ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

IMPLEMENTACIÓN DE ANSIBLE PARA LA CONFIGURACIÓN DE UNA VPN

IMPLEMENTAR ANSIBLE PARA LA CONFIGURACIÓN DE UNA VPN

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO SUPERIOR EN REDES Y TELECOMUNICACIONES

ELVIS ALEXIS TABANGO MATANGO

DIRECTOR: ING. ITALO ALEXANDER CARREÑO MENDOZA

DMQ, marzo 2023

CERTIFICACIONES

Yo, ELVIS ALEXIS TABANGO MATANGO declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

Elvis Alexis Tabango Matango

elvis.tabango@epn.edu.ec

elvisttabango@gmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por Elvis Alexis Tabango Matango, bajo mi supervisión.

Ing. Italo Alexander Carreño Mendoza

DIRECTOR

Italo.carreño@epn.edu.ec

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

ELVIS TABANGO

DEDICATORIA

El proyecto implementado se encuentra dedicado a Dios, por haberme brindado salud y fuerza para alcanzar cada una de mis metas propuestas.

De igual manera a mi familia, quienes, con su paciencia, amor, y esfuerzo me han acompañado en este proceso, sin negar su apoyo incondicional en cada una de mis etapas.

A mi sobrino, que se ha convertido en un hermano más y me ha brindado esa confianza para contarle cada uno de mis problemas personales.

A mis mascotas, princesa, boster (que en paz descanse), y tobias. Que con sus travesuras siempre me lograban sacar una sonrisa en aquellos momentos de tristeza.

Elvis Tabango

AGRADECIMIENTO

Agradezco a toda mi familia, que a pesar de no ser la más grande y perfecta, siempre ha sabido mantenerse en pie, y ha sabido sobrellevar cada uno de los problemas de la mejor manera.

En especial a mi hermana, que se ha convertido en una figura materna y paterna a la vez, le agradezco desde el más pequeño detalle que me ha sabido brindar en todas las etapas de la vida, comenzando desde que era un niño hasta el día de hoy.

Agradezco al ser más amado en este mundo, mi mamá, que, a pesar de sus problemas, siempre me ha sabido brindar todo el apoyo incondicional más que nadie, doy gracias por sus consejos, y su amor.

Doy gracias a mis docentes tanto escolares, de bachillerato, y universitarios, por haber sabido compartir sus conocimientos, y más que todo por aportar un poquito de ese granito de arena para la formación de este profesional.

Elvis Tabango

ÍNDICE DE CONTENIDOS

RESUMEN

En el siguiente trabajo de titulación, se realiza la automatización de la implementación de una red VPN utilizando Ansible para la configuración de, enrutadores, servidor, y dispositivo final, que en este caso viene a ser un cliente VPN. Todo este proceso permite conseguir una mejor eficiencia, precisión, seguridad, y gestión en la implementación.

El proyecto se encuentra distribuido a través de cinco secciones, siendo la primera sección la descripción del proyecto, dentro del cual, se da a conocer la teoría de las herramientas necesarias para el despliegue de una VPN con Ansible; además, se encuentra detallado los objetivos específicos que permitirán conseguir el resultado esperado, así como la limitación de la implementación del proyecto a través del planteamiento de un alcance.

En la segunda sección, se encuentra detallado la metodología de la investigación, el cual despliega cada uno de los objetivos específicos planteados con anterioridad, con el fin de cumplir a detalle los mismos mediante instrucciones que se irán explicando en el desarrollo del proyecto.

En la tercera sección está contenido el desarrollo de cada uno de los objetivos específicos con su respectiva justificación, de igual manera se detallan los resultados obtenidos, como parte de esta sección.

Las dos últimas secciones detallan las conclusiones obtenidas en base a lo implementado, y las recomendaciones dirigidas a interesados en la implementación de un proyecto similar.

PALABRAS CLAVE: *playbook*, VPN, wg, *wireguard*, ansible, enrutador.

ABSTRACT

In the following graduation project, the automation of the implementation of a VPN network using ansible for the configuration of routers, server, and end device, which in this case is a VPN client, is carried out. This entire process allows for better efficiency, accuracy, security, and management in the implementation.

The project is divided into five sections, with the first division begin the project description, in which the theory of the necessary tools for deploying a VPN with ansible is outlined. Additionally, specific objectives that will allow achieving the expected result are detailed, as well as the limitations of the project implementation through the establishment of a scope.

In the second section, the research methodology is detailed, which deploys each of the previously stated specific objectives, in order to fulfill them in detail through instructions that will be explained in the development of the project.

In the third section, the development of each of the specific objectives with its respective justification is contained, and the results obtained as part of the division are also detailed.

The last two sections detail the conclusions obtained based on the implementation and recommendations directed towards those interested in implementing a simi.ar project.

KEYWORDS: *playbook, vpn, wg, wireguard, ansible, router.*

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

El presente proyecto tiene el objetivo de evidenciar la utilidad de Ansible, en tareas de automatización mediante la creación de una red simulada con GNS3 y el hipervisor VirtualBox. En especial, se utiliza Ansible para automatizar el despliegue de un protocolo VPN basado en *wireguard.*

El despliegue de este componente involucra el uso de herramientas disponibles en Ansible, como: *playbook*, inventario y módulos. Estos operan en conjunto para permitir el despliegue de una configuración remota, facilitando el despliegue del protocolo OSPF, así como la escritura de diferentes parámetros necesarios para la puesta en marcha del túnel VPN.

1.1 Objetivo general

Implementar Ansible para la configuración de una VPN.

1.2 Objetivos específicos

- Instalar la herramienta de Ansible en el nodo controlador.
- Montar los diferentes dispositivos de red en un simulador de redes.
- Implementar las configuraciones mediante la herramienta Ansible
- Realizar pruebas de funcionamiento y verificación de los resultados obtenidos.

1.3 Alcance

El presente proyecto conlleva la implementación de una topología de red, donde los dispositivos logren tener conectividad configurándolos de forma automatizada mediante la herramienta de Ansible. Para esto se debe establecer una comunicación SSH entre el nodo controlador, máquina principal donde se instala Ansible, y los diferentes dispositivos de red. Con esto se despliega de manera remota las configuraciones necesarias para la implementación de la topología de red establecida.

1.4 Marco Teórico

Enrutamiento

En el contexto de *networking* el enrutamiento hace referencia al proceso en el cual se seleccionan las rutas que tomarán los paquetes para lograr transportarse de un origen hacia un destino [1].

El proceso se realiza a través de dispositivos de *hardware*, los cuales se les conoce como enrutadores y que trabajan en la capa 3 (red) del modelo OSI [2]. Antes de comenzar el proceso de enrutamiento, los enrutadores, dependiendo de la configuración del administrador de red, establecen ciertos criterios a tomarse en cuenta, como el número de enrutadores que se atravesarán en todo el trayecto, el ancho de banda de las rutas, los tipos de protocolos habilitados en los enlaces, la latencia hacia el destino, entre otras.

Tipos de enrutamiento

Los enrutadores para llevar a cabo la comunicación entre dos redes necesitan contar con una tabla de enrutamiento en la cual se detallen las rutas que deben seguir los paquetes. El administrador de red debe fijar rutas específicas que han de seguir los paquetes para alcanzar su destino, esto evidentemente contribuye a la seguridad, pero su complejidad se incrementa con el aumento de las subredes a comunicar [3].

En cambio, con el enfoque del enrutamiento dinámico, los paquetes son dirigidos basándose en un algoritmo el cual se encarga de tomar en cuenta criterios antes de establecer una ruta de destino [4]. Dentro de esta categoría existen protocolos de enrutamiento como OSPF, ISIS, BGP, EIGRP, cada uno mantiene su propia complejidad, por lo que es importante conocer el funcionamiento del algoritmo presente en los protocolos mencionados.

Open Shortest Path First

Protocolo de enrutamiento OSPF miembro de los protocolos de enrutamiento dinámicos, utiliza un modelo de estado del enlace, modelo en el cual se busca en primera instancia conocer el estado actual del enlace, que comparten los enrutadores, la información recopilada sirve como base para poder crear una topología de toda la red [5].

Los enrutadores que son configurados con OSPF solamente inundan la red con paquetes *Link State Advertisement* (LSA) en dos escenarios; al iniciar su operación por primera vez dentro de la red, y cuando existe un cambio de topología.

VPN

Una *virtual private network* (VPN), mantiene la privacidad del usuario ocultando su dirección IP real. De manera tradicional, las actividades que realiza el usuario en internet se ven reflejadas en el *internet service provider* (ISP), es decir, que cualquier consulta que se realice a internet pasa a través del servidor del ISP [6].

Mediante el uso de una VPN se crea un túnel seguro, en el cual la información del usuario es cifrada, la información atraviesa el servidor del proveedor de Internet, pero esta vez de manera cifrada, por lo que a pesar de que conste el registro de salida a internet en los servidores del ISP, este no será descifrable debido a que solamente el servidor VPN será capaz de hacerlo, de igual manera para poder navegar en internet se lo realiza a través de la dirección que provee el servidor VPN.

Protocolo VPN *wireguard*

Desarrollado en el año 2016 con el objetivo de optimizar el uso y la administración de un protocolo VPN. La creación de *wireguard* tiene un enfoque distinto a otros protocolos VPN existentes, comenzando por la manera en cómo encripta la información. El protocolo usa criptografía moderna ChaCha20 [7] perteneciente al grupo de cifrado de flujo seguro, así llegando a usar claves de cifrado de 128 o 256 bits.

Las líneas de código con la que cuenta el protocolo para su funcionamiento son de una longitud considerable, lo que refleja como resultado una velocidad superior ante los protocolos tradicionales como OpenVPN [8]. Además, es de código abierto, esto facilita que los desarrolladores busquen nuevos enfoques del mismo, probándolos en diferentes productos.

Fue desarrollado inicialmente para sistemas Linux, pero esto ha ido mejorando hasta tener compatibilidad con *Windows*, Mac, Android, inclusive iOS. Debido a la popularidad en crecimiento de este protocolo, las empresas enfocadas en brindar servicios VPN como Mullad VPN, y NordVPN, ya cuentan con el funcionamiento de *wireguard* en sus planes de oferta [9].

Tipos de VPN

VPN de acceso remoto

En este tipo de VPN, los clientes acceden a los recursos privados a través de caminos seguros, pero el tipo de enlace no es permanente y solo se habilita cuando el cliente decide acceder a cualquier recurso de manera privada [11]. Este tipo de solución requiere un buen dimensionamiento de la capacidad del servidor central, de lo contrario puede causar cuellos de botella en la infraestructura de red.

VPN de sitio a sitio

A diferencia de la solución anterior, la conexión que mantiene el cliente y el servidor es de manera permanente. La presente solución es muy usada en empresas que cuentan

con varias sucursales debido a que las diferentes subredes son unificadas en una sola intranet y de esa manera evitan el espionaje exterior.

Funcionamiento de Ansible

Ansible es un *software* libre de automatización de tareas remotas, disponible para sistemas operativos Linux. El *software* cuenta con varias herramientas para su funcionamiento, siendo SSH uno de los protocolos fundamentales para administrar equipos de manera remota [12]. Por tal motivo, los equipos que se deseen administrar deben mantener en estado de actividad el protocolo mencionado. Ansible cuenta con diferentes herramientas disponibles para mejorar los procesos de automatización de tareas, detallándose los más fundamentales a continuación:

Módulos: Son librerías de comandos disponibles en Ansible, los cuales pueden ser ejecutados desde un *playbook* o una línea de comandos. Existen varios módulos disponibles, pero algunos requieren una instalación adicional dependiendo del uso de Ansible, de igual manera no todos los módulos son compatibles con todas las versiones de esta herramienta, debido a que es de distribución libre y sigue en continua mejora desde su primer lanzamiento [13].

***Playbooks*:** Son archivos que pueden ser escritos en lenguaje YAML. Dentro de los *playbooks* se detallan paso a paso los comandos a ejecutarse, estos procesos están contenidos en un bloque a los cuales se les conoce como tareas. Estos archivos usan tres guiones para indicar el inicio de un documento de configuración, pero en caso de no utilizarlos no se producirá ningún error [13].

Inventarios: Archivos en los cuales van detallados los *hosts* a ser administrados, estos *hosts* pueden ser agrupados para un mejor manejo de los mismos. Ansible necesariamente debe contar con el acceso a este archivo, debido a que es lo primero que lee antes de ejecutar cualquier tarea remota, por tal razón, el archivo de *hosts* o inventario viene creada en la ruta */etc/ansible/hosts* [14].

Simulador de red GNS3

Comprende una colección de herramientas que permiten la simulación de redes de computadoras. Usado en escenarios de simulación de IOS de enrutadores Cisco, ATM, *firewalls*, y PIX. La herramienta hace uso de la plataforma *dynamips*, mediante la cual se emula las versiones 1700, 2600, 3600, 3700, y 7200 de Cisco [15].

El *software* mantiene una interfaz sencilla, lo que facilita el armado de topologías de red, además se resalta que esta herramienta es de código abierto, compatible con los SO: Windows, Mac OS, y Linux [16].

2 METODOLOGIA

Para la ejecución del presente proyecto se aplicó una investigación exploratoria con el fin de recabar información necesaria sobre los diferentes protocolos VPN, y las características que debe tener una herramienta de automatización de TI. La información obtenida permitió adquirir una mejor comprensión sobre el funcionamiento de una VPN y la manera en cómo se podría automatizar estos escenarios.

En primer lugar, se creó una máquina virtual de tipo servidor con un sistema operativo Ubuntu de versión 18.04 a través del hipervisor VirtualBox, el cual conformaría la central controladora. Una vez instalado el sistema operativo se agregó el repositorio de la herramienta Ansible, seguido de ello se realizó la instalación de la herramienta en conjunto con lenguaje de programación *Python*. A continuación, también se crearon dos máquinas virtuales con un sistema operativo Ubuntu 22.04 de escritorio, las cuales cumplirían el rol de servidor y cliente VPN.

Posteriormente, se identificó como candidato para la simulación de red a la herramienta GNS3, para lo cual se descargó una versión OVA del *software* y se agregó al hipervisor logrando así comunicar el hipervisor con el *software* de simulación de red GNS3, y ello también garantizó que las máquinas virtuales alojadas en VirtualBox podían ser utilizadas como elementos dentro de GNS3. De igual manera se instaló en el simulador de red el IOS del enrutador **cisco C3725.**

Se creó la topología de la red con los enrutadores y la máquina controladora, se configuró los enrutadores, así como la máquina controladora con los comandos necesarios para permitir la comunicación SSH.

Establecida la topología de red y garantizada la comunicación SSH entre la máquina controladora y los enrutadores, se definió un *playbook* Ansible el cual contiene los comandos cisco para el establecimiento de comunicación entre enrutadores a través del protocolo de enrutamiento OSPF.

Definido el *playbook* de configuración OSPF y verificado su correcto funcionamiento. Se definió las máquinas virtuales tanto para el servidor como para el cliente VPN a los cuales se les estableció una dirección IP estática, de igual manera se estableció las

conexiones SSH entre máquina controladora - servidor VPN, y máquina controladora - cliente VPN.

Completado satisfactoriamente los pasos anteriores. Dentro de la máquina controladora se añadió un *playbook* Ansible con las instrucciones necesarias para el despliegue de configuración del servidor VPN a través del protocolo *wireguard*, también se creó un *playbook* con los respectivos comandos para establecer un cliente VPN.

Finalmente, como parte de las pruebas de funcionamiento se verificó que la interfaz de *wireguard*, a la cual se le identifica como wg0 estuviese habilitado tanto en el servidor como en el cliente. Además, se usó el comando *ping* para verificar la existencia de transmisión de paquetes entre el servidor y el cliente.

3 RESULTADOS

El proyecto conlleva la simulación de una red VPN, para ello se realiza el uso de la herramienta GNS3, en el cual se añaden: un servidor, una máquina cliente, un dispositivo controlador (Ansible), y dispositivos de enrutamiento (*routers*). Todos estos dispositivos tienen conectividad entre sí, debido a que se implementa el protocolo OSPF para permitir la comunicación en la red simulada; la configuración de los equipos se lo realiza mediante Ansible, herramienta que se encuentra instalada en una máquina de tipo servidor, y este a su vez mantiene conexión con los enrutadores a través de un *switch* que cumple la función de distribución de información entre controladora y subred, para permitir el despliegue de configuraciones remotas mediante el uso de *playbooks*.

3.1 Instalación de Ansible en el nodo controlador

Instalación de Ubuntu *Server*

Se creó una máquina virtual Ubuntu *server* 18.04.6 LTS con las características detalladas en la Figura 3.1, dentro de esta máquina virtual se alojará la herramienta Ansible, además cabe destacar, que los recursos asignados están en función de su aplicación, es decir, que al ser una máquina de tipo servidor no contará con una interfaz gráfica y solamente estará enfocada en la ejecución de comandos remotos [16].

Figura 3.1 Características Servidor Ubuntu

Durante la instalación del sistema operativo Ubuntu *Server*, es necesario marcar la casilla que se resalta en la Figura 3.2 para la instalación del servidor OpenSSH, esto debido a que Ansible requiere usar una conexión SSH para poder trabajar en remoto, y también se lo realiza con la finalidad de evitar cualquier error durante la instalación manual de OpenSSH.

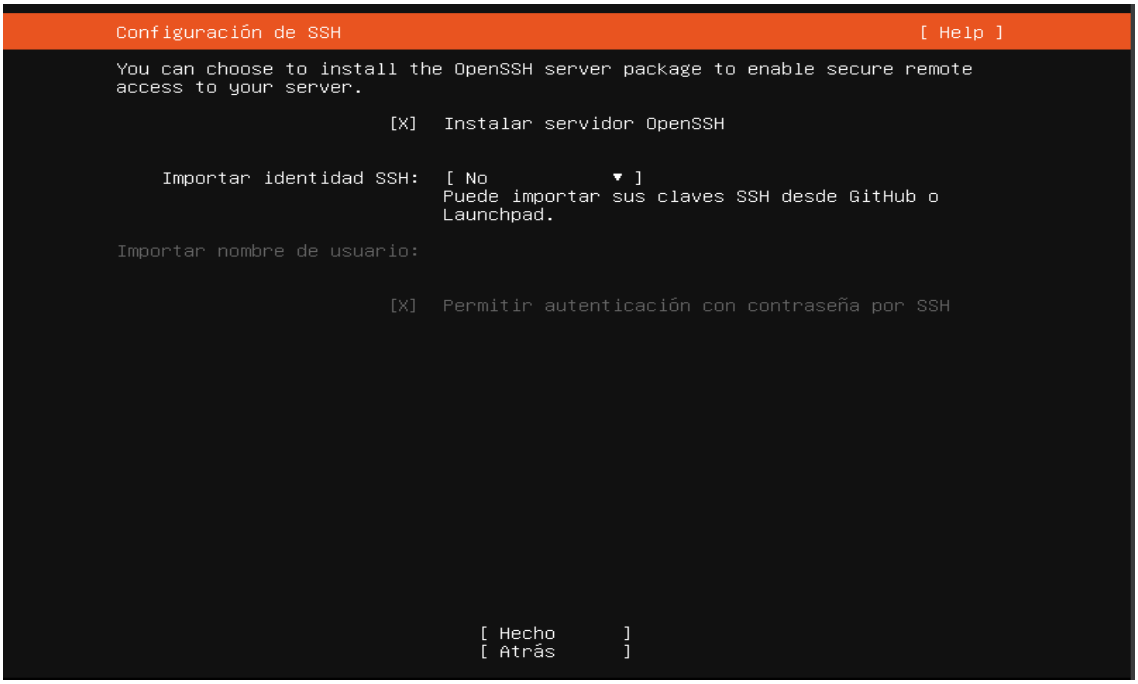

Figura 3.2 Servidor OpenSSH como requisito de instalación.

Instalación de Ansible 2.9.27

Finalizada la instalación del servidor, se inicia con los pasos previos a la instalación de Ansible. En primer lugar, se realiza una actualización de los repositorios mediante el comando ***sudo apt update*** mostrado en la Figura 3.3 para mantener las versiones de los paquetes actualizadas.

```
elvis@elvis:~$ sudo apt update
[sudo] password for elvis:
Obj:1 http://ec.archive.ubuntu.com/ubuntu bionic InRelease
Des:2 http://ec.archive.ubuntu.com/ubuntu bionic-updates InRelease [88,7 kB]
Des:3 http://ec.archive.ubuntu.com/ubuntu bionic-backports InRelease [83,3 kB]
Des:4 http://ec.archive.ubuntu.com/ubuntu bionic-security InRelease [88,7 kB]
Des:5 http://ec.archive.ubuntu.com/ubuntu bionic/main Translation-es [364 kB]
22% [5 Translation-es 1.219 B/364 kB 0%]_
```

Figura 3.3 Actualización de repositorios

Siguiente a ello, se requiere realizar una abstracción de repositorios para la instalación de archivos de paquete personal (PPA), es decir que a través del comando ***software-properties-common*** que se muestra al inicio de la Figura 3.4, se añade la funcionalidad de agregar *software* ajeno a los repositorios de APT y administrarlos como cualquier aplicación [17].

```
elvis@elvis:~$ sudo apt install software-properties-common
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  python3-software-properties
Se actualizarán los siguientes paquetes:
  python3-software-properties software-properties-common
2 actualizados, 0 nuevos se instalarán, 0 para eliminar y 24 no actualizados.
Se necesita descargar 33,8 kB de archivos.
Se utilizarán 0 B de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] s
Des:1 http://ec.archive.ubuntu.com/ubuntu bionic-updates/main amd64 software-properties-common all 0
.96.24.32.18 [10,1 kB]
Des:2 http://ec.archive.ubuntu.com/ubuntu bionic-updates/main amd64 python3-software-properties all
0.96.24.32.18 [23,8 kB]
Descargados 33,8 kB en 1s (48,8 kB/s)
(Leyendo la base de datos ... 67436 ficheros o directorios instalados actualmente.)
Preparando para desempaquetar .../software-properties-common_0.96.24.32.18_all.deb ...
Desempaquetando software-properties-common (0.96.24.32.18) sobre (0.96.24.32.14) ...
Preparando para desempaquetar .../python3-software-properties_0.96.24.32.18_all.deb ...
Desempaquetando python3-software-properties (0.96.24.32.18) sobre (0.96.24.32.14) ...
Configurando python3-software-properties (0.96.24.32.18) ...
Configurando software-properties-common (0.96.24.32.18) ...
Procesando disparadores para dbus (1.12.2-1ubuntu1.4) ...
Procesando disparadores para man-db (2.8.3-2ubuntu0.1) ...
```

Figura 3.4 Abstracción de repositorio

Añadida la funcionalidad anterior, APT permite adquirir nuevos repositorios. En la Figura 3.5, se ilustra la adición del repositorio de Ansible para su correspondiente instalación, esto se lo realiza a través del comando ***apt-add-repository ppa:ansible/ansible***.

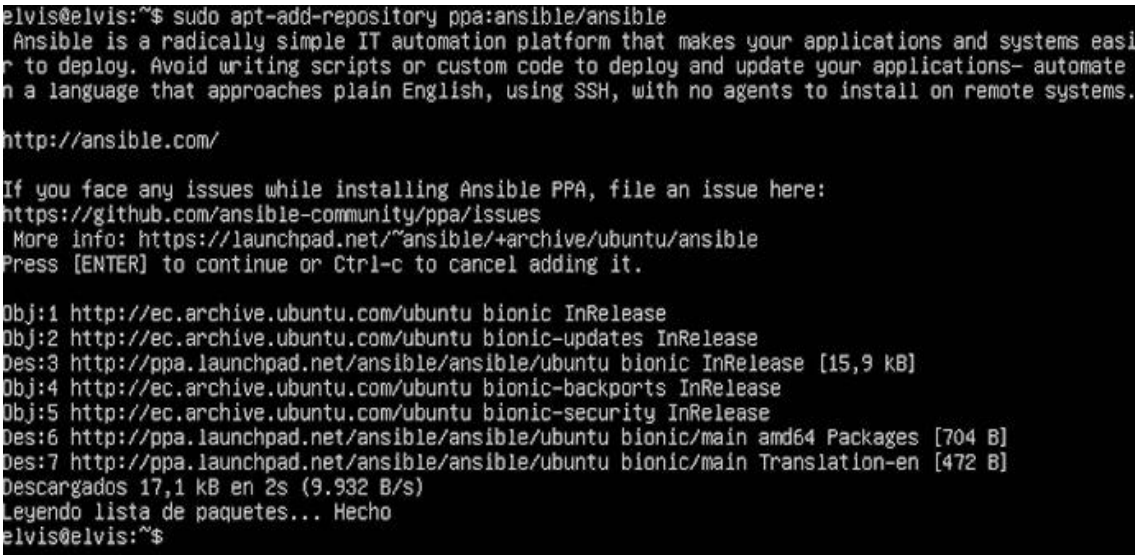

Figura 3.5 Adición del repositorio ppa:ansible/ansible

Completado los pasos anteriores, es posible realizar la instalación de Ansible a través de APT, para lo cual, con el comando ***apt install ansible*** en modo de superusuario, detallado el despliegue del mismo en la Figura 3.6, se inicia la instalación de la herramienta controladora.

Figura 3.6 Instalación de Ansible

Uno de los requisitos que requiere este *software* para poder funcionar es contar con *Python* instalado dentro del sistema operativo, por ello, seguido de la instalación de Ansible mostrada en la Figura 3.6, se digita el comando ***apt install pyhton3-argcomplete*** mostrado al inicio de la Figura 3.7. Además, se activa *Python* global para asegurar la disponibilidad para Ansible. Estos detalles son esenciales para el buen funcionamiento de la herramienta de automatización.

Figura 3.7 Instalación de Python

Para poder continuar con lo propuesto en el proyecto, se comprobó que Ansible y *Python* estén instalados correctamente en el servidor Ubuntu. Como resultado se observa en la Figura 3.8 que la versión instalada de Ansible es la 2.9.27 y la versión de *Python* es 2.7.17.

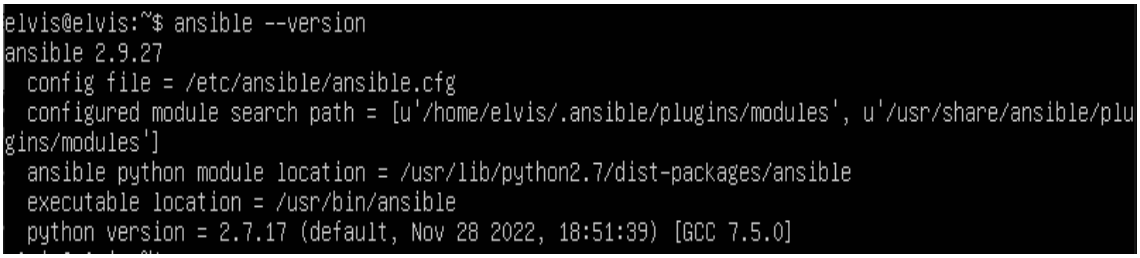

Figura 3.8 Revisión de versión de Ansible

3.2 Montaje de dispositivos en el simulador de redes GNS3

Instalación de módulo Cisco

Ansible al ser un *software* libre, constantemente continúa agregando nuevas funcionalidades, por tal razón durante su primera instalación solamente incluye módulos básicos.

Debido a que el objetivo fue manipular dispositivos Cisco con Ansible, se realiza la instalación de un módulo específico para la marca de equipos mencionado. Esto se lo realiza manualmente con el comando ***ansible-galaxy collection install cisco.ios*** mostrado en la Figura 3.9.

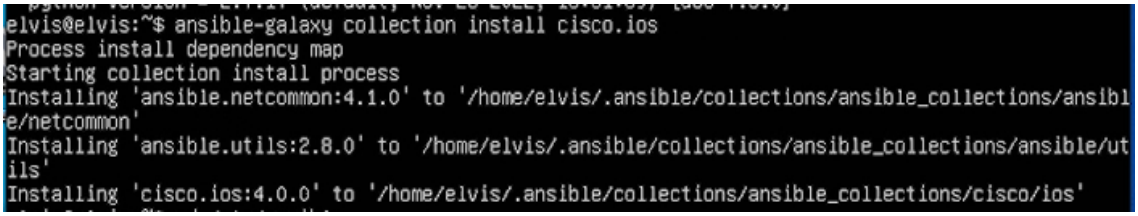

Figura 3.9 Instalación del módulo Cisco.ios

Diseño de topología y conexión de dispositivos

Previo a la implementación de la topología en el simulador de redes GNS3, es importante configurar adecuadamente las máquinas virtuales (Controlador, Servidor, Cliente) contenidas en el hipervisor VirtualBox. Esto incluye, el cambio de adaptador de red a controlador genérico, para que la herramienta de simulación pueda tomar el control sobre dichas máquinas, lo mencionado se contrasta en la Figura 3.10.

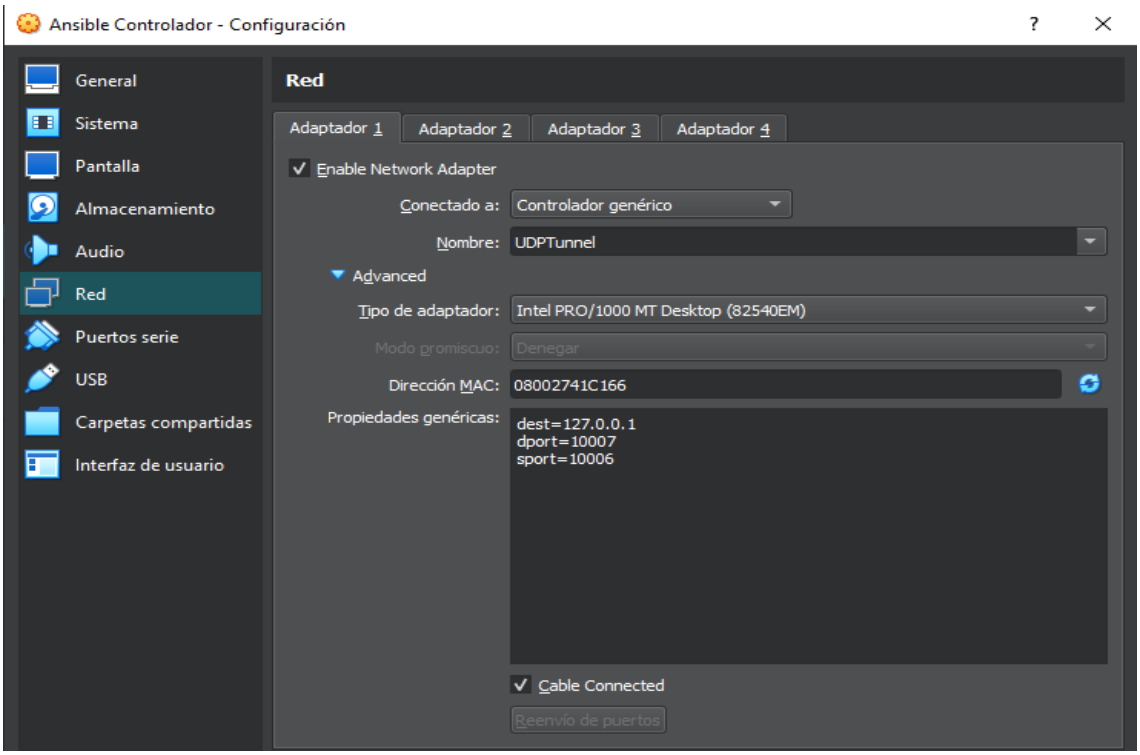

Figura 3.10 Adaptador de red en modo genérico

En la Figura 3.11, se estableció la configuración de los equipos conectados a GNS3, para llegar a este apartado se debe dirigir al panel de GNS3 desde el cual se añaden las máquinas virtuales provenientes de VirtualBox. Siguiente a ello se marcó la casilla mostrada en la Figura 3.11 para permitirle a GNS3 usar un mismo adaptador de red en

diferentes puertos, así evitando el conflicto de uso de adaptadores, problema que suele ser frecuente en la herramienta utilizada.

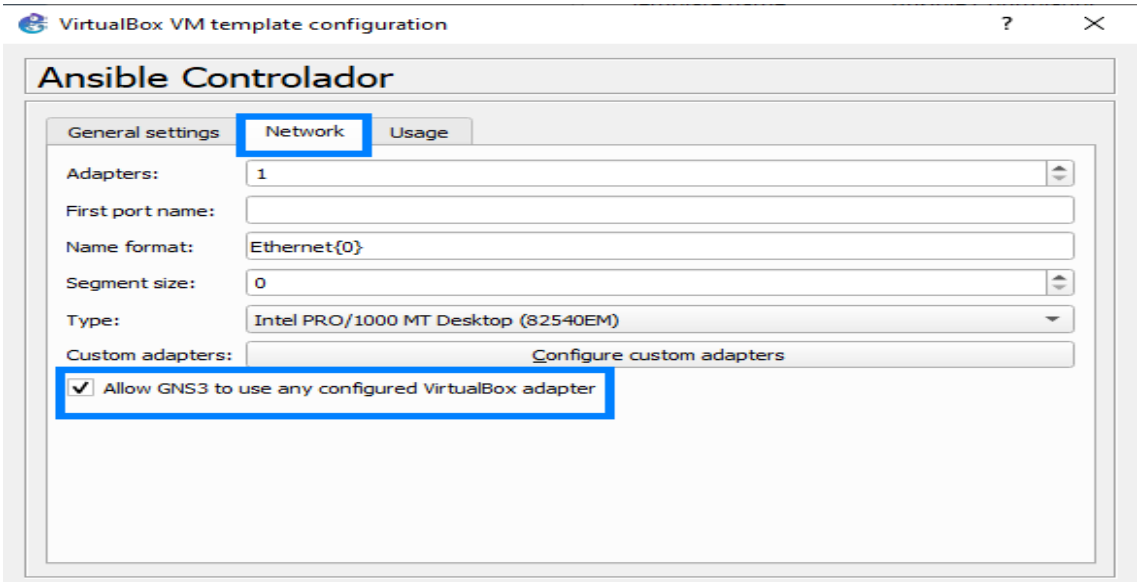

Figura 3.11 Adaptador de red en GNS3.

A este punto del apartado se cuenta con todo lo necesario para realizar la implementación de la topología de red. En la Tabla 3.1, se identifica el dispositivo, la función del mismo dentro de la implementación, las interfaces a usarse, dirección IP, y *Gateway*.

Tabla 3.1 Identificación de dispositivos en la topología de red implementada.

Nomenclatura	Función	Interfaces en uso	Conexión con:	Dirección IP	Gateway
R1	Enrutar	f 0/0	*Switch*	192.169.56.6	N/A
		f 0/1	Servidor	192.160.45.1	N/A
		f 2/0	Internet		N/A
		s 1/0	R2	10.1.1.1	N/A
R2	Enrutar	f 0/0	*Switch*	192.169.56.7	N/A
		f 0/1	Cliente	192.60.5.1	N/A
		f 2/0	Internet		N/A
		s 1/0	R1	10.1.1.2	N/A
Switch	Conexión	e 0	Controlador	N/A	N/A
		e 1	R1	N/A	N/A
		e 2	R2	N/A	N/A
Servidor		e 0	R1	192.160.45.5	192.160.45.1
Cliente		e 0	R2	192.60.5.5	192.60.5.1
Controlador	Configuración	e 0	*Switch*	192.169.56.4	192.169.56.1

El armado de la topología y el uso de los dispositivos detallados en la **Tabla *3.1*** se evidencian en la Figura 3.12. El nodo Controlador detallado como ***"AnsibleControlador-1"*** mantiene la comunicación entre los enrutadores (R1, R2) dentro de la subred **192.169.56.0/24**, para lo cual se hace uso de un *switch* al cual se le identifica con la nomenclatura **"*Switch1*".** Esto es necesario para la configuración inicial de los enrutadores. Específicamente, para el despliegue de configuración OSPF mediante Ansible.

Por otra parte, el enrutador 1 denominado **"R1"** mantiene una conexión serial con **"R2",** el cual conecta la subred **192.160.45.0/24** del servidor hacia una nube NAT que permite mantener una conexión a Internet desde el simulador GNS3. El mismo escenario se plantea para el enrutador 2, a diferencia de que este enrutador conecta la subred 192.60.5.0/24 de cliente con Internet.

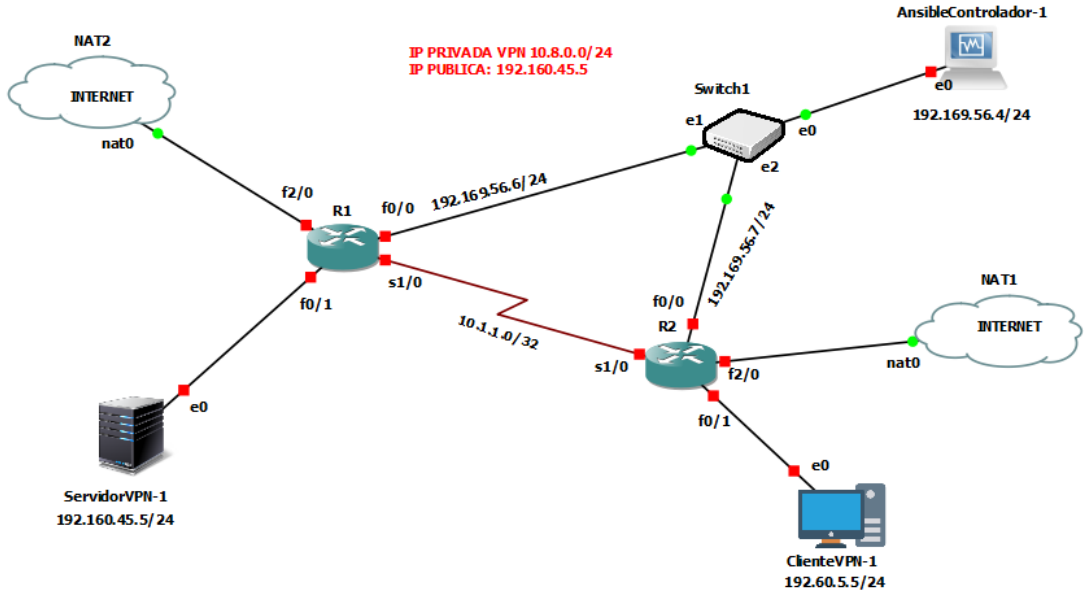

Figura 3.12 Topología de Red

Comunicación SSH controlador-enrutadores

Ansible usa un inventario de *hosts* ubicado en la ruta */etc/ansible/hosts*, dentro de este archivo se detallan los elementos a ser controlados. La declaración de este inventario comienza con la división de secciones a través de corchetes, seguido se especifica el nombre con el cual Ansible identificará el host, tal como se puede apreciar en la Figura 3.13.

Dentro de una sección también es posible declarar las variables que se usarán para el control de los elementos, estas variables pueden constituir ciertos parámetros como: nombre de usuario, contraseña, el tipo de dispositivo, etc.

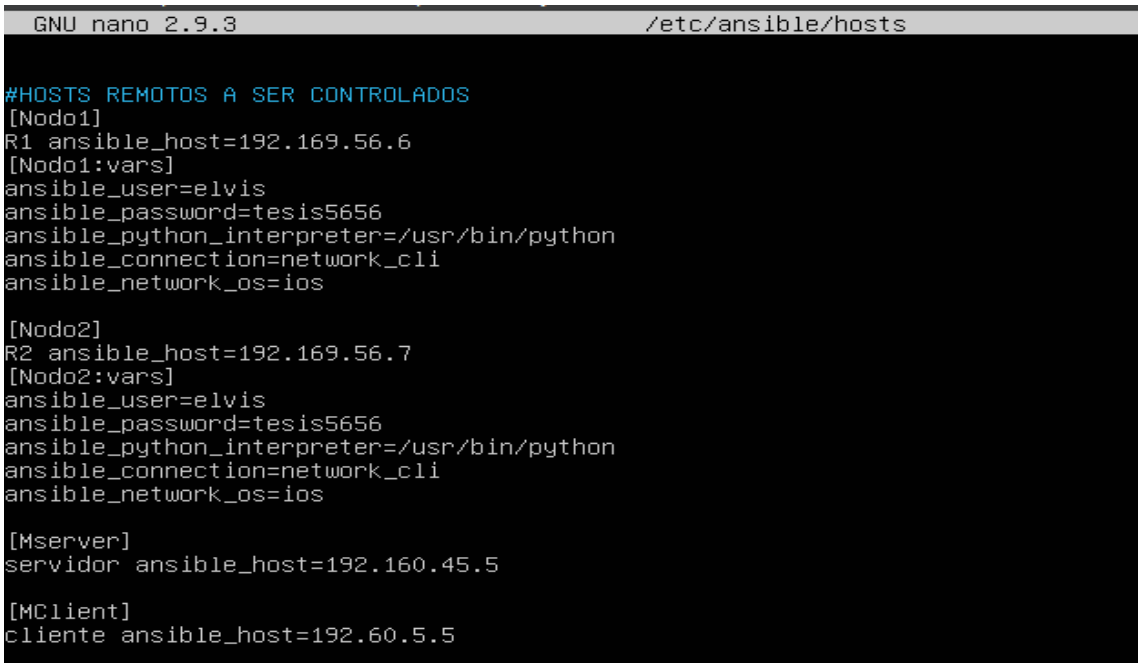

Figura 3.13 Inventario de *Hosts*

En la Figura 3.14, se presenta el cambio de parámetros del archivo *ssh_config* contenida en la ruta */etc/ssh/ssh_config*, con el fin de establecer una comunicación SSH sin errores. En primer lugar se eliminó el comentario de la línea ***ciphers aes128-ctr,aes192-ctr,aes256-ctr,aes128-cbc,3des-cbc*** para detallar los algoritmos de cifrado seguro que puede usar Ansible para enviar datos a través de la red, de igual manera se añadió la línea ***KexAlgorithms +diffie-hellman-group1-sha1*** para permitir el establecimiento de una conexión segura entre dos pares. El algoritmo Diffie-Hellman permite compartir una clave secreta entre dos dispositivos, y a través del algoritmo de *hash* ***sha1*** se da veracidad de que la clave compartida es única y segura.

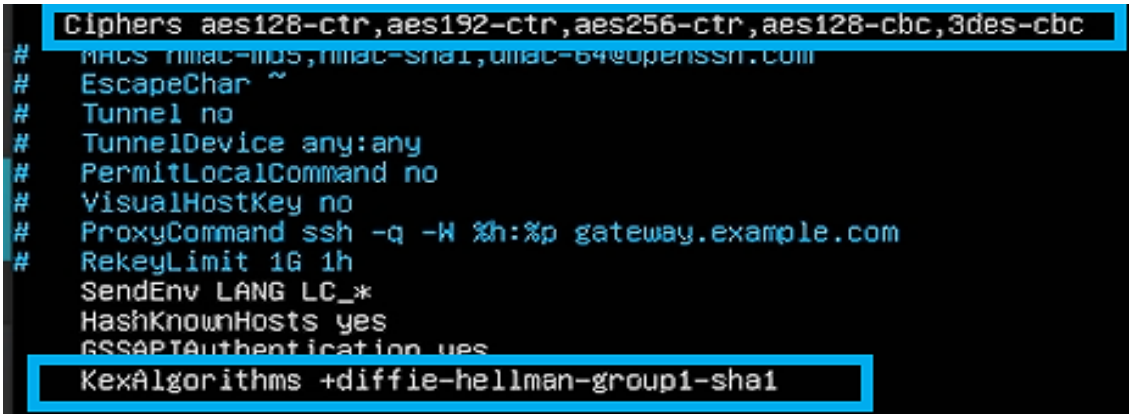

Figura 3.14 Previa comunicación SSH.

En los enrutadores, fue necesario generar una clave RSA con una longitud de 1024 bits para permitir el cifrado y descifrado de datos, así como habilitar la transmisión SSH versión 2 para permitir conexiones SSH en el dispositivo. Los comandos necesarios para el establecimiento de SSH se detallan en la Figura 3.15.

```
router(config)#ip domain-name elvis
router(config)#crypto key generate rsa
The name for the keys will be: router.elvis
Choose the size of the key modulus in the range of 360 to 2048 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1024
% Generating 1024 bit RSA keys, keys will be non-exportable...[OK]

router(config)#
*Mar  1 00:01:51.795: %SSH-5-ENABLED: SSH 1.99 has been enabled
router(config)#ip ssh version 2
router(config)#line vty 0 15
router(config-line)#login local
router(config-line)#transport input ssh
router(config-line)#exit
router(config)#username elvis privilege 15 password 0 tesis5656
router(config)#
```

Figura 3.15 Comandos para habilitar SSH en un enrutador Cisco.

El nodo controlador requiere contar con claves SSH, y una manera de conseguirlo es a través del comando ***ssh-keygen*** que por defecto genera un par de claves RSA. Las claves que se generan se almacenan en el archivo *id_rsa* contenida en la ruta ***/home/elvis/.ssh/id_rsa***. La ejecución, así como, el resultado del comando mencionado anteriormente se detalla en la Figura 3.16.

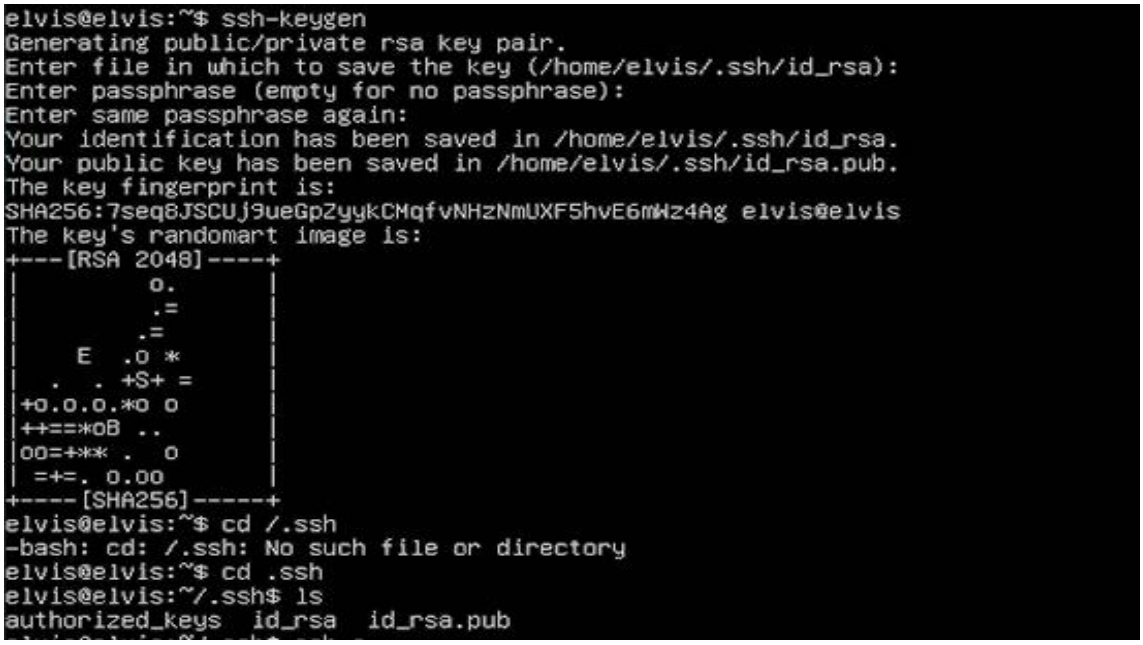

Figura 3.16 Generando claves de cifrado y descifrado

Para finalizar el proceso de conexión SSH entre el nodo controlador y los enrutadores, se realizó la copia de la clave pública SSH del nodo controlador al enrutador remoto con el comando ***ssh-copy id elvis@192.168.1.1****,* dado que permite evitar el inicio de sesión en los enrutadores cada vez que se requiera realizar una conexión SSH.

En la Figura 3.17 se detalla la ejecución del comando mencionado en el párrafo anterior, y como resultado se presenta una nota en la que se menciona que, se intente iniciar sesión en el dispositivo con el formato ***ssh elvis@[ip del dispositivo remoto].*** Siguiendo el formato presentado al ejecutar el comando ***ssh-copy id,*** se logró iniciar sesión correctamente en el dispositivo, que para el presente caso es el enrutador 1.

Es necesario recordar que la copia de la clave pública se lo realizó a todos los *hosts* contenidos en el inventario de Ansible, siendo estos: servidor, cliente, R1, R2.

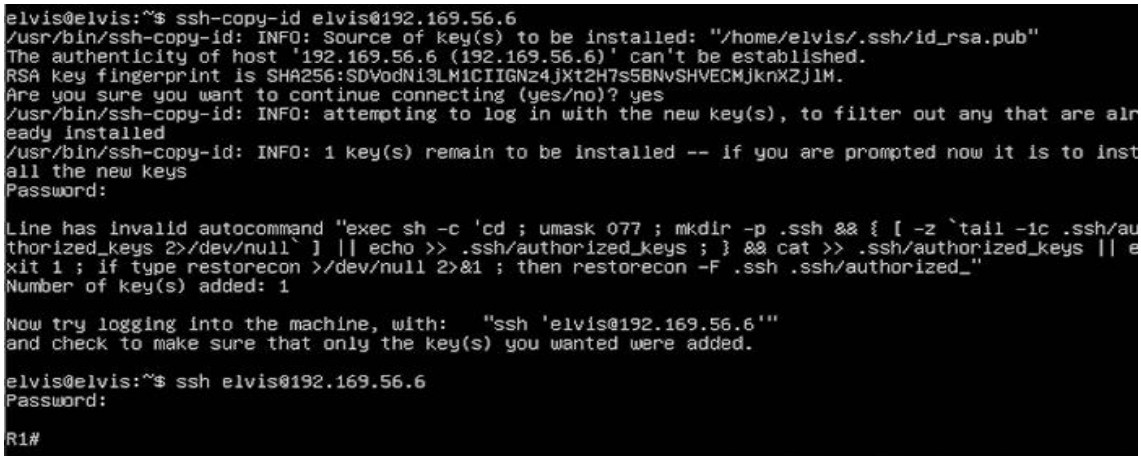

Figura 3.17 Uso de comando *ssh-copy-id*

Lo anterior comprueba el correcto funcionamiento de SSH a través de la línea de comandos, pero no garantiza que la herramienta de automatización también pueda establecer comunicación con los *hosts* remotos, por tal razón, se ejecutó el módulo *ping* de la herramienta Ansible, tal cual se lo puede observar en la Figura 3.18.

Dentro del contexto de Ansible se sabe que la conexión fue exitosa, al enviar un paquete de *ping* al *host* remoto, y este lo corresponde a través de una respuesta, a la cual se le denomina *pong,* en otras palabras, el *pong* es el opuesto del *ping* debido a que este da respuesta a una solicitud.

```
elvis@elvis:~$ ansible -m ping all
R2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
R1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
servidor | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
cliente | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

Figura 3.18 Comprobación de resultado *ping-pong*

3.3 Configuraciones mediante la herramienta de Ansible

Configuración de OSPF en el enrutador 1

Dentro del directorio raíz del nodo controlador, se redactaron los *playbooks* para el despliegue de las configuraciones, comenzando por las líneas de OSPF para el enrutador 1 y 2. La redacción del *playbook* comienza con tres guiones al inicio del documento "---", debido a que de esa manera se especifica el inicio de un documento en YAML, en la siguiente línea mediante ***gather_facts*** en su valor falso se obliga a que no se recopile información acerca del sistema del enrutador, siguiente a ello en la tercera línea se le especificó a Ansible que las tareas se ejecutaran sobre R1.

Seguido de la instrucción *task*, se definen las tareas con el comando ***name***, además se toma en cuenta que la separación de los contenidos de cada sección lleve un formato estructurado. La primera tarea que se le asigna es renombrar al host, para lo cual se

usa el módulo ***cisco.ios.ios_hostname***, dentro del cual se especifica que se configurará el *hostname* como R1, y a través de ***state: merged*** se le da la instrucción de sobrescribir los cambios en el archivo de configuración del enrutador, los comandos a usarse se los puede observar en Figura 3.19.

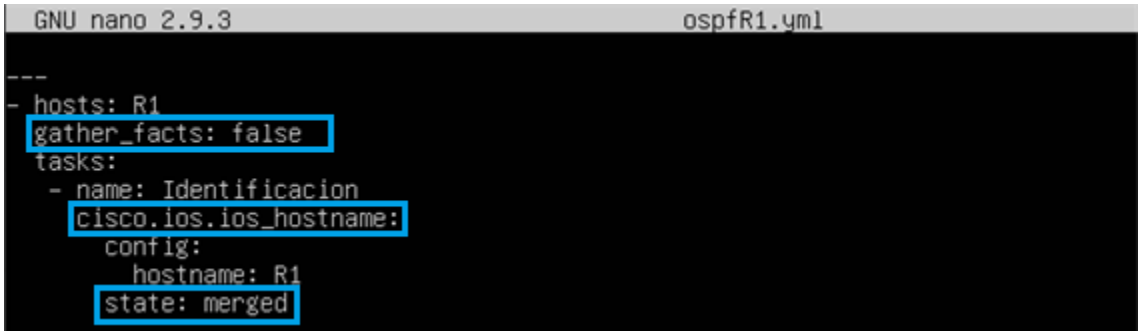

Figura 3.19 Líneas de configuración para el cambio de *Hostname*

En la Figura 3.20 se observa la implementación de dos tareas. En la primera tarea, a la cual se le identifica como "Ip en la pública" se hace uso del módulo ***cisco.ios.ios_l3_interfaces*****,** en la siguiente línea se usa el módulo *config* para dar la instrucción de despliegue de configuración IP en el serial 1/0 con la dirección IPv4 10.1.1.1 y con una máscara 255.255.255.252.

Dado que el módulo anterior no soporta *enable*, comando el cual es equivalente a un ***no shutdown*** de la CLI de cisco**,** se realiza el despliegue de una nueva tarea para la misma interfaz pero con el módulo ***cisco.ios.ios.interfaces*****.**

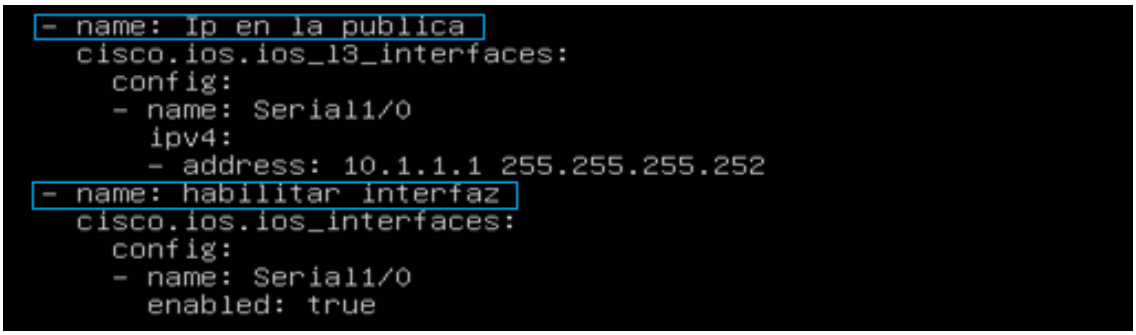

Figura 3.20 Direccionamiento WAN

En la tarea de nombre **"parámetros ospf"** mostrado en la Figura 3.21, a través del módulo de cisco ios *config*, se detalla el bloque de líneas que se ejecutarán en el dispositivo cisco. En la primera línea se especifica que se iniciara la configuración del protocolo OSPF con un *id* de proceso 1, seguido a ello se detallan las redes directamente conectadas al enrutador con el formato ***network [dirección_de_red]***

[mascara] área [número entero]. Es importante recordar que tanto el *id* de proceso como el número de área deben ser los mismos.

```
  - name: parametros ospf
    cisco.ios.ios_config:
      lines:
      - router ospf 1
      - network 192.160.45.0 255.255.255.0 area 0
      - network 10.1.1.0 255.255.255.0 area 0
      - network 192.169.56.0 255.255.255.0 area 0
```

Figura 3.21 Parámetros de configuración OSPF

Con el comando **ansible-*playbook,*** seguido del documento YAML en el cual se redactó las instrucciones para la configuración del enrutador 1 se inicia la ejecución de tareas a través de Ansible. En la última sección denominada *play recap* o resumen de ejecución, se muestra con un ***ok*** que se han completado exitosamente 4 tareas, y a través de ***change*** se presenta que durante la ejecución se han realizado 3 cambios de estado en el *host*, es decir que los parámetros se han escrito por primera vez en el archivo de configuración del enrutador 1, lo mencionado se ilustra en la Figura 3.22.

```
elvis@elvis:/$ ansible-playbook ospfR1.yml

PLAY [R1] **********************************************************************

TASK [Identificacion] **********************************************************
ok: [R1]

TASK [Ip en la publica] ********************************************************
changed: [R1]

TASK [habilitar interfaz] ******************************************************
changed: [R1]

TASK [parametros ospf] *********************************************************
[WARNING]: To ensure idempotency and correct diff the input configuration lines should be similar
to how they appear if present in the running configuration on device
changed: [R1]

PLAY RECAP *********************************************************************
R1                         : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=
0    ignored=0
```

Figura 3.22 Ejecución de *playbook ospfR1.yml*

Configuración de OSPF en el enrutador 2

La configuración del *playbook* para el enrutador 2 comienza con el mismo formato presentado en la Figura 3.19, el único parámetro que presenta cambios es el nombre del enrutador remoto definido en el inventario, así como el valor de *hostname*, observe la Figura 3.23.

```
 GNU nano 2.9.3                                  ospfR2.yml

---
- hosts: R2
  gather_facts: false
  tasks:
   - name: Identificacion
     cisco.ios.ios_hostname:
       config:
         hostname: R2
       state: merged
```

Figura 3.23 Configuración inicial del enrutador 2

Similar a las tareas del enrutador 1, se separó la configuración de la interfaz serial 1/0, así como la habilitación de la interfaz. En la primera tarea se usó el módulo *l3 interfaces* de cisco para añadir la dirección IP a la interfaz, mientras que, en la tarea 2 se usó el módulo *interfaces* para levantar la interfaz serial 1/0, lo mencionado se encuentra en la Figura 3.24.

```
  - name: Ip en la publica
    cisco.ios.ios_l3_interfaces:
      config:
      - name: Serial1/0
        ipv4:
        - address: 10.1.1.2 255.255.255.252
  - name: habilitar interfaz
    cisco.ios.ios_interfaces:
      config:
      - name: Serial1/0
        enabled: true
```

Figura 3.24 Direccionamiento WAN del enrutador R2

En la Figura 3.25 se observa las líneas para el apartado de configuración del enrutador 2 tomando en cuenta que el enrutador 1 contenía un *id* de proceso OSPF 1, y estaba ubicada en el área 0.

```
  - name: parametros ospf
    cisco.ios.ios_config:
      lines:
      - router ospf 1
      - network 192.60.5.0 255.255.255.0 area 0
      - network 10.1.1.0 255.255.255.0 area 0
```

Figura 3.25 Parámetros de configuración OSPF del enrutador 2

Una vez definido los parámetros para el levantamiento del protocolo OSPF en el enrutador 2, se ejecuta el *playbook* a través del comando digitado en la Figura 3.26, y

como resumen de ejecución se obtiene que las cuatro tareas se cumplieron exitosamente, mientras que tres presentaron cambios de estado en el enrutador 2, estos cambios se presentan a nivel de direccionamiento IP, habilitación de interfaz, y la habilitación del protocolo OSPF.

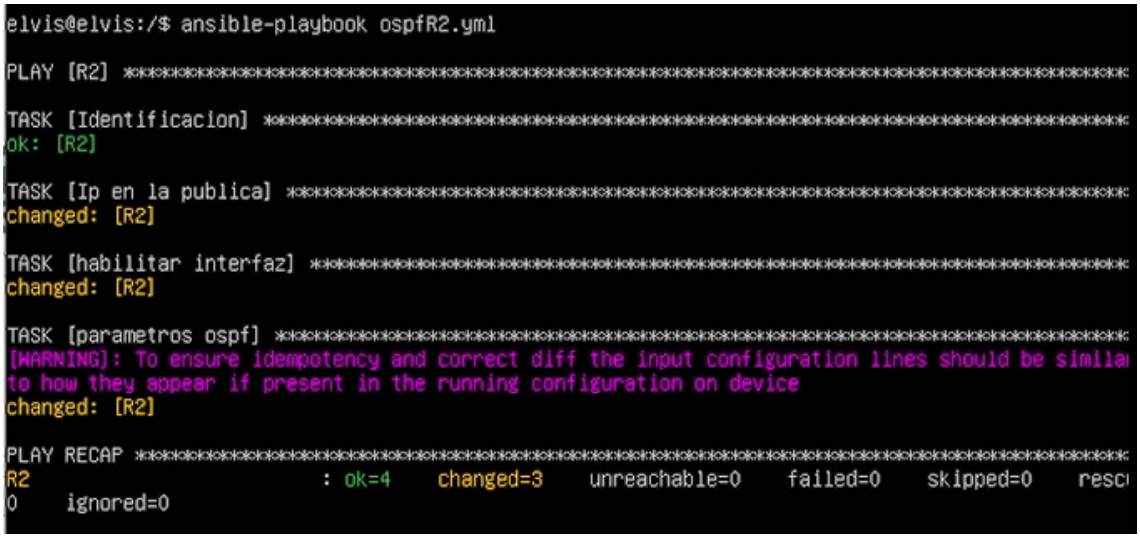

Figura 3.26 Ejecución del *playbook ospfR2.yml*

En la Figura 3.27 se aprecia los cambios que se han realizado de manera remota a través de la ejecución de los *playbooks*, como cabecera se tiene presente el usuario privilegiado el cual funciona con un nivel de acceso total 15, es decir que, el usuario que se conecte de manera remota puede realizar cambios administrativos en el equipo, el parámetro mencionado dio la posibilidad que las tareas para la sección de ***configure*** del equipo cisco se ejecutaran sin ningún tipo de problemas.

Para el enlace WAN entre lo enrutadores se usó la interfaz serial 1/0 con la red 10.1.1.0 de mascara 255.255.255.252, dando así la posibilidad de usar solamente dos direcciones IP, siendo asignada la dirección 10.1.1.1 a la interfaz serial del enrutador 1, mientras que 10.1.1.2 al enrutador 2.

Para la configuración del protocolo OSPF se mantuvo los mismos parámetros tanto en el enrutador 1 así como en el 2, el único cambio realizado por parte de Ansible fue la especificación de las redes conectadas a cada enrutador, todo lo comentado anteriormente se muestra como parte del resultado de la ejecución de los *playbooks* de manera remota en la Figura 3.27.

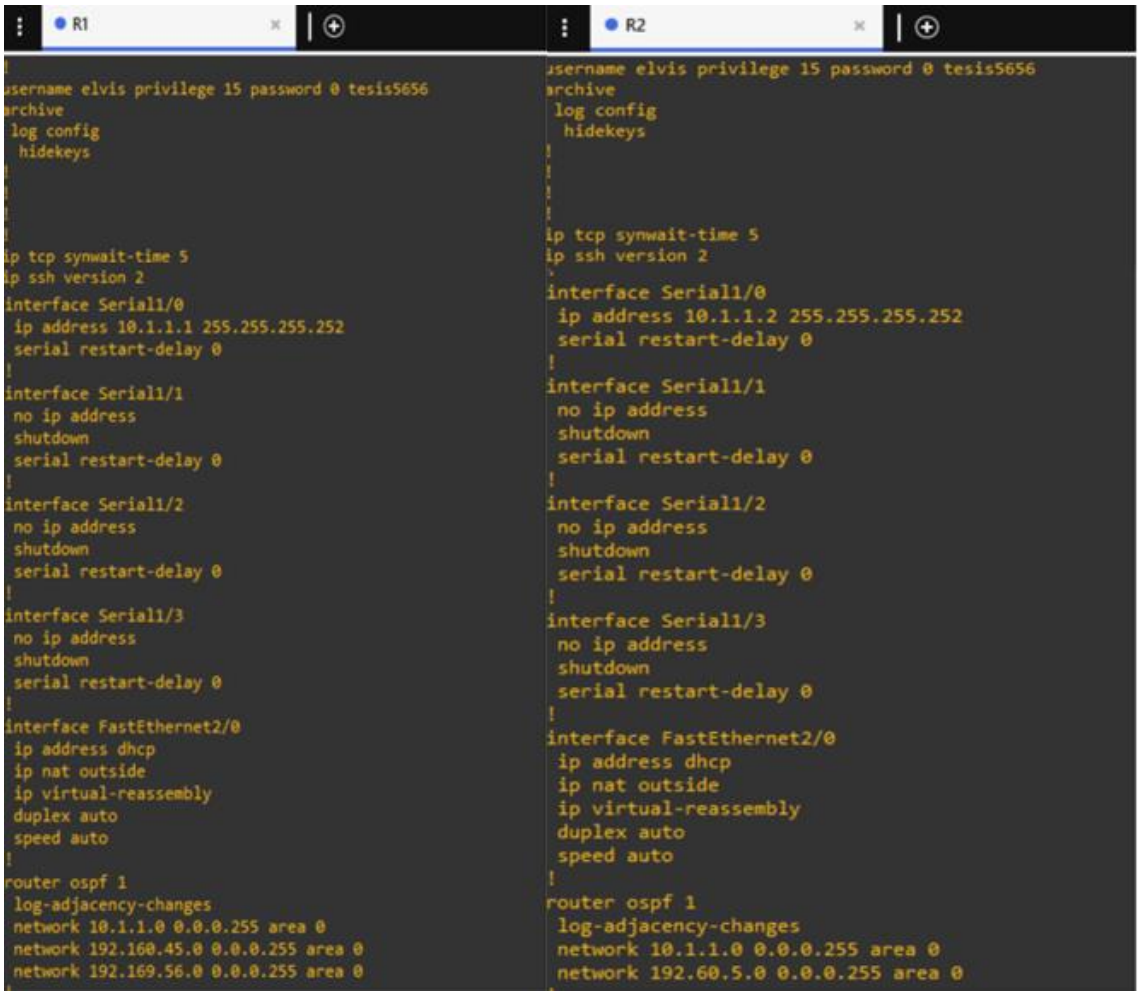

Figura 3.27 Cambios de configuración en los enrutadores

Configuración de Servidor VPN

Similar a anteriores *playbooks,* se inició la redacción del documento YAML con "---"**,** siguiente a ello se definió el *host* en el cual se ejecutará el *playbook,* que para este caso es el servidor, en la siguiente línea con ***become: yes*** se menciona que las tareas se lo realizarán con privilegios administrativos, mientras que, con ***become_method: sudo*** se hace referencia a que se usará el usuario sudo para el despliegue de lo mencionado.

En la Figura 3.28, se establecen dos tareas, siendo estas la actualización de repositorios y la instalación de *wireguard*. En la primera tarea se usa el módulo APT con las instrucciones ***update_cache: true*** para actualizar los paquetes de los repositorios, seguido de ***cache_valid_time: 3600*** comando con el cual se verifica que los repositorios fueron actualizados hace una hora, en caso de no ser así los actualiza nuevamente, y con ***force_apt_get: true*** se insta que durante la ejecución del *playbook* se use *apt-get* en vez de APT, debido a que el primer comando maneja los paquetes de mejor manera.

En la segunda tarea con el módulo APT se despliega las instrucciones para la instalación de *wireguard*, identificando con el comando ***name*** el paquete a instalarse**,** mientras que en la siguiente línea con ***state: present*** se indica la condición de que, si el paquete *wireguard* no existe, se realizará la instalación del mismo, pero en caso de ya tener instalado el paquete no se realizará una nueva instalación, y por último con ***update_cache: yes*** se actualiza nuevamente los repositorios.

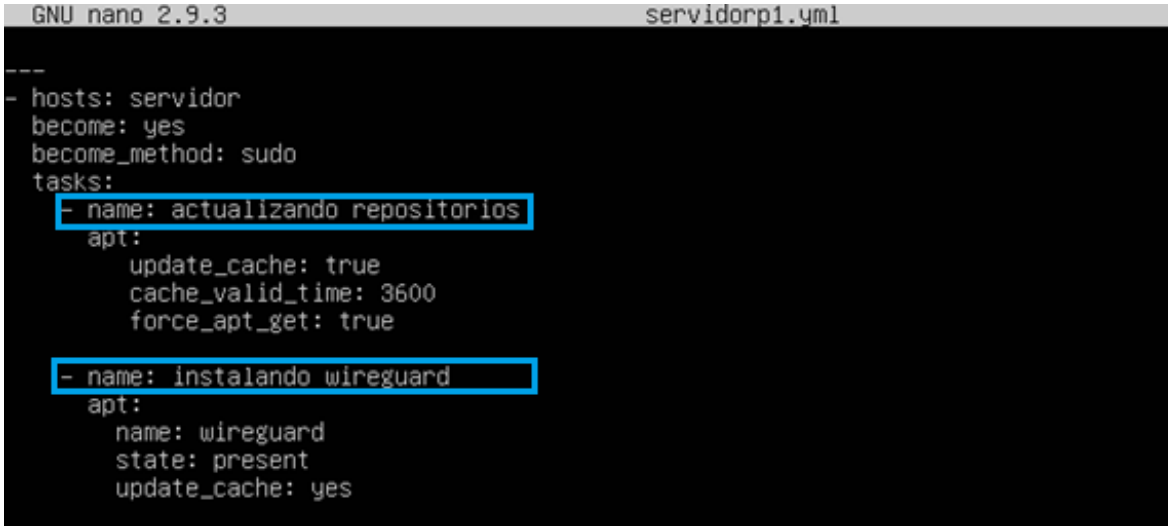

Figura 3.28 Actualización de repositorios e instalación de *wireguard*.

La Figura 3.29, contiene el detalle de instrucciones de tres tareas, siendo la primera la generación de claves públicas y privadas con ***shell: wg genkey | tee privatekey | wg pubkey > publickey*****,** comando que unifica tres instrucciones, es decir que a través de *wg genkey* se genera una clave privada y se almacena en un archivo de nombre *privatekey*, mientras que la clave pública se construye en base a la privada, y dicha clave se almacena en el archivo *publickey*.

Después en la tarea dos con el fin de verificar, se ejecutó el módulo *shell* con la instrucción ***cat privatekey*** que permite visualizar el contenido del archivo *privatekey*, la salida de la instrucción anterior se lo almacena en la variable ***command_output*****,** mientras que con *debug* se imprime dicha salida durante la depuración del *playbook*.

De igual manera en la tercera tarea, mediante el módulo *slurp* se almacenó el contenido del archivo *privatekey* en una variable de nombre ***private*****.**

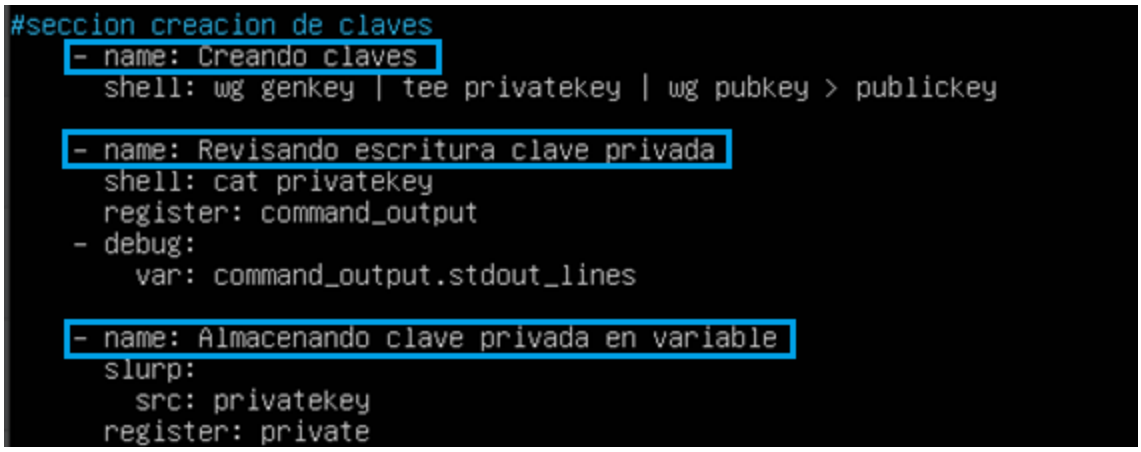

Figura 3.29 Uso de *genkey* para la generación de claves.

En la Figura 3.30 se observa las tareas empleadas para el despliegue de configuración de la interfaz wg0.

En la primera tarea, se utilizó el módulo ***file*** para crear el archivo de configuración denominado ***wg0.conf***. Mediante ***path*** se especifica ***/etc/wireguard/wg0.conf*** como la ruta en la cual se creará el archivo**.** La instrucción ***state: touch*** verifica la existencia del archivo para evitar la sobreescritura de líneas. En otras palabras, se asegura que ***wg0.conf*** solamente sea creado en caso de no existir dentro la ruta especificada.

En la segunda tarea se usó el módulo ***blockinfile*** para agregar varias líneas de comandos dentro de un archivo especificado. Haciendo uso del comando ***path*** se especifica la ruta del archivo de escritura, y utilizando el parámetro ***block,*** seguido de "**|**" se inicia la redacción de las líneas de configuración para el adaptador wg0 especificando los siguientes puntos:

Al inicio del archivo de configuración, se añade el formato **[*Interface*]** para indicar que cualquier cambio se lo realice en la interfaz.

- ***Privatekey:*** la clave privada necesaria para el descifrado de los mensajes de los clientes es leído de la variable *private* y de igual manera decodificada debido a que slurp por defecto codifica el mensaje cuando almacena contraseñas.
- ***Address:*** Se especifica la dirección IP para la interfaz wg0, también se menciona que a través de la interfaz se permitirá el tráfico con direcciones IP 10.8.0.x, siendo x un número entre 2 y 254.
- ***SaveConfig:*** con el valor en *true* se da la propiedad de guardar los cambios realizados en la interfaz.

- ***Post Up/Down*:** con *postup* se establecen las reglas para permitir el tráfico mientras este activa la interfaz, y a través de *postdown* dichas reglas se eliminan una vez detenida la interfaz.
- ***ListenPort:*** Se habilita el puerto 51820 para el tráfico *wireguard*.

En la tarea tres se levanta la interfaz wg0 utilizando los parámetros descritos anteriormente y haciendo uso del comando ***command: wg-quick up wg0.*** En la última tarea, se imprime en la pantalla del usuario controlador, la clave pública que genera el servidor *wireguard*.

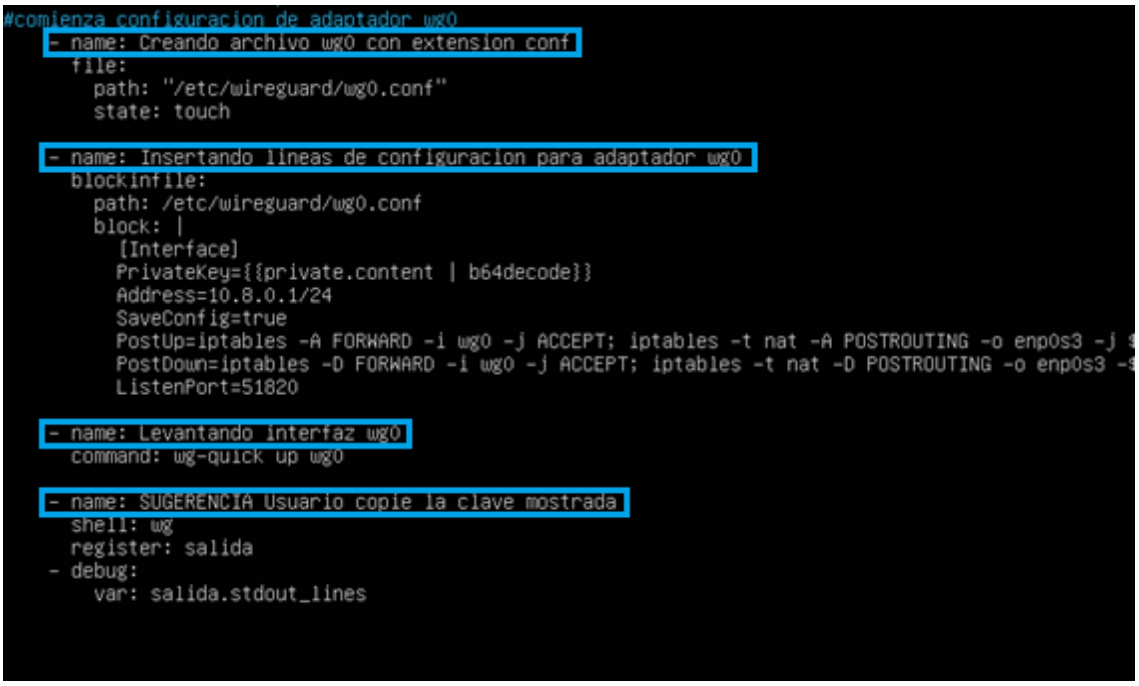

Figura 3.30 Configuración del adaptador wg0

En la Figura 3.31, se observa los resultados obtenidos al ejecutar el *playbook* de configuración del servidor VPN. La ejecución del *playbook* se inicia siguiendo el formato del comando ***ansible-playbook –ask-become-pass [archivo.yml].*** Haciendo uso del formato se especifica que se ejecutará un *playbook* con tareas privilegiadas, y se le proporcionara una contraseña para hacer uso del usuario sudo en el *host* remoto, que en este caso es el servidor. Por tal razón previo a la ejecución del *playbook* se solicita la contraseña.

Además, se observa en el resumen de ejecución que se han realizado seis cambios de estado en el servidor y se han logrado completar todas las tareas del *playbook*.

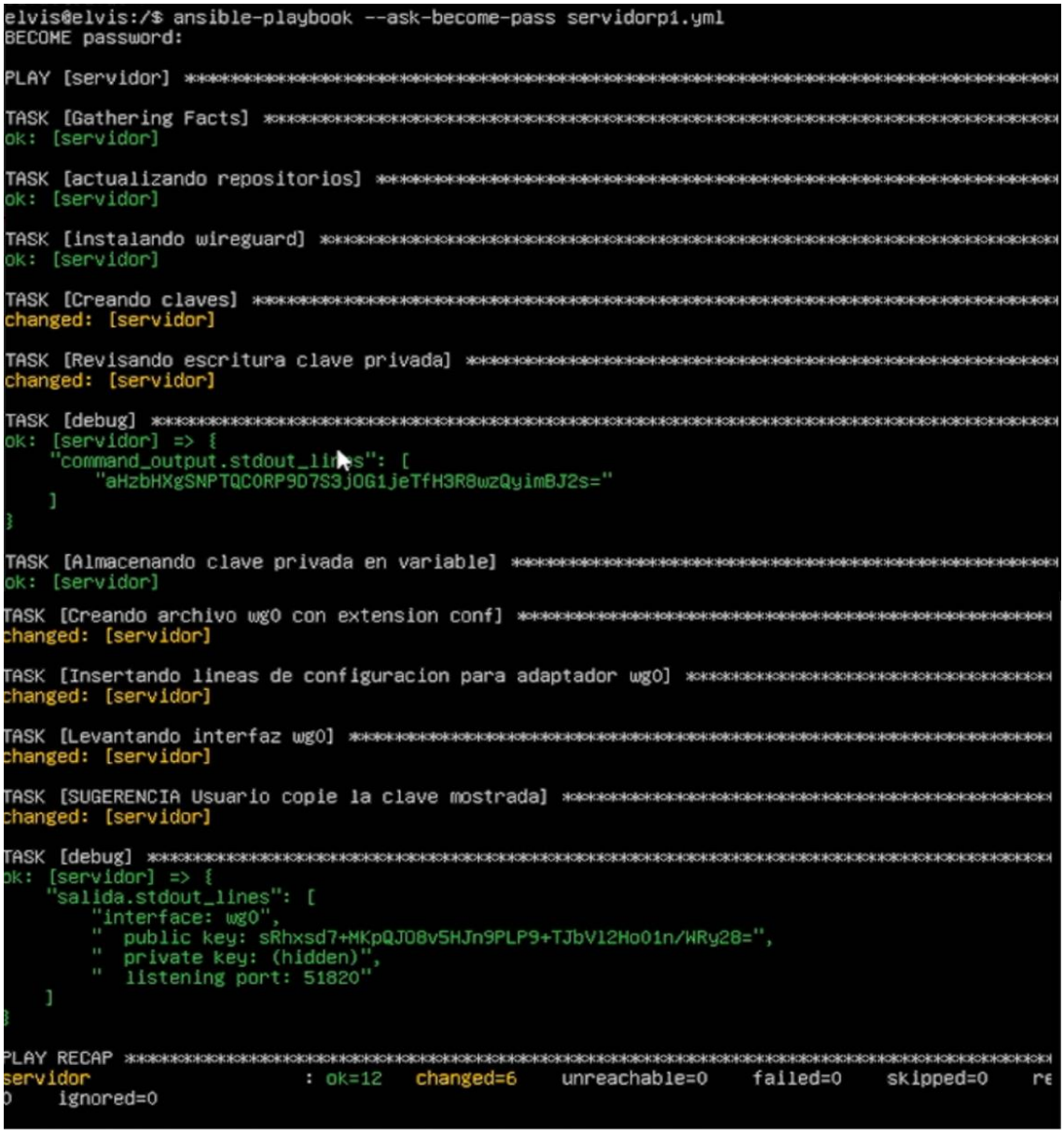

Figura 3.31 Ejecución del *playbook servidorp1.yml*

Completado la ejecución de tareas por parte de Ansible, se realizó la verificación del funcionamiento de la interfaz wg0. Como se puede apreciar en la Figura 3.32, se detalla la clave pública a través del cual los clientes podrán conectarse al túnel *wireguard*, y de igual manera se puede notar que el puerto en el cual se escucha el tráfico para wg0 es el 51820.

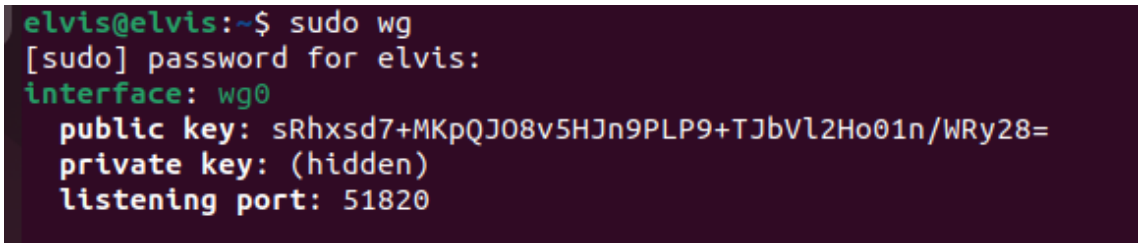

Figura 3.32 Verificación del estado de la interfaz wg0

Configuración de Cliente VPN

En la configuración del cliente, los pasos ejecutados para el despliegue del mismo son similares a los implementados en la configuración del servidor. En la Figura 3.33 se observa que a inicios del *playbook* se especifica el *host* sobre el cual correrán las configuraciones, además mediante ***become*** se especifica que se usará el método sudo para ejecutar las tareas de Ansible en modo privilegiado.

En la primera tarea se realiza la actualización de repositorios contenidos en la máquina cliente mediante el módulo APT, y se especifica el uso de ***apt-get*** como herramienta de manejo de paquetes, mientras que, en la segunda tarea se procede a la instalación de *wireguard*, para lo cual se especifica con ***name,*** que el paquete a instalar será el protocolo mencionado anteriormente, y seguido a ello se realizará la actualización de repositorios.

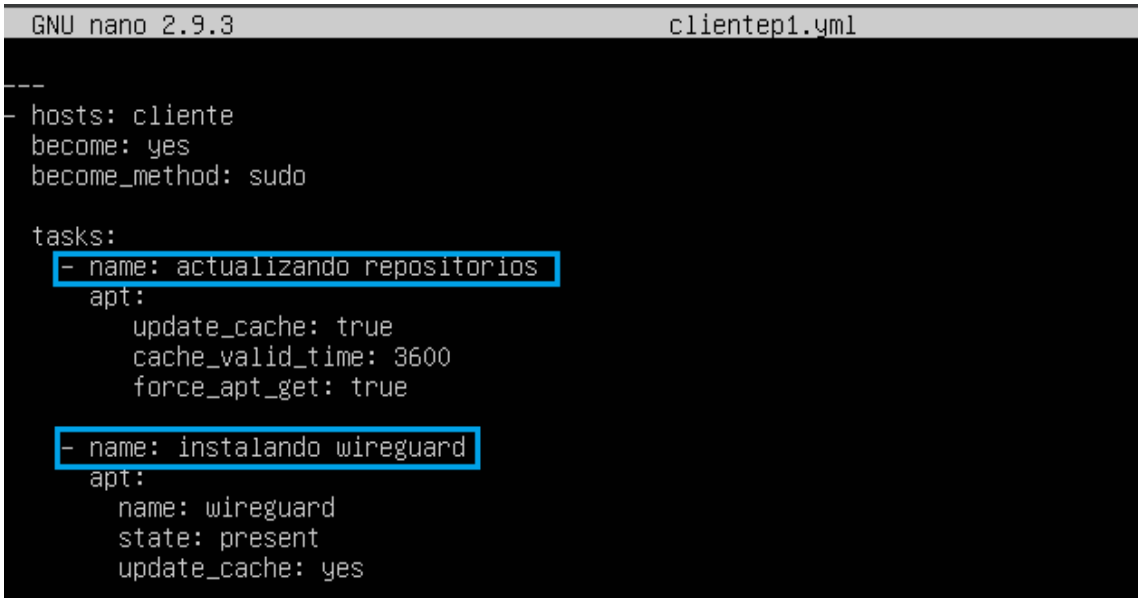

Figura 3.33 Actualización de repositorios del cliente

Seguido se realizó la escritura de 3 tareas, detallándose como primera, la creación de claves con el módulo ***shell*****,** para lo cual se usó la misma línea de instrucción especificada para la creación de claves del servidor, obsérvese la Figura 3.29 en la cual se detalla lo mencionado anteriormente.

Generadas las claves, en la tarea 2 se verifica la creación de clave privada a través del registro de salida del comando ***cat privatekey***. Es decir, que el resultado que se genera al momento de ejecutar el comando en el *host* remoto cliente, este es almacenado en la variable *command_output,* para luego con un módulo ***debug*** presentar el contenido de dicha variable en la pantalla del administrador durante la ejecución del *playbook*.

Posterior a la verificación del cumplimiento de la tarea 2, se almacenó la clave privada con el módulo slurp, para lo cual fue necesario indicar la fuente del contenido, que en este caso es el archivo *privatekey*, y la variable de registro al cual se le denominó ***private*****,** todo lo mencionado anteriormente se presenta en la Figura 3.34.

```
#seccion creacion de claves
    - name: Creando claves
      shell: wg genkey | tee privatekey | wg pubkey > publickey

    - name: Revisando escritura clave privada
      shell: cat privatekey
      register: command_output
    - debug:
        var: command_output.stdout_lines

    - name: Almacenando clave privada en variable
      slurp:
        src: privatekey
      register: private
```

Figura 3.34 Creación de claves de cliente

En la Figura 3.35, se muestra las tareas empleadas para la configuración de la interfaz wg0. En la primera tarea, se observa el módulo usado para la creación del archivo de configuración ***wg0.conf***, el cual debe estar contenido dentro del directorio *wireguard*, por lo que se usó el módulo ***file*** con el comando ***path*** para indicar que el archivo mencionado anteriormente debe alojarse en la ruta ***/etc/wireguard/wg0.conf.***

En la siguiente tarea se muestran los parámetros usados para la configuración de la interfaz *wg0*. La escritura de estos parámetros se lo realizó a través del módulo ***blockinfile*****,** dentro del cual se usó el comando ***block*** para añadir las siguientes líneas de configuración:

Al iniciar la redacción se añade **[Interface]** para indicar que las líneas redactadas debajo de este serán aplicadas a la interfaz **wg0** del cliente.

- ***PrivateKey:*** La clave privada usada para el descifrado de mensajes entrantes a la interfaz se lee desde la variable definida en la Figura 3.34, a la cual se le denominó ***private*****.** Para leer el contenido de la variable se usó el parámetro ***content*** seguido de la instrucción ***b64decode*** con el fin de revertir el proceso de codificación que realiza el módulo **slurp.**
- ***Address:*** Con la presente línea se especifica que la interfaz wg0 del cliente será identificada con la IP 10.8.0.2/24, y de igual manera se especifica que en la interfaz se admite solamente las direcciones IP 10.8.0.X entrantes.
- ***SaveConfig:*** A través de la línea en modo ***true*** se le especifica que los cambios realizados sean guardados para la interfaz.

- ***ListenPort:*** El puerto de escucha para la recepción de datos será el 51820.

Seguido se detalla mediante la línea **[*Peer*]** el inicio de las configuraciones para acceder al servidor *wireguard* a través de la interfaz **wg0.**

- ***Publickey:*** El *host* cliente enviará el tráfico a través de la interfaz que se está configurando actualmente, esta información de envio debe ser cifrada con la clave pública del servidor para mantener la privacidad de la información. También es importante considera que en caso de que la información no sea cifrada con la clave pública del servidor, este no comprenderá el contenido del mensaje y no podrá tomar las acciones de enrutamiento, por ello como parámetro de la línea de configuración se le especifica la clave pública del servidor a través del cual se realizará la operación de cifrado.
- ***AllowedIPs:*** Se especifica que cualquier dirección IP saliente o tráfico del cliente, sea redirigido por la interfaz wg0, debido a que esta interfaz mantendrá la conexión a través de un túnel con el servidor VPN *wireguard*.
- ***Endpoint:*** La pareja del *host* cliente necesita ser identificado por una dirección IP pública, es decir que es necesario conocer la IP a través del cual se maneja el tráfico sin escenario de una VPN, por tal razón se le especifica la dirección 192.160.45.5 mediante el cual se le identifica al Servidor, así como el puerto de escucha 51820 el cual fue configurado en la interfaz wg0 del servidor.

La última tarea presente en la Figura 3.35, muestra la clave pública del cliente en pantalla. Esto debido a que el administrador necesita conocer esta clave para añadir los clientes al servidor; para ello mediante ***register*** se captura la información que arroja el comando ***cat publickey*** y se lo almacena en la variable salida, posterior a ello con el módulo ***debug*** se muestra de la variable ***salida*** el contenido almacenado en ella durante la ejecución de la tarea.

```
#comienza configuracion de adaptador wg0
    - name: Creando archivo wg0 con extension conf
      file:
        path: "/etc/wireguard/wg0.conf"
        state: touch

    - name: Insertando lineas de configuracion para adaptador wg0
      blockinfile:
        path: /etc/wireguard/wg0.conf
        block: |
          [Interface]
          PrivateKey={{private.content | b64decode}}
          Address=10.8.0.2/24
          SaveConfig=true
          ListenPort=51820
          [Peer]
          Publickey=sRhxsd7+MKpQJO8v5HJn9PLP9+TJbVl2HoO1n/WRy28=
          AllowedIPs=0.0.0.0/0
          Endpoint=192.160.45.5:51820

    - name: Copie su llave publica
      command: cat publickey
      register: salida
    - debug:
        var: salida.stdout_lines
```

Figura 3.35 Configuración del adaptador wg0 en el cliente

En la Figura 3.36, se contrasta los resultados obtenidos después de la ejecución del *playbook* "clientep1.yml". Dentro de la sección correspondiente al resumen de tareas ejecutadas, se obtiene que se han finalizado correctamente 11 tareas, y han existido 6 cambios de estado en la máquina del cliente, siendo estos cambios, la creación, verificación, y almacenamiento de claves, así como la configuración del archivo wg0 ubicado en la ruta ***/etc/wireguard/***.

```
elvis@elvis:/$ ansible-playbook --ask-become-pass clientep1.yml
BECOME password:

PLAY [cliente] *****************************************************************************************

TASK [Gathering Facts] *********************************************************************************
ok: [cliente]

TASK [actualizando repositorios] ***********************************************************************
ok: [cliente]

TASK [instalando wireguard] ****************************************************************************
ok: [cliente]

TASK [Creando claves] **********************************************************************************
changed: [cliente]

TASK [Revisando escritura clave privada] ***************************************************************
changed: [cliente]

TASK [debug] *******************************************************************************************
ok: [cliente] => {
    "command_output.stdout_lines": [
        "aCKnAwWtREcas5P1RHfACS8qjm5OR7i+56TH+at4OEs="
    ]
}

TASK [Almacenando clave privada en variable] ***********************************************************
changed: [cliente]

TASK [Creando archivo wg0 con extension conf] **********************************************************
changed: [cliente]

TASK [Insertando lineas de configuracion para adaptador wg0] *******************************************
changed: [cliente]

TASK [Copie su llave publica] **************************************************************************
changed: [cliente]

TASK [debug] *******************************************************************************************
ok: [cliente] => {
    "salida.stdout_lines": []
}

PLAY RECAP *********************************************************************************************
cliente                    : ok=11   changed=6    unreachable=0    failed=0    skipped=0    rescued=
0    ignored=0
```

Figura 3.36 Ejecución del *playbook clientep1.yml*

En la Figura 3.37, se observa la ejecución del comando ***sudo wg0*** en el *host* cliente, y como resultado se aprecia que la interfaz configurada por Ansible se ha realizado correctamente. Como parámetros de la interfaz se observa que se han establecido una clave pública para el cliente, así como una clave privada, la cual se encuentra oculta, también se aprecia que el puerto de escucha es el 51820.

Además, dentro de la configuración se aprecia que se ha detallado la clave pública del par servidor. Resaltado en color celeste se observa que se ha empezado a enviar tráfico, pero todavía no existe respuesta por parte del servidor, esto es debido a que en el servidor aún falta añadir la identificación del cliente.

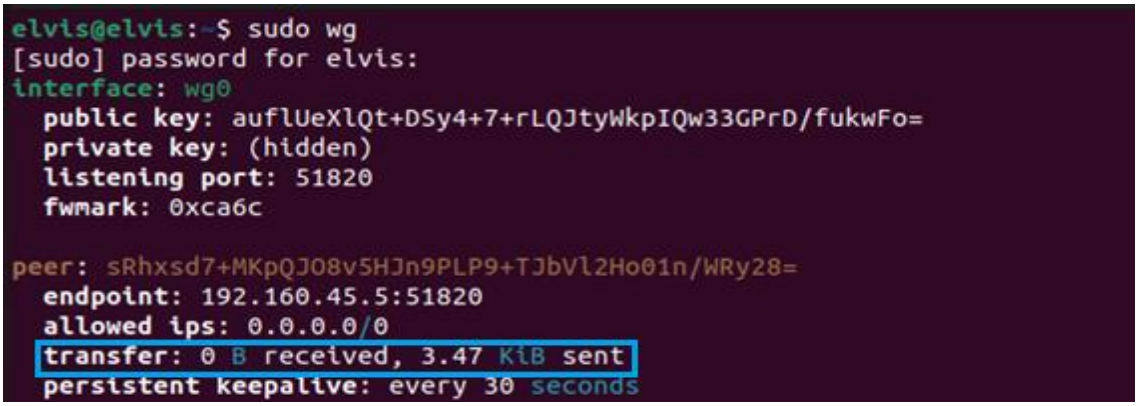

Figura 3.37 Verificación del estado de la interfaz wg0 del cliente

Conexión de clientes a túnel *wireguard*

El servidor VPN requiere conocer la clave pública del cliente para hacer llegar el tráfico a su interfaz, y que el contenido de los mensajes sea descifrado correctamente. En la Figura 3.38, se detallan las líneas de configuración que debe seguir Ansible cada vez que se requiera añadir un par cliente al servidor.

La redacción inicial del *playbook* se lo realizo de manera similar a los anteriores, es decir que las tareas se ejecutaran en el servidor con privilegios de superusuario. Seguido a ello se define la opción de Ansible ***vars_prompt*** el cual da la posibilidad de definir variables a usar. Para este escenario se especificó que se debe definir una variable de nombre **ccpública,** con el cual se hace referencia a la clave pública del cliente, seguido a ello se establece el comando ***prompt*** con un mensaje que se presentará en la pantalla del administrador indicando que debe ingresar la clave pública del cliente a añadir, la opción ***private*** en valor **no,** da la instrucción de que Ansible no oculte la entrada del usuario con "******", debido a que un error en la escritura de la clave pública no permitirá la comunicación entre el cliente y el servidor.

Dentro de la opción ***task*** de Ansible se establecen tres tareas, siendo la primera la utilización del módulo ***shell*** con el comando ***wg set wg0 peer {{ccpública}} allowed-ips 10.8.0.2/32*****,*** mencionando con ***wg set*** que se realizará la configuración de la interfaz **wg0** para añadir un par, seguido se añade la clave pública del par el cual es leído desde la variable **ccpública** definida en la sección de ***vars_prompt*****,** seguido a ello se establece que estrictamente se permita la dirección IP 10.8.0.2 por tal razón se establece una máscara de 32 bits.

En la segunda tarea se realiza la modificación del archivo ***sysctl.conf*** para permitir el intercambio de tráfico entre interfaces, es decir que el tráfico entrante a wg0 será direccionado a la interfaz real que contiene la dirección pública del servidor. Para llevar a cabo lo mencionado anteriormente se usó el módulo ***lineinfile,*** que permite añadir una línea en un archivo especificado, a través del comando ***path*** se menciona que el archivo a modificar se encuentra en la ruta ***/etc/systcl.conf*****,** y a través de ***line*** se establece que en el archivo se añada la línea ***net.ipv4.ip_forward=1*****.**

En la última tarea se aplican los cambios realizados en el archivo ***sysctl.conf*** por tal razón se usó el módulo ***shell*** con el comando ***sysctl -p*****,** el cual permite aplicar los cambios de manera inmediata durante el funcionamiento del sistema operativo**.** Todo lo mencionado anteriormente se lo puede observar en la Figura 3.38.

```
 GNU nano 2.9.3                                  addclient.yml

---
- hosts: servidor
  become: yes
  become_method: sudo
  vars_prompt:
    - name: ccpublica
      prompt: Ingrese clave publica del cliente
      private: no

  tasks:
    - name: Agregando al cliente wireguard
      shell: wg set wg0 peer {{ccpublica}} allowed-ips 10.8.0.2/32

    - name: Modificando sysctl
      lineinfile:
        path: /etc/sysctl.conf
        line: net.ipv4.ip_forward=1

    - name: Guardadno cambios
      shell: sysctl -p
```

Figura 3.38 Parámetros para añadir clientes

módulo *playbook* de Ansible. Previo a la ejecución de tareas, la herramienta solicita al usuario como entrada, la clave pública del cliente, parámetro el cual debe estar escrito

adecuadamente. En la Figura 3.39, se observa el resumen de los resultados obtenidos, pudiéndose destacar que han existido dos cambios de estado en el servidor, siendo estos, la adición del cliente al túnel *wireguard* y la aplicación de cambios al archivo *sysctl*.

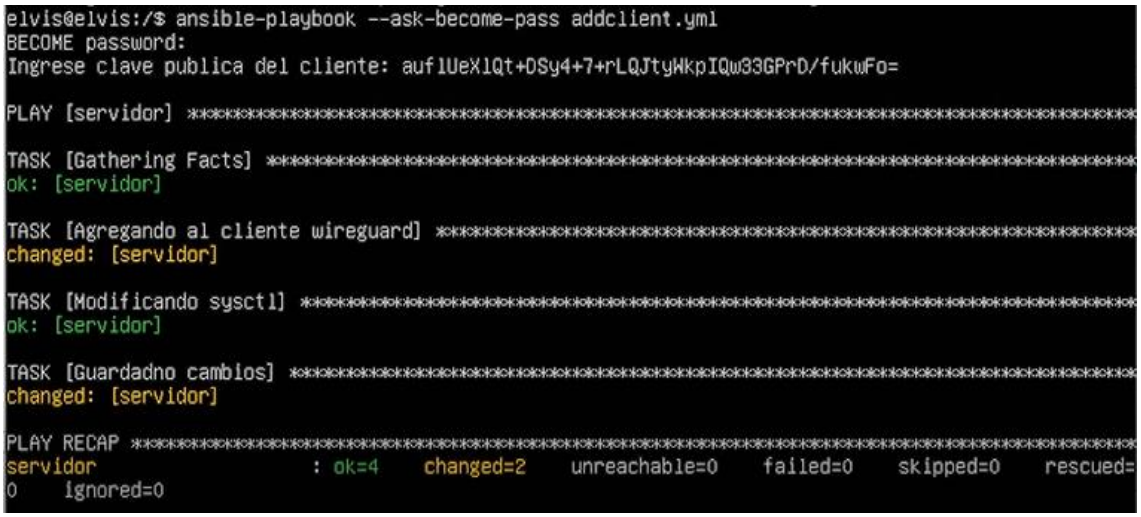

Figura 3.39 Ejecución del playbook addclient.yml

Ejecutado el archivo *addclient.yml* con Ansible, se procede a verificar nuevamente el estado de la interfaz wg0 del servidor tal como se lo evidencia en la Figura 3.40, y se puede apreciar que los parámetros de clave pública, privada, y puerto de escucha mostrados en la Figura 3.32 siguen presentes, por lo que se comprueba el funcionamiento del comando ***saveconfig=true*** definido en la Figura 3.30, de igual manera se aprecia el efecto del comando ejecutado a través del *playbook*, debido a que se muestra la clave pública del par cliente, la dirección IP pública del cliente, así como el puerto de escucha del mismo. Resaltado en color celeste se evidencia que la comunicación entre servidor y cliente empieza a funcionar.

Figura 3.40 *Peer Wireguard*

3.4 Pruebas de funcionamiento y verificación de resultados obtenidos

Prueba de *ping* hacia el servidor

En la Figura 3.41, se observa resaltado de un color verde el terminal del servidor VPN. Dentro de este terminal se evidencia los resultados que arroja la entrada del comando ***sudo wg***, lo importante a resaltar en la terminal es la sección de ***latest handshake***, debido a que este parámetro brinda información sobre el tiempo de intercambio de mensajes entre servidor y cliente, es decir que cada vez que el servidor llegue a autenticar al cliente, el momento de la autenticación quedará registrada en el parámetro de ***handshake,*** tal es el caso que se observa en la siguiente figura, donde el servidor ha realizado una autenticación hace 1 minuto y 20 segundos. Con lo mencionado anteriormente se verifica la conexión exitosa entre el cliente-servidor y viceversa.

En la misma Figura 3.41, se aprecia resaltado en un recuadro de color celeste el uso del comando *ping* en el *host* cliente con el cual se verificó una respuesta por parte del servidor.

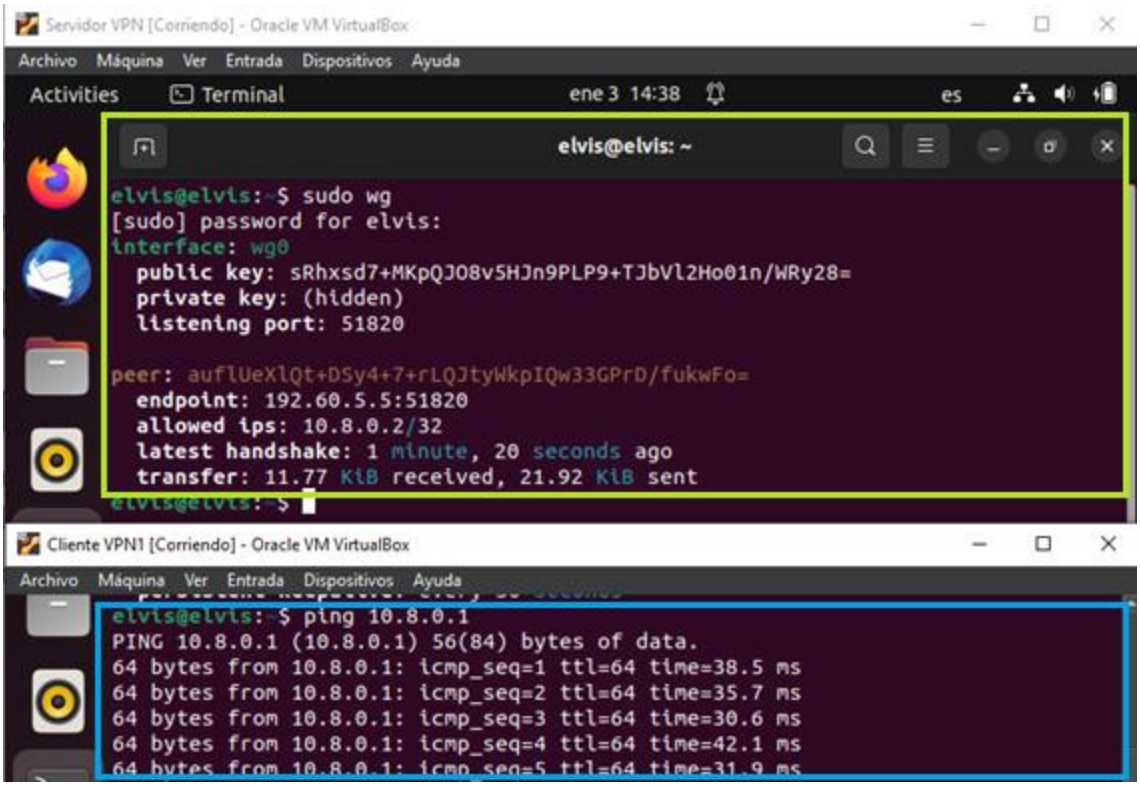

Figura 3.41 Prueba de *ping* Cliente-Servidor

Análisis de tráfico en servidor

Mediante el uso del comando ***tcpdump -envi wg0 hosts 8.8.8.8*** en el cual se menciona que ejecutado ***tcpdump*** se debe considerar mostrar la dirección IP del *host*, así como detalles adicionales del tiempo, se define la dirección para la cual se capturara los paquetes.

Para visualizar el estado del tráfico se debe considerar ejecutar una petición de cliente VPN a cualquier dirección IP pública, por ello en el *host* Cliente se realizó el uso del comando *ping* para conocer la conectividad con la DNS de Google, al iniciar este requerimiento en el *host* cliente, el servidor de manera instantánea enruta la dirección origen del cliente VPN hacia la dirección pública de DNS de Google y viceversa, observe la Figura 3.42.

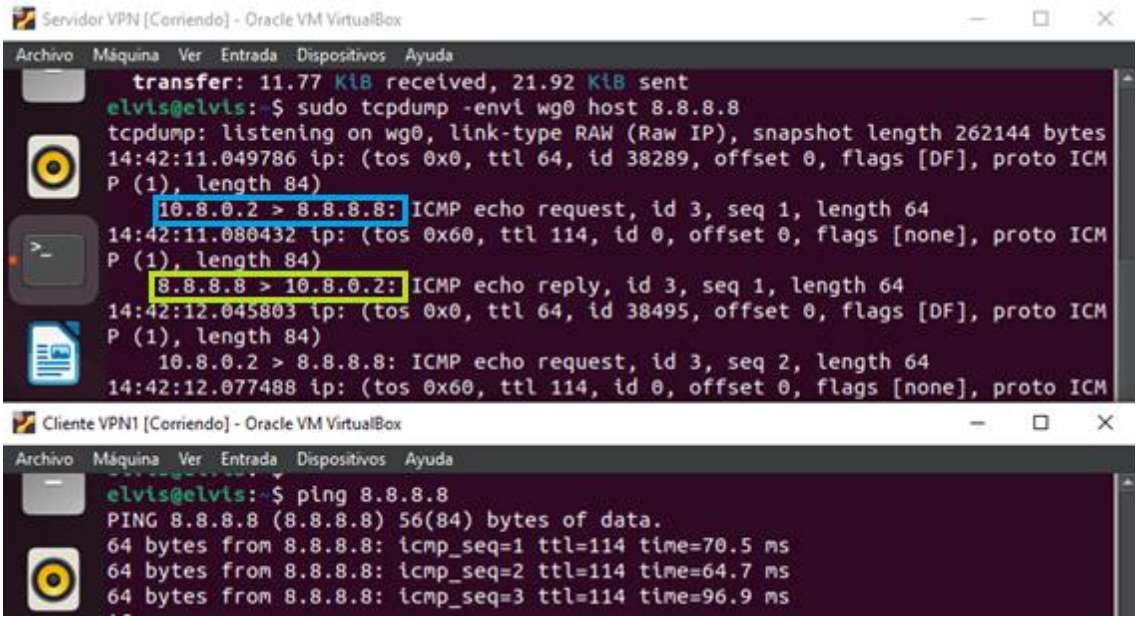

Figura 3.42 Enrutamiento a través del servidor

4 CONCLUSIONES

- La herramienta de automatización usada en el proyecto se encuentra escrita en *Python,* por tal razón es necesario que este lenguaje de programación se encuentre instalado en la máquina local dentro de la cual se alojará Ansible. Suele existir confusión en que los elementos remotos deben contar con *Python* para que se lleve a cabo la ejecución de tareas, argumento que es falso, debido a que las tareas son escritas en YAML y serán reconocidas por el *host* remoto cuando Ansible realice la ejecución de cada una de las tareas.
- El despliegue de un protocolo de enrutamiento dinámico como OSPF al requerir un cierto nivel de conocimiento sobre su principio de funcionamiento, puede provocar que el administrador de red incurra en errores. Con la ayuda de Ansible la probabilidad de error de configuración se reduce de manera considerable, posibilitando así la reducción del tiempo de despliegue y comunicación en la red.
- El protocolo SSH es requerido durante la instalación de Ansible, no solo por contar con una comunicación cifrada, sino que también este protocolo dentro de un enrutador permite autenticar usuarios remotos de confianza mediante una clave SSH. Este escenario se lo evidencia al momento de realizar la copia de la clave SSH en los enrutadores, tal cual se lo puede apreciar en la Figura 3.17.
- El protocolo SSH puede usar ciertos algoritmos que permitan el intercambio de claves entre dos pares, pero la mayoría de ellos no son compatibles con los enrutadores Cisco, por ello es necesario añadir el algoritmo Diffie-Hellman en conjunto con SHA1 en el archivo */etc/ssh/sshd_config*, para obligar a SSH que use el algoritmo mencionado anteriormente, debido a que es mayormente compatible con todos los modelos.
- El lenguaje YAML llega a constituirse de una forma estructural y legible, lo que facilita la redacción y el entendimiento de la mayoría de los usuarios, así mejorando la eficiencia de la gestión de un sistema, lo cual facilita la automatización de configuraciones sin la necesidad de aplicar un alto nivel de programación. Por esta razón, Ansible lo usa para la redacción de sus documentos, siendo estos: *playbooks* e inventarios.
- La administración de dispositivos finales es posible con Ansible, debido a que el único requisito de esta herramienta es que el elemento cuente con el protocolo SSH instalado para lograr aplicar el principio de su funcionamiento, el cual plantea realizar configuraciones de tareas en masa, así como de tareas repetitivas, de igual manera esto permite demostrar que Ansible puede llegar a

funcionar como una arquitectura cliente-servidor, debido a que funciona de manera centralizada.

- Una VPN se caracteriza por brindar seguridad y privacidad al usuario, tal es el caso por el cual se usan las claves públicas compartidas para realizar el cifrado de la información. Mientras que las claves privadas están ocultas, debido a que estas se usan para el descifrado de la información, así garantizando la privacidad de los paquetes transportados a través de una red VPN.
- Las pruebas de verificación y su correcto funcionamiento se los realizó separándolos en diferentes *playbooks* de configuración, dando, así como resultado conectividad entre servidor y cliente VPN, así como salida a Internet a través de una dirección privada ofrecida por la VPN.
- La administración a través del uso de la herramienta de Ansible obliga a cumplir un procedimiento estructurado, comenzando por la generación y copia de claves SSH en los *hosts* remotos para permitir la autenticación, y así desplegar cada una de las tareas que se implementan en los archivos de configuración (*playbooks*), de igual manera este procedimiento involucra tomar en consideración el uso de equipos adecuados, así como de módulos compatibles con la versión de Ansible en uso.
- El simulador de GNS3 al permitir simular dispositivos reales de red, permite evidenciar los problemas que pueden presentarse dentro de un ambiente real de implementación, con lo cual se pone en práctica todos los conocimientos adquiridos durante la formación académica.

5 RECOMENDACIONES

- Dentro de VirtualBox se recomienda habilitar una nueva interfaz en modo de controlador genérico, con el fin de establecer una conexión sin errores, entre GNS3 y VirtualBox. Esto debido a que el archivo OVA de GNS3 al ser importado al hipervisor, suele ejecutar por defecto un solo adaptador de red en modo NAT, lo que dificulta la conexión de máquinas virtuales, además al realizar este procedimiento se da la posibilidad de que GNS3 acceda a internet a través del adaptador NAT, y se comunique con VirtualBox usando el controlador genérico.
- Durante la configuración de SSH en los enrutadores cisco es importante asignar una longitud de 1024 bits a la clave RSA, de lo contrario al momento de realizar la copia de la clave pública desde Ansible se genera un error de longitud de

clave, debido a que a un menor tamaño se incumple con la mínima longitud para garantizar seguridad y compatibilidad en el enrutador 3725.

- Dentro de un *playbook* si se usa pipes "|" o "&" para la implementación de una tarea, es conveniente usar el módulo *shell* en vez de *command*. Debido a que este último puede llegar a interpretarlos de manera diferente durante la ejecución de las tareas, esto porque el módulo *command* no tiene acceso directo a un ambiente *shell,* tal es el caso de la tarea implementada en la Figura 3.34, en el cual, si se optara por usar *command,* Ansible no hubiese podido leer la clave privada generada para continuar con la construcción de la clave pública.
- Al manejar variables con el módulo *slurp* es recomendable usar el comando ***b64decode*** para permitir la decodificación de contenidos almacenados, dado que este módulo suele almacenar las contraseñas de una manera distinta a los otros contenidos.
- Se debe verificar que tanto el servidor como el cliente tengan habilitados el puerto 51820 UDP. Esto es necesario porque *wireguard* por defecto funciona en el puerto mencionado anteriormente, y en caso de no contar con esto la comunicación entre servidor y cliente nunca llegará a establecerse, obligando así al administrador de red a habilitar dicho puerto mediante el comando *ufw allow 51820/udp*.
- Es necesario llevar a cabo una investigación acerca del módulo cisco.ios, con el fin de que se permita conocer las funcionalidades, requisitos y limitantes que ofrece para Ansible. Esto debido a que al ser un *software* libre constantemente continúa evolucionando, lo que hace necesario conocer a detalle las nuevas características que este pone a disposición del público.
- Tanto el servidor *wireguard* como el cliente deben contar con la clave pública de su par, para llevar a cabo el proceso de cifrado y descifrado de datos, es decir que el cliente debe tener consigo la clave pública del servidor, y el servidor debe contar con la clave pública del cliente y estos deben estar escritos correctamente, de lo contrario la lectura de mensajes resulta errónea haciendo imposible la solicitud de peticiones por parte del cliente al servidor.
- Se debe tomar en cuenta que cualquier modificación manual en el archivo de configuración *wireguard*, haciendo referencia con esto a wg0.conf, no tendrá efecto dentro de la interfaz mientras el servicio de *wireguard* este habilitado, por tal razón es indispensable detener el servicio mediante el uso del comando *wg-quick down wg0*, antes de una intervención manual. Además, se debe considerar realizar respaldos de las configuraciones realizadas.

REFERENCIAS BIBLIOGRÁFICAS

[1] R. Hat, «Red Hat,» 21 Junio 2022. [En línea]. Available: https://www.redhat.com/es/topics/automation/learning-ansible-tutorial. [Último acceso: 2 12 2022].

[2] j. T. point, «java T point,» [En línea]. Available: https://www.javatpoint.com/computer-network-routing. [Último acceso: 6 12 2022].

[3] V. M. Carlos, «Enrutamiento estatico,» de *Redes Telematicas*, España, Paraninfo, S.A, 2014, p. 47.

[4] E. Limones, «OpenWebinars,» 24 Septiembre 2021. [En línea]. Available: https://openwebinars.net/blog/enrutamiento-estatico-vs-dinamico/. [Último acceso: 20 12 2022].

[5] P. Pedamkar, «Educba,» Educba, [En línea]. Available: https://www.educba.com/what-is-ospf/. [Último acceso: 1 1 2023].

[6] C. J. G. S. C. Z. Diego Álvarez Delgado, «Redes Privadas Virtuales (VPN),» *Redes de Computadores 1,* p. 9, 2014.

[7] L. Constantin, «CSO,» IDG Communications, 2 Abril 2020. [En línea]. Available: https://www.csoonline.com/article/3434788/what-is-wireguard-secure-simple-vpn-still-in-development.html. [Último acceso: 1 1 2023].

[8] R. Hodge, «CNET,» 10 Febrero 2022. [En línea]. Available: https://www.cnet.com/tech/services-and-software/what-is-wireguard-the-vpn-term-explained-and-whether-you-need-it/. [Último acceso: 2 1 2023].

[9] JOSH, «All Things Secured,» 18 Marzo 2022. [En línea]. Available: https://www.allthingssecured.com/vpn/faq/what-is-wireguard/. [Último acceso: 2 1 2023].

[10] NordLayer, «2022,» Marzo, 29 Marzo 2022. [En línea]. Available: https://nordlayer.com/blog/types-of-vpn-and-its-protocols/. [Último acceso: 3 1 2023].

[11] D. v. B. S. v. d. B. N. Z. Rutger van den Berg, «DESOSA,» 4 Marzo 2022. [En línea]. Available: https://desosa.nl/projects/ansible/2020/03/04/current-and-future-vision-of-ansible.html. [Último acceso: 4 1 2023].

[12] I. Moustakis, «Spacelift,» 26 Julio 2022. [En línea]. Available: https://spacelift.io/blog/ansible-modules. [Último acceso: 4 1 2023].

[13] R. M. Lorin Hochstein, Ansible up and Running, Julio: 20, 2017.

[14] A. R. M. F. J. R. R. M. L. P. M. R. M. Fernando Pereñíguez-García, «Herramientas de virtualización,» de *VNUML vs GNS3 en el desarrollo de laboratorios de redes virtuales*, Real, 2012, p. 8.

[15] M. W. D. Saravia, «Simulación de redes de computadoras con GNS3 e integración de máquinas virtuales,» *ITCA-FEPADE,* vol. 1, p. 9.

[16] J. McAllister, Implementing Devops with Ansible 2, Packt Publishing Ltd, 2017.

[17] J. F. F. A. Daniel Oh, Practical Ansible 2, Livery: Packt Publishing, 2020.

[18] Ashley, «OperaVPS,» 20 Septiembre 2022. [En línea]. Available: https://operavps.com/wireguard-vpn-protocol/. [Último acceso: 3 1 2023].

6 ANEXOS

La lista de los **Anexos** se muestran a continuación:

ANEXO I. Certificado de originalidad

ANEXO II. Enlaces

ANEXO III. Conjunto de datos extensos

ANEXO I: Certificado de Originalidad

CERTIFICADO DE ORIGINALIDAD

Quito, D.M. 2 de marzo de 2023

De mi consideración:

Yo, ITALO ALEXANDER CARREÑO MENDOZA, en calidad de Director del Trabajo de Integración Curricular titulado IMPLEMENTACIÓN DE ANSIBLE PARA LA CONFIGURACIÓN DE UNA VPN elaborado por el estudiante ELVIS ALEXIS TABANGO MATANGO de la carrera en TECNÓLOGÍA SUPERIOR EN REDES Y TELECOMUNICACIONES, certifico que he empleado la herramienta Turnitin para la revisión de originalidad del documento escrito completo, producto del Trabajo de Integración Curricular indicado.

El documento escrito tiene un índice de similitud del 13%.

Es todo cuanto puedo certificar en honor a la verdad, pudiendo el interesado hacer uso del presente documento para los trámites de titulación.

NOTA: Se adjunta el link del informe generado por la herramienta Turnitin.

https://epnecuador-my.sharepoint.com/:b:/g/personal/italo_carreno_epn_edu_ec/Eaj3omiHOq9NgBXwELvka0QBLHo1suiyPW4w7vNsJg0XEQ?e=ZIyn9S

Atentamente,

ITALO ALEXANDER CARREÑO MENDOZA

Docente

Escuela de Formación de Tecnólogos

ANEXO II: Enlaces

Anexo II.I Código QR de la implementación y pruebas de funcionamiento

ANEXO III: Códigos Fuente

Playbook OSPF enrutador 1

```
---
- hosts: R1
  gather_facts: false
  tasks:
   - name: Identificacion
     cisco.ios.ios_hostname:
       config:
         hostname: R1
       state: merged
   - name: Ip en la publica
     cisco.ios.ios_l3_interfaces:
       config:
       - name: Serial1/0
         ipv4:
         - address: 10.1.1.1 255.255.255.252
   - name: habilitar interfaz
     cisco.ios.ios_interfaces:
       config:
       - name: Serial1/0
         enabled: true
   - name: parametros ospf
     cisco.ios.ios_config:
       lines:
       - router ospf 1
```

```
      - network 192.160.45.0 255.255.255.0 area 0
      - network 10.1.1.0 255.255.255.0 area 0
      - network 192.169.56.0 255.255.255.0 area 0
```

Playbook OSPF enrutador 2

```
---
- hosts: R2
  gather_facts: false
  tasks:
   - name: Identificacion
     cisco.ios.ios_hostname:
       config:
         hostname: R2
       state: merged
   - name: Ip en la publica
     cisco.ios.ios_l3_interfaces:
       config:
       - name: Serial1/0
         ipv4:
         - address: 10.1.1.2 255.255.255.252
   - name: habilitar interfaz
     cisco.ios.ios_interfaces:
       config:
       - name: Serial1/0
         enabled: true
   - name: parametros ospf
     cisco.ios.ios_config:
       lines:
```

```
      - router ospf 1
      - network 192.160.5.0 255.255.255.0 area 0
      - network 10.1.1.0 255.255.255.0 area 0
```

Playbook de configuración Servidor VPN

```
---
- hosts: servidor
  become: yes
  become_method: sudo
  tasks:
    - name: actualizando repositorios
      apt:
        update_cache: true
        cache_valid_time: 3600
        force_apt_get: true
    - name: instalando wireguard
      apt:
        name: wireguard
        state: present
        update_cache: yes
#seccion creacion de claves
    - name: Creando claves
      shell: wg genkey | tee privatekey | wg pubkey > publickey
    - name: Revisando escritura clave privada
      shell: cat privatekey
      register: command_output
    - debug:
        var: command_output.stdout_lines
```

```
  - name: Almacenando clave privada en variable
    slurp:
      src: privatekey
    register private
#SECCION CONFIGURACION DE ADAPTADOR WG0
  - name: Creando archivo wg0 con extension conf
    file:
      path: "/etc/wireguard/wg0.conf"
      state: touch
  - name: Insertando lineas de configuracion para adaptador wg0
    blockinfile:
      path: /etc/wireguard/wg0.conf
      block: |
        [Interface]
        PrivateKey={{private.content | b64decode}}
        Address=10.8.0.1/24
        SaveConfig=true
        PostUp=iptables -A FORWARD -i wg0 -j ACCEPT; iptables -t nat -A POSTROUTING -o enp0s3 -j MASQUERADE;
        PostDown=iptables -D FORWARD -i wg0 -j ACCEPT; iptables -t nat -D POSTROUTING -o enp0s3 -j MASQUERADE;
        ListenPort=51820
  - name: Levantando interfaz wg0
    command: wg-quick up wg0
  - name: Usuario copie la clave pública mostrada
    shell: wg
    register: salida
```

```
    - debug:
        var: salida.stdout_lines
```

Playbook de configuración de Cliente VPN

```
---
- hosts: cliente
  become: yes
  become_method: sudo
  tasks:
    - name: actualizando repositorios
      apt:
        update_cache: true
        cache_valid_time: 3600
        force_apt_get: true
    - name: instalando wireguard
      apt:
        name: wireguard
        state: present
        update_cache: yes
#SECCION DE CREACION DE CLAVES
    - name: Creando claves
      shell: wg genkey | tee privatekey | pubkey > publickey
    - name: Revisando escritura clave privada
      shell: cat privatekey
      register: command_output
    - debug:
        var: command_output.stdout_lines
    - name: Almacenando clave privada en variable
```

```
    slurp:
      src: privatekey
    register: private
#FINALIZA  LA SECCION DE CREACION DE CLAVES
#SECCION CONFIGURACION  DE ADAPTADOR WG0
  - name: Creando archivo wg0 con extension conf
    file:
      path: "/etc/wireguard/wg0.conf"
      state: touch
  - name: Insertando lineas de configuracion para adaptador wg0
    blockinfile:
      path: /etc/wireguard/wg0.conf
      block: |
        [Interface]
        PrivateKey={{private.content | b64decode}}
        Address=10.8.0.2/24
        SaveConfig=true
        ListenPort=51820

        [Peer]
        Publickey=[Añadir clave publica]
        AllowedIPs=0.0.0.0/0
        Endpoint=192.160.45.5:51820
  - name: Copie su llave publica
    command: cat publickey
    register: salida
  - debug:
```

```
        var: salida.stdout_lines
```

Playbook para añadir clientes VPN

```
---
- hosts: servidor
  become: yes
  become_method: sudo
  vars_prompt:
    - name: ccpública
      prompt: Ingrese clave pública de cliente
      private: no
  tasks:
    - name: Agregando al cliente wireguard
      shell: wg set wg0 peer {{ccpública}} allowed-ips 10.8.0.2/32
    - name: Modificando sysctl
      lineinfile:
        path: /etc/sysctl.conf
        line: net.ipv4.ip_forward=1
    - name: Guardando cambios sysctl
      shell: sysctl -p
```