-INFO: +----------------------------------------
----------------------------------+
2023-06-15T05:16:46: %EASYPY-INFO: {'bringup': {}, 'cleaners': {}, 'devices': {}
}

2023-06-15T05:16:46: %ATS-INFO: Checking all devices are up and ready is disable
```

.....

```
2023-06-15T05:16:51: %EASYPY-INFO: `-- common_cleanup
                                   PASSED
2023-06-15T05:16:51: %EASYPY-INFO:     `-- clean_everything
                                   PASSED
2023-06-15T05:16:51: %EASYPY-INFO: Sending report email...
2023-06-15T05:16:51: %EASYPY-INFO: Missing SMTP server configuration, or failed
to reach/authenticate/send mail. Result notification email failed to send.
2023-06-15T05:16:52: %EASYPY-INFO: Done!

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

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

Parte 4: Usar Genie para analizar la salida del comando IOS.

Usaremos **Genie** para tomar salida IOS no estructurada y analizarla en salida JSON.

Paso 1: Cree un archivo YAML de banco de pruebas.

Las herramientas pyATS y Genie utilizan un archivo YAML para saber a qué dispositivos conectarse y cuáles son las credenciales adecuadas. Este archivo se conoce como archivo de prueba. Genie incluye funcionalidad integrada para crear el **archivo testbed** para usted.

a. Introduzca el comando genie - -help para ver todos los comandos disponibles. Utilice el parámetro <Commands>, como se muestra a continuación, para el comando create. Observe que el **testbed es una de las opciones para el comando create.**

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

Commands:
    create              Create Testbed, parser, triggers, ...
    develop             Puts desired pyATS packages into development mode
    diff                Command to diff two snapshots saved to file or directory
    dnac                Command to learn DNAC features and save to file (Prototy
pe)
    learn               Command to learn device features and save to file
    parse               Command to parse show commands
    run                 Run Genie triggers & verifications in pyATS runtime envi
ronment
    shell               enter Python shell, loading a pyATS testbed file and/or
pickled data
    undevelop           Removes desired pyATS packages from development mode

General Options:
  -h, --help            Show help

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

Observe que el **testbed** es una de las opciones para el comando create.

" testbed crear un archivo de banco de pruebas automáticamente "

```
(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$
```

Para crear su archivo YAML testbed, ingrese el siguiente comando. El parámetro —output creará un archivo testbed.yml en un directorio llamado yaml. El directorio se creará automáticamente. El parámetro —encode-password codificará las contraseñas en el archivo YAML. El parámetro interactive significa que se le hará una serie de preguntas. Responda "NO" a las tres primeras preguntas. Y luego proporcione las siguientes respuestas para crear el archivo testbed.yaml.

- Nombre de host : CSR1kv
- Dirección IP: Debe coincidir con su dirección CSR1kv IPv4 que descubrió anteriormente en este laboratorio. Aquí se muestra 192.168.56.101.
- Nombre de usuario: Este es el nombre de usuario local utilizado para ssh, que es **cisco**
- Contraseña predeterminada: Esta es la contraseña local utilizada para ssh, que es **cisco123!**
- Protocolo: SSH junto con el grupo de intercambio de claves esperado por el router.
- OS: El sistema operativo en el router,**iosxe**

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie create testbed inter
active --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):
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$
```

Usamos cat para ver el archivo **testbed.yml** en el directorio **yaml**. Observe sus entradas en el archivo YAML. Su contraseña SSH está cifrada y la contraseña de habilitación "pedirá" al usuario que introduzca la contraseña si es necesario.

```
(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: '%ENC{w5PDosOUw5fDosKQwpbCmMKH}'
    os: iosxe
    type: iosxe
```

//aqui hubo un error que luego corregi, aqui puse **ip: 192.168.56.101,** pero deberia ser para mi caso **192.168.56.102** segun mi router virtual csr1000v

Paso 2: Utilizar Genie para analizar el resultado del comando show ip interface brief en JSON.
activamos el router virtual. IMAGEN DEL ROUTER VIRTUAL CON IP **192.168.56.102**

```
CSR1kv>show ip interface brief
Interface              IP-Address      OK? Method Status
GigabitEthernet1       192.168.56.102  YES DHCP   up
CSR1kv>
```

Usando el archivo YAML testbed, **invocamos a Genie** para analizar el resultado no estructurado del comando “show ip interface brief ” en JSON estructurado. Este comando incluye el comando IOS que se va a analizar (show ip interface brief), el archivo YAML testbed (testbed.yml) y el dispositivo especificado en el archivo testbed (CSR1kv).
Usamos el siguiente comando:
“genie parse “show ip interface brief” - -testbed-file yaml/testbed.yml - -devices CSR1kv”

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie parse "show ip inter
face brief" --testbed-file yaml/testbed.yml --devices CSR1kv
Using the default YAML encoding key since no key was specified in configuration.
THIS IS A SHARED KEY AND IS NOT SECURE, PLEASE RUN `pyats secret keygen` AND ADD
 TO YOUR pyats.conf FILE BEFORE ENCODING ANY VALUES.
2023-06-15T06:01:38: %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, ignorin
g
  0%|                                                          | 0/1 [00:00<?, ?it/s]
{
  "interface": {
    "GigabitEthernet1": {
      "interface_is_ok": "YES",
      "ip_address": "192.168.56.102",
      "method": "DHCP",
      "protocol": "up",
      "status": "up"
    }
  }
}
100%|                                                          | 1/1 [00:01<00:00,  1.38s/it]

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

Paso 3: Utilice Genie para analizar el resultado del comando show version en JSON.
Para otro ejemplo, para analizar el resultado no estructurado se utiliza el comando show version en JSON estructurado.

“genie parse “show version” - -testbed-file yaml/testbed.yml - -devices CSR1kv”

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie parse "show version"
 --testbed-file yaml/testbed.yml --devices CSR1kv
Using the default YAML encoding key since no key was specified in configuration.
THIS IS A SHARED KEY AND IS NOT SECURE, PLEASE RUN `pyats secret keygen` AND ADD
 TO YOUR pyats.conf FILE BEFORE ENCODING ANY VALUES.
2023-06-15T06:09:03: %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, ignorin
g
  0%|                                                    | 0/1 [00:00<?, ?it/s]
{
  "version": {
    "chassis": "CSR1000V",
    "chassis_sn": "9J8Y0QB13YY",
    "compiled_by": "mcpre",
    "compiled_date": "Sun 05-Sep-21 00:37",
    "copyright_years": "1986-2021",
    "curr_config_register": "0x2102",
    "disks": {
      "bootflash:.": {
        "disk_size": "6188032",
        "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",
```

.....

```
    "image_type": "production image",
    "label": "RELEASE SOFTWARE (fc3)",
    "last_reload_reason": "reload",
    "license_level": "ax",
    "license_type": "N/A(Smart License Enabled)",
    "location": "Gibraltar",
    "main_mem": "723108",
    "mem_size": {
      "non-volatile configuration": "32768",
      "physical": "2271876"
    },
    "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": "21 minutes",
    "uptime_this_cp": "25 minutes",
    "version": "16.12.6",
    "version_short": "16.12",
    "xe_version": "16.12.06"
  }
}
100%|                                                    | 1/1 [00:01<00:00,  1.34s/it]

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

Parte 5: Utilizar Genie para comparar configuraciones.

Como ha visto, Genie se puede usar para analizar comandos show en json estructurado. Genie también se puede utilizar para:

- Tomar capturas de pantalla de configuraciones anualmente y hacer comparaciones entre ellas.
- Automatizar las implementaciones de pruebas en un entorno virtual para realizar pruebas antes de la implementación en producción.
- Para solucionar problemas de configuraciones, haciendo comparaciones entre dispositivos.

En las partes 5 y 6, verá cómo hacer una comparación entre dos salidas diferentes.

Paso 1: Agregue una dirección IPv6 a CSR1kv.

```
CSR1kv>enable
CSR1kv#conf term
Enter configuration commands, one per line.  End with CNTL/Z.
CSR1kv(config)#interface gig 1
CSR1kv(config-if)#ipv6 address 2001:db8:acad:56::101/64
```

Paso 2: Utilice Genie para verificar la configuración y analizar la salida en JSON.

Analizar la salida no estructurada del comando show ipv6 interface en JSON estructurado. Utilice el parámetro —output para enviar la salida a un directorio verify-ipv6-1. Observe el resultado en el que Genie dice que se crearon dos archivos.

“genie parse “show ipv6 interface gig 1” - -testbed-file yaml/testbed.yml - -devices csr1kv - -output verify-ipv6-1

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie parse "show ipv6 int
erface gig 1" --testbed-file yaml/testbed.yml --devices CSR1kv --output verify-i
pv6-1
Using the default YAML encoding key since no key was specified in configuration.
THIS IS A SHARED KEY AND IS NOT SECURE, PLEASE RUN `pyats secret keygen` AND ADD
 TO YOUR pyats.conf FILE BEFORE ENCODING ANY VALUES.
2023-06-15T06:39:16: %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, ignorin
g
100%|                                                    | 1/1 [00:03<00:00,  3.67s/it]
+==============================================================================+
| 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$
```

Enumere los archivos creados por Genie en el directorio verify-ipv6-1. Observe que hubo dos archivos creados con nombres similares, pero uno que termina en _console.txt y el otro en _parsed.txt. El nombre de cada archivo incluye el nombre del dispositivo y el comando IOS utilizado en el comando Genie parse.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l verify-ipv6-1
total 16
-rw-rw-rw- 1 devasc devasc 4646 Jun 15 06:39 connection_CSR1kv.txt
-rw-rw-r-- 1 devasc devasc  760 Jun 15 06:39 CSR1kv_show-ipv6-interface-gig-1_co
nsole.txt
-rw-rw-r-- 1 devasc devasc  877 Jun 15 06:39 CSR1kv_show-ipv6-interface-gig-1_pa
rsed.txt
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Utilice cat para examinar el contenido del archivo _console.txt. Observe tanto la dirección de unidifusión global IPv6 que configuró como una dirección local de vínculo EUI-64 automática.
NOTAMOS QUE APARECE EL IPv6 QUE AGREGAMOS EN EL ROUTER VIRTUAL (en global unicast)

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat verify-ipv6-1/CSR1kv_s
how-ipv6-interface-gig-1_console.txt
+++ CSR1kv with via 'cli': 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:FE76:7A35
  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:FF76:7A35
  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$
```

Utilice cat para examinar el contenido del archivo _parsed.txt. Este es el archivo JSON analizado del comando show ipv6 interface gig 1.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat verify-ipv6-1/CSR1kv_s
how-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:FE76:7A35": {
        "ip": "FE80::A00:27FF:FE76:7A35",
        "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:FF76:7A35"
    ],
    "mtu": 1500,
    "oper_status": "up"
  },
  "_exclude": []
}(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Paso 3: Modifique la dirección local del vínculo IPv6.

En el router virtual CSR1kV VM agregue la siguiente dirección IPv6:

```
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
CSR1kv(config-if)#
```

Paso 4: Utilice Genie para verificar la configuración y analizar la salida en JSON.

Analice la salida no estructurada del comando show ipv6 interface en JSON estructurado. Utilice el parámetro —output para enviar la salida a un directorio diferente verify-ipv6-2. Puede usar el historial de comandos para recuperar el comando anterior (flecha arriba). Solo asegúrese de cambiar el 1 a un 2 para crear un nuevo directorio verify-ipv6-2 .

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie parse "show ipv6 int
erface gig 1" --testbed-file yaml/testbed.yml --devices CSR1kv --output verify-i
pv6-2
Using the default YAML encoding key since no key was specified in configuration.
THIS IS A SHARED KEY AND IS NOT SECURE, PLEASE RUN `pyats secret keygen` AND ADD
 TO YOUR pyats.conf FILE BEFORE ENCODING ANY VALUES.
2023-06-15T06:54:35: %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, ignorin
g
100%|                                                                           | 1/1 [00:02<00:00,  2.22s/it]
+==============================================================================+
| 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$
```

Enumere los archivos creados por Genie en el directorio verify-ipv6-2. Estos son similares a los dos archivos que creó antes de cambiar la dirección local del vínculo IPv6.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ ls -l verify-ipv6-2
total 16
-rw-rw-rw- 1 devasc devasc 4644 Jun 15 06:54 connection_CSR1kv.txt
-rw-rw-r-- 1 devasc devasc  743 Jun 15 06:54 CSR1kv_show-ipv6-interface-gig-1_co
nsole.txt
-rw-rw-r-- 1 devasc devasc  846 Jun 15 06:54 CSR1kv_show-ipv6-interface-gig-1_pa
rsed.txt
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Examinamos el contenido de cada archivo usando **cat.** Los cambios se resaltan en la salida siguiente.

console.txt

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat verify-ipv6-2/CSR1kv_s
how-ipv6-interface-gig-1_console.txt
+++ CSR1kv with via 'cli': 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$
```

parsed.txt

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat verify-ipv6-2/CSR1kv_s
how-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"
      },
        "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$
```

Paso 5: Utilice Genie para comparar la diferencia entre las configuraciones.

En el paso anterior, es bastante fácil encontrar el cambio a la dirección local del vínculo IPv6. Pero suponga que estaba buscando un problema en una configuración compleja. Quizás, está tratando de encontrar una diferencia entre una configuración OSPF en un enrutador que está recibiendo las rutas adecuadas y otro enrutador que no lo está, y desea ver las diferencias en sus configuraciones OSPF. O tal vez, está tratando de detectar la diferencia en una larga lista de sentencias ACL entre dos enrutadores que se supone que tienen políticas de seguridad idénticas. Genie puede hacer la comparación por usted y hacer que sea fácil encontrar las diferencias.

–Utilice el siguiente comando para que Genie encuentre las diferencias entre los dos archivos JSON analizados. Observe que la salida le indica dónde puede encontrar las comparaciones de Genie. En este caso, el primer nombre de archivo es la configuración anterior y el segundo nombre de archivo es la configuración actual.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ genie diff verify-ipv6-1 v
erify-ipv6-2
1it [00:00, 367.66it/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$
```

Utilice cat para ver el contenido del archivo con las diferencias. El signo más + indica adiciones y el signo menos - indica lo que se eliminó.

```
(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$ cat ./diff_CSR1kv_show-ipv
6-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:FE76:7A35:
-   ip: FE80::A00:27FF:FE76:7A35
-   origin: link_layer
-   status: valid
  joined_group_addresses:
-  index[2]: FF02::1:FF76:7A35
+  index[2]: FF02::1:FF56:1(csr1kv) devasc@labvm:~/labs/devnet-src/pyats/csr1kv$
```

Parte 6: Limpieza de laboratorio e investigación adicional.

Paso 1: Desactivamos el entorno virtual de Python.

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

Paso 2: Explore más casos de uso de PyATS y Genie.

Resumen , conclusiones, Observaciones:

RESUMEN:

Parte2:

Creamos un entorno virtual (venv) “csr1kv” y ahí usamos la herramienta pyATS, en este entorno virtual se crearan carpetas como bin, y un archivo pyvenv.cfg este ultimo archivo apunta a la ubicación de su instalacion de python.

Parte3:

Instalamos pyATS y verificamos la instalacion con “pyats --help”. Luego clonamos el repositorio de scripts de pyATS y enumeramos los archivos en el directorio actual.
Examinamos los scripts basicos “basic_example_script.py”, este script tiene bloques de configuracion comun, bloques de prueba multiples y un bloque de limpieza comun. Estos script pyATS se pueden ejecutar como archivos independientes o compilarlo en un trabajo para ejecutarlo con un todo.
Despues ejecutamos manualmente pyATs . Mostramos la ejecución del trabajo pyATS y se resalta que el script de prueba pasa todas las tareas requeridas, excepto una prueba intencionalmente falla.

Parte4:

Usaremos Genie para analizar la salida no estructurada de comandos IOS y convertirla en salida JSON estructurada. Crearemos un archivo yaml de prueba en la que brindaremos informacion como nombre de host, direccion IP (del router virtual), nombre de usuario, contraseñas.
Usamos el router virtual csr1kv y mostramos la direccion IPv4 y usando el archivo yaml para analizar la salida en formato JSON estructurado. esto del comando “show ip interface brief”, tambien hacemos para el comando “show version”.

Partr5:

Aqui exploramos como usar genie para comparar configuraciones, esto es util para realizar pruebas antes de la implementación , detectar diferencias entre dispositivos.
Agregaremos una direccion IPv6 a csrk1v, usando Genie verificamos la configuración y analizamos la salida en formato JSON estructurada. se crean 2 archivos txt.
Modificamos la direccion local de IPv6 en csr1kv. nuevamente analizamos usando Genie y observamos que nuevamente se crean 2 archivos txt.
Y comparamos las diferencias entre las 2 configuraciones **genie diff verify-ipv6-1 verify-ipv6-2**

CONCLUSIONES:

pyATS nos proporciona una plataforma sólida para la automatización de pruebas en entornos de red. permite la creación de scripts de prueba flexibles y escalables, y brinda herramientas para ejecutar, monitorear y generar informes sobre las pruebas realizadas.
El uso de Genie junto con pyATS ofrece una forma eficiente y poderosa de analizar y procesar la salida de comandos IOS, lo que facilita el desarrollo de scripts y aplicaciones para la gestión y automatización de redes.
Genie nos ofrece una solución potente y eficiente para comparar y analizar configuraciones y detectar diferencias en el entorno de red, lo que contribuye a una gestión más efectiva y automatizada de las configuraciones de los dispositivos.

OBSERVACIONES:
En mi caso en el router virtual, mi ip sale 192.168.56.102 y yo habia creado el testbed.yml con 192.168.56.101 y cuando invocabamos a Genie para analizar el resultado no estructurado del comando show ip interface brief en JSON estructurado. me salia error, lo cambie mi testbed.yml por el ip me que corresponde.

En el router virtual el prompt csr1kv> indica que estás en un modo de usuario ejecutivo (EXEC mode) en lugar de estar en la CLI de configuración. En este modo, no podemos ingresar directamente al modo de configuración usando el comando "conf term".
Para acceder al modo de configuración digitamos csr1kv> enable, ahi ingresamos al modo privilegiado el prompt cambiará a "csr1kv#" , ahora si escribimos csr1kv# conf term , csr1kv(config)#.