TecnoCampus
Escola Superior Politècnica

Centre adscrit a la
upf. Universitat Pompeu Fabra Barcelona

Grado en Ingeniería Informática de Gestión y Sistemas de Información

Plataforma de monitorización de red

Memoria

Jaime Antonio Rodriguez Gonzales
TUTOR: Pere Barberan Agut

Curso 2022-2023

TecnoCampus
Mataró-Maresme

Dedicatoria

A mi madre, Cecilia, a mi novia Karen y a mi familia en general, por su amor, apoyo y comprensión incondicionales en mi trayectoria académica. También dedico este trabajo a la memoria de mi profesor Leonard Janer, quien fue una gran inspiración. Esto es un reconocimiento a todas las personas que han sido parte fundamental en mi desarrollo y crecimiento.

Agradecimientos

Agradezco a mi tutor Pere Barberan, mis amigos y Damon Albarn.

Abstract

This project introduces a basic network monitoring solution through the implementation of automation and an interactive web platform. The aim is to expedite network equipment configuration, improve efficiency, and reduce human errors. Additionally, it establishes a strong connection between network topology information and the web environment. Through an automated process and a dynamic web interface, the monitoring of network equipment configuration is facilitated.

Resum

Aquest projecte presenta una solució bàsica de monitorització de xarxa mitjançant la implementació d'automatització i una plataforma web interactiva. Es busca agilitzar la configuració d'equips de xarxa, millorar l'eficiència i reduir errors humans. A més, s'estableix una connexió sòlida entre la informació de la topologia de xarxa i l'entorn web. Mitjançant un procés automatitzat i una interfície web dinàmica, es facilita la monitorització de la configuració dels equips de xarxa.

Resumen

Este proyecto presenta una solución básica de monitorización de red mediante la implementación de automatización y una plataforma web interactiva. Se busca agilizar la configuración de equipos de red, mejorar la eficiencia y reducir errores humanos. Además, se establece una conexión sólida entre la información de la topología de red y el entorno web. Mediante un proceso automatizado y una interfaz web dinámica, se facilita la monitorización de la configuración de los equipos de red.

Índice

Índice de figuras

Glosario de términos

Framework	Esquema que proporciona una estructura básica para el desarrollo de software.
TIC	Tecnologías de la información y comunicación.
DevOps	Modelo de trabajo que promueve el Desarrollo de aplicaciones en menos tiempo y constantes implementaciones y/o revisiones.
SSH	Protocolo de administración remota a través de autenticación.
API	Fragmento de código que permite la comunicación entre dos componentes.
Back end	Parte de una aplicación que se dedica a tratar los datos y la programación de sus principales funcionalidades.
NAT	Traductor de direcciones de red.
SCP	Protocolo de transferencia de archivos de forma segura.

1. Introducción

Este proyecto tiene como meta implementar la automatización de equipos de red en un entorno de simulación como es GNS3 ya que esta plataforma nos permite tanto el despliegue de las topologías de red como la automatización del testeo de la configuración y haciendo uso del lenguaje de programación Python, el cual provee de diversas tecnologías y/o herramientas para la automatización de redes como pueden ser las librerías Netmiko y NAPALM.

Además, se pretende crear una plataforma web funcional con ayuda del Framework denominado Flask y que esta sea capaz de consultar la información de los equipos de red configurados en la topología que se utiliza como ejemplo.

De este modo, se quiere combinar tanto los conocimientos de programación como los de administración de redes aprendidos a lo largo del grado para dar el primer paso en el mundo de la automatización y crear una solución que hoy en día es muy utilizada y demandada en el mundo de las TIC.

2. Marco teórico y análisis de referentes

2.1. Contexto

A medida que pasan los años, el sector TI avanza a pasos agigantados y el rol del DevOps es cada vez más importante, y en el ámbito de las soluciones de red, el despliegue de topologías y su configuración de forma escalable y segura es un elemento imprescindible.

En la actualidad, los departamentos de TI necesitan rapidez y flexibilidad a la hora de gestionar, configurar, mantener y administrar servicios basados en tecnologías tanto convencionales como en la nube. En ese sentido, resulta conveniente contar con una plataforma de automatización de red dotada de las herramientas necesarias para automatizar las operaciones de red.

La automatización en este contexto va de la mano con técnicas de lógica de programación y el resultado de esto permite la configuración de pocos y muchos dispositivos de red en muy poco tiempo, agilizando el trabajo del administrador de red, reduciendo recursos y disminuyendo el riesgo de errores por parte del error humano.

Por otro lado, la automatización también resulta muy útil al momento de monitorizar el estado ya sea de las redes, dispositivos o puertos, obteniendo información relevante para el usuario y con esto aprovecharla para optimizar el rendimiento de la red a futuro o simplemente detectar algún fallo que haya ocurrido en cierto momento.

2.2. Antecedentes

Los sistemas de gestión de redes han permanecido prácticamente inalterados durante los últimos años, pese a la constante evolución de las tecnologías que los sustentan. Previamente a la automatización de la red, la configuración y el mantenimiento de esta se realizaban manualmente. No obstante, a medida que las redes se hacían más complejas, como también el número de dispositivos en la red aumentaba, quedó claro que la forma de gestionar los procesos de configuración y el mantenimiento de la red de forma tradicional resultaba excesivamente lenta, tediosa y susceptible de cometer errores como para cubrir eficazmente las exigencias que plantean los cambios vertiginosos en el volumen de trabajo.

Entre las tecnologías utilizadas antes de la automatización de la red destacan:

- **Scripts y programas de automatización de tareas:** Anteriormente, los administradores de red empleaban *scripts* y programas hechos a medida para automatizar las tareas habituales en este ámbito, como la configuración de dispositivos de red, la creación de VLANs, la definición de ACLs, etc. Estos programas solían estar escritos en lenguajes de programación como Python, Bash, Perl, y otros, ejecutándose en un dispositivo de red o en un servidor de automatización. Esto exigía unos conocimientos avanzados de programación y era susceptible de cometer errores si no estaba correctamente diseñado y mantenido.
- **SNMP (Simple Network Management Protocol):** SNMP es un protocolo que se utiliza para monitorizar y administrar dispositivos de red. Hasta la automatización de la red, los administradores recurrían a esta tecnología para supervisar y recibir alertas de eventos en los equipos de red, tales como fallos de *hardware*, congestión de la red, etc.
- **RMON (Remote Monitoring):** RMON es una extensión de la tecnología SNMP destinada a que los administradores de red controlen y analicen detalladamente el tráfico de red. Antes, solían utilizar RMON para identificar y resolver problemas de rendimiento de la red.
- **TFTP (Trivial File Transfer Protocol):** TFTP es un protocolo que se utiliza para la transferencia de archivos entre diferentes dispositivos de red. Por ejemplo, antes de que existiera la automatización de redes, sus administradores utilizaban el protocolo TFTP para enviar archivos de configuración y actualizar el *software* de los equipos de red.

En definitiva, antes de la automatización, existían herramientas y protocolos como los *scripts* personalizados, SNMP, RMON y TFTP que permitían automatizar tareas y agilizar el proceso de administración de redes. Dichas utilidades resultaban mucho menos sofisticadas que las actuales aplicaciones para automatizar el trabajo de un gestor de red, pero supusieron una evolución importante respecto a la automatización de las labores que se llevan a cabo en la red.

Al automatizar la administración sobre los recursos y prestaciones que ofrecen las redes, se aumenta la velocidad y flexibilidad de los terminales, que así pueden responder con eficacia a las exigencias de las organizaciones actuales.

Cabe mencionar que la automatización de redes constituye esencialmente un elemento de la revolución de las redes inteligentes, es decir, que este proceso permite a los programas informáticos determinar de qué manera aprovechar los recursos para poder llegar a cumplir los objetivos que se han establecido en un inicio.

Por otro lado, en internet existen comunidades de desarrolladores y documentaciones oficiales que se encargan de brindar los primeros pasos para automatizar dispositivos de red, así́ como para crear y desarrollar plataformas web con distintas tecnologías, bien sea con fines educativos o más profesionales. Analizando algunas soluciones y diferenciando sus ventajas e inconvenientes, se puede obtener lo mejor de cada una de ellas y comenzar a planificar el diseño del proyecto.

Además, podemos encontrar más información al respecto en trabajos anteriores, como es el caso del proyecto [1] realizado en 2022 por Víctor Vázquez Mira, cuyo título es "Automatización de redes: Un caso práctico". En el proyecto se exponen los aspectos fundamentales de la automatización de dispositivos de red, así́ como también de sus principales herramientas para lograrlo. Para demostrar la utilidad de las distintas formas de automatizar redes, se desarrolló́ un caso práctico en el que mediante programación de lenguaje Python y diversas herramientas del lenguaje propio como del entorno en donde se desarrolló, se pudo llegar a un resultado final y posteriormente realizar la comparativa del uso de todas estas tecnologías.

Este proyecto, si bien es cierto, es un buen punto de partida para entender y tener una primera idea sobre la automatización con Python y sus librerías, como también el uso del entorno GNS3 en el lado de la virtualización y simulación de equipos de red.

2.3. Necesidades de información

A continuación, se exponen los principales elementos que se abordarán en el proyecto a nivel teórico, siendo imprescindible conocer dichos aspectos como condición necesaria para comprender a grandes rasgos el funcionamiento de lo que se pretende desarrollar en el proyecto.

2.3.1. Medios de automatización

En Python existen distintas opciones para la automatización de redes, siendo las librerías las que mejor se adaptan para la solución que se quiere lograr en este proyecto. Todas estas librerías son lo suficientemente conocidas en la comunidad de desarrolladores, sobre todo en el ámbito de la automatización, y que a través de los años han podido mejorar sus características y tener hoy en día un gran nivel de madurez que se acoplan muy bien al lenguaje de programación.

Las librerías que se pretenden utilizar en el presente proyecto son dos y esto es así debido a que, aunque cada una de ellas es suficientemente potente, en la práctica son diferentes y aquellos procesos que a alguna le es imposible realizar, puede hacerlo la otra.

Netmiko

Netmiko es una de las principales librerías y de las más populares ya que una de sus principales características es que ofrece una interfaz unificada que permite conectarse y comunicarse con facilidad a los equipos de red de distintos fabricantes mediante el uso del protocolo SSH. Esta librería abstrae muchas de las complejidades que supone configurar un dispositivo de red de manera ordinaria. [2]

En este proyecto, se ha elegido Netmiko porque se trata de una librería de Python altamente compatible con múltiples fabricantes de dispositivos de red, lo que permite a los responsables de la red interactuar con ellos mediante programación y automatizar las tareas administrativas de la misma.

Asimismo, como se ha mencionado antes, Netmiko ofrece una interfaz unificada para interactuar con los equipos de red de distintos fabricantes, lo que agiliza el proceso de

automatización de tareas y permite a un administrador de red emplear los mismos comandos y secuencias de comandos en distintos dispositivos.

Por otra parte, Netmiko es compatible con diversos protocolos de gestión de red, como SSH, Telnet y SNMP, lo que permite a los operadores de red escoger el protocolo que mejor se adapte a sus necesidades.

Otra de las razones por la elección de Netmiko es su gran comunidad de desarrolladores y miembros, permitiendo acceder a una gran variedad de recursos, como documentación, ejemplos de código y foros de discusión, ayudando así al constante crecimiento y evolución de la tecnología haciéndola más estable, robusta y consistente.

NAPALM

Una de las principales dificultades al momento de realizar una conexión a equipos de red es el hecho de que existen distintos fabricantes y los requerimientos de acceso o configuración muchas veces cambian. NAPALM es una librería que propone facilitar de manera unificada el acceso a los dispositivos de red, así como también la obtención de datos y la gestión de los diferentes parámetros de los distintos proveedores.

Para ello, utiliza una API unificada que hace posible acceder a los datos y gestionar la configuración al momento de la automatización, y todo esto independientemente del Sistema Operativo del dispositivo. [3]

Se ha decidido utilizar NAPALM por tratarse de una potente librería Python que hace posible la automatización de tareas de manejo de redes sobre equipos de diferentes fabricantes.

NAPALM también es totalmente compatible con una gran variedad de fabricantes de dispositivos de red, como Cisco, Juniper, Arista, Huawei y más. Gracias a esta gran compatibilidad, cualquier administrador de red puede utilizar NAPALM para automatizar tareas de administración de red en diversos dispositivos, sin necesidad de memorizar los distintos comandos o APIs de cada proveedor.

NAPALM también ofrece una extensa gama de funciones que permiten automatizar las tareas de gestión de red, como la recopilación de información de dispositivos, la configuración de dispositivos y la implementación de políticas de seguridad.

Por último, al igual que Netmiko, NAPALM cuenta con una activa y creciente comunidad de usuarios y desarrolladores, que están dispuestos a ofrecer recursos y soporte a las futuras implementaciones con esta tecnología.

Como dato importante a destacar, tanto Netmiko como NAPALM requieren una curva de aprendizaje mínima y cuentan con una documentación muy completa, lo que los acerca a un amplio abanico de usuarios, desde principiantes hasta expertos en redes.

Otras Tecnologías

Una vez analizadas las tecnologías utilizadas en este proyecto, es importante tener en cuenta otras herramientas de automatización de redes que, si bien no forman parte de la solución, podrían ser útiles para su aplicación en futuras implementaciones. A continuación, se describen algunas herramientas adicionales:

- **Ansible:** Es una solución de automatización que permite realizar tareas de configuración, despliegue y administración de sistemas informáticos de forma autónoma. Ansible emplea un lenguaje específico de dominio (DSL) que se basa en YAML y que permite determinar los distintos niveles del sistema y programar tareas de forma automática.
- **Puppet:** Es una plataforma de control de configuración que sirve para determinar el aspecto deseable de los sistemas y permite automatizar la configuración y control de servidores, aplicaciones y aplicaciones informáticas. Se trata de una potente solución ampliamente utilizada en empresas.
- **Chef:** Es un programa de ayuda a la configuración que emplea un método de aprendizaje programado para automatizar la configuración, despliegue y mantenimiento de infraestructuras informáticas. Se trata de una herramienta sumamente ágil que se ajusta perfectamente a una gran gama de sistemas de infraestructura.
- **SaltStack:** SaltStack es una tecnología de automatización de sistemas que permite la administración de tareas, la orquestación de configuraciones y la supervisión de infraestructuras a mayor escala.

En conclusión, aunque existen múltiples tecnologías disponibles para la automatización de redes, se ha optado por utilizar Netmiko y NAPALM en este proyecto debido a sus características únicas, como la interfaz unificada, la compatibilidad con múltiples dispositivos, la flexibilidad, la facilidad de uso y el soporte activo. Además, se ha valorado especialmente la curva de aprendizaje relativamente baja de estas herramientas, lo que las hace accesibles para una amplia variedad de usuarios con diferentes niveles de experiencia en redes. Al seleccionar las herramientas adecuadas para automatizar tareas de redes, es importante evaluar cuidadosamente las necesidades específicas del proyecto y las características de cada herramienta para lograr una implementación exitosa y eficiente.

2.3.2. Lenguajes de programación y Frameworks

Python

Python es un lenguaje de programación orientado a objetos, interpretado, es decir, que no hace falta que se compile el programa al momento de la ejecución, de alto nivel, semánticamente dinámico y es el adecuado al momento de realizar tareas de automatización o crear sitios web debido a que existen distintas librerías y herramientas que facilitan el trabajo y son muy potentes para su uso en estos ámbitos. [4]

Además, a través de su sencilla sintaxis, Python resulta fácil de aprender y mantener, porque también trae consigo una gran comunidad de desarrolladores que a través de foros de internet ayudan a que el lenguaje crezca más y se mantenga robusto, como también facilitan el avance continuo y rápido del que esté aprendiendo el lenguaje de programación.

Flask

Flask es un Framework web ligero, fácil de usar y útil para desarrollar aplicaciones web con muy pocas líneas de código [5]. Es una opción excelente para crear una aplicación *back end* de monitorización de equipos de redes informáticas, ya que proporciona un conjunto de herramientas y bibliotecas que facilitan y agilizan el desarrollo web. Además, Flask es altamente personalizable y escalable, lo que significa que se puede ajustar para satisfacer las necesidades específicas de una aplicación.

Para el caso de una aplicación de monitorización de equipos de red, Flask podría utilizarse para gestionar peticiones HTTP y datos enviados y recibidos de dispositivos de red. Así, por

ejemplo, podría emplearse para recibir y almacenar datos de monitorización como las estadísticas de tráfico, el uso de CPU, la utilización de memoria, entre otras cosas. En el caso que se requiera, los datos podrían almacenarse en una base de datos para su posterior análisis y visualización.

En lo que respecta al *front end*, Flask resulta útil también para configurar una interfaz de usuario en la que se puedan mostrar los datos de monitorización de forma clara y concisa. Con Flask se pueden desarrollar páginas web dinámicas que actualicen los datos en tiempo real, lo que facilita la visualización de la información de monitorización en tiempo real. Asimismo, con Flask se pueden integrar diversas herramientas de visualización de datos como D3.js y Plotly, permitiendo la creación de gráficos interactivos y adaptables para representar los datos de monitorización.

Así pues, el uso de Flask para crear una aplicación *back end* de monitorización de equipos informáticos de red y un *fron tend* de visualización de datos es una opción excelente por su facilidad de uso, flexibilidad y personalización. Gracias a su conjunto de herramientas y bibliotecas, Flask facilita y agiliza el desarrollo web, lo que permite desplegar aplicaciones de forma más rápida y eficiente. Por otra parte, Flask es sumamente escalable y configurable, lo que significa que puede adaptarse para satisfacer las necesidades específicas de una aplicación de monitorización de equipos de red.

2.3.3. Plataformas de simulación de redes

GNS3

GNS3 es una plataforma de simulación de software de red que permite, entre otras cosas, emular, configurar, testear redes virtuales y reales. [6]

Con GNS3 es posible simular topologías tanto pequeñas como complejas y de esta manera aproximarse al mundo real, replicando el funcionamiento e interconexión de distintos dispositivos de distintas fabricantes, todo esto sin necesidad de disponer de ningún tipo de *hardware* dedicado, como routers o switches. Además ofrece una fácil e intuitiva interfaz gráfica en donde se pueden diseñar y configurar las topologías, enlazando o conectando distintos dispositivos que pueden ser añadidos a través de las imágenes de los distintos sistemas operativos o que también se pueden agregar a través de una máquina virtual que

exista previamente en algún entorno de virtualización, como puede ser VirtualBox, para que de esta manera simplemente se tenga otra opción de emular el equipo, en el caso de que se requiera.

En el mercado existen distintas opciones de software que ayudan a simular redes como por ejemplo EVE-NG que, a diferencia de GNS3, se instala como una máquina virtual, no es de código abierto y no ofrece todas sus funcionalidades en la suscripción gratuita. En el caso de GNS3, su facilidad para trabajar desde tu propio ordenador en local facilita las cosas al momento de descargar todo lo necesario para poder utilizar los distintos dispositivos que están disponibles en su tienda oficial y funciona perfecto tanto para pequeñas y grandes topologías, teniendo en cuenta siempre la potencia del ordenador en el que se trabaje.

3. Objetivos y alcance

3.1. Objetivos

Los objetivos que se pretenden lograr en este proyecto son:

- Comprender los aspectos básicos de automatización de red y aplicarlos en una solución que sea funcional.
- Agilizar la configuración de distintos equipos de red de manera automatizada.
- Aumentar la eficiencia en las tareas de configuración de equipos de red.
- Gestionar recursos y disminuir el tiempo en las tareas repetitivas.
- Reducir la probabilidad de error humano provocado por el uso de procesos manuales al momento de manipular la red.
- Establecer una conexión entre la información obtenida por parte de la topología de red y el entorno web.
- Facilitar la monitorización de la configuración de los equipos de red gracias a la obtención de la información necesaria a través de una plataforma web.

3.2. Público potencial

El público potencial son todas aquellas personas que tienen la responsabilidad de administrar equipos de red, ya que en el día a día, las tareas que realizan suelen ser repetitivas y rutinarias, por lo que este proyecto propone ofrecer y facilitar una solución distinta al método clásico y así reducir el tiempo y costes de manera significativa.

3.3. Alcance

El alcance establece hasta qué punto puede llegar el proyecto y cuáles son sus limitaciones de cara a lograr los objetivos de este.

A continuación, se indica el alcance que conlleva realizar el proyecto:

- Definición de los requerimientos tanto funcionales como tecnológicos.
- Elaboración del estudio de la viabilidad del proyecto.

- Evaluar y aprender algunas de las herramientas existentes para el despliegue automatizado de redes.
- Diseñar y desarrollar el despliegue, configuración, y la correcta administración
- de los diferentes equipos de red.
- Implementar una plataforma web para probar el entorno y su correcto
- funcionamiento.

4. Metodología

En la fase inicial de este proyecto ha de tenerse en cuenta tanto el desarrollo y la implementación de la solución como los distintos puntos del proyecto que son necesarios para cumplir con los objetivos. Es decir, además del resultado final que se quiere lograr en este proyecto, es imprescindible antes realizar este mismo documento explicando cada punto como por ejemplo los requerimientos, la planificación, presupuestos, análisis de viabilidad.

- Elaboración del anteproyecto.
- Definición de los requerimientos
- Estudio de viabilidad
- Diseño del producto
- Desarrollo del producto
- Documentación de la memoria

4.1. Metodología de desarrollo de software

Es preciso que un proyecto informático disponga de un esquema de desarrollo cuyo objetivo sea definir la metodología a seguir a lo largo de todo el proyecto. En este sentido, podemos distinguir dos grandes grupos en los que se dividen estos modelos, sean por un lado los llamados modelos tradicionales y por otro los ágiles. [7]

Hoy en día, las metodologías ágiles se están imponiendo y se utilizan a gran escala, incluso en las grandes empresas, ya que contribuyen enormemente al trabajo en equipo. Por este motivo, para el desarrollo de este proyecto se utiliza uno de los modelos ágiles denominado SCRUM, debido a que esta metodología hace hincapié en la agilidad y la adaptabilidad, ya que funciona de forma cíclica.

4.1.1. Fases en Scrum

Todas las etapas de SCRUM pertenecen a un mismo fin que pretende satisfacer los requisitos y necesidades del jefe de proyecto y, a su vez, respetar las fechas de entrega del proyecto.

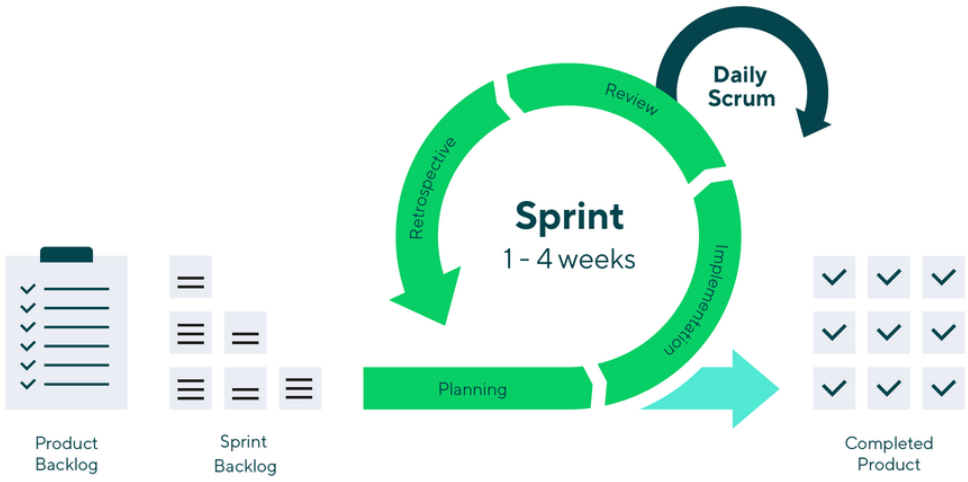

Fig. 4.1. Fases SCRUM

Como se puede apreciar en la imagen anterior, entre las principales características de la metodología SCRUM se pueden destacar las siguientes fases:

- **Inicio:** En esta primera fase se realiza el estudio y análisis inicial del proyecto mediante la definición concreta de las exigencias fundamentales que debe cumplir el sprint. De acuerdo con las metodologías ágiles, un sprint es un mini-proyecto cuya duración no excede de un mes y que tiene como meta alcanzar los objetivos globales y concretos del proyecto.
- **Planificación y estimación:** En esta fase es cuando se determinan cuestiones del proyecto como las funcionalidades, objetivos, posibles riesgos, periodos de entrega, etc.
- **Implementación:** En la implementación o desarrollo, el equipo tiene la responsabilidad de desempeñar las tareas que se asignen a un sprint específico.
- **Revisión y retrospectiva:** Cuando todo haya sido trazado y puesto en práctica, es necesario revisar el avance del trabajo, proceso cuya finalidad no es otra que la retroalimentación del grupo o la valoración interna del trabajo.

4.1.2. Roles en Scrum

La metodología Scrum se caracteriza por tener tres roles fundamentales con funciones claramente identificables y bien diferenciadas. Entre ellos nos encontramos al Scrum Master, Product Owner y Equipo de Desarrollo. A continuación, se muestra para cada rol sus características y sus aspectos más destacados.

- **Scrum Master:** Se le define como un líder, ya que es responsable de garantizar que se sigan las directrices establecidas en lo descrito en la Guía Scrum, como también de velar por la eficacia y el logro de los objetivos marcados para todo el equipo, y además estar presente para orientar y tratar de ayudar a quien lo necesite.
- **Product Owner:** A diferencia del resto de roles que se definen más adelante, el rol del Product Owner solo puede ser asignado a una única persona y la función principal que realiza esta persona es la de hacer de intermediario de las necesidades de las diversas partes involucradas en el desarrollo del producto.
- **Equipo de Desarrollo:** Es el grupo de personas encargadas de diseñar y desarrollar el producto en cada sprint.

5. Definición de requerimientos

Con el fin de alcanzar los objetivos que se han decidido, a continuación, se indican los requerimientos funcionales que debe alcanzar el producto final.

- Establecer una conexión con los dispositivos de red para acceder a su configuración.
- Configurar los dispositivos de red de manera automatizada.
- Obtener información de los dispositivos.
- Mostrar la información a través de una plataforma web.

En el caso de los requerimientos tecnológicos, es necesario cumplir lo siguiente:

- Implementar una solución de automatización de equipos de red con ayuda del lenguaje de programación Python y sus librerías Netmiko y NAPALM.
- Realizar el despliegue de topologías de red a través de la plataforma de emulación GNS3.
- Disponer de un entorno de virtualización para el uso de GNS3 y los posibles equipos que cuenten con un entorno gráfico para mejor visualización.
- Desarrollar la plataforma web utilizando el Framework Flask de Python.

6. Topología de red

6.1. Diseño

El primer paso antes de empezar a programar y probar la automatización de los equipos o incluso antes de diseñar el entorno web en el que se van a monitorizar, es el de diseñar un entorno de red que sea adecuado para el propósito de este trabajo.

Para este caso en concreto, luego de valorar distintas opciones de topologías disponibles, se ha decidido utilizar una en concreto, la cual es un recurso que se nos otorgó en la asignatura de tercer año denominada Sistemas y Servicios. Esta topología cuenta con lo necesario para hacer las pruebas y puesta en marcha de nuestra solución automatizada. A continuación, presentamos la topología montada en el entorno de emulación GNS3 y posteriormente una descripción detallada de la misma.

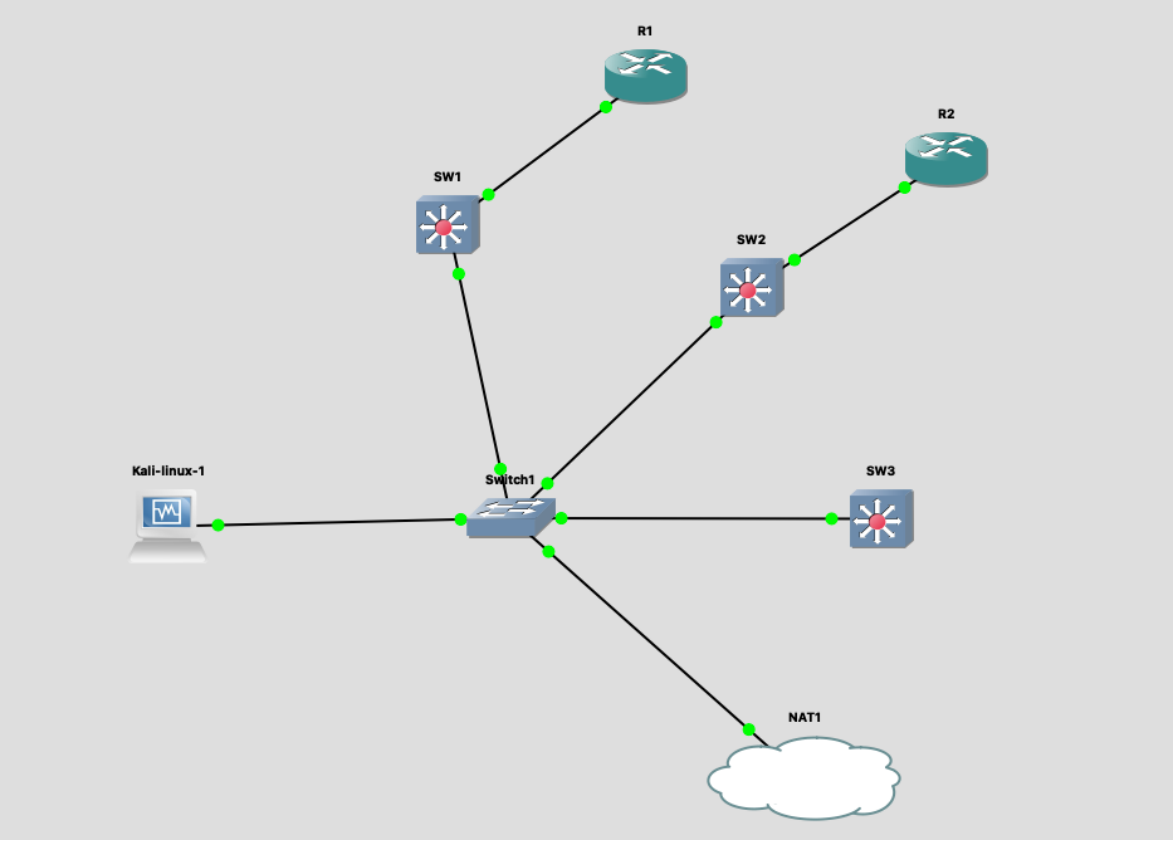

Fig. 6.1. Topología de red

6.2. Características

En la topología de red mostrada en la figura anterior, la cual se utilizará para las pruebas con las librerías de automatización, se pueden encontrar los siguientes dispositivos:

- Una máquina Kali Linux
- Un switch no gestionable
- 3 switches IOSvL2
- 2 routers IOSv
- 1 conexión NAT hacia el exterior

Como se ha mencionado con anterioridad, el diseño y despliegue de la topología de red se ha tenido en cuenta utilizando los recursos de la plataforma GNS3, la cual se ajusta a las necesidades del proyecto ya que provee de distintos equipos y terminales de distintas marcas y versiones, haciendo de este entorno una gran aproximación del mundo real.

6.3. Configuración del entorno

Es necesario mencionar en este apartado el montaje de la topología de red mostrado en la figura 6.1., pues la plataforma GNS3, con la cual estamos trabajando, requiere de cierta configuración previa y posterior para el buen funcionamiento del entorno y además su uso conlleva una búsqueda de las imágenes de los distintos equipos a utilizar.

Pues bien, el primer requerimiento de esta plataforma, además de la aplicación en sí la cual se ha descargado en formato ejecutable en el ordenador del estudiante, es una máquina virtual en la cual tiene como fin alojar las distintas imágenes de los equipos a descargar. Para lograr esto, GNS3 dispone de una máquina virtual llamada GNS3 VM, la cual está disponible para su libre descarga y uso desde su sitio web oficial.

Una vez descargada tanto la aplicación como su máquina virtual, el siguiente paso es importar la máquina virtual en un entorno de virtualización, en este caso se ha optado por utilizar Virtual Box, debido a su simple instalación y uso.

El siguiente requisito es descargar las imágenes de los dispositivos a utilizar. GNS3 proporciona una lista diversa de dispositivos de distintos fabricantes por lo que obtenerlas es tan fácil como buscar la versión que más se acomode a nuestras necesidades y descargarla

directamente en su sitio web oficial. Para este proyecto se han descargado tanto las imágenes del switch IOSvL2 y del router IOSv, por otro lado, el resto de los terminales y recursos se han obtenido de la propia plataforma ya que vienen con la instalación de esta.

Luego de descargar los elementos a automatizar, es necesario proporcionar en la plataforma una forma de poder conectarse a estos dispositivos. Es por eso, que tanto la parte de programación y conexión por parte del que toma el papel de administrador de red, es decir, el estudiante, se desarrolla a través de una máquina Kali Linux, la cual se ha importado a la plataforma GNS3 como una máquina virtual de VirtualBox, es decir, que esta máquina proviene del ordenador del estudiante. Esta máquina virtualizada tiene como fin realizar las tareas de conexión, programación y compilado del código final, aprovechando su terminal gráfica para mayor facilidad y agilidad al momento de crear las pruebas y monitorear la configuración.

Otro aspecto de la configuración de la maquina Kali Linux, es que este debe obtener su IP de forma dinámica, es decir a través de DHCP, y esto es posible a través del elemento nativo NAT, que existe en GNS3. Con este elemento, es posible obtener una conexión a internet desde nuestra máquina de administración.

Por último, teniendo en cuenta que la máquina en donde se realizan las tareas de automatización posee una IP asignada dinámicamente, nos aseguramos primero el rango en el que esta se encuentra. En este caso, es del tipo 192.168.122.x, por lo que el siguiente paso, es configurar los equipos de red con una IP dentro de ese rango para hacer posible la conexión dentro de la misma red. Con esta configuración inicial, los dispositivos, tanto switches como routers, ahora poseen las IPs 192.168.122.10, .11, .12, .13, .14 y .15 y se pueden comunicar entre ellos.

7. Planteamiento

Como parte del proceso para llegar a cumplir con los objetivos del proyecto, utilizaremos el lenguaje de programación Python y en conjunto de sus librerías Netmiko y NAPALM para poder desarrollar un *script* que pueda lograr la exitosa configuración de los equipos de red, pero también es necesario definir los casos prácticos, es decir, las tareas que se tienen que implementar con estas librerías, con el fin de tener una visión clara de la solución a la que hay que llegar.

7.1. Tecnologías de automatización

Las librerías de automatización de Python que se utilizan a lo largo del desarrollo del proyecto se han puesto a prueba durante la fase posterior a la entrega del ante proyecto con tal de conocer, aprender, y comprobar sus funcionalidades. Estas requieren seguir cierta sintaxis en el código de Python y además una previa configuración en los equipos de red, como, por ejemplo, habilitar la conexión SSH en cada uno de ellos. Pues bien, a continuación, se muestra la configuración inicial que se necesita en ambos casos.

Ambas librerías, es decir Netmiko y NAPALM, como ya se ha mencionado antes, permiten conectarse a los dispositivos de red a través del protocolo de conexión SSH, protocolo que se caracteriza por ser mucho más seguro que TELNET.

Estando en el entorno de simulación de red, el primer paso es habilitar este tipo de conexión en cada uno de los dispositivos a los que se requieran configurar, y además de eso es necesario asignarle ciertos parámetros que se muestran en la siguiente figura:

```
enable
conf terminal
username developer password cisco123
username developer privilege 15

line vty 0 4
     login local
     transport input all

ip domain-name cisco.com
crypto key generate rsa
1024
end
```

Fig. 7.1. Configuración inicial de dispositivos para habilitar conexión SSH

Por otro lado, una vez hecho esta parte de configuración inicial, se requiere agregar el servidor remoto en la lista de hosts conocidos dentro del fichero denominado known_hosts de la máquina en donde queremos que se envíen los comandos de configuración, es decir, en nuestra máquina Kali Linux. Para poder agregar el host y acceder remotamente desde la máquina Kali Linux, es necesario escribir el siguiente comando:

```
# ssh developer@direccion_IP_terminal
```

Fig. 7.2. Comando de terminal para conectarse a los dispositivos

La figura muestra primero el protocolo que se utiliza para la conexión, en este caso SSH, seguido del nombre de usuario que se le ha asignado a cada uno de los dispositivos, y por último la dirección IP asignada del dispositivo final. Una vez ejecutado el comando, se nos solicitará ingresar la contraseña del usuario al que se necesita acceder. Si es la primera vez que accedemos a este, se nos mostrará un mensaje de que dicho host no está agregado a la lista de hosts conocidos y por ende hay que aceptar con una respuesta afirmativa.

Además, como se ha visto previamente, nuestro entorno de red está conformado en su totalidad por equipos IOS, y para realizar cambios en la configuración de estos, en el caso de NAPALM, es necesario habilitar SCP en cada uno de ellos.

```
conf terminal
ip scp server enable
end
```

Fig. 7.3. Comando para habilitar SCP

Los drivers IOS de NAPALM funcionan mejor con el protocolo SCP, el cual se basa en SSH para mayor seguridad.

En resumen, tanto Netmiko como NAPALM pueden realizar diversas tareas de configuración conectándose a los dispositivos a través del protocolo SSH, por lo cual es imprescindible configurar esta conexión en cada dispositivo de nuestra topología. A partir de aquí ya podemos realizar tareas de automatización con la librería.

7.2. Casos prácticos

El contexto en el que nos encontramos al momento de realizar el trabajo de automatización es en un entorno de simulación de redes a través de la plataforma GNS3, la cual, se aproxima lo mejor posible a un entorno real y por lo cual, para realizar una mejor comprensión y realización de las distintas tareas, se ha decidido definirlas en este apartado. Se han escogido las que normalmente se suelen repetir y que para el administrador de redes le puede costar más tiempo configurar a diario.

- Configuración básica de switches
- Configuración básica de routers
- Creación de VLANs en switches
- Implementación de ACLs en routers

La selección de estos casos surge al momento de identificar las tareas de configuración rutinarias que se han podido realizar a lo largo de las distintas asignaturas del grado dedicadas a la gestión y administración de redes.

Para poder demostrar la utilidad de las herramientas que se utilizan en este trabajo, comenzamos con una configuración sencilla tanto de los switches como también de los routers, teniendo en cuenta una serie de instrucciones que se encuentran presentes dentro de

un archivo de texto. A través de esto, lo que se busca es simplificar la configuración de los dispositivos, reduciendo el tiempo que requiere administrar cada uno de ellos.

Por otra parte, como segundo caso práctico lo que se busca demostrar es la automatización al momento de crear VLANs, las cuales, si bien es cierto que el hecho de crearlas de una forma manual no supone mucha complejidad, sí que suele ser una tarea tediosa cuando estas son numerosas ya que sus instrucciones son repetitivas.

Por último, otra de las configuraciones que se tienen en cuenta al momento de filtrar qué tipo de paquetes pueden acceder o no a un determinado dispositivo, y que son, en muchos casos, de gran utilidad gracias a la seguridad que supone implementarlas, son las ACLs. En este último caso práctico, lo que se busca es crear ACLs en algunos dispositivos, estableciendo distintas políticas de seguridad para cada uno.

8. Implementación

A continuación, se abordan los aspectos más relevantes de la parte hecha con Python, como pueden ser estructura de código, ficheros de configuración, resultados, etc. Con esto se pretende demostrar la manera en la que se ha dado solución a los distintos casos prácticos especificados en la fase de planteamiento. Seguidamente, se explican en detalle las cualidades más destacables de cada implementación.

A fin de contar con un grupo similar de pruebas con ambas tecnologías, Netmiko y NAPALM, aplicaremos estos casos del mismo modo o de forma similar dentro de las limitaciones de cada tecnología.

Las implementaciones en ambos casos cuentan con una clase de Python en común, denominada connect.py, la cual almacena la información de los dispositivos, y también realiza la conexión tanto con Netmiko como también NAPALM.

```
cisco_devices = {
    'iosv_l2_SW1': {
        DEVICE_TYPE: 'cisco_ios',
        IP: '192.168.122.10',
        USERNAME: 'developer',
        PASSWORD: 'cisco123'
    },
    'iosv_l2_SW2': {
        DEVICE_TYPE: 'cisco_ios',
        IP: '192.168.122.11',
        USERNAME: 'developer',
        PASSWORD: 'cisco123'
    },
    'iosv_l2_SW3': {
        DEVICE_TYPE: 'cisco_ios',
        IP: '192.168.122.12',
        USERNAME: 'developer',
        PASSWORD: 'cisco123'
    },
    'iosv_l3_R1': {
        DEVICE_TYPE: 'cisco_ios',
        IP: '192.168.122.13',
        USERNAME: 'developer',
        PASSWORD: 'cisco123'
    },
    'iosv_l3_R2': {
        DEVICE_TYPE: 'cisco_ios',
        IP: '192.168.122.14',
        USERNAME: 'developer',
        PASSWORD: 'cisco123'
    }
}
```

Fig. 8.1. Diccionario en Python que contiene los dispositivos

En la figura, se puede apreciar un diccionario que almacena los cincos dispositivos que se utilizan como ejemplo en este proyecto. Cada uno de ellos posee un tipo, una dirección IP, la cual debe estar dentro del rango anteriormente mencionado, un usuario y una contraseña. Estos dos últimos campos, son necesarios para poder conectarse a través de SSH.

8.1. Automatización de red con Netmiko

Tal como se ha mencionado en el punto anterior, la implementación con Netmiko utiliza una clase que contiene, en este caso, una función que realiza la conexión con los dispositivos utilizando esta librería.

```
def netmiko_connect(device_type):

    print(f"***** Conectándose a {cisco_devices[device_type][IP]} *****\n")

    connection = Netmiko(
        cisco_devices[device_type][IP],
        device_type=cisco_devices[device_type][DEVICE_TYPE],
        username=cisco_devices[device_type][USERNAME],
        password=cisco_devices[device_type][PASSWORD]
    )

    return connection
```

Fig. 8.2. Función que realiza la conexión con Netmiko

La función crea una instancia de la clase Netmiko utilizando los campos del dispositivo como parámetros y retornando este nuevo objeto. Esto permitirá poder acceder al dispositivo y realizar cambios en su configuración. La clase la podemos importar directamente en donde tengamos nuestro código que realiza la automatización y de esta manera conectar los equipos que sean necesarios.

Además, cabe mencionar que la implementación con esta librería está enfocada en la utilización de ficheros que contienen la configuración deseada para los equipos, la cual se explica más adelante según cada caso. Con esto se busca en el proyecto estandarizar el uso de estos ficheros como única fuente de información de datos que la librería tiene que reconocer para poder logar la configuración en los equipos, con el fin de que, si en un futuro las instrucciones cambian y/o escalan dentro del fichero, la configuración se pueda ejecutar sin ningún problema independientemente de su longitud.

```
for device_name in [SW1, SW2, SW3, R1, R2]:

    connection = netmiko_connect(device_name)
```

Fig. 8.3. Ciclo de iteraciones sobre los dispositivos para su tratamiento con Netmiko

Antes de pasar a explicar cada caso práctico, cabe mencionar que las pruebas de configuración se realizan en un ciclo que itera sobre cada dispositivo que se encuentra en nuestra topología, tal y como se muestra en la figura. A partir de aquí, se utiliza el nombre de cada uno de los dispositivos como referencia para poder utilizar la función que se ha explicado en la figura 8.2, y que luego de retornarnos una instancia con la conexión hecha, esta se guarda en una variable que nos será de utilidad más adelante cuando se desee hacer una llamada a alguna de las funciones que nos proporciona la librería.

8.1.1. Configuración básica de switches

Empezamos las pruebas de la librería gestionando la configuración básica de los tres switches que se encuentran en nuestra topología. Para poder resolver este caso práctico, es necesario, antes que nada, poder gestionar el fichero con el que se está trabajando. En pocas palabras, se necesita extraer cada línea del fichero, para luego poder procesarlas con la librería Netmiko.

```
def read_config(config_file):
    with open(config_file) as f:
        config = f.read().splitlines()

    return config
```

Fig. 8.4. Función que lee y extrae el contenido de un fichero en líneas

Con el fin de poder extraer las líneas se ha creado una función, mostrada en la figura anterior, que puede ser reutilizada para cualquier caso, simplemente especificando el nombre del fichero como parámetro, luego se procesan las líneas y se retornan con el fin de poder utilizarlas luego. Algunas de las funciones que son propias de esta librería, como en el caso de la que cumple la función de enviar comandos de configuración hacia el dispositivo, permiten en su mayoría pasarles por parámetro una lista de comandos, es decir una lista que

contenga una o más cadenas de texto. Es por eso, que se ha decidido extraer las líneas y convertirlas en una lista en donde cada línea es un elemento. De esta manera no surgirán problemas al momento de utilizar el contenido del fichero con Netmiko.

```
ip name-server 8.8.8.8 255.255.255.255
ip domain-lookup
vtp mode server
interface range Gi 0/2 - 3, Gi 1/0 - 3, Gi 2/0 - 3, Gi 3/0 - 3
shutdown
```

Fig. 8.5. Contenido del fichero basic_switch_config

El contenido del fichero mostrado en la figura especifica cada uno de los comandos que se aplicarán en el switch, teniendo instrucciones que van desde el host detallado que hace de proveedor de información de nombres para DNS hasta el apagado de las interfaces que no están conectadas, todas estas líneas son leídas por la función mostrada en la figura 8.4. y guardadas para su futura utilización. Es importante tener en cuenta que los comandos utilizados, en este caso para dispositivos IOS, tienen que ser válidos para este tipo de sistema operativo, en caso contrario, el dispositivo no podrá ser configurado correctamente ya que no reconocerá la sintaxis errónea que está procesando.

```
lines = read_config(config_file)
print(f"----- Configurando {config_message} {device_name} -----\n")
change_config(connection, lines)
```

Fig. 8.6. Código en Python que realiza la configuración de los switches con Netmiko

En el código de la figura, podemos observar que se extraen las líneas necesarias de cada comando de configuración a partir del nombre del fichero, que en este caso es el de los switches, y estas se envían a una función que aplica la nueva configuración. Una vez obtenidas las líneas que contienen las instrucciones, utilizamos la función de nuestra clase connect.py que realiza la conexión a través de Netmiko. Cabe mencionar, tal como se había mencionado antes, en la figura 8.3. cada vez que se itera sobre cada uno de los dispositivos, se guarda una conexión, por lo que este es el momento de utilizar dicha conexión para aplicar los cambios. La función acepta la conexión y las líneas de configuración como parámetros.

```
def change_config(connection, lines):
    try:
        output = connection.send_config_set(lines)
        return print(f"{output}\n\n----- Configuración exitosa -----\n")
    except:
        return print("\n\n----- Configuración fallida -----\n")
```

Fig. 8.7. Función que configura un dispositivo utilizando Netmiko

Dentro de la función que realiza la configuración, utilizamos la instancia de Netmiko recibida como parámetro y con esta podemos hacer llamadas a distintas funciones que pertenecen a la librería. La función que nos ocupa en esta ocasión es la denominada send_config_set, la cual recibe por parámetro las líneas de configuración previamente obtenidas desde el fichero. Internamente, la función envía la secuencia de comandos de configuración al dispositivo de red remoto y espera una respuesta del dispositivo después de cada comando. Cada comando se envía como una cadena de texto a través de la conexión SSH. Tras enviar cada comando de configuración, la función espera una respuesta del dispositivo de red remoto que también es recibida a través de SSH. Cuando se han enviado todos los comandos de configuración, la función cierra la conexión SSH con el dispositivo de red remoto. Para finalizar, nos devuelve una cadena de texto de todos los comandos enviados como resultado, que luego imprimimos para poder visualizar el correcto funcionamiento del código.

8.1.2. Configuración básica de routers

Al igual que con los switches, los routers se pueden configurar manteniendo la misma sintaxis vista previamente. Es por eso, que aprovechamos el código anterior para poder configurar los routers.

Esta vez utilizamos un fichero de configuración distinto ya que los routers tienen ciertas instrucciones específicas, lo cual hace que los diferencie de los switches en cuanto a los comandos que pueden leer y procesar cada uno.

```
ip route 0.0.0.0 0.0.0.0 192.168.122.1
ip name-server 8.8.8.8 255.255.255.255
ip domain-lookup
router ospf 1
network 0.0.0.0 255.255.255.255 area 0
```

Fig. 8.8. Contenido del fichero basic_router_config

El contenido del fichero mostrado en la figura especifica cada uno de los comandos que se aplicarán en el router, teniendo instrucciones que van desde la creación de una ruta estática hasta la aplicación del enrutamiento OSPF. Todas estas líneas son leídas por la función mostrada en la figura 8.4. y guardadas para su futura utilización.

El código utilizado para gestionar la configuración es exactamente el mismo al que se ha mostrado en las figuras 8.6 y 8.7 que realizan tanto la parte de obtención de los comandos desde el fichero y la posterior configuración de los switches.

8.1.3. Creación de VLANs en switches

Para la creación de las redes de área local virtuales se ha buscado hacerlo con un número razonable de cinco para poder comprobar su efectividad. Con esto, buscamos no solo verificar que se han aplicado las configuraciones correctamente, sino también que se pueden crear más de una VLAN.

```
vlan 10
 name VLAN10
vlan 20
 name VLAN20
vlan 30
 name VLAN30
vlan 40
 name VLAN40
vlan 50
 name VLAN50
```

Fig. 8.9. Contenido del fichero vlans_conf

La figura muestra la sintaxis correcta para los dispositivos IOS, los cuales cuentan con el id de la VLAN y su respectivo nombre que se le desea añadir para diferenciarlos.

```
# Creación de VLANs en los switches
if device_name == SW1 or device_name == SW2 or device_name == SW3:
    config_file = 'vlans_conf'
    lines = read_config(config_file)
    print(f"----- Creando VLANs -----\n")
    change_config(connection, lines)
```

Fig. 8.10. Código en Python para crear VLANs con Netmiko

Dado que las VLAN son propias de los switches, se necesita aplicarse únicamente a estos, pues si un router obtuviese estos comandos, no habría resultados correctos, o en su defecto, las funciones de Netmiko nos devolvería un error.

Una vez comprobados los dispositivos a los que se les aplicarán los comandos, se procede a realizar el mismo proceso que hemos visto previamente en las configuraciones básicas de cada dispositivo. En este caso, cada VLAN se crea una por una asignándole su respectivo nombre.

8.1.4. Implementación de ACLs en routers

Utilizando la tecnología Netmiko, se ha podido resolver el caso práctico de automatización de configuración de ACLs en routers en un entorno GNS3. Para ello, nuevamente se ha seguido la metodología que hemos visto en todos los casos que consiste en extraer la configuración de comandos a aplicar en las ACLs de un archivo de texto plano, el cual se adjunta a continuación:

```
ip access-list extended SSH_TRAFFIC
 permit tcp 10.1.1.0 0.0.0.255 10.2.2.0 0.0.0.255 eq 22
 deny ip any any
ip access-list extended ALLOW_HTTP_OUT
 permit tcp 192.168.1.0 0.0.0.255 any eq 80
ip access-list extended BLOCK_TELNET_IN
 deny tcp host 10.10.10.1 any eq 23
ip access-list extended ALLOW_ICMP
 permit icmp any any
ip access-list extended ALLOW_DNS
 permit udp any any eq domain
```

Fig. 8.11. Contenido del fichero acls_conf

Dentro del archivo de texto nos podemos encontrar con listas como la primera la cual permite el tráfico SSH entrante desde una subred específica y bloquear todo lo demás, la segunda lista que permite el tráfico HTTP saliente de una subred específica, otra que bloquea el tráfico Telnet entrante desde una dirección IP específica, además de otra que hace posible el tráfico ICMP desde cualquier dirección IP, y por último tenemos una lista que permite el tráfico DNS entrante y saliente.

Luego, se procedió a programar el siguiente código Python utilizando Netmiko para aplicar dicha configuración en cada router:

```python
# Implementación de ACLs en los routers
if device_name == R1 or device_name == R2:
    config_file = 'acls_conf'
    lines = read_config(config_file)
    print(f"----- Implementando ACLs -----\n")
    change_config(connection, lines)
```

Fig. 8.12. Código en Python para implementar ACLs con Netmiko

El código diferencia los dispositivos que son switches de los routers para poder aplicar las políticas de acceso únicamente a estos últimos. Una vez hecho esto, se procede a leer el archivo de texto y luego establecer las listas en el dispositivo. Como ya hemos visto previamente en los otros casos prácticos, la función change_config realiza todo el proceso de envío de comandos hacia el dispositivo final.

De esta forma, se logra una automatización eficiente y sencilla para la configuración de ACLs en routers utilizando Netmiko, lo cual facilita en gran medida la administración de redes y la implementación de políticas de seguridad en los sistemas informáticos.

8.2. Automatización de red con NAPALM

A pesar de que se tenía pensando en un principio trabajar con cinco dispositivos con el fin de implementar las configuraciones utilizando las librerías Netmiko y NAPALM, finalmente luego de realizar distintas pruebas en el entorno de GNS3, nos percatamos de ciertos problemas para poder conectarse y configurar los dispositivos debido a temas de rendimiento. Dentro del entorno GNS3 los motivos de rendimiento al utilizar NAPALM para configurar múltiples dispositivos pueden verse influenciados por factores específicos del entorno de simulación de red como pueden ser:

- **Recursos de CPU y memoria de la máquina virtual de GNS3:** En caso de que los dispositivos de red que se están configurando se estén ejecutando en una máquina virtual de GNS3, lo cual es así, es posible que los recursos de CPU y memoria de la máquina virtual limiten el desempeño de la configuración.
- **Ancho de banda de la red virtual de GNS3:** Puede ocurrir que la red virtual GNS3 tenga un ancho de banda limitado, y esto afecte al rendimiento de la operación de configuración.
- **Tipo de dispositivo de red:** En GNS3, algunos tipos de dispositivos de red pueden requerir más recursos que otros.

Luego de realizar distintas pruebas en el entorno, se llegó a la conclusión de que efectivamente era imposible seguir con la configuración utilizando NAPALM por lo que era necesario recurrir a una solución que permita seguir desarrollando el trabajo. Es por eso, que se decidió realizar las diferentes pruebas con tan solo dos de ellos, facilitando así la velocidad al momento de que la librería pueda realizar las tareas necesarias que se les solicita.

Para la parte de la implementación utilizando la librería NAPALM se utiliza una función que al igual que con Netmiko, crea la conexión con los dispositivos.

```python
def napalm_connect(device_name, device):

    print(f"***** Conectándose al dispositivo {device_name}: {device[IP]} *****\n")
    splitted = device[DEVICE_TYPE].split('_')
    device_type = splitted[1]
    driver = napalm.get_network_driver(device_type)

    splitted_device = device_name.split('_')
    device_element = splitted_device[2]

    connection = driver(
        hostname=device[IP],
        username=device[USERNAME],
        password=device[PASSWORD],
        timeout=60,
        optional_args={'secret' : 'enable_password',
                       'global_delay_factor': 2,
                       'keepalive': True,
                       'inline_transfer': True,
                       'alt_host_keys': True,
                       'allow_agent': True,
                       'ssh_strict': True,
                       'system_host_keys': True,
                       'session_timeout': 60,
                       'verbose': True,
                       'prompt': f'{device_element}#'}
    )

    return connection
```

Fig. 8.13. Función que realiza la conexión con NAPALM

La función recibe como parámetro tanto el nombre del dispositivo que se va a conectar como también el dispositivo mismo que proviene de la librería. Con esta información, el proceso que realiza la función es primero obtener el tipo de dispositivo que le llega mediante parámetro, y esto es posible gracias a uno de los campos dentro del diccionario el cual es el tipo de dispositivo. Ahora, el siguiente paso es utilizar la librería NAPALM para poder realizar la conexión. Esto se soluciona con el método proveniente de la librería denominado get_network_driver, el cual recibe como parámetro el nombre del sistema operativo del dispositivo, en este caso, IOS y luego, este método prepara todo lo necesario para poder realizar la conexión.

El resultado de este método externo se puede asignar a una variable que utilizaremos para poder crear una instancia de la conexión con el dispositivo, utilizando los campos del dispositivo para luego retornar el resultado para poder utilizarlo más adelante.

Para realizar las pruebas con NAPALM también se utilizan ficheros los cuales serán los que tengan la información de los comandos a aplicar en los dispositivos.

```
for device_name, device in devices.items():
    if device_name == SW2 or device_name == SW3 or device_name == R2:
        continue

    napalm_device = napalm_connect(device_name, device)
    napalm_device.open()
```

Fig. 8.14. Ciclo de iteraciones sobre los dispositivos para su tratamiento con NAPALM

Antes de pasar a explicar cada caso práctico, cabe mencionar que las pruebas de configuración se realizan en un ciclo que itera sobre el primer switch (SW1) y el primer router (R1), los cuales se encuentran en nuestra topología y que anteriormente se comentó serían los principales y únicos equipos con los que se realizan las pruebas. Para aprovechar el diccionario de dispositivos que se ha mostrado en la figura 8.1. se ha realizado la comprobación de que, si el dispositivo a tratar en ese momento es alguno de los que previamente se han mencionado, los descarte inmediatamente del proceso y prosiga a tratar el siguiente hasta encontrar a los que nos ocupan en esta parte de la implementación. Una vez identificado el equipo a tratar, se utiliza el nombre de cada uno de los dispositivos como referencia para poder utilizar la función que se ha explicado en la figura 8.13., y que luego de retornarnos una instancia con la conexión hecha, esta se guarda en una variable que nos será de utilidad más adelante cuando se desee hacer una llamada a alguna de las funciones que nos proporciona la librería. Por ejemplo, antes de realizar alguna configuración es necesario primero abrir la conexión. Esto es posible gracias a la función open que sirve para entablar una conexión con un dispositivo de red que emplea un determinado controlador, que es propio de cada fabricante y de cada modelo de dispositivo. La función permite establecer la conexión y retorna un objeto de sesión que hace posible la ejecución de comandos y configuraciones en el dispositivo.

8.2.1. Configuración básica de switches

El primero de los casos es la configuración básica de los switches, que en este caso es solo uno. Pues bien, al igual que con Netmiko, la configuración se realiza a partir de un fichero que contiene todos los comandos necesarios para probar la librería. Por otra parte, como se

ha visto anteriormente, este fichero ha de procesarse para su posterior utilización en cada caso.

```
def read_config(config_file):
    with open(config_file) as f:
        config = f.read()

    return config
```

Fig. 8.15. Función que lee y extrae el contenido de un fichero en cadena de texto

Para poder resolver el tema de los ficheros y su contenido, se ha programado una función que abre el fichero y nos retorna una cadena de texto, que a diferencia de Netmiko, es importante no separarlas por comando ya que las funciones de NAPALM usualmente reciben como parámetro toda la cadena de texto con los comandos que se necesitan aplicar.

El contenido del fichero es exactamente el mismo que se ha mostrado en la figura 8.5. por lo que no hace falta entrar en detalles y en su lugar comentaremos cómo funciona NAPALM al momento de enviar la configuración hacia los dispositivos.

```
config = read_config(config_file)
print(f"\n----- Configurando {config_message} {device_name} -----\n")
change_config(napalm_device, config)
```

Fig. 8.16. Código en Python que realiza la configuración de los switches con NAPALM

El bloque de código mostrado procesa el fichero que se le ha indicado dependiendo de si el equipo con el que se está trabajando es un *switch* o un *router* y esta cadena de comandos es almacenada en una variable que luego se utiliza para enviar las instrucciones al dispositivo que toque.

```python
def change_config(napalm_device, config):
    try:
        napalm_device.load_merge_candidate(config=config)
        print(napalm_device.compare_config())
        napalm_device.commit_config()
        return print("\n----- Configuración exitosa -----\n")
    except:
        return print("\n----- Configuración fallida -----\n")
```

Fig. 8.17. Función que configura un dispositivo utilizando NAPALM

Como parte del código de los casos prácticos se ha decidido crear una función reutilizable que se encarga de enviar cualquier tipo de comando hacia los dispositivos. En el caso de la configuración general, la función recibe el correspondiente dispositivo del diccionario y los comandos que se emplearán.

El primer paso que realiza la función es a través de la conexión previamente establecida con el dispositivo, y mediante la instancia de la conexión se llama a la función propia de NAPALM denominada *load_merge_candidate* la cual permite transferir una configuración candidata en formato de cadena de texto a un dispositivo de red y fusionarla con la configuración actual del dispositivo. La configuración candidata es una de carácter provisional que se carga en el dispositivo sin ser aplicada, lo que permite realizar pruebas y comprobar que los cambios que se van a realizar no producirán ningún tipo de incidencia en el correcto manejo del dispositivo.

Después de cargar la configuración deseada, se compara la configuración actual del dispositivo con la configuración deseada mediante la función *compare_config* y posteriormente se aplican los cambios necesarios para fusionar ambas configuraciones. Hay que tener en cuenta que la configuración candidata puede presentar comandos no válidos o imposibles de aplicar en el dispositivo, por lo que es conveniente validar la misma antes de aplicarla.

8.2.2. Configuración básica de routers

Como ocurre con el *switch* que hemos configurado, el *router* se puede configurar utilizando la misma sintaxis vista anteriormente. Por eso utilizamos el mismo código y funciones para configurar el *router* que nos interesa.

A su vez utilizamos el fichero para *routers* el cual tiene ciertas instrucciones específicas, lo cual hace que los diferencie al fichero de los *switches* en cuanto a los comandos ya que cada uno lee y procesa únicamente las instrucciones que son propias del dispositivo. Una vez más todas líneas son leídas por la función mostrada en la figura 8.15. y guardadas para su futura utilización.

En cuanto al código que se utiliza para enviar y confirmar los cambios de configuración es el mismo que se ha mostrado en la figura 8.17. ya que ese bloque de código forma parte del ciclo que realiza el programa por cada dispositivo en nuestro diccionario de dispositivos, por lo que en el caso del *router* realizaría las mismas instrucciones que se han visto en la configuración básica del *switch*.

8.2.3. Creación de VLANs en switches

Para resolver este caso práctico se requiere utilizar la misma lógica vista con anterioridad tanto para el caso de la librería Netmiko, como en el caso de las configuraciones hechas en los casos previos con NAPALM.

```
# Creación de VLANs en el switch
if device_name == SW1:
    config_file = 'vlans_conf'
    config = read_config(config_file)
    print("----- Creando VLANs -----\n")
    change_config(napalm_device, config)
```

Fig. 8.18. Código en Python para crear VLANs con NAPALM

Lo que realiza el bloque de código mostrado es verificar que realmente se trata de un dispositivo de tipo *switch* y procede a obtener los comandos a aplicar y posteriormente realizar enviar y aceptar la configuración, en el caso de que todo sea correcto.

Las funciones utilizadas ya se han explicado en las configuraciones hechas tanto para el *switch* como el *router*, por lo que a continuación se explicarán algunas cuestiones que surgen luego de obtener los resultados de este caso práctico:

Una vez realizadas varias pruebas de creación de VLANs utilizando NAPALM para la configuración de las VLAN en un *switch* dentro del entorno de virtualización GNS3, se ha encontrado que, en cada una de las pruebas, las VLAN no se han podido crear correctamente.

A pesar de que se realizaron varias pruebas para determinar la causa por la cual NAPALM no estaba configurando correctamente las VLAN en el *switch*, ninguna de ellas arrojó una respuesta clara. Se verificó que la versión del IOS del *switch* era compatible con la versión de NAPALM utilizada, se consultó la documentación de NAPALM y se hizo un análisis detallado de los mensajes de error generados durante la configuración. Estos mensajes confirmaban que a pesar de que NAPALM enviaba los comandos de configuración hacia el dispositivo final, por alguna razón no se aceptaban los cambios y estos no se reflejaban en la configuración del equipo. Sin embargo, todo parecía estar en orden y no se encontró ninguna razón evidente que explicara la causa del problema. Se sospechó que podría ser un problema con la configuración de GNS3, como la configuración de la red virtual o la interfaz de *loopback*, pero todas estas configuraciones parecían estar correctas. Se decidió seguir investigando y consultar en foros dedicados a este tópico para encontrar la solución al problema y poder configurar las VLAN correctamente en el *switch*, pero lastimosamente no se pudo obtener una respuesta que ofrezca solución a este problema.

8.2.4. Implementación de ACLs en routers

Utilizando la librería NAPALM se ha podido resolver de manera eficiente la automatización de la configuración de listas de control de acceso en el entorno de GNS3. En este caso práctico, también se extrae la configuración de comandos necesarios para la creación de una ACL desde un archivo de texto plano, el cual ya hemos visto previamente cuando se ha realizado la configuración con Netmiko y que se puede consultar en la figura 8.15.

Una vez que se tiene la configuración en un archivo de texto plano, se puede utilizar la librería NAPALM para automatizar el proceso de configuración de la ACL en un dispositivo de red. El siguiente código Python muestra cómo se puede hacer esto:

```
# Implementación de ACLs en el router
if device_name == R1:
    config_file = 'acls_conf'
    config = read_config(config_file)
    print("----- Implementando ACLs -----\n")
    change_config(napalm_device, config)
```

Fig. 8.19. Código en Python para implementar ACLs con NAPALM

Debido a que la aplicación de estas listas de control de acceso, en este caso, son exclusivas del *router*, se ha de diferenciar del resto de dispositivos para poder seguir con la ejecución del programa. Como segundo paso, se procesa el archivo de texto y luego con ayuda de la función *change_config*, que se ha creado previamente y que se utiliza para cada caso, cargamos la configuración al dispositivo y la confirmamos en caso de que no haya incidencias. Una vez que se han cargado los comandos en la memoria de candidatos, se debe aplicar la configuración en el dispositivo de red mediante el uso de la función *commit_config*. Este proceso aplica los cambios realizados en la memoria de candidatos a la configuración en ejecución.

8.3. Diseño de la interfaz

El diseño de la interfaz de nuestra aplicación web de monitorización de red se ha desarrollado de forma básica y sencilla con el objetivo de demostrar y comprobar su correcto funcionamiento. Para su creación, hemos utilizado la herramienta de diseño gráfico Canva, que nos ha permitido elaborar un *mockup* visualmente atractivo y funcional.

Fig. 8.20. Mockup de la web de monitorización

La interfaz principal de nuestra aplicación se basa en una tabla que muestra de manera organizada y estructurada la información de los equipos o *hosts* a monitorizar. Cada fila de la tabla representa un equipo y contiene datos importantes para su identificación y estado de disponibilidad.

Los campos de información que se incluyen en cada fila de la tabla son los siguientes:

Fig. 8.21. Campos de información en la tabla de monitorización

1. **Hostname:** El nombre del equipo o host bajo monitorización.
2. **IP:** La dirección IP asignada al equipo en la red.
3. **MAC:** La dirección física o MAC (Media Access Control) del equipo.
4. **Disponibilidad:** Indica si el equipo está actualmente disponible o no.

Además de los campos mencionados anteriormente, hemos agregado tres botones adicionales en cada fila de la tabla para proporcionar una experiencia interactiva al usuario. Estos botones permiten acceder a detalles específicos del equipo seleccionado. Los botones disponibles son los siguientes:

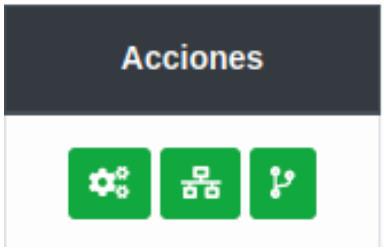

Fig. 8.22. Botones de acción informativos

1. **Configuración:** Al presionar este botón, se abrirá una vista detallada que mostrará la configuración actual del equipo seleccionado. Aquí se puede encontrar información relevante, como la configuración de red, los servicios habilitados y cualquier otro dato importante para la monitorización.

Fig. 8.23. Resultado del comando *show config*

2. **Tabla ARP:** Al hacer clic en este botón, se mostrará una vista específica que contendrá la tabla ARP (Address Resolution Protocol) del equipo seleccionado. Esta tabla proporciona información sobre las direcciones IP y las direcciones MAC asociadas en la red.

Fig. 8.24. Resultado del comando *show arp*

3. **Interfaces:** Al presionar este botón, se abrirá una vista que mostrará información detallada sobre las interfaces del equipo seleccionado. Esta sección brinda datos sobre las interfaces de red, como la velocidad, el estado y cualquier otra información relevante para el monitoreo.

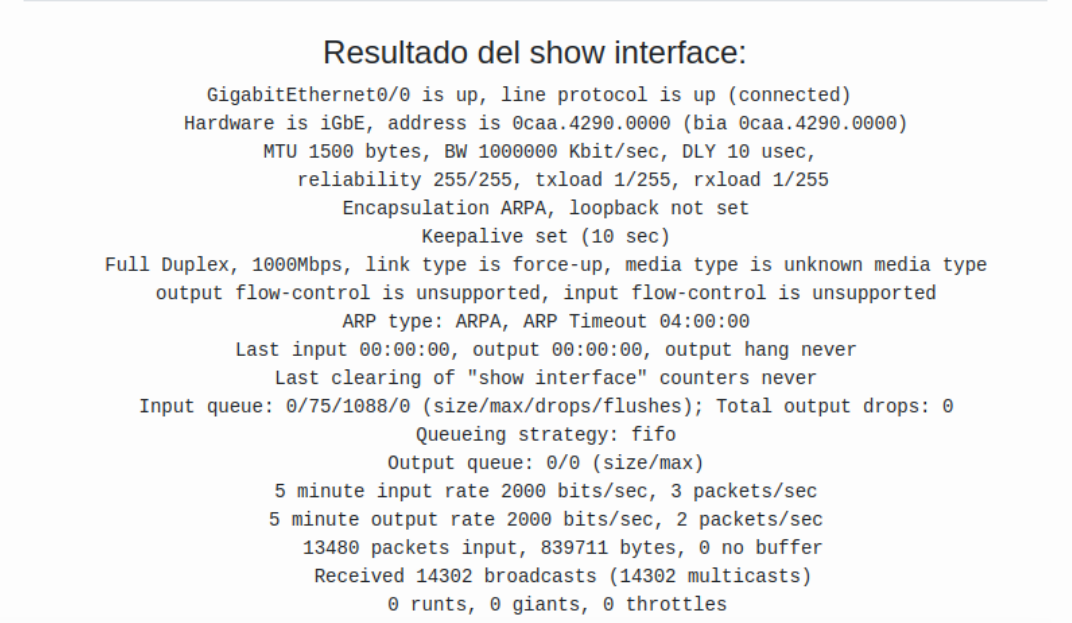

Fig. 8.25. Resultado del comando *show interface*

En resumen, nuestra interfaz diseñada en Canva presenta una tabla intuitiva que muestra los equipos o *hosts* a monitorizar, junto con su información básica de identificación y estado. Los botones de acceso a los detalles específicos brindan una funcionalidad adicional para obtener información más detallada sobre cada equipo. Este diseño básico y sencillo asegura que la aplicación sea fácil de usar y cumpla con los requisitos de monitorización de red.

8.4. API REST

La API REST implementada en nuestra aplicación de monitorización de red se ha desarrollado utilizando Flask en Python. Esta API sigue los principios de la arquitectura REST (Representational State Transfer) para ofrecer una interfaz de programación de aplicaciones eficiente y estandarizada que facilita la gestión de los hosts en el contexto de la monitorización de red.

La API REST se basa en los siguientes principios y características clave:

1. **Arquitectura Cliente-Servidor:** La API sigue el modelo cliente-servidor, donde el cliente realiza solicitudes HTTP a través de la red y el servidor procesa esas solicitudes y envía respuestas adecuadas. En el contexto de nuestra aplicación de monitorización de red, el cliente puede ser una aplicación o una interfaz de usuario que necesita interactuar con la API para realizar operaciones relacionadas con la gestión de *hosts*.
2. **Protocolo HTTP:** La API utiliza el protocolo HTTP como el medio de comunicación entre el cliente y el servidor. Cada operación CRUD (Crear, Leer, Actualizar y Eliminar) se mapea a un método HTTP específico:
    - **GET:** Utilizado para recuperar información de los *hosts* registrados. Al realizar una solicitud GET al punto final "/hosts", se devuelve una respuesta JSON que contiene todos los *hosts* existentes y su información relevante. Esto permite obtener una visión general de los *hosts* monitorizados.
    - **POST:** Permite agregar nuevos *hosts* a la colección. Al realizar una solicitud POST al punto final "/hosts" con los datos JSON del *host* a agregar, se registra en la colección de hosts. Esto facilita la adición de nuevos equipos a la monitorización de red.
    - **PUT:** Utilizado para actualizar información de un *host* existente. Al realizar una solicitud PUT al punto final "/hosts" con el parámetro "hostname" y los datos JSON actualizados del host, se actualiza la información del *host* correspondiente. Esto posibilita la modificación de los detalles de configuración o cualquier otro dato relevante de un *host* en particular.
    - **DELETE:** Permite eliminar un *host* específico de la colección. Al realizar una solicitud DELETE al punto final "/hosts" con el parámetro "hostname", se elimina el

host correspondiente. Esto permite retirar hosts de la monitorización de red cuando ya no son necesarios.

3. **Puntos finales (Endpoints):** La API define *endpoints* que representan recursos específicos dentro del contexto de la monitorización de red. En nuestro caso, el punto final principal es "/hosts", el cual permite gestionar la colección de *hosts* monitorizados. Se pueden agregar *endpoints* adicionales para ampliar la funcionalidad de la API en el futuro, como, por ejemplo, "/hosts/{hostname}/config" para obtener la configuración detallada de un *host* específico.

La API REST implementada en nuestra aplicación de monitorización de red ofrece una interfaz coherente y estructurada para interactuar con los *hosts*. Siguiendo los principios REST, permite realizar operaciones CRUD sobre los *hosts*, lo que facilita la gestión y el monitoreo de la red. Además, al utilizar el protocolo HTTP y devolver respuestas en formato JSON, la API es interoperable y se integra fácilmente con otras aplicaciones o herramientas de monitorización de red.

8.5. Implementación de la web

8.5.1. Estructura de carpetas

El proyecto de nuestra aplicación desarrollada en Flask sigue una estructura de carpetas y archivos organizada para facilitar la gestión y modularidad del código. Aunque la estructura puede variar según las necesidades específicas del proyecto, en este caso particular se ha utilizado una estructuración sencilla.

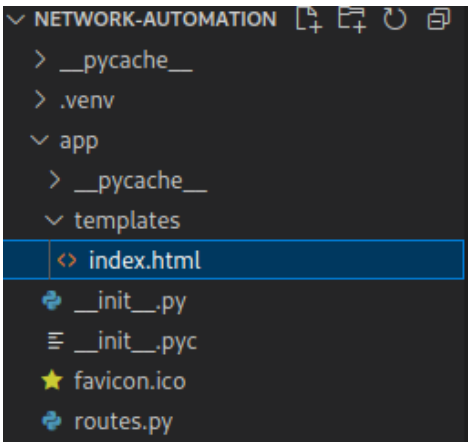

Fig. 8.26. Estructura del proyecto Flask

1. **Carpeta raíz:** El proyecto se encuentra en una única carpeta denominada "app". Esta carpeta contiene todos los archivos y carpetas relacionados con la aplicación web de monitorización de red.

2. **Archivo "init.py":** Este archivo, llamado "init.py", es un archivo especial en Flask que se utiliza para inicializar la aplicación. Contiene la configuración y la creación de la instancia de la aplicación Flask, así como cualquier otra configuración necesaria.

3. **Archivo de rutas REST:** El proyecto incluye otro archivo Python, generalmente denominado "routes.py" o similar, que se encarga de definir las rutas y las funciones asociadas a cada punto final de la API REST. En este archivo, se utiliza el decorador "@app.route" para mapear las URL de las rutas y definir las funciones que se ejecutarán al acceder a cada ruta específica.

4. **Carpeta "templates":** Dentro de la carpeta "app", encontramos una subcarpeta llamada "templates". Esta carpeta se utiliza para almacenar los archivos de plantillas HTML utilizados en la interfaz de usuario de la aplicación. En particular, en nuestro proyecto se encuentra el archivo "index.html", que representa la página principal de nuestra aplicación web.

En general, esta estructura sencilla del proyecto en Flask con una única carpeta "app" contiene el archivo de inicialización "init.py" que configura la aplicación Flask, el archivo de rutas REST y la carpeta "templates" que almacena los archivos de plantillas HTML. Esta estructura es adecuada para proyectos más pequeños y puede adaptarse y expandirse según las necesidades específicas del proyecto.

8.5.2. Estructura del back end

El *back end* de la aplicación web de monitorización de red está diseñado para proporcionar la lógica y las funcionalidades necesarias para interactuar con los datos de la red y servirlos a la interfaz de usuario. A continuación, se detallan las principales funciones y rutas presentes en el *back end*:

1. **Ruta principal – “/”**

La ruta principal, definida como "@app.route('/')", es la ruta inicial de la aplicación que se ejecuta al iniciar la aplicación en el *localhost*, en el puerto 5050. Esta ruta realiza las siguientes acciones:

- Ejecuta la función *network_host_discovery()* para descubrir los *hosts* en la red.
- Obtiene los hosts mediante la función *get_hosts()*, que recupera los *hosts* almacenados en una variable global.
- Guarda los hosts en la sesión utilizando *session['hosts'] = hosts* para que estén disponibles en otras partes de la aplicación y así utilizarlos más adelante.
- Renderiza el archivo de plantilla "index.html" utilizando la función *render_template('index.html', hosts=hosts)*, pasando los *hosts* como parámetro para su uso en la interfaz de usuario.

2. **Ruta – "/get_current_hosts" (Método GET)**

   La ruta "/get_current_hosts" se utiliza para obtener los *hosts* actuales de la sesión. Realiza lo siguiente:

   - Obtiene los *hosts* de la sesión mediante *hosts = session.get('hosts', {})*. Si no hay *hosts* en la sesión, se asigna un diccionario vacío como valor predeterminado.
   - Devuelve una respuesta JSON utilizando *jsonify({'hosts': hosts})*, que contiene los *hosts* obtenidos.

3. **Ruta – "/ping" (Método POST)**

   La ruta "/ping" se utiliza para realizar un *ping* a un *host* específico. Realiza lo siguiente:

   - Obtiene el *host* del cuerpo de la solicitud JSON mediante *host = request.json['host']*.
   - Realiza un ping al host utilizando la función *ping_host(host)*, que implementa la lógica para enviar un *ping* al *host* y devuelve el resultado.
   - Devuelve el resultado del *ping* utilizando *return ping_result*.

4. **Ruta – "/update" (Método POST)**

   La ruta "/update" se utiliza para actualizar un *host* específico. Realiza lo siguiente:

- Obtiene el *host* del cuerpo de la solicitud JSON mediante *host = request.json['host']*.
- Realiza la actualización del *host* utilizando la función *update_host(host)*, que implementa la lógica para actualizar el *host* según los requisitos del sistema.
- Devuelve el resultado de la actualización utilizando *return update_result*.

Además de estas rutas principales, existen tres rutas adicionales para obtener información específica de los dispositivos:

5. **Ruta – "/show_config" (Método POST)**

   Esta ruta se utiliza para obtener la configuración de un *host* específico. Realiza lo siguiente:

   - Obtiene el nombre del *host* desde el formulario mediante *hostname = request.form.get('hostname')*.
   - Realiza la conexión al *host* utilizando la librería Netmiko y ejecuta el comando "show config" para obtener la configuración.
   - Guarda el resultado en la variable *result*.
   - Obtiene los *hosts* de la sesión mediante *hosts = session.get('hosts', {})*.
   - Renderiza el archivo de plantilla "index.html" utilizando *render_template('index.html', hosts=hosts, show_config_result=result),* pasando los *hosts* y el resultado de la configuración como parámetros.

6. **Ruta – "show_arp" (Método POST)**

   Esta ruta se utiliza para obtener la tabla ARP de un *host* específico. Realiza lo siguiente:

   - Obtiene el nombre del *host* desde el formulario mediante *hostname = request.form.get('hostname')*.
   - Realiza la conexión al *host* utilizando la librería Netmiko y ejecuta el comando "show arp" para obtener la tabla ARP.
   - Guarda el resultado en la variable *result*.
   - Obtiene los *hosts* de la sesión mediante *hosts = session.get('hosts', {})*.

- Renderiza el archivo de plantilla "index.html" utilizando *render_template('index.html', hosts=hosts, show_arp_result=result),* pasando los *hosts* y el resultado de la tabla ARP como parámetros.

7. **Ruta – "show_interface" (Método POST)**

   Esta ruta se utiliza para obtener la información de las interfaces de un *host* específico. Realiza lo siguiente:

   - Obtiene el nombre del *host* desde el formulario mediante *hostname = request.form.get('hostname')*.
   - Realiza la conexión al *host* utilizando la librería Netmiko y ejecuta el comando "show interface" para obtener la información de las interfaces.
   - Guarda el resultado en la variable *result*.
   - Obtiene los *hosts* de la sesión mediante *hosts = session.get('hosts', {})*.
   - Renderiza el archivo de plantilla "index.html" utilizando *render_template('index.html', hosts=hosts, show_interface_result=result)*, pasando los *hosts* y el resultado de la información de las interfaces como parámetros.

Estas rutas y funciones permiten interactuar con los *hosts* de la red, realizar acciones como ejecutar el comando *ping* y actualizar los mismos, y además de obtener información específica de los dispositivos para mostrarla en la interfaz de usuario.

8.5.3. Integración con la API REST para obtener datos en tiempo real

En esta sección, se explicará la integración del *back end* de la aplicación con una API REST para obtener datos en tiempo real. Se utilizarán funciones externas que contienen la lógica necesaria para interactuar con la API. A continuación, se detallan las funciones utilizadas:

1. **Función – "get_hosts()"**

   Esta función se encarga de obtener los *hosts* desde la API REST. Realiza lo siguiente:

   - Define la URL de la API REST, en este caso, "http://127.0.0.1:5000/hosts".
   - Realiza una solicitud GET a la URL utilizando *requests.get(url).*
   - Verifica si la solicitud fue exitosa mediante *response.raise_for_status()*. Si la solicitud no fue exitosa (código >= 400), se lanza una excepción.

- Obtiene los *hosts* desde la respuesta JSON utilizando response.json().
- Imprime un mensaje de éxito y devuelve los *hosts* obtenidos. En caso de error, imprime un mensaje de error y devuelve un diccionario vacío.

2. **Función – "network_host_discovery()"**

   Esta función se encarga de descubrir *hosts* en la red utilizando *arping*. Realiza lo siguiente:

   - Define la red a escanear en la variable *network*, por ejemplo, "192.168.122.0/24".
   - Utiliza la librería Scapy para enviar una solicitud *arping* a la red y obtener las respuestas.
   - Imprime un resumen de las respuestas obtenidas.
   - Por cada respuesta, extrae la dirección IP, la dirección MAC y el nombre de *host* (obtenido mediante *socket.gethostbyaddr*) si está disponible.
   - Crea un diccionario de host con la información obtenida y llama a la función *update_host(host)* para actualizar el *host*.

3. **Función – "update_host(host)"**

   Esta función se encarga de actualizar un *host* utilizando la API REST. Realiza lo siguiente:

   - Obtiene el nombre de *host* desde el diccionario *host*.
   - Define la URL de la API REST, en este caso, "http://127.0.0.1:5000/hosts".
   - Define los parámetros de la solicitud, que incluyen el nombre de *host* como parte de los parámetros y el diccionario host como datos JSON.
   - Realiza una solicitud PUT a la URL utilizando *requests.put(url, params=params, json=host)*.
   - Verifica si la respuesta tiene un código de estado diferente a 200. Si es así, muestra un mensaje de error y devuelve el mensaje.
   - Si la respuesta es exitosa, muestra un mensaje de éxito y devuelve el mensaje.

4. **Función – "ping_host(host)"**

   Esta función se encarga de realizar un *ping* a un *host* específico. Realiza lo siguiente:

   - Obtiene el nombre de *host* desde el diccionario *host*.

- Ejecuta el comando *ping* utilizando la librería *subprocess.run()* con los parámetros adecuados.
- Verifica el código de retorno del comando *ping* para determinar si el *ping* fue exitoso o no.
- Actualiza el estado de disponibilidad del *host* en el diccionario *host* en función del resultado del *ping*.
- Imprime un mensaje indicando si el *ping* se realizó correctamente o no.
- Devuelve un estado de verdadero o falso para poder utilizarlo más adelante.

8.5.4. Características y funcionalidades de la interfaz de usuario (front end)

Este apartado se centra en el desarrollo de la interfaz de usuario (*front end*) de la aplicación de monitorización de *hosts*. El objetivo principal de esta interfaz es presentar visualmente los datos obtenidos del *back end* y permitir la interacción del usuario con la aplicación.

La interfaz de usuario es una parte fundamental de cualquier aplicación, ya que es el medio a través del cual los usuarios interactúan con el sistema. En el caso de una aplicación de monitorización de *hosts*, la interfaz de usuario desempeña un papel crucial al presentar los datos de los *hosts* y permitir al usuario realizar acciones relacionadas con la monitorización.

La interfaz de usuario se desarrolla utilizando HTML, CSS y JavaScript. Además, se utiliza el lenguaje de plantillas Jinja para generar dinámicamente el contenido de la página en función de los datos obtenidos del *back end*. A continuación, se detallan las principales funcionalidades y características de la interfaz de usuario, junto con los fragmentos de código relevantes para su implementación.

- **Estructura de la página**

```
<div class="container">
    <h1>Monitorización de Hosts</h1>
```

Fig. 8.27. Contenedor principal de la web

El código HTML define la estructura principal de la página. El contenido se encuentra dentro de un contenedor con la clase "container", que está centrado y rodeado por un borde. Esto se logra utilizando el siguiente fragmento de código:

- **Estilos personalizados**

```
<style>
    /* Estilos personalizados */
    body {
        padding: 20px;
        background-color: #f2f2f2;
        font-family: Arial, sans-serif;
        background-repeat: no-repeat;
        background-size: cover;
        background-position: center;
    }

    .container {
        max-width: 800px;
        margin: 0 auto;
        text-align: center;
        /* Centra el contenido dentro del contenedor */
        background-color: rgba(255, 255, 255, 0.8);
        padding: 20px;
        border-radius: 8px;
        box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    }
```

Fig. 8.28. Estilos personalizados

Se utilizan estilos CSS para personalizar la apariencia de la interfaz. Esto incluye el ajuste del tamaño de la fuente, los colores de fondo, los bordes y las sombras. A continuación, se muestra un ejemplo de los estilos utilizados:

Estos estilos definen propiedades como el color de fondo del cuerpo de la web, su posición, tamaño, etc., como también el ancho máximo del contenedor, los márgenes, el color de fondo, el espaciado interno, los bordes redondeados y la sombra.

- **Tabla de hosts**

La tabla de *hosts* muestra los detalles de cada *host* obtenidos del *back end*. Los datos se generan dinámicamente utilizando el lenguaje de plantillas Jinja. A continuación, se muestra un ejemplo de cómo se genera una fila de la tabla para cada *host*:

```
<tbody>
    {% for host in hosts.values() %}
    <tr>
        <td>{{ host.hostname }}</td>
        <td>{{ host.ip }}</td>
        <td>{{ host.mac }}</td>
        <td>
            {% if host.availability %}
            <i class="fas fa-check-circle status-icon status-true"></i> Disponible
            {% else %}
            <i class="fas fa-times-circle status-icon status-false"></i> No disponible
            {% endif %}
        </td>
        <td>···
        </td>
    </tr>
    {% endfor %}
</tbody>
```

Fig. 8.29. Tratamiento de hosts utilizando Jinja

En este fragmento de código, se utiliza un bucle "{% for host in hosts.values() %}" para iterar sobre la lista de *hosts* y generar las filas correspondientes. Cada fila de la tabla contiene celdas "<td>" que muestran los valores de los atributos de cada *host*, como "Hostname", "IP" y "MAC". Además, se utiliza una estructura condicional "{% if host.availability %}" para mostrar un icono y un mensaje de "Disponible" o "No disponible" según el estado del *host*.

- **Botones de acción**

  La interfaz de usuario incluye botones de acción para que el usuario realice acciones relacionadas con la monitorización de *hosts*. Cada botón está asociado a una función JavaScript que se ejecuta al hacer clic en él. A continuación, se muestra un ejemplo de cómo se implementa un botón de acción para mostrar la configuración de un *host*:

```
<form action="{{ url_for('show_config') }}" method="POST">
    <input type="hidden" name="hostname" value="{{ host.hostname }}">
    <button type="submit" class="btn btn-primary btn-sm btn-action" data-toggle="tooltip"
        data-placement="top" title="Show Config">
        <i class="fas fa-cogs"></i>
    </button>
</form>
```

Fig. 8.30. Botón de acción para mostrar configuración

En este caso, el botón se encuentra dentro de un formulario que utiliza el método POST para enviar los datos al *back end*. Se incluye un campo oculto "<input type="hidden">" para enviar el nombre de *host* correspondiente al botón. Al hacer

clic en el botón, se enviará una solicitud al *back end* utilizando la ruta especificada en el atributo *action* del formulario.

- **Funciones JavaScript**

  El código JavaScript se utiliza para implementar funcionalidades interactivas en la interfaz de usuario. A continuación, se describen las funciones JavaScript relevantes:

  Función "$(document).ready()"

  Esta función se ejecuta cuando el documento HTML ha sido completamente cargado y está listo para interactuar con JavaScript. En este caso, se utiliza para asignar un controlador de eventos al botón de "Refrescar" una vez que la página se ha cargado:

```
$(document).ready(function () {
    // Al hacer clic en el botón de refrescar
    $("#btn-refresh").click(function () {
        pingHosts();
    });
});
```

Fig. 8.31. Función javascript principal

Al hacer clic en el botón de "Refrescar", se ejecuta la función *pingHosts()*, que realiza una solicitud AJAX al *back end* para obtener los datos actualizados de los hosts.

Función "pingHosts()"

Esta función realiza una solicitud AJAX al *back end* para obtener los datos actualizados de los *hosts*. Utiliza el método GET para obtener los *hosts* y luego itera sobre cada uno de ellos para enviar una solicitud AJAX individual para realizar un ping al *host*:

```javascript
function pingHosts() {
    $.ajax({
        type: "GET",
        url: "/get_current_hosts",
        success: function (response) {
            // Convertimos el objeto en un array iterable de hosts
            const hosts = Object.values(response.hosts);

            // Iteramos sobre los hosts
            hosts.forEach(function (host) {
                $.ajax({
                    type: "POST",
                    url: "/ping",
                    data: JSON.stringify({ 'host': host }),
                    contentType: "application/json",
                    success: function (pingResponse) {
                        console.log(JSON.parse(pingResponse));
                        updateHost(host, pingResponse);
                    },
                    error: function (error) {
                        console.error(error);
                        updateHost({ ...host, availability: false });
                    }
                });
            });
        },
        error: function (error) {
            console.error(error);
        }
    });
}
```

Fig. 8.32. Función javascript que interactúa con el *back end*

La respuesta obtenida del *back end* se utiliza para actualizar la tabla de hosts en la interfaz de usuario mediante la función *updateHost(),* la cual permite enviar los datos actualizados de cada host a la API y devolver el resultado para poder así mostrarlos en la web. Además, sirve también para poder añadir nuevos dispositivos que se vayan activando con el paso del tiempo, es decir, si surge el caso de que un dispositivo se enciende, pues al refrescar la web, podemos visualizar este nuevo equipo en la tabla de *hosts*.

```javascript
function updateHost(host, pingResponse) {
    host.availability = pingResponse;
    $.ajax({
        type: "POST",
        url: "/update",
        data: JSON.stringify({ 'host': host }),
        contentType: "application/json",
        success: function (response) {
            refreshPage();
        },
        error: function (error) {
            console.error(error);
            refreshPage();
        }
    });
}
```

Fig. 8.33. Función javascript que actualiza los hosts mediante AJAX

Estos son solo algunos ejemplos de las funcionalidades y características implementadas en la interfaz de usuario. A lo largo del desarrollo, se utilizan otras funciones y fragmentos de código JavaScript para realizar diferentes acciones, como mostrar la configuración de un *host*, mostrar la tabla de direcciones ARP o mostrar las interfaces de red del *host* seleccionado. Todos estos elementos están coherentemente vinculados con el código del *back end*, asegurando así el correcto funcionamiento de la aplicación de monitorización de *hosts*.

En conclusión, la interfaz de usuario desarrollada para la aplicación de monitorización de hosts utiliza HTML, CSS y JavaScript, junto con el lenguaje de plantillas Jinja para integrarse con el *back end*. Los fragmentos de código proporcionados en este informe ilustran cómo se implementan las funcionalidades clave de la interfaz, incluyendo la tabla de *hosts* y los botones de acción. Estos elementos interactúan coherentemente con el código del *back end*, permitiendo a los usuarios visualizar y gestionar la monitorización de *hosts* de manera efectiva.

9. Conclusiones

En el desarrollo de este Trabajo Final de Grado titulado "Plataforma de monitorización de red", se lograron alcanzar exitosamente los objetivos planteados, obteniendo resultados significativos y demostrando la eficacia de la automatización y monitorización en la configuración de redes.

En primer lugar, se logró comprender a fondo los aspectos fundamentales de la automatización de red y su aplicación en una solución funcional. Mediante el uso de las librerías Netmiko y NAPALM, se automatizó el proceso de configuración de los equipos de red, lo que permitió agilizar las tareas y aumentar la eficiencia en su configuración.

Sin embargo, durante el desarrollo del proyecto, se enfrentaron desafíos relacionados con la librería NAPALM. Debido a su alta demanda de recursos, se optó por ajustar el enfoque original y trabajar con tan solo dos dispositivos de red en lugar de cinco, con el fin de garantizar un rendimiento óptimo. Además, se observó que NAPALM presentaba dificultades al configurar las VLANs, aunque se lograron realizar correctamente las demás configuraciones.

Por otro lado, se implementó una plataforma web sencilla utilizando el Framework Flask. Esta plataforma web permitió visualizar de manera clara y concisa la configuración actual de los dispositivos de red, así como la tabla ARP y las interfaces disponibles. Además, se incluyó una función de actualización en tiempo real para verificar la disponibilidad de los dispositivos mediante un *ping* y actualizar su estado.

En conclusión, este proyecto ha demostrado la importancia y los beneficios de la automatización de la configuración de redes, así como la monitorización en tiempo real de los equipos. A pesar de los desafíos encontrados con la librería NAPALM, se lograron cumplir los objetivos establecidos, sentando las bases para futuros desarrollos y mejoras en este campo. La implementación de la plataforma web proporcionó una herramienta efectiva para facilitar la monitorización y gestión de la configuración de los equipos de red.

10. Bibliografía

[1] Automatización de redes: Un caso práctico [en línea] [consulta: 15 de enero de 2023]. Disponible en https://ruc.udc.es/dspace/bitstream/handle/2183/31657/VazquezMira_Victor_TFG_2022.pdf?sequence=3

[2] Network Configuration with Netmiko What is Netmiko? [en línea] [consulta: 19 de enero de 2023]. Disponible en https://www.packetcoders.io/netmiko-the-what-and-the-why/

[3] NAPALM Network Automation Python: Introducción e Instalación [en línea] [consulta: 20 de enero de 2023]. Disponible en https://codingnetworks.blog/es/napalm-network-automation-python-introduccion-e-instalacion/

[4] What is Python? Executive Summary [en línea] [consulta: 21 de enero de 2023]. Disponible en https://www.python.org/doc/essays/blurb/

[5] What is Flask Python [en línea] [consulta: 21 de enero de 2023]. Disponible en https://pythonbasics.org/what-is-flask-python/

[6] Getting Started with GNS3 [en línea] [consulta: 21 de enero de 2023]. Disponible en

https://docs.gns3.com/docs/