ISSN 1992-6510
e-ISSN 2520-9299

REALIDAD y Reflexión
Reality and Reflection

61

Realidad y Reflexión es una publicación periódica de carácter semestral de la Universidad Francisco Gavidia
Año 25, N° 61, Enero-Junio 2025. San Salvador, El Salvador, Centroamérica

Reality and Reflection is a biannual periodical publication of the Francisco Gavidia University
Year 25, N° 61, January-June 2025. San Salvador, El Salvador, Central America

Automatización de infraestructura de redes: eficiencia de tareas concurrentes utilizando lenguaje de programación Python

Network infrastructure automation: concurrent task efficiency using Python programming language

David Ernesto Mena Cañas
Ingeniería en Sistemas Informáticos, Universidad Evangélica de El Salvador, El Salvador
Profesional certificado en administración de infraestructura de redes, CISCO CCNA, CCNP SD-WAN
IT Architecture Network Manager, PRICESMART
davermeca@gmail.com
https://orcid.org/0009-0009-2221-0176

Fecha de recepción: 13 de enero de 2025
Fecha de aprobación: 25 de febrero de 2025
DOI:

RESUMEN

El crecimiento de las redes en los últimos años ha incrementado la cantidad de dispositivos que deben configurarse y administrarse, lo que complejiza la labor de los administradores y vuelve inviable la gestión individual. Ante este panorama, la automatización de redes —mediante software que configura, aprovisiona, administra y prueba dispositivos de forma automática [1]— permite optimizar los procesos y minimizar errores humanos. En este contexto, el lenguaje Python se emplea en pruebas de automatización de redes [1]–[3], [6], gracias a su facilidad de adaptación, manejo de datos y variedad de bibliotecas. Su simplicidad y amplio ecosistema lo convierten en una opción viable para estas tareas. Destacan especialmente las librerías que permiten ejecutar procesos en paralelo o simultáneamente, lo que reduce significativamente los tiempos frente a tareas secuenciales [3]. Además, Python cuenta con bibliotecas multivendor que posibilitan la conexión con equipos de distintos fabricantes, ampliando así las posibilidades de integración. En este trabajo, se valida la eficiencia de dos librerías orientadas a la ejecución paralela: Threading y concurrent. futures, que contribuyen a acelerar la realización de tareas similares en varios dispositivos al mismo tiempo. Estas librerías se emplearán para recolectar datos, realizar cambios y respaldar configuraciones de red. Por último, se evalúa un mecanismo de cifrado de contraseñas como parte de las estrategias de automatización, fortaleciendo la seguridad en los procesos de configuración remota.

Palabras clave: automatización de redes, automatización de tareas, programación de redes, programación de tareas, seguridad.

ABSTRACT

The growth of networks in recent years has increased the number of devices that need to be configured and managed, which complicates the work of administrators and makes individual management unfeasible. Against this backdrop, network automation - through software that automatically configures, provisions, manages and tests devices [1] - makes it possible to optimize processes and minimize human errors. In this context, the Python language is used in network automation testing [1]-[3], [6], thanks to its ease of adaptation, data handling and variety of libraries. Its simplicity and broad ecosystem make it a viable option for these tasks. Particularly noteworthy are the libraries that allow processes to be executed in parallel or simultaneously, which significantly reduces time compared to sequential tasks [3]. In addition, Python has multivendor libraries that make it possible to connect with equipment from different manufacturers, thus expanding integration possibilities. In this work, we validate the efficiency of two parallel execution oriented libraries: Threading and concurrent.futures, which contribute to speed up the performance of similar tasks in several devices at the same time. These libraries will be used to collect data, make changes and support network configurations. Finally, a password encryption mechanism is evaluated as part of the automation strategies, strengthening security in remote configuration processes.

Keywords: *network automation, task automation, network programming, task programming, security.*

Introducción

La automatización de las infraestructuras de red utilizando el lenguaje de programación Python se ha convertido en una habilidad relevante en el desarrollo de aplicaciones modernas, especialmente aquellas que implican la administración de grandes infraestructuras, gracias a la versatilidad de este lenguaje. Python es ampliamente preferido en el mercado por su simplicidad y facilidad de uso. Existen diversos estudios y artículos, los cuales se detallan en las referencias bibliográficas de este artículo; esto permite a desarrolladores de todos los niveles crear soluciones de red eficaces y escalables. En la actualidad, muchos proveedores y profesionales de administración de infraestructuras de red lo han adoptado por las razones mencionadas.

Otro de los motivos de su popularidad es la extensa biblioteca estándar, que ofrece módulos adaptables a distintas áreas de la ciencia. Además, bibliotecas externas como Threading y Netmiko proporcionan herramientas avanzadas para gestionar solicitudes, conexiones SSH y tareas de administración repetitivas, tales como respaldos, configuraciones, obtención de estado o monitoreo de equipos.

Python también es compatible con paradigmas modernos, como la programación concurrente por hilos o paralela, que permite manejar múltiples conexiones de manera eficiente. Este es el objeto de estudio de este artículo. Se abordará cómo, al utilizar estas librerías, el proceso de automatización reduce los tiempos al ejecutar las tareas de manera paralela en lugar de secuencial. Esto resulta determinante en aplicaciones como la verificación del estado de la red, respaldos de configuraciones de dispositivos de red, servidores web, sistemas de mensajería y servicios de configuración a gran escala. Su amplia comunidad de desarrolladores genera abundante documentación y soporte, lo que facilita la resolución de problemas.

Múltiples empresas como Cisco, Google, Dropbox y Netflix, entre otras, utilizan Python en sus sistemas de red, lo que refuerza su presencia en el mercado. Gracias a todo lo anterior, Python destaca en la automatización de redes por su combinación de simplicidad, versatilidad y soporte robusto, lo que lo convierte en una opción destacada para el ámbito del *networking*.

Propósito

Las redes se han vuelto cada vez más complejas, en gran parte por el escalamiento y las ventajas que ofrecen para los negocios. Sin embargo, esto implica una carga administrativa considerable que se torna inviable e insostenible si no se cuenta con herramientas que permitan adaptarse con agilidad a los cambios en los requerimientos empresariales. Estas demandas, además, deben ser atendidas de manera inmediata y sin errores en su implementación. Por ello, el mercado exige herramientas de automatización que contribuyan a la reducción de costos, a una mayor velocidad de respuesta y a una gestión ágil de la infraestructura de red [6].

En ese sentido, se confirmará si Python puede ejecutar tareas de forma concurrente con mayor eficiencia y en menor tiempo, en comparación con la ejecución secuencial, que limita el aprovechamiento del procesador. Este es un aspecto de base por el cual la industria se orienta hacia la automatización, ya que facilita la escalabilidad y el crecimiento [7].

El otro componente a validar es un mecanismo de cifrado y descifrado de contraseñas, especialmente orientado a la protección de información crítica, la cual forma parte de los aspectos mínimos de seguridad en el entorno tecnológico. Como se señala en la fuente citada: «La principal tarea de la seguridad informática es la de minimizar los riesgos; en este caso provienen de muchas partes, puede ser de la entrada de datos...» [5].

Alcance de la investigación

Se delimitará la generación de *scripts* de automatización de red para la configuración y obtención de información en equipos Cisco, tales como *routers* y *switches*, que operan bajo el sistema IOS/IOS-XE. Se utilizará el lenguaje de programación Python, empleando la librería Netmiko. Asimismo, se validará la eficiencia de la librería Threading, la cual proporciona una manera de manejar múltiples tareas de forma concurrente, y de concurrent.futures, que provee una interfaz de alto nivel para ejecutar invocables de manera asincrónica. Ambas permiten realizar configuraciones en paralelo de forma más eficiente.

Finalmente, se evaluará uno de los mecanismos de cifrado de contraseñas. Cryptography es un paquete desarrollado por Python Cryptographic Authority, que facilita la implementación de mecanismos criptográficos dentro del *software*. Entre sus componentes destaca la librería Fernet, que, según su documentación oficial [4], es una implementación de cifrado asimétrico, también denominado «clave secreta».

Como es sabido, el almacenamiento de credenciales en texto plano constituye uno de los puntos más vulnerables en la gestión de equipos, al representar una violación directa a las medidas de seguridad. En palabras de la fuente consultada: «La principal tarea de la seguridad informática es la de minimizar los riesgos; en este caso provienen de muchas partes, puede ser de la entrada de datos...» [5].

El enfoque se centra en equipos del proveedor Cisco, aplicados sobre una topología de pruebas, con el objetivo de validar la versatilidad en la implementación de los *scripts* desarrollados.

Métodos

El propósito de la investigación giró en dos grandes temas: primero, validar la eficiencia de las librerías Threading y concurrent.futures en cuanto a tareas concurrentes automatizadas; y segundo, analizar la

funcionalidad de cifrado para las contraseñas de los dispositivos de red —de manera que no queden en formato plano— provista por la librería Fernet, como parte de las buenas prácticas de seguridad. El diagrama que se utilizó de referencia es el de la Figura 1.

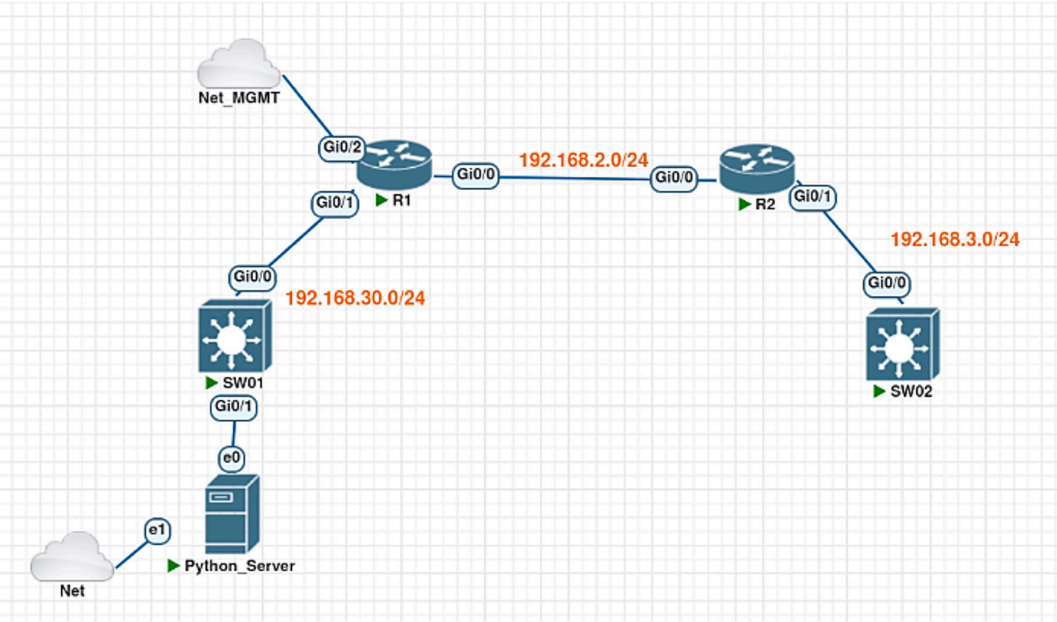

Figura 1. Topología de redes para las pruebas

Preliminarmente, se realizó la búsqueda de información relacionada con la programación de redes y las librerías de Python Threading y concurrent.futures para ejecutar tareas concurrentemente, así como acerca de la librería de seguridad. La búsqueda se efectuó en la base de datos EBSCOhost y en Google Scholar, utilizando los términos «network automation python», «Python threading» y «Security Python», sin aplicar filtros, con el objetivo de visualizar todos los documentos disponibles. Posteriormente, se llevó a cabo un proceso de selección de aquellos que cumplieran con el contenido requerido para la investigación, a fin de obtener su correspondiente cita según el estilo IEEE. Asimismo, se consultaron *whitepapers* en las páginas web de los vendors y otros contenidos en la Web, siempre que resultaran pertinentes como apoyo para la investigación.

Softwares

Se utilizó la infraestructura de virtualización de funciones de red (NFVI, por sus siglas en inglés) denominada PNETLab, versión 6, con el hipervisor KVM 5.0 y el sistema operativo Linux, distribución 20.04.6 LTS. Para los equipos *guest* se empleó CentOS Linux 8 (Core) y Python 3.6, con las librerías

Netmiko, Threading y concurrent.futures, con el propósito de evaluar el desempeño del procesamiento simultáneo. Para la parte de cifrado de contraseñas, se usó la librería Fernet. También se utilizaron imágenes IOS de Cisco: Switches versión 15.2 y Routers versión 15.7(3)M3.

Hardware

El equipo de hardware empleado fue una computadora portátil Dell Latitude 5420, con procesador 11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz, con 4 núcleos físicos y 8 hilos, capacidad de 64 bits, arquitectura multi-core, soporte para *hardware* Threading, protección de ejecución, virtualización mejorada y 64 GB de memoria RAM DDR4 a 3200 MHz. Se incorporó además una unidad de estado sólido Fanxiang S201 2T NVMe2.

Procedimiento de pruebas y mediciones

Para obtener los resultados de la investigación, se utilizó la topología mostrada en la Figura 1, con las especificaciones de *hardware* y *software* indicadas anteriormente. Se emplearon dos imágenes de *switches* (SW01, SWI02) y dos de enrutadores (R1, R2) para las pruebas, los cuales se configuraron con su respectivo direccionamiento IP y enrutamiento, de manera que la topología presentara conectividad física y lógica a través de la red, con diferentes segmentos. Así, el servidor (Python Server) fue capaz de conectarse a cada uno de ellos y autenticarse para obtener la información correspondiente de cada *switch* y *router* de la red.

Una vez establecida la conectividad desde el servidor y el acceso a cada equipo, el servidor ejecutó una serie de *scripts* o códigos de automatización cuya función fue medir el tiempo requerido para la obtención, configuración y respaldo de datos de cada elemento de red. Inicialmente, las tareas se realizaron de forma secuencial y, posteriormente, en paralelo. Para ello, el servidor contó con siete *scripts* que se ejecutaron desde el servidor denominado Python Server, de los cuales seis validaron tres tipos de operaciones: respaldo de configuraciones, lectura de información y configuración de nodos. Estas operaciones se realizaron tanto de forma secuencial como en paralelo, utilizando las librerías Threading y concurrent.futures, con el objetivo de determinar la eficiencia temporal de ambos enfoques de ejecución. Se buscó confirmar que la automatización de tareas en paralelo con dichas librerías permite una mayor eficiencia. Los *scripts* mostraron el tiempo de ejecución al finalizar cada proceso, lo que permitió obtener los datos necesarios para realizar una comparación temporal, usualmente expresada en segundos.

Finalmente, se evaluó el proceso de cifrado y descifrado mediante la librería Fernet, la cual se encarga de convertir información en texto plano en una cadena ilegible como mecanismo de seguridad. Esta cadena solo puede ser descifrada por el mismo sistema que generó el cifrado, mediante la llave correspondiente utilizada en el proceso. A continuación, se presentan los diagramas de operación de los *scripts* en la Figura 2.

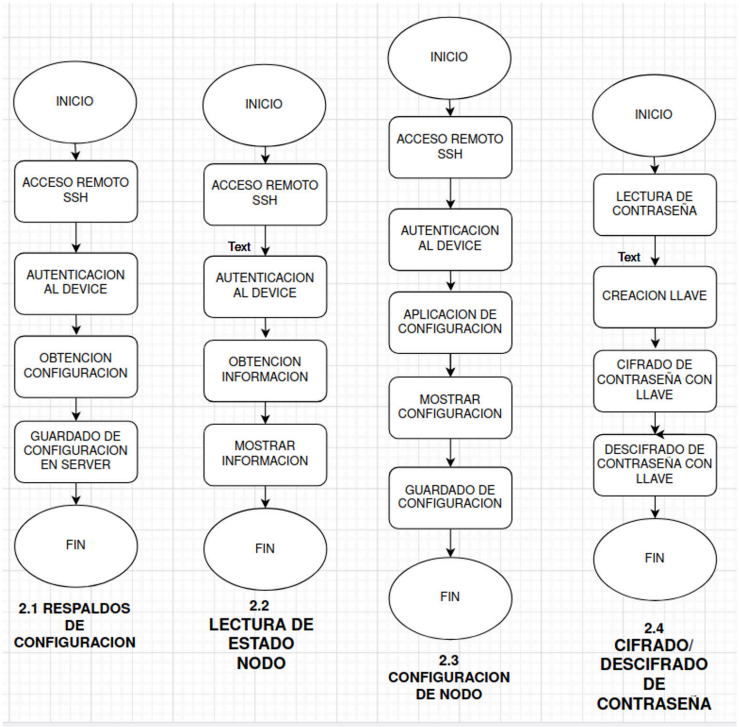

Figura 2. Diagrama de flujo de los *scripts*

Resultados

Como se mencionó en el apartado anterior, para obtener una muestra de tiempo de referencia se ejecutaron los *scripts* (2.1, 2.2, 2.3) diez veces, y en cada ejecución se registró un tiempo. Posteriormente, se calculó el promedio correspondiente. En primera instancia, las librerías no fueron activadas, con el fin de obtener un tiempo de referencia mediante la ejecución secuencial del proceso, cuyo resultado se presenta en la Tabla 1.

Tabla 1. Resumen de tiempos por *script* sin habilitar Threading

Nombre *script*	**Función**	**No Threading / Secuencial, segundos**
test_multi_backup2.py	Generación de respaldos	11.69
test_multi_backup.py		11.22
test_multi_get_conf2.py	Obtener información	12.45
test_multi_get_conf.py		12.55
test_multi_put_conf2.py	Configuración	14.78
test_multi_put_conf.py		14.88

La tabla anterior hace referencia a la ejecución de los *scripts* sin habilitar las librerías de ejecución de tareas en paralelo concurrent.futures (con sufijo terminado en 2) y Threading, lo cual sirve como referencia del tiempo de ejecución secuencial. La ejecución de los *scripts* se realizó de forma automatizada utilizando el siguiente *script* o código mostrado en la Figura 3.

```
[root@localhost test_python]# cat run.sh
#!/bin/bash
for file in $(ls -l test*.py | cut -d" " -f9);
        do
        echo "procesando file:" $file
                        for i in {1..10};
                                do
                                echo "procesamiento" $i
                                echo "procesando archivo de log:" $file.out.log
                                /bin/python3.6 $file >> $file.out.log ;
#                               grep -i script $file.out.log >> $file.summ_count.log
                        done
done
```

Figura 3. Automatización de ejecución *scripts* Python

Con lo anterior, se generó el archivo necesario para obtener los tiempos de ejecución por *script* y registrar el tiempo tomado en cada ejecución, como se puede apreciar en la Figura 4.

```
[root@localhost logs_pre]# grep scr *.log
test_multi_backup2.py.out.log:The script finished executing in 11.71 seconds.
test_multi_backup2.py.out.log:The script finished executing in 11.72 seconds.
test_multi_backup2.py.out.log:The script finished executing in 11.92 seconds.
test_multi_backup2.py.out.log:The script finished executing in 11.72 seconds.
test_multi_backup2.py.out.log:The script finished executing in 11.73 seconds.
test_multi_backup2.py.out.log:The script finished executing in 11.55 seconds.
test_multi_backup2.py.out.log:The script finished executing in 11.55 seconds.
test_multi_backup2.py.out.log:The script finished executing in 11.53 seconds.
test_multi_backup2.py.out.log:The script finished executing in 11.74 seconds.
test_multi_backup2.py.out.log:The script finished executing in 11.72 seconds.
test_multi_backup.py.out.log:The script finished executing in 11.2 seconds.
test_multi_backup.py.out.log:The script finished executing in 11.39 seconds.
test_multi_backup.py.out.log:The script finished executing in 11.21 seconds.
test_multi_backup.py.out.log:The script finished executing in 11.21 seconds.
test_multi_backup.py.out.log:The script finished executing in 11.2 seconds.
test_multi_backup.py.out.log:The script finished executing in 11.19 seconds.
test_multi_backup.py.out.log:The script finished executing in 11.41 seconds.
test_multi_backup.py.out.log:The script finished executing in 11.0 seconds.
test_multi_backup.py.out.log:The script finished executing in 11.22 seconds.
test_multi_backup.py.out.log:The script finished executing in 11.19 seconds.
```

Figura 4. Obteniendo tiempos de ejecución

Una vez ejecutado lo anterior, se procedió a activar en los *scripts* las librerías concurrent.futures y Threading, las cuales permiten ejecutar las tareas en forma paralela. Los resultados obtenidos se presentan en la Tabla 2.

Tabla 2. Resumen de tiempos por *script* con Threading o ejecución de tareas en paralelo activo

Nombre *script*	**Función**	**Threading / Secuencial, segundos**
test_multi_backup2.py	Generación de respaldos	3.09
test_multi_backup.py		3.00
test_multi_get_conf2.py	Obtener información	3.30
test_multi_get_conf.py		3.33
test_multi_put_conf2.py	Configuración	3.94
test_multi_put_conf.py		3.97

Con lo anterior, ahora es posible realizar una comparativa sobre el uso de las librerías, cuyos resultados se presentan en la Tabla 3, evidenciando una diferencia favorable en el rendimiento al emplear librerías para la ejecución de tareas en paralelo.

Tabla 3. Resumen comparativo de tiempos por *script* con Threading o procesamiento en paralelo desactivado/activado

Nombre *script*	**Función**	**No Threading / Secuencial, segundos**	**Threading / Simultáneo, segundos**	**% Eficiencia**
test_multi_backup2.py	Generación de respaldos	11.69	3.09	73.56 %
test_multi_backup.py		11.22	3.00	73.28 %
test_multi_get_conf2.py	Obtener información	12.45	3.30	73.47 %
test_multi_get_conf.py		12.55	3.33	73.46 %
test_multi_put_conf2.py	Configuración	14.78	3.94	73.33 %
test_multi_put_conf.py		14.88	3.97	73.35 %

De acuerdo con los resultados, se observa un mejor desempeño al activar las librerías que permiten la ejecución de tareas simultáneas, como Threading, en los *scripts*, obteniéndose así una mayor eficiencia en comparación con su no activación. Es importante destacar que el uso de recursos en la topología no se vio afectado.

Finalmente, como parte del procedimiento, se evaluó la eficiencia entre ambas librerías, cuyos resultados se presentan en la Tabla 4.

Tabla 4. Resumen comparativo desempeños librerías concurrent.futures y Threading

Nombre *script*	**Función**	**No Threading / Secuencial, segundos**	**Threading / Simultáneo, segundos**	**% Eficiencia**	**% Eficiencia entre *scripts***	***Script* con mejor eficiencia simultánea**
test_multi_backup2.py	Generación de respaldos	11.69	3.09	73.56 %	3.01 %	test_multi_backup.py / concurrent.futures
test_multi_backup.py		11.22	3.00	73.28 %		
test_multi_get_conf2.py	Obtener información	12.45	3.30	73.47 %	0.81 %	test_multi_get_conf2.py / threading
test_multi_get_conf.py		12.55	3.33	73.46 %		
test_multi_put_conf2.py	Configuración	14.78	3.94	73.33 %	0.60 %	test_multi_put_conf2.py / threading
test_multi_put_conf.py		14.88	3.97	73.35 %		

Como se observa, al utilizar y comparar ambas librerías, la diferencia en el desempeño es mínima, por lo que ambas pueden habilitarse. No obstante, se debe tomar en consideración lo siguiente: concurrent.futures se recomienda para casos en los que se requiera ejecutar tareas en paralelo utilizando diferentes CPU, mientras que Threading resulta útil cuando se desea ejecutar concurrencia en un mismo CPU, aprovechando la distribución de tiempo entre las tareas a ejecutar.

A continuación, se presenta un extracto del código que muestra cómo se habilitan las librerías concurrent.futures y Threading para su implementación en los *scripts* de automatización en Python:

- *Script* test_multi_backup.py: realizar respaldos de los nodos utilizando la librería concurrent.futures.

```
from netmiko import ConnectHandler
import time
import concurrent.futures
import datetime
import re
```

Figura 5. Habilitando la librería concurrent.futures

```
with concurrent.futures.ThreadPoolExecutor() as exe:
    ip_addresses = fetch_ip_addresses()
    results = exe.map(backup_rtr_configuration, ip_addresses)
```

Figura 6. Utilizando la librería concurrent.futures

Para los *scripts* subsecuentes, se mantiene la misma definición y habilitación de la librería concurrent.futures. La principal variación en los archivos test_multi_get_conf.py y test_multi_put_conf.py radica en la definición de las tareas destinadas a obtener la información y a configurar el nodo, respectivamente:

- *Script* test_multi_backup2.py: realizar respaldos de los nodos utilizando la librería Threading.

```
import threading
from netmiko import ConnectHandler, NetmikoAuthenticationException
import time
import concurrent.futures
import datetime
import re
```

Figura 7. Habilitando la librería Threading

```
threads = []
for device in devices_list:
    thread = threading.Thread(target=configure_device, args=(device,))
    threads.append(thread)
    thread.start()

for thread in threads:
    thread.join()
```

Figura 8. Utilizando la librería Threading

Para los *scripts* subsecuentes, se mantiene la misma definición y habilitación de la librería Threading. La principal variación en los archivos test_multi_get_conf2.py y test_multi_put_conf2.py consiste en la definición de las tareas destinadas a obtener la información y configurar el nodo, respectivamente.

Como parte de los mecanismos de seguridad, se evaluaron las opciones de cifrado que ofrece Python. En este punto, se analizó la librería Fernet, que forma parte de la librería Cryptography [8]. Esta emplea un mecanismo simétrico basado en los algoritmos de cifrado AES-CBC más HMAC. Las validaciones se realizaron de la siguiente manera: el *script* genera una llave que servirá para cifrar y descifrar la contraseña, tal como se muestra en la Figura 9.

Figura 9. Cifrando/Descifrando contraseñas con librería Fernet

Tal como se aprecia en la figura, el *script* lee la contraseña en texto plano en el tercer campo, separado por coma; este corresponde al paso 1. Como paso 2, se genera una llave única con la cual se cifra la contraseña. Finalmente, utilizando la misma llave, se procede a descifrarla. Con este procedimiento, es posible aplicar mecanismos de aseguramiento de contraseñas, permitiendo almacenar dicho valor, que posteriormente solo podrá ser accedido o descifrado mediante la misma llave con la cual fue generado el *password* o token.

Discusión

Como se ha podido constatar y confirmar mediante pruebas, el lenguaje de programación Python ofrece herramientas para la automatización de redes. Tal como se demostró, es posible ejecutar tareas o procesos de forma secuencial por defecto; sin embargo, se puede lograr mayor eficiencia en los scripts si se aprovecha el procesamiento en paralelo, lo cual fue comprobado en este estudio mediante el uso de las librerías Threading y concurrent.futures. En las pruebas, se ejecutaron tareas de automatización en las que se evidenció una mejora en el rendimiento de hasta un 73.6 % en la ejecución de tareas automatizadas en paralelo, utilizando dichas librerías del lenguaje Python. Esto confirma que su uso es viable y contribuye a la automatización de diversos procesos o tareas concurrentes.

Es importante destacar que esta comparativa puede variar según el *hardware*, los recursos del sistema, la cantidad de nodos o equipos involucrados, entre otras variables. Lo que resalta en el estudio es que los *scripts* pueden optimizarse mediante el uso de estas librerías, lo que permite reducir el tiempo de ejecución en la automatización de procesos con Python. No se pretende establecer comparaciones con otras tecnologías, ya que el enfoque se centra en el lenguaje Python como objeto de estudio.

Con lo anterior, se abre un abanico de oportunidades para identificar qué procesos o tareas pueden automatizarse con Python, el cual es actualmente uno de los lenguajes más utilizados en la industria de la automatización de redes [1], [2], [3], [6].

También es importante destacar que la ejecución de tareas en paralelo se convierte en una herramienta estratégica para aquellas actividades que requieren automatización dentro de la infraestructura de red, tales como respaldos, configuraciones base y obtención de datos críticos a gran escala. Esta automatización puede complementar otros mecanismos, como los ofrecidos por SNMP (*Simple Network Management Protocol*), el cual, según se publicó en 1988, fue concebido para supervisar y controlar redes grandes y complejas de una manera más sencilla [9], o bien reemplazar tareas manuales de conexión equipo por equipo, lo cual resulta especialmente tedioso en infraestructuras de gran tamaño.

Asimismo, se confirmó que Python dispone de herramientas de cifrado que permiten proteger las credenciales de acceso. Esto significa que, si se cuenta con una base de datos centralizada de contraseñas en un sistema automatizado desarrollado en este lenguaje, es posible asegurar las credenciales de acceso a los nodos gestionados. Más aún, se puede evitar el riesgo de seguridad que implica mantener las contraseñas en texto plano, una práctica lamentablemente común en procesos de automatización con Python. En este sentido, la librería Fernet, que implementa métodos de cifrado simétrico, puede contribuir a no comprometer los sistemas de automatización desarrollados en Python, considerando el uso cada vez más extendido de este lenguaje y la necesidad de incorporar buenas prácticas de seguridad [4], [5], [8], [9].

Referencias

[1] S. Pérez, H. Facchini, A. Dantiacq, B. Roberti y F. Hidalgo, «Plataformas de Automatización de Red», en XXIV Edición del Workshop de Investigadores en Ciencias de la Computación 2, 2021, pp. 1–5. [En línea]. Disponible en: https://sedici.unlp.edu.ar/bitstream/handle/10915/143282/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y

[2] G. Milios, Network automation using Python, 2021. International Helenic University. [En línea]. Disponible en: https://repository.ihu.edu.gr/xmlui/bitstream/handle/11544/29802/Network%20Automation%20using%20Python%28final%29.pdf?sequence=1

[3] D. Wicaksono y B. Soewito, «Application of the Multi-Threading Method and Python Script for the Network Automation», Journal of Syntax Literate, vol. 9, no. 6, 2024. [En línea]. Disponible en: https://jurnal.syntaxliterate.co.id/index.php/syntaxliterate/article/view/16345/10172

[4] T. Devendra, «Simple way to encode a string according to a password», Stack Overflow, 2013–2024. [En línea]. Disponible en: https://stackoverflow.com/questions/2490334/simple-way-to-encode-a-string-according-to-a-password

[5] J. Sánchez, Introducción a la seguridad informática. Barcelona, España: Editorial UOC, 2018. [En línea]. Disponible en: https://www.google.com.sv/books/edition/INTRODUCCI%C3%93N_A_LA_SEGURIDAD_INFORM%C3%81TIC/5Z9yDwAAQBAJ?hl=en&gbpv=1&dq=almacenan+las+credenciales+de+los+equipos+cisco+ataques&printsec=frontcover

[6] M. H. Mortensen, Economic Benefits of Network Automation. ACG Research/Cisco, 2021. [En línea]. Disponible en: https://www.cisco.com/c/dam/en/us/products/collateral/cloud-systems-management/network-services-orchestrator/acg-economic-benefits-of-network-automation.pdf

[7] J. Ratanpara y N. Pithadia, «Efficient Network Management through Python Scripting», 2024. [En línea]. Disponible en: https://www.researchgate.net/profile/Nehaben-Pithadia/publication/379695034_Efficient_Network_Management_through_Python_Scripting/links/66160d-9b39e7641c0ba87f6e/Efficient-Network-Management-through-Python-Scripting.pdf

[8] cryptography.io, «Fernet.generate_key», 2013–2024. [En línea]. Disponible en: https://cryptography.io/en/latest/fernet/#cryptography.fernet.Fernet.generate_key

[9] W. Stallings, Fundamentos de seguridad en redes, 1.ª ed. México: Pearson, 2004, p. 262. [En línea]. Disponible en: https://www.google.com.sv/books/edition/Fundamentos_de_seguridad_en_redes/cjsHVSwbHwoC?hl=en&gbpv=1&dq=snmp+en+redes&pg=PA262&printsec=frontcover