Título del trabajo:	Automatización de redes informáticas con Python
Nombre del autor:	Adrián Manuel Conde Gil
Nombre del consultor:	Manuel Jesús Mendoza Flores
Fecha de entrega:	03/01/2020
Área del Trabajo Final:	Administración de redes y sistemas operativos
Titulación	Grado de Ingeniería de Tecnologías y Servicios de Telecomunicación
Idioma del trabajo:	Español
Palabras clave:	Automatización, Redes, Networking, Python, YANG, NETCONF, RESTCONF, GNS3

Resumen del Trabajo

Debido a la velocidad a la que la industria informática avanza, la tecnología que en un momento puede llegarse a considerar como innovadora, se puede convertir en obsoleta al poco tiempo. En cierto modo, se podría decir que, hasta hace pocos años, esto no era tan aplicable a la administración de redes informáticas, donde los conocimientos de diferentes protocolos de red, junto al manejo de la interfaz de línea de comandos (CLI) han sido más que suficientes.

Sin embargo, debido al avance de la tecnología y a una cada vez mayor dependencia a Internet, donde incluso la interrupción más mínima del servicio puede provocar grandes pérdidas a la organización, los roles del administrador de redes se tornan obsoletos a la par que la provisión manual del servicio, así como la resolución de incidentes, se vuelven inviables cuando la velocidad es un factor crucial. Por lo que es de vital importancia que los profesionales adapten esta nueva metodología para adaptarse a las exigencias de la industria.

Este cambio de paradigma no viene sin dificultades, debido a la irrupción de tantas herramientas nuevas, al final de las cuales solamente un conjunto de ellas se adopta en la industria. Debido a ello, este proyecto pretende realizar un análisis del contexto actual y sus precedentes, además de realizar una prueba de concepto a través de Python, el cual es el lenguaje de programación más utilizado en la actualidad para automatizar redes informáticas.

Abstract

Due to the speed at which computer industry progresses, technology which, at a due moment may be considered as state-of-the-art, can as well be determined obsolete in a short span of time. This was, in some sense, not that appliable to computer network administration, where network protocol and command line interface (CLI) knowledge has been considered more than sufficient, until now.

However, due to technological progress and an ever-increasing reliance on the Internet, where even the slightest service interruption can cause heavy economical losses to the organization, former network administrator roles are becoming obsolete as manual service provision and problem resolution are not feasible when speed is of vital importance. Due to this, it is crucial that network professionals adopt this new methodology to comply to industry needs.

This paradigm shift does not come without its own difficulties, due to the emergence of many new tools, which, in the end, only a few of them are adopted by the industry. Therefore, this project aims to research the current context and its precedents, as well as to provide proof of concept through Python, which is currently the most widely industry supported programming language for network automation.

ÍNDICE

Índice de figuras

Índice de tablas

1 INTRODUCCIÓN

1.1 RESUMEN DEL CONTEXTO ACTUAL

La industria de las redes informáticas, al igual que los demás sectores de la informática, ha avanzado a un ritmo vertiginoso, posibilitando el diseño de topologías capaces de transferir tasas de datos que hasta hace pocos años eran impensables. Sin embargo, y de forma muy peculiar, la forma en la que se gestionan estas redes apenas ha cambiado, donde el uso de la interfaz de línea de comandos (CLI, por sus siglas en inglés) sigue siendo la forma principal de administrar los distintos dispositivos que forman una red informática. Esta metodología se mantiene hasta la actualidad, pero el cada vez mayor número de dispositivos que componen una red, sumado a la necesidad de reaccionar a las distintas eventualidades de forma casi inmediata, han provocado que la industria esté buscando otras alternativas que puedan adaptarse a las circunstancias actuales. Es por ello por lo que, de forma reciente, la automatización está adquiriendo cada vez mayor impacto en esta industria, y, aunque el manejo del CLI sigue siendo una habilidad requerida, se está tornando en insuficiente por sí sola, y donde, además, los principales fabricantes realizan un esfuerzo activo para integrar estas habilidades incluso en sus certificaciones más básicas.

Es por todo ello que han aparecido multitud de soluciones diferentes, como programas de automatización (Ansible, Salt), SDN (OpenDayLight) y librerías para multitud de lenguajes de programación diferentes, de los cuales, el más destacable es Python, el cual ha sido adoptado por casi la totalidad de la industria debido a su facilidad de uso y flexibilidad a la hora de diseñar “scripts”. Por tanto, queda claro que no existe una sola manera de proveer de automatización a una red, pero lo que si es evidente es que este cambio de paradigma se está arraigando cada vez más en la industria, y, por lo tanto, no puede ser ignorado. Es por esta razón que en este proyecto se pretende explorar el impacto que toda esta serie de cambios está teniendo en la industria, y cómo hasta una implementación básica de automatización en una red puede marcar una gran diferencia en comparación con el uso exclusivo del CLI.

1.2 OBJETIVOS DEL PROYECTO

Los objetivos que se pretenden alcanzar con este proyecto son:

- Proporcionar trasfondo sobre las anteriores propuestas para dotar de programabilidad y automatización a las redes informáticas con el fin de comprender mejor el contexto actual.

- Ofrecer una introducción al lenguaje YANG, así como a los dos protocolos estandarizados en la actualidad que hacen uso de este: NETCONF y RESTCONF; además de las razones por las que se están convirtiendo en una alternativa eficaz al uso del CLI.

- Proporcionar una introducción a las librerías de uso más común en Python para automatizar topologías de red.

- Ofrecer un punto de partida a aquellas personas interesadas en aprender a automatizar redes informáticas a través de herramientas bien establecidas en la industria.

- Demostrar las ventajas que ofrece la automatización a través de su aplicación en una topología virtualizada de red.

- Investigar la posibilidad de usar YANG para realizar configuraciones independientes del fabricante del dispositivo en cuestión.

1.3 Evaluación de riesgos

Los riesgos identificados son los siguientes, donde la probabilidad de que ocurra uno de ellos durante el desarrollo de este proyecto puede ser: baja, media o alta.

- **Riesgo 1.** Posibilidad de que cualquiera de las imágenes utilizadas en la emulación de la topología disponga de errores que no permitan utilizar alguna de las funcionalidades requeridas, incluidos fallos de conectividad aun cuando la configuración sea correcta.
  Probabilidad del riesgo: media.

- **Riesgo 2.** Incompatibilidad de alguna de las librerías usadas para automatización con algún dispositivo en concreto.
  Probabilidad del riesgo: baja.

- **Riesgo 3.** Insuficiencia de recursos físicos para mantener activa la totalidad de la simulación de red.
  Probabilidad del riesgo: media.

- **Riesgo 4.** Posibilidad de que alguna de las imágenes utilizadas no sea compatible con métodos de configuración estandarizados (independientes del vendedor).
  Probabilidad del riesgo: alta.

1.4 Plan de contingencia para los riesgos identificados

- **Riesgo 1.** Se utilizará una imagen con una versión distinta del mismo dispositivo cuando sea posible, en caso contrario, se utilizará una imagen de un dispositivo análogo de un fabricante diferente.

- **Riesgo 2.** Se hará uso de una imagen de un dispositivo similar que sea compatible con las librerías utilizadas.

- **Riesgo 3.** Se utilizarán imágenes de dispositivos similares que hagan menor uso de recursos del sistema; en caso de que no fuera suficiente, se simplificaría la topología de red.

- **Riesgo 4.** Se procederá a hacer uso de configuraciones específicamente diseñadas para el dispositivo en concreto.

1.5 Alcance del proyecto

La parte teórica del proyecto se limitará a ofrecer una introducción teórica al lenguaje YANG que sea suficiente para poder comprender la mayoría de los modelos publicados, sin entrar en aquellos detalles del lenguaje cuyo uso es más anecdótico. Esto es aplicable a los protocolos basados en YANG, cuya exposición de todos sus detalles dificultaría la lectura y el objetivo de proporcionar un punto de partida a la automatización de redes informáticas.

La parte práctica del proyecto se limitará a configurar de forma automatizada una topología de red de baja complejidad a través de diferentes métodos; esta topología estará compuesta exclusivamente por conmutadores y enrutadores, además de dispositivos finales para poder evaluar la conectividad de la red una vez automatizada su configuración; se excluyen por tanto el uso de dispositivos más complejos como pueden ser cortafuegos o balanceadores de carga.

1.6 PLANIFICACIÓN DEL PROYECTO

Número de sección	Nombre de la sección	Descripción del apartado	PEC estimada de entrega	Fecha estimada de realización
1	Introducción	Proporciona la introducción del proyecto, así como el contexto actual.	PEC 1	3-10-2020
2	Evolución de la metodología de administración de redes	Expande la descripción del contexto actual y proporciona una presentación de los precedentes.	PEC 2	5-11-2020
3	Introducción al lenguaje Yang y sus protocolos	Proporciona una introducción al lenguaje YANG, así como a los protocolos NETCONF y RESTCONF.	PEC 2	5-11-2020
4	Automatización de una topología de red pequeña a través de Python	Parte principal del proyecto, donde se automatiza la red a través de diferentes librerías para Python.	PEC 3	9-12-2020
5	Presentación de conclusiones y líneas futuras	Sección donde se presentan las conclusiones obtenidas tras la realización del proyecto y líneas futuras del tema del proyecto.	Entrega Final	3-01-2021
I	Referencias	Disposición de todas las fuentes de información utilizadas en la memoria.	Entrega Final	3-01-2021

II	Recursos adicionales	Sección donde se enumeran los recursos utilizados para facilitar el desarrollo del proyecto.	Entrega Final	3-01-2021
III	Glosario	Sección donde se definen todas aquellas palabras utilizadas en el proyecto cuyo significado a simple vista pueda resultar incierto.	Entrega Final	3-01-2021
N/A	Desarrollo de la presentación en PowerPoint	Archivo de presentación del proyecto en diapositivas.	Entrega Final	3-01-2021
N/A	Desarrollo del vídeo de presentación	Presentación en formato audiovisual del proyecto.	Entrega Final	3-01-2021

Tabla 1. Planificación del Proyecto

1.7 LISTA DE ENTREGABLES

Nombre del archivo	Tipo de archivo	Descripción
Memoria_TFG.pdf	Documento PDF	Contiene la memoria del proyecto.
Presentacion_PPT_TFG.pptx	Documento PowerPoint	Contiene la presentación en diapositivas del proyecto.
Presentacion_MP4_TFG.mp4	Video en formato MP4	Contiene la presentación audiovisual del proyecto.
Codigo_TFG.zip	Comprimido en formato ZIP	Contiene el código de todas las soluciones presentadas en el proyecto; cada solución está organizada en una carpeta propia.

Tabla 2. Lista de Entregables

2 EVOLUCIÓN DE LA METODOLOGÍA DE LA ADMINISTRACIÓN DE REDES

2.1 EXPANSIÓN DEL CONTEXTO ACTUAL

Aunque este proyecto tiene como uno de sus objetivos principales el poder demostrar de forma práctica la utilidad que proporciona la capacidad de automatización en redes informáticas, es conveniente entender previamente la razón por la cual esta nueva realidad se está imponiendo en todos los trabajos orientados a la planificación, configuración y monitorización de redes informáticas, para así poder contextualizar los apartados posteriores adecuadamente.

Como comentan algunos medios, la posición del ingeniero de redes está en pleno cambio, algunos medios especializados incluso admiten que es un término que está quedándose obsoleto, incluso llegando a admitir que "está muriendo" **[1].** Obviamente, no se puede negar que están habiendo cambios convulsos que ponen, cuanto menos en duda, la perdurabilidad de esta posición en la industria, pero en realidad los profesionales de las redes informáticas seguirán siendo necesarios en la organización **[2]**, eso sí, el abanico de conocimientos y habilidades de los que han de disponer está cambiando considerablemente, quedando obsoleta la metodología de configuración y mantenimiento manual de redes a través del CLI de cada dispositivo. Aunque este modelo de trabajo sigue vigente hoy en día, la industria ha comenzado a impulsar el cambio de mentalidad y a entrenar a futuros y actuales profesionales en un nuevo paradigma donde la automatización de todo tipo de procesos están a la orden del día, por lo que es cierto que la posición del ingeniero de redes tal y como se conoce desaparecerá a medio/largo plazo, pero no así la necesidad de profesionales con conocimientos en protocolos de red y que además sean capaces de crear sus propias soluciones de automatización tanto en configuración como resolución de problemas; a este tipo de nuevo rol no se le ha dado un nombre en específico, pero sí es cierto que una de las acepciones más populares es la de "network developer", que, por definición, se considera a aquel profesional que desarrolla soluciones para la red informática, es decir, en cierta manera, tratar a la red como un conjunto programable y no como un amalgama de dispositivos que manejar de forma individual **[3]**.

Multinacionales como Cisco han comenzado a impulsar certificaciones orientadas a este nuevo paradigma, conocidas como DevNet **[4]**, aunque no es la única, ya que, otras multinacionales, como Juniper, por ejemplo, también ofrecen sus propias certificaciones orientadas a la automatización **[5].** Por tanto, en una primera toma de vista, es cierto que la industria sigue ofreciendo sus certificaciones con cierto nivel conservadurismo, en el sentido de que el uso del CLI es fundamental, pero sí dando por entendido de que se está produciendo un profundo cambio en este sector de la industria.

Dicho lo cual, en cierto modo se podría decir que los ingenieros de redes se están viendo forzados a adoptar la llamada cultura "DevOps", donde han de ser responsables tanto del desarrollo como del correcto funcionamiento y ejecución de sus propios proyectos, y es esta una de las razones principales por las que el abanico de habilidades a desarrollar es mucho mayor que el de hace unos años, como las comentadas anteriormente, pero sin excluir a otras, como puede ser el manejo de sistemas Linux, debido a que la mayoría de los servidores son ejecutados en este tipo de sistemas, además de la mayoría de los propios dispositivos de red. Por todo ello, se podría considerar que la posición de ingeniero de redes no ha dejado de existir, y sigue siendo común en la industria, pero sí que está dejando de ser necesario tal y como se ha concebido hasta ahora, debido al crecimiento exponencial de las redes, donde la configuración manual es simplemente inviable y propicia a errores humanos.

2.2 PRECEDENTES

Antes de introducir los protocolos de configuración más recientes para redes informáticas, es conveniente presentar de forma breve los antecedentes que han dado lugar a la situación actual. En primer lugar, uno de los protocolos más conocidos para la administración de redes, ya que permite la conectividad remota a través de líneas de comandos es Telnet, cuyo uso a nivel profesional es prácticamente anecdótico debido a que no proporciona ningún tipo de encriptado en la transmisión de los datos. Debido a esta inseguridad y a la necesidad de proveer encriptado y autenticación en la transmisión de los datos, a mediados de los años 90 se desarrolló un segundo protocolo llamado SSH; esta versión se conoce hoy en día como SSHv1, y es considerada insegura, mientras que una segunda versión, conocida como SSHv2, fue desarrollada por la IETF y publicada como estándar en 2006, la cual es utilizada como protocolo para la administración remota de todo tipos de dispositivos de forma segura. Curiosamente, SSHv2 no se utiliza solamente por sí solo como protocolo de administración remota, sino que otros protocolos de aparición más reciente lo utilizan como medio de transporte para la comunicación remota, entre ellos, uno de los protocolos de administración de redes de mayor importancia en la actualidad, NETCONF.

Si bien Telnet y SSH son protocolos cuya finalidad es el establecimiento de sesión remota, y, sobre todo, este último, en su segunda versión, es usado constantemente para todo tipo de operaciones, no proporcionan de por sí ninguna funcionalidad que permita manejar los dispositivos de forma programática, lo cual es lógico, ya que no era ese el objetivo con el que fueron concebidos. Aun así, antes de continuar, es conveniente explicar que se podría considerar como la programabilidad en un dispositivo; en ese aspecto no hay una definición específica, aunque la que Cisco ofrece puede ayudar a la comprensión "*La programabilidad de red se considera actualmente como el conjunto de herramientas y mejores prácticas para desplegar, administrar y solucionar problemas en dispositivos de red*" **[6]**. A partir de esta definición, no es difícil adivinar que el uso del CLI de forma manual como interfaz de administración no entra dentro de "mejores prácticas", debido a lo repetitivo que resulta aplicar la misma configuración una y otra vez, por no considerar el importante riesgo que supone el factor humano debido a la posibilidad de implementar configuraciones erróneas. Dicho lo cual, como se verá posteriormente, si es posible automatizar el CLI, y, aunque no sea la situación ideal, en ocasiones no habrá alternativa debido a que muchos dispositivos existentes no soportan protocolos de administración basados en el lenguaje YANG.

Por supuesto, el lenguaje de modelado YANG no es el primer intento de desarrollar una forma estandarizada de describir un dispositivo de red, para ello hay que adentrarse en el protocolo SNMP, que usa su propia estructuración de la información a través de SMIv2; este protocolo está considerado como fundamental para ayudar en la monitorización de red, pero en realidad, SNMP no está concebido solamente para la monitorización, sino que se desarrolló con el objetivo de proporcionar una interfaz que permitiera realizar además configuraciones sobre los dispositivos de una forma estandarizada, sin embargo, esto no se consiguió, quedando relegado como un protocolo de monitorización. Esto es debido a diferentes factores, entre los que se podrían destacar los siguientes:

1. En primer lugar, existen deficiencias en el diseño de SNMP que lo hacen inviable como protocolo para la configuración, es destacable por ejemplo su inseguridad, ya que no fue hasta su tercera versión que ofreció la capacidad de poder encriptar contraseñas, por lo que, aun cuando se publicó esta última versión, muchos dispositivos ejecutaban versiones antiguas e incluso hasta en 2002 se recomendaba desactivar SNMP siempre que no fuera estrictamente necesario **[7].** Por otro lado, otro de sus defectos es el uso de UDP como protocolo de transporte, lo cual lo hace inviable cuando se requiere una seguridad en la correcta transmisión de los datos.

2. En segundo lugar, la estructura definida en SMIv2 es bastante difícil de leer, proporciona demasiadas ramificaciones y se hace extremadamente compleja cuando es rellenada con todos los elementos configurables de un dispositivo; además, la situación se complicaba puesto que las MIBs de la gran mayoría de dispositivos no exponían suficientes elementos configurables, y aquellos elementos expuestos solían ser específicos para el vendedor, es decir, no existe una MIB estándar.

3. En tercer y último lugar, un elemento muy diferenciador de SMIv2 con respecto a YANG es la incapacidad para diferenciar entre los datos de configuración y los de estado operacional, lo cual hace más complicada la lectura de estos y también el procesamiento de estos de una forma programática. Además, debido a que SMIv2 no usa lenguajes de enmarcado como XML, el output generado por SNMP no es fácilmente procesable a través de la programación.

Los puntos comentados no son sino un un subconjunto de todos las razones por las que SNMP no ha conseguido convertirse en el protocolo de administración de redes por excelencia. Como referencia, se muestra a continuación una pequeña parte de una MIB:

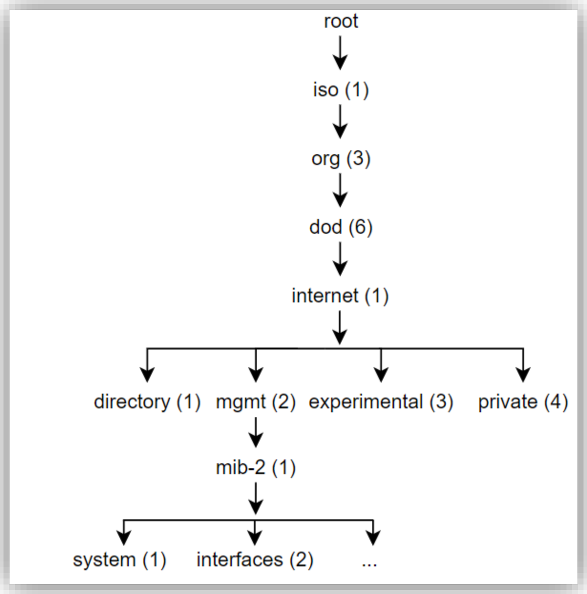

Figura 1. Estructura de una MIB

Por ejemplo, para hacer referencia a “interfaces”, habría que seguir la estructura del árbol con su numeración específica, que, entonces sería ‘1.3.6.1.2.1.2’; a cada uno de estos identificadores se les conocen como OID, y, aunque su navegación numérica es sencilla, sin ayuda de algún software especializado, entender qué funcionalidad específica exponen ciertos OIDs es complicado de determinar.

Analizadas las razones por las que SNMP/SMIv2 han sido rechazados por la industria como protocolo de configuración, se procederá a analizar YANG, el cual fue desarrollado a propósito de solventar todos los problemas que arrastraba SMIv2 y que fue desarrollado específicamente para NETCONF.

3 INTRODUCCIÓN AL LENGUAJE YANG Y SUS PROTOCOLOS

3.1 LENGUAJE DE MODELADO YANG

Para llegar a comprender de forma adecuada los protocolos basados en YANG (NETCONF y RESTCONF), es necesario realizar una introducción al propio lenguaje para facilitar la comprensión de estos. YANG es un lenguaje de modelado que fue oficialmente diseñado para el protocolo NETCONF **[8]** y estandarizado en 2010 **[9]**, y permite formalizar la estructura de los datos que forman un dispositivo de red (aunque en realidad puede definir a cualquier tipo de dispositivo incluso otros objetos); este lenguaje de por sí no permite hacer una comunicación directa con un dispositivo, sino que sirve como la definición, y la instanciaciones específicas de los datos se forman a partir de esta estructuración y a través de un lenguaje de enmarcado, como pueden ser XML, JSON o YAML.

YANG proporciona funcionalidades muy avanzadas en su lenguaje, de las cuales, algunas de ellas no se utilizan apenas en modelos para dispositivos de red, por lo que en este proyecto solo se van a presentar aquellas que se pueden ver con asiduidad en la mayoría de los modelos existentes, las cuales quedan bien representadas en el modelo “openconfig-if-ip” **[10]:**

```
module: openconfig-if-ip
  augment /oc-if:interfaces/oc-if:interface/oc-if:subinterfaces/oc-if:subinterface:
    +--rw ipv4
       +--rw addresses
       |  +--rw address* [ip]
       |     +--rw ip        -> ../config/ip
       |     +--rw config
       |     |  +--rw ip?              oc-inet:ipv4-address
       |     |  +--rw prefix-length?   uint8
       |     +--ro state
       |     |  +--ro ip?              oc-inet:ipv4-address
       |     |  +--ro prefix-length?   uint8
       |     |  +--ro origin?          ip-address-origin
```

Figura 2. Vista en formato “tree” de un modelo YANG

La imagen presentada corresponde a la vista en formato “tree” de un modelo YANG “openconfig-if-ip”; este tipo de formato permite leer fácilmente el modelo en contraposición de su formato original, que sería el siguiente:

```
container ipv4 {
  description
    "Parameters for the IPv4 address family.";

  container addresses {
    description
      "Enclosing container for address list";

    list address {
      key "ip";
      description
       "The list of configured IPv4 addresses on the interface.";

      leaf ip {
        type leafref {
          path "../config/ip";
        }
        description "References the configured IP address";
      }

      container config {
        description "Configuration data for each configured IPv4
        address on the interface";

        uses ipv4-address-config;

      }

      container state {

        config false;
        description "Operational state data for each IPv4 address
        configured on the interface";

        uses ipv4-address-config;
        uses ipv4-address-state;
      }
```

Figura 3. Formato original de un modelo YANG

Dicho lo cual, se procede por tanto a introducir los conceptos más elementales de YANG:

1. En primer lugar, está el nombre del **módulo**, definido por la palabra clave ‘module’; el nombre del módulo ha de identificar de forma inequívoca y universal al modelo, por lo que este ha de ser usado de forma exclusiva para el modelo para el que ha sido definido.

2. En segundo lugar, se define el **contenedor**, declarado por la palabra clave “container”; este elemento no tiene un valor de por sí, sino que agrupa a una serie de elementos que parten de este nodo; en este caso, en el formato “tree” se puede observar que hay cuatro contenedores: “ipv4”, “addresses”, “config” y “state”.

3. En tercer lugar, se define la **lista**, declarada por la palabra clave “list”; una lista, al igual que el contenedor, no tiene un valor específico, sino que puede albergar elementos internos, sin embargo, tiene ciertas diferencias. La primera es que permite la posibilidad de que existan varias instancias de esta, como, por ejemplo, una lista de direcciones como es en este caso. En segundo lugar, como una lista pretende definir una serie de elementos, es necesaria una llave, “key” (identificada entre corchetes), que permita distinguir un elemento de la lista con respecto al otro, por eso es sencillo identificar en el árbol que “address” corresponde a este tipo de elemento, aunque también, el carácter * adjunto al nombre de esta indica de por sí que el elemento dado es una lista.

4. En cuarto lugar, se definen los elementos terminales del modelo, los que tienen un valor declarado; a este tipo de valores se les llaman **hojas**, “leaf”, con la analogía del árbol, donde todos los elementos anteriores por los que es necesario navegar para llegar al “leaf” serían las ramas; en este caso, es sencillo ver que los elementos dentro de los containers “config” y “state” son hojas, con sus tipos de valor definidos; aunque también lo es “ip” junto debajo de la declaración de la lista “address”; esto es un caso especial que se va a comentar en el punto siguiente.

5. En quinto lugar, en este modelo, se está haciendo uso de un tipo especial en YANG llamado “**leafref**”; este tipo de variable, como su nombre indica, es una referencia a otro “leaf” o un “leaf-list” este último permite tener varios elementos definidos en vez de uno), en este caso, siguiendo la referencia dada “-> ../config/ip”, este “leafref” hace referencia al elemento “ip” dentro de la lista “address”.

Por último, en este modelo se está haciendo uso de "**augment**", lo cual permite añadir los nodos definidos a otro modelo distinto; en este caso, aumenta el modelo "openconfig-interfaces", esto se puede saber porque "oc-if" se define como un prefijo para este modelo al principio del documento. También, se puede conocer sobre qué nodo incide el aumento; en este caso, es sobre el nodo "subinterface", que a su vez tiene sus nodos superiores, tal y como se puede observar en la figura 2. En la siguiente ilustración se muestra como YANG importa sus módulos y le asigna un prefijo a cada uno de ellos:

```
// import some basic types
import openconfig-inet-types { prefix oc-inet; }
import openconfig-interfaces { prefix oc-if; }
```

Figura 4. Importación de modelos YANG y asignación de prefijos

Sabiendo ahora el significado de cada uno de los elementos que aparecen en el árbol del modelo "openconfig-if-ip", se puede determinar que es un modelo cuyo propósito, en este caso (el modelo completo es más extenso y con aumentos adicionales para otros modelos) define el direccionamiento IPv4 para una subinterfaz. Además, como se comentó anteriormente, YANG separa los datos de configuración con respecto a los operacionales, y en este caso es fácil de discernir cuál es cual, debido a que aquellos elementos que empiecen con el prefijo "rw" (read-write) son aquellos configuracionales, y cuyos valores se pueden por tanto modificar, mientras que aquellos con el prefijo "ro" (read-only) son operacionales, y solo indican el estado, en este caso, del direccionamiento IPv4. Por último, algunos nodos van acompañados en su definición por el carácter "?"; esto indica que el valor indicado es opcional, y no es necesario para validar una instancia dada del modelo, aunque en este caso en concreto, la llave de la lista "address" es "ip", y se da el caso que "ip" hace referencia al elemento opcional de su mismo nombre en el contenedor "config", por lo que en realidad este elemento es necesario definirlo dos veces para que una instancia de datos sea válida a partir de este modelo.

Ahora bien, este lenguaje no tiene mayor utilidad si no existiera la posibilidad de que los dispositivos se comunicaran a través de este modelado de los datos, pero para la instanciación de datos reales, se hace uso de los lenguajes de enmarcado, de los cuales, los tres más conocidos y utilizados junto a YANG son XML, JSON y YAML, por lo que se va a mostrar un ejemplo de configuración del modelo mostrado utilizando cada uno de estos lenguajes:

```xml
<ipv4
    xmlns="http://openconfig.net/yang/interfaces/ip">
    <addresses>
        <address>
            <ip>10.255.254.1</ip>
            <config>
                <ip>10.255.254.1</ip>
                <prefix-length>30</prefix-length>
            </config>
        </address>
    </addresses>
</ipv4>
```

Figura 5. Instanciación de una dirección IPv4 en el modelo OpenConfig - XML

```json
{
   "openconfig-if-ip:ipv4":{
      "addresses":{
         "address":[
            {
               "ip":"10.255.254.1",
               "config":{
                  "ip":"10.255.254.1",
                  "prefix-length":30
               }
            }
         ]
      }
   }
}
```

Figura 6. Instanciación de una dirección IPv4 en el modelo OpenConfig - JSON

```yaml
---
'openconfig-if-ip:ipv4':
  addresses:
    address:
    - ip: 10.255.254.1
      config:
        ip: 10.255.254.1
        prefix-length: 30
```

Figura 7. Instanciación de una dirección IPv4 en el modelo OpenConfig - YAML

Como se puede observar, la estructura es idéntica en cada uno de los distintos lenguajes de modelado, debido a que se sigue el patrón definido por el modelo YANG, esto permite que, al generar una instanciación válida de los datos, si el dispositivo tiene el modelo en cuestión implementado, pueda hacer las modificaciones pertinentes sin importar el fabricante ni el sistema operativo que ejecute. Además, se puede observar que, aunque la estructura entre los distintos lenguajes de enmarcado es idéntica, en el caso de usar XML, es necesario definir formalmente el "XML namespace (xmlns)", mientras que en JSON y YAML es suficiente con especificar el nombre del módulo.

3.2 NETCONF

Como se comentó anteriormente, SNMP no fue eficaz como protocolo que estandarizara la configuración de los dispositivos de red debido a las deficiencias arrastradas desde la primera versión de este, y SMI/SMIv2 no proporcionaba suficientes funcionalidades como para aunar a todos los dispositivos de red, por lo que al final se creaban MIBs muy específicas para cada dispositivo, impidiendo crear configuraciones universales. Dicho lo cual, se comenzó el proyecto de crear una nueva versión de SMI, conocida como SMIng, que pudiera dar respuesta a las carencias observadas en las versiones anteriores, pero que nunca llegó a pasar de la fase experimental. Por otro lado, la IETF era consciente de que SNMP había fracasado como protocolo de configuración a favor del uso del CLI, por lo que en 2003 creó un nuevo grupo de trabajo que se ocuparía de desarrollar un nuevo estándar, cuyo resultado daría lugar al protocolo de configuración de red NETCONF, publicado por primera vez en el RFC 4741 **[8]** en 2006, pero que sigue en constante desarrollo, con extensiones publicadas en posteriores RFC.

Este nuevo protocolo suple muchas de las carencias que hacían que SNMP fuera rechazado como herramienta de configuración estandarizada, y por tanto viable para ser automatizada:

- Cumple los principios de ACID, una serie de propiedades que definen cómo deben de ser las transacciones de una base de datos para considerarse seguras de ejecutar y que no puedan inestabilizar el funcionamiento de esta; en el entorno de configuración de redes se consideran muy deseables puesto que una configuración inconsistente puede inestabilizar toda la red:

    - **Atomicidad**: El principio por el cual una transacción, que puede estar formada por varias instrucciones, solo entra en vigor si todas estas instrucciones se ejecutan, de otro modo, se descarta, y por tanto se evita cualquier riesgo de inconsistencia en la configuración.

    - **Consistencia**: Asegura que las transacciones realizadas nunca dejen a la base de datos, o en este caso, el dispositivo de red, en un estado de configuración intermedio.

    - **Aislamiento** (“Isolation” en inglés): Asegura que una transacción siempre se va a realizar sin interferencias de otras configuraciones, aunque existan más de una transacción en funcionamiento de forma consistente.

    - **Durabilidad**: Es la propiedad que garantiza que una transacción seguirá vigente una vez ejecutada, aunque exista cualquier interrupción del funcionamiento del dispositivo.

- Desde su concepción se desarrolló para que todas sus comunicaciones fuesen seguras, para ello hace uso de otros protocolos, que proporcionan tales capacidades. Existen diferentes protocolos compatibles que NETCONF puede usar como transporte, como por ejemplo el uso de SSL/TLS, pero en realidad el protocolo que más se utiliza para ello y cuya implementación es más común en dispositivos compatibles con NETCONF es usar SSHv2, de esta manera, se utilizan todas las ventajas de este protocolo, siendo innecesario crear una forma específica de transporte para NETCONF.

- Al contrario que SNMP, NETCONF distingue entre el estado operacional y los datos de configuración a través de YANG, por lo que la información deseada es mucho más fácil de procesar por ordenador y también de leer a simple vista. Además, permite interaccionar con distintos archivos de configuración, conocidos como "datastores", esto ofrece la capacidad de "rollback" si la configuración aplicada es incorrecta.

- Fácil capacidad de expansión, al ser posible implementar funcionalidades más allá de las básicas ("capabilties"), y también nuevos modelos que permitan interaccionar con otras funciones del dispositivo.

- Gran aceptación en la industria, y cooperación entre distintas organizaciones para crear modelos universales; la entidad más destacable en este aspecto es OpenConfig **[11]**, que ofrece modelos YANG mucho más extensos que los proporcionados por la IETF.

- YANG es un lenguaje de modelado mucho más sencillo de leer para los humanos y provee información sobre los nodos que se definen en él a través de nombres significativos, lo cual contrasta en gran medida con la creciente complejidad de SMI/SMIv2, además, los modelos son publicados en Github a disposición de cualquier usuario **[10]**.

Las puntos descritos anteriormente no son sino una pequeña muestra de todos los problemas que los protocolos basados en YANG pretenden resolver con respecto a su antecesor, SNMP, el cual, como se ha comentado, fue insuficiente como protocolo de administración, resultando en que el uso del CLI continuara siendo la herramienta principal de configuración de redes, pero que, hoy en día, se torna cada vez más y más inviable, por eso se impulsaron nuevos protocolos que hicieran uso de lenguajes orientados específicamente a la administración de redes, como es YANG, siendo NETCONF el primer protocolo que hace uso del mismo (aunque anteriormente se usaba con distintos lenguajes de modelado, ya que YANG fue estandarizado posteriormente a NETCONF).

El “stack” de protocolos para NETCONF es el siguiente:

Figura 8. Pila de protocolos en NETCONF

Aunque existen alternativas, SSHv2 es el protocolo casi en exclusiva utilizado para transportar NETCONF; la versión de SSHv1, además de considerarse obsoleta e insegura, no permite el uso de subsistemas, por lo que es directamente incompatible con el protocolo [**12**], y, además, muchos dispositivos no permiten ni siquiera configurar SSHv1.

Directamente superior a la capa de transporte se sitúa la capa de mensajes; en este caso existen dos posibles tipos de mensajes que se pueden realizar con NETCONF, el cual usa “Remote Procedure Calls” (RPCs) a través de XML para transmitir los mensajes. Mientras que <rpc> define una solicitud, <rpc-reply> es la respuesta a esta solicitud; como pueden existir varias sesiones activas, se hace un seguimiento de ellas a través del campo “session-id”, que identifica de forma inequívoca una sesión activa, y, por tanto, un <rpc> con un “session-id” determinado, tendrá como respuesta del servidor un <rpc-reply> con el mismo valor.

Como se ha explicado, existe tanto <rpc> como <rpc-reply>; es el cliente por tanto el que realiza las peticiones, y por tanto el que emite los <rpc>, mientras que el servidor emitirá las respuestas correspondientes a través de <rpc-reply>. Existen multitud de solicitudes diferentes que un cliente puede realizar, las cuales son, según el RFC6241 **[13]**:

Operación	Descripción
<get-config>	Obtiene información de la configuración del dispositivo.
<edit-config>	Permite modificar parte o toda la configuración de un dispositivo.
<copy-config>	Permite crear y reemplazar una configuración entera desde un archivo (datastore) a otro.
<delete-config>	Permite borrar un datastore, exceptuando el <running> datastore, el cual no puede ser borrado.
<lock>	Permite bloquear un datastore para evitar cambios concurrentes de distintas transacciones.
<unlock>	Operación inversa a <lock>; se ha de usar cuando se ha terminado de configurar un datastore bloqueado anteriormente (aunque el cierre de sesión desbloquea cualquier elemento bloqueado, es conveniente usar <unlock> por si no se diera el caso de una terminación apropiada de la conexión).

<get>	Obtiene información del estado y la configuración del dispositivo.
<close-session>	Cierra la sesión; la ventaja de usar esta operación con respecto a terminar la sesión directamente es que permite a NETCONF liberar todos los “locks” y recursos en uso.
<kill-session>	Fuerza el fin de la sesión; es similar a <close-session>, pero aborta cualquier transacción que esté en proceso para devolver el dispositivo a su estado estable anterior antes de cerrar la sesión.

Tabla 3. Operaciones en NETCONF

Por otro lado, si bien NETCONF soporta tres tipos de “datastore” diferentes, no siempre han de estar todos presentes en el dispositivo, ya que, en realidad, solamente es obligatorio incluir <running>, el cual está definido en las capacidades básicas de NETCONF. Aun así, el servidor puede incluir los demás “datastore” si posee las capacidades mostradas a continuación:

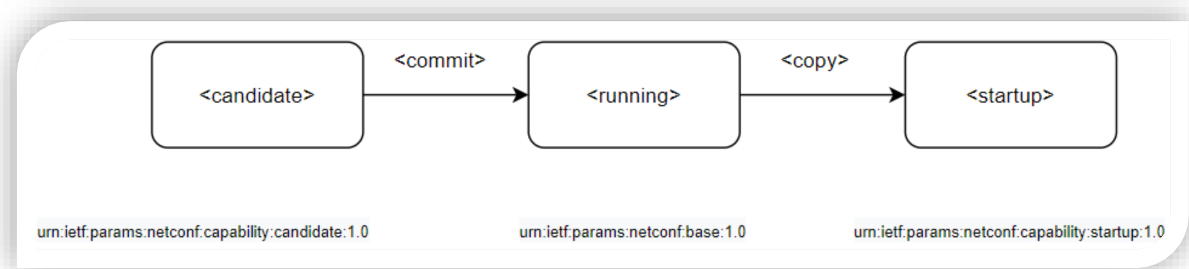

Figura 9. “Datastores” en NETCONF

Por tanto, según la ilustración mostrada, un cliente solamente podrá contactar con un “datastore” determinado si la capacidad pertinente está activada en el servidor, de otro modo, el servidor responderá con un error e informando de que la capacidad correspondiente no está soportada actualmente. En la figura siguiente se muestra cómo se intenta acceder a <candidate> “datastore” sin que el servidor tenga la capacidad correspondiente para ello:

```
ncclient.operations.rpc.RPCError: Unsupported capability :candidate
```

Figura 10. Error al intentar modificar <candidate> sin soporte para ello

Por último, en el lugar superior del "stack", se define el propio "payload", que, como se explicó anteriormente, usa el lenguaje de modelado YANG y XML como único lenguaje de enmarcado compatible.

Para terminar este subapartado, se procede a analizar el proceso de inicio de conexión entre dos dispositivos NETCONF, lo que sería el cliente (Manager) y el servidor (Agent). Para establecer una conexión correcta, los dispositivos deben compartir con el otro extremo las "capabilities" que soportan cada uno; esto es un proceso fundamental, ya que solamente aquellas "capabilities" soportadas por ambos extremos serán las que el "Manager" pueda usar para comunicarse con el "Agent. Para este intercambio de "capabilities" se usan los mensajes "Hello":

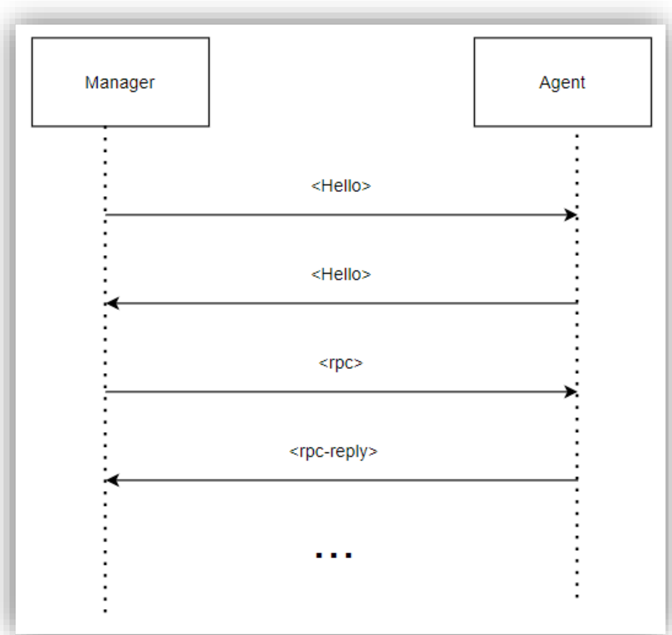

Figura 11. Diagrama de establecimiento de sesión en NETCONF

A partir del esquema se podría suponer que es el cliente (“Manager”), quien envía primero su <Hello> al ser el que inicia la conexión y posteriormente el servidor (“Agent”) responde con su propio <Hello>, pero en realidad, tras el establecimiento de conexión entre los dos dispositivos, cada uno enviará sin esperar al otro su propio mensaje <Hello>; en cualquier caso, el servidor no tratará los <rpc> del cliente mientras que este intercambio no se haya realizado por ambas partes.

A continuación, se muestra un ejemplo real usando directamente la aplicación ssh en Ubuntu, aunque en este caso el “Agent” envía primero su mensaje <Hello>, ya que se está realizando una conexión de forma manual, por lo que solamente cuando el “Manager” envía su propio <Hello> se considera establecida la conexión y pueden iniciarse mensajes <rpc>. También se puede ver como los modelos soportados por el servidor son comunicados al cliente (esto es una opción propia del dispositivo, algunos no informan de los modelos instalados en su mensaje <hello>), de esta manera, el cliente podrá interactuar con estos modelos según los “capabilities” que haya negociado con el servidor (se saber cuándo se muestra un “capability” en vez de un modelo porque, al contrario que en un modelo YANG, la dirección urn incluye la palabra “capability:”):

Mensaje <hello> del servidor

```
root@PC:~# ssh username@192.168.1.1 -p 830 -s netconf
username@192.168.1.1's password:
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
<capability>urn:ietf:params:netconf:base:1.1</capability>
<capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
<capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:RFC-1212?module=RFC-1212</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:RFC-1215?module=RFC-1215</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:RFC1155-SMI?module=RFC1155-SMI</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:RFC1213-MIB?module=RFC1213-MIB</capability>
<capability>urn:ietf:params:xml:ns:yang:smiv2:RFC1315-MIB?module=RFC1315-MIB</capability>
<capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
<capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
<capability>urn:ietf:params:netconf:capability:interleave:1.0</capability>
<capability>
        urn:ietf:params:netconf:capability:notification:1.1
      </capability>
</capabilities>
<session-id>36</session-id></hello>]]>]]>
```

Mensaje <hello> del cliente

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <capabilities>
        <capability>urn:ietf:params:netconf:base:1.0</capability>
    </capabilities>
</hello>]]>]]>
```

Figura 12. Establecimiento de sesión NETCONF a través de un terminal

Una vez introducido el mensaje <hello> en el cliente SSH, si está correctamente estructurado, la conexión queda establecida, y se pueden enviar mensajes <rpc>. Otra cosa a tener en cuenta es que el id de la sesión lo establece el “Agent” y lo envía con su mensaje “<hello>; el “Manager” no envía en ningún momento la id de la sesión en su mensaje <hello> (de hecho, si es así NETCONF cierra la sesión inmediatamente). Además, el protocolo solo exige como mínimo que se soporten las capacidades base, definidas en **“urn:ietf:params:xml:ns:netconf:base:1.0“**. De forma consiguiente, una vez establecida una sesión, se pueden enviar <rpc> al servidor; en la siguiente figura se muestra una operación <get> como ejemplo:

Operación <get> enviada al servidor

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="65">
<get>
<filter type="subtree">
<interfaces-state xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
<interface>
<name>GigabitEthernet1</name>
</interface>
</interfaces-state>
</filter>
</get>
</rpc>]]>]]>
```

Respuesta <rpc-reply> del servidor

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="65"><data><interfaces-state xmlns="urn:ietf:param
s:xml:ns:yang:ietf-interfaces"><interface><name>GigabitEthernet1</name><type xmlns:ianaift="urn:ietf:params:xml:ns:yang:
iana-if-type">ianaift:ethernetCsmacd</type><admin-status>up</admin-status><oper-status>up</oper-status><last-change>2020
-12-09T04:58:37.000526+00:00</last-change><if-index>1</if-index><phys-address>0c:a7:4e:98:50:00</phys-address><speed>102
4000000</speed><statistics><discontinuity-time>2020-12-09T02:56:24.000796+00:00</discontinuity-time><in-octets>708278</i
n-octets><in-unicast-pkts>8276</in-unicast-pkts><in-broadcast-pkts>0</in-broadcast-pkts><in-multicast-pkts>0</in-multica
st-pkts><in-discards>0</in-discards><in-errors>0</in-errors><in-unknown-protos>0</in-unknown-protos><out-octets>1644432<
/out-octets><out-unicast-pkts>7287</out-unicast-pkts><out-broadcast-pkts>0</out-broadcast-pkts><out-multicast-pkts>0</ou
t-multicast-pkts><out-discards>0</out-discards><out-errors>0</out-errors></statistics></interface></interfaces-state></d
ata></rpc-reply>]]>]]>
```

Figura 13. Transmisión de <rpc> y respuesta <rpc-reply> en NETCONF a través de un terminal

Se puede ver de esta manera que se puede interaccionar con el protocolo NETCONF a través de una consola interactiva SSH, aunque ello supone un proceso engorroso y no recomendado en absoluto, además de que el output obtenido no es fácil de leer de esta manera; para ello, existen soluciones mejores, como, por ejemplo, el uso de scripts Python con librerías específicas (en este proyecto se usará la librería ncclient) que permitan interactuar con NETCONF de forma programática (al fin y al cabo, NETCONF no está diseñado para ser una interfaz orientado a los humanos como en el caso del CLI).

3.3 RESTCONF

El protocolo NETCONF no es el único en usar el lenguaje de modelado YANG, aunque sí el primero. Existe otro protocolo estandarizado más nuevo que se basa en la arquitectura REST, llamado RESTCONF **[14]** que funciona como alternativa**,** pero no sustituto, puesto que este último no soporta actualmente todas las opciones que ofrece NETCONF. La forma de interacción es ciertamente similar a NETCONF debido al uso del mismo lenguaje de modelado, aunque con algunas diferencias fundamentales debido a que RESTCONF es, al fin y al cabo, un protocolo RESTful, por lo que hace uso de las operaciones HTTP para llevar a cabo sus transacciones, y, por supuesto, usa HTTPS como método de transporte, en vez de SSHv2, por lo que en este caso no se trata de una sesión activa, como en NETCONF, sino una o más peticiones que el cliente hace al servidor a través de las URLs expuestas con las operaciones correspondientes, las cuales se exponen en la siguiente tabla **[14]**:

Operación/Método HTTP	Equivalente NETCONF	Descripción	Equivalencia con CRUD
OPTIONS	<none>	Petición en la que el cliente solicita al servidor qué métodos soporta para un recurso determinado.	N/A
HEAD	<get> y <get-config>	Petición en la que el cliente solicita al servidor solamente los headers en comparación al método GET que obtiene toda la información útil.	READ
GET	<get> y <get-config>	Solicita al servidor datos y metadatos de un recurso.	READ
POST	<edit-config>	Permite al cliente solicitar al servidor la creación en este último de un nuevo recurso o solicitar una operación específica (comparable a <rpc> en NETCONF).	CREATE
PUT	<copy-config> y <edit-config>	Método similar a POST, aunque en POST es el servidor quien decide la localización de los nuevos recursos, mientras que en PUT es el cliente quien especifica la dirección del nuevo recurso; también sirve para editar un recurso existente, por tanto.	CREATE y UPDATE

PATCH	<edit-config>	Permite la edición de un recurso, aunque al contrario que POST y PUT, en caso de que el recurso a modificar no exista, PATCH creará tal recurso.	UPDATE
DELETE	<edit-config>	Permite borrar el recurso solicitado del servidor.	DELETE

Tabla 4. Operaciones en RESTCONF

A continuación, se muestra una solicitud análoga al <rpc> mostrado como ejemplo en el apartado anterior con NETCONF en la que se obtiene la información pertinente a la interfaz "GigabitEthernet1" de un dispositivo de red usando RESTCONF con la aplicación curl:

Figura 14. Petición GET en RESTCONF a través de cURL

También es posible la configuración del dispositivo a través de RESTCONF, en el siguiente ejemplo se muestra la configuración de una interfaz y una posterior consulta de la configuración usando modelos de datos OpenConfig a través de la aplicación Postman **[15]**, la cual permite gestionar de forma más cómoda las peticiones en HTTP que usando curl:

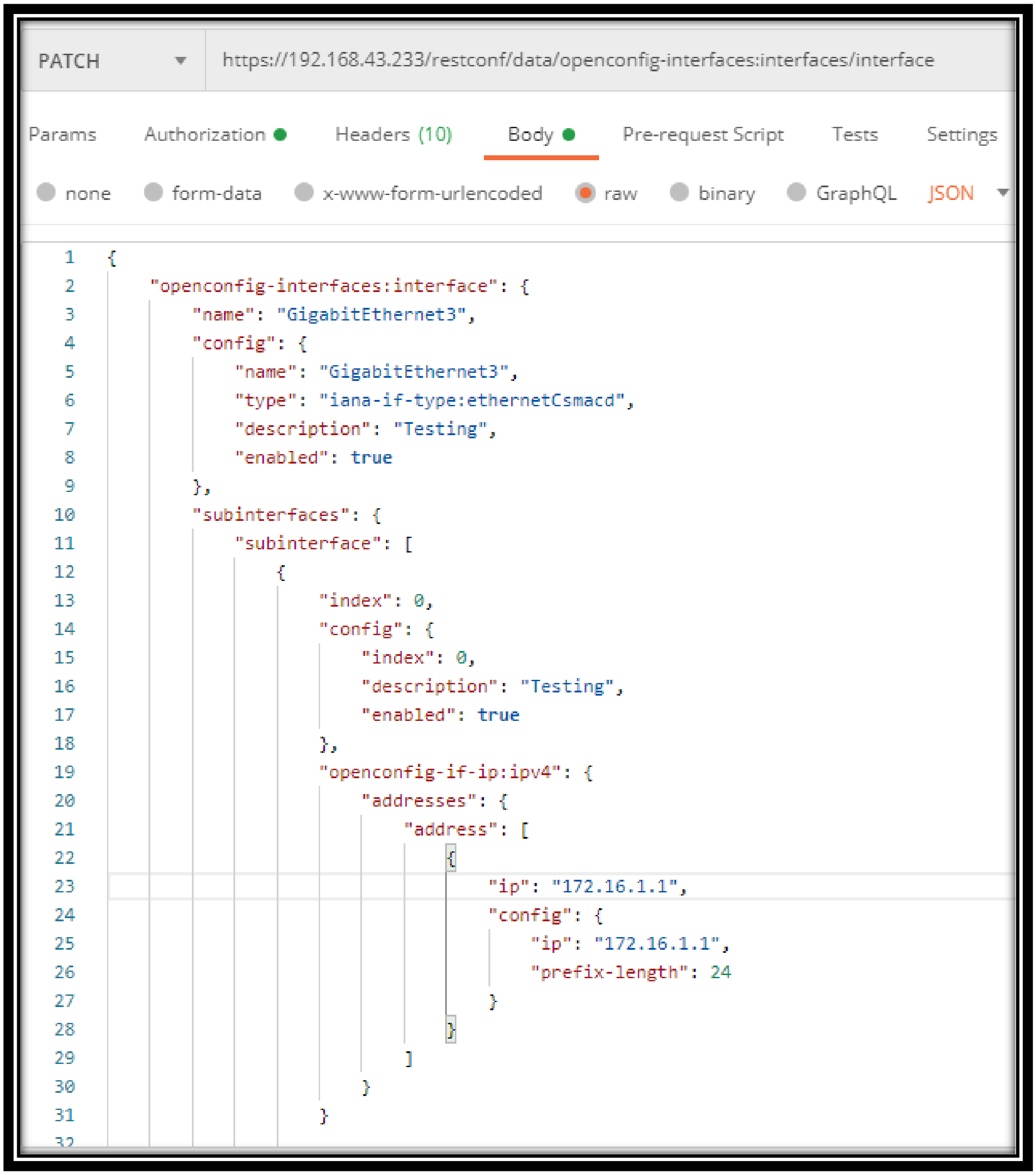

Figura 15. Petición PATCH para modificar asignar direccionamiento IP a una interfaz

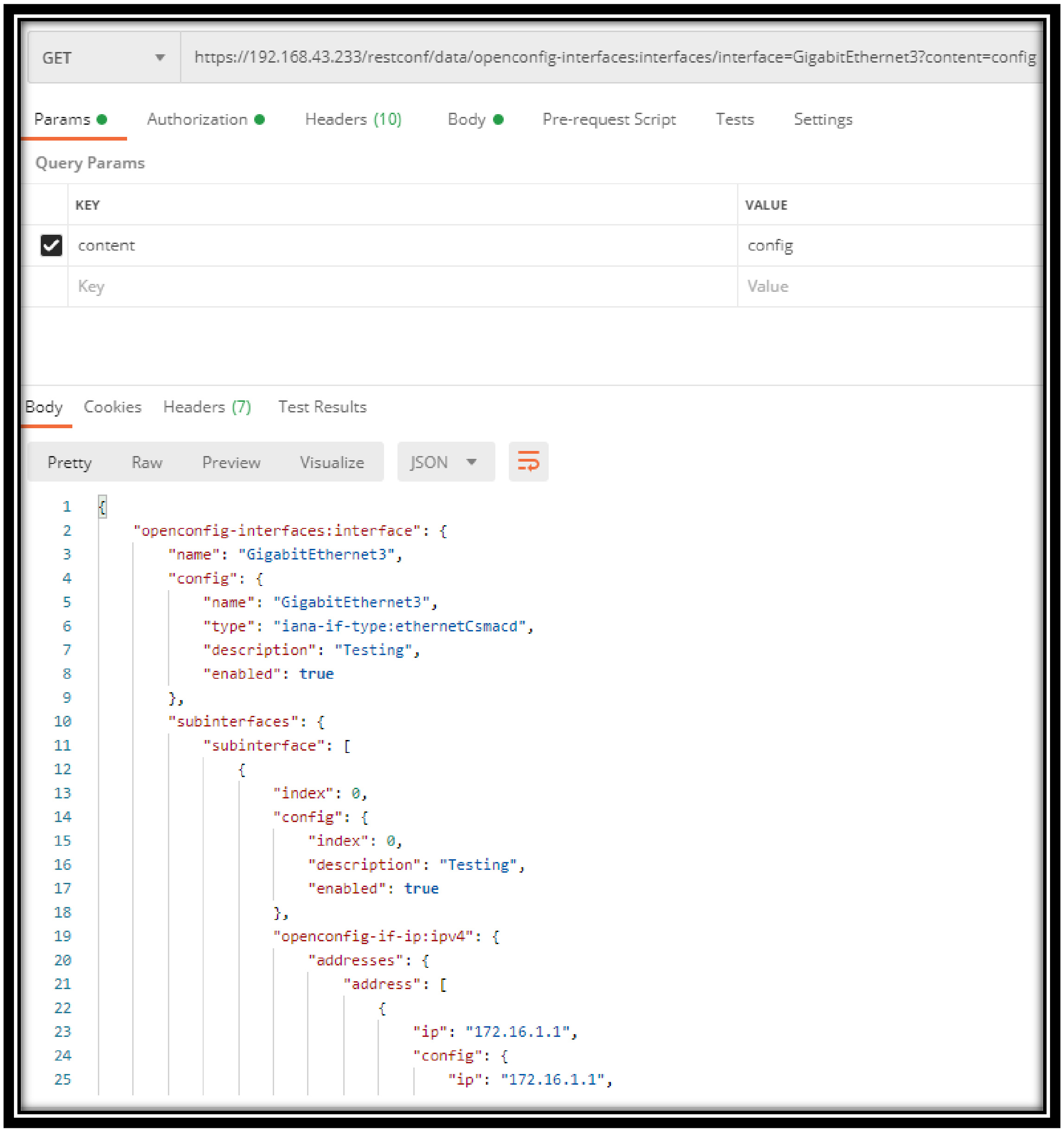

Figura 16. Petición GET para obtener los datos de configuración de una interfaz

Por último, hay que considerar que la información devuelta por el servidor, aunque es similar al apartado anterior ya que se está haciendo referencia al mismo recurso, sí es cierto que es más legible, esto es debido a que RESTCONF permite solicitar la información tanto en XML como en JSON, aunque, si fuera necesario, se podría obtener la información en XML:

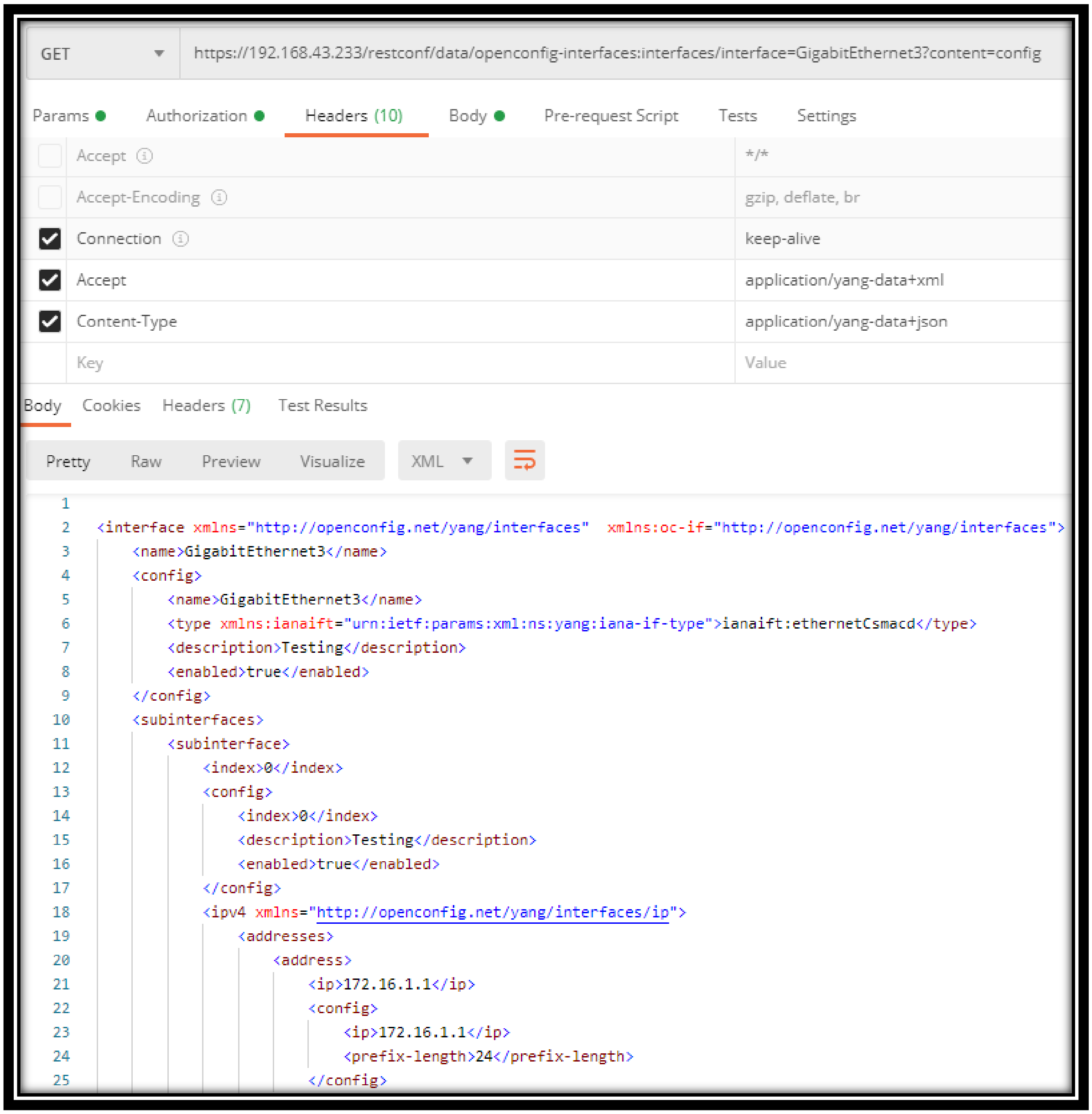

Figura 17. Petición GET con respuesta codificada en XML

Concluyendo, como se ha comentado al principio de este apartado, RESTCONF en la actualidad es más una alternativa que soporta un subconjunto de todas las operaciones que NETCONF puede realizar actualmente. Por ejemplo, RESTCONF no tiene una función análoga a <lock>, o la capacidad de interactuar de forma específica con un “datastore” determinado, ni tampoco la capacidad de realizar <commit> ni <rollback> **[16]**, por lo que, para ciertas situaciones, es posible que RESTCONF no sea el protocolo más apropiado; de todas formas, proporciona una alternativa cuando el uso de este resulte más cómodo que una sesión NETCONF, pues, como se ha podido ver, las solicitudes por RESTCONF son sencillas de realizar a través de programas como curl o Postman.

4 AUTOMATIZACIÓN DE LA CONFIGURACIÓN EN UNA RED PEQUEÑA

En la sección anterior se han presentado los protocolos basados en YANG, así como una introducción a aquellos elementos más usados en los modelos; estos modelos ofrecen programabilidad a los dispositivos que lo soportan, es decir, ofrece una serie de normas y una interfaz común por las que se pueden automatizar de forma sencilla, sin embargo, existen también una gran cantidad de dispositivos que no soportan estos protocolos, por lo general, debido a su antigüedad, pero cuya presencia de estos en la industria está generalizada, ya que, al fin y al cabo, estos dispositivos son costosos y no se suelen cambiar a menos que sea absolutamente necesario, por lo que en estos casos, la única forma de automatizar la configuración de estos es a través del uso del CLI, para lo cual existen diferentes alternativas, pero en este proyecto se va a hacer una demostración con una de las librerías más utilizadas para este fin, Netmiko **[17]**, para el lenguaje Python, el cual permite simplificar la configuración de dispositivos de distintos vendedores y la obtención de información a través del protocolo SSH.

Para simular la red, se utilizará el software de código abierto GNS3 **[18]**, este programa permite crear topologías de red virtualizadas a través de imágenes completas de distintos sistemas operativos, por lo que es posible diseñar redes cuyo comportamiento sea muy cercano al de una red física. A su vez, las herramientas utilizadas para la automatización serán explicadas a la vez que se presenta la metodología realizada.

4.1 AUTOMATIZACIÓN DEL CLI

En primer lugar, se dispone a automatizar una topología de red pequeña, cuyos dispositivos ejecutan Cisco IOS **[19]**, los cuales no implementan modelos YANG, y, por lo tanto, no son configurables con los métodos presentados en la sección anterior, así que por lo tanto se realizará la automatización por comandos de CLI a través de Netmiko. La topología de la red es la siguiente:

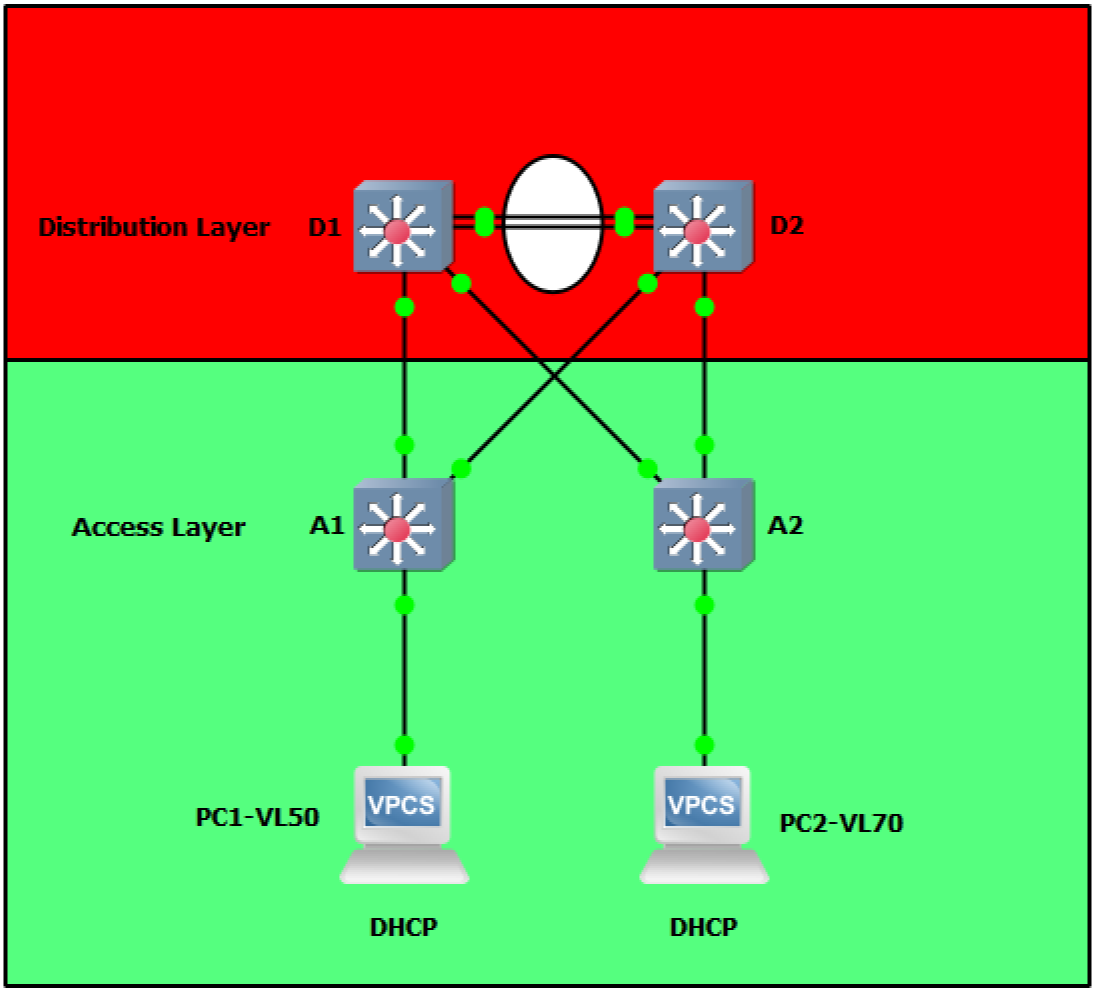

Figura 18. Topología de red inicial en GNS3

Esta topología está formada por dos capas, la inferior, llamada capa de acceso, expone sus puertos a los dispositivos finales, y solamente conectan directamente con otros dispositivos a través de los conmutadores de la capa superior, la capa de distribución. En segundo lugar, los conmutadores de la capa de acceso funcionan exclusivamente en la capa 2, mientras que los de la capa de distribución funcionan adicionalmente en la capa 3, funcionando como servidor DHCP (D1) y de enrutamiento. Por lo tanto, el dominio de capa 2, y, por ende, el de STP, se extiende desde los conmutadores de la capa de acceso hasta las interfaces de D1 y D2, incluido el "etherchannel" entre estos dos. Antes de poder automatizar estos dispositivos, es necesario habilitar SSH, para ello, al ser dispositivos con una configuración nueva, sería necesario conectarse a ellos a través de un cable de consola, pero en este caso, a través de GNS3 se puede emular esta conexión.

La configuración para habilitar SSH en cada uno de los cuatro dispositivos será la misma, que se muestra a continuación (obviamente la contraseña debe ser más segura, aunque en este entorno de pruebas no es necesario, por lo que se realiza una configuración lo más sencilla posible):

```
! Hostname will vary depending on the device being configured
hostname A1
ip domain name GNS3LAB
crypto key generate rsa modulus 2048

username admin privilege 15 secret adminpassword

ip ssh version 2

! Interface on which the manager will connect to the device
interface G2/0
! Necessary if the configured device is a Layer 3 Switch
    no switchport
!
    ip address dhcp
    no shutdown

line vty 0 4
    transport input ssh
    login local
```

Figura 19. Configuración básica de SSHv2 para IOS

Una vez habilitado SSHv2 en todos los dispositivos, es necesario conectar el host a estos; como los dispositivos están virtualizados dentro de una máquina, es necesario habilitar una interfaz en modo “bridge” con esta, cuya configuración dependerá en exclusiva del software de virtualización correspondiente (GNS3 soporta multitud de hipervisores). Una vez hecho esto, el host se puede conectar a través de un conmutador virtual con los demás dispositivos:

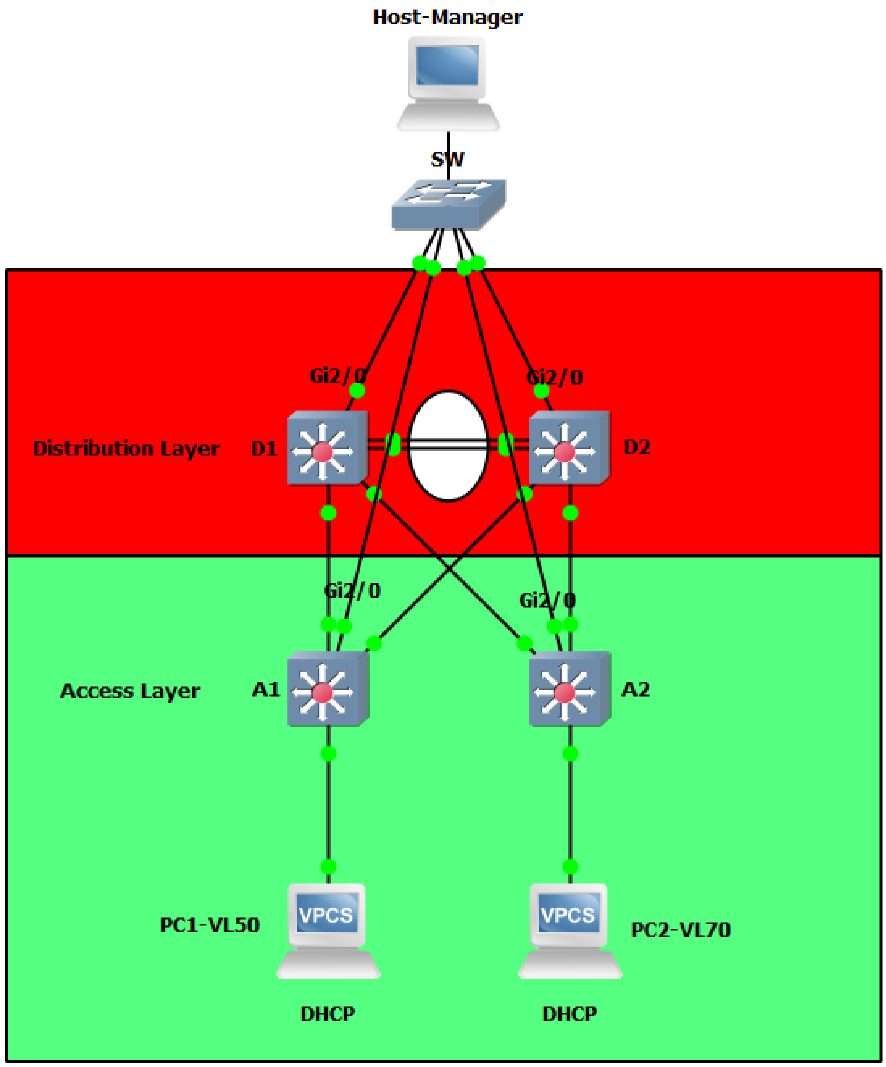

Figura 20. Topología de red inicial con el host conectado a los "switches"

Una vez realizado este paso, las interfaces de los conmutadores adquieren una IP privada propia, ya que la conexión en modo "bridge" es equivalente a conectar otro dispositivo físico desde el punto de vista de la red, por lo que la máquina virtual tiene acceso al mismo servidor DHCP que el host:

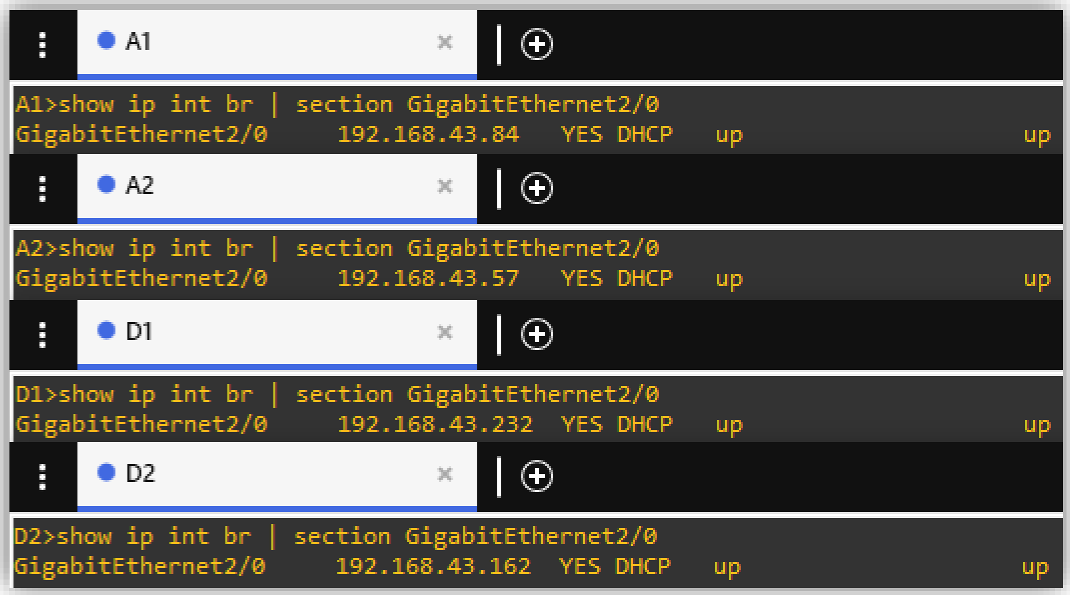

Figura 21. Output con la información IP de la interfaz G2/0 en los cuatro dispositivos

Una vez configurados los dispositivos, se hace uso de Netmiko, para ello, existen multitud de posibilidades de programar los scripts pertinentes para llevar a cabo la automatización, aunque en este proyecto se va a presentar una propuesta que hace uso del lenguaje de enmarcado YAML y un lenguaje de modelado específico para Python conocido como Jinja2 **[20]**, que permite formar estructuras de datos a partir de la sustitución de valores a través de variables y la aplicación de bloques lógicos análogos a los utilizados en cualquier lenguaje de programación.

La arquitectura de esta solución es la siguiente:

- A través de YAML se define la configuración de cada dispositivo de una manera que es fácilmente modificable y legible por cualquier persona; una parte de la configuración de A2 se presenta a continuación:

```yaml
### DEVICE CONFIGURATION ###
# Basic Configuration
hostname: A2
secret: 'enablepassword'
domain: 'GNS3Lab'

# Layer 2 Common Configuration
layer_2:
  vlans:
    - vlan: 50
      name: Sales
    - vlan: 60
      name: IT Department
    - vlan: 70
      name: Administration
    - vlan: 500
      name: Laboratory
    - vlan: 1000
      name: NATIVE
```

Figura 22. Captura del archivo de configuración A2.yaml

- Obviamente, a través del archivo YAML solamente no será suficiente para traducir los datos almacenados de una forma que el dispositivo a configurar pueda comprender, para ello, se hace uso de Jinja2, donde se define la estructura de los datos, así como variables cuyos valores asignados dependerán de aquellos contenidos en el archivo YAML; para configurar las VLANs a partir de los datos definidos en el archivo YAML mostrado en la figura anterior, habría que iterar sobre cada "key:value" definido:

```jinja
{% for vlan in layer_2.vlans %}
    vlan {{ vlan.vlan }}
    name {{ vlan.name }}
{% endfor %}
```

Figura 23. Bloque lógico en Jinja2 para generar configuración de VLANs en IOS

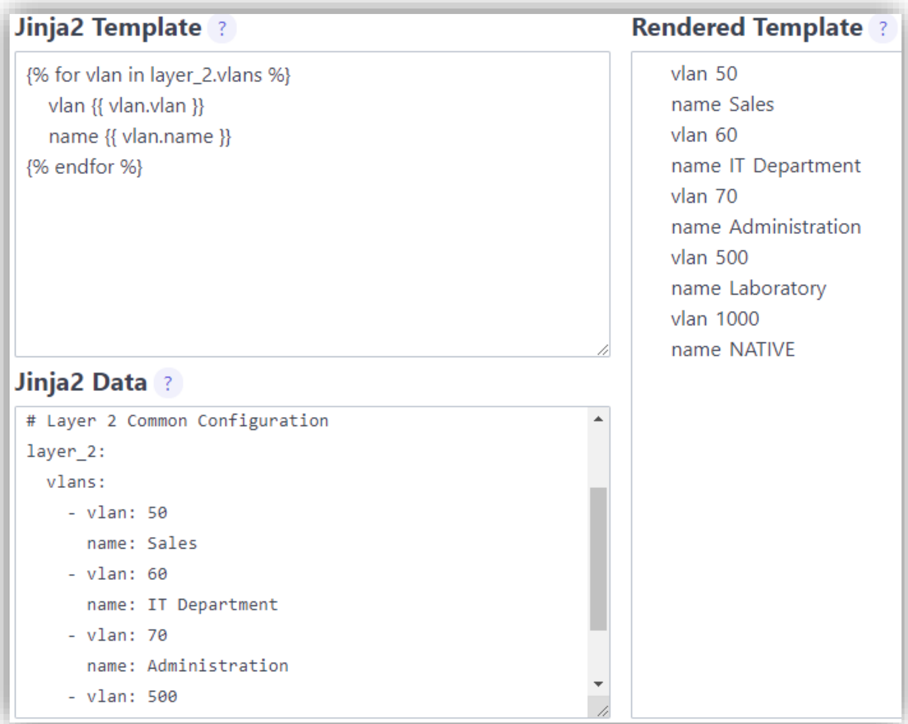

Figura 24. Output de comandos de configuración obtenidos a través de YAML y Jinja2 con J2Live [RE2]

- Una vez transformada la información contenida en YAML a través de Jinja2, será necesario enviarla a través de SSH, que es cuando entra en juego la librería Netmiko, es decir, al final habría que realizar un algoritmo que ejecutara los siguientes pasos:

    1. Leer información contenida en YAML y la transforme en un conjunto de listas y diccionarios iterables a través de Python; para ello se hace uso de la librería yaml.

    2. Definir un entorno para Jinja2 que cargue los "templates" con las opciones deseadas (dependiendo de los parámetros definidos en el entorno, un mismo "template" puede dar formato a los datos de manera algo diferente), y posteriormente, cargar la variable o el archivo con los datos YAML traducidos a

tipos iterables en Python (diccionarios y listas, en este caso) y el "template" con el que se van a interpretar estos datos. El resultado de estos dos primeros pasos es equivalente al resultado obtenido en "Rendered Template" en la figura 24.

3. Una vez obtenidos las líneas de comandos para la configuración, es necesario enviarlas a través de SSH, para lo cual se usa Netmiko; en este caso hay multitud de formas para realizar el algoritmo correspondiente; el algoritmo propuesto en este proyecto permite identificar los dispositivos a configurar a través de sus direcciones IP y sus datos de conexión específicos. Además, la transmisión de los datos se ejecuta en hilos de procesamiento diferentes a través de la librería threading en Python, lo cual permite acelerar la ejecución del programa, especialmente si se configurar muchos dispositivos a la vez.

El código del programa, así como los archivos de configuración y los "templates" usados en este proyecto están incluidos en el archivo Codigo_TFG.zip. Además, para facilitar la comprensión, se presenta la estructura básica del modelo propuesto (aunque el código del programa usado en este proyecto es hace uso de "threading" y define los hosts en otro archivo YAML, la lógica básica seguida es idéntica a la de la siguiente figura):

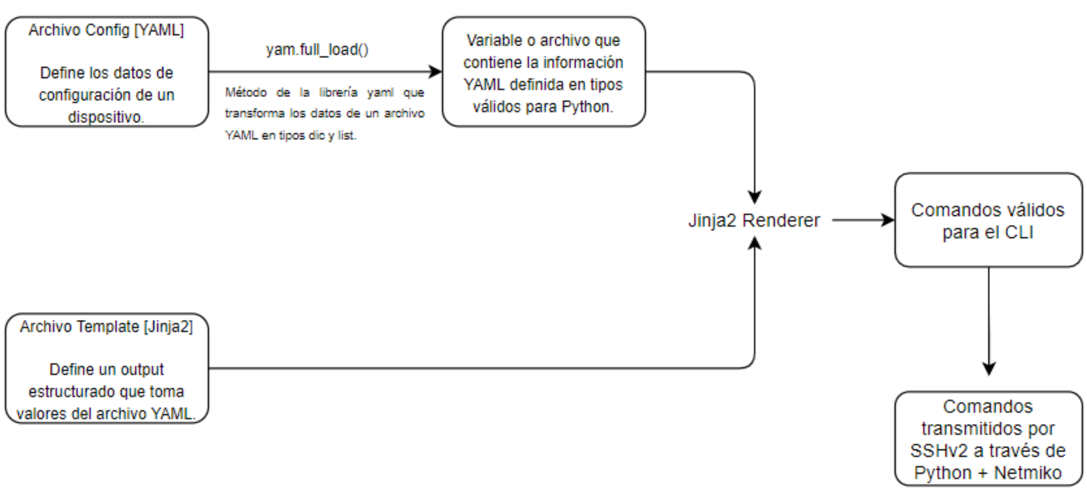

Figura 25. Arquitectura básica de la solución propuesta en esta sección

Una vez mostrada la arquitectura del algoritmo, se procede a demostrar cómo automatizar la configuración de estos cuatro dispositivos, en este caso, primero se definen los hosts en el archivo YAML correspondiente, los cuales tienen las direcciones definidas en la figura 22:

```yaml
---
hosts:
  - ip: 192.168.43.84
    ssh_user: admin
    ssh_password: adminpassword
    device_type: cisco_ios
    config_file: A1.yaml
  - ip: 192.168.43.57
    ssh_user: admin
    ssh_password: adminpassword
    device_type: cisco_ios
    config_file: A2.yaml
  - ip: 192.168.43.232
    ssh_user: admin
    ssh_password: adminpassword
    device_type: cisco_ios
    config_file: D1.yaml
  - ip: 192.168.43.162
    ssh_user: admin
    ssh_password: adminpassword
    device_type: cisco_ios
    config_file: D2.yaml
```

Figura 26. Captura del archivo hosts.yaml

Como se puede observar, contiene todos los datos necesarios para realizar una conexión por Netmiko, así como el nombre del archivo de configuración correspondiente; este archivo contiene, como se comentó anteriormente, los parámetros del dispositivo pertinente codificados en YAML, además, en el caso de este algoritmo, también se disponen los archivos jinja2 a utilizar, de esta manera no es necesario pasarlos a través de parámetros por la línea de comandos o en el código del propio programa:

```yaml
---
# Templates to be loaded
Templates:
  - IOS-BasicConf.jinja2
  - IOS-Interfaces.jinja2
  - IOS-VLAN.jinja2
  - IOS-STP.jinja2

### DEVICE CONFIGURATION ###
# Basic Configuration
hostname: D2
secret: 'enablepassword'
domain: 'GNS3Lab'
```

Figura 27. Captura del archivo D2.yaml con la lista de "templates"

De esta manera, ya solo es necesario iniciar el programa:

```
Configuration data has been saved successfully as ./OutputData/A2 10.12.2020-14.56.27.txt.
A2 has been configured successfully.
Configuration data has been saved successfully as ./OutputData/A1 10.12.2020-14.56.28.txt.
A1 has been configured successfully.
Configuration data has been saved successfully as ./OutputData/D2 10.12.2020-14.56.36.txt.
D2 has been configured successfully.
Configuration data has been saved successfully as ./OutputData/D1 10.12.2020-14.56.57.txt.
D1 has been configured successfully.
```

Figura 28. Output del programa tras haberse ejecutado correctamente

Y efectivamente, se puede observar como el archivo de configuración es correctamente extraído del dispositivo y guardado en el sistema:

A1 10.12.2020-14.56.28.txt

```
Using 2130 out of 262144 bytes, uncompressed size = 4346 bytes
!
! Last configuration change at 13:52:21 UTC Thu Dec 10 2020 by admin
!
version 15.2
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
service compress-config
!
hostname A1
!
boot-start-marker
boot-end-marker
!
!
enable secret 5 $1$BiiE$q5XCsTThLODAPezNq48RZ.
!
username admin privilege 15 secret 5 $1$ybg3$kwm.JRH04B1ZrO/DW8qrF0
no aaa new-model
!
!
!
!
!
vtp mode off
!
!
!
ip domain-name GNS3Lab
ip cef
no ipv6 cef
!
!
!
spanning-tree mode mst
spanning-tree portfast edge default
spanning-tree portfast edge bpduguard default
spanning-tree extend system-id
```

Figura 29. Archivo de configuración obtenido como Output tras la ejecución del programa

De esta manera, a través de Netmiko, YAML y Jinja2, es posible automatizar el CLI de un dispositivo, en este caso de un sistema Cisco IOS. Por supuesto, este tipo de automatización es solamente recomendable en el caso de que no se puedan usar protocolos basados en YANG, debido a que versiones futuras del CLI pueden ser incompatibles con los "templates" configurados. Otra gran desventaja de este tipo de automatización es que se requiere un conocimiento profundo del CLI a automatizar, debido a que, al fin y al cabo, es equivalente a que el propio usuario emitiera cada uno de los comandos generados por el programa uno por uno, por lo que cualquier tipo de comando que requiera de la activación de otro previo puede dar fallo en la configuración (por ejemplo, un caso dado en la realización de este programa fue declarar una interfaz como tipo "trunk" antes de configurar su encapsulado, lo cual no está permitido en Cisco IOS, y que implicaba por tanto que esta interfaz no quedara configurada correctamente y que la red no funcionase).

Por supuesto, este tipo de automatización trae sus ventajas también, en primer lugar, evita la repetición del uso de comandos en cada CLI, lo cual es engorroso y susceptible a errores; en segundo lugar, una vez se domina Jinja2, es fácil realizar nuevos "templates" si fuera necesario; en tercer lugar, permite proveer de automatización a una interfaz que originalmente no está diseñada para ello; y cuarto, facilita considerablemente la visión holística de la red a configurar, esto es debido a que a través de los ficheros de configuración YAML se puede observar con gran facilidad los parámetros definidos para cada dispositivo, lo cual hace menos complicado la configuración y la resolución de posibles errores.

Por último, se realiza una breve prueba para ver que existe conectividad entre el PC1-VL50 y el PC2-VL70, situado cada uno en una VLAN y "subnet" diferentes. Para ello, se les asignan direcciones IPv4 dinámicamente (el dispositivo D1 está configurado como servidor DHCP para las subnets de las VLAN 50 y 70) y se realiza el ping:

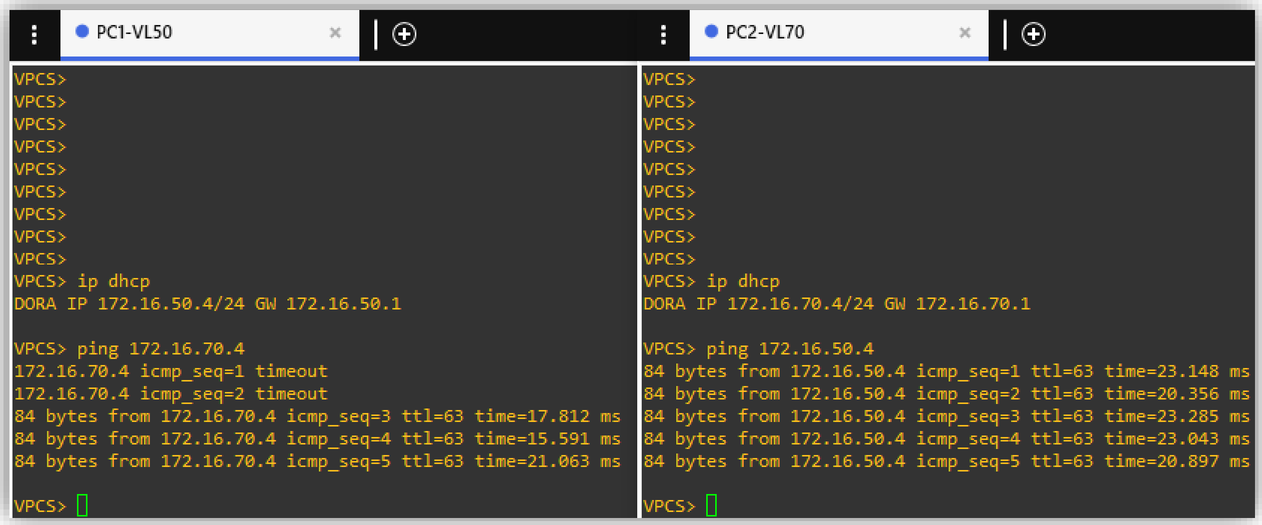

Figura 30. Verificación de conectividad entre dos usuarios tras la configuración de red

Y como se puede observar, la configuración es correcta, por lo que al final, una vez realizado el algoritmo, solamente hace falta diseñar los “templates” correspondientes y rellenar los archivos de configuración y de hosts correspondientes, y en cuestión de segundos se consigue una topología totalmente configurada. Aun así, como se ha comentado anteriormente, es preferible usar protocolos YANG cuando sea posible, por lo que en el siguiente apartado se va a expandir la red con dispositivos compatibles y que sean configurados a través de NETCONF.

4.2 AUTOMATIZACIÓN A TRAVÉS DE NETCONF

En este apartado se procede a expandir la topología mostrada anteriormente, para ello, se usarán dos dispositivos compatibles con NETCONF, uno ejecuta el sistema operativo IOS-XE **[19]** y el otro, Junos **[21]**. Dicho lo cual, al igual que en el apartado anterior, es necesario interaccionar con el CLI de ambos dispositivos, ya que es necesario configurar una interfaz con la que el Manager pueda comunicarse a estos, y, además, es necesario habilitar SSHv2 y NETCONF en cada dispositivo. En el caso del dispositivo IOS-XE, la configuración es similar, ya que usa el mismo CLI que Cisco IOS, mientras que, en el caso de Junos, habrá que aplicar la configuración con comandos compatibles a su propio CLI; la topología final, así como las configuraciones iniciales son las siguientes:

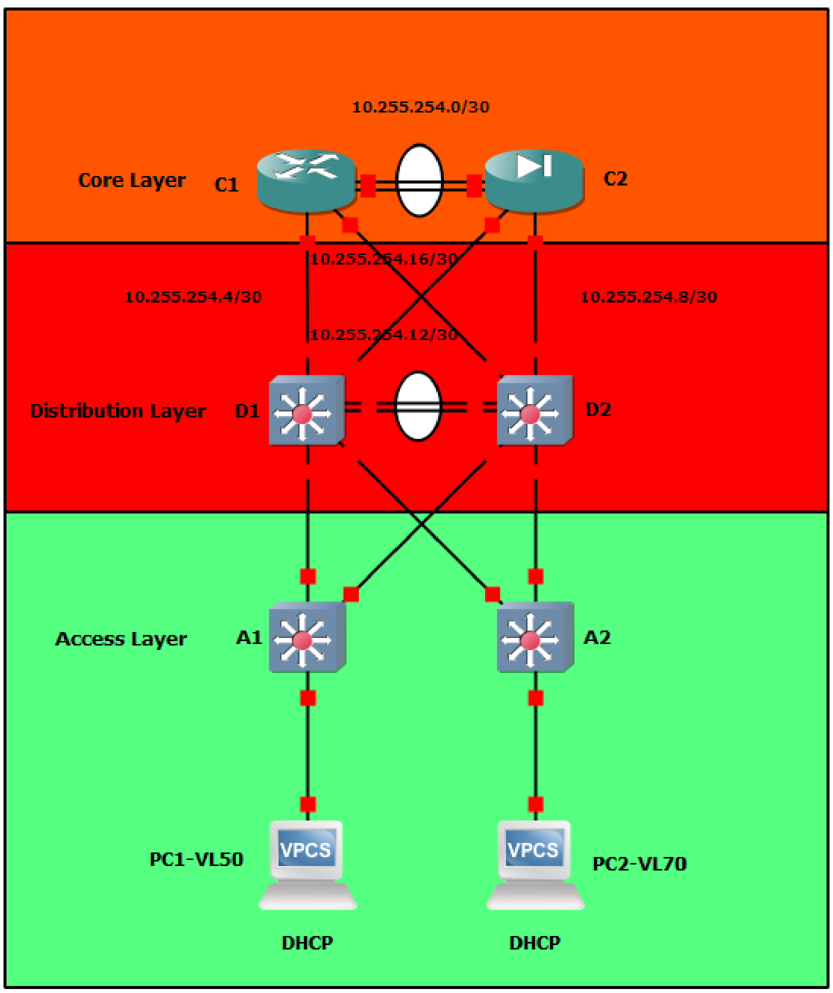

Figura 31. Ilustración de la topología con la capa de núcleo implementada

```
! Hostname will vary depending on the device being configured
hostname C1
ip domain name GNS3LAB
crypto key generate rsa modulus 2048

username admin privilege 15 secret adminpassword

ip ssh version 2

! Starts NETCONF server on the device
netconf-yang
! Enables <candidate> datastore on the device
! This command also disables :writable-running capability
netconf-yang feature candidate-datastore

! Interface on which the manager will connect to the device
interface G9
    ip address dhcp
    no shutdown

line vty 0 4
    transport input ssh
    login local
```

Figura 32. Configuración inicial para habilitar NETCONF en un dispositivo Cisco IOS XE

```
# Sets the system hostname
set system host-name C2
# After issuing the next command, password must be entered and retyped manually
# Plain text passwords are insecure, and thus only being used here for testing purposes.
# If root authentication is not configured, configuration will not be commited.
set system root-authentication plain-text-password

# Sets DHCPv4 on an interface (in this case, fxp0)
set interfaces fxp0 unit 0 family inet dhcp
# Enables SSH and NETCONF through SSH
set system services ssh
set system services netconf ssh
# Enables OpenConfig schemas on the device
set system schema openconfig unhide

# Creates a user named junosauto with absolute privileges.
set system login user junosauto class super-user
# After issuing the next command, password must be entered and retyped manually
set system login user junosauto authentication plain-text-password
```

Figura 33. Configuración inicial para habilitar NETCONF en un dispositivo Juniper

Por otro lado, al implementar la capa de núcleo, es necesario modificar los dispositivos D1 y D2 para dotar de conectividad a las interfaces expuestas a la misma, además de la configuración correspondiente para OSPFv2, ya que es el protocolo de enrutamiento que se pretende usar en esta topología, por lo que entonces habrá que definir un "template" adicional para OSPFv2, el cual se muestra a continuación:

IOS-OSPFv2.jinja2

```
router ospf {{ services.routing.ospfv2.pid }}
    router-id {{ services.routing.ospfv2.rid }}
{% for p_itf in services.routing.ospfv2.passive_interfaces %}
    passive-interface {{ p_itf }}
{% endfor %}

{%- for area in services.routing.ospfv2.areas %}
    {% if area_type != 'standard' %}
    area {{ area.area_number }} {{ area.area_type }}
    {% endif %}

    {% for interface_name, interface_value in area.interfaces.items() %}
    interface {{ interface_name }}
        ip ospf {{ services.routing.ospfv2.pid}} area {{ area.area_number }}
        ip ospf network {{ interface_value.network_type }}
    {% endfor %}
{% endfor -%}
```

Figura 34. "Template" escrito en Jinja2 para configurar OSPFv2 en Cisco IOS

En pocas líneas de código, es posible interpretar los datos de un archivo YAML para obtener una configuración válida para habilitar OSPFv2 en un dispositivo Cisco IOS; la configuración de D1 en específico es la siguiente:

```yaml
routing:
  ospfv2:
    pid: 1
    rid: 100.0.0.1
    # Passive interfaces are defined here to simplify Jinja2 template
    # as they are configured in the ospf section and not on each interface
    passive_interfaces: [vlan 50, vlan 60, vlan 70, vlan 500, vlan 1000]
    areas:
      - area_number: 0.0.0.1
        area_type: stub
        interfaces:
          vlan 50:
            network_type: point-to-point
          vlan 60:
            network_type: point-to-point
          vlan 70:
            network_type: point-to-point
          vlan 500:
            network_type: point-to-point
          vlan 1000:
            network_type: point-to-point
          G1/0:
            network_type: point-to-point
          G1/1:
            network_type: point-to-point
```

Figura 35. Configuración de OSPFv2 para el dispositivo D2

Para verificar que la plantilla definida funciona correctamente, se puede hacer uso de un "parser", en este caso, a través de la herramienta J2Live **[RE2]** utilizada con anterioridad:

```
router ospf 1
  router-id 100.0.0.1
    passive-interface vlan 50
    passive-interface vlan 60
    passive-interface vlan 70
    passive-interface vlan 500
    passive-interface vlan 1000
    area 0.0.0.1 stub

    interface vlan 50
        ip ospf 1 area 0.0.0.1
        ip ospf network point-to-point
    interface vlan 60
        ip ospf 1 area 0.0.0.1
        ip ospf network point-to-point
    interface vlan 70
        ip ospf 1 area 0.0.0.1
        ip ospf network point-to-point
```

Figura 36. Fragmento del output renderizado a partir de los datos de las figuras 34 y 35

En cuanto a las interfaces, solamente es necesario definirlas con el mismo patrón con el que se definieron las demás, y Python generará automáticamente la información pertinente.

Dicho lo cual, se ha conseguido generar una configuración correcta para los dispositivos A1, A2, D1 y D2 en esta nueva topología donde se incluye una capa de núcleo, pero aún quedan por configurar los dispositivos C1 y C2, a los cuales solamente se les ha aplicado aún la configuración inicial (Figuras 32 y 33). Para ello, aunque en el caso del dispositivo Cisco IOS XE (C1), al compartir el mismo CLI que Cisco IOS, se podría automatizar a través de Netmiko, pero también expone modelos YANG a través de NETCONF, por lo que, en esta sección será configurado a través de NETCONF junto con el dispositivo Juniper (C2).

Antes de comenzar, por supuesto, es necesario usar una librería que permita realizar conexiones con el protocolo NETCONF a través de Python, para ello, en este proyecto se hace uso de la librería ncclient **[22]**, la cual es una de las más conocidas y ampliamente utilizada en la industria; su utilización es ciertamente parecida a Netmiko en cuanto al establecimiento de sesión, donde a través de introducir los parámetros correspondientes, el programa realiza automáticamente el intercambio inicial de mensajes <hello>, lo cual había que hacer de forma manual si se usaba un terminal SSH. Aunque tras este establecimiento de sesión, se podría discutir que es donde terminan las similitudes entre ambas librerías, ya que ncclient requiere que el usuario tenga ciertamente conocimientos en el protocolo NETCONF, puesto que ayuda a enviar los <rpc> con las operaciones correspondientes a través de sus propios métodos preestablecidos, pero el responsable de estructurar correctamente los datos útiles, en el caso por ejemplo de configuraciones o de petición de datos con filtros establecidos, es del usuario. Dicho lo cual, en este apartado se procede a configurar los dispositivos C1 y C2 a través de esta librería, así como haciendo uso de YAML y Jinja2, por lo que al final, aunque el procedimiento es parecido al usado anteriormente, ahora los “templates” tienen que dar estructura a archivos XML que sean aceptados por el dispositivo correspondiente.

Para poder configurar estos dispositivos, es necesario saber primero qué modelos soportan cada uno de ellos; hay distintas maneras de consultarlo, por ejemplo, extrayendo la información del propio dispositivo a través de ncclient, pero no es necesario, ya que los fabricantes suben sus modelos a repositorios Github.

De esta forma se pueden acceder a los modelos de casi cualquier dispositivo tanto para Cisco **[23]** como para Juniper **[24]**.

Dentro de un mismo dispositivo se pueden encontrar diferentes tipos de modelos, tomando el caso de C1, cuya sistema operativo es Cisco IOS XE 16.7.1, al entrar en la carpeta correspondiente desde el repositorio **[23]**, se pueden encontrar los siguientes modelos:

- **Modelos nativos:** Son aquellos que son totalmente específicos para el sistema operativo en cuestión, por lo que no ofrece compatibilidad ninguna con dispositivos que ejecuten sistemas operativos diferentes; se pueden identificar fácilmente porque el nombre del archivo comienza con el nombre del sistema al que hace referencia, por ejemplo "Cisco-IOS-XE-interfaces.yang" define las interfaces en IOS XE exclusivamente.

- **Modelos estandarizados:** Referido a aquellos creados por una institución cuyos estándares son aceptados por la industria, en este caso, estos modelos definen diferentes componentes de un dispositivo de forma neutral, por ejemplo "ietf-interfaces.yang" podría usarse como modelo para configurar las interfaces en dispositivos totalmente diferentes, si ambos lo tienen implementado.

- **Desviaciones (Deviations):** Son aquellos modelos que implementan otro, ya sea nativo o estandarizado, pero que incluye modificaciones particulares del vendedor, o cuya implementación no sea totalmente idéntica a la definida en el modelo original, por lo tanto, al igual que en los modelos nativos, su aplicación se limita al sistema operativo para el que ha sido definido; un ejemplo podría ser "cisco-xe-openconfig-vlan-deviation.yang".

- **Modelos OpenConfig:** Este tipo de modelos están definidos por OpenConfig, un grupo de trabajo formado por distintas instituciones y multinacionales cuyo objetivo es proporcionar modelos multiplataforma; la diferencia entre los modelos estandarizados y los modelos OpenConfig es que estos últimos son actualizados con mucha más frecuencia y permiten realizar configuraciones mucho más avanzadas, por lo que, aunque oficialmente no sean modelos estándares, es cierto la industria los está implementado como tales, aun así, su implementación es limitada hoy en día, y no existe tanta documentación como en el caso de los modelos nativos.

En resumidas, los modelos estandarizados son extremadamente limitados y no permiten realizar configuraciones avanzadas, por lo que, dentro de lo posible, y siempre que el dispositivo lo permita, es recomendable usar OpenConfig, pero es cierto que muchos de ellos implementan versiones antiguas que no soportan todas las capacidades que se pueden observar en los modelos actuales, los cuales se pueden consultar en su repositorio Github **[10]**. Así que, en muchas ocasiones, no queda otra opción que utilizar los modelos nativos, que son los que definen de forma más completa un dispositivo, en detrimento de la compatibilidad entre distintos sistemas operativos.

Tras esta explicación, cabe decir que, en este proyecto, se han utilizado principalmente modelos nativos, debido a las implementaciones limitadas de OpenConfig en los dispositivos C1 y C2 (aunque de forma más acuciada en C1); aun así, se hace uso del modelo “OpenConfig-Interfaces.yang” para configurar las interfaces de ambos dispositivos ya que este si está totalmente integrado en ambos. Para ello primero es necesario diseñar un “template” que permita crear configuraciones siguiendo el modelo OpenConfig para las interfaces; una forma de facilitar la lectura del modelo es verlo en formato árbol, lo cual es sencillo de realizar usando la herramienta pyang **[25]**:

```
$ pyang -f tree openconfig-interfaces.yang
module: openconfig-interfaces
  +--rw interfaces
     +--rw interface* [name]
        +--rw name                -> ../config/name
        +--rw config
        |  +--rw name?            string
        |  +--rw type             identityref
        |  +--rw mtu?             uint16
        |  +--rw loopback-mode?   boolean
        |  +--rw description?     string
        |  +--rw enabled?         boolean
```

Figura 37. Formato "tree" de una parte del modelo "openconfig-interfaces"

De la vista del modelo, se puede inferir qué no existe una sección que permita definir el direccionamiento de capa 3; esto es debido a que OpenConfig realiza un diseño totalmente modular de sus modelos, por lo que hay que hacer uso de otros modelos para poder configurar IPv4 en las interfaces. En la siguiente figura se muestra una parte del modelo "openconfig-if-ip":

```
$ pyang -f tree openconfig-if-ip.yang

module: openconfig-if-ip
  augment /oc-if:interfaces/oc-if:interface/oc-if:subinterfaces/oc-if:subinterface:
    +--rw ipv4
       +--rw addresses
       |  +--rw address* [ip]
       |     +--rw ip        -> ../config/ip
       |     +--rw config
       |     |  +--rw ip?              oc-inet:ipv4-address
       |     |  +--rw prefix-length?   uint8
```

Figura 38. Formato "tree" de parte del modelo "openconfig-if-ip"

Se puede ver que este modelo permite la configuración del direccionamiento IP a una interfaz, pero para situar correctamente el "schema" definido, es necesario ver que este modelo realiza un "augment" a OpenConfig-Interfaces (cuyo prefijo, oc-if es fácilmente deducible, pero que se puede verificar en los propios archivos de ambos modelos; en cierto modo es similar a comparar "openconfig-interfaces" con una superclase y "openconfig-if-ip" como una subclase, que extiende la funcionalidad del modelo de la superclase). Por tanto, con estos dos esquemas, ya se puede crear un "template" en jinja2, y aportar los datos pertinentes a través de YAML (todos los archivos mostrados están en el fichero Codigo_TFG.zip)

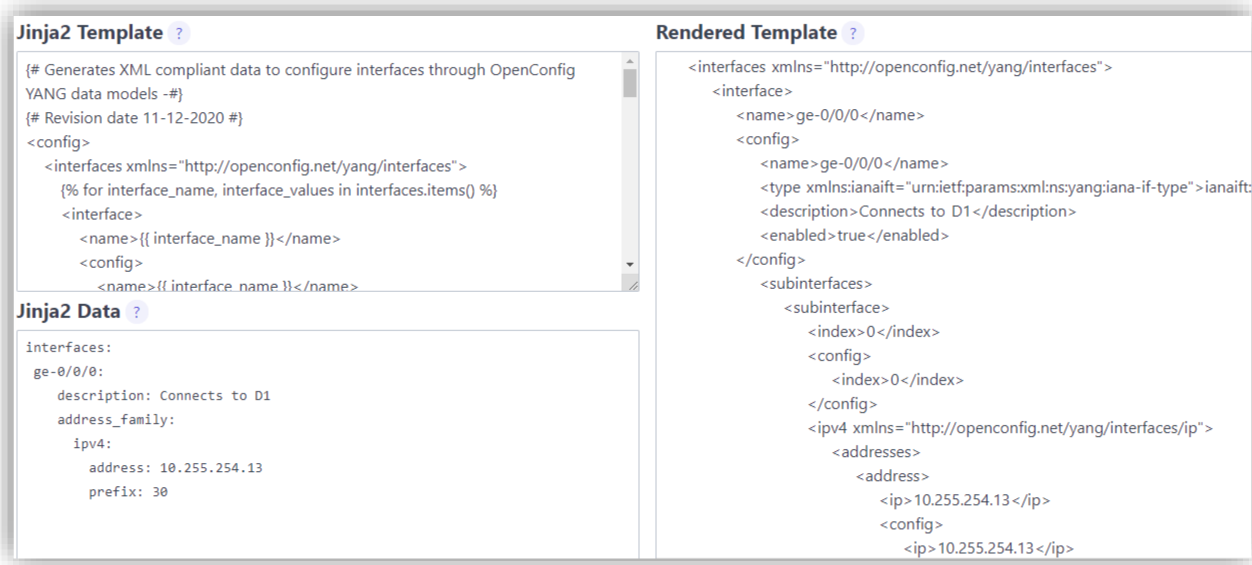

Figura 39. Output renderizado a partir de un "template" y datos codificados en YAML en J2Live [RE2]

También es conveniente tener en cuenta que el modelo define de forma muy concreta algunos campos, como por ejemplo <type>, cuyos posibles tipos están descritos en el modelo estándar "iana-if-type"; para adecuarse a este tipo de requerimientos, sí se hace necesario consultar el archivo del propio modelo, o también ver ejemplos de aplicación de este. La metodología aplicada es similar para los "templates" diseñados para los modelos nativos, aunque algunos son tan complejos que se hace inviable trabajar exclusivamente con el output de pyang, por lo que, en tal caso, es recomendable ver ejemplos de aplicación tanto en la web, o realizar una configuración previa en los dispositivos, y después extraer el output a través de ncclient en el caso de IOS XE, o a través del propio CLI en el caso de Junos, ya que este último estructura directamente su configuración siguiendo su modelo nativo en JSON, pero también se puede obtener el output en XML:

Figura 40. Output de datos de configuración en XML en Junos a través de su CLI

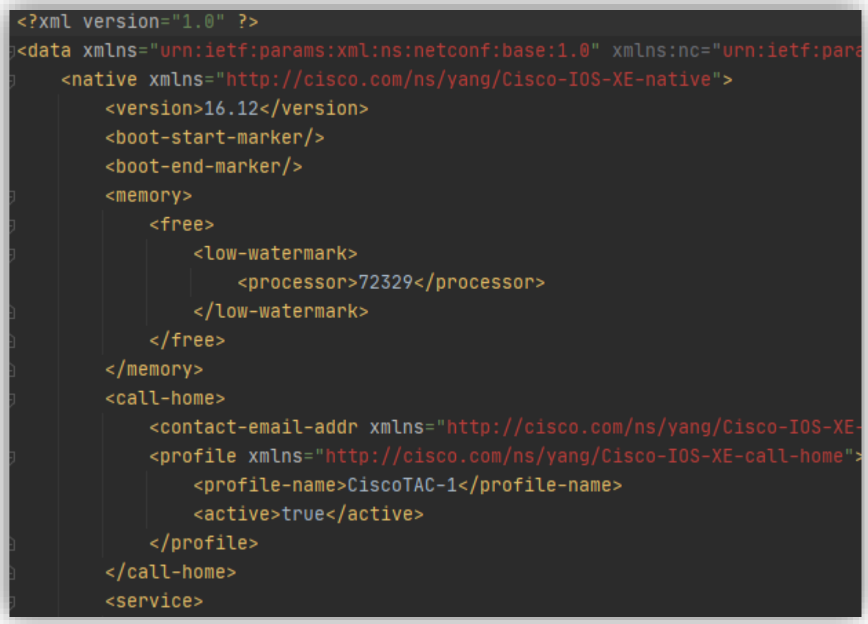

Figura 41. Output de datos de configuración en XML en IOS XE a través de ncclient

Una vez realizados los “templates”, y con una estructura de datos en YAML similar al código utilizado para Netmiko, se puede automatizar la configuración de los dispositivos. Es necesario tener en cuenta que se ha separado la configuración en distintos segmentos, ya que NETCONF no permitirá aplicar ciertas configuraciones sin pasos previos, como es el caso de asociar interfaces a una interfaz virtual para EtherChannel en IOS XE sin haberse definido esta primero, por lo que se realizan las configuraciones siguiendo un orden lógico, y además facilita la resolución de problemas si la configuración falla, al dividirla en distintos <rpc>. Entonces, solamente queda iniciar cada uno de los programas, el que usa Netmiko, y el segundo con ncclient, para configurar toda la red:

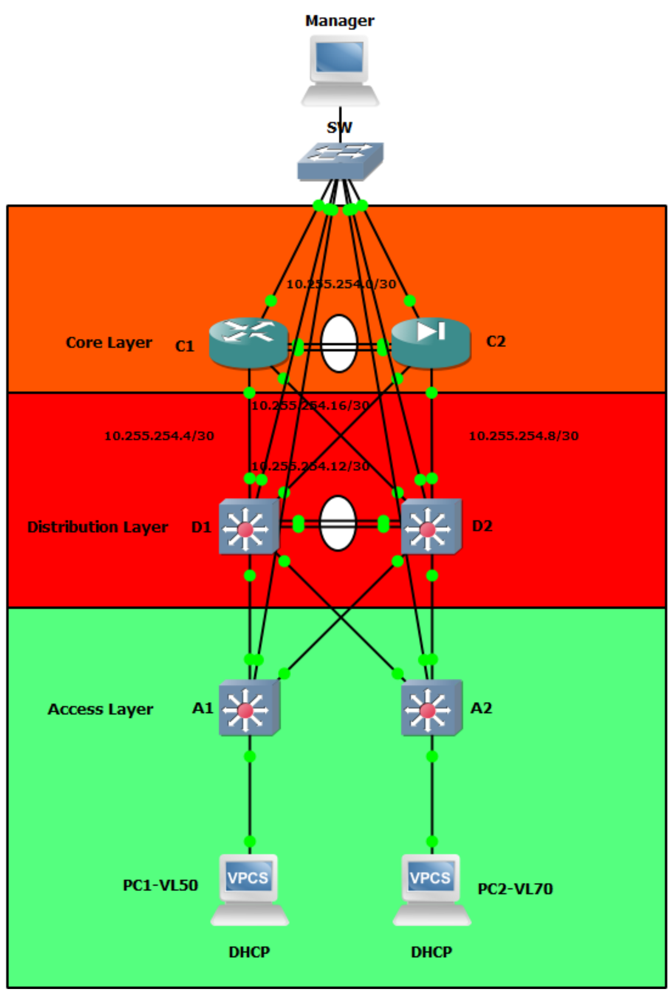

Figura 42. Topología completa con el manager conectado a todos los dispositivos

Se aplica la configuración inicial en cada dispositivo (Figuras 19, 32 y 33), se configuran los archivos "host.yaml" de cada proyecto y se inician; puede darse el caso de que ncclient devuelva error de "timeout" debido a que el servidor tarde en procesar algunos <rpc>:

```
    raise TimeoutExpiredError('ncclient timed out while waiting for an rpc reply.')
ncclient.operations.errors.TimeoutExpiredError: ncclient timed out while waiting for an rpc reply.
```

Figura 43. Error de "timeout" en ncclient

Para evitar en la medida de lo posible este error, es conveniente aumentar el "timeout" en ncclient (el valor predeterminado es 30s):

```
host_data = {'device_params': {'name': device['device_type']},
             'host': device['ip'],
             'port': 830,
             'username': device['ssh_user'],
             'password': device['ssh_password'],
             'hostkey_verify': False,
             'manager_params': {"timeout": 90}}
```

Figura 44. Timeout aumentado a 90 segundos en ncclient

Una vez terminada la ejecución de ambos programas, se obtienen los siguientes mensajes indicando el nombre de los archivos de configuración para cada dispositivo:

```
Configuration data for C1 has been saved successfully as ./OutputData/C1/C1 12.12.2020-21.24.41.xml.
Configuration data for C2 has been saved successfully as ./OutputData/C2/C2 12.12.2020-21.25.57.xml.

Process finished with exit code 0
```

```
Configuration data for A2 has been saved successfully at ./OutputData/A2 12.12.2020-21.26.17.txt.
A2 has been configured successfully.
Configuration data for A1 has been saved successfully at ./OutputData/A1 12.12.2020-21.26.17.txt.
A1 has been configured successfully.
Configuration data for D2 has been saved successfully at ./OutputData/D2 12.12.2020-21.26.40.txt.
D2 has been configured successfully.
Configuration data for D1 has been saved successfully at ./OutputData/D1 12.12.2020-21.26.58.txt.
D1 has been configured successfully.

Process finished with exit code 0
```

Figura 45. Configuración completada de forma simultánea a través de ncclient y Netmiko

Tras ello, se puede observar la configuración obtenida de cada uno; cabe decir que en el caso de ncclient, la información obtenida en ambos dispositivos es la almacenada en <running>, pero el funcionamiento es diferente entre IOS XE y Junos. En el caso de Junos, la información se guarda en el dispositivo al hacer <commit>, mientras que en IOS XE es necesario enviar un <rpc> equivalente a emitir "write memory" en el CLI, ya que este no tiene entre sus capacidades aquella que define <startup> y no funciona con un sistema de commit, como en el caso de Junos. A continuación, se muestran pequeñas secciones de los archivos de configuración para C1 y D1 respectivamente:

C1 12.12.2020-21.24.41.xml


```xml
<?xml version="1.0" ?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:param
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
        <version>16.12</version>
        <boot-start-marker/>
        <boot-end-marker/>
        <memory>
            <free>
                <low-watermark>
                    <processor>72329</processor>
                </low-watermark>
            </free>
        </memory>
        <call-home>
            <contact-email-addr xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-c
            <profile xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-call-home">
                <profile-name>CiscoTAC-1</profile-name>
                <active>true</active>
            </profile>
        </call-home>
        <service>
            <timestamps>
```


Figura 46. Parte del archivo de configuración obtenido de C1

D1 12.12.2020-21.26.58.txt

```
ip dhcp pool DHCP_VL70
 network 172.16.70.0 255.255.255.0
 default-router 172.16.70.1
 dns-server 172.16.70.1
!
!
ip domain-name GNS3Lab
ip cef
no ipv6 cef
!
!
!
spanning-tree mode mst
spanning-tree portfast edge default
spanning-tree portfast edge bpduguard default
spanning-tree extend system-id
!
spanning-tree mst configuration
 name MST-GNS3
 revision 1
 instance 1 vlan 50, 60, 70
```

Figura 47. Parte del archivo de configuración obtenido de D1

Adicionalmente, se realizan diferentes comprobaciones para verificar que la configuración se ha aplicado correctamente a todos los dispositivos, como el correcto funcionamiento de LACP entre C1 y C2:

```
C1>show etherchannel summary
Flags:  D - down        P/bndl - bundled in port-channel
        I - stand-alone s/susp - suspended
        H - Hot-standby (LACP only)
        R - Layer3      S - Layer2
        U - in use      f - failed to allocate aggregator

        M - not in use, minimum links not met
        u - unsuitable for bundling
        w - waiting to be aggregated
        d - default port


Number of channel-groups in use: 1
Number of aggregators:           1

Group  Port-channel  Protocol    Ports
------+-------------+-----------+-----------------------
1       Po1(RU)         LACP     Gi6(bndl) Gi7(bndl)
```

Figura 48. Verificación del etherchannel entre C1 y C2

Se puede observar que el funcionamiento es correcto, y es que, aunque sean sistemas diferentes, el protocolo estandarizado LACP permite realizar un "etherchannel" de forma sencilla. También se procede a verificar que el enrutamiento funciona correctamente, para ello se muestra la tabla de D2:

```
D2#show ip ro
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR

Gateway of last resort is 10.255.254.17 to network 0.0.0.0

O*IA  0.0.0.0/0 [110/2] via 10.255.254.17, 00:03:28, GigabitEthernet1/1
      10.0.0.0/8 is variably subnetted, 7 subnets, 2 masks
O IA     10.255.254.0/30
           [110/2] via 10.255.254.17, 00:03:28, GigabitEthernet1/1
           [110/2] via 10.255.254.9, 00:03:18, GigabitEthernet1/0
O        10.255.254.4/30
           [110/2] via 10.255.254.17, 00:03:28, GigabitEthernet1/1
C        10.255.254.8/30 is directly connected, GigabitEthernet1/0
L        10.255.254.10/32 is directly connected, GigabitEthernet1/0
O        10.255.254.12/30
           [110/2] via 10.255.254.9, 00:03:18, GigabitEthernet1/0
```

Figura 49. Porción de la table de enrutamiento de D2

Se puede observar que D2 ha aprendido las subnets correctamente, y también se puede comprobar que el área 0.0.0.1, en la que se sitúa este dispositivo está correctamente definida como “stub”, ya que define como salto predeterminado la interfaz que conecta con el área 0.0.0.0 (OSPF destina determina como destino predeterminado área 0.0.0.0 para aquellos enrutadores situados en un área “stub” **[26]**).

Los dispositivos finales también pueden comunicarse entre ellos, e incluso con la red definida entre el “etherchannel” entre C1 y C2:

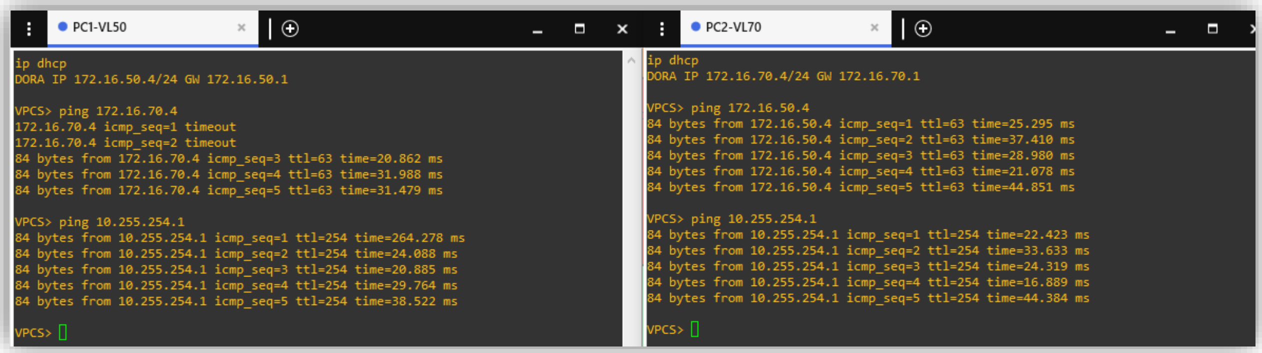

Figura 50. Comprobación de ping entre las VLAN 50 y 70; y la subred 10.255.254.0/30

Como se puede comprobar, los pings muestran que existe conectividad desde la capa de distribución hasta la capa de núcleo, así como interconectividad entre las VLAN 50 y 70, por lo que, una vez realizada la codificación del programa y los “templates”, crear una configuración nueva es tan sencillo como definir los parámetros en archivos YAML y ejecutar los programas, además, el uso de modelos YANG permite que la interacción con el CLI se limite a habilitar el servicio SSH y NETCONF, aunque no sin sus complicaciones, ya que es necesario investigar los modelos a usar, y, dependiendo del fabricante y la versión del dispositivo usado, puede que la compatibilidad con OpenConfig sea limitada, lo cual es bastante común exceptuando en los dispositivos más modernos y orientados totalmente a la programación, como puede ser Cisco NX-OS **[27]**, el cual dispone de una guía muy práctica para generar configuraciones usando los modelos OpenConfig **[28]**.

4.3 AUTOMATIZACIÓN A TRAVÉS DE RESTCONF

Aunque en los apartados 4.1 y 4.2 se ha conseguido configurar completamente la topología mostrada a través de automatizar el CLI y usar NETCONF, existe la posibilidad de automatizar la configuración utilizando RESTCONF, para ello, existen diferentes alternativas, como es el caso de usar aplicaciones especializadas en peticiones HTTP, como es el caso de curl y Postman, pero también es posible realizar estas peticiones a través de Python, para ello, se hace uso de una librería desarrollada para ello, llamada "requests" **[29]**. Dicho lo cual, en esta sección se va a configurar una topología pequeña con el protocolo de enrutamiento BGP.

En los casos anteriores, se ha usado el lenguaje Jinja2 como base para formar los distintos "payloads", y si bien se podría realizar de nuevo en este caso, RESTCONF soporta JSON, además de XML, por lo que la propuesta que se hace es un poco distinta en este caso, la cual consiste en lo siguiente en casos de configuración:

- Desarrollar archivos de configuración YAML que sigan la estructura de un modelo YANG.

- Transformar directamente el contenido del archivo YAML a JSON, dando como resultado un archivo de configuración válido para ser transmitido por RESTCONF (esto no es posible en NETCONF ya que XML requiere del uso de "namespaces", los cuales no son compatibles en JSON ni YAML).

Esto facilita mucho la configuración, ya que es posible desarrollar archivos YAML directamente a través de la estructura del modelo YANG, como es el caso de la figuras 6 y 7; además, en caso de obtener información de un dispositivo, se puede transformar esta información de JSON a YAML, por lo que la lectura de los datos es mucho más sencilla. Dada la explicación, la topología en GNS3 es la que se muestra a continuación:

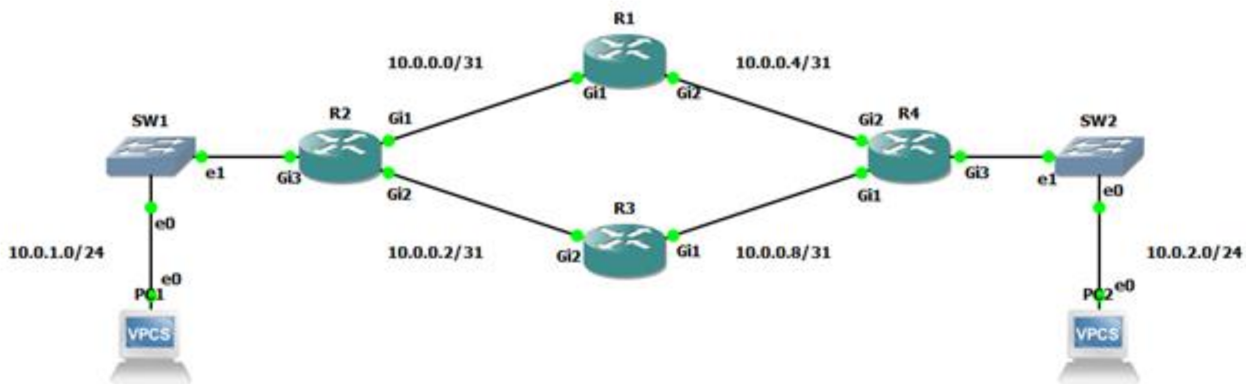

Figura 51. Topología de red EIGRP

La topología mostrada está formada por "routers" que ejecutan IOS XE, y se va a configurar a través de RESTCONF las interfaces y el protocolo de enrutamiento interno, que en este caso será EIGRP; a los dispositivos finales se les asignarán la primera dirección IP asignable de forma manual.

Para configurar las interfaces, se usará el modelo OpenConfig, idéntico al de la sección anterior, pero en este caso, transformando el formato del "payload" a JSON; por otro lado, para configurar EIGRP, se usará el modelo nativo para IOS XE. Primero, es necesario dotar de conectividad básica a todos los dispositivos, para ello se aplica una configuración idéntica a la de la figura 32, pero añadiendo en el modo de configuración el comando "restconf" (sin comillas), de esta manera queda habilitado y se pueden realizar las peticiones.

Para determinar la estructura de la configuración de EIGRP en YANG, se ha optado por realizar la configuración en un "router" de forma manual primero, y posteriormente extraer la configuración a través de Postman, de esta manera, se pueden crear los archivos YAML correspondientes para los demás dispositivos y cambiar la configuración a través de Python. Por último, los dispositivos finales usarán la segunda dirección asignable de sus respectivas "subnets"; se considerará que la topología está correctamente configurada si estos logran comunicarse a través de ping.

En la siguiente ilustración se muestra la configuración de la interfaz GigabitEthernet2 para el dispositivo R1:


```yaml
'openconfig-interfaces:interfaces':
  interface:
  - name: GigabitEthernet2
    config:
      name: GigabitEthernet2
      type: iana-if-type:ethernetCsmacd
      description:
      enabled: true
    subinterfaces:
      subinterface:
        - index: 0
          config:
            index: 0
          openconfig-if-ip:ipv4:
            addresses:
              address:
                - ip: 10.0.0.4
                  config:
                    ip: 10.0.0.4
                    prefix-length: 31
```


Figura 52. Configuración de la interfaz GigabitEthernet2 para R1 en YAML

Como se puede observar, la estructura recuerda a la de la figura 7, y es que, al contrario que en las secciones 4.1 y 4.2, estos documentos siguen a la perfección la estructura de los modelos YANG a los que hacen referencia, por lo que su transformación a JSON a través de Python proporciona un "payload" válido para usar a través de RESTCONF. La topología con el host conectado es la siguiente:

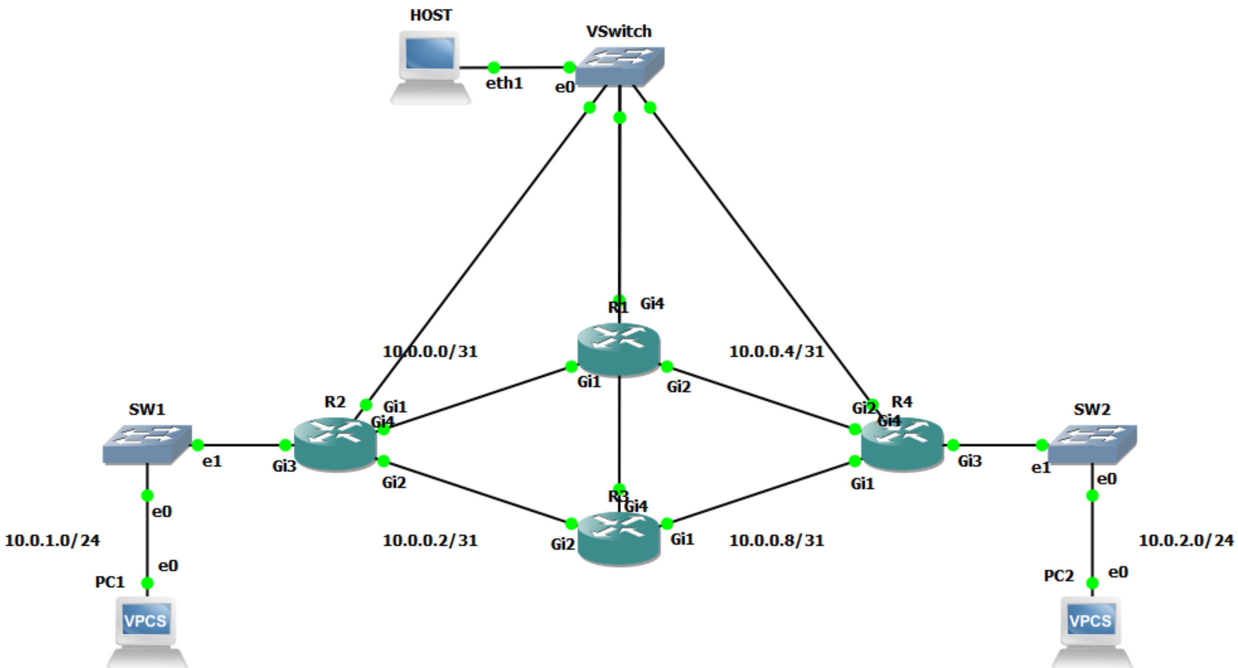

Figura 53. Topología de red EIGRP con el host conectado a los enrutadores

Tras ello, solo queda ejecutar el algoritmo. Algo a tener en cuenta es que, como estos dispositivos no tienen una configuración previa aparte de la inicial, se puede usar el método PATCH, que funciona efectivamente como "merge", sin embargo, si existiese una configuración anterior con los elementos incluidos en el "payload", sería necesario usar los métodos PUT o DELETE; el problema de estos métodos es que eliminan todo lo que exista en el "payload", por ejemplo, si se pretende reemplazar una configuración a través de PUT, habría que identificar de forma mucho más específica la dirección donde se encuentra esta configuración, por ejemplo:

- Se configura una interfaz a través de PATCH; como PATCH funciona como "merge", solamente modifica la configuración que no exista en el dispositivo a modificar. Esto permite flexibilidad, ya que se puede hacer referencia a un campo desde distintas perspectivas.

- Si por alguna razón se desea modificar esta configuración, PATCH no funcionaría, ya que hay una configuración existente previa, por lo que habría que identificar el campo específico que se pretende modificar y usar PUT; como PUT reemplaza el campo totalmente por lo que haya en el "payload", si la perspectiva es demasiado amplia, la configuración quedaría inconsistente y el dispositivo la rechaza. Esto es igualmente aplicable a DELETE.

En el programa desarrollado para esta sección, se usa el método PATCH, y está configurado para informar si la transacción PATCH de cada archivo falla o es válida, en este caso, el resultado obtenido para la configuración de la topología es el siguiente:

```
Transaction for Interfaces/R1-itf.yaml file applied successfully.
Transaction for Interfaces/R4-itf.yaml file applied successfully.
Transaction for Interfaces/R2-itf.yaml file applied successfully.
Transaction for Interfaces/R3-itf.yaml file applied successfully.
Transaction for Routing/R1-eigrp.yaml file applied successfully.
Transaction for Routing/R4-eigrp.yaml file applied successfully.
Transaction for Routing/R2-eigrp.yaml file applied successfully.
Transaction for Routing/R3-eigrp.yaml file applied successfully.

Process finished with exit code 0
```

Figura 54. Output del programa tras ejecutar las transacciones PATCH

Algo importante observado en comparación con el uso de NETCONF y netmiko es que las transacciones se realizan de forma casi inmediata en RESTCONF; aunque en el caso de NETCONF y RESTCONF puede deberse a que los protocolos REST son muy ágiles y al menor "overhead" que supone un "payload" en JSON que, en XML, la razón exacta queda incierta, pero, en el caso de Netmiko, se puede sospechar que aplicar cada línea de comando puede suponer una mayor lentitud que el uso de protocolos como RESTCONF.

A continuación, se muestra la prueba de conectividad entre el PC1 y el PC2, así como la tabla de enrutamiento de R1 para mostrar que EIGRP está funcionando correctamente:

```
PC1> ip 10.0.1.2/24 10.0.1.1
Checking for duplicate address...
PC1 : 10.0.1.2 255.255.255.0 gateway 10.0.1.1
```

```
PC2> ip 10.0.2.2/24 10.0.2.1
Checking for duplicate address...
PC1 : 10.0.2.2 255.255.255.0 gateway 10.0.2.1
```

Figura 55. Asignación manual de direcciones IP a PC1 y PC2

```
PC1> ping 10.0.2.2
10.0.2.2 icmp_seq=1 timeout
84 bytes from 10.0.2.2 icmp_seq=2 ttl=61 time=3.224 ms
84 bytes from 10.0.2.2 icmp_seq=3 ttl=61 time=2.843 ms
84 bytes from 10.0.2.2 icmp_seq=4 ttl=61 time=3.322 ms
84 bytes from 10.0.2.2 icmp_seq=5 ttl=61 time=2.150 ms
```

Figura 56. Ping desde PC1 a PC2 para verificar la conectividad

```
R1#show ip route | section D
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       n - NAT, Ni - NAT inside, No - NAT outside, Nd - NAT DIA
       o - ODR, P - periodic downloaded static route, l - LISP
D         10.0.0.2/31 [90/3072] via 10.0.0.0, 00:00:27, GigabitEthernet1
D         10.0.0.8/31 [90/3072] via 10.0.0.5, 00:00:24, GigabitEthernet2
D         10.0.1.0/24 [90/3072] via 10.0.0.0, 00:00:22, GigabitEthernet1
D         10.0.2.0/24 [90/3072] via 10.0.0.5, 00:00:27, GigabitEthernet2
```

Figura 57. Sección de la table de enrutamiento R1 mostrando las entradas de EIGRP

De esta manera, se ha podido configurar una topología de red de forma automática; una vez desarrollado el algoritmo, solo hace falta configurar los archivos YAML correspondientes, lo cual ayuda si existen ejemplos previos, además una ventaja de ello es que se pueden instanciar modelos YANG en YAML, lo cual simplifica bastante la creación de estos archivos, ya que se pueden usar guías como la de NX-OS **[27]** que ofrece multitud de ejemplos con modelos OpenConfig.

Para culminar esta sección, se va a mostrar cómo modificar a través de Postman y la petición PUT una configuración existente, por ejemplo, en R1 se va a configurar la interfaz GigabitEthernet3 que no está siendo utilizada en la topología. Lo primero es configurar una dirección IP, lo cual se puede hacer a través de PATCH, para ello, se puede codificar la configuración en YAML, y transformarla en JSON:


```json
{
  "openconfig-interfaces:interfaces": {
    "interface": [
      {
        "name": "GigabitEthernet3",
        "config": {
          "name": "GigabitEthernet3",
          "type": "iana-if-type:ethernetCsmacd",
          "description": null,
          "enabled": true
        },
        "subinterfaces": {
          "subinterface": [
            {
              "index": 0,
              "config": {
                "index": 0
              },
              "openconfig-if-ip:ipv4": {
                "addresses": {
                  "address": [
                    {
                      "ip": "55.55.55.55",
                      "config": {
                        "ip": "55.55.55.55",
                        "prefix-length": 8
                      }
                    }
                  ]
                }
              }
            }
          ]
        }
      }
    ]
  }
}
```

```yaml
'openconfig-interfaces:interfaces':
  interface:
  - name: GigabitEthernet3
    config:
      name: GigabitEthernet3
      type: iana-if-type:ethernetCsmacd
      description:
      enabled: true
    subinterfaces:
      subinterface:
        - index: 0
          config:
            index: 0
          openconfig-if-ip:ipv4:
            addresses:
              address:
                - ip: 55.55.55.55
                  config:
                    ip: 55.55.55.55
                    prefix-length: 8
```


Figura 58. Transformación de datos en YAML a JSON con json2yaml [RE4]

Se puede observar a simple vista que haber escrito directamente el documento en JSON hubiese sido una complicación innecesaria; ahora se realiza la petición PATCH a través de Postman y copiando el contenido de JSON en el apartado “body” y seleccionando como tipo de datos “raw” y JSON:

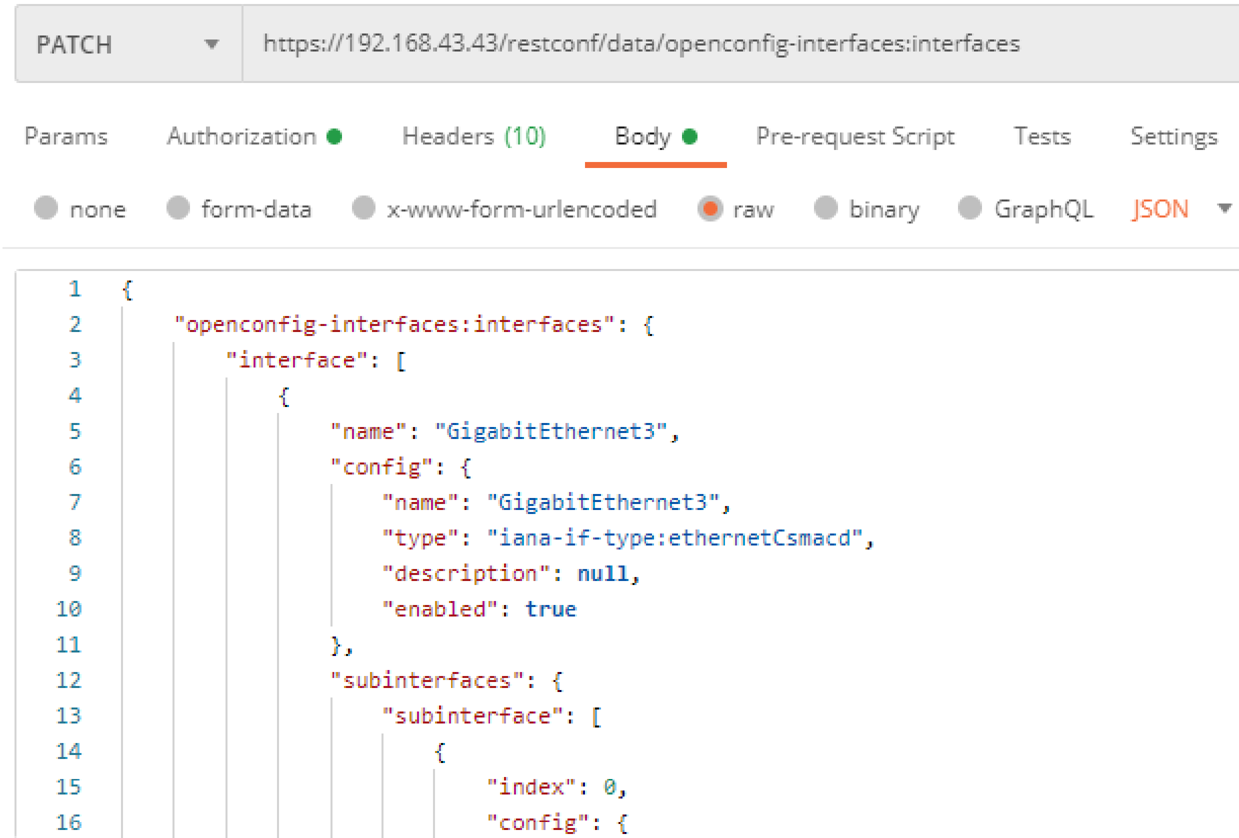

Figura 59. Transacción PATCH de R1 a través de Postman

El resultado se muestra correctamente en el dispositivo:

```
R1#show ip interface brief | section GigabitEthernet3
GigabitEthernet3        55.55.55.55     YES other  down                    down
```

Figura 60. Verificación de direccionamiento de la interfaz GigabitEthernet3 en el CLI

A partir de este momento, no se puede realizar otra asignación de dirección IPv4 a esta interfaz a través de PATCH, ya que existe una configuración existente y por tanto la transacción daría lugar a otra configuración de direccionamiento que en ningún caso surtiría efecto. Dicho lo cual, se puede solucionar de dos formas, o se hace DELETE y un posterior PUT/PATCH, o directamente se realiza PUT si lo que se desea es asignar otra dirección a la interfaz, que es lo que se mostrará en este documento.

Como se explicó anteriormente, la interfaz GigabitEthernet3 es física, y por tanto no se puede borrar ni alterar algunos de sus elementos, si se realiza un PUT con el "payload" utilizado anteriormente pero cambiando solamente la información pertinente al direccionamiento, sería equivalente a borrar todos los campos que no apareciesen en el archivo JSON, tras lo que directamente el dispositivo rechazaría, ya que existe más información pertinente a la interfaz, como su MAC, pero como la operación PATCH realiza un "merge", esta no se toca, pero en el caso de PUT, la borraría si no se incluye en los datos útiles.

Una solución por tanto sería incluir todos los campos que falten, y aun así eso no asegura que la petición sea permitida, por lo que para ello va a ser necesario incluir solamente el campo a modificar, lo cual se puede visualizar fácilmente a través de la siguientes figuras:

La URL utilizada para modificar la configuración de la interfaz con PATCH:

```
https://192.168.43.43/restconf/data/openconfig-interfaces:interfaces
```

Figura 61. URL utilizada en las transacciones PATCH

La URL que hace referencia exclusivamente al direccionamiento IPv4 en la subinterfaz 0:

```
https://192.168.43.202/restconf/data/openconfig-interfaces:interfaces/interface=GigabitEthernet1/subinterfaces/subinterface=0/openconfig-if-ip:ipv4
```

Figura 62. URL más específica para las transacciones PUT y DELETE

Para cambiar el direccionamiento, habrá que usar la URL que especifica de forma exacta a su sección correspondiente, tras lo cual, también habrá que adaptar el "payload" para que solo haga referencia a este campo:

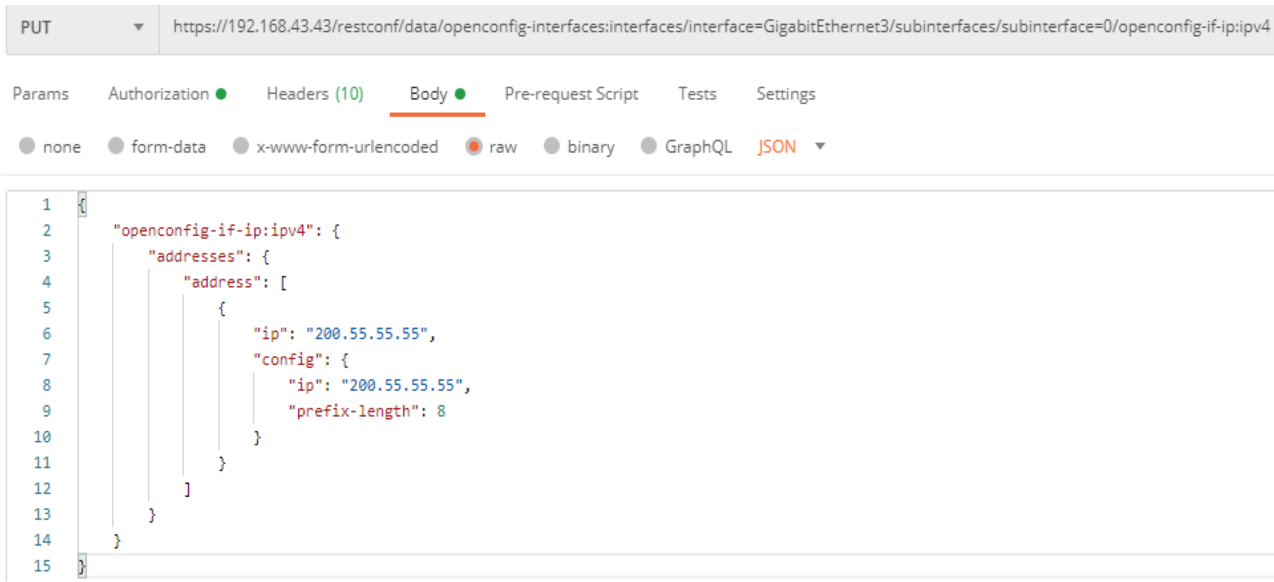

Figura 63. Petición PUT a través de Postman

Como se puede observar, a medida que se hace más específica la URL, el archivo JSON también se simplfica, pues solo tiene que hacer referencia al campo a modificar, y el resultado es el siguiente:

```
R1>show ip interface brief | section GigabitEthernet3
GigabitEthernet3       200.55.55.55    YES other  down                  down
```

Figura 64. Verificación de direccionamiento de la interfaz GigabitEthernet3 tras petición PUT

Aunque también se puede observar a través de Postman:

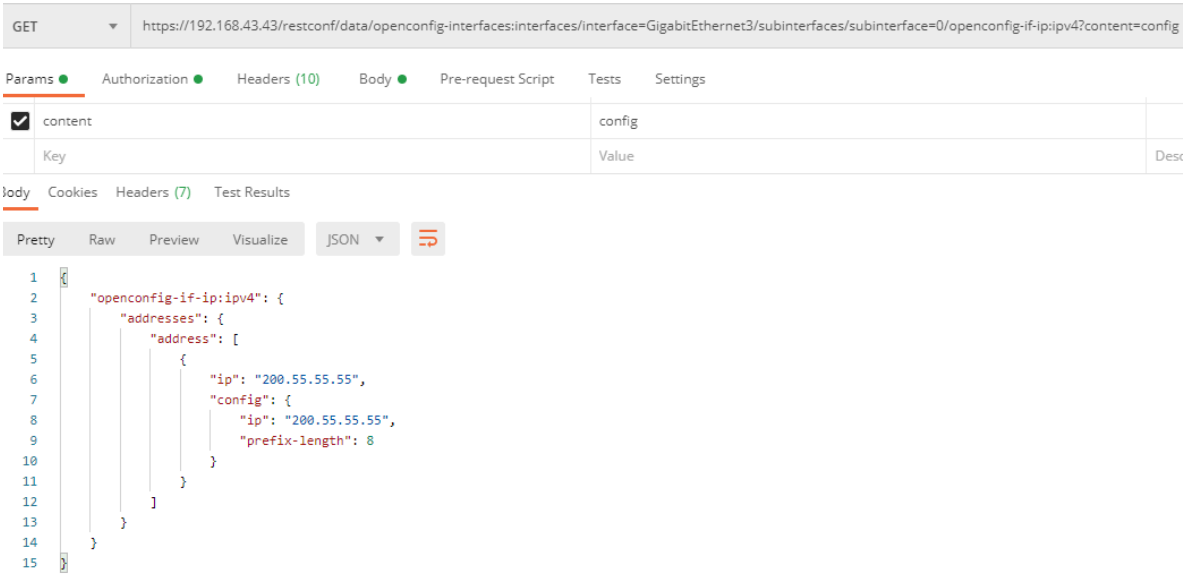

Figura 65. Verificación del direccionamiento de GigabitEthernet3 a través de petición GET

Y, por último, se puede borrar el direccionamiento a través de DELETE:

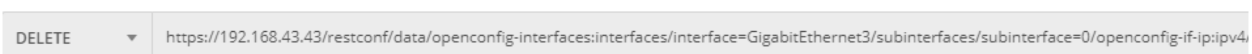

Figura 66. Petición DELETE a través de Postman

Como se puede observar, ha borrado toda la sección pertinente a “openconfig-if-ip:ipv4”:

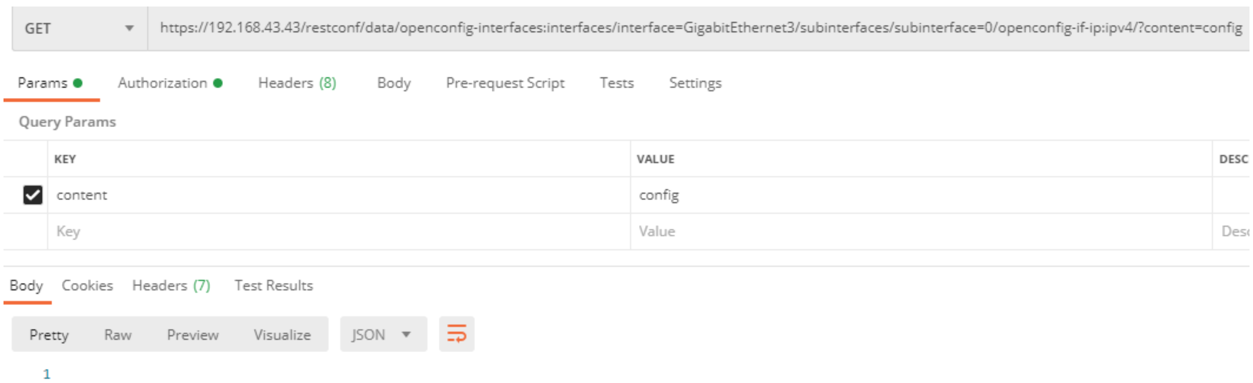

Figura 67. Verificación del direccionamiento de GigabitEthernet3 posterior a DELETE en Postman

Y el propio sistema operativo muestra que la interfaz ya no tiene ninguna dirección:

```
R1>show ip interface brief | section GigabitEthernet3
GigabitEthernet3       unassigned      YES unset  down
```

Figura 68. Verificación del direccionamiento de GigabitEthernet3 posterior a DELETE en el CLI

Tras ello, sería posible asignar otra dirección IP a través de PATCH o PUT de cualquiera de las maneras que se ha mostrado anteriormente; con PATCH también se podría haber hecho una transacción más específica, pero la ventaja de este método es que como solamente combina la información que no se encuentre en el dispositivo, por lo que permite mayor flexibilidad.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 CONCLUSIONES

En este proyecto se ha pretendido ofrecer una introducción a las automatización de dispositivos de red a través de Python y las librerías más utilizadas para estos fines, y cuyo estudio ha dado lugar a las siguientes conclusiones:

1. En primer lugar, es evidente que existen multitud de formas diferentes de proveer de automatización, por ejemplo, en este proyecto se ha hecho uso de ficheros YAML para separar la configuración y el direccionamiento del propio código del programa, pero nada hubiese impedido que se pudieran haber desarrollado métodos, por ejemplo en la sección 4.1, para configurar las VLANs pasando los parámetros directamente, o que en la sección 4.2 se hubiese realizado un programa donde la creación de los archivos XML hubiera sido totalmente responsabilidad del usuario. Lo que se pretende decir es que no existe una forma única o perfecta de crear soluciones universales, todo depende del contexto.

2. En segundo lugar, como se comentó anteriormente, RESTCONF es sin duda alguna el protocolo que más rápido realiza las transacciones, y la automatización por CLI a través de Netmiko, el más lento; en el caso de NETCONF, es probable que dependa del tamaño de los <rpc>, pero sigue siendo curioso que configuraciones similares en RESTCONF apenas lleven menos de cinco segundos incluso, aun así, esto no implica que un protocolo sea superior que el otro, depende de nuevo del contexto, por un lado, NETCONF permite realizar configuraciones de una forma más robusta, debido a su capacidad de poder comunicarse con diferentes archivos de configuración ("datastore"), así como el poder aplicar configuraciones en distintas fases, gracias a <commit>, y también a su capacidad de situar al dispositivo en un estado anterior si la configuración nueva no funcionara de forma correcta. Todo ello viene al coste de que el protocolo es mucho más complejo que en el caso de RESTCONF, y con muchas más opciones disponibles, muchas de las cuales, como se ha ido observado en el transcurso del proyecto, no son implementadas dependiendo del vendedor y del dispositivo en uso.

RESTCONF, por otro lado, permite hacer uso del modelo YANG pero sin la complejidad de NETCONF, esto se ha podido ver muy bien en los casos de uso de Postman, la cual es una aplicación con una interfaz de uso muy sencilla y completa, y permite realizar transacciones de forma rápida sin hacer falta el uso de código en Python o de otras herramientas como Ansible; de hecho resulta mucho más cómodo para verificar configuraciones sin tener que hacer uso del CLI, además de la flexibilidad que proporcionan los parámetros y la profundidad a la que se necesite llegar especificándola a través del URL. Por supuesto, el hecho de poder usar Python para interaccionar con RESTCONF también resulta muy útil, y en ciertos casos, una alternativa a NETCONF. Pero, en conclusión, a este punto, para configuraciones más avanzadas, sería más apropiado y seguro usar NETCONF, y hacer uso de RESTCONF para configuraciones más livianas y para visualizar la configuración o el estado de un dispositivo, pero, de nuevo, esto depende de cada organización y su contexto.

3. En tercer lugar, el uso de YANG implica, desde luego, el aprendizaje de un nuevo lenguaje y una nueva forma de entender la forma en la que se administran las redes; esto no quiere decir en absoluto que el CLI vaya a desaparecer, sino que, más bien, va a convivir con estos protocolos, ya que es otra forma de administración, y, además, proporciona un acceso de más bajo nivel al dispositivo en cuestión. Por supuesto, el poder configurar un dispositivo a tan bajo nivel proporciona ventajas, sobre todo porque pueden darse problemas que sean más fáciles de verificar a través de CLI que usando protocolos YANG (o incluso puede darse que RESTCONF y/o NETCONF no funcionen correctamente y por tanto no quede otra opción), por lo que el comprender estos nuevos protocolos ofrece más flexibilidad al administrador, así como una manera de poder programar y automatizar los dispositivos a través de interfaces desarrolladas para ello. Además, en caso de que no se puedan configurar dispositivos a través de YANG, siempre queda la opción de automatizar a través del CLI, para lo cual se necesitan conocimientos extensos, ya que, simplemente se le está delegando a un programa

4. En cuarto lugar, se puede observar que la posibilidad de realizar configuraciones universales es posible (por ejemplo, las interfaces en dispositivos Cisco y Juniper) a través de su máximo exponente, que es actualmente OpenConfig, el problema de ello es que la implementación de estos modelos YANG depende exclusivamente de que los vendedores los incluyan en sus dispositivos, lo cual requiere de un trabajo adicional, ya que estos modelos no siguen la forma de ningún sistema operativo de red propietario, por lo que cada fabricante ha de poder realizar las modificaciones pertinentes para que una instancia de un modelo OpenConfig se traduzca correctamente a la estructura de configuración del sistema operativo. También, es cierto que, si se tienen conocimientos en un sistema operativo, es más cómodo utilizar modelos nativos, ya que estos siguen la estructura de la configuración propia del dispositivo, al contrario que OpenConfig.

5. En quinto lugar, es evidente de que, si bien YANG tiene más de una década de antigüedad, hace muy pocos años que se está empezando a adoptar de forma masiva por los vendedores y por los administradores de red, y aun así sigue siendo un gran desconocido y su uso queda anecdótico, esto es posiblemente debido a la reticencia de aquellos profesionales de aprender a realizar su trabajo de una nueva forma, aunque, discutiblemente, también podría ser por el hecho de que ha sido en este último lustro (2015-2020) cuando se ha comenzado a observar una irrupción de documentación, tanto el libros, videotutoriales, "whitepapers", y demás; y que, aun así, en algunos casos ha resultado insuficiente para este proyecto, o muy difícil de encontrar.

6. Y en sexto y último lugar, el camino a la automatización de la red es desde luego el aprendizaje de un nuevo paradigma, donde existen tantas herramientas y formas de realizar un mismo fin que resulta fácil desconcertar a aquellas personas que quieran comenzar este camino debido a la multitud de posibilidades, por lo que en este proyecto, al fin y al cabo, se ha pretendido presentar tres de las herramientas que actualmente están en uso (prueba de ello es que la propia Cisco incluye estas librerías en algunos de sus exámenes de certificación **[30]**). De todas formas, todo este nuevo paradigma está tomando tal impulso que es posible que aparezcan nuevas herramientas o librerías que superen a las presentadas en este proyecto. Lo que si queda claro es que, independientemente de que aparezcan nuevas herramientas, la necesidad de saber programar se ha asentado en el mundo de las redes informáticas, puesto que proporciona al

profesional independencia de cualquier tipo de herramienta, como por ejemplo Ansible, y en el posible caso de que una organización no pudiera o no quisiera implementar un programa específico en un momento dado, siempre queda como opción el crear las soluciones por uno mismo.

5.2 LÍNEAS FUTURAS

En este último apartado de la memoria, se pretende ofrecer una visión orientada al futuro de aquellas herramientas y soluciones que están tomando el liderazgo en el sector de la automatización de redes informáticas o pueden hacerlo potencialmente en un futuro:

- Python se ha establecido como el principal lenguaje de programación para crear soluciones personalizadas, por lo que se espera que en un futuro siga manteniendo este control, sobre todo, cuando soluciones tan importantes hoy en día como es Ansible **[31]**, están escritas en Python.

- Ansible se ha convertido en la aplicación principal para automatizar todo tipos de redes, y, aunque tiene sus limitaciones (que no se van a comentar en este proyecto), el conocimiento de esta herramienta se considera fundamental en muchas organizaciones en la actualidad.

- Cisco ha desarrollado un “framework” llamado pyATS **[32]**, creada en Python, que está en constante evolución y que tiene multitud de funcionalidades, entre ellos la configuración de dispositivos y el testeo de topologías de red de forma automática. Aunque no es una herramienta muy conocida debido a su reciente publicación y su desarrollo activo, tiene el potencial de convertirse en un gran aliado para el administrador de red.

- Es posible que la interacción a tan bajo nivel con protocolos como NETCONF y RESTCONF a través de librerías como ncclient y requests quede en desuso, ya que otras librerías y/o aplicaciones pueden abstraer esta interacción. Aun así, siempre es conveniente conocer el funcionamiento de estos protocolos para facilitar la resolución de posibles problemas.

- Como se ha comentado anteriormente, el CLI no va a desaparecer, pero, si es posible que la interacción directa con este quede relegada poco a poco a un segundo plano a medida que la automatización de este se vaya haciendo más sencilla y el uso de aplicaciones como Ansible siga extendiéndose por las organizaciones.

- La posibilidad de realizar configuraciones universales se está convirtiendo poco a poco en una realidad; aunque aún queda trayecto, se puede observar cómo en soluciones más actuales, como Cisco NX-OS, y Juniper en las versiones más recientes de su sistema operativo Junos, la implementación de OpenConfig es mucho más completa que en los casos estudiados en este proyecto. Esto es posible que facilite el desarrollo de herramientas como las comentadas en los puntos anteriores, pero sin excluir a otras como pueden ser tecnologías como SDN y NFV.

I REFERENCIAS

1. Middleton, K. (2019, September). *Network Engineering is Dying (Except at Cloud Providers)*. Medium.

   https://medium.com/datadriveninvestor/network-engineering-is-dying-except-at-cloud-providers-86649af45afa

2. CareerBuilder. (2016, September). *A glimpse into the future of network engineers*.

   https://www.careerbuilder.com/advice/a-glimpse-into-the-future-of-network-engineers

3. Perkin, R. (2018, October). *Do Network Engineers Need to Be Developers?* IOD (iamondemand).

   https://iamondemand.com/blog/network-engineers-need-developers/

4. Cisco. (n.d.). *DevNet Certification.* Cisco DevNet. Retrieved December 30, 2020, from

   https://developer.cisco.com/certification/

5. Juniper. (n.d.). *Automation and DevOps Certification Track*. Retrieved December 30, 2020, from

   https://www.juniper.net/uk/en/training/certification/certification-tracks/devops?tab=jnciadevops

6. Iliesiu, A. (2017, May). *Defining Network Programmability*. Cisco Blogs.

   https://blogs.cisco.com/networking/defining-network-programmability

7. Beckenhauer, B. (2002, February). *SNMP Alert 2002: What is it all about?* SANS Institute.

   https://www.sans.org/reading-room/whitepapers/protocols/snmp-alert-2002-about-375

8. IETF. (2006, December). *RFC 4741. NETCONF Configuration Protocol.*

   https://tools.ietf.org/html/rfc4741

9. IETF. (2010, October). *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF).*

   https://tools.ietf.org/html/rfc6020

10. OpenConfig. (n.d.). Repository for publishing OpenConfig models, documentation, and other material for the community. Openconfig/public (Github repository). Retrieved December 30, 2020, from

    https://github.com/openconfig/public

11. OpenConfig. (n.d.). *Vendor-neutral, model driven network management designed by users*.

    https://www.openconfig.net

12. IETF. (2011, June). *RFC6242. Using the NETCONF protocol over Secure Shell (SSH).*

    https://tools.ietf.org/html/rfc6242

13. IETF. (2011, June). *Network Configuration Protocol (NETCONF).*

    https://tools.ietf.org/html/rfc6241

14. IETF. (2017, January). *RFC8040. RESTCONF Protocol.*

    https://tools.ietf.org/html/rfc8040

15. Postman. (n.d.). *The Collaboration Platform for API Development.*

    https://www.postman.com

16. Dainese, A. (n.d.). *NETCONF and RESTCONF*. IpSpace.

    https://www.ipspace.net/kb/CiscoAutomation/070-netconf.html

17. Byers, K. (n.d.). *Multi-vendor library to simplify Paramiko SSH connections to network devices.* Ktbyers/netmiko (Github repository). Retrieved December 30, 2020, from

https://github.com/ktbyers/netmiko

18. GNS3. (n.d.). *The software that empowers networks professionals.*

https://www.gns3.com

19. Cisco. (n.d.). *Networking Software (IOS & NX-OS).*

https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-software-releases-listing.html

20. *A very fast and expressive template engine*. (2020, December). pallets/jinja (Github repository). Retrieved December 30, 2020, from

https://github.com/pallets/jinja

21. Juniper. (n.d.). *Junos Network Operating System*.

https://www.juniper.net/us/en/products-services/nos/junos

22. *Python library for NETCONF clients*. (n.d.). ncclient/ncclient (Github repository). Retrieved December 30, 2020, from

https://github.com/ncclient/ncclient

23. OpenConfig. (n.d.). *YANG modules from standards organizations such as the IETF, The IEEE, The Metro Ethernet Forum, open source such as Open Daylight or vendor specific modules*. YangModels/yang (Github repository). Retrieved December 30, 2020, from

https://github.com/YangModels/yang

24. Juniper. (n.d.). *Junos Yang module*. Juniper/yang (Github repository). Retrieved December 30, 2020, from

https://github.com/Juniper/yang

25. *An extensible YANG validator and converter in python.* (2020, December). mbj4668/pyang (Github repository). Retrieved December 30, 2020, from

https://github.com/mbj4668/pyang

26. Cisco. (2016, July). *How OSPF Injects a Default Route into a Stub or Totally Stub Area*.

https://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/47869-ospfdb10.html

27. Cisco. (n.d.). *Cisco NX-OS*.

https://www.cisco.com/c/en/us/products/ios-nx-os-software/nx-os

28. Cisco. (n.d.). *Cisco Nexus OPENCONFIG YANG, Release 9.2x*. Cisco DevNet.

https://developer.cisco.com/docs/openconfig-yang-release-9-2x

29. *Requests: HTTP for Humans (TM).* (n.d.).

https://requests.readthedocs.io/en/master

30. Cisco. (n.d.). *EXAUTO Exam Topics.* Cisco Learning Network.

https://learningnetwork.cisco.com/s/enauto-exam-topics

31. Red Hat. (n.d.). *Ansible is Simple IT Automation.* Ansible.

https://www.ansible.com

32. Cisco. (n.d.). *Accelerating your DevOps with pyATS & Genie.* Cisco Devnet.

https://developer.cisco.com/pyats

II RECURSOS ADICIONALES

En este apartado se recopilan aquellas herramientas que se han utilizado como apoyo para facilitar y mejorar la realización del proyecto:

RE1. Draw.io

Descripción: Utilidad online gratuita que permite realizar multitud de diagramas de forma sencilla e intuitiva. Las figuras 1, 9, 11 y 25 están realizadas a través de esta herramienta.

URL: https://app.diagrams.net

RE2. J2Live

Descripción: Esta utilidad permite obtener el output correspondiente a partir de un "template" escrito en Jinja2 y datos codificados en JSON o YAML, lo cual ha facilitado la resolución de errores y el refinamiento en el diseño de los distintos "templates" utilizados en este proyecto. También informa de si existe algún problema en el archivo YAML o JSON. Las figuras 24, 36 y 39 muestran ejemplos de uso de esta herramienta.

URL: https://j2live.ttl255.com

RE3. XML Viewer – Code Beautify

Descripción: Esta herramienta online permite estructurar cualquier archivo válido en XML de manera que sea fácilmente legible, lo cual facilita la lectura de "payload" en XML cuando no está bien estructurado.

URL: https://codebeautify.org/xmlviewer

RE4. JSON to YAML

Descripción: Herramienta online que permite transformar de forma inmediata datos en JSON a YAML y viceversa, lo cual resulta muy útil para formar "payloads" para RESTCONF a partir de ficheros YAML. Un ejemplo de uso de esta herramienta se puede observar en la figura 58.

URL: https://www.json2yaml.com

III GLOSARIO

ACID: Modelo en el que se identifican cuatro propiedades que se consideran necesarias para que los datos en una base de datos sean procesados de forma segura.

Cable de consola: Cable que se conecta a la interfaz con0 que permite acceder de forma directa a un dispositivo de red; suelen ser muy útiles cuando estos están sin configurar y no disponen de conectividad.

Capability: En NETCONF, un capability, o capacidad permite a un dispositivo de red ofrecer una serie de funcionalidades. El diseño modular de NETCONF permite a un vendedor usar solamente las que le interese (exceptuando la básica).

Command Line Interface (CLI): Interfaz de línea de comandos.

Commit: En el contexto de la informática, commit significa confirmar, y permite realizar una serie de cambios de configuración que no entran en efecto hasta que se confirman.

cURL: Aplicación UNIX similar a Postman, pero sin una interfaz gráfica.

Datastore: Palabra que en español se podría traducir como "almacén de datos". En el caso de NETCONF, cada datastore es una instancia que almacena una configuración determinada del dispositivo.

DevNet: Programa lanzado por Cisco que ofrece multitud de recursos para fomentar el desarrollo de profesionales orientados a la automatización en dispositivos Cisco.

DevOps: Paradigma en el que la persona responsable del desarrollo de una solución es también la que lo opera, por lo que fomenta la retroalimentación de estos dos ciclos de un proyecto.

EIGRP: Protocolo de enrutamiento interno vector-distancia propietario de Cisco.

Etherchannel: Tecnología que permite combinar varios puertos físicos Ethernet en una unidad lógica que combine el ancho de banda de todos ellos.

Hipervisor: Programa informático que permite separar los componentes físicos de un dispositivo con respecto a su sistema operativo, por lo que permite instanciar varios a la vez. A estas instancias se les conocen como máquinas virtuales.

Internet Engineering Task Force (IETF): Organización responsable en gran parte del fomento y desarrollo de nuevas tecnologías para Internet.

Jinja2: Lenguaje de modelado para Python y utilizado en aplicaciones como Ansible que permite estructurar los datos y sustituirlos por variables.

LACP: Protocolo estandarizado que permite la formación dinámica de etherchannels.

MIB: Instancia estructurada en lenguaje SMI/SMIv2 en un dispositivo que ejecuta SNMP.

Manager y Agent: Términos propios de protocolos como NETCONF y SNMP; los managers son aquellos dispositivos que se conectan a los agentes para obtener información de ellos y modificar sus configuración si fuera necesario.

Merge: Podría traducirse como combinar. En el caso de la informática, una operación merge significa que se aplica aquella configuración que no sea conflictiva con otra existente.

Network Developer: Posición profesional orientada al desarrollo de soluciones para las redes informáticas a partir de distintas herramientas, aunque con especial presencia de la programación.

OpenConfig: Agrupación informal de distintas organizaciones y multinacionales cuyo objetivo es proporcionar diferentes herramientas para poder administrar dispositivos de red de forma universal sin importar su fabricante.

Overhead: Información que permite transportar los datos útiles (payload) pero que no tiene utilidad por si mismos fuera del transporte de estos datos.

Payload: Se puede traducir como "datos útiles". Un payload es aquella información con la que se pretende realizar un cambio en algún dispositivo.

Postman: Aplicación que permite realizar transacciones HTTP de forma sencilla con cualquier dispositivo compatible.

RESTful: Modelo de arquitectura informática en el que se hace uso de los métodos HTTP para controlar una aplicación. Este modelo tiene ciertas propiedades que deben de llevarse a cabo para que una interfaz se considere RESTful, ya que existen aplicaciones que hacen uso de métodos HTTP, pero no se consideran RESTful.

Read-only: Significa solo lectura, se refiere a que una instancia con esta propiedad solo se puede leer, pero no modificar.

Read-write: Significa lectura/escritura, se refiere a que una instancia con esta propiedad se puede leer y modificar.

Remote Procedure Call (RPC): Permite la ejecución de subrutinas en un programa remoto; en cierto modo es equivalente a hacer uso de métodos de un código.

Request for Comments (RFC): Son los documentos a través de los cuales la IETF publica sus trabajos; existen varios tipos dependiendo si es una propuesta para un nuevo protocolo, solo informacional, etc.

Rollback: Podría traducirse como "retroceder". En informática, un rollback es cuando se vuelve a un estado de configuración anterior en el tiempo.

SMI/SMIv2: Lenguaje de estructuración para las MIB en SNMP. Se podría simplificar diciendo que YANG sería su equivalente para NETCONF y RESTCONF.

SMIng: Fue un lenguaje que estuvo en desarrollo con el objetivo de resolver los problemas de SMIv2; aunque fue cancelado, muchas de sus propiedades se aplican en YANG.

SSL y TLS: Protocolos que ofrecen servicios de encriptado y autenticación a través de HTTP; TLS es el sucesor de SSL, que se considera inseguro en la actualidad.

Script: Código informático que no requiere ser compilado para su ejecución, sino que es un intérprete el que ejecuta el archivo a partir del propio código en cada iteración.

Software-defined Networking (SDN): Metodología en la que se separa el plano de control de los dispositivos del de datos, y se centraliza el primero para poder administrar la red de forma unificada.

Stub: Tipo de área especial que permite optimizar las tablas de enrutamiento de distintos dispositivos en OSPF.

Template: Se podría definir como un modelado específico para datos, por lo tanto, en Jinja2, una misma serie de datos puede ofrecer un output diferentes dependiendo del template usado.

Timeout: Contador por el que se considera que una transacción es inválida si supera el tiempo establecido; en ese caso, se considera que se ha dado un error de timeout.

URL: Identifica la dirección de un recurso web.

Whitepaper: Documento que tiene información detallada y muy revisada por una fuente autoritativa sobre un tema determinado. En el caso de este documento, whitepaper se refiere a los manuales de uso de los fabricantes.

XML namespace: Propiedad de XML que permite identificar de forma única distintos elementos.