



UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

TRABAJO DE INTEGRACIÓN CURRICULAR

TEMA:

“AUTOMATIZACIÓN DE CONFIGURACIONES DE REDES DE DATOS MEDIANTE
UNA HERRAMIENTA DE INFRAESTRUCTURA COMO CÓDIGO (IAC)”

**Trabajo de titulación previo a la obtención del título de Ingeniera en
Telecomunicaciones**

Línea de investigación: Desarrollo, aplicación de software y cybersecurity (seguridad cibernética)

AUTOR:

Sayari Anahí Tobar Puerres

DIRECTOR:

Msc. Hernán Mauricio Domínguez Limaico

Ibarra - Ecuador 2026



UNIVERSIDAD TÉCNICA DEL NORTE BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1005235898		
APELLIDOS Y NOMBRES:	Tobar Puerres Sayari Anahí		
DIRECCIÓN:	Armando Hidrovo 12-42 y Reinaldo Chávez		
EMAIL:	satobarp@utn.edu.ec / sayaritobar16@gmail.com		
TELÉFONO FIJO:		TELÉFONO MÓVIL:	0985155594

DATOS DE LA OBRA	
TÍTULO:	AUTOMATIZACIÓN DE CONFIGURACIONES DE REDES DE DATOS MEDIANTE UNA HERRAMIENTA DE INFRAESTRUCTURA COMO CÓDIGO (IAC)
AUTOR (ES):	TOBAR PUERRES SAYARI ANAHÍ
FECHA: DD/MM/AAAA	27/01/2026
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	☒ PREGRADO ☐ POSGRADO
TÍTULO POR EL QUE OPTA:	INGENIERA EN TELECOMUNICACIONES
DIRECTOR:	MSC. HERNÁN MAURICIO DOMÍNGUEZ LIMAICO
ASESOR	MSC. FABIÁN GEOVANNY CUZME RODRÍGUEZ

2. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 27 días del mes de enero del 2026.

EL AUTOR:



Sayari Anahí Tobar Puerres

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

Ibarra, 27 de enero del 2026

MSC. HERNÁN MAURICIO DOMÍNGUEZ LIMAICO

DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

CERTIFICA:

Haber revisado el presente informe final del trabajo de Integración Curricular, el mismo que se ajusta a las normas vigentes de la Universidad Técnica del Norte; en consecuencia, autorizo su presentación para los fines legales pertinentes.



(f)

Msc. Hernán Mauricio Domínguez Limaico

C.C.: 1002379301

DEDICATORIA

Este trabajo dedico a las personas más importantes de vida quienes son mi motivación

A mi madre Mayra Puerres, por su amor incondicional, por ser una amiga y estar siempre a mi lado brindándome palabras de aliento y ser mi ejemplo a seguir.

A mi padre Héctor Tobar, por ser un padre amoroso y responsable, siendo el pilar de la familia, quien cree en mi en todo momento para ayudarme alcanzar mis objetivos.

A mi hermano Francisco, por ser mi cómplice, por su paciencia y compañía en todo momento.

A mi novio Scott Flores, por ser mi compañero en este proceso, brindándome su amor y apoyo constante.

Le dedicó a mi ángel en el cielo, mi querido abuelito Abelardo Tobar, quien estuvo en cada paso desde pequeña, por su cariño, apoyo y enseñanzas que siguen presentes.

Por último, a mi amigo de cuatro patas, Coco quien da un cariño sincero y me contagia de su alegría.

Tobar Puerres Sayari Anahí

AGRADECIMIENTO

Primero agradezco a Dios y al Divino Niño Jesús por su protección, guiar mis pasos y brindarme sabiduría para terminar esta etapa profesional.

A mis padres, Héctor Tobar y Mayra Puerres, por inculcarme los buenos valores, por su amor, por su esfuerzo, sacrificio y apoyo incondicional al depositar su confianza en mí. Este logro es suyo.

Agradezco a mi hermano, a mi tía Vero, a mis abuelitos Juan y Elvia, quienes han estado presentes en cada etapa, con cariño y siempre viendo por mi bienestar.

También, agradezco a mis amigos, por su amistad, por siempre poder contar con ellos y los momentos compartidos

De manera especial expreso mi agradecimiento a mi director el Msc. Mauricio Domínguez y a mi asesor Msc. Fabián Cuzme, por su tiempo dedicado y su orientación durante el desarrollo del trabajo de titulación.

Tobar Puerres Sayari Anahí

RESUMEN EJECUTIVO

Este trabajo presenta la implementación de Infraestructuras como Código (IaC) mediante la integración de la herramienta de automatización Ansible en infraestructuras de redes tradicionales, las cuales se despliegan como escenarios de simulación en la plataforma GNS3, con el objetivo de comprender y evaluar el comportamiento de la herramienta de automatización para lograr una gestión centralizada de configuraciones en dispositivos de red, lo cual, permite reducir la intervención manual, el tiempo de administración y errores humanos. La investigación adopta un enfoque teórico y práctico que incluyen el diseño de tres escenarios de red, basados en infraestructuras empresariales pequeñas y medianas, además presentan complejidad progresiva, debido al incremento de dispositivos y tareas de configuración. Dichas topologías se conforman por dispositivos como routers y switches de fabricantes Cisco y Mikrotik, los cuales, se gestionan desde el nodo de control Ansible mediante una estructura organizada de inventarios, variables y playbooks escritos en formato YAML, juntamente con el motor de plantilla Jinja2 para generar una configuración dinámica. Por consiguiente, se evidencia en los resultados que la configuración automatizada con Ansible proporciona mayor eficiencia operativa, escalabilidad, confiabilidad y estabilidad respecto a métodos convencionales para la administración de redes.

Palabras clave: Infraestructura como Código (IaC), gestión centralizada, Ansible, automatización, playbooks, configuración manual, GNS3.

ABSTRACT

This work presents the implementation of the Infrastructure as Code (IaC) approach through the integration of the Ansible automation tool in traditional network infrastructures, which are deployed as simulation scenarios on the GNS3 platform, with the purpose of understanding and evaluating the behavior of the automation tool to achieve centralized management of network device configurations. This approach allows reducing manual intervention, administration time, and human errors. The research adopts a theoretical and practical approach that includes the design of three network scenarios based on small and medium enterprise infrastructures, which present progressive complexity due to the increase in the number of devices and configuration tasks. These topologies, composed of routers and switches from manufacturers such as Cisco and Mikrotik, are managed from an Ansible control node using an organized structure of inventories, variables, and playbooks written in YAML format, together with the Jinja2 template engine to generate dynamic configurations. As a result, it is shown that automated configuration using Ansible provides higher operational efficiency, scalability, reliability, and stability compared to conventional network administration methods.

Keywords: Infrastructure as Code (IaC), centralized management, Ansible, automation, playbooks, manual configuration, GNS3.

LISTA DE SIGLAS

IaC: Infraestructura como Código (Infrastructure as Code)

SSH: Shell Seguro (Secure Shell)

BGP: Border Gateway Protocol

MPLS: Conmutación Multiprotocolo de Etiquetas (Multiprotocol Label Switching)

LDP: Protocolo de Distribución de Etiquetas (Label Distribution Protocol)

QoS: Calidad de Servicio (Quality of Service)

ACL: Lista de Control de Acceso (Access Control List)

VPN: Red Privada Virtual (Virtual Private Network)

OSPF: Open Shortest Path First

STP: Protocolo de Árbol de Expansión (Spanning Tree Protocol)

VLAN: Red de Área Local Virtual (Virtual Local Area Network)

IPv4: Protocolo de Internet versión 4 (Internet Protocol version 4)

GNS3: Simulador Gráfico de Redes 3 (Graphical Network Simulator 3)

YAML: YAML Ain't Markup Language

CLI: Interfaz de Línea de Comandos (Command Line Interface)

LAN: Red de Área Local (Local Area Network)

WAN: Red de Área Amplia (Wide Area Network)

ÍNDICE DE CONTENIDOS

Capítulo I: Antecedentes.....	1
1.1. Tema	1
1.2. Problema	1
1.3. Objetivos.....	4
1.3.1. Objetivo General.....	4
1.3.2. Objetivos Específicos.....	4
1.4. Alcance	4
1.5. Justificación	8
Capítulo II: Fundamentación Teórica	10
2.1. Automatización de red	10
2.1.1. Automatización de red con scripts	11
2.1.2. Automatización de red con software.....	12
2.1.3. Interfaces de programación de aplicaciones (API)	14
2.2. Formato de datos.....	16
2.2.1. XML.....	17
2.2.2. JSON	18
2.2.3. YAML.....	19
2.3. Sistema de control de versiones.....	20
2.3.1. Git	21
2.3.2. Pruebas de código	22

2.4. Infraestructura como Código (IaC).....	23
2.4.1. Lenguaje de infraestructura declarativo	24
2.4.2. Lenguaje de infraestructura imperativo	25
2.5. Herramientas de Infraestructura como Código	25
2.5.1. Ansible	26
2.5.2. Puppet	28
2.5.3. Chef.....	30
Capítulo III: Implementación y funcionamiento.....	32
3.1. Requerimientos de software.....	32
3.1.1. Selección de entorno de simulación.....	34
3.1.2. Selección de sistemas virtualizados de networking	42
3.1.3. Selección de Herramienta de automatización	49
3.2. Requerimientos de hardware.....	58
3.3. Diseño de topologías de red.....	59
3.1.1. Escenario 1: Diseño de red básico VLANs y STP	65
3.1.2. Escenario 2: Diseño de red con MPLS-VPN	69
3.1.3. Escenario 3: Diseño de red MPLS y QoS	74
3.4. Instalación del servicio de automatización Ansible.....	78
3.5. Configuración de los escenarios de red	85
3.5.1. Preparación del entorno	85
3.5.2. Desarrollo de playbook para escenario 1	93

3.5.3. Desarrollo de playbook para escenario 2	107
3.5.4. Desarrollo de playbook para escenario 3	155
Capítulo IV: Evaluación de Resultados	183
4.1. Análisis de la gestión de configuraciones centralizada.....	183
4.1.1. Autenticación para conexiones SSH	183
4.1.2. Advertencias durante la ejecución de playbooks	187
4.1.3. Evidenciar cambios aplicados en los dispositivos	188
4.2. Tiempo de ejecución de los playbooks con Ansible	191
4.2.1. Comparación del tiempo entre configuración automatizada y manual....	203
4.3. Optimización para automatización de configuraciones	216
4.3.1. Agrupación de hosts por roles.....	217
4.3.2. Definir playbooks por función	218
Conclusiones y Recomendaciones	229
Conclusiones	229
Recomendaciones	232
Bibliografía	234
Anexos	244

ÍNDICE DE FIGURAS

Figura 1	Técnica “Árbol de problemas”	3
Figura 2	Diagrama de las fases de la metodología PPDIIOO	5
Figura 3	Ejemplo de script para crear un servidor.....	12
Figura 4	Ejemplo de una estructura de plantilla cookbooks.....	13
Figura 5	Comunicación entre aplicaciones a través de una API.....	14
Figura 6	Estructura de un archivo XML	17
Figura 7	Comparación entre formatos de datos XML y JSON.....	18
Figura 8	Ejemplo de formato de dato YAML.....	20
Figura 9	Ejemplo de código declarativo	25
Figura 10	Arquitectura de herramienta Ansible	27
Figura 11	Estructura de playbook en Ansible.....	28
Figura 12	Arquitectura de herramienta Puppet.....	29
Figura 13	Arquitectura de herramienta Chef	31
Figura 14	ISO/IEC 25010: Características para evaluar herramientas de software ..	33
Figura 15	Diseño jerárquico de redes cisco	61
Figura 16	Diseño jerárquico de dos niveles.....	62
Figura 17	Topología de red básico VLANS, STP	66
Figura 18	Topología de red MPLS-VPN	71
Figura 19	Topología de red MPLS y QoS	75
Figura 20	Descarga del contenedor de GNS3.....	81
Figura 21	Importar el archivo del contenedor	81
Figura 22	Selección del tipo de instalación del contenedor.....	82
Figura 23	Instrucciones del contenedor	82
Figura 24	Contenedor de automatización disponible	83

Figura 25	Configuración del contenedor de Ansible	83
Figura 26	Asignación de IP de la interfaz del contenedor	84
Figura 27	Actualización de paquetes en el contenedor.....	84
Figura 28	Verificación de la instalación de Ansible en contenedor	85
Figura 29	Ingreso al archivo de configuración de red de Ansible	86
Figura 30	Configuración de direcciones IP en interfaces del contenedor	86
Figura 31	Asignación de IP a interfaz de gestión	87
Figura 32	<i>Habilitar SSH para dispositivos Cisco</i>	88
Figura 33	Habilitación de SSH en Mikrotik	89
Figura 34	Conexión de Ansible con todos los dispositivos	89
Figura 35	Establecimiento de conexión remota desde Ansible	90
Figura 36	Estructura del proyecto Ansible	91
Figura 37	Creación de carpetas y archivos en Ansible	92
Figura 38	Resolución de nombres de hosts en escenario 1.....	94
Figura 39	Definición de dispositivos para escenario 1	95
Figura 40	Archivos para definición de variables en escenario 1	95
Figura 41	Definición de variables para el grupo routers	96
Figura 42	Definición de variables para el grupo switches.....	96
Figura 43	Definición de variables para el grupo switches_acceso	97
Figura 44	Configuración del router Cisco en escenario 1	99
Figura 45	Configuración del switch de core en escenario 1	100
Figura 46	Tareas de configuración en switch de core en escenario 1.....	101
Figura 47	Tareas de configuración en switch de core en escenario 1.....	102
Figura 48	Configuración de switches de acceso en escenario 1	103
Figura 49	Asignar IP a interfaz de VLAN60 en Switch de acceso 1.....	104

Figura 50	Asignar IP a interfaz de VLAN80 en Switch de acceso 2.....	104
Figura 51	Ejecución del playbook para escenario 1	106
Figura 52	Validación de configuraciones aplicadas en escenario 1	106
Figura 53	Estructura del proyecto Ansible para escenario 2	107
Figura 54	Resolución de nombres de hosts en escenario 3.....	108
Figura 55	Definición de hosts para equipos Cisco en escenario 2.....	109
Figura 56	Definición de hosts para equipos Mikrotik en escenario 2	109
Figura 57	Definición de variables para todos los dispositivos en escenario 2	110
Figura 58	Definición de variables para routers de núcleo en escenario 2	111
Figura 59	Definición de variables para router de borde1 en escenario 2	114
Figura 60	Definición de variables para router de borde2 en escenario 2	115
Figura 61	Definición de variables para routers clientes en escenario 2- Parte 1	116
Figura 62	Definición de variables para routers clientes en escenario 2- Parte 2.....	117
Figura 63	Establecimiento del bloque PLAY para IPv4 en escenario 2.....	119
Figura 64	Configuración de IPv4 en equipos Cisco en escenario 2	122
Figura 65	Configuración de IPv4 en equipos Mikrotik en escenario 2	124
Figura 66	Establecimiento del bloque PLAY para OSPF en escenario 2.....	126
Figura 67	Configuración de OSPF en equipos Cisco en escenario 2	128
Figura 68	Establecimiento del bloque PLAY para MPLS en escenario 2.....	129
Figura 69	Configuración de MPLS en equipos Cisco en escenario 2- Parte 1.....	130
Figura 70	Configuración de MPLS en equipos Cisco en escenario 2- Parte 2.....	131
Figura 71	Establecimiento del bloque PLAY para iBGP en escenario 2	133
Figura 72	Configuración de iBGP entre routers PE en escenario 2.....	135
Figura 73	Configuración de VRFs en routers PE en escenario 2	137
Figura 74	Configuración de eBGP en routers PE en escenario 2- Parte 1.....	139

Figura 75	Configuración de eBGP en routers PE en escenario 2- Parte 2.....	140
Figura 76	Configuración de eBGP en routers CE en escenario 2.....	143
Figura 77	Configuración de distribución de rutas en routers PE en escenario 2.....	144
Figura 78	Establecimiento del PLAY para anunciar rutas CE en escenario 2	145
Figura 79	Configuración para anunciar rutas CE en escenario 2	147
Figura 80	Ejecución del playbook IPv4 referente al escenario 2	148
Figura 81	Validación de configuración IPv4 en escenario 2	149
Figura 82	Ejecución del playbook OSPF referente al escenario 2	149
Figura 83	Validación de configuración OSPF en escenario 2	150
Figura 84	Ejecución del playbook MPLS referente al escenario 2.....	150
Figura 85	Validación de configuración MPLS en escenario 2	151
Figura 86	Validación de configuración iBGP en routers PE en escenario 2	151
Figura 87	Validación de configuración VRFs en escenario 2	152
Figura 88	Ejecución del playbook eBGP en routers PE referente al escenario 2	152
Figura 89	Validación de configuración eBGP en router PE en escenario 2	153
Figura 90	Validación de configuración eBGP en router CE en escenario 2.....	153
Figura 91	Validación de distribución de rutas PE en escenario 2	154
Figura 92	Validación de anuncio de rutas CE en escenario 2	154
Figura 93	Estructura del proyecto Ansible para escenario 3	155
Figura 94	Definición de hosts para equipos Cisco en escenario 3.....	156
Figura 95	Definición de hosts para equipos Mikrotik en escenario 3	157
Figura 96	Definición de variables en archivo hosts del escenario 3.....	157
Figura 97	Variables en común de los dispositivos para escenario 3	158
Figura 98	Variables para definir interfaces en Routers Proveedor-P1	159
Figura 99	Variables para definir interfaces en Routers Proveedor-P2	159

Figura 100	Variables para definir interfaces en Routers Proveedor-P3	160
Figura 101	Variables para definir interfaces en Routers Proveedor-P4	160
Figura 102	Variables para definir interfaces en Routers Bordes-PE1	161
Figura 103	Variables para definir interfaces en Routers Bordes-PE2 y PE3	161
Figura 104	Variables para definir interfaces en Routers Clientes-CE1	162
Figura 105	Variables para definir interfaces en Routers Clientes-CE2 y CE3.....	162
Figura 106	Configuración de IPv4 en equipos Cisco en escenario 3	164
Figura 107	Configuración de IPv4 en equipos Mikrotik en escenario 3	165
Figura 108	Establecimiento del bloque PLAY para configuración OSPF en escenario 3.....	166
Figura 109	Configuración OSPF para dispositivos Cisco en escenario 3	168
Figura 110	Configuración OSPF para dispositivos Mikrotik en escenario 3	169
Figura 111	Configuración de MPLS para dispositivos Cisco en escenario 3.....	171
Figura 112	Subtareas para configuración de MPLS en escenario 3	172
Figura 113	Configuración de listas de control de acceso (ACL) en escenario 3	173
Figura 114	Creación de class-map para cada tráfico en escenario 3	174
Figura 115	Configuración de la política QoS de salida en escenario 3	175
Figura 116	Configuración de la política QoS de entrada en escenario 3.....	177
Figura 117	Aplicación de políticas QoS en escenario 3	177
Figura 118	Ejecución del playbook IPv4 referente al escenario 3	179
Figura 119	Validación de configuración IPv4 en escenario 3	179
Figura 120	Ejecución del playbook OSPF referente al escenario 3	180
Figura 121	Validación de configuración OSPF en escenario 3	180
Figura 122	Ejecución del playbook MPLS referente al escenario 3.....	181
Figura 123	Validación de configuración MPLS en escenario 3	181

Figura 124	Ejecución del playbook QoS referente al escenario 3	182
Figura 125	Validación de configuración QoS en escenario 3	182
Figura 126	Lista de hosts autenticados	184
Figura 127	Modificación de nombre del host	184
Figura 128	Error por falta de autenticación en host	185
Figura 129	Autenticación de host con nuevo identificador	185
Figura 130	Nueva entrada registrada por renombre de host	186
Figura 131	Aplicación de configuraciones al equipo renombrado	186
Figura 132	Advertencias durante la ejecución del playbook en Ansible	187
Figura 133	Ejemplo de aplicación de configuración eBGP en equipos Mikrotik ...	189
Figura 134	Verificación directa en el equipo Mikrotik de la configuración eBGP ..	190
Figura 135	Verificación de configuraciones en la consola de equipo Cisco	190
Figura 136	Archivo Bash para medir el tiempo total de ejecución de playbooks ...	193
Figura 137	Tiempo total referente de ejecución del playbook - Escenario 1	195
Figura 138	Tiempo total referente de ejecución de playbooks - Escenario 2	198
Figura 139	Tiempo total referente de ejecución de playbooks - Escenario 3	200
Figura 140	Agrupación de hosts por rol en inventario de Ansible	217
Figura 141	Tiempo de referencia en ejecución del playbook por función IPv4	219
Figura 142	Tiempo de referencia en ejecución del playbook por función OSPF	221
Figura 143	Tiempo de referencia en ejecución del playbook por función MPLS ...	223
Figura 144	Tiempo de referencia en ejecución del playbook por función QoS	225
Figura 145	Actualización de paquetes en el sistema	244
Figura 146	Instalación de herramientas requeridas por GNS3	245
Figura 147	Instalación del entorno GNS3	246
Figura 148	Ingreso a la interfaz gráfica de GNS3	246

Figura 149	Actualización de paquetes en sistema	247
Figura 150	Instalación de paquetes de última versión	248
Figura 151	Agregar el repositorio de ansible al sistema.....	248
Figura 152	Instalación de Ansible	248
Figura 153	Verificación de la instalación de Ansible.....	249

ÍNDICE DE TABLAS

Tabla 1	<i>Características de herramientas de infraestructura como código (IaC)</i>	31
Tabla 2	Abreviaturas de requisitos para evaluación del software de simulación	36
Tabla 3	Requerimientos de compatibilidad para software de simulación	37
Tabla 4	Requerimientos de eficiencia de desempeño para software de simulación	38
Tabla 5	Requerimientos de flexibilidad para software de simulación	40
Tabla 6	Evaluación de la herramienta de simulación	41
Tabla 7	Comparación de parámetros de plataformas de networking en GNS3	43
Tabla 8	Abreviaturas de los requisitos para evaluación de sistemas de virtualización	
	networking	44
Tabla 9	Requerimientos de compatibilidad para sistemas de virtualización de	
	networking	45
Tabla 10	Requerimientos de eficiencia de desempeño para sistemas de virtualización	
	de networking	46
Tabla 11	Evaluación del sistema de virtualización de networking	48
Tabla 12	Características principales de herramientas de IaC	49
Tabla 13	Abreviaturas de los requisitos para evaluación de la herramienta IaC	51
Tabla 14	Requerimientos de compatibilidad para herramienta IaC	52
Tabla 15	Requerimientos de eficiencia de desempeño para herramienta IaC	53
Tabla 16	Requerimientos de seguridad para herramienta IaC	55
Tabla 17	Evaluación de la herramienta de Infraestructura como Código (IaC)	56
Tabla 18	Dispositivos que integran el primer escenario	67
Tabla 19	Direccionamiento lógico mediante VLANs	68
Tabla 20	Direccionamiento IPv4 para el primer escenario	69
Tabla 21	Dispositivos que integran el segundo escenario	72

Tabla 22	Direccionamiento IPv4 para el segundo escenario	73
Tabla 23	Dispositivos que integran el tercer escenario	76
Tabla 24	Direccionamiento IPv4 para el tercer escenario	77
Tabla 25	Comparación entre máquina virtual y Docker	79
Tabla 26	Comandos para habilitar SSH en dispositivos Cisco	87
Tabla 27	Comandos para habilitar SSH en dispositivos Mikrotik	88
Tabla 28	Instrucciones en común para el desarrollo de los playbooks	93
Tabla 29	Instrucciones del playbook para configuración del escenario 1	97
Tabla 30	Estado de ejecución para tareas del playbook	105
Tabla 31	Instrucciones del playbooks para configuración IPv4 del escenario 2	118
Tabla 32	Instrucciones del playbook para configuración OSPF del escenario 2	124
Tabla 33	Instrucciones del playbook para configuración IPv4 del escenario 3	163
Tabla 34	Instrucciones del playbook para configuración OSPF del escenario 3	165
Tabla 35	Instrucciones del playbook para configuración MPLS del escenario 3	169
Tabla 36	Tiempo promedio de configuraciones iniciales de forma manual	191
Tabla 37	Tiempo total promedio de la ejecución de playbook - Escenario 1	196
Tabla 38	Tiempo total promedio de la ejecución de playbooks - Escenario 2	198
Tabla 39	Tiempo total promedio de la ejecución de playbooks - Escenario 3	200
Tabla 40	Calcular el tiempo total promedio del proceso de configuración automatizada	202
Tabla 41	Tiempo total promedio de la configuración automatizada mediante Ansible de los tres escenarios	203
Tabla 42	Tiempo total promedio de configuraciones manuales- Escenario 1	204
Tabla 43	Tiempo total promedio de configuraciones manuales- Escenario 2	206
Tabla 44	Tiempo total promedio de configuraciones manuales- Escenario 3	210

Tabla 45	Comparación de tiempos de configuración automatizada y manual	216
Tabla 46	Tiempo total promedio de ejecución del playbook por función IPv4	219
Tabla 47	Tiempo total promedio de ejecución del playbook por función OSPF	221
Tabla 48	Tiempo total promedio de ejecución del playbook por función MPLS	223
Tabla 49	Tiempo total promedio de ejecución del playbook por función QoS	225
Tabla 50	Tiempo total promedio de ejecución de playbooks por función	227

Capítulo I: Antecedentes

1.1. Tema

AUTOMATIZACIÓN DE CONFIGURACIONES DE REDES DE DATOS MEDIANTE UNA HERRAMIENTA DE INFRAESTRUCTURA COMO CÓDIGO (IAC)

1.2. Problema

El continuo avance de la tecnología en redes de comunicación, se enfrenta a un crecimiento de adopción de servicios por computación en la nube y arribo de Internet de las Cosas (IoT), lo que conlleva a un incremento de clientes y a la vez un aumento masivo de conexiones de dispositivos a la red, por tanto, se requiere explorar nuevas tendencias tecnológicas que permitan una adaptación ágil a las demandas de configuración a los dispositivos que conforman las infraestructuras de red, las cuales están sujetas a prestar una alta disponibilidad en base a sus niveles de rendimiento y capacidad (Ríos & Ríos, 2016). Las redes tradicionales ante este tipo de exigencias llegan a los límites de su capacidad de manejo de grandes cantidades de datos, de modo dichas limitantes presentan inconvenientes para la administración de políticas de red que gestionen todo el tráfico generado por este tipo de hosts, además, muestran desafíos para adaptarse a modificaciones, mantenimiento y fallos por reorganización de la red, por lo que se ve la necesidad de modificar sus estructuras para automatizar procesos organizativos (Conde Gil, 2021).

Tradicionalmente, la gestión de la infraestructura de tecnología de la información (TI) se realiza de manera manual, instalando y configurando los dispositivos físicos uno a uno. Este enfoque manual conlleva problemas de escalabilidad, disponibilidad y, sobre todo, mayor retardo en la respuesta a las demandas de la red (Camas Mainato & Riera Taza, 2022). Esto se debe a que la configuración manual puede generar retrasos en el acceso a las aplicaciones que

se alojan en la red, especialmente en momentos de alta demanda. Además, la falta de automatización dificulta la detección y corrección rápida de fallos, lo que afecta la accesibilidad y el rendimiento de la red en conjunto. Adicionalmente, los errores humanos también son más probables en configuraciones manuales extensas, lo que aumenta el riesgo de interrupciones no planificadas, vulnerabilidades de seguridad y fallos parciales en la red. Estos problemas pueden tener un impacto significativo en la productividad, adaptación rápida de avances tecnológicos y la satisfacción del cliente (Wågbrant & Dahlén Radic, 2022).

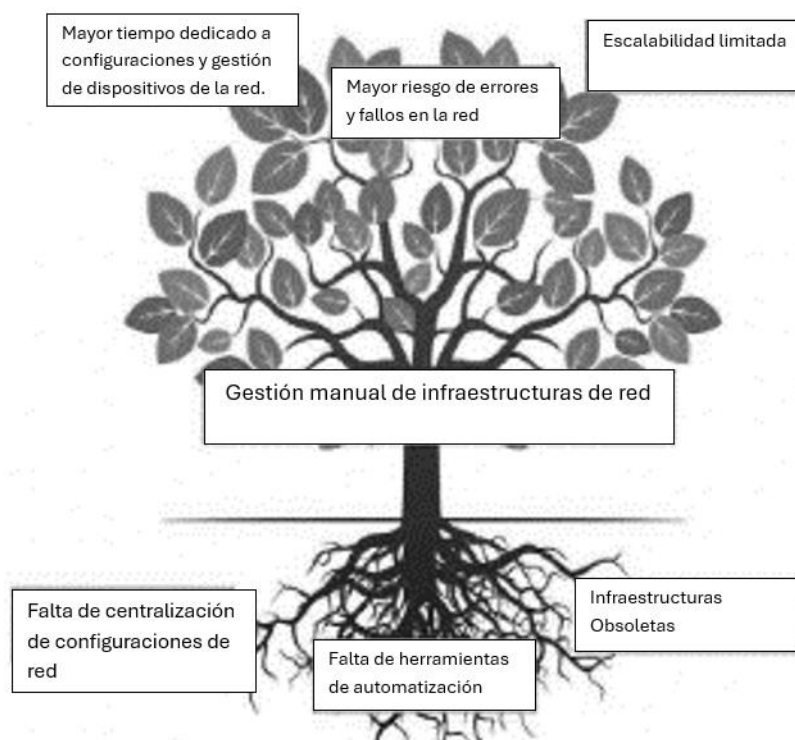
Para abordar estos desafíos, la automatización de infraestructuras como código (IaC) se presenta como una solución ideal debido a que emplea prácticas y herramientas del desarrollo de software, de modo que se garantiza que las acciones por realizar sean exhaustivamente probadas antes de aplicarse a los elementos críticos de redes de datos que correspondientes a los dispositivos de red. La automatización como código fundamenta su funcionamiento en el uso de scripts para configurar la infraestructura de red manera programática en lugar de realizar configuraciones manuales en los dispositivos, lo que permite obtener una infraestructura altamente elástica, es decir, repetible y escalable si las necesidades de la infraestructura recaen en aumentar sus capacidades (Romero, 2021).

Finalmente, la gestión manual de la infraestructura de red presenta una serie de desafíos que pueden obstaculizar el crecimiento y la eficiencia de las empresas en un entorno competitivo y en constante evolución. La dependencia de métodos manuales no solo conlleva riesgos técnicos, como la falta de escalabilidad y disponibilidad, sino que también genera costos operativos significativos (Dominguez-Quintero & Vargas-Lombardo, 2021). Ante esta situación, la automatización de infraestructuras como código emerge como una solución integral que promete mejorar la agilidad, la confiabilidad y la rentabilidad de la gestión de la infraestructura de red. Al adoptar prácticas y herramientas del desarrollo de software, las empresas pueden garantizar una configuración consistente, una respuesta rápida a las demandas

del mercado y una adaptación fluida a los avances tecnológicos. Por tanto, invertir en la automatización de infraestructuras como código no solo mejora la eficiencia operativa, sino que también fortalece la posición competitiva de las empresas en un mundo digitalmente transformado (Caisatoa, 2023).

Lo mencionado en la sección del problema en este anteproyecto se basa en la técnica llamada "árbol de problemas", como se muestra en la Figura 1. Esta técnica facilita la identificación del problema central y las relaciones de causa y efecto que lo generan, con el objetivo de resolver el problema identificado a través del desarrollo del proyecto.

Figura 1
Técnica “Árbol de problemas”



1.3. Objetivos

1.3.1. Objetivo General

Implementar una infraestructura como código (IaC) mediante el uso de una herramienta de automatización de configuraciones de red en infraestructuras de redes de datos tradicionales.

1.3.2. Objetivos Específicos

- Establecer la fundamentación teórica sobre el estudio de la herramienta que permiten la automatización de las funciones de red en infraestructuras de redes de datos abarcando aspectos sobre configuraciones, protocolos, componentes y dispositivos para la automatización en la red.
- Identificar la herramienta de simulación que aloje la infraestructura de red que utilizará el enfoque de automatización de funciones mediante herramientas de software libre.
- Desplegar escenarios de simulación de infraestructuras como código (IaC).
- Evaluar los resultados del despliegue del escenario identificando como la herramienta de automatización permiten la gestión centralizada de configuraciones de red.

1.4. Alcance

El principal enfoque del proyecto es comprender el funcionamiento de la herramienta de automatización y su implementación en infraestructuras de redes de datos. El objetivo es centralizar las funciones de cada dispositivo a través de un nodo central que posea la capacidad de gestionar y controlar las operaciones de la red. Por lo tanto, el marco teórico del trabajo se basa en la investigación aplicada, que implica recopilar información bibliográfica sobre las funciones de automatización en infraestructuras de redes de datos incluyendo configuraciones, protocolos, componentes y dispositivos que puedan manejar las herramientas de

automatización de código abierto disponibles. Esto proporcionará los conocimientos necesarios para la implementación práctica de escenarios de red que utilicen este tipo de tecnologías. Los escenarios de estudio de la tecnología de automatización serán desplegados mediante un software de simulación de diseño de redes y se establecerá tres topologías de red para demostrar la automatización en infraestructuras de red. De modo que, el tema central del proyecto identifica y explora las posibilidades de mejora y optimización en la gestión de redes a través de la automatización. (Zhang et al., 2023)

Para abordar todas las fases relacionadas con el desarrollo del trabajo de integración curricular se basa en la metodología PPDIOO, la cual consta de seis etapas fundamentales descritas en Figura 2.

Figura 2
Diagrama de las fases de la metodología PPDIOO



Comienza con la preparación inicial, donde se establecen los requisitos de la fundamentación teórica. La etapa de planeación implica identificar los requisitos de la red en función de las necesidades del usuario. Luego, en la fase de diseño, se crea un diseño detallado que cumple con los requisitos técnicos, proporcionando la base para la implementación. La etapa de implementación involucra la construcción o incorporación de dispositivos a la red, seguida de pruebas finales para garantizar el correcto funcionamiento y la disponibilidad continua, corrigiendo cualquier error que pueda surgir. Finalmente, en la etapa de optimización,

se gestionan y resuelven problemas de funcionamiento, y si es necesario, se rediseña la red para mejorar su rendimiento si no cumple con las expectativas iniciales.

La primera sección del desarrollo del proyecto corresponde al análisis de la fundamentación teórica mediante la recopilación en fuentes bibliográficas que permitan identificar y contextualizar cómo las herramientas de automatización permiten que infraestructuras de red puedan ser gestionadas de forma centralizada. Esto es relacionado con las primeras fases de la metodología correspondientes a las fases de preparación y planificación ya que permite establecer una sólida base de conocimientos teóricos que permita identificar cómo la automatización de redes facilita la mejora del rendimiento y la eficiencia en la gestión de recursos de red.

Durante la fase del diseño de la red, tanto el hardware que conforman la infraestructura de red y las herramientas de automatización se adaptarán en el software de simulación de redes. En este sentido, la gestión de la red se llevará a cabo mediante un servidor que posea la herramienta de automatización. Además, en fases previas será necesario identificar los dispositivos de redes de datos compatibles con dicha herramienta. Para finalmente, reconocer las configuraciones de los dispositivos seleccionados con la finalidad de establecer su interoperabilidad con el software de automatización. Esta fase de diseño permite identificar de mejor manera las capacidades de la herramienta de automatización de redes de datos y su integración con los dispositivos de red.

En el paso de implementación, el proceso consta en instalar el servicio de automatización en una máquina virtual, seguido de la interconexión y configuración de los dispositivos que conforman la infraestructura de red. Respecto al despliegue de los escenarios de prueba, se establecerán tres: Inicialmente, se desarrollará un entorno básico que permitirá comprender el funcionamiento de la herramienta de automatización de dispositivos de red. El segundo escenario estará centrado en la creación de una infraestructura de red que posea una

cantidad considerable de dispositivos de red compatibles con la herramienta de automatización para aplicar temas de interconectividad, gestión de dispositivos y seguridad. Por último, el tercer escenario está relacionado en identificar las capacidades de la herramienta al manejar gran número de dispositivos y determinar como la herramienta responde ante demandas intensivas.

En la fase de operación, se verificará la funcionalidad de estos escenarios mediante la ejecución de los scripts de configuración, así como realizar pruebas de rendimiento y escalabilidad donde se midan los tiempos de ejecución y uso de recursos en el despliegue del número de nodos gestionados de manera incremental.

Finalmente, al concluir la implementación y evaluación de los tres escenarios propuestos, se identificarán áreas clave para la optimización en la automatización de infraestructuras de red. En el primer escenario, se podrá perfeccionar el conocimiento sobre la configuración y operación básica de los dispositivos automatizados, lo que permitirá mejorar la documentación y los procedimientos de instalación inicial. El segundo escenario revelará oportunidades para optimizar la interconectividad, gestión y seguridad en infraestructuras de red complejas, asegurando una integración más eficiente de la automatización a los dispositivos. Finalmente, el tercer escenario destacará las mejoras necesarias en la escalabilidad y rendimiento de la herramienta de automatización, facilitando su uso en entornos con un gran número de dispositivos y proporcionando estrategias para manejar de manera más efectiva redes de gran escala. Estas optimizaciones contribuirán significativamente a la robustez y eficiencia del sistema automatizado de gestión de redes.

1.5. Justificación

El presente proyecto de investigación surge con el objetivo de abordar los desafíos relacionados con la disponibilidad, escalabilidad y tiempos de configuración en redes tradicionales. El crecimiento exponencial de dispositivos de red (switches, routers, entre otros) en estas infraestructuras ha generado un aumento significativo en la complejidad de la gestión de la red, lo que ha destacado la necesidad de implementar soluciones eficientes. La integración de herramientas de automatización se ha vuelto crucial en este contexto, ya que facilitan la configuración centralizada de dispositivos que se encuentran presentes en la red.

En base a los Objetivos de Desarrollo Sostenible (ODS), presentados por la Naciones Unidas, el objetivo que se relaciona para justificar el desarrollo del presente trabajo de grado es el objetivo 9: “Construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación”, el cual, tiene como metas lo siguiente: incremento de investigaciones científicas, promover la adopción y apoyo al desarrollo de nuevas tecnologías (ONU, 2023). Además, con estas metas se enfoca garantizar el bienestar humano, desarrollo económico y no permitir limitaciones al acceso a la información. Por tanto, este trabajo aporta la innovación de una herramienta de automatización emergente, para promover la investigación y mejorar la gestión de las redes ya que, en la actualidad, el incremento constante tanto de usuarios y de dispositivos en las infraestructuras ha producido un incremento en la complejidad de la gestión de la red, ya que, toma mayor tiempo las configuraciones de cada dispositivo y en algunos casos el servicio puede verse interrumpido. No obstante, las herramientas de automatización permiten mejorar la administración de la red y optimizar las configuraciones de los dispositivos con la finalidad de mejorar el despliegue y funcionamiento de infraestructuras de la red con mejores prestaciones de interconectividad hacia los hosts. (Bone Andrade et al., 2021)

Además, el Ministerio de Telecomunicaciones y Sociedad de la información presenta en su publicación denominada AGENDA DE TRANSFORMACIÓN DIGITAL DEL ECUADOR 2022- 2025 la existencia de 17 de áreas que requieren de la Transformación Digital, denominándolas pilares (MINTEL, 2022). De todas estas áreas expuestas en la publicación, el pilar 8 se asocia con la finalidad del presente proyecto, debido a que el objetivo principal de esta sección recae en la fomentación de tecnologías en la industria. Por consecuente, el desarrollo de la presente investigación se alinea con la finalidad de este pilar, ya que presenta como las herramientas de automatización permiten la configuración masiva de dispositivos de red de forma centralizada, lo que en un futuro puede ser aplicado en ambientes industriales.

De este modo, al estar familiarizados con innovación y procesos emergentes como la automatización, las organizaciones que acogen las Tecnologías de la Información y Comunicación (TIC) en diferentes sectores de la sociedad como educación, salud, social y seguridad, podrán seguir el ritmo de los constantes cambios en la tecnología para brindar mejor rendimiento en la red.

Capítulo II: Fundamentación Teórica

En el segundo capítulo se recopila información relevante de fuentes bibliográficas confiables, que permita comprender la teoría para el desarrollo del presente trabajo, enfocado a la automatización de configuraciones de red mediante la implementación de infraestructura como código (IaC). Por tanto, se abordan diferentes conceptos relacionados con la automatización de redes e IaC, además, se investiga sobre herramientas para ejecutar infraestructura como código, y los aspectos técnicos necesarios para entender su funcionamiento, así como la gestión de recursos, mediante configuraciones y el uso de formatos de datos compatibles, con la finalidad de implementar y gestionar centralizadamente las configuraciones de red en entornos de infraestructura.

2.1. Automatización de red

Se conoce como una metodología a la automatización de redes, la cual permite a los individuos tener menor participación humana en procesos de implementación, configuración, operación y administración de dispositivos tanto físicos como virtuales que pertenecen a cualquier tipo de red o centro de datos. Ya que, muchos de estos procesos que tradicionalmente se realizan de forma manual puedan ser reemplazados por un software (Pérez et al., 2021).

Para Tandazo (2022), la automatización de redes, no se limita a ningún tipo de red ya sea, redes de área extensa (WAN), redes de área local (LAN), redes definidas por software (SDN), redes en la nube o redes inalámbricas, independientemente de la cantidad de dispositivos, por tanto, es altamente aplicable para reducir tiempos en procesos de administración en la red, debido a que, evita configuraciones repetitivas, minimización de errores humanos en la gestión, además, reduce el mantenimiento, los costos operativos, y tiempo de inactividad de la red, lo cual convierte a una red eficiente, escalable y confiable.

Además, los procesos de gestión de configuraciones que se realizan continuamente en los dispositivos pueden ser adoptados en diferentes tipos de redes que facilite la administración de redes complejas y dinámicas (Arzo et al., 2021). Estos procesos pueden ser automatizados mediante diferentes métodos, como el uso de interfaces de línea de comandos (CLI) a través de la programación de scripts, interfaz gráfica de usuario (GUI), y también con la implementación de interfaces de programación de aplicaciones (API) o uso de software de automatización definido (Froehlich & Scarpati, 2022).

2.1.1. Automatización de red con scripts

La automatización de red basada en scripts, y en referencia a Froehlich & Scarpati (2022) utiliza un conjunto de secuencia de comandos, los cuales son escritos desde cero en un lenguaje de programación, comúnmente de código abierto como Python, Perl, Ruby y Bash, debido a su flexibilidad y fácil de entender por el programador para automatizar la infraestructura. Además, los scripts pueden ser personalizados en base a requerimientos específicos de la red, ya que se administran estos archivos por separado y son ejecutados directamente en los dispositivos de la red mediante interfaces de línea de comandos (CLI). Este tipo de automatización es muy usado en entornos que requieren procedimientos consistentes, repetitivos y de uso general. Por otro lado, los scripts solían utilizar un kit de desarrollo de software (SDK) para interactuar con la API del proveedor de la red o en la nube, y operar en la infraestructura con los servicios que proporcionen (Morris, 2021).

Figura 3

Ejemplo de script para crear un servidor

```
import 'cloud-api-library'

network_segment = CloudApi.find_network_segment('private')

app_server = CloudApi.find_server('my_application_server')
if(app_server == null) {
  app_server = CloudApi.create_server(
    name: 'my_application_server',
    image: 'base_linux',
    cpu: 2,
    ram: '2GB',
    network: network_segment
  )
  while(app_server.ready == false) {
    wait 5
  }
  if(app_server.ok != true) {
    throw ServerFailedError
  }
  app_server.provision(
    provisioner: servermaker,
    role: tomcat_server
  )
}
```

Nota. Obtenido de (Morris, 2021).

La Figura 3 muestra un ejemplo escrito en lenguaje Python para definir la automatización en la red mediante script, trata de la creación de un servidor de manera programática, mediante la definición de recursos como CPU, memoria, imagen del sistema operativo y función que se aplicará, además implementa lógica de control para evitar que el servidor sea duplicado, entonces, al terminar lo mencionado, se procede a configurar el mismo. Sin embargo, al utilizar el script, no maneja cambios en los atributos del servidor y puede ser complejo cuando se deba gestionar múltiples servidores, por tal razón, se debería modificar el script manualmente según sea necesario o escribir uno nuevo.

2.1.2. Automatización de red con software

La automatización de red basada en software utiliza plataformas o herramientas que ofrecen plantillas para crear, configurar y ejecutar tareas en los dispositivos presentes en la red como routers, switches, firewalls, entre otros, sin la necesidad de escribir manualmente scripts de configuración (Froehlich & Scarpatti, 2022). Por tanto, este tipo de automatización permite

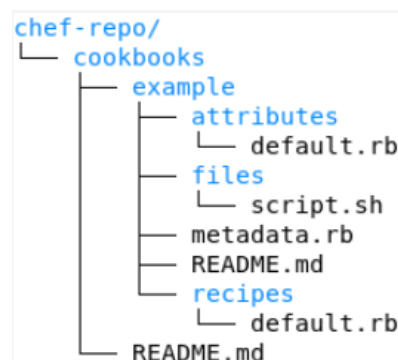
una gestión centralizada de las configuraciones y aprovisionamiento de red para múltiples dispositivos y servicios desde una única plataforma que permita la automatización de la red, mediante interfaces gráficas de usuarios e interfaces de programación de aplicaciones (API), lo cual, reduce la administración de la red.

Las plantillas de las herramientas para la automatización con software contienen una serie de acciones para la configuración de la red, las cuales son enviadas a través de protocolos como SSH, gestores de clientes, entre otros, y posteriormente deben ser probadas en los nodos destinos (Morris, 2021). Sin embargo, cada herramienta utiliza nombres diferentes para referirse a sus módulos de código fuente, por ejemplo, playbooks, cookbooks, manifests y templates. Estos términos se utilizan para organizar, versionar y desplegar estos módulos de configuración.

A continuación, en la Figura 4, se presenta la estructura de una plantilla conocida como ‘cookbooks’ utilizada en la herramienta de automatización Chef, la cual, está escrita en lenguaje de programación Ruby. Este ejemplo muestra un cookbook con el nombre de ‘examples’ el cual, incluye atributos, archivos y recetas con documentación que describe las especificaciones para la configuración y gestión estructurada del entorno (Zammar, 2024).

Figura 4

Ejemplo de una estructura de plantilla cookbooks



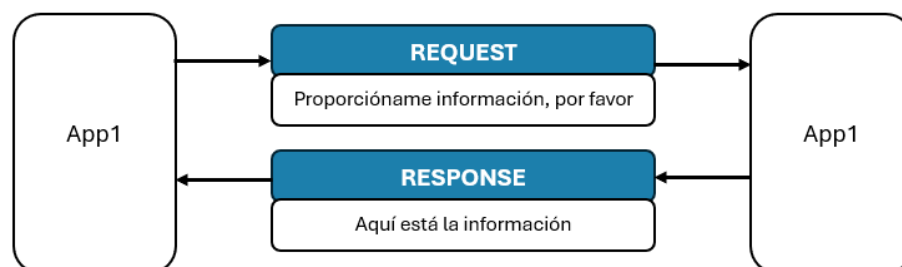
Nota. Obtenido de (Zammar, 2024).

2.1.3. Interfaces de programación de aplicaciones (API)

Las APIs son consideradas como un mecanismo compuesto por un conjunto de funciones diseñadas para actuar como una interfaz para la configuración e interacción con otras aplicaciones o softwares, también se usa para la comunicación entre el software de un dispositivo con el software de otro dispositivo, facilitando tanto el monitoreo como la configuración de dichos componentes (Oswalt et al., 2022). En otras palabras, las APIs se comportan como intermediarios para que los desarrolladores accedan a ciertas funciones específicas de un software o servicio mediante la programación, algunos ejemplos de interfaz de programación de aplicaciones que menciona (Jackson et al., 2020) son, una aplicación web, un sistema operativo, hardware informático, biblioteca de software, entre otros.

Además, la comunicación a través de la interfaz se basa en una arquitectura de solicitud y respuesta como muestra la Figura 5, que representa dos aplicaciones diferentes las cuales intercambian información a través de una API (Jackson et al., 2020). En este esquema, la App1 envía una solicitud hacia la App2 que puede actuar como servidor para procesar la solicitud y responder con la información solicitada, siguiendo el esquema de request-response, siendo muy común en servicios web para definir la estructura y transferencia de mensajes.

Figura 5
Comunicación entre aplicaciones a través de una API



Nota. Adaptada de (Jackson et al., 2020).

Además, mencionan dos diferentes tipos de API desde el punto de vista de la automatización de redes que son northbound y southbound.

- **Northbound APIs**

Este tipo de API es usado para la comunicación entre un controlador de red con el software de gestión, por tanto, aplicaciones de nivel superior como sistemas de administración de red (NMS) y las interfases gráficas de usuario (GUI) permite interactuar con el controlador y que sea posible monitorear y gestionar la red (Jackson et al., 2020). Por lo general, la mayoría de APIs emplean cifrado para proteger la información y garantizar la seguridad.

Un ejemplo es cuando un operador de red accede a un controlador para administrar la red, donde se utiliza una API northbound basada en REST (Transferencia de estado representacional) para transferir información al software de gestión, adicional para asegurar la información se implementa un cifrado TLS (Transport Layer Security) (Jackson et al., 2020). Con la finalidad de permitir la integración de herramientas de IaC, para desplegar configuraciones, servicios o políticas de red. Según Oswalt et al. (2022) , las APIs REST, son conocidas como RESTful, que puede adherir los sistemas o dispositivos de una red como controladores, esto se basa en una arquitectura cliente-servidor, donde no se guarda ningún estado de este modelo y es comúnmente usando para la comunicación en contexto de HTTP.

- **Southbound APIs**

Las APIs Southbound pueden ser usadas para aplicar cambios desde el software de gestión del controlador hacia los dispositivos individuales en la red (Jackson et al., 2020), ya que, si un operador de red realiza cambios en dispositivos como switches, routers o puntos de acceso inalámbrico, el controlador a través del software de gestión enviará los cambios correspondientes a los dispositivos. Por tanto, a través de una interfaz programática, las APIs permiten la interacción con los componentes de una red, ya que facilita tanto la modificación del plano de datos en un dispositivo y también aspectos relacionados con su configuración y políticas, así garantiza la distribución uniforme y coherente de los cambios en las infraestructuras de red.

2.2. Formato de datos

Los autores Jacobs & Walsh (2004), definen a los formatos de datos como una representación para la interpretación correcta de los datos que codifican información, por tanto, no son lenguajes de programación. Entonces, un formato de datos estructurado permite el intercambio de información y procesamiento automático, facilitando la interoperabilidad para compartir datos entre plataformas o sistemas diferentes de manera estandarizada, eficiente y flexible, lo que permite una evolución gradual dando como resultado mejoras y nuevos formatos de datos más robustos, a comparación del primer formato HTML que era estático, no extensible y fue utilizado en el servicio de Internet, World Wide Web.

Los formatos de datos pueden representar varios tipos de datos, para la comunicación de palabras, números y objetos, los cuales pueden presentarse en diferente manera, como string, enteros, booleanos para formar estructuras de datos complejas que almacenen valores u objetos mediante listas o matrices, también permite organizar los valores mediante pares de clave y valor, sin embargo, los tipos de datos varía según la implementación (Edelman et al., 2018).

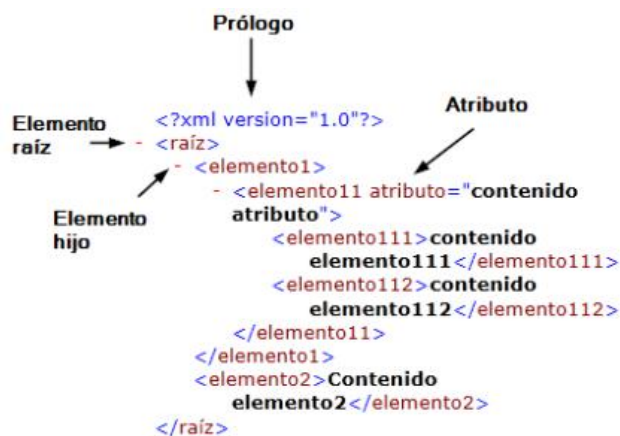
Los formatos de datos están presentes en aplicaciones como la automatización y la programabilidad de la red, por tanto, para realizar el proceso de comunicación, cada herramienta de infraestructura como código trabaja con diferentes formatos de datos según su compatibilidad, para la transmisión de información, o a la vez puede soportar más de un tipo de formato, además se debe considerar la seguridad del intercambio de datos en un entorno abierto. Algunas características ideales de un formato de datos incluyen una estructura jerárquica para indicar las relaciones entre objetos, legibilidad para humanos, capacidad de actualización del archivo, admitir datos binarios y texto, permitir el control de versiones para manejar la evolución gradual de la información, entre otros (Greenfield et al., 2015).

2.2.1. XML

El formato de dato XML, proviene del lenguaje estandarizado SGML que se creó para el almacenamiento e intercambio de información, pero por su complejidad, el Word Wide Web Consortium (W3C) desarrollo XML como un Lenguaje de Marcado Extensible (eXtensible Markup Language) (Hernández et al., 2017), por tanto, se considera a XML como un metalenguaje para definir otros lenguajes mediante código y estructurar grandes documentos de manera legible independientemente del sistema operativo. (Silva, 2021)

Según Pompa (2022), un documento XML es de texto plano, que hacen uso de etiquetas definidas por el usuario para determinar los elementos y estructurar el contenido del documento, sin especificar la visualización final de los datos, esto se debe a la separación del contenido y la presentación. Entonces la estructura del archivo contiene prólogo y el cuerpo como se muestra en la Figura 6, en la parte del prólogo, se declara el archivo como XML y puede especificar la versión y codificación específica, por otro lado, en el cuerpo del archivo tiene una estructura jerárquica de árbol ya que, contiene un elemento raíz o padre del que se derivan subelementos, los cuales se los determina mediante etiquetas que tienen el siguiente formato <nombre del elemento>, que indica tanto el inicio como final de cada uno. Además, pueden incluir atributos que proporcionan información adicional sobre el elemento.

Figura 6
Estructura de un archivo XML



Nota. Obtenido de (Martel, 2021)

2.2.2. JSON

El formato de dato JSON, permite el intercambio de datos ligeros para una comunicación entre servidor y cliente, ya que, maneja un bajo volumen de flujo de datos, ahorro de recursos, aumenta la velocidad y reducción en tiempo de transmisión de datos, además, es compatible con herramientas de automatización (Pompa, 2022). Este formato usa sintaxis de JavaScript y contiene dos estructuras de datos: la primera consiste en objetos que contienen pares clave-valor delimitados por llaves '{}', y la segunda es ordenar la estructura de datos, la cual, puede incluir objetos, matrices o listas, permitiendo expresar y decodificar los datos de manera sencilla sin vincularse a ningún lenguaje de programación específico.

En la Figura 7 se muestran dos tipos de formatos de datos que contienen la misma información, en el inciso (a) se muestra un ejemplo con estructura XML, mientras que en el inciso (b) se indica un ejemplo de una estructura JSON. El autor (Yang, 2012), menciona que ambos ejemplos tienen estructuras con métodos de caligrafía similares y legibles, sin embargo, JSON es más conciso con su formato, ya que no considera elemento raíz ni los subelementos hijos, tampoco usa sangría para delimitar los bloques de datos, sino coloca directamente el objeto dentro de la lista llamada 'books', conteniendo los pares clave-valor que indican el nombre del elemento y su valor de características de un libro separado por ":" , además indica el alcance de sus bloques con [] y {}.

Figura 7

Comparación entre formatos de datos XML y JSON

<pre> <booklist> <computer> <name>java</name> <author>jim</author> <price>50.00</price> </computer> </booklist> </pre>	<pre> books=[{ name: java, author: jim, price: 50.00 }]; </pre>
(a)	(b)

Nota. Obtenido de (Yang, 2012)

2.2.3. *YAML*

El formato *YAML* se propuso en el año 2001 que significa “*YAML Ain't Markup Language*”, el cual se conoce como un formato mejorado de *JSON*, este es un lenguaje de serialización de datos liviano para intercambiar datos estructurados mediante la creación de archivos de configuración, de manera fácil de leer y editar debido a su gramática sencilla ya que disminuye la cantidad de caracteres estructurales y permite caracteres *Unicode* para representar caracteres en diferentes idiomas, además, es muy usado en archivos de configuración, intercambio de datos entre lenguajes, entre otros para implementación de infraestructura (Ben-Kiki et al., 2009).

YAML, permite a los usuarios declarar tipos de datos mediante nodos, donde cada nodo incluye datos, y puede crear referencias de otros nodos en el mismo archivo. Por tanto, tiene una estructura clara con baja posibilidad de errores en la entrada de datos, como menciona (Malin, 2011). Cada nodo representa una estructura única de datos nativos ya sea listas, escalares y pares de clave-valor para crear estructuras complejas, además, cada nodo puede incluir etiquetas locales y globales para delimitar e identificar el conjunto de valores. Las etiquetas locales inician con un signo de exclamación y son particulares de una sola aplicación, mientras que las etiquetas globales se definen como *URI* y pueden utilizarlas en todas las aplicaciones. (Eriksson & Hallberg, 2011)

Este formato de archivo utiliza las siguientes sintaxis: la sangría de espacios en blanco para delimitar la jerarquía de datos, se indica con un guion (-) la continuidad para mencionar listas de elementos y los pares clave-valor se separa con dos puntos, además los comentarios inician con un “#”, los nodos referenciados se marcan con un “*” y es opcional colocar entre comillas las cadenas de texto (Edelman et al., 2018).

En la Figura 8, se evidencia un archivo en formato *YAML* para la configuración de una interfaz de red, donde en la parte superior indica la clave y valor raíz, seguido de las

características de la interfaz como su nombre, descripción, estado habilitado y tipo de IP para colocar a continuación una lista de direcciones IP con su respectiva máscara de subred cada una representa elementos de la lista que inician cada una con un guion (-).

Figura 8

Ejemplo de formato de dato YAML

```
ietf-interfaces:interface:
  name: GigabitEthernet2
  description: Wide Area Network
  enabled: true
  ietf-ip:ipv4:
    address:
      - ip: 172.16.0.2
        netmask: 255.255.255.0
      - ip: 172.16.0.3
        netmask: 255.255.255.0
      - ip: 172.16.0.4
        netmask: 255.255.255.0
```

Nota. Obtenido de (Walton, 2020)

2.3. Sistema de control de versiones

El Version Control System (VCS), es una práctica de software que permite el almacenamiento de archivos y de todas las versiones pasadas y mejoradas en una base de datos, como lo menciona (Zolkifli et al., 2018), por tanto, es posible el seguimiento de cada cambio realizado en el archivo, ya que, varios equipos o personas puede trabajar al mismo tiempo en el mismo código fuente, además, en caso de errores puede regresar a una revisión anterior para continuar el desarrollo. El VCS, evita procesos manuales y pérdidas de los archivos, facilitando la gestión del código. Morris (2016), señala que al definir el código e introducirlas a un sistema de control de versiones (VCS) se obtiene algunas características: (Vuorre & Curley, 2018)

- Ramificación y fusión del código: evita conflictos al trabajar en la fuente de código principal entre diferentes equipos, ya que, pueden ser no compatibles, por tanto, puede trabajar independientemente en diferentes ramas del código y posteriormente volver a fusionar con el código troncal.

- Trazabilidad: cada cambio queda registrado en una estructura jerárquica de árbol que contiene las carpetas y archivos, indicando quién realizó el cambio y qué se modificó.
- Correlación: se mantiene los archivos de configuración y especificaciones en control de versiones es más sencillo rastrear posibles problemas, mediante las etiquetas y números de versión.
- Automatización: cada cambio que es confirmado del código puede activarse acciones automáticamente, como pruebas continuas e integraciones de los cambios frecuentes de código.

2.3.1. Git

Es un sistema de control de revisión distribuido en plataformas de desarrollo, a través de una licencia de software libre para la gestión de código, que fue creado por Linus Torvalds en 2005 (Spinellis, 2012). Git permite mantener el historial de los cambios para la gestión de infraestructuras como código y realizar un trabajo colaborativo sin necesidad de duplicar archivos, además, proporciona a los desarrolladores una copia privada y completa del repositorio, permitiendo trabajar independientemente y sincronizar los cambios. Por otro lado, monitorea el estado de todos los archivos y gestiona las revisiones, manteniendo un conjunto de valores de la configuración en cada repositorio, como nombres, correo electrónico del usuario del repositorio. Este sistema es posible que no venga preinstalado en algunos sistemas operativos como Windows, macOS, pero en Linux es probable que ya se encuentre instalado, sin embargo, se puede descargar desde su sitio oficial.

De acuerdo con Blauw (2018), Git utiliza una estructura de múltiples árboles, para proporcionar un control flexible y gestión eficiente para el flujo de trabajo, el cual, organiza los archivos en cuatro componentes:

- **Directorio de Trabajo:** es un espacio para el almacenamiento del código fuente correspondiente a la rama¹ en la que se está trabajando, en caso de cambiar de rama, el contenido del directorio se actualiza automáticamente, ya que cada rama tiene su propio historial y permite reflejar el estado actual del proyecto.
- **Área de preparación:** se considera un área intermedia, para almacenar los cambios antes de ser confirmados, esto permite una revisión previa al código, para posteriormente sincronizar con el código fuente original y se guarden permanentemente.
- **Repositorio local:** todos los archivos y carpetas que pertenecen al código fuente se encuentran almacenados en este repositorio, es decir todo su historial de confirmaciones de cambios (commit) en el proyecto, incluidas las ramas, por tanto, ayuda a recuperar estados anteriores del proyecto.
- **Repositorio remoto:** es una copia del repositorio local que puede ser almacenado en un servidor externo como GitHub, para realizar cambios al código desde ubicaciones remotas, facilitando un trabajo colaborativo.

2.3.2. Pruebas de código

Es una metodología para garantizar la fiabilidad de una infraestructura de red gestionada mediante código, estas pruebas son estructuradas y priorizan la escritura de la secuencia de comandos antes de crear y modificar la infraestructura, optimizando el desarrollo. Además, existen incluye varios tipos de pruebas, las cuales en referencia a (Jackson et al., 2020) y (Morris, 2021) son las siguientes:

- **Pruebas Unitarias:** este se centra en pequeñas partes del código, la cual se interesa por la parte lógica y el funcionamiento de una única función del código como recursos, configuraciones u operaciones específicas.

¹ Ramas: líneas de desarrollo independientes que permiten trabajar de manera separada en tareas o correcciones sin comprometer el código fuente troncal en el repositorio.

- Pruebas de integración: verifica el funcionamiento entre varios componentes que trabajan entre sí, ya que, si se desarrolla diferentes módulos por distintos dispositivos se debe asegurar la integración.
- Pruebas funcionales: también se conoce como prueba de extremo a extremo, ya que se verifica el sistema o infraestructura de manera completa en base a los requerimientos especificados para que operen todas las partes correctamente en conjunto.
- Pruebas progresivas: es una combinación de pruebas unitarias con pruebas más integradas, ya que, antes de la integrar los cambios, permite realizar pruebas y limpia el código de las pruebas fallidas o mejorar la eficiencia del código eliminando errores.

2.4. Infraestructura como Código (IaC)

La infraestructura como código definido por Dominguez-Quintero & Vargas-Lombardo (2021), es la práctica para la automatización de tareas de gestión y aprovisionamiento de infraestructuras de TI (Tecnología de Información) a través del desarrollo de código, es decir, implica especificar su infraestructura en formato de texto utilizando lenguajes de infraestructura y organizar las configuraciones con los formatos de datos, ya que se crea archivos de configuración donde se guarda cada cambio realizado para modificar, ejecutar, configurar y administrar la infraestructura de manera dinámica.

Este tipo de infraestructuras se trata como código fuente con control de versiones, ya que, se convierte en una entidad programable y gestionable, similar al software o un dato, a través de prácticas de desarrollo de software como mecanismos de automatización de pruebas, sistema de control de versiones, orquestación de despliegue e implementación de recursos, además admite metodologías de desarrollo de software como Integración Continua (CI) y Entrega Continua (CD), para garantizar la calidad, eficiencia y confiabilidad de procesos relacionados para la manipulación de la infraestructura (Morris, 2016).

Por tanto, para implementar este enfoque y automatizar las configuraciones de red en los dispositivos se requiere utilizar herramientas de IaC, las cuales permiten dos tipos de configuraciones según (Wågbrant & Dahlén Radic, 2022). La primera trata sobre la configuración basada en agentes, la cual opera en un modelo “pull”, que consiste en instalar agentes en dispositivos de la red para solicitar el envío de configuraciones desde un servidor central, aunque este método puede ser limitante para los dispositivos que tienen sistemas operativos cerrados que impidan la instalación de procesos o software adicionales. Por ello, proveedores de software han trabajado para desarrollar otro tipo de configuración sin agentes basadas en un método “push”, donde no requiere de peticiones, sino las herramientas de automatización enviarán los cambios de las configuraciones a los dispositivos de destino directamente, mediante protocolo SSH.

2.4.1. Lenguaje de infraestructura declarativo

Este tipo de lenguaje es la forma de codificación usada para definir el estado final deseado para la infraestructura de red, sin necesidad de incluir la lógica detallada paso a paso para crear recursos y configuraciones (Morris, 2016). El lenguaje declarativo es comúnmente usado por herramientas de infraestructura como código que son Ansible, Chef, Puppet y Terraform, las cuales, son encargadas de interpretar el código, comparar el estado actual con el deseado y realizar los cambios necesarios para alcanzar ese estado. Por tanto, este enfoque se considera más limpio y directo, ya que, se separa la especificación de lo que se desea lograr, del cómo lograrlo, pero se aplica para definir entornos reutilizables y cuando no existe mucha variación en los resultados deseados. Además, los lenguajes declarativos usados en IaC se escriben en formato de datos como YAML, JSON, XLM. Un ejemplo proporcionado por (Morris, 2021), define características deseadas para una máquina virtual en formato de código YAML, especificando la imagen, recursos de procesamiento de CPU y cantidad de RAM como indica el ejemplo de la Figura 9, en caso de que se desee aumentar o disminuir los parámetros,

se requiere editar el archivo de configuración y que la herramienta de infraestructura la vuelva a ejecutar.

Figura 9

Ejemplo de código declarativo

```
virtual_machine:
  name: my_application_server
  source_image: 'base_linux'
  cpu: 2
  ram: 2GB
  network: private_network_segment
  provision:
    provisioner: servermaker
    role: tomcat_server
```

Nota. Obtenido de (Morris, 2021).

2.4.2. Lenguaje de infraestructura imperativo

El enfoque de codificación imperativo consiste en especificar una secuencia de comandos que deben ejecutarse en un determinado orden para lograr el estado deseado de la infraestructura (Jackson et al., 2020). A diferencia del lenguaje declarativo, este enfoque detalla los pasos intermediarios para alcanzar el estado final, por tanto, se tiene control en el proceso y es mayormente usado cuando los resultados deseados son diferentes. Además, los scripts a desarrollar permiten incluir lógica de control como bucles, condicionales, funciones, entre otros para la gestión y configuración de los dispositivos en la infraestructura de red. Sin embargo, las herramientas de IaC, pueden incluir enfoques declarativos e imperativos para la gestión de infraestructuras complejas y dinámicas.

2.5. Herramientas de Infraestructura como Código

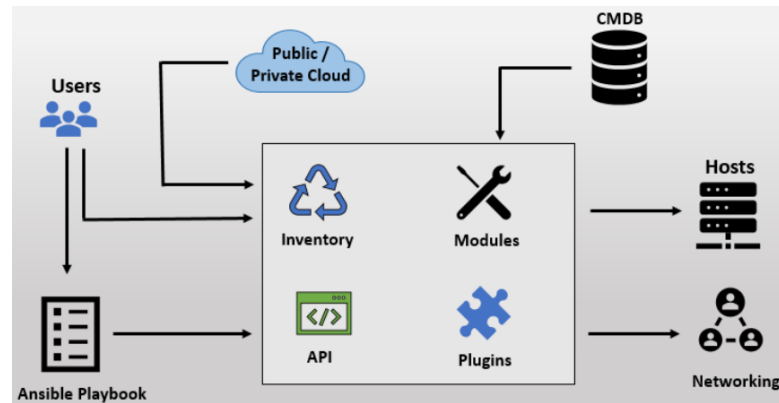
Se realiza la automatización en las redes mediante la implementación de IaC utilizando herramientas específicas, que permite conocer características, configuraciones y funcionalidades de la red con solo leer el código de los archivos de configuración, por tanto, reduce las interacciones manuales de la red, además permite gestionar de manera repetible y reutilizable. Algunas herramientas de Infraestructura como Código, son Cloudformation y Terraform, las cuales se consideran como herramientas de orquestación, ya que permite

provisionar los servidores y dispositivos, es decir preparan la infraestructura de red a través de código especificando los recursos que se desea asignar, sin embargo, las herramientas de automatización más utilizadas son las que se presenta a continuación, las cuales son herramientas de gestión de configuraciones de software en servidores y dispositivos que se ejecutan en la red (Wågbrant & Dahlén Radic, 2022).

2.5.1. Ansible

Es una herramienta de gestión y despliegue de software de código abierto basado en lenguaje de programación Python, desarrollada por RedHat, y es utilizada para la configuración y gestión automática tanto de servidores como de dispositivos de red (Wågbrant & Dahlén Radic, 2022), además, es una herramienta que no requiere de agentes, pero requiere de una instalación inicial en el servidor o máquina virtual para gestionar los demás dispositivos de la red mediante scripts escritos por el usuario usando el formato de datos YAML, los cuales se conectan a los dispositivos mediante SSH para ejecutar las tareas de configuración, entonces Ansible debe iniciar sesión y ejecutarse a través de un playbook y hosts destinatarios que van a ser gestionados.

Por otra parte, tomando como referencia Dutta (2023), indica la arquitectura de Ansible como se muestra en la Figura 10, la cual, se basa en el nodo de control y los hosts que serán administrados. Por tanto, se utiliza un archivo de hosts también como inventario, para enlistar los hosts a configurar, este archivo contiene direcciones IP, nombre de dominio de los hosts para acceder a la conexión remota, también los hosts pueden agruparse según la función que desempeñen, ubicación u otras características, además el inventario se encuentra en el nodo de control.

Figura 10*Arquitectura de herramienta Ansible*

Nota. Obtenido de (Dutta, 2023).

Otra parte importante en la arquitectura son los playbooks, los cuales son archivos que contiene un conjunto de instrucciones para determinar las configuraciones y roles a aplicar a los elementos enlistados en el inventario, utilizando la sintaxis YAML. En referencia a (RedHat, 2022), los playbooks se compone de los siguientes partes como muestra la Figura 11:

- **Plays:** se conoce como jugadas, las cuales son un conjunto de tareas ordenadas para la configuración y ejecución en los hosts, además, pueden existir una o más jugadas en un playbook.
- **Tasks:** también son conocidas como tareas, y se refiere a las acciones que se automatizan dentro de los plays, y permiten realizar llamados a los módulos.
- **Módulos:** son programas pequeños predefinidos de Ansible y son insertados en los hosts conectados para configurarlos y ejecutar tareas para la automatización, logrando el estado deseado en la infraestructura, también es posible escribir módulos nuevos para especificaciones particulares, sin embargo, una vez que cumplen las tareas, ansible los elimina.
- **Variables:** ajusta la ejecución de los playbooks en diferentes sistemas y pueden ser definidas en los inventarios, tareas o plays, ya que, únicamente con un comando se puede ejecutar el playbook, tomando en cuenta distintas versiones de paquetes, la

ubicación o ruta de archivos, sin embargo, para aplicarlas mantiene un orden según la prioridad.

En el ejemplo de una estructura de playbook de la Figura 11 se indican las siguientes partes:

(1) plays, (2) tasks, (3) módulos y (4) variables.

Figura 11

Estructura de playbook en Ansible

```
-1name: Install Apache and start the service
hosts: webservers
vars:4
  web_pkg: httpd
  firewall_pkg: firewalld
  web_service: httpd
  firewall_service: firewalld
  python_pkg: python-httpplib2
  rule: http
tasks:2
  - name: Install the required packages
    yum:3
      name:
        - "{{ web_pkg }}"
        - "{{ firewall_pkg }}"
        - "{{ python_pkg }}"
      state:3latest

  - name: Create web content to be served
    copy:3
      content: "Example web content"
      dest: /var/www/html/index.html
```

Nota. Obtenido de (Yunga, 2018)

2.5.2. Puppet

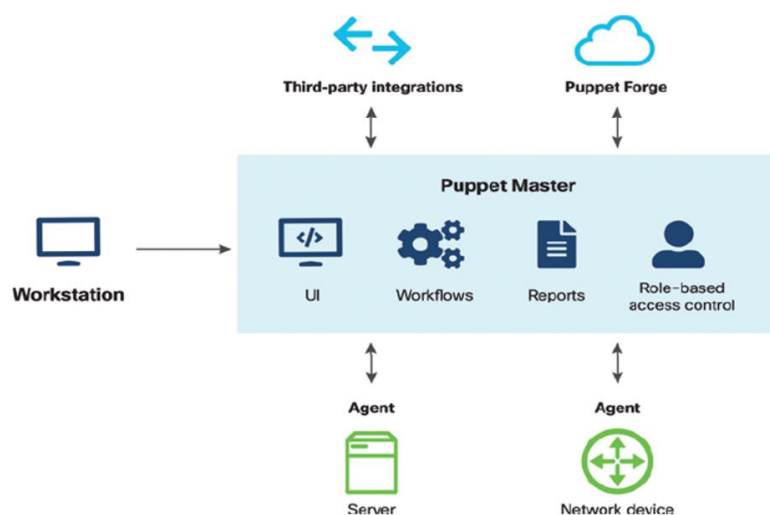
Es una herramienta de gestión de código abierto que se basa en lenguaje de programación Ruby para la automatización de configuraciones de servidores y dispositivos de las infraestructuras de red, la cual fue creada por Puppet Labs Limited, además, como se describe en (Wågbrant & Dahlén Radic, 2022), Puppet utiliza agentes para configurar y ejecutar los dispositivos mediante agentes proxy, sin embargo, puede causar limitaciones, por lo que se creó una herramienta de adición sin agentes llamada Puppet Bot para establecer la comunicación con los hosts mediante SSH o Windows Remote Management (WinRM) y utiliza archivos de manifiesto para indicar los cambios de configuraciones escritos en sintaxis DLS que es un lenguaje específico de dominio.

La estructura de Puppet se basa en un modelo de cliente/servidor, donde los agentes instalados en los hosts actúan como clientes, los cuales informan al servidor conocido como Puppet Master sobre el estado de las tareas que se están ejecutando, además, se encarga de la administración de los dispositivos y analizar la información de las configuraciones recibidas por parte de los agentes, con la finalidad de crear catálogos para definir cómo debe configurarse el host específico (IPCisco, 2020). Estas guías se envían al host y compara el estado actual con el catálogo, en caso de existir cambios deben ser aplicados, posteriormente el host vuelve a enviar el informe de estado al servidor. Por otra parte, el servidor proporciona otras características como interfaz gráfica, integración con otras herramientas, gestionar permisos de acceso y un repositorio llamado Puppet forge para compartir archivos de manifiesto como se muestra en la Figura 12 de la arquitectura de Puppet (Jackson et al., 2020).

Otros componentes en la arquitectura de Puppet, los autores Rahman & Parnin (2023), menciona a Puppet Manifest, que son archivos para escribir un conjunto de instrucciones de automatización en un lenguaje declarativo, los cuales tiene una extensión. pp (Puppet policy) y se almacenan en el nodo central que es Puppet Master. Además, puede incluir clases, que actúa como marcador de posición, recursos y variables para la construcción del código.

Figura 12

Arquitectura de herramienta Puppet

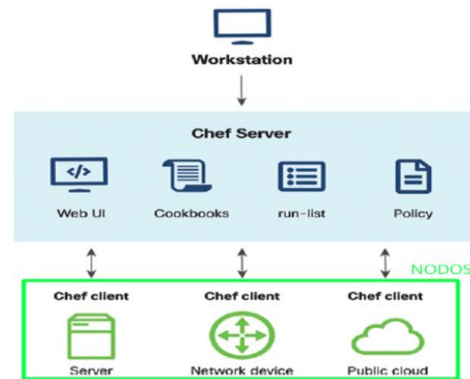


Nota. Obtenido de (Jackson et al., 2020)

2.5.3. Chef

Es una herramienta de código abierto para la gestión de configuraciones en dispositivos o servidores de red, convirtiéndola en una infraestructura como código. Chef sigue un modelo de cliente-servidor para la comunicación, también, se basa en agentes y es escrito con un lenguaje de programación de Ruby, similar a Puppet, además (Kumaran et al., 2022) menciona los componentes de la estructura de la herramienta Chef que indica en la Figura 13 para su funcionamiento.

- Workstation: son las estaciones de trabajo, donde los usuarios pueden crear y comprobar cookbooks para las configuraciones, por tanto, es el punto de interacción entre los desarrolladores y la infraestructura mediante herramientas de línea de comandos como chef y knife.
- Servidor Chef Infra: actúa como centro de datos para el almacenamiento de los cookbooks y políticas que se aplicarán a los nodos conectados al servidor chef para las configuraciones de cada uno.
- Cliente Chef Infra: se refiere a los nodos que serán administrados por el chef server, además los clientes solicitan detalles de configuración, como recetas o cookbooks hacia el chef server para configurar el estado deseado específico de cada nodo, además se informa al servidor chef sobre las actualizaciones, también, se debe tener en cuenta que, en cada nodo gestionado, se debe instalar un cliente chef.
- Cookbooks: es un archivo que contiene agrupaciones de recetas (recetas), recursos, atributos, archivos para distribuir información, bibliotecas personalizadas, entre otros, para definir el conjunto de instrucciones de configuraciones y políticas que se aplicaran en cada nodo, estos archivos, son compartidos desde el Workstation hacia el servidor chef infra, para respaldar los requisitos de la automatización. Los cookbooks son escritos con un lenguaje específico de dominio (DLS) que pueden basarse en Ruby.

Figura 13*Arquitectura de herramienta Chef*

Nota. Obtenido de (Jackson et al., 2020)

De las herramientas de infraestructura como código mencionadas, la Tabla 1 presenta una comparativa de sus aspectos técnicos relacionadas con el software, como sistema operativo compatible, lenguaje, la estructura de datos y métodos de comunicación, esto permite reconocer las similitudes y diferencias entre ellas, para automatizar las configuraciones de red.

Tabla 1*Características de herramientas de infraestructura como código (IaC)*

Características	Ansible	Puppet	Chef
Sistema Operativo	Linux, macOS	Linux, macOS, Windows	Linux, macOS
Versión Actual	11.1.0 (2024)	7.32.1 (2024)	18.5.0 (2024)
Implementación	Sin agentes	Con agentes	Con agentes
Lenguaje de desarrollo	Python	Ruby	Ruby
Tipo de lenguaje	Declarativo/Imperativo	Declarativo	Declarativo
Comunicación	SSH	SSH/ WinMR/ SSL	HTTP/SSL
Estructura de datos	YAML	YAML/DLS	DLS

Nota. Adaptada de (Wågbrant & Dahlén Radic, 2022) y (Yunga, 2018)

Capítulo III: Implementación y funcionamiento

Para este apartado se considera la selección de los requerimientos de software para el entorno de simulación, así como el hardware tanto físico como virtuales, para llevar a cabo la etapa de diseño de tres infraestructuras de red compatibles con la herramienta de automatización elegida. Además, se describirá la etapa de implementación, especificando la integración de la programabilidad mediante el desarrollo de los scripts para llevar a cabo la configuración automatizada de los dispositivos en los tres diferentes escenarios planteados, con la finalidad de verificar el funcionamiento de la herramienta de automatización mediante la ejecución de los archivos de configuración y garantizar una gestión centralizada.

3.1. Requerimientos de software

Esta sección se identifican los requerimientos necesarios que se debe cumplir para seleccionar la herramienta de simulación y la herramienta de automatización, con el objetivo de llevar a cabo el planteamiento de las topologías de red, ya que, estas deben ser compatibles para verificar el funcionamiento de los scripts de automatización de configuraciones en equipos de red, garantizando así el cumplimiento del objetivo del proyecto.

Para dicho proceso de evaluación y selección de las herramientas se considera el análisis realizado por (Cuzme-Rodríguez et al., 2020), para un caso de estudio sobre la evaluación de la calidad del software.

Por lo tanto, los requerimientos considerados se determinan a partir de las características establecidas en el modelo ISO/IEC 25010, las cuales, se muestran en la Figura 14, para determinar la calidad del software según las propiedades que ofrece cada herramienta (ISO 25010, 2024).

Cabe mencionar, que no se van a considerar todas las características de la norma, sino se evaluará desde ciertos contextos, enfocados a las necesidades más críticas que debe cumplir

el software para el desarrollo del proyecto, ya que, otros criterios son menos aplicables y genera una evaluación más extensa sin información importante.

Para ello, se evaluarán distintas plataformas con opciones similares para seleccionar aquella que cumpla con el mayor número de requerimientos de prioridad alta.

Figura 14

ISO/IEC 25010: Características para evaluar herramientas de software

CALIDAD DEL PRODUCTO SOFTWARE								
ADECUACIÓN FUNCIONAL	EFICIENCIA DE DESEMPEÑO	COMPATIBILIDAD	CAPACIDAD DE INTERACCIÓN	FIABILIDAD	SEGURIDAD	MANTENIBILIDAD	FLEXIBILIDAD	PROTECCIÓN
COMPLETITUD FUNCIONAL	COMPORTAMIENTO TEMPORAL	COEXISTENCIA	RECONOCIBILIDAD DE ADECUACIÓN	AUSENCIA DE FALLOS	CONFIDENCIALIDAD	MODULARIDAD	ADAPTABILIDAD	RESTRICCIÓN OPERATIVA
CORRECCIÓN FUNCIONAL	UTILIZACIÓN DE RECURSOS	INTEROPERABILIDAD	APRENDIZABILIDAD	DISPONIBILIDAD	INTEGRIDAD	REUSABILIDAD	ESCALABILIDAD	IDENTIFICACIÓN DE RIESGOS
PERTINENCIA FUNCIONAL	CAPACIDAD		OPERABILIDAD	TOLERANCIA A FALLOS	NO-REPUDIO	ANALIZABILIDAD	INSTALABILIDAD	PROTECCIÓN ANTE FALLOS
			PROTECCIÓN FRENTE A ERRORES DE USUARIO	RECUPERABILIDAD	RESPONSABILIDAD	CAPACIDAD DE SER MODIFICADO	REEMPLAZABILIDAD	ADVERTENCIA DE PELIGRO
			INVOLUCRACIÓN DEL USUARIO		AUTENTICIDAD	CAPACIDAD DE SER PROBADO		INTEGRACIÓN SEGURA
			INCLUSIVIDAD		RESISTENCIA			
			ASISTENCIA AL USUARIO					
			AUTO-DESCRIPTIVIDAD					

Nota. Obtenido de (ISO 25010, 2024)

De acuerdo con la norma ISO/IEC 25010, mostrada en las Figura 14, se consideran tres características para establecer los requerimientos de las herramientas de software: compatibilidad, eficiencia de desempeño y flexibilidad. Dentro de cada una de estas se desglosa elementos específicos que ayudan a la evaluación del sistema. En primer lugar, al hablar de compatibilidad, se considera el elemento de coexistencia e interoperabilidad, donde se identifica la integración con otras plataformas en un mismo entorno. Como segundo punto, la eficiencia de desempeño se analiza mediante el uso de recursos, y la capacidad del sistema para ofrecer un rendimiento óptimo. Finalmente, la característica de flexibilidad se valora mediante aspectos como la instabilidad, disponibilidad y la capacidad de interacción, permitiendo que la herramienta se adapte a distintos escenarios de simulación y brinde facilidad de aprendizaje para el usuario.

Otro criterio tomado en cuenta para establecer los requerimientos para la selección de la herramienta de automatización es la seguridad, ya que, se enfoca en la protección de información, y permite reducir posibles acciones maliciosas (ISO 25010, 2024), este criterio, considera las siguientes subelementos: la confidencialidad, la cual, es responsable de evitar acceso de productos o usuarios no autorizados a los datos, también se considera la subcaracterística de integridad, para impedir modificaciones o pérdida de datos por personal no autorizado o errores producidos por el software, por tanto, esto garantiza un entorno de trabajo confiable.

3.1.1. Selección de entorno de simulación

Para la automatización de configuraciones en dispositivos de red, se requiere seleccionar el entorno de simulación que permita validar la funcionalidad de la herramienta de IaC. Por tanto, se analizará tres herramientas utilizadas en el campo de redes de datos para alojar las topologías a simular, las cuales, se describen a continuación:

- **EVE-NG**

La herramienta de simulación Emulated Virtual Environment-Next Generation permite diseñar y crear topologías de redes complejas a través de su interfaz gráfica web, además, permite la compatibilidad con distintos proveedores de dispositivos como routers, conmutadores y firewalls. EVE-NG es compatible con hardware Linux para su instalación, sin embargo, no es compatible con MAC, ya que puede dar inconvenientes en la ejecución de las imágenes de dispositivos. También, tiene la opción de instalar el software como máquina virtual por lo que es necesario un hipervisor, sin embargo, se debe considerar los recursos de hardware, para la simulación de dispositivos, ya que, es muy demandante (EVE-NG, 2025). Por ende, esta herramienta tiene alta escalabilidad, eficiencia y flexibilidad para crear y validar los diseños de topologías de red antes de implementarlos en escenarios reales.

- **eNSP**

La herramienta de software Enterprise Network Simulation Platform (eNSP), desarrollada por Huawei, puede ser instalada mediante el registro del usuario en la página oficial, para acceder y descargar el software, el cual es compatible con varios sistemas operativos. Sin embargo, está destinada al diseño, creación y configuración de topologías de red tanto pequeñas como robustas exclusivamente con productos de comunicación del fabricante Huawei. eNSP es una herramienta de emulación que admite una amplia variedad de protocolos y alta interoperabilidad con herramientas de virtualización y analizador de tráfico como Wireshark. Además, cuenta con una interfaz gráfica, facilitando la interacción con los usuarios, lo que reduce la curva de aprendizaje. Por tanto, es una herramienta que permite el aprendizaje de nuevas tecnologías y familiarización en entornos de configuración de dispositivos de red de Huawei, proporcionando escalabilidad, seguridad en la red y rendimiento, en el entorno simulado (Huawei, 2024).

- **GNS3**

Su nombre extenso es Graphical Network Simulator 3, el cual, es un software de código abierto, compatible con sistemas operativos Windows, Linux y MAC, y que consta de dos componentes para su funcionamiento: la interfaz gráfica de usuario (GUI) y la máquina virtual GNS3, por tanto, requiere de una herramienta de virtualización. Ambos componentes se complementan para reducir sobrecarga en la PC local, ya que, en la GUI permite a los usuarios crear las topologías mientras en la VM se puede ejecutar los procesos de los dispositivos (GNS3, 2016). Además, es considerado como un emulador gratuito, ya que, se configura y ejecuta imágenes reales de dispositivos de diferentes fabricantes, como: cisco, juniper, arista y soportar varios protocolos de red, también puede simular equipos que pueden ser creados por GNS3 para simular sus funcionalidades y ser aplicados a distintos escenarios (Agapidis, 2024). Siendo posible simular topologías pequeñas, robustas y mixtas, es decir, permite la conexión

entre equipos reales y virtuales que pueden estar alojados en una máquina local, en servidores o en la nube. Por tanto, sus prestaciones son óptimas para realizar pruebas en el entorno de redes.

- **Evaluación de la herramienta de simulación**

La evaluación de las tres herramientas descritas se basará en los requerimientos de compatibilidad, eficiencia de desempeño y flexibilidad, establecidos en el apartado 3.1. Para ello, se plantea abreviaturas para cada uno, los cuales, se muestran en la Tabla 2, facilitando la identificación de los términos sin recurrir a su terminología extensa.

Tabla 2

Abreviaturas de requisitos para evaluación del software de simulación

Descripción	Abreviatura
Requerimiento de compatibilidad	CRe
Requerimiento de eficiencia de desempeño	EDRe
Requerimiento de flexibilidad	FRe

Una vez definido las abreviaturas de los requerimientos en la Tabla 2, se procede a detallar los requisitos para evaluar en base a la prioridad requerida para el desarrollo del proyecto. Por tanto, se establece tres niveles de prioridad: alta, media y baja, indicando si el requerimiento es fundamental, parcialmente necesario o sin ningún impacto significativo.

a) Requerimiento de compatibilidad para software de simulación

Este requerimiento evalúa la compatibilidad del software con otros productos, en particular, la subcaracterística de coexistencia busca analizar la funcionalidad del software al ejecutarse en diferentes sistemas operativos y de manera simultánea con otras herramientas como analizadores de tráfico, sin implicar intercambio directo de información, por otro lado, el elemento de interoperabilidad evalúa la compatibilidad con plataformas de networking de distintos fabricantes, además de soportar protocolos y tecnologías para configuraciones de

dispositivos en simulaciones de red de manera efectiva. Asimismo, se centra en la capacidad de comunicación con la herramienta de IaC, permitiendo trabajar en conjunto, bajo el mismo entorno de simulación. En base a lo mencionado, la Tabla 3, detalla el requerimiento de compatibilidad compuesta por sus elementos de coexistencia e interoperabilidad, asignándole su prioridad correspondiente, para evaluar la herramienta y seleccionar la más adecuada.

Tabla 3

Requerimientos de compatibilidad para software de simulación

Requerimiento de compatibilidad (CRe)				
Nº	Definición del requerimiento	Prioridad		
		Alta	Media	Baja
Requerimiento de coexistencia				
CRe1	La herramienta de simulación debe ser compatible para la instalación en diferentes sistemas operativos, sin necesitar configuraciones extras.	X		
CRe2	Permite la ejecución sin interferencias y de forma simultánea con herramienta de analizador de tráfico.			X
Requerimiento de interoperabilidad				
CRe3	Permitir la integración de diferentes plataformas de networking en el mismo entorno.	X		
CRe4	Capaz de soportar diferentes protocolos y tecnologías para la configuración de equipos, además garantizar el flujo del tráfico.	X		

CRe5	El software debe permitir la integración e interacción con herramientas de automatización y máquinas virtuales alojadas en herramienta de virtualización.	X
CRe6	Requiere de la interacción con sistemas de monitoreo o servicios en la nube para una gestión remota.	X

b) Requerimiento de eficiencia de desempeño para software de simulación

El requerimiento se enfoca en garantizar eficiencia de desempeño de la herramienta de simulación bajo determinados criterios de uso, por tanto, el producto de software debe cumplir con sus funciones optimizando el consumo de recursos del sistema, como almacenamiento, RAM y CPU, ya que, es fundamental para obtener un rendimiento estable, evitando una sobrecarga que afecte negativamente a la eficiencia, principalmente en la inicialización y la ejecución de los procesos de configuraciones en los dispositivos de red dentro de las simulaciones. La descripción de cada requerimiento se presenta en la Tabla 4, en base a las características de función de desempeño de la herramienta.

Tabla 4

Requerimientos de eficiencia de desempeño para software de simulación

Requerimiento de eficiencia de desempeño (EDRe)				
Nº	Definición del requerimiento	Prioridad		
		Alta	Media	Baja
Requerimiento de utilización de recursos				

EDRe1	Permite gestionar la RAM y CPU sin exceder lo especificado para evitar fallos y sobrecargas en las simulaciones.	X
EDRe2	La plataforma optimiza el almacenamiento sin generar archivos innecesarios.	X
Requerimiento de capacidad		
EDRe4	Permite la configuración de varios dispositivos dentro de una misma simulación y con la capacidad de informar al usuario los recursos que se están usando.	X
EDRe5	La herramienta de simulación proporciona el acceso simultaneo de múltiples usuarios con permisos para la administración del sistema.	X

c) Requerimiento de flexibilidad para software de simulación

Dentro de requerimiento de flexibilidad se describen ciertos parámetros fundamentales para evaluar la adaptabilidad bajo circunstancias cambiantes en el entorno de simulación, por tanto, se despliegan subcaracterísticas para complementar al requisito que son la inestabilidad para reconocer la complejidad de la misma en caso de requerir complementos o configuraciones extras y el tipo de licencia que posee para acceder a sus prestaciones del sistema, por otro lado, se considera la disponibilidad de la herramienta, para garantizar que ante fallos inesperados se tenga un acceso a documentación y soporte dentro de una comunidad activa, finalmente se considera la característica de capacidad de interacción con el usuario para identificar la facilidad del manejo de la herramienta mediante funciones que esta herramienta, las cuales, puede ser interfaz gráfica o interfaz de línea de comandos ya que reduciría la curva de aprendizaje. Por tanto, en la Tabla 5, se detalla los requerimientos en base a la característica

de flexibilidad para determinar el uso de la herramienta en simulación de redes de datos para configuraciones de equipos mediante la automatización.

Tabla 5

Requerimientos de flexibilidad para software de simulación

Requerimiento de flexibilidad (FRe)				
N°	Definición del requerimiento	Prioridad		
		Alta	Media	Baja
Requerimiento de inestabilidad				
FRe1	La herramienta es de instalación sencilla, sin requerir de dependencia complejas de acceder.		X	
FRe2	El proceso de instalación y configuraciones iniciales son intuitivos.			X
FRe3	Debe ser una herramienta gratuita y ofrecer sus funciones sin limitaciones.		X	
Requerimiento de disponibilidad				
FRe4	El sistema garantiza la recuperación de la simulación en caso de incorrectas configuraciones cierres y fallos inesperados		X	
FRe5	La herramienta proporciona documentación y una comunidad amplia de sitios oficiales para solucionar posibles errores	X		
Requerimiento de capacidad de interacción				
FRe6	Tiene interfaz gráfica que permite al usuario facilitar el manejo de la herramienta para la simulación de infraestructuras de red de datos.	X		

FRe7	La plataforma tiene interfaz de líneas de comandos para configurar los equipos de red presentes en las topologías de red.	X
-------------	---	---

Ahora, se identifica los requerimientos que obtuvieron la prioridad más alta, los cuales, se presentan en la Tabla 6, para evaluar las tres herramientas de simulación, considerando la calificación con valor de 1 si cumple con el requerimiento, caso contrario se le asignará un valor de 0, por tanto, se elegirá la herramienta que tenga la mayor cantidad de puntos, ya que, se garantiza la opción óptima para el desarrollo del proyecto.

Tabla 6
Evaluación de la herramienta de simulación

Requerimientos	Herramientas de Simulación		
	EVE-NG	eNSP	GNS3
CRe1	0	0	1
CRe3	1	0	1
CRe4	1	1	1
CRe5	1	0	1
EDRe1	1	1	1
EDRe4	1	0	1
FRe5	1	0	1
FRe6	1	1	1
Total	7	3	8

Con los resultados obtenidos en la evaluación de la Tabla 6, se determina que la herramienta que cumple con la mayor cantidad de requerimientos es GNS3, debido a su amplia compatibilidad con múltiples plataformas de networking, soporte de distintos protocolos y

tecnologías, además que permite la interacción con herramientas de automatización, y herramientas de virtualización necesarias para las configuraciones de los dispositivos de red. Por otro lado, la herramienta de Huawei eNSP presenta ciertas desventajas, ya que solo se ejecuta correctamente en Windows 10, y admite únicamente dispositivos de su mismo fabricante, además la integración con herramientas de automatización y virtualización no es directa, sino requiere de complementos y configuraciones adicionales para la administración de configuraciones (Huawei, 2024). Mientras que EVE-NG, tiene una alta compatibilidad, escalabilidad y eficiencia, pero su implementación requiere de recursos computacionales considerables y su instalación se limita únicamente a una máquina virtual de sistema operativo Linux.

Por ende, se selecciona GNS3 como la herramienta de simulación adecuada, debido a requerimientos de eficiencia de desempeño más accesibles y que garanticen la ejecución de las topologías. Siendo sus requisitos mínimos de 8 GB de RAM siendo lo recomendado 16 GB, junto con 4 núcleos, y con almacenamiento disponible de 35 GB (GNS3, 2016), sin embargo, pueden ser gestionados para garantizar su uso eficiente. Además, GNS3 cuenta con el componente adicional que es la máquina virtual, lo cual, mejorar el rendimiento y escalabilidad, permitiendo simular redes robustas, así mismo posee características de flexibilidad y capacidad de interacción con el usuario, lo que reduce la curva de aprendizaje.

3.1.2. Selección de sistemas virtualizados de networking

La herramienta de simulación seleccionada en el apartado 3.1.1 es GNS3, el cual, posee flexibilidad para manejar diversas plataformas de simulaciones de red, sin embargo, las más usadas son las presentadas en la Tabla 7, las cuales, permiten emular los dispositivos de diferentes fabricantes. Por tanto, se identifican características de cada plataforma como el tipo de imagen para la simulación del equipo, la memoria RAM y tamaño aproximado del archivo, con el fin de reconocer sus parámetros necesarios para el funcionamiento y poder realizar

posteriormente los diseños de las topologías, destacando, que la automatización de configuraciones es compatible con cualquier fabricante del equipo.

Tabla 7

Comparación de parámetros de plataformas de networking en GNS3

Parámetros	Arista vEOS	Juniper vMX	Cisco IOS	Mikrotik CHR
Sistema	EOS	Junos OS	IOS / IOSv	RouterOS
Operativo				
Aplicabilidad	Centro de datos, Infraestructuras en la nube	Proveedores de servicios tradicionales	Redes empresariales, campus, formación académica	Redes empresariales pequeñas y medianas como ISP
Uso de hipervisores	QEMU, VMWare VirtualBox	QEMU, VMWare Oracle VirtualBox	KVM VMWare	VMWare KVM VirtualBox Hyper-V
Extensiones de imágenes de dispositivos	.qcow2 .vmdk .ova	.qcow2 .vmdk .img .ova	.bin .ova .qcow2 .img	.qcow2 .vmdk .iso .img
RAM	~2048 MB	~1024 MB	~512 MB	~512 MB
Tamaño de imagen	~612 MB	~1500 MB	~150 MB	~150 MB

Nota. Adaptado de (Arista Networks, 2015), (Juniper Networks, 2024), (Mikrotik, 2025) y (GNS3, 2025)

- **Evaluación de sistemas virtualizados de networking**

Para la evaluación de plataformas de red, se requiere determinar requerimientos para garantizar la funcionalidad del entorno, donde permita simular dispositivos de diferentes fabricantes en diseños de topologías de red alojados en GNS3, por tanto, para el desarrollo del proyecto se va a seleccionar dos sistemas de virtualización, cuyas configuraciones serán automatizadas para obtener eficiencia en la gestión de configuraciones.

Los criterios para la evaluación de plataformas de networking se basa en la norma ISO/IEC 25010, pero se considera principalmente las características: compatibilidad y eficiencia de desempeño, con sus respectivas subcaracterísticas, las cuales, fueron mencionadas en el apartado 3.1.

A continuación, en la Tabla 8 se presenta las abreviaturas de los requisitos de compatibilidad y eficiencia de desempeño, con la finalidad de permitir una rápida identificación de la descripción de cada requerimiento en el proceso de evaluación de la plataforma de virtualización de networking.

Tabla 8

Abreviaturas de los requisitos para evaluación de sistemas de virtualización networking

Descripción	Abreviatura
Requerimiento de compatibilidad	CSvR
Requerimiento de eficiencia de desempeño	EDSvR

a) **Requerimientos de compatibilidad para el sistema de virtualización**

La compatibilidad en los sistemas de virtualización se centra en la capacidad de trabajar en conjunto con otras herramientas de software o componentes en simulaciones alojadas en GNS3, obteniendo un entorno funcional. Respecto a la subcaracterística de interoperabilidad garantiza la comunicación entre equipos virtualizados de diferentes fabricantes, también la interacción con hipervisores para permitir la ejecución de los dispositivos y el intercambio de

información con el servicio de automatización mediante protocolos admitidos para la gestión de configuraciones en los dispositivos, por otro lado, el subelemento de coexistencia determina la ejecución simultánea de diferentes sistemas de networking o herramientas de software sin provocar interferencias que afecte las simulaciones de topologías.

Entonces con lo mencionado, se detalla en la Tabla 9, los requerimientos que respaldan el enfoque de cada subcaracterísticas para el criterio de compatibilidad en sistemas virtualizados de networking.

Tabla 9

Requerimientos de compatibilidad para sistemas de virtualización de networking

Requerimiento de compatibilidad (CSvR)				
Nº	Definición del requerimiento	Prioridad		
		Alta	Media	Baja
Requerimiento de coexistencia				
CSvR1	Los equipos virtualizados de diferentes fabricantes pueden construir una misma topología en el escenario de simulación.	X		
Requerimiento de interoperabilidad				
CSvR2	Los dispositivos presentes en la topología de red son capaces de establecer comunicación a través de diferentes protocolos de red.	X		
CSvR3	El sistema de virtualización admite servicios de automatización para las configuraciones de los equipos mediante la herramienta IaC.	X		
CSvR4	Es posible exportar o importar los dispositivos virtualizados, para reutilizar en otras topologías de red.			X

b) Requerimientos de eficiencia de desempeño para el sistema de virtualización

La eficiencia de desempeño se centra en la optimización de recursos de la máquina anfitriona al ejecutar los equipos virtualizados en GNS3, sin afectar el rendimiento de operación del dispositivo. Dentro de esta característica se despliega el subelemento de uso de recursos, que se encarga de garantizar el funcionamiento del dispositivo virtualizado en base a la cantidad de recursos que demanda como memoria RAM, CPU y almacenamiento, sin sobrepasar los valores especificado, por otro lado, la subcaracterística de capacidad considera los límites máximos para simular una cierta cantidad de dispositivos junto con el servicio de automatización, por tanto, ayuda a identificar un entorno de simulación con rendimiento estable. Estos criterios, se detallan en cada requisito, los cuales, se observan en la Tabla 10, y se asignarán una etiqueta de prioridad para evaluar y seleccionar dos sistemas virtualizados de networking más adecuados.

Tabla 10

Requerimientos de eficiencia de desempeño para sistemas de virtualización de networking

Requerimiento de eficiencia de desempeño (EDSvR)				
N°	Definición del requerimiento	Prioridad		
		Alta	Media	Baja
Requerimiento de utilización de recursos				
EDSvR1	El sistema de virtualización evita sobrecargas en la máquina anfitriona, ya que usa de manera eficiente recursos de RAM y CPU.	X		
EDSvR2	Los tiempos de inicialización de los dispositivos virtualizados son bajos para tener una operación fluida.		X	
Requerimiento de capacidad				

EDSvR3	Es posible ejecutar varios dispositivos virtualizados en la misma topología sin afectar el rendimiento de los equipos.	X
EDSvR4	El equipo virtualizado puede integrarse a una infraestructura de red ya existente sin comprometer su operatividad inicial.	X
EDSvR5	El dispositivo es capaz de tolerar fallas operativas como reinicios, sin alterar el funcionamiento de los demás equipos conectados en la red.	X

Con los requerimientos detallados sobre compatibilidad y eficiencia de desempeño para los sistemas de virtualización de networking, se identifican los que tienen etiqueta de alta, media o baja prioridad, ya que, los de categoría alta son esenciales que se deben cumplir obligatoriamente, los de prioridad media pueden ser omitidos en situaciones específicas, pero puede afectar parcialmente al desarrollo del proyecto, mientras que los de etiqueta baja no influyen en el funcionamiento.

Por tanto, para la evaluación de las cuatro plataformas de virtualización para equipos presentadas en la Tabla 11, se considera únicamente los requisitos de prioridad alta, ya que, son indispensables para un funcionamiento correcto en las simulaciones de topologías de red.

Los valores para la evaluación de los sistemas de virtualización son dos: 1 en caso de cumplir con el requerimiento y el valor de 0 en caso contrario, en base a esto, se van a seleccionar dos plataformas de virtualización que obtengan el mayor puntaje total, las cuales, permitirán automatizar las configuraciones de los dispositivos alojados en GNS3.

Tabla 11
Evaluación del sistema de virtualización de networking

Requerimientos	Sistemas de virtualización de networking			
	Arista vEOS	Juniper vMX	Cisco IOS	Mikrotik CHR
CSvR1	1	1	1	1
CSvR2	1	1	1	1
CSvR3	1	1	1	1
EDSvR1	0	0	1	1
EDSvR3	0	0	1	1
EDSvR4	1	1	1	1
Total	4	4	6	6

Debido a parámetros como la compatibilidad, bajos requerimientos de hardware para su buen desempeño, se selecciona las plataformas de CISCO IOS y Mikrotik CHR para trabajar con la simulación de sus equipos en los diseños de redes alojados en GNS3, estas plataformas destacan por ser livianas, ya que, requiere de una cantidad baja de recursos de aproximadamente 512 MB de RAM y un tamaño de archivos de 150 MB. Por tanto, esto aporta a una mayor escalabilidad debido a sus recursos bajos, permite crear topologías con más dispositivos, sin comprometer el rendimiento ni el funcionamiento tanto de la máquina anfitriona como del dispositivo virtualizado.

Además, equipos de fabricantes CISCO y Mikrotik son muy usados en infraestructuras de redes, especialmente en el ámbito empresarial como proveedores de servicio de internet (ISP), debido a que admite amplios tipos de protocolos para la comunicación entre los diferentes dispositivos, también tienen la capacidad de adaptarse a estructuras ya diseñadas sin afectar la operación inicial, además equipos como Mikrotik son mayormente adquiridos debido a sus bajos costos a comparación de los otros fabricantes (Luna, 2024).

Sin embargo, todos los sistemas de virtualización de networking evaluados en la Tabla 11, admiten nuevas tecnologías como la automatización, creación de infraestructuras de redes sostenibles y la interoperabilidad entre equipos de diferentes fabricantes dentro de un mismo entorno de simulación.

3.1.3. Selección de Herramienta de automatización

En el apartado de selección de la herramienta de Infraestructura de como código (IaC), enfocada a la automatización de configuraciones de dispositivos de red, se considera cinco plataformas comúnmente usadas, que son: Ansible, Puppet, Chef, Terraform y AWS CloudFormation. Por tanto, en la Tabla 12, se recopila características relevantes como la estructura usada para los archivos de código y su función principal que tiene dentro de infraestructuras IaC, permitiendo identificar su modelo de operación.

Tabla 12
Características principales de herramientas de IaC

Características	Ansible	Puppet	Chef	Terraform	AWS CloudFormation
Sistema operativo	Linux, macOS	Linux, macOS, Windows	Linux, macOS	Linux, macOS, Windows	Servicio en la nube
Lenguaje de Infraestructura	Declarativo/Imperativo	Declarativo	Declarativo/Imperativo	Declarativo	Declarativo
Implementación	Sin agentes	Con agentes y	Con agentes	Sin agentes	Sin agentes

		sin			
		agentes			
Lenguaje de desarrollo	Python	Ruby	Ruby	HCL	Desarrollado por AWS
Nombre de archivos de código	Playbook	Manifest o	Cookbook	Archivos con extensión .tf	Plantillas
Formato de archivos de código	YAML	YAML/DLS	(Ruby)	JSON/HCL	YAML/JSON
Enfoque en IaC	Gestión de configuraciones del sistema	Gestión de configuraciones de servidores	Gestión de configuraciones de servidores	Aprovisionamiento de infraestructura en la nube	Aprovisionamiento de infraestructura en la nube
Licencia	Gratuito	Gratuito/pagado	Gratuito/pagado	Gratuito/pagado	Gratuito/pagado
Comunidad y documentación	Activo y Amplia	Activo y Amplia	Activo y Escasa	Activo y Amplia	Activo y Amplia

Nota. Adaptado de (Santoso & Sari, 2022) y (Bruschetti et al., 2023)

- **Evaluación de la herramienta de Infraestructura como Código (IaC)**

Se debe determinar los requisitos que deben de cumplir para elegir el producto de software que brinde funcionalidades adecuadas para el desarrollo del proyecto, que tiene como objetivo la automatización de configuraciones en dispositivos de red, mediante la simulación de distintos escenarios alojados en la herramienta de simulación seleccionada en el apartado 3.1.1.

Los requerimientos para la evaluación de las herramientas de IaC, se definen a partir de la norma ISO/IEC 25010 que permite determinar la calidad del producto de software. Sin embargo, se considera las características de compatibilidad, eficiencia de desempeño y seguridad, cada una está compuesta por sus respectivas subelementos que complementan su evaluación, los cuales, se mencionan en el apartado 3.1.

A continuación, se representa abreviaturas usadas para los requerimientos de compatibilidad, eficiencia de desempeño, y seguridad con la finalidad de una identificación rápida del requisito, como se muestra la Tabla 13.

Tabla 13

Abreviaturas de los requisitos para evaluación de la herramienta IaC

Descripción	Abreviatura
Requerimiento de compatibilidad	CHRe
Requerimiento de eficiencia de desempeño	EDHRe
Requerimiento de seguridad	SHRe

a) **Requerimientos de compatibilidad para la herramienta IaC**

El requerimiento de compatibilidad busca evaluar la capacidad de la herramienta de automatización para integrarse con otros softwares, garantizando su funcionalidad dentro de un mismo entorno. Por otro lado, su elemento de interoperabilidad se centra en garantizar la interacción de la herramienta, ya que debe ser posible el intercambio de información, mediante

protocolos de comunicación, y sea compatible en diferentes infraestructuras virtualizadas de manera local o en la nube, mientras que la subcaracterística de coexistencia se basa en la ejecución simultánea de la herramienta, con otras plataformas IaC, sin causar interferencias, además asegura su compatibilidad para ejecutarse en diferentes sistemas operativos.

Todos estos requerimientos mencionados se detallan en la Tabla 14 permitiendo a la vez analizar la flexibilidad de la herramienta para trabajar en diferentes entornos sin afectar su funcionamiento.

Tabla 14
Requerimientos de compatibilidad para herramienta IaC

Requerimiento de compatibilidad (CHRe)				
N°	Definición del requerimiento	Prioridad		
		Alta	Media	Baja
Requerimiento de coexistencia				
CHRe1	La herramienta de automatización es compatible para su ejecución en diferentes sistemas operativos como Linux, masOS y Windows.	X		
CHRe2	Puede trabajar más de una herramienta de automatización en el mismo entorno de simulación para complementar las configuraciones de la infraestructura.			X
CHRe3	El software debe contar con alternativas de personalización según el entorno donde se ejecute y necesidades del usuario.			X
Requerimiento de interoperabilidad				

CHRe4	El software de automatización permite la integración con herramientas de virtualización y plataformas en la nube para la automatización en diferentes entornos.	X
CHRe5	La herramienta está enfocada para automatizar configuraciones de diferentes fabricantes de dispositivos de red mediante el uso de protocolos de comunicación.	X
CHRe6	Se puede integrar la plataforma en distintos entornos de redes, ya sea en infraestructuras locales, híbridas o servicios en la nube, sin afectar su comunicación.	X

b) Requerimientos de eficiencia de desempeño para la herramienta IaC

El presente requerimiento se centra en evaluar la capacidad que tiene la herramienta de automatización para gestionar adecuadamente el uso de los recursos del sistema, por tanto, se analiza los recursos usados por la herramienta dentro del hardware como: RAM, CPU y almacenamiento, garantizando que no excedan los valores indicados ya que, puede provocar saturaciones que afecten al rendimiento de sus funciones de automatización, además el elemento de capacidad permite evaluar la habilidad de la herramienta para gestionar infraestructuras simuladas, dinámicas y escalables. A continuación, se indica en la Tabla 15 la descripción de cada requerimiento que complementan la evaluación.

Tabla 15
Requerimientos de eficiencia de desempeño para herramienta IaC

Requerimiento de eficiencia de desempeño (EDHRe)				
Nº	Definición del requerimiento	Prioridad		
		Alta	Media	Baja

Requerimiento de utilización de recursos		
EDHRe1	La herramienta de automatización garantiza el uso eficiente de la RAM y CPU para evitar el consumo máximo de recursos durante la ejecución procesos de automatización.	X
EDHRe2	Permite la administración y uso correcto del almacenamiento para evitar acumulación de archivos basura.	X
Requerimiento de capacidad		
EDHRe3	El software de automatización es capaz de configurar alta cantidad de dispositivos de red en topologías de simulación, sin reducir su rendimiento.	X
EDHRe4	Tiene la capacidad de gestionar infraestructuras cambiantes y escalables.	X
EDHRe5	Optimización del tiempo en configuraciones debido a la ejecución de varios procesos simultáneamente en la plataforma.	X
EDHRe6	Permite la herramienta la reutilización de archivos de configuración para optimizar el proceso de implementación y evitar duplicaciones de código.	X

c) Requerimientos de seguridad para la herramienta IaC

El requerimiento de seguridad garantiza la protección de la información que va a ser enviada desde la herramienta de automatización hacia los dispositivos de red presentes en las topologías simuladas. En este contexto, con la confidencialidad busca la transmisión segura y acceso autorizado a las configuraciones de los equipos, mientras la integridad se encarga que el estado de trabajo no se vea afectado en caso de pérdidas, alteraciones o errores en los archivos de configuración. En base a lo mencionado, se muestra en la Tabla 16, la descripción de los requerimientos a evaluar en el contexto de la seguridad.

Tabla 16
Requerimientos de seguridad para herramienta IaC

Requerimiento de seguridad (SHRe)				
N°	Definición del requerimiento	Prioridad		
		Alta	Media	Baja
Requerimiento de confidencialidad				
SHRe1	El software permite una comunicación segura con servicios de la nube mediante APIs que requieren de autenticación con tokens			X
SHRe2	La herramienta admite protocolos seguros como SSH para una transmisión confiable de la información.	X		
SHRe3	Es posible cifrar contraseñas o datos sensibles en archivos de configuración para garantizar un acceso autorizado	X		
SHRe4	La herramienta puede integrarse con controles de acceso a los usuarios, para asignar permisos en base a su rol que desempeñe.		X	

Requerimiento de integridad		
SHRe5	Tiene la capacidad de verificar los cambios realizados en cada configuración automatizada mediante un sistema de versionado.	X
SHRe6	La herramienta puede validar los archivos de configuración antes de la ejecución en cada dispositivo de red.	X
SHRe7	El software permite el registro detallado de cada acción realizada para la configuración de los equipos mediante logs.	X

Una vez definido los requerimientos de compatibilidad, eficiencia de desempeño y seguridad, se identifica aquellos que obtuvieron la etiqueta de prioridad alta, para posteriormente evaluar las cinco herramientas de IaC, como muestra la Tabla 17.

La evaluación se realiza en base a dos valores; en caso de cumplir con el requerimiento se coloca el valor de 1, caso contrario se le asignará un valor de 0, por tanto, se va a seleccionar la herramienta que tenga el total más alto, ya que será el que cumpla con la mayor cantidad de requerimientos esenciales para el desarrollo del proyecto.

Tabla 17
Evaluación de la herramienta de Infraestructura como Código (IaC)

Requerimientos	Herramientas de Infraestructura como Código (IaC)				
	Ansible	Puppet	Chef	Terraform	AWS CloudFormation
CHRe1	0	1	0	1	0
CHRe4	1	1	1	1	1

CHRe5	1	0	0	0	0
EDHRe1	1	1	1	1	1
EDHRe3	1	0	0	0	0
EDHRe4	1	1	1	1	1
EDHRe5	1	1	1	1	1
EDHRe7	1	1	1	1	1
SHRe2	1	1	1	0	0
SHRe3	1	1	1	1	1
SHRe6	1	1	1	1	1
Total	10	9	8	8	7

En base a los parámetros evaluados y presentados en la Tabla 17, la herramienta de IaC con el puntaje más alto es Ansible, siendo la que cumple con la mayor cantidad de requerimientos para un funcionamiento óptimo. Ya que, su enfoque está destinada a la gestión de configuraciones de los dispositivos de red, dentro de una infraestructura ya creada, además tiene una curva de aprendizaje sencilla debido al uso de formato de archivos de código YAML, el cual, es fácil de interpretar por el humano, otro factor, es que no requiere de configuraciones adicionales como instalación de agentes en los nodos a gestionar, por tanto, permite minimizar la administración de configuraciones del sistema (Ansible, 2025).

Por otro lado, las herramientas Puppet y Chef se enfocan a la gestión de configuraciones de servidores o aplicaciones, como la instalación y actualización de paquetes en una máquina virtual previamente creada (Progress Chef, 2025). Mientras que las herramientas como (AWS CloudFormation, 2023) y (Terraform Community, 2025), se centran en el aprovisionamiento de recursos de infraestructuras en la nube, sin embargo, las cinco opciones evaluadas cuentan

con óptimo consumo de recursos, así como la capacidad de administrar topologías escalables y dinámicas.

Además, Ansible proporciona mecanismo de acceso seguro a los equipos, ya que, se enfoca a la configuración de dispositivos de red de manera remota mediante protocolo confiable SSH, el cual, cifra la comunicación y autentica el acceso, sin embargo, en caso de requerir la interacción entre la herramienta IaC y servicios externos como AWS, Azure, se necesita el uso de APIs, lo cual, requiere de autenticación por token ya sea temporales o permanentes. También, es posible cifrar datos sensibles dentro de los playbooks y validar las configuraciones para ejecutarlas, sin requerir de herramientas adicionales.

3.2. Requerimientos de hardware

La máquina anfitriona que se usará debe soportar el consumo de los recursos requeridos para las simulaciones de los diseños de red, y permitir automatizar sus configuraciones mediante la herramienta de IaC elegida en el apartado 3.1.2. La computadora portátil cuenta con las siguientes características: un procesador Intel Core i5-10300H CPU de décima generación con velocidad base de 2.50GHz, 4 núcleos de procesamiento, 16 GB de memoria RAM y con un almacenamiento disponible de 512 GB. Estas características, proporcionan la capacidad y potencia necesarias, tanto en el almacenamiento como procesamiento, para diseñar y gestionar topologías con cantidades escalables de dispositivos dentro de la herramienta de simulación seleccionada en el apartado 3.1.1, correspondiente a GNS3, el cual, estará alojado en un sistema operativo Debian correspondiente a Linux para obtener mejor estabilidad y rendimiento del entorno.

Anexo I se presenta una guía de instalación del software de simulación GNS3 en un sistema operativo Debian, para alojar los diseños de los escenarios destinados a la automatización de configuraciones de red.

3.3. Diseño de topologías de red

El diseño de las topologías de red se plantea con un enfoque de redes empresariales, priorizando la validación de manera práctica las configuraciones automatizadas a los dispositivos de red, a través de Ansible, que es la herramienta de Infraestructura como Código (IaC) seleccionada en el apartado 3.1.2, donde dichos diseños de redes estarán simulados en la herramienta seleccionada en el apartado 3.1.1: GNS3

Los escenarios se enfocan al sector de redes empresariales debido a sus constantes cambios en infraestructuras y alta acogida a nuevas tecnologías innovadoras para operar y contribuir a soluciones que permitan mejorar la eficiencia de operaciones en su red y mantenerse en un mercado competente y creciente, además proporcionar un servicio de calidad a sus usuarios, reducir gastos operativos y errores humanos para la administración (Caisalitin & Quillupangui, 2015). Sin embargo, antes de implementar o modificar las redes reales, se debe validar el funcionamiento de nuevas tecnologías mediante simulaciones, ya que, puede generar costos significativos en la adquisición de hardware o software para el despliegue de nuevas infraestructuras.

En concordancia al alcance planteado en el apartado 1.4 se desplegará tres escenarios escalables, incorporando nuevos dispositivos y funciones para simular distintos entornos, los cuales, se podrán gestionar las configuraciones de forma centralizada desde el nodo principal de control, lo cual, reducirá las configuraciones manuales y el tiempo dedicado a la administración de redes.

Además, los tres diseños de infraestructura de red, se consideran escenarios experimentales, ya que, se establecen los parámetros de simulación para validar la funcionalidad de la automatización en las configuraciones de redes para la gestión de los nodos en entornos controlados (Josheff & Márquez, 2015). Al ser diseños experimentales no dependen de infraestructuras predefinidas, por tanto, pueden ser manipuladas para evaluar

distintas condiciones y determinar configuraciones óptimas de manera práctica en entornos empresariales antes de implementar en escenarios reales. (Blanchet et al., 2021)

Para los diseños de las topologías de red experimentales enfocado al área empresarial se toma en cuenta recomendaciones de (Cisco, 2025), permitiendo identificar la estructura de los dispositivos para satisfacer la conectividad de organizaciones pequeñas y medianas:

Los fundamentos de diseño de una red establecida por Cisco (2021), indican que la implementación de una red es necesario poseer un modelo jerárquico que tenga características de escalabilidad, flexibilidad, redundancia y disponibilidad, sin embargo, la estructura específica dependerá de las características del sitio de implementación, entonces, en ciertos entornos se puede necesitar una, dos o las tres capas del modelo jerárquico, dependiendo si los dispositivos están distribuidos en una sola planta o edificio, también, una red completa puede estar comprendida por varias sedes. Pero el diseño de las redes empresariales inicia desde redes pequeñas que llegan a usar un único conmutador en redes LAN hasta conexiones más complejas (Cisco, 2020).

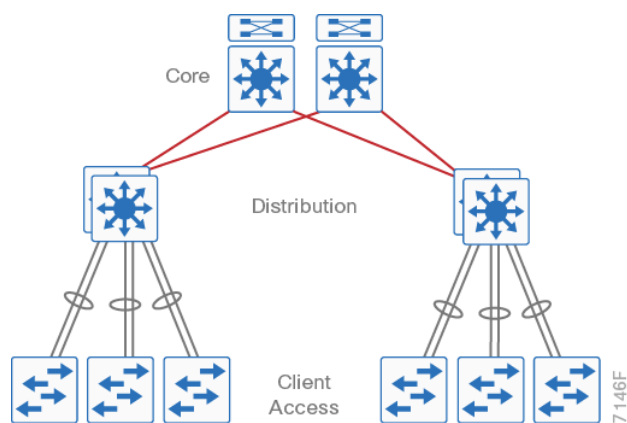
El modelo jerárquico que establece Cisco (2021) para arquitecturas LAN y WAN, requiere el establecimiento de tres capas, siendo estas: la capa de acceso, que se encarga de permitir la conexión directa a la red de los dispositivos finales; la capa de distribución, que se encarga de agregar las capas de acceso y proporciona conectividad a servicios basados en red, así como acceso hacia la WAN e Internet; y finalmente la capa del núcleo (core), encargada de proporcionar una comunicación rápida y confiable entre los distintos segmentos de la red, garantizando un alto rendimiento y estabilidad para entornos con elevado tráfico y múltiples conexiones.

Este modelo jerárquico destaca por su capacidad para reducir la complejidad operativa, facilitando la administración y permitiendo realizar cambios, actualizaciones o correcciones

con mínimo impacto sobre el resto de la infraestructura. Además, la modularidad que ofrece este diseño permite que la red se adapte fácilmente al crecimiento futuro, integrando nuevas conexiones o servicios sin afectar significativamente su funcionamiento global. La Figura 15 ilustra el modelo jerárquico propuesto por (Cisco, 2021), evidenciando gráficamente la interrelación estratégica y operativa entre estas capas.

Figura 15

Diseño jerárquico de redes cisco



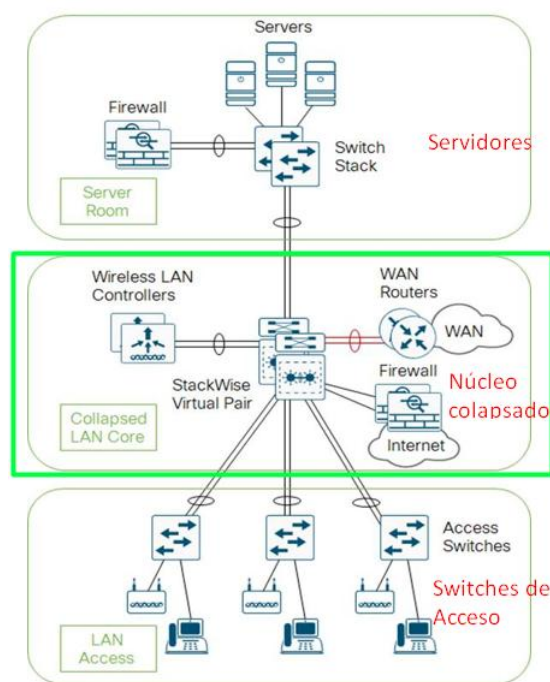
Nota. Obtenido de (Cisco, 2020)

Por otra parte, se menciona en Cisco (2020), un modelo jerárquico de dos niveles para el caso de empresas pequeñas o que no tiene un crecimiento significativo, estos modelos combinan la capa de núcleo con la capa de distribución en una sola, denominado núcleo colapsado, es decir, las funcionalidades que realizan en ambas capas pueden fusionarse en un único dispositivo como switches de multicapa, mientras que la capa de acceso se mantiene para la conexión final hacia los usuarios. Este enfoque permite reducir la complejidad operativa y disminuir los costos asociados al implementar menos equipos especializados, sin comprometer las prestaciones necesarias, de modo que la Figura 16 ilustra esta estructura, mostrando cómo el núcleo colapsado cumple simultáneamente con las funciones críticas de conectividad entre los switches de acceso y la interconexión hacia los servicios esenciales como servidores, routers WAN y acceso a Internet. De esta manera, la capa fusionada proporciona una

arquitectura compacta pero robusta, capaz de adaptarse a las exigencias operativas y escalables del entorno de redes empresariales.

Figura 16

Diseño jerárquico de dos niveles



Nota. Adaptado de (Cisco, 2020)

Por lo tanto, para empresas pequeñas o medianas con diseños multicapa, implementan funcionalidades que mejor se adapten a sus necesidades específicas, sin embargo, dentro de las recomendaciones de (Cisco, 2021), menciona para mejores prácticas, la implementación de protocolo de capa dos, STP (Spanning Tree Protocol), con la finalidad de tener enlaces redundantes y proporcionar alta disponibilidad de la red conmutada, sin generar bucles lógicos. También considera el uso de la tecnología EtherChannel para conexiones entre conmutadores de distribución y la capa de acceso, permitiendo la agrupación de varios enlaces físicos en un único enlace lógico, por tanto, permite aprovechar el uso de todos los enlaces disponibles y evitar su bloqueo al tener redundancia de enlaces, en consecuencia, se tiene varias ventajas como balanceo de carga, eficiencia en el enlace, mayor ancho de banda, y menor

almacenamiento tanto en direcciones MAC en puertos de entrada así como de las tablas de reenvío en los conmutadores, permitiendo mejorar los tiempos de convergencia en la red.

También se menciona en Cisco (2021), como buena práctica, la segmentación de la red a través de VLANs, para organizar a los usuarios o departamentos en grupos lógicos, entonces a cada grupo se asigna una VLAN única, permitiendo una estructura lógica más organizada en la red, esto puede ser gestionado de manera más eficiente a través del protocolo de Enlace Troncal de VLAN (VTP), ya que, permite configurar a los dispositivos en modo servidor y cliente para crear, eliminar o cambiar de nombre a las VLANs en toda la red, por lo tanto contribuye a ser una red escalable y segura, para evitar que usuarios accedan sin autorización. Además, las VLANs evitan una estructura de red plana, la cual, trabaja en un mismo segmento de red todos sus dispositivos, provocando congestión y bajo rendimiento debido a propagación de dominios de difusión, sin embargo, una red plana tiene bajos costos en administración y mantenimiento, pero no es una práctica recomendada por su baja eficiencia.

Por otro lado, para empresas con infraestructuras WAN, posee mayor cantidad de dispositivos y sucursales, que requieren mejorar su desempeño, garantizando la conectividad y seguridad ante accesos no autorizados, por tanto, en base a (Cisco, 2005), una solución considerada es implementar tecnologías como MPLS VPN, para permitir la interconexión eficiente entre las distintas sucursales a través, de una topología mallada, permitiendo escalabilidad sin afectar su arquitectura inicial, así como flexibilidad a incorporarse con nuevas tecnologías como SDN y automatización. También MPLS VPN, reduce la gestión de la red sin configuraciones complejas como túneles IPsec (Internet Protocol Security) donde agregan cifrados los paquetes que se envían, y puede provocar sobrecarga en el procesamiento de comunicación. Esto debido a que MPLS trabaja con asignación de etiquetas a los paquetes para el enrutamiento en lugar de métodos convencionales de enrutamientos IP, lo cual aporta a una transmisión de datos con baja latencia en tiempo real, además, al trabajar con VPN, el cual,

crea redes virtuales privadas, que proporciona independencia en gestionar su propio direccionamiento y rutas, usando segmentación lógica VRF (Virtual Routing and Forwarding), por tanto, permite garantizar la seguridad y privacidad, sin reducir el rendimiento en la red de extremo a extremo (Srinivash & Sivapunniyam, 2016).

Además, para redes empresariales tanto LAN o WAN, requieren de políticas para mejorar la función de transporte de aplicaciones empresariales como: tráfico de video, voz y datos, en una misma red, pero manteniendo una buena calidad del servicio, por tanto, las redes deben ofrecer servicios medibles para gestionar parámetros como ancho de banda, jitter, latencia y pérdida de paquetes (Cisco, 2005). Entonces, tecnologías como calidad de servicio (QoS), permite administrar el tráfico, mediante la priorización de aplicaciones que trabajan en tiempo real, ya que, son los servicios más críticos ante la pérdida de paquetes, de esta manera evita una competitividad entre los servicios por los recursos de la red, garantizando una buena experiencia al usuario final.

En este contexto, se describe a continuación las configuraciones que se considerará en cada escenario empresarial en base a recomendaciones presentadas. Es importante mencionar, que todas las infraestructuras a diseñar se presentan como simulaciones, las cuales estarán conformadas por un servidor Ansible alojado en un contenedor que se conecta a un switch de capa dos para simular una red de gestión ya que, actúa como medio de interconexión entre ansible con todos los equipos, por tanto, el servicio de automatización tiene una conexión directa a cada nodo secundario, porque en los siguientes escenarios de simulación los dispositivos no tienen ninguna configuración inicial, además cada enlace físico tiene el fin de ejemplificar las conexiones en entornos de prueba para el envío de un conjunto de configuraciones mediante el protocolo remoto SSH y comprender el funcionamiento de la herramienta IaC.

Esto permite una gestión centralizada, priorizando la configuración mediante los playbooks de manera consistente y uniforme hacia todos los equipos, además que minimiza la intervención manual con configuraciones tradicionales.

Sin embargo, hay que considerar que, en casos reales los dispositivos se encuentran físicamente distribuidos en diferentes ubicaciones y no existe un switch para interconectarlos como en la simulación. Por tanto, no se requiere de enlaces físicos directos para conectar Ansible con los dispositivos, ya que, en escenarios reales existe una infraestructura de red previamente interconectada por el proveedor de servicio como una WAN o VPN corporativa, que permite alcanzar los nodos secundarios mediante una red de gestión lógica para la administración de la red empresarial. Para ello, es necesario realizar manualmente una configuración por consola a cada equipo, que consiste en habilitar SSH y asignar a la interfaz del equipo una dirección IP para integrar a la red de gestión. Con esto ya es posible que Ansible alcance a todos los dispositivos para aplicar las configuraciones de forma centralizada.

3.1.1. Escenario 1: Diseño de red básico VLANs y STP

El primer diseño de red propuesto corresponde a una topología básica, donde existen un número limitado de dispositivos, siendo posible aplicarse a estructuras de red para empresas pequeñas como un centro de atención médica privada (Aguilar, 2022), por tanto, se puede adaptar a organizaciones del sector de salud, que operan en una sola localidad gestionando internamente, pero siendo posible una escalabilidad a futuro.

La topología mostrada en la Figura 17, se compone por un switch genérico de GNS3 de capa dos, el cual, se conecta al nodo de control, donde se encuentra la herramienta de automatización encargada de distribuir las configuraciones a sus respectivos dispositivos a través las interfaces del conmutador. Además, incluye dos conmutadores de capa dos, los cuales, pertenecen a la capa de distribución, estos están interconectados entre sí con enlaces redundantes y forman parte de una red LAN, adicional, se conecta un switch de capa tres hacia

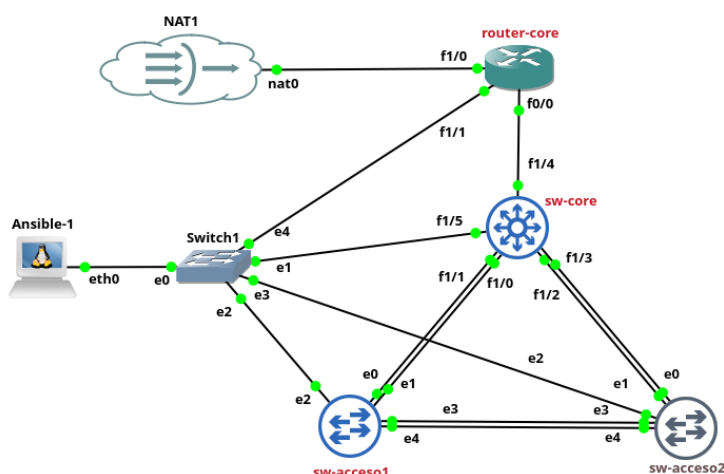
un enrutador que conforman a la capa core, para permitir la comunicación con otras redes y conexión a una nube NAT para tener acceso desde mi red interna a una red externa como internet.

El esquema se centra en configuraciones básicas como: la configuración de nombres de hosts y el protocolo remoto SSHv2, además la asignación de un usuario específico con accesos permitidos para la configuración y ejecución remota de comandos en los equipos.

Además, se va a incorporar VLANs para la segmentación lógica de la red a nivel de conmutación, esto permitirá la optimización del rendimiento mediante la agrupación de departamentos. Al ser redes lógicas independientes, se implementará enlaces troncales para permitir el envío de tráfico de diferentes VLANs por un mismo enlace físico. Por otra parte, al tener enlaces redundantes, se usará el protocolo STP (Spanning Tree Protocol) con la finalidad de evitar bucles y colisiones de paquetes, que puedan desestabilizar la red, también, para optimizar las configuraciones como es la creación de VLANs, se empleará el protocolo VTP tanto para el switch de core que actuará como servidor VTP y los switches de capa 2 como clientes VTP. Finalmente, para obtener una comunicación entre los grupos segmentados, se implementará inter-VLAN routing on a stick mediante la configuración de subinterfaces en el router para asegurar la gestión del tráfico de las diferentes VLANs.

Figura 17

Topología de red básico VLANs, STP



a) Dispositivos que integran el primer escenario

En base al diseño de la primera topología ilustrada en la Figura 17, se presenta a continuación en la Tabla 18, los dispositivos y servidor que conforman el escenario, detallando información sobre la RAM que consume cada equipo en la simulación para obtener un dimensionamiento de los recursos a utilizar y no sobreesaturar las capacidades de la máquina anfitriona, ya que, podría afectar la operación de la simulación, también se identifica el fabricante, el sistema operativo y su función que desempeña en la arquitectura de red básica con implementación de VLANs y STP.

Tabla 18
Dispositivos que integran el primer escenario

Host name	Dispositivo virtualizado	Fabricante	RAM	Sistema Operativo	Función	Cantidad
router-core	C7200 Series	Cisco	512 MB	CISCO IOS	Router de Core	1
sw-core	3000 Series	Cisco	512 MB	CISCO IOS	Switch de Core	1
sw-acceso	Vios_12 (For virtualization)	Cisco	256 MB	CISCO IOS(Experimental)	Switch de Distribución	2
Switch	Ethernet Generic Switch	GNS3	-	-	Interconexión de dispositivos	1

Ansible-1	Contenedor	GNS3	-	Ubuntu	Automatizació	1
	Network			Focal	n de	
	Automation			(20.04) LTS	configuracione	
					s	

b) Direccionamiento de red IPv4 del primer escenario

A continuación, se presenta el direccionamiento lógico para la segmentación de red mediante 9 VLANs, asociadas a áreas de trabajo operativas en una empresa pequeña del sector médico. Dicho esto, se toma de referencia la dirección IP de clase B, 10.10.X.0/16, donde el tercer octeto corresponde al ID de la VLAN para facilidad de organización, la cual, genera subredes con máscara de /24, lo que permite obtener un amplio rango de IP utilizables, considerando la inclusión de hosts ante un crecimiento futuro de la empresa. Por tanto, en la Tabla 19, se recopila las direcciones IP de cada VLAN, el nombre del departamento e identificador de la VLAN.

Tabla 19

Direccionamiento lógico mediante VLANs

Nombre VLAN	ID VLAN	Dirección de red
Datos	10	10.10.10.0/24
Atención clínica	20	10.10.20.0/24
Diagnostico	30	10.10.30.0/24
VoIP	40	10.10.40.0/24
Videovigilancia	50	10.10.50.0/24
TICS	60	10.10.60.0/24
Biométricos	70	10.10.70.0/24
Talento Humano	80	10.10.80.0/24
Farmacia	90	10.10.90.0/24

Por otra parte, una vez segmentada la red, se presenta en la Tabla 20, el direccionamiento IP de cada dispositivo que conforma el primer escenario, para ello, se considera la IP 10.20.0.0/26 para la administración general de la red, la cual, proporciona 62 IPs utilizables, permitiendo una escalabilidad en la red. En la presente fase, se asigna la

dirección IP con su respectiva máscara a cada enlace físico de los equipos, sin embargo, para el caso del R1, se incluye subinterfaces, ya que, serán usadas para el enrutamiento entre VLANs, mientras que en los Switches de capa tres y dos, la dirección IP no se asigna a la interfaz sino a la VLAN1 que es la nativa para la de gestión correspondiente del equipo a la VLAN1, por último, para las interfaces dirigidas a la nube NAT se aplica una dirección IP mediante DHCP.

Tabla 20
Direccionamiento IPv4 para el primer escenario

Dispositivo	Interfaces/ subinterfaces	Dirección IP	Máscara de subred
router- core	f1/0	DHCP	
	f1/1	10.0.0.12	255.255.255.192
	f0/0.10	10.10.10.1	255.255.255.0
	f0/0.20	10.10.20.1	255.255.255.0
	f0/0.30	10.10.30.1	255.255.255.0
	f0/0.40	10.10.40.1	255.255.255.0
	f0/0.50	10.10.50.1	255.255.255.0
	f0/0.60	10.10.60.1	255.255.255.0
	f0/0.70	10.10.70.1	255.255.255.0
	f0/0.80	10.10.80.1	255.255.255.0
	f0/0.90	10.10.90.1	255.255.255.0
sw-core	VLAN1	10.0.0.13	255.255.255.192
sw-acceso1	VLAN1	10.0.0.11	255.255.255.192
sw-acceso2	VLAN1	10.0.0.14	255.255.255.192
Ansible	eth0	10.0.0.1	255.255.255.192

3.1.2. Escenario 2: Diseño de red con MPLS-VPN

El diseño que se presenta en la Figura 18, corresponde a la topología del segundo escenario, conformada por una estructura de red MPLS-VPN, las cuales son: Customer Edge (CE) representados por los routers BCliente, los cuales son equipos de los clientes que se dirigen a las LAN finales; Provider Edge (PE) se refiere a los routers del proveedor y están conectados directamente a los equipos del cliente que son los enrutadores PBorder; Provider Router (P) corresponde al router del núcleo de la red en este caso identificados PNúcleo, por otro lado, la estructura de MPLS se compone de lo siguiente: Label Edge Router (LER) para

asignar o eliminar etiquetas en el Router de borde conocido como (PE); Label Switching Router (LSR) realiza el cambio de etiquetas en los routers de núcleo y Label Switched Path (LSP) se refiere al trayecto que toma el paquete mediante etiquetas asignadas dentro de la red MPLS.

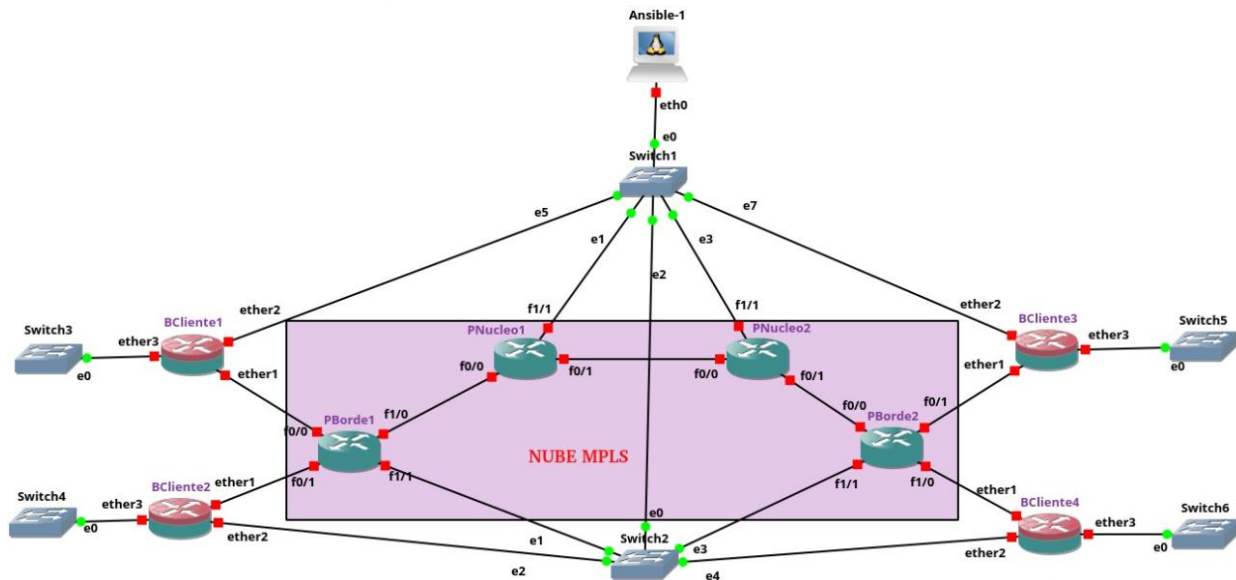
En base a lo mencionado, todos los enrutadores que conforma el escenario se conectan a un punto central, el cual corresponde a un conmutador genérico de GNS3 que está directamente conectado con la herramienta Ansible para el envío de sus respectivas configuraciones de cada dispositivo a través de su interfaz.

Dentro de las configuraciones que se realizarán a través de archivos de código como los playbooks, son las configuraciones básicas, habilitación del protocolo MPLS en los routers que conforman el núcleo de la red, para el intercambio de etiquetas a través de LDP (Label Distribution Protocol) que distribuye a cada paquete la etiqueta, por otro lado, los routers de borde (PE) asignan las etiquetas a los paquetes y almacenan varias tablas de enrutamiento en un mismo dispositivo mediante VRF (Virtual Routing and Forwarding), además, se debe configurar enrutamiento BGP en los routers de borde (PE) para gestionar la conectividad con otros (PE) a través de una VPN, también se considera el enrutamiento OSPF para el intercambio de rutas entre los routers de borde y routers del cliente (CE), es decir, se encarga de distribuir y publicar las rutas dentro de cada extremo de la red y el routers de borde recibe las rutas en cada VRF del cliente para ser compartida entre los distintos sitios del cliente mediante BGP, por lo tanto, al configurar dichos enrutamientos dinámicos, se implementa una interfaz lógica conocida como Loopback, que servirá como un identificador único para el equipo (router), además aporta confiabilidad, manteniendo las sesiones de enrutamiento dinámico activas. Con todo lo mencionado, se permitirá que la red pueda conectarse entre extremos a través de la VPN, con un enrutamiento segmentado que optimiza la administración del tráfico.

En concordancia con las características mencionadas, este tipo de estructuras de red son consideradas para empresas medianas del sector de financiero, como Cooperativas de Ahorro

y Crédito correspondientes a segmento 2, como se menciona en (Collahuazo, 2016), que cuentan con la matriz y sucursales situadas en diferentes ciudades, las cuales, pueden comunicarse de un extremo a otro, permitiendo su continuidad operativa.

Figura 18
Topología de red MPLS-VPN



a) Dispositivos que integran el segundo escenario

Una vez definido la estructura de red del segundo escenario como muestra la Figura 18, se procede a definir en la Tabla 21, las características de los dispositivos virtualizados y servidor de automatización que conforman el entorno de simulación, donde se detalla la serie del equipo con su respectivo fabricante ya sea MikroTik o Cisco, también se indica el consumo de memoria RAM para su rendimiento estable, así como su sistema operativo y la función que cumple dentro del escenario para configurar una red MPLS-VPN.

Tabla 21
Dispositivos que integran el segundo escenario

Host	Dispositivo	Fabricante	RAM	Sistema	Función	Cantidad
name	virtualizado			Operativo		
PNucleo	C7200	Cisco	512	CISCO IOS	LSR	4
PBorde	Series		MB		LER	
BCliente	CCR1036-	Mikrotik	134	RouterOS	Clientes	4
	8G-2S+		MB			
Switch	Ethernet	GNS3	-	-	Interconexión	2
	Generic				de dispositivos	
	Switch					
Ansible-	Contenedor	GNS3	-	Ubuntu	Automatización	1
1	Network			Focal	de	
	Automation			(20.04) LTS	configuraciones	

b) Direccionamiento de red IPv4 del segundo escenario

Correspondiente al segundo escenario, se realiza la asignación de direcciones IP y la máscara de subred a las interfaces de todos los dispositivos, como se indica en la Tabla 22. Para ello, se toma de referencia la IP privada de clase B 192.168.0.0/19, donde se agrupará en subredes según la topología mediante la técnica de VLSM, obteniendo máscaras de /30 para enlaces WAN y /26 para las redes LAN situadas en los extremos de la estructura de red. Por otra parte, debido a aspectos de escalabilidad y separación del tráfico, se considera una red diferente correspondiente a 172.16.0.0/25 destinada a la red gestión, con 126 IPs utilizables, para las interfaces que se conectan al servicio de automatización. Por último, las interfaces virtuales (Loopback) tienen una dirección fija y única de máscara /32, permitiendo la identificación del equipo.

Tabla 22
Direccionamiento IPv4 para el segundo escenario

Dispositivo	Interfaz	Dirección IP	Máscara de subred
PNucleo1	f0/0	192.168.0.1	255.255.255.252
	f0/1	192.168.0.5	255.255.255.252
	f1/1	172.16.0.5	255.255.255.128
	Lo0	4.4.4.4	255.255.255.255
PNucleo2	f0/0	192.168.0.6	255.255.255.252
	f0/1	192.168.0.9	255.255.255.252
	f1/1	172.16.0.6	255.255.255.128
	Lo0	5.5.5.5	255.255.255.255
PBorde1	f1/0	192.168.0.2	255.255.255.252
	f0/0	192.168.50.1	255.255.255.252
	f0/1	192.168.50.5	255.255.255.252
	f1/1	172.16.0.4	255.255.255.128
	Lo0	3.3.3.3	255.255.255.255
PBorde2	f0/0	192.168.0.10	255.255.255.252
	f0/1	192.168.50.9	255.255.255.252
	f1/0	192.168.50.13	255.255.255.252
	f1/1	172.16.0.7	255.255.255.128
	Lo0	6.6.6.6	255.255.255.255
BCliente1	ether1	192.168.50.2	255.255.255.252
	ether2	172.16.0.2	255.255.255.128
	ether3	192.168.100.1	255.255.255.192
	Lo0	1.1.1.1	255.255.255.255
BCliente2	ether1	192.168.50.6	255.255.255.252
	ether2	172.16.0.3	255.255.255.128
	ether3	192.168.100.65	255.255.255.192
	Lo0	2.2.2.2	255.255.255.255
BCliente3	ether1	192.168.50.10	255.255.255.252
	ether2	172.16.0.8	255.255.255.128
	ether3	192.168.100.129	255.255.255.192
	Lo0	7.7.7.7	255.255.255.255
BCliente4	ether1	192.168.50.14	255.255.255.252
	ether2	172.16.0.9	255.255.255.128
	ether3	192.168.100.193	255.255.255.192
	Lo0	8.8.8.8	255.255.255.255
Ansible	ether0	172.16.0.1	255.255.255.128

3.1.3. Escenario 3: Diseño de red MPLS y QoS

El siguiente diseño de red, corresponde al tercer escenario, donde se incorpora más dispositivos que las anteriores topologías, debido a esto, en la Figura 19, se muestra la conexión de cuatro enrutadores con una estructura mallada, proporcionando disponibilidad a los clientes, ya que, existe redundancia en los enlaces en caso de fallos.

Este tipo de topología está orientado a empresas que requieren de comunicación entre sucursales situadas en distintos lugares, las cuales, se componen por redes LAN, como lo son empresas medianas de Proveedores de Servicios de Internet (ISP) (Bustos, 2023). Estas organizaciones cuentan con sucursales que requieren de acceso hacia la LAN principal correspondiente a la matriz, la cual, también es un punto centralizado para el control de la red, por tanto, para la interconexión entre las diferentes LAN, cuentan con una red MPLS

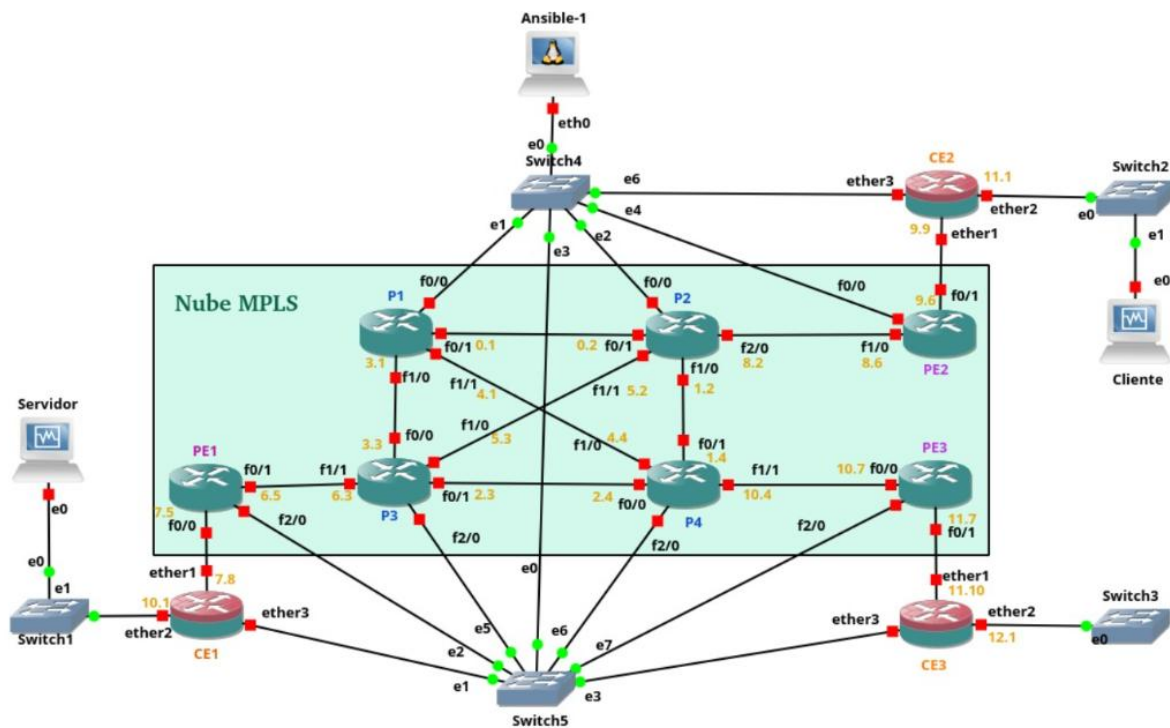
En este contexto, en el tercer diseño mostrado en la Figura 16, se considera las conexiones de los routers hacia un conmutador principal para distribuir las configuraciones por su respectiva interfaz a cada equipo. Además, en tres extremos se encuentran nodos finales que alojan al servidor que sitúa a servicios como: web, ftp y base de datos, mientras en otros dos nodos diferentes se encuentran los clientes que necesitan acceder a dichos servicios.

Este diseño mantiene una estructura de una red MPLS, considerando sus componentes principales que son los siguientes: LER para la asignación o eliminación de etiquetas en los routers de borde (PE) correspondiente a los equipos RBorde y con LSR (Label Switching Router) permite el cambio de etiquetas en los routers de núcleo de la red (P) representados por RProveedor, por otro lado, los enrutadores ubicados a los extremos, que son los RClientes están conectados a switches genéricos, permitiendo el acceso a los usuarios finales, es decir, el servidor y los clientes.

Dentro de la topología propuesta se va a considerar la automatización de configuraciones de tecnologías como MPLS para el envío de paquetes de una forma más

eficiente que la manera tradicional, en la cual, se basa en direcciones IP para cada salto, ya que usa etiquetas para que los paquetes sean encaminados a su destino. Además, se implementará el QoS basado en el modelo de servicios diferenciados (DiffServ), por ello, se clasifica el tráfico a través de ACLs , también se considera configuraciones de marcaje de paquetes, establecimiento la posibilidad de dropeo y el nivel de prioridad según la clases de tráfico, de este modo, en momentos de congestión de la red, los paquetes con mayor probabilidad de descarte se eliminan, y los que poseen una clase de prioridad alta va a realizar un envío asegurado de los paquetes.

Figura 19
Topología de red MPLS y QoS



a) Dispositivos que integran el tercer escenario

De manera similar a los anteriores diseños, en la Tabla 23 se detalla características de todos los dispositivos, hosts y servidores que integran la estructura del tercer escenario ilustrado en la Figura 19, los cuales son, la serie de cada equipo, el fabricante, sistema operativo para su ejecución, la función que cumple dentro de la red y el consumo de RAM requerida por

cada dispositivo con la finalidad de obtener los recursos totales que se usará en la simulación del escenario, considerando que no sobrepase las capacidades de la máquina anfitriona detallados en la sección 3.2.

Tabla 23

Dispositivos que integran el tercer escenario

Host	Dispositivo	Fabricante	RAM	Sistema	Función	Cantidad
name	virtualizado			Operativo		
P-- PE	C7200	Cisco	512	CISCO	LSR- LER	7
	Series		MB	IOS		
CE	CCR1036-	Mikrotik	134	RouterOS	QoS	3
	8G-2S+		MB			
Switch	Ethernet	GNS3	-	-	Interconexión	5
	Generic				de dispositivos	
	Switch					
Ansible-1	Contenedor	GNS3	-	Ubuntu	Automatización	1
	Network			Focal	de	
	Automation			(20.04)	configuraciones	
				LTS		
Servidor	Máquina	Canonical	1024	Ubuntu	Aloja servicios	1
	virtual	Ltd.	MB	Server	de red	
				20.04.6		
				LTS		
Cliente1	Máquina	The Debian	2048	Debian 12	Acceder a	2
Cliente2	virtual	Project	MB	Bookworm	servicios	

b) Direccionamiento de red IPv4 del tercer escenario

En esta parte se presenta el direccionamiento IP del tercer escenario, por tanto, para todos dispositivos se asigna la IP y máscara de subred correspondiente a cada interfaz como muestra la Tabla 24, donde se usa la técnica de VLSM para el agrupamiento en subredes, comenzando con la red principal de dirección IP privada de clase B 172.16.0.0/16, a partir de esta, se obtiene máscaras de /30 para enlaces WAN, además, para aislamiento de tráfico se considera otra red con IP 192.168.10.0/21, obteniendo máscara de /24 para las redes LAN. También, por motivos de organización y escalabilidad futura, se asigna una red dedicada a la gestión de la red con la dirección 10.10.10.0/25, donde es posible ocupar las 126 IPs utilizables, ya que, no es segmentada. Por último, al ser un escenario que usará protocolo dinámico se configura una interfaz virtual (Loopback) como parte de confiabilidad ya que, no depende de una interfaz física en caso de que falle y permite la identificación del equipo de red y continuidad de la operación, por tanto, se usa una dirección única de /32.

Tabla 24
Direccionamiento IPv4 para el tercer escenario

Dispositivo	Interfaz	Dirección IP	Máscara de subred
RProveedor1	f0/1	172.16.0.1	255.255.255.0
	f1/0	172.16.3.1	255.255.255.0
	f1/1	172.16.4.1	255.255.255.0
	f0/0	10.10.10.1	255.255.255.0
	Lo0	1.1.1.1	255.255.255.255
RProveedor2	f0/1	172.16.0.2	255.255.255.0
	f1/0	172.16.1.2	255.255.255.0
	f1/1	172.16.5.2	255.255.255.0
	f2/0	172.16.8.2	255.255.255.0
	f0/0	10.10.10.2	255.255.255.0
	Lo0	2.2.2.2	255.255.255.255
RProveedor3	f0/0	172.16.3.3	255.255.255.0
	f0/1	172.16.2.3	255.255.255.0
	f1/0	172.16.5.3	255.255.255.0
	f1/1	172.16.6.3	255.255.255.0
	f2/0	10.10.10.3	255.255.255.0
	Lo0	3.3.3.3	255.255.255.255
RProveedor4	f0/0	172.16.2.4	255.255.255.0
	f0/1	172.16.1.4	255.255.255.0

	f1/0	172.16.4.4	255.255.255.0
	f1/1	172.16.10.4	255.255.255.0
	f2/0	10.10.10.4	255.255.255.0
	Lo0	4.4.4.4	255.255.255.255
RBorde1	f0/0	172.16.7.5	255.255.255.0
	f0/1	172.16.6.5	255.255.255.0
	f2/0	10.10.10.5	255.255.255.0
	Lo0	5.5.5.5	255.255.255.255
RBorde2	f1/0	172.16.8.6	255.255.255.0
	f0/1	172.16.9.6	255.255.255.0
	f0/0	10.10.10.6	255.255.255.0
	Lo0	6.6.6.6	255.255.255.255
RBorde3	f0/0	172.16.10.7	255.255.255.0
	f0/1	172.16.11.7	255.255.255.0
	f2/0	10.10.10.7	255.255.255.0
	Lo0	7.7.7.7	255.255.255.255
RCliente1	ether1	172.16.7.8	255.255.255.0
	ether2	192.168.10.1	255.255.255.0
	ether3	10.10.10.8	255.255.255.0
	Lo0	8.8.8.8	255.255.255.255
RCliente2	ether1	172.16.9.9	255.255.255.0
	ether2	192.168.11.1	255.255.255.0
	ether3	10.10.10.9	255.255.255.0
	Lo0	9.9.9.9	255.255.255.255
RCliente3	ether1	172.16.11.10	255.255.255.0
	ether2	192.168.12.1	255.255.255.0
	ether3	10.10.10.11	255.255.255.0
	Lo0	10.10.10.10	255.255.255.255
Ansible-1	eth0	10.10.10.254	255.255.255.0
Servidor	eth0	192.168.10.2	255.255.255.0
Cliente1	eth0	192.168.11.2	255.255.255.0
Cliente2	eth0	192.168.12.2	255.255.255.0

3.4. Instalación del servicio de automatización Ansible

En la presente sección se describe el proceso de instalación de la herramienta Ansible para gestionar de forma centralizada la configuración de los dispositivos de red, mediante archivos de código en los tres escenarios planteados anteriormente, para su simulación en GNS3.

Para ello, se puede considerar dos alternativas para la instalación de Ansible:

- La primera opción es instalar Ansible en una máquina virtual pura, como se indica en el **Anexo 2**. Dicha alternativa, está orientada tanto para implementaciones en escenarios de simulación diferentes a GNS3 y para estructuras de redes reales.
- La segunda opción es usar la herramienta de Infraestructura como Código desde un contenedor, como el proporcionado por GNS3 llamado ‘Network Automation’, el cual, proporciona herramientas como Netmiko, Pyntc, NAPALM y Ansible que están específicamente diseñadas para tareas de automatización, además que el contenedor pueda ser integrado en el entorno de simulación.

A continuación, en la Tabla 25 se presenta una comparativa de las características para la operación de ambas opciones mencionadas.

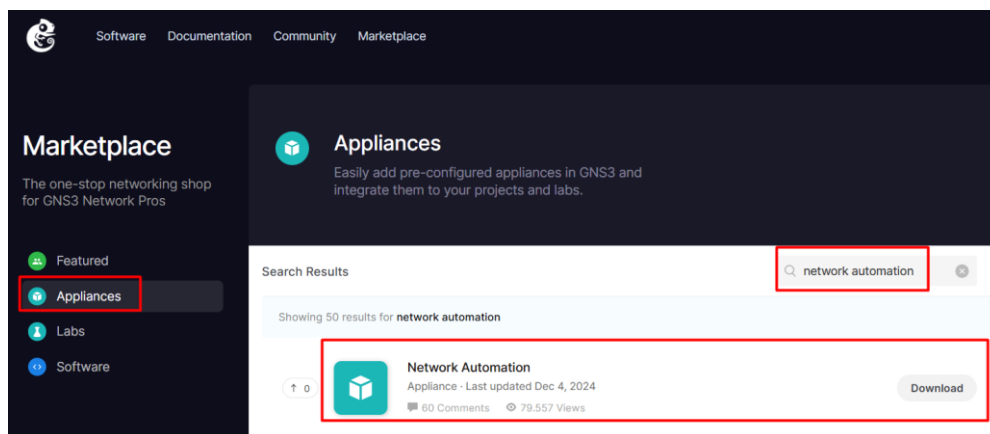
Tabla 25
Comparación entre máquina virtual y Docker

Característica	Máquina Virtual	Contenedor (Docker)
Sistema Operativo	Tiene un sistema operativo completo con todos los recursos.	Basada en una versión mínima del sistema operativo, solo con paquetes esenciales, servicios, y repositorios esenciales.
Aislamiento de operación	Ofrece completo aislamiento debido a que ejecuta su propio sistema operativo.	Tiene aislamiento en procesos, por tanto, no afecta al sistema de la máquina anfitriona.
Consumo de recursos	Tiene mayor consumo de recursos como CPU, almacenamiento y RAM	Bajo consumo de recursos, ya que comparte el kernel del sistema

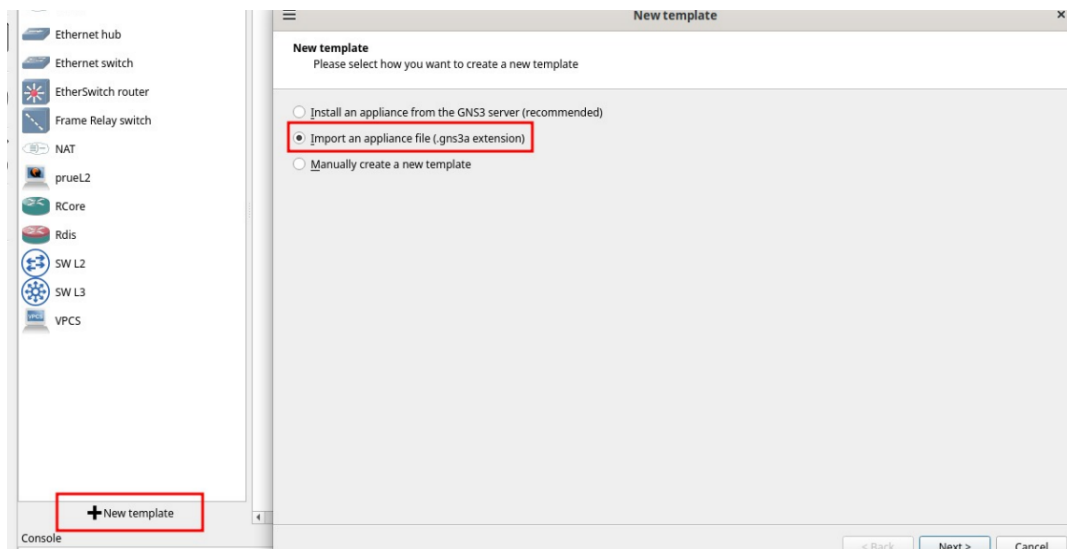
	para su virtualización, provocando mayor tiempo de arranque.	operativo anfitrión, por tanto, su inicialización es rápido.
Instalación	Se debe instalar y configurar el sistema operativo completo en un hipervisor.	Descarga e instalación rápida, además tiene integración directa con GNS3.
Portabilidad	Menos portátil por su gran tamaño, y tardía exportación de su sistema operativo completo.	Fácil de movilizar a otro entorno de trabajo, ya que, es liviano y no requiere configuraciones adicionales.
Enfoque de trabajo	Estructuras de red reales y entornos completos de virtualización.	Topologías de red simuladas

Una vez presentadas las características en la Tabla 25, se opta por utilizar el contenedor ‘Network Automation’ de GNS3, el cual, proporciona herramientas como Netmiko, Pyntc, NAPALM y Ansible, que es la que se usará para el proyecto. Dicho contenedor, se basa en un sistema operativo Ubuntu y ofrece varias ventajas como: el bajo consumo de recursos, integración directa con GNS3 y portabilidad en los entornos de simulación sin necesidad de configuraciones adicionales, ya que corre sobre el mismo kernel del sistema operativo de la máquina anfitriona y no pierde compatibilidad con los dispositivos virtualizados.

El proceso de instalación del contenedor comienza con su descarga desde la página principal de GNS3, en la sección Marketplace, donde se puede localizar mediante su nombre, ‘Network Automation’, como se indica la Figura 20.

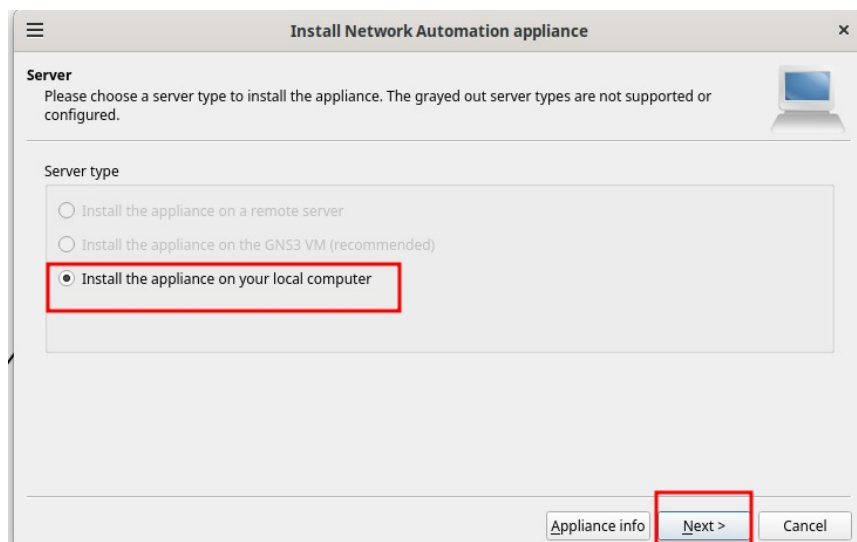
Figura 20*Descarga del contenedor de GNS3*

Una vez descargado el contenedor, la Figura 21 muestra cómo desde la interfaz gráfica principal de GNS3 en la sección "All Devices", se añade un nuevo componente. Para ello, es necesario importar el archivo .gns3a del contenedor desde la carpeta donde fue descargado.

Figura 21*Importar el archivo del contenedor*

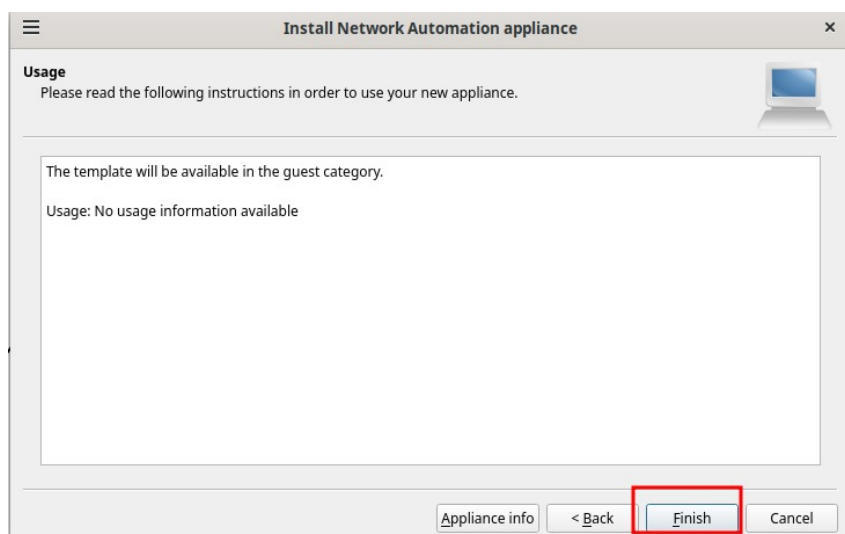
A continuación, en la Figura 22, se muestra la selección del tipo de instalación del contenedor. En este caso, se elige la opción de instalarlo en la máquina local, ya que es donde se ejecuta GNS3 y permite garantizar su disponibilidad en los entornos simulados.

Figura 22
Selección del tipo de instalación del contenedor



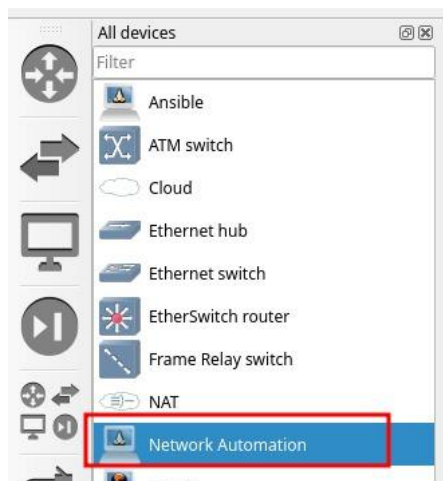
La última ventana, mostrada en la Figura 23, proporciona instrucciones de uso del contenedor. En ella se indica únicamente que el contenedor estará disponible en la categoría "Guest" (invitados), que agrupa servidores y contenedores para apoyan a las simulaciones de dispositivos de redes tradicionales como conmutadores y enrutadores, con esto se finaliza el proceso de instalación.

Figura 23
Instrucciones del contenedor



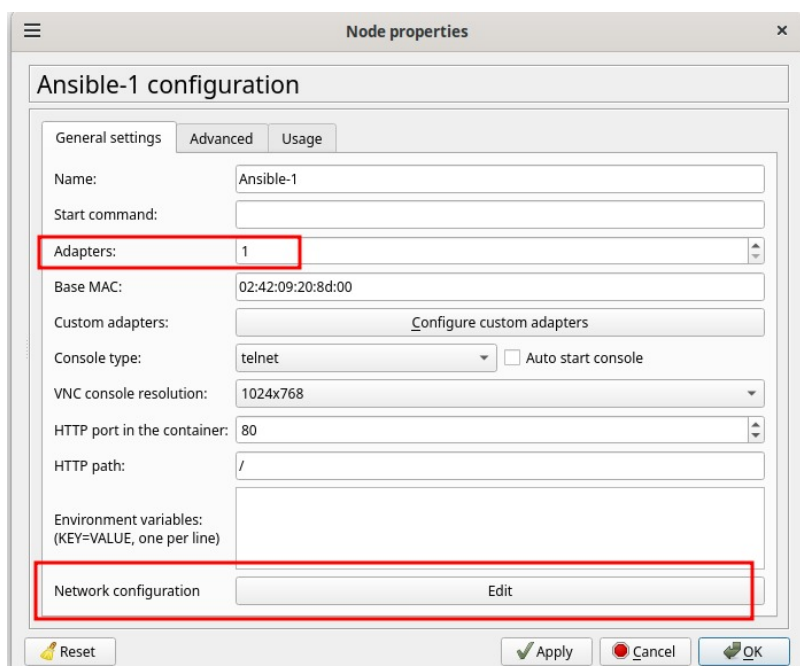
Al terminar correctamente el proceso de instalación, el contenedor se visualiza en la sección "All Devices", como se muestra en la Figura 24.

Figura 24
Contenedor de automatización disponible



A continuación, en contenedor que aloja Ansible debe ser ubicado en el espacio de simulación de GNS3 para su configuración mediante la ventana de las propiedades del nodo, como indica la Figura 25. Los parámetros que pueden ajustarse son: el nombre del contenedor para su identificación en la topología, el número de adaptadores de red y la asignación de direcciones IP a cada interfaz del contenedor a través de la opción 'Edit', según las condiciones del escenario de simulación.

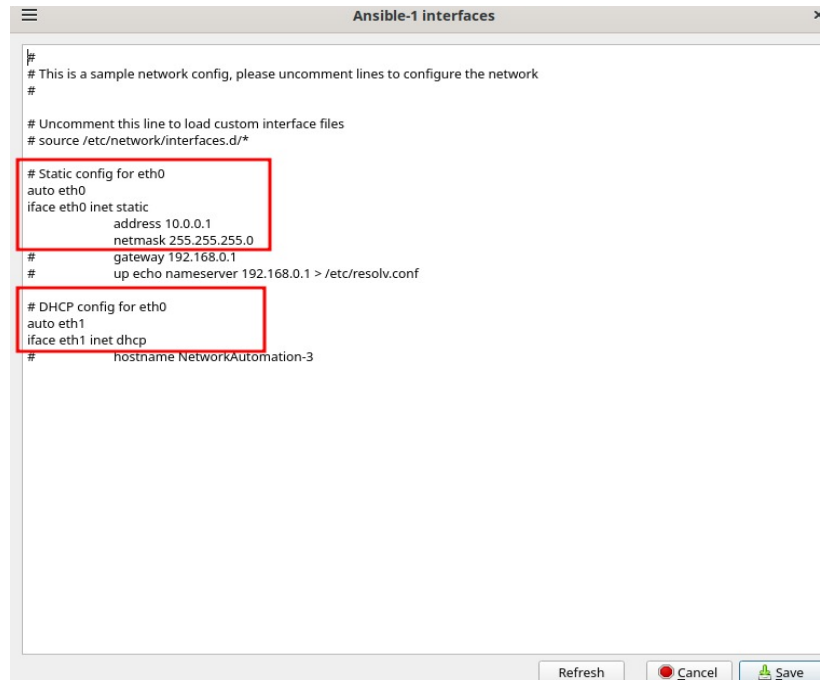
Figura 25
Configuración del contenedor de Ansible



En el apartado de la configuración de red se abre otra ventana, donde es posible asignar una dirección IP a cada interfaz del contenedor, ya sea estática o mediante DHCP, como se indica en la Figura 26.

Figura 26

Asignación de IP de la interfaz del contenedor



Posteriormente, se inicia el contenedor y se ejecuta el comando `apt update && apt upgrade`, como muestra la Figura 27, con el fin de actualizar e instalar las versiones actuales de los paquetes ya instalados que estén disponibles en el repositorio del sistema Ubuntu y de Ansible, ya que, son los configurados por defecto en el contenedor.

Figura 27

Actualización de paquetes en el contenedor

```

root@Ansible-1:~# apt update && apt upgrade
Hit:1 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:2 http://ppa.launchpad.net/ansible/ansible/ubuntu focal InRelease
Hit:3 http://archive.ubuntu.com/ubuntu focal InRelease
Get:4 https://apt.releases.hashicorp.com focal InRelease [12.9 kB]
Hit:5 http://archive.ubuntu.com/ubuntu focal-updates InRelease

```

Por último, se verifica tanto la instalación y versión de Ansible mediante el comando `ansible --versión`, como se muestra en la Figura 28, además detalla la ruta del archivo de configuración de ansible, versión de Python utilizada, ya que Ansible se basa en este lenguaje

para su funcionamiento, además, indica la instalación de librerías como libyaml y jinja2 para el desarrollo de archivos de configuración.

Figura 28

Verificación de la instalación de Ansible en contenedor

```
root@Ansible-1:~# ansible --version
ansible [core 2.12.10]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share
/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /root/.ansible/collections:/usr/share/ansible/co
llections
  executable location = /usr/bin/ansible
  python version = 3.8.10 (default, Mar 18 2025, 20:04:55) [GCC 9.4.0]
  jinja version = 2.10.1
  libyaml = True
root@Ansible-1:~#
```

3.5. Configuración de los escenarios de red

Una vez añadidos los dispositivos virtualizados y la herramienta Ansible en los escenarios de simulación de GNS3, se procede al despliegue de las topologías de red, donde se inicia con la preparación del entorno y posteriormente al desarrollo de playbooks para cada escenario, los cuales, están escritos con una estructura de datos YAML, permitiendo la gestión centralizada y eficiente de las configuraciones de los equipos de red.

3.5.1. Preparación del entorno

La preparación del entorno indica el procedimiento inicial que se debe aplicarse en los tres escenarios de red, para el funcionamiento correcto de la automatización de configuraciones de los dispositivos de red con Ansible. Este proceso incluye intervención manual de: la configuración de las interfaces de red del contenedor y la interfaz de gestión de los equipos, habilitación del protocolo SSH para el acceso remoto en todos los dispositivos, y la creación de la estructura del proyecto mediante carpetas que alojan a los archivos de configuración, con la finalidad de tener un entorno organizado y claro para la gestión de automatización.

Para empezar, se debe asignar de forma manual la dirección IP a las interfaces de red que usará el contenedor de Ansible, por tanto, en la consola se escribe el comando nano

/etc/network/interfaces como se muestra en la Figura 29, para acceder y editar el texto del archivo de configuración de red.

Figura 29

Ingreso al archivo de configuración de red de Ansible



```

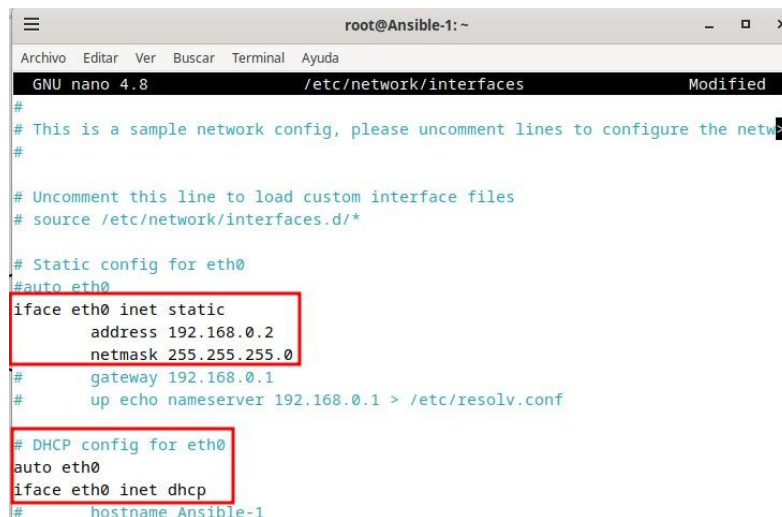
root@Ansible-1: ~
Archivo Editar Ver Buscar Terminal Ayuda
root@Ansible-1:~# nano /etc/network/interfaces
root@Ansible-1:~#

```

Una vez dentro del archivo como indica la Figura 30, se deben descomentar las líneas correspondientes a la configuración del direccionamiento de las interfaces que se usarán, ya sea con asignación estática o dinámica, según el escenario y el direccionamiento establecido en cada topología.

Figura 30

Configuración de direcciones IP en interfaces del contenedor



```

GNU nano 4.8 /etc/network/interfaces Modified
#
# This is a sample network config, please uncomment lines to configure the network
#
# Uncomment this line to load custom interface files
# source /etc/network/interfaces.d/*
# Static config for eth0
#auto eth0
iface eth0 inet static
    address 192.168.0.2
    netmask 255.255.255.0
# gateway 192.168.0.1
# up echo nameserver 192.168.0.1 > /etc/resolv.conf
# DHCP config for eth0
auto eth0
iface eth0 inet dhcp
# hostname Ansible-1

```

El siguiente paso, es la configuración de la interfaz de gestión de todos los dispositivos presentes en cada topología, en base al direccionamiento previamente definido. A través de estas interfaces se van a automatizar las configuraciones desde el nodo de control. Para el caso de equipos de fabricante cisco, se asigna a la interfaz una dirección IP correspondiente a la red de gestión y se habilita la interfaz como se muestra en la Figura 31.

Figura 31*Asignación de IP a interfaz de gestión*

```

P1#conf ter
Enter configuration commands, one per line. End with CNTL/Z.
P1(config)#int f0/0
P1(config-if)#ip add 10.10.10.1 255.255.255.0
P1(config-if)#no sh
P1(config-if)#
*Apr 28 15:24:18.063: %LINK-3-UPDOWN: Interface FastEthernet0/0

```

De la misma forma para todos los dispositivos presentes en cada escenario, se debe habilitar el protocolo de acceso remoto SSH, para permitir el envío de cada configuración descrita en el playbook hacia los equipos, por ende en la Tabla 26, muestra los comandos usados para habilitar el SSH en equipos Cisco.

Tabla 26*Comandos para habilitar SSH en dispositivos Cisco*

Comando	Función
conf t	Ingresa al modo de configuración global para realizar cambios permanentes en el equipo.
ip domain-name	Define el nombre de dominio, donde se puede usar cualquier nombre válido.
crypto key generate rsa	Genera claves RSA para encriptación asimétrica de la comunicación SSH.
2048	Se especifica el tamaño de las claves en bits.
username ansible privilege 15 secret ansible	Permite crear un usuario local con privilegios de acceso completo que corresponde al nivel 15, además se asigna una contraseña a dicho usuario.
ip ssh version 2	Especifica la versión de SSH a utilizar.
line vty 0 4	Configura el número de líneas virtuales de terminal (VTY) para permitir 5 conexiones simultaneas al dispositivo.
login local	Determina el acceso al equipo mediante autenticación local, usando las credenciales previamente definidas.
transport input ssh	Permite el acceso a los dispositivos mediante solo conexiones SSH.
no ip domain-name	Elimina el nombre de dominio creado.
do wr	Permite guardar las configuraciones realizadas.

Con los comandos descritos son ejecutados como muestra la Figura 32, para habilitar SSH, entre ellos incluye generar claves RSA, crear un usuario privilegiado y definir la

contraseña para su acceso, además es necesario guardar las configuraciones realizadas en caso de reinicios del dispositivo.

Figura 32

Habilitar SSH para dispositivos Cisco

```
P1#conf t
Enter configuration commands, one per line. End with CNTL/Z.
P1(config)#ip domain-name local.local
P1(config)#crypto key generate rsa
The name for the keys will be: P1.local.local
Choose the size of the key modulus in the range of 360 to 4096 for your
General Purpose Keys. Choosing a key modulus greater than 512 may take
a few minutes.

How many bits in the modulus [512]: 2048
% Generating 2048 bit RSA keys, keys will be non-exportable...
[OK] (elapsed time was 9 seconds)

P1(config)#username ansible privilege 15 secret ansible
P1(config)#ip ssh version 2
P1(config)#line vty 0 4
P1(config-line)#login local
P1(config-line)#transport input ssh
P1(config-line)#no ip domain-name
P1(config)#do wr
Warning: Attempting to overwrite an NVRAM configuration previously written
by a different version of the system image.
Overwrite the previous NVRAM configuration?[confirm]
*Apr 28 15:25:38.343: %SSH-5-ENABLED: SSH 1.99 has been enabled
[confirm]
Building configuration...
[OK]
```

A continuación, se muestra en la Figura 33, el mismo procedimiento inicial de asignar una dirección IP a la interfaz de gestión y habilitar el protocolo SSH para todos los routers Mikrotik, además verifica si el servicio de acceso remoto esta activado en la configuración. Por ende en la Tabla 27, se describe los comandos usados para habilitar SSH en dispositivos Mikrotik.

Tabla 27

Comandos para habilitar SSH en dispositivos Mikrotik

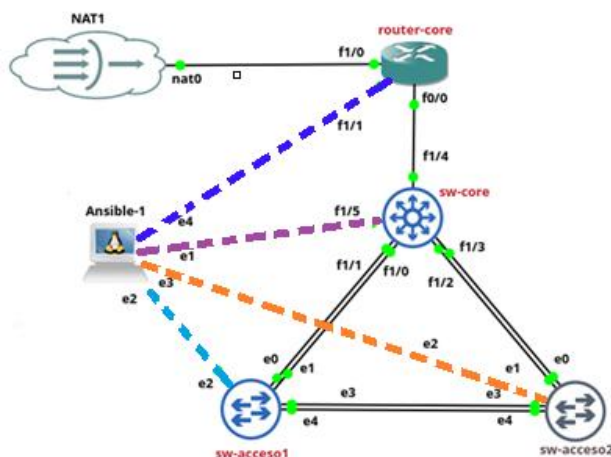
Comando	Función
/ip address add address=10.10.10.11/24 interface=ether3	Asigna una dirección IP a una determinada interfaz.
/user add name=ansible group=full password=ansible	Crea un usuario local con privilegios de acceso completo, además se define una contraseña a dicho usuario.
/ip service enable ssh	Permite habilitar SSH para las conexiones a los dispositivos.
/ip service print	Es opcional, ya que, sirve para visualizar los servicios habilitados.

```
/ip service set ssh          Activa SSH para garantizar su funcionamiento.
disabled=no
```

Figura 33*Habilitación de SSH en Mikrotik*

```
[admin@MikroTik] > /ip address add address=10.10.10.11/24 interface=ether3
[admin@MikroTik] >
[admin@MikroTik] > /ip service enable ssh
[admin@MikroTik] > /ip service print
Flags: X - DISABLED, I - INVALID
Columns: NAME, PORT, CERTIFICATE, VRF, MAX-SESSIONS
#  NAME  PORT  CERTIFICATE  VRF  MAX-SESSIONS
0  telnet  23      main         20
1  ftp    21      main         20
2  www    80      main         20
3  ssh    22      main         20
4 X www-ssl 443    none         main  20
5  api    8728    main         20
6  winbox 8291    main         20
7  api-ssl 8729    none         main  20
[admin@MikroTik] > /ip service set ssh disabled=no
[admin@MikroTik] >
```

Luego, es necesario comprobar el acceso remoto desde Ansible hacia cada uno de los dispositivos mediante conexión SSH. Este proceso se realiza manualmente y de forma individual, ya que la comunicación de Ansible con los equipos es directa e independiente, como se muestra en la Figura 34.

Figura 34*Conexión de Ansible con todos los dispositivos*

Para establecer la conexión SSH entre Ansible y cada dispositivo, se usa el usuario previamente creado junto a la IP o nombre del host como muestra la Figura 35. Cuando se establece la primera conexión, se pide confirmar la autenticidad para guardar la clave en la lista

de hosts conocidos, y permitir el establecimiento de conexión de manera automática en futuras conexiones.

Figura 35

Establecimiento de conexión remota desde Ansible

```
root@Ansible-1:~# ssh ansible@10.10.10.1
The authenticity of host '10.10.10.1 (10.10.10.1)' can't be established.
RSA key fingerprint is SHA256:Vzoo/nIWkItX+cMBYV89mUHQdsdwtZpz0d++k3AUa4.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.10.1' (RSA) to the list of known hosts.
Password:
P1#
P1#
```

Por otro lado, cuando la cantidad de dispositivos es grande, puede llevar más tiempo verificar la autenticidad uno a uno, por ello otra opción en los escenarios de red, es añadir en la sección o archivo de variables generales del proyecto la siguiente línea:

ansible_host_key_checking=False

Esta línea, evita prompts (pregunta interactiva al establecer la primera conexión), además no verifica la llave RSA del dispositivo, por tanto es posible la ejecución del playbook sin errores.

Sin embargo, en el archivo `~/.ssh/known_hosts` no se guardará las llaves de los dispositivos conocidos, sino solo se guarda en memoria del proceso la llave pública del equipo mientras la sesión este activa, ya que para la conexión desde Ansible hacia los dispositivos de red se usa *network_cli*, es decir, abre una conexión ssh para acceder a la interfaz de línea de comandos del equipo remoto usando la librería *paramiko* o *pylibssh*, las cuales no interactúan con el archivo `~/.ssh/known_hosts` como la librería OpenSSH.

Por ende, al usar la conexión *network_cli* se ve obligado a la verificación de huellas de manera manual, lo cual, es lo más recomendado, ya que al hacerlo manual se usa el cliente OpenSSH del sistema, por tanto, se guardarán las llaves RSA de los equipos.

Entonces, es importante verificar previamente la clave, para no perder seguridad cuando cualquier dispositivo desconocido quiera establecer conexión sin autorización.

Posteriormente, en la Figura 36, se indica la estructura que tendrá el proyecto de Ansible, mediante la creación de carpetas donde se situarán los archivos YAML, con el objetivo de tener un entorno organizado para facilitar la gestión de configuraciones. El nombre de las carpetas y archivos son personalizados, sin embargo se crearon las siguientes:

- **ansible:** es la carpeta principal del proyecto que contiene todos los archivos para la automatización.
- **inventory:** dicha carpeta aloja un archivo .ini o .yaml para definir los hosts, direcciones IPs para establecer conexión con los dispositivos de red.
 - **group_vars:** es una subcarpeta ubicada dentro de inventory, usada para definir variables comunes de cada grupo de dispositivos en un archivo .yaml, esto permite mejorar la organización en el inventario.
- **playbooks:** contiene los playbooks, que están escritos en formato de datos YAML, los cuales, describen las configuraciones a automatizar en los dispositivos definidos en el inventario.

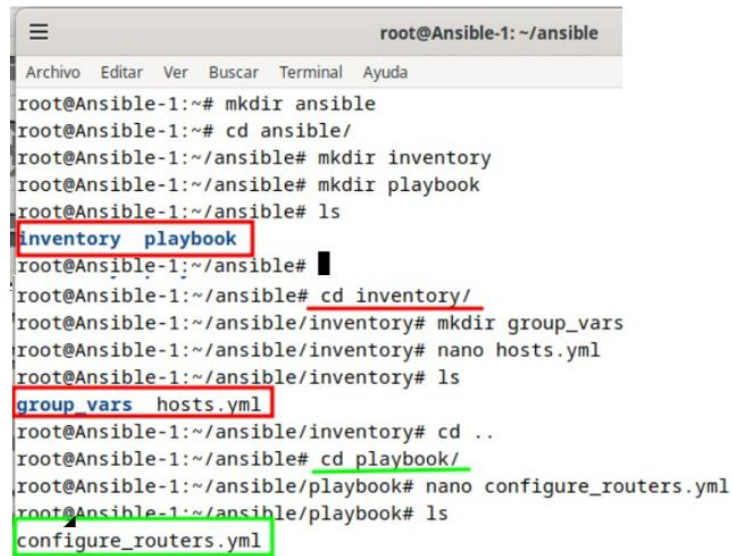
Figura 36

Estructura del proyecto Ansible

```
ansible/  
├─ inventory/  
│   ├── hosts.yaml  
│   └─ group_vars/  
│       └─ cisco_routers.yaml  
├─ playbook/  
│   └─ configure_routers.yaml
```

Por último, se indica en la Figura 37, la creación de las carpetas y de archivos .yaml en la consola de Ansible para mantener una estructura organizada.

Figura 37
Creación de carpetas y archivos en Ansible



```

root@Ansible-1: ~/ansible
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
root@Ansible-1:~# mkdir ansible
root@Ansible-1:~# cd ansible/
root@Ansible-1:~/ansible# mkdir inventory
root@Ansible-1:~/ansible# mkdir playbook
root@Ansible-1:~/ansible# ls
inventory  playbook
root@Ansible-1:~/ansible# 
root@Ansible-1:~/ansible# cd inventory/
root@Ansible-1:~/ansible/inventory# mkdir group_vars
root@Ansible-1:~/ansible/inventory# nano hosts.yml
root@Ansible-1:~/ansible/inventory# ls
group_vars  hosts.yml
root@Ansible-1:~/ansible/inventory# cd ..
root@Ansible-1:~/ansible# cd playbook/
root@Ansible-1:~/ansible/playbook# nano configure_routers.yml
root@Ansible-1:~/ansible/playbook# ls
configure_routers.yml

```

Al terminar con la preparación del entorno para los tres escenarios, se procede con el desarrollo de los playbooks, los cuales, contiene de manera estructurada las tareas que se ejecutarán, según las necesidades de configuración, además se detallará el contenido del archivo de hosts y del archivo de variables situados en el inventario, con el objetivo de entender el contenido de cada playbook a usar en la automatización de configuraciones.

La estructura de un playbook utiliza indentación para tener una estructura jerárquica que permita representar el contenido de cada parte, el cual, está compuesto por los siguientes elementos:

- **Play:** conocido como jugada, el cual, es el bloque principal que agrupa un conjunto de tasks (tareas), este siempre inicia con un (-) seguido un nombre descriptivo para el play y se diferencia por ser el nivel más alto de indentación.
- **hosts:** se sitúa seguido del nombre del play, el cual indica a que dispositivo o grupo de dispositivos definidos en el inventario se aplicará las configuraciones.
- **Tasks:** también se conoce como tareas, y son la lista de acciones que se aplica a los hosts gestionados, las cuales se encuentran dentro del bloque del Play, y a cada tarea se le asigna un nombre descriptivo para su identificación.

- **Variables:** son valores que tienen en común un grupo de dispositivos, permitiendo reutilizar en las tareas, las variables pueden definirse directamente en el playbook o en una carpeta como `group_vars`.
- **Módulos:** es un componente o unidad funcional que permite ejecutar acciones específicas sobre los dispositivos a gestionar como realizar las configuraciones automatizadas, por lo general se sitúan debajo del nombre de cada tarea.

Por otra parte, las instrucciones en común que se usan para las configuraciones generales situadas en los playbooks de cada escenario son las mostradas en la Tabla 28.

Tabla 28

Instrucciones en común para el desarrollo de los playbooks

Instrucción	Descripción
- name:	Permite definir el nombre del play y de la tarea, solo para identificar la acción en el playbook.
hosts:	Indica a que grupo de dispositivos definidos en el inventario, se ejecutarán las tareas.
gather_facts: no	Indica que hechos (información) del sistema no serán recopilados antes de la ejecución de tareas.
connection: network_cli	Define el tipo de conexión hacia los dispositivos, en este caso mediante la interfaz de línea de comandos (CLI)
ios_config:	Módulo para dispositivos cisco, que respeta la jerarquía para escribir los comandos.

3.5.2. Desarrollo de playbook para escenario 1

Una vez completado los pasos de la preparación del entorno que son la configuración de la interfaz de gestión, habilitación de SSH en cada dispositivo y siguiendo con la estructura del proyecto de ansible definida; como primer paso se edita el archivo local del sistema del contenedor, `/etc/hosts` para asociar la dirección IP de la interfaz de gestión de cada dispositivo con sus nombres de host, como se muestra en la Figura 38, con la finalidad que ansible resuelva localmente el nombre del host usados en el inventario, ya que, para el escenario 1, la conexión hacia los dispositivos será mediante nombre de host en lugar de la IP.

Figura 38*Resolución de nombres de hosts en escenario 1*

```

root@Ansible-1: ~
GNU nano 4.8 /etc/hosts
127.0.1.1    Ansible-1
127.0.0.1    localhost
::1         localhost ip6-localhost ip6-loopback
fe00::0     ip6-localnet
ff00::0     ip6-mcastprefix
ff02::1     ip6-allnodes
ff02::2     ip6-allrouters
10.0.0.11    sw-acceso1
10.0.0.12    router-core
10.0.0.13    sw-core
10.0.0.14    sw-acceso2

```

A continuación se edita el archivo `hosts.ini`, que de acuerdo con la Figura 35, su ubicación forma parte de `inventory` (inventario), con la finalidad de definir los dispositivos de red a los que se conectará Ansible para automatizar las configuraciones, en este, las conexiones se realizarán mediante nombre de host.

Los dispositivos definidos se los agrupa en base a su función que cumplirán, que son *routers*, *switches* y *switches_acceso*, como se muestra en la Figura 39. Por tanto, en cada línea se representa un dispositivo con sus parámetros para el establecimiento de conexión, entre ellos se define:

- nombre del dispositivo: es el nombre identificador que se establece al dispositivo de manera personalizada, el cual, usará ansible en el playbooks.
- `ansible_host`: la IP o nombre del host para establecer conexión con el equipo
- `ansible_user`: el nombre de usuario creado previamente en el dispositivo.
- `ansible_password`: contraseña de acceso específico del usuario creado.
- `ansible_network_os`: el sistema operativo del dispositivo

- `ansible_connection`: el tipo de conexión que usará Ansible para acceder al dispositivo, el cual es a través de la interfaz de línea de comandos (CLI).

Figura 39*Definición de dispositivos para escenario 1*

```

GNU nano 4.8 hosts
[router]
router-core ansible_host=router-core

[switches]
sw-core ansible_host=sw-core

[switches_acceso]
sw-acceso1 ansible_host=sw-acceso1
sw-acceso2 ansible_host=sw-acceso2

[all:vars]
ansible_connection=network_cli
ansible_network_os=ios
ansible_user=ansible
ansible_password=ansible

```

Posteriormente, se crea archivos `.yaml` dentro de la carpeta `group_vars` para definir las variables de cada grupo de dispositivos como indica la Figura 40, los cuales son: `routers.yaml`, `switches.yaml` y `switches_acceso.yaml`, para una mejor organización.

Figura 40*Archivos para definición de variables en escenario 1*

```

root@Ansible-1:~/ansible# ls
group_vars hosts playbook.yaml
root@Ansible-1:~/ansible# cd group_vars/
root@Ansible-1:~/ansible/group_vars# ls
routers.yaml switches.yaml switches_acceso.yaml
root@Ansible-1:~/ansible/group_vars# nano routers.yaml

```

Dentro del archivo correspondiente al grupo de routers, se define una variable llamada `router_subinterfaces`, donde contiene una lista de comandos a ejecutar en el grupo definido, como indica en la Figura 41. Las líneas de configuración son: el levantamiento de la interfaz física; creación de subinterfaces para cada VLAN, usando la encapsulación `dot1Q` y asignación de IP correspondiente a cada red VLAN; además a cada subinterfaz se declara parte de la red interna permitiendo que tenga salida a internet, a través del proceso NAT.

Figura 41*Definición de variables para el grupo routers*

```

GNU nano 4.8 routers.yml
router_subinterfaces:
- interface f0/0
- no shutdown
- interface f0/0.10
- encapsulation dot1q 10
- ip address 10.10.10.1 255.255.255.0
- ip nat inside
- interface f0/0.20
- encapsulation dot1q 20
- ip address 10.10.20.1 255.255.255.0
- ip nat inside
- interface f0/0.30
- encapsulation dot1q 30
- ip address 10.10.30.1 255.255.255.0
- ip nat inside
- interface f0/0.40
- encapsulation dot1q 40
- ip address 10.10.40.1 255.255.255.0
- ip nat inside
- interface f0/0.50
- encapsulation dot1q 50
- ip address 10.10.50.1 255.255.255.0
- ip nat inside
- interface f0/0.60
- encapsulation dot1q 60
- ip address 10.10.60.1 255.255.255.0
- ip nat inside
- interface f0/0.70
- encapsulation dot1q 70
- ip address 10.10.70.1 255.255.255.0
- ip nat inside
- interface f0/0.80
- encapsulation dot1q 80
- ip address 10.10.80.1 255.255.255.0
- ip nat inside
- interface f0/0.90
- encapsulation dot1q 90
- ip address 10.10.90.1 255.255.255.0
- ip nat inside

```

A continuación, la Figura 42, muestra el contenido del archivo switches.yml, donde se define la variable `trunk_interfaces_cr`, la cual contiene las interfaces físicas de switch core que serán configuradas como troncales para permitir el transporte de múltiples VLANs.

Figura 42*Definición de variables para el grupo switches*

```

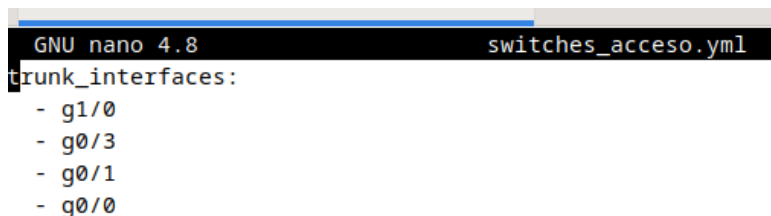
root@Ansible-1: ~/ansible/group_vars
GNU nano 4.8 switches.yml
trunk_interfaces_cr:
- f1/0
- f1/1
- f1/2
- f1/3
- f1/4

```

De la misma manera, para el grupo `switches_acceso` se define la variable `trunk_interfaces` para indicar las interfaces físicas de los switches de acceso que serán configuradas troncales, tal como se muestra la Figura 43.

Figura 43

Definición de variables para el grupo `switches_acceso`



```
GNU nano 4.8 switches_acceso.yml
trunk_interfaces:
- g1/0
- g0/3
- g0/1
- g0/0
```

Paso siguiente, se comienza con la configuración general de la topología mediante la definición de diferentes jugadas en el archivo `playbook.yml`, las cuales contienen su propio conjunto de tareas para aplicarlas a los tres grupos de dispositivos definidos en la Figura 39, de este modo en la Tabla 29, se presenta las instrucciones usadas dentro de las jugadas para la configuración del primer escenario.

Tabla 29

Instrucciones del `playbook` para configuración del escenario 1

Instrucción	Descripción
<code>ansible.netcommon.cli_command:</code>	Módulo de la colección <code>ansible.netcommon</code> para ejecutar directamente los comandos, ya que no sigue una jerarquía estructurada.
<code>lines:</code>	Define la lista de comandos, los cuales serán ejecutados en el equipo.
<code>loop:</code>	Permite ejecutar varias veces la misma tarea para cada valor de una lista.
<code>command: </code>	Indica la lista de comandos que serán escritos como un solo bloque.
<code>zip</code>	Empareja datos de dos listas diferentes.
<code>failed_when: false</code>	Evita detener la ejecución del <code>playbooks</code> en caso de errores o advertencias no críticas.

La primera jugada (PLAY 1) se señala en la Figura 44, correspondiente al bloque de configuración del router, donde se evidencia su jerarquía interna y permite diferenciarse de otras jugadas (PLAY) posteriores. Por tanto, en este primer PLAY se define: el host a donde se aplicará las configuraciones que es el grupo de ‘routers’; y el tipo de conexión

correspondiente a CLI para permitir automatizar el enrutamiento inter-VLAN y la conexión a redes externas como la salida a internet. A continuación, se detalla cada tarea que conforma el PLAY 1:

- **Configurar interfaz de salida para NAT:** a la interfaz f1/0 se le asigna una IP por DHCP, además es marcada como 'ip nat outside' para que sea designada como interfaz de salida del tráfico interno para conexión a redes externas, ya que va a traducir la ip privada (origen) a una ip pública.
- **Configurar NAT sobrecargado:** se define una ACL estándar para permitir todo el tráfico de cualquier IP origen (red interna) pueda ser traducida a una única IP pública para que todos los dispositivos puedan acceder a internet, a través de la misma interfaz de salida f1/0.
- **Configuración de interfaz y subinterfaces:** se llama a la variable 'router_subinterfaces' definida en el archivo 'routers.yml' para la configuración de interfaces.
- **Configuración de NAT inside:** a todas las subinterfaces definidas en la variable 'router_subinterfaces' se las marca como ip nat inside para indicar que pertenece a la red interna, esto se realiza mediante un bucle for que inicia con número actual del bucle que es 10 hasta 90 en saltos de 10.

Figura 44*Configuración del router Cisco - Escenario 1*

```

GNU nano 4.8                                playbook.yml
- name: Configurar router Cisco
  hosts: routers
  gather_facts: no
  connection: network_cli

  tasks:
    - name: Configurar interfaz de salida y NAT
      ios_config:
        lines:
          - interface f1/0
          - ip address dhcp
          - ip nat outside
          - no shutdown

    - name: Configurar NAT sobrecargado
      ios_config:
        lines:
          - access-list 1 permit any
          - ip nat inside source list 1 interface fastEthernet1/0 overload

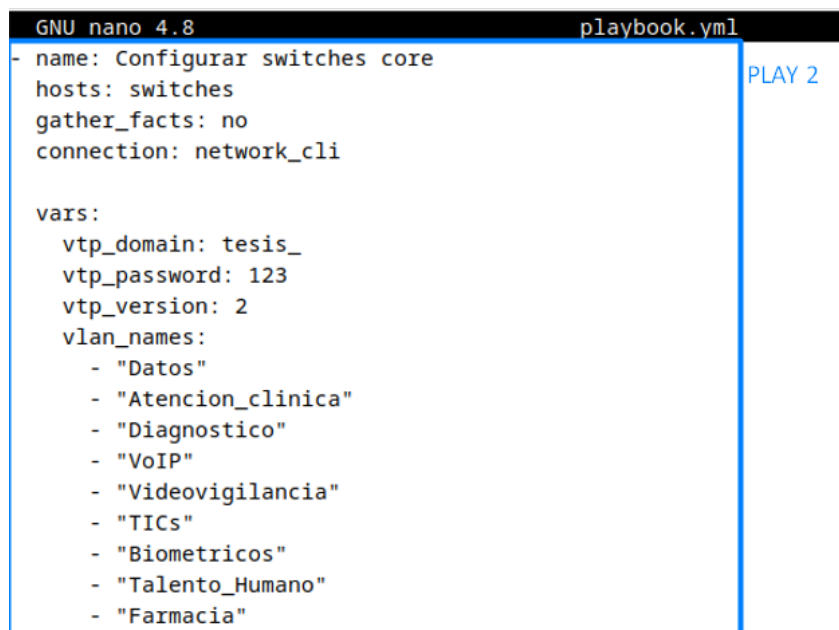
    - name: Configurar interfaz principal y subinterfaces
      ios_config:
        lines: "{{ router_subinterfaces }}"

    - name: Configurar NAT inside
      ansible.netcommon.cli_command:
        command: |
          configure terminal
          {% for id in range(10, 91, 10) %}
          interface f0/0.{{ id }}
          ip nat inside
          exit
          {% endfor %}
        end

- name: Configurar switches core
  hosts: switches
  gather_facts: no
  connection: network_cli

```

Seguidamente, se presenta el segundo play, que tiene la finalidad de configurar el switch de core, indicado en la Figura 45, donde las configuraciones están destinadas al grupo de ‘switches’, además se define localmente las variables para usarlas en las tareas del mismo bloque que son, el dominio VTP y los nombres de cada VLAN desde la 10 a la 90. Ya que, en este bloque, se definirá al switch de core en modo VTP servidor y se realizará la creación de las VLANs.

Figura 45*Configuración del switch de core - Escenario 1*

```
GNU nano 4.8      playbook.yml
- name: Configurar switches core
  hosts: switches
  gather_facts: no
  connection: network_cli

  vars:
    vtp_domain: tesis_
    vtp_password: 123
    vtp_version: 2
    vlan_names:
      - "Datos"
      - "Atencion_clinica"
      - "Diagnostico"
      - "VoIP"
      - "Videovigilancia"
      - "TICs"
      - "Biometricos"
      - "Talento_Humano"
      - "Farmacia"

PLAY 2
```

A continuación, en la Figura 46, indica las siguientes tareas para configurar el switch de core, que corresponden al PLAY 2:

- **Configurar interfaces troncales:** cada interfaz definida en la variable `trunk_interfaces_cr` se establecerán como una interfaz trunk con encapsulación 802.1Q, para transportar el tráfico de varias VLANs usando etiquetas.
- **Establecer root bridge STP:** se define la prioridad 0 (prioridad baja) para que el switch de core sea el root bridge en todas las VLANs, y así poder calcular los caminos más cortos desde este punto de referencia con la finalidad de evitar bucles en la red.

Para ello se usa la instrucción 'loop' para recorrer el ID de todas las VLANs generadas de la 10 a la 90 en saltos de 10.

Figura 46*Tareas de configuración en switch de core en escenario 1*

```

GNU nano 4.8                                playbook.yml
tasks:
  - name: Configurar interfaces troncales
    ios_config:
      parents: "interface {{ item }}"
      lines:
        - switchport trunk encapsulation dot1q
        - switchport mode trunk
        - no shutdown
      loop: "{{ trunk_interfaces_cr }}"

  - name: Establecer prioridad 0 como root bridge para VLANs 10-90
    ios_config:
      lines:
        - "spanning-tree vlan {{ item }} priority 0"
      loop: "{{ range(10, 91, 10) | list }}"

```

Además, se definen secuencialmente otras tareas en el mismo bloque del PLAY 2 correspondiente al switch de core, las cuales tienen comandos que se ejecutan en un modo especial de configuración llamado ‘vlan database’ que se accede desde el modo de operación EXEC privilegiado (Switch#), con el fin de configurar parámetros relacionados con VLANs y VTP, ya que se está trabajando en series de switches antiguos donde ciertos comandos no están disponibles dentro del modo de configuración global (configure terminal).

Las tareas para configuración de VLANs y VTP, se muestran en la Figura 47 y son detalladas a continuación:

- **Creación de VLANs:** se crean VLANs mediante un bucle for que genera valores de un rango del 10 a 90 en saltos de 10 para la variable ‘i’.
- **Asignar nombres a las VLANs:** los nombres de VLANs definidos en la variable ‘vlan_names’ son tomadas para asociarlas a través del filtro ‘zip’ con el número de la VLAN correspondiente de la 10 a la 90.
- **Configuración de servidor VTP:** se define al switch de core como servidor VTP para distribuir y gestionar las VLANs, además se define el dominio, contraseña y versión del VTP para permitir la administración de los switches que pertenecen al mismo dominio VTP.

Figura 47*Tareas de configuración en switch de core en escenario 1*

```

GNU nano 4.8                                playbook.yml
- name: Crear VLANs 10-90 usando vlan database
  ansible.netcommon.cli_command:
    command: |
      vlan database
      {% for i in range(10, 91, 10) %}
      vlan {{ i }}
      {% endfor %}
      exit

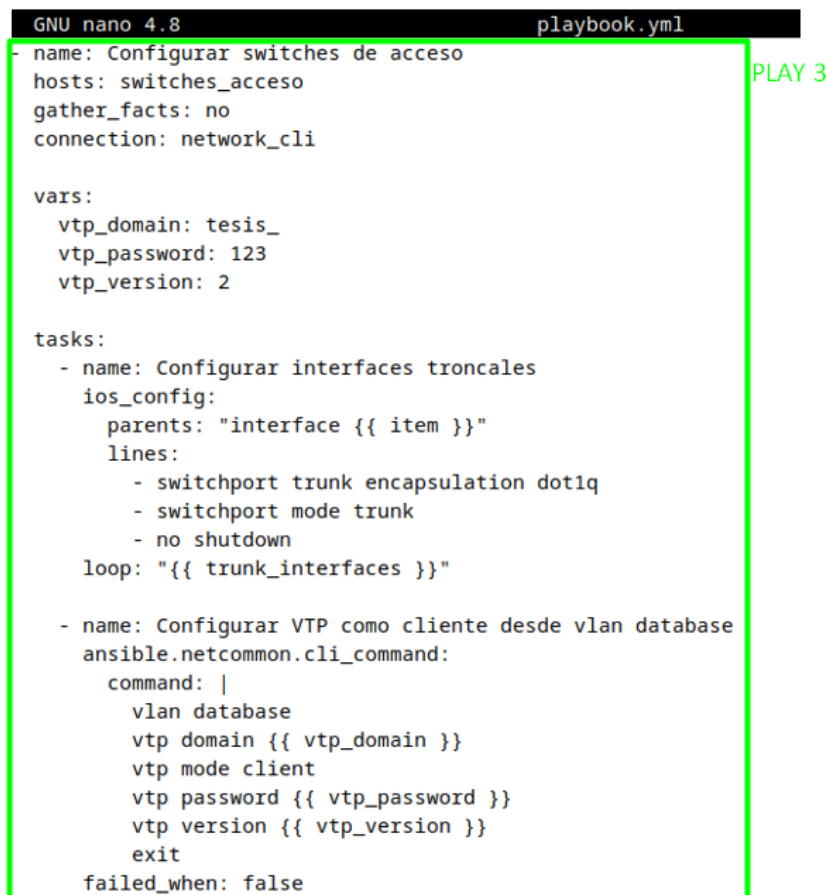
- name: Asignar nombres a VLANs
  ansible.netcommon.cli_command:
    command: |
      vlan database
      {% for id, name in range(10, 91, 10) | zip(vlan_names) %}
      vlan {{ id }} name {{ name }}
      {% endfor %}
      exit

- name: Configurar VTP usando modo EXEC (no config)
  ansible.netcommon.cli_command:
    command: |
      vlan database
      vtp domain {{ vtp_domain }}
      vtp mode server
      vtp password {{ vtp_password }}
      vtp version {{ vtp_version }}
  failed_when: false

```

La última jugada (PLAY) se muestra en la Figura 48, corresponde a la configuración de los switches de acceso, a través de una conexión a la interfaz de línea de comandos para configurar a dichos conmutadores como modo VTP cliente para que reciban las VLANs creadas desde el switch de core. Las tareas que forman parte de este play son:

- **Configuración de interfaces troncales:** todas las interfaces listadas en la variable `trunk_interfaces` en el archivo `switches_acceso.yml`, van a ser recorridas con la instrucción ‘loop’ para configurarles como interfaces troncales y con encapsulación 802.1Q para que el tráfico de diferentes VLANs sea etiquetado.
- **Configuración VTP como cliente:** se configura los switches de acceso como VTP cliente, para ser posible recibir las VLANs creadas del switch de core, ya que pertenecen al mismo dominio VTP.

Figura 48*Configuración de switches de acceso - Escenario 1*


```

GNU nano 4.8                                playbook.yml
- name: Configurar switches de acceso
  hosts: switches_acceso
  gather_facts: no
  connection: network_cli

  vars:
    vtp_domain: tesis_
    vtp_password: 123
    vtp_version: 2

  tasks:
    - name: Configurar interfaces troncales
      ios_config:
        parents: "interface {{ item }}"
        lines:
          - switchport trunk encapsulation dot1q
          - switchport mode trunk
          - no shutdown
        loop: "{{ trunk_interfaces }}"

    - name: Configurar VTP como cliente desde vlan database
      ansible.netcommon.cli_command:
        command: |
          vlan database
          vtp domain {{ vtp_domain }}
          vtp mode client
          vtp password {{ vtp_password }}
          vtp version {{ vtp_version }}
          exit
        failed_when: false
  
```

Por otro lado, para contrastar la configuración automatizada mediante un playbook y la configuración tradicional, en la Figura 49, se muestra los comandos convencionales que son ingresados manualmente al CLI para asignarlas direcciones IP a las interfaces de VLAN ya sea en el sw-acceso1 o sw-acceso2, según la VLAN que vaya a operar cada switch, con la finalidad de verificar la comunicación entre VLANs.

Como ejemplo en la Figura 49, se muestra la asignación de una IP a la interfaz de la VLAN 60 en el sw-acceso1, lo cual, indica que dicho switch va a operar dentro de la VLAN60. Además, se configura una ruta por defecto en el switch de acceso 1, donde indica que todo tráfico sin una ruta definida se envía a la puerta de enlace de la VLAN60 configurada en el router, la cual se encarga de enviará dicho tráfico a su destino, que es una red exterior, es decir internet.

Figura 49*Asignar IP a interfaz de VLAN60 en Switch de acceso 1*

```

sw-acceso1#conf ter
Enter configuration commands, one per line. End with CNTL/Z.
sw-acceso1(config)#int vlan60
sw-acceso1(config-if)#
*May 19 21:20:53.844: %LINEPROTO-5-UPDOWN: Line protocol on Interface Vlan60, changed state to down
sw-acceso1(config-if)#ip add 10.10.60.2 255.255.255.0
sw-acceso1(config-if)#no sh
sw-acceso1(config-if)#exit
sw-acceso1(config)#
*May 19 21:21:11.586: %LINK-3-UPDOWN: Interface Vlan60, changed state to up
*May 19 21:21:12.586: %LINEPROTO-5-UPDOWN: Line protocol on Interface Vlan60, changed state to up
sw-acceso1(config)#ip route 0.0.0.0 0.0.0.0 10.10.60.1

```

De la misma forma, se asigna una dirección IP a la interfaz de la VLAN 80, para indicar que el switch de acceso 2 va a operar en dentro de dicha VLAN, y así comprobar la comunicación entre VLANs diferentes, también se configura una ruta por defecto para redirigir el tráfico con destino desconocido y permitir la conexión a una red externa como Internet, como muestra en la Figura 50.

Por tanto, este procedimiento de asignar manualmente la IP a cada interfaz resulta repetitivo, el cual toma mayor tiempo de configuración y está propenso a errores, en este contexto, se justifica la implementación de Ansible como herramienta de automatización.

Figura 50*Asignar IP a interfaz de VLAN80 en Switch de acceso 2*

```

sw-acceso2#conf ter
Enter configuration commands, one per line. End with CNTL/Z.
sw-acceso2(config)#int vlan80
sw-acceso2(config-if)#
*May 19 21:23:31.728: %LINEPROTO-5-UPDOWN: Line protocol on Interface Vlan80, changed state to down
sw-acceso2(config-if)#ip add 10.10.80.2 255.255.255.0
sw-acceso2(config-if)#exit
sw-acceso2(config)#int vlan80
sw-acceso2(config-if)#no sh
sw-acceso2(config-if)#
*May 19 21:24:08.950: %LINK-3-UPDOWN: Interface Vlan80, changed state to up
*May 19 21:24:09.950: %LINEPROTO-5-UPDOWN: Line protocol on Interface Vlan80, changed state to up
sw-acceso2(config-if)#exit
sw-acceso2(config)#ip route 0.0.0.0 0.0.0.0 10.10.80.1

```

- **Validación de configuraciones del escenario 1**

La validación de las configuraciones de cada playbook se realiza desde el contenedor Ansible, mediante la ejecución del playbook con el comando `ansible-playbook` seguido del

nombre del archivo que contiene los hosts a los que se aplicarán las configuraciones y luego el nombre del archivo del playbook.

Durante la ejecución del playbook, Ansible proporciona información detallada sobre el estado de cada tarea, las cuales se detallan en la Tabla 30.

Sin embargo, cada playbook contiene varias tareas, por tanto, genera una salida extensa en la consola que detalla el estado de cada acción, por tal motivo en la documentación se indicará únicamente la parte inicial y el resumen final de la ejecución, con el objetivo de verificar tanto el inicio y fin de un proceso de automatización exitoso.

Tabla 30

Estado de ejecución para tareas del playbook

Estado de tarea	Descripción
ok	La tarea ya fue previamente aplicada al dispositivo de manera correcta.
changed	Indica que se produce un cambio en la configuración del dispositivo, ya que se ejecuta la tarea correctamente.
failed	Existe un error al ejecutar la tarea en el dispositivo.
skipped	La tarea fue omitida por condicionales definidos en el playbook.
unreachable	Enumera las conexiones fallidas hacia el host.
rescued	Indica que la tarea fue recuperada de un fallo.
ignored	Cuenta las tareas que fallaron y fueron ignoradas.

También hay que considerar para todos los playbooks, que durante su ejecución pueden generarse advertencias, las cuales no son errores que impiden la aplicación correcta de las tareas en los dispositivos, más bien son mensajes que proporcionan información para mantener buenas prácticas. Las advertencias más comunes en las simulaciones de automatización son: asegurar la idempotencia (evitar ejecutar tareas que ya fueron aplicadas), avisos de

compatibilidad de versiones con colecciones de módulos, y uso de controladores alternativos para conexiones SSH . Estas sugerencias no afectan al resultado final del proceso, por tanto en el resumen final de la ejecución del playbook no se registra tareas fallidas.

Para el primer escenario se valida el funcionamiento del playbook mediante su ejecución con el comando `ansible-playbook -i hosts playbook.yml`, como indica la Figura 51, donde se evidencia como las tareas (TASK) se van aplicando y produciendo cambios en los dispositivos en cada PLAY.

Figura 51
Ejecución del playbook para escenario 1

```

root@Ansible-1: ~/ansible
root@Ansible-1:~/ansible# ansible-playbook -i hosts playbook.yml

PLAY [Configurar router Cisco] *****
*****

TASK [Configurar interfaz de salida y NAT] *****
*****
[WARNING]: To ensure idempotency and correct diff the input configuration lines should be similar to how they appear if present i
n the running configuration on device
changed: [router-core]

TASK [Configurar NAT sobrecargado] *****
*****
changed: [router-core]

TASK [Configurar interfaz principal y subinterfaces] *****
*****
changed: [router-core]

TASK [Configurar NAT inside] *****
*****
ok: [router-core]

```

Al terminar con la ejecución del playbook, en la Figura 52, se evidencia un resumen de los resultados obtenidos (PLAY RECAP) para cada dispositivo gestionado, por tanto, muestra el total de los cambios realizados y las tareas aplicadas correctamente, además al tener un total de 0 fallos, se puede confirmar que el playbook aplicó todas las configuraciones de forma automatizada.

Figura 52
Validación de configuraciones aplicadas en escenario 1

```

PLAY RECAP *****
router-core      : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
sw-acceso1      : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
sw-acceso2      : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
sw-core         : ok=5    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

```

3.5.3. Desarrollo de playbook para escenario 2

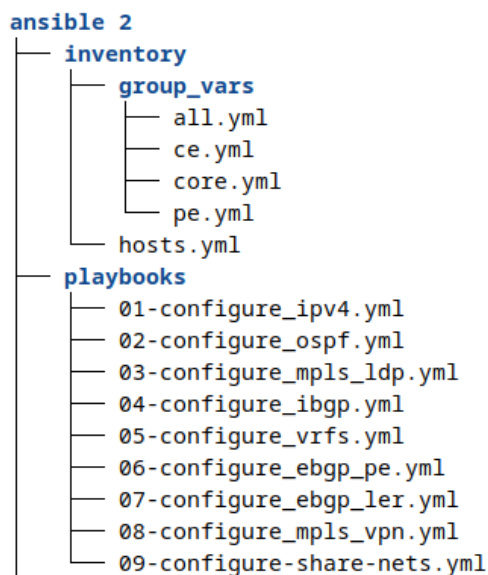
Primeramente, se debe asignar una dirección IP a la interfaz de gestión y habilitar el protocolo SSH en cada dispositivo, con el fin de establecer la conexión remota desde Ansible hacia todos los equipos a configurar, tal como se describe en el proceso del apartado 3.5.1.

Una vez preparado el entorno, se establece la estructura del proyecto ansible para la automatización del segundo escenario, como se muestra en la Figura 53, esta arquitectura se organiza en dos directorios principales que son: inventory y playbooks.

El directorio inventory contiene el archivo 'hosts.yml' donde se listan los hosts a los que se va a conectar y la carpeta 'group_vars' contiene los archivos .yaml que almacena las variables de cada grupo de dispositivos. Por otro lado, el directorio 'playbooks' reúne todos los scripts de configuración, los cuales se dividen por funciones específicas y serán ejecutados uno tras de otro, ya que facilita la comprensión, reutilización del código y tener una escalabilidad en el proyecto.

Figura 53

Estructura del proyecto Ansible para escenario 2



A continuación, es necesario editar el archivo local del contenedor llamado /etc/hosts ya que, se agrega las direcciones IP de la interfaz de gestión de cada dispositivo y se asocian con el nombre respectivo del host, tal como se muestra en la Figura 54. Esto permite que ansible

resuelva los nombres de host de forma local, por lo tanto las conexiones hacia los equipos de red se establecerán usando el nombre del host en lugar de la dirección IP.

Figura 54

Resolución de nombres de hosts - Escenario 3

```

GNU nano 4.8 /etc/hosts
127.0.1.1    Ansible-1
127.0.0.1    localhost
::1         localhost ip6-localhost ip6-loopback
fe00::0     ip6-localnet
ff00::0     ip6-mcastprefix
ff02::1     ip6-allnodes
ff02::2     ip6-allrouters

172.16.0.2   BCliente1
172.16.0.3   BCliente2
172.16.0.4   PBorde1
172.16.0.5   PNucleo1
172.16.0.6   PNucleo2
172.16.0.7   PBorde2
172.16.0.8   BCliente3
172.16.0.9   BCliente4

```

Siguiendo la estructura del proyecto definida en la Figura 53, se procede a escribir el archivo `hosts.yml` para definir los dispositivos a configurar, para ello, se crea un grupo principal llamado 'all', el cual aloja a todos los dispositivos, y mediante el parámetro 'children' permite dividir en subgrupos según la función que a cumplir los routers que son: core (núcleo), pe (borde de proveedor) y ce (borde del cliente), como se muestra en la Figura 55.

Además, se indica los subgrupos de core y pe, donde cada uno tiene una lista de hosts definidos individualmente con un nombre que represente al equipo y con sus respectivos parámetros de conexión detallados a continuación:

- `ansible_host`: dirección IP o nombre del equipo para establecer la conexión.
- `platform`: clave personalizada opcional, para identificar el sistema operativo del dispositivo y usarla internamente en los playbooks.
- `ansible_connection`: método de conexión hacia los dispositivos de red.
- `ansible_network_os`: sistema operativo del dispositivo de red.

Figura 55
Definición de hosts para equipos Cisco - Escenario 2

```
GNU nano 4.8 inventory/hosts.yml
all:
  children:
    core:
      hosts:
        PNucleo1:
          ansible_host: PNucleo1
          platform: ios
          ansible_connection: network_cli
          ansible_network_os: ios
        PNucleo2:
          ansible_host: PNucleo2
          platform: ios
          ansible_connection: network_cli
          ansible_network_os: ios

  pe:
    hosts:
      PBorde1:
        ansible_host: PBorde1
        platform: ios
        ansible_connection: network_cli
        ansible_network_os: ios
      PBorde2:
        ansible_host: PBorde2
        platform: ios
        ansible_connection: network_cli
        ansible_network_os: ios
```

El siguiente subgrupo de dispositivos listados en el archivo hosts.yml corresponde a los routers ‘ce’, como se indica en la Figura 56, donde a cada equipo se coloca los mismos parámetros de conexión mencionados previamente, pero con sistema operativo ‘routeros’ ya que, corresponden a equipos Mikrotik.

Figura 56
Definición de hosts para equipos Mikrotik - Escenario 2

```
GNU nano 4.8 inventory/hosts.yml
ce:
  hosts:
    BCliente1:
      ansible_host: BCliente1
      platform: routeros
      ansible_connection: network_cli
      ansible_network_os: routeros
    BCliente2:
      ansible_host: BCliente2
      platform: routeros
      ansible_connection: network_cli
      ansible_network_os: routeros
    BCliente3:
      ansible_host: BCliente3
      platform: routeros
      ansible_connection: network_cli
      ansible_network_os: routeros
    BCliente4:
      ansible_host: BCliente4
      platform: routeros
      ansible_connection: network_cli
      ansible_network_os: routeros
```

Posteriormente, se define las variables para cada grupo de dispositivos declarados en el inventario 'hosts.yml', las cuales se almacenan en archivos .yml diferentes dentro de la carpeta 'group_vars'. Dichos archivos, deben tener el mismo nombre que el grupo de dispositivos, ya sea 'core', 'ce', 'pe' o para todo el grupo en general 'all'.

En el caso de la Figura 57, indica las siguientes variables comunes que serán aplicas a todos los dispositivos que pertenezcan al grupo 'all', para evitar repetir dichos valores en el archivo hosts.yml:

- `ansible_user`: nombre del usuario privilegiado creado previamente en el dispositivo para establecer el inicio de sesión SSH.
- `ansible_password`: contraseña del usuario para la conexión.

Figura 57

Definición de variables para todos los dispositivos en escenario 2



```
GNU nano 4.8 all.yml
ansible_user: ansible
ansible_password: ansible
```

El siguiente archivo escrito es core.yml, el cual alojar variables específicas, como se muestra en la Figura 58, las cuales, serán usadas posteriormente para la configuración de los routers de núcleo de la red MPLS, identificados como PNucleo1 y PNucleo2.

Las variables definidas para cada router de núcleo (core) estan agrupadas bajo la clave principal `core_routers`, esto permite acceder desde los playbooks a las siguientes variables de los routers:

- `loopbacks`: variable que contiene la interfaz loopback, indicando su nombre y la dirección IP.
- `physical`: define la lista de interfaces físicas con su respectivo nombre y dirección IP, estas son las interfaces conectadas en la nube MPLS.
- `ospf`: variable que contiene dos claves: el número de proceso de OSPF y las interfaces participantes en dicho proceso (Loopback y físicas).

- `mpls_ldp_interfaces`: define la lista de interfaces donde se habilitará el protocolo MPLS y LDP para el intercambio de etiquetas.
- `ibgp_peers`: esta variable referencia al sistema autónomo (SA) que tienen en común los routers de core, con el fin de identificar al proveedor, ya que, los routers de núcleo solo transportan paquetes conmutados por etiquetas MPLS y no establecen sesiones BGP para intercambiar las rutas de los clientes,

Figura 58*Definición de variables para routers de núcleo en escenario 2*

```

GNU nano 4.8                                core.yml
core_routers:
  PNucleo1:
    loopbacks:
      - name: loopback0
        ipv4: 4.4.4.4/32
    physical:
      - name: f0/0
        ipv4: 192.168.0.1/30
      - name: f0/1
        ipv4: 192.168.0.5/30

    ospf:
      process: 1
      interfaces:
        - Lo0
        - f0/0
        - f0/1
    mpls_ldp_interfaces:
      - f0/0
      - f0/1
    ibgp_peers:
      - neighbor: 5.5.5.5
        remote_as: 65000

  PNucleo2:
    loopbacks:
      - name: loopback0
        ipv4: 5.5.5.5/32
    physical:
      - name: f0/0
        ipv4: 192.168.0.6/30
      - name: f0/1
        ipv4: 192.168.0.9/30

    ospf:
      process: 1
      interfaces:
        - Lo0
        - f0/0
        - f0/1
    mpls_ldp_interfaces:
      - f0/0
      - f0/1
    ibgp_peers:
      - neighbor: 4.4.4.4
        remote_as: 65000

```

El siguiente archivo de variables alojado en la carpeta ‘group_vars’ corresponde a pe.yml, en donde contiene variables para los routers de borde de proveedor, las cuales están agrupadas bajo la clave de nivel superior ‘pe_routers’ y dentro de ella se define variables tanto para el PBorde1 y PBorde2.

Para cada router de borde de proveedor (PE) se define variables similares a las del router de núcleo que son loopbacks, physical, ospf, mpls donde en cada una se lista las interfaces correspondientes para una correcta operación del núcleo MPLS, además se agregan las siguientes variables para el PBorde1, ilustradas en la Figura 59:

- asn: se especifica el valor, que corresponde al número del sistema autónomo al que pertenece el router, el cual, corresponde al mismo del proveedor (65000).
- ibgp_peers: esta variable define el router vecino (RBorde2) especificando su IP y el número del sistema autónomo, el cual es el mismo que RBorde1, correspondiente a 65000. El ASN al ser igual entre ellos, pertenecen al mismo SA, siendo posible establecer sesiones iBGP para el intercambio de rutas entre routers de borde, ya que, son los únicos que conocen las rutas de los clientes.
- vrfs: es la variable que contiene la lista de instancias VRFs (Virtual Routing and Forwarding) a definir en cada router de borde, con el fin de permitir que un router tenga varias tablas de enrutamiento, las cuales son independientes para aislar las redes de distintos clientes. Para ello, cada VRF se estructura con los siguientes parámetros:
 - name: nombre de la VRF que representa a una red virtual privada (VPN) para aislar el tráfico de un cliente en específico dentro del router de borde. Cada router de borde tiene 2 VRFs que son ‘red1’ y ‘red2’.
 - rd: hace referencia al Route Distinguisher, el cual tiene el siguiente formato RBorde: ID de VRF, y sirve como un identificador único de cada

VRF, ya que, el router de borde añade el RD a cada ruta que pertenece a esa VRF, con el fin de diferenciar de otras rutas que tengan las mismas IPs en otras VRFs.

- `route_targets`: define los Route Targets (RT) mediante el siguiente formado ASN: identificador, los cuales son usados para clasificar rutas que serán importadas y exportadas entre VRFs situadas en diferentes routers de borde. Ya que, si dos VRFs comparten los mismos RT forman parte de la misma VPN y pueden compartir rutas.
- `interfaces`: variable que define la interfaz asociada a cada VRF, es decir la interfaz del router de borde que va hacia el router cliente.
- `ebgp_peers`: esta variable define por cada VRF las sesiones eBGP, entre el router de borde de proveedor (PE) y el router cliente (CE), ya que pertenecen a distintos sistemas autónomos. Por tanto, se especifica la dirección IP y ASN del router cliente vecino (BCliente), para que sea posible anunciar las rutas del CE al PE y sean almacenadas en su VRF correspondiente.

Además, se debe considerar que el número ASN de cada CE es distinto, para ser posible el aislamiento lógico entre clientes.

Figura 59*Definición de variables para router de borde1 en escenario 2*

```

GNU nano 4.8                                     pe.yml
pe_routers:
  PBorde1:
    asn: 65000
    loopbacks:
      - name: loopback0
        ipv4: 3.3.3.3/32
    physical:
      - name: f1/0
        ipv4: 192.168.0.2/30
    ospf:
      process: 1
      interfaces:
        - f1/0
    mpls_ldp_interfaces:
      - f1/0
    ibgp_peers:
      - neighbor: 6.6.6.6
        remote_as: 65000
    vrf:
      - name: red1
        rd: "3.3.3.3:1" # Route Distinguisher unico para red1 en PBorde1
        route_targets:
          export: "65000:100" # Exporta rutas con este RT
          import: "65000:100" # Importa rutas que llevan este RT
        interfaces:
          - name: f0/0
            ipv4: 192.168.50.1/30
        ebgp_peers:
          - neighbor: 192.168.50.2
            remote_as: 65001

      - name: red2
        rd: "3.3.3.3:2" # Route Distinguisher unico para red2 en PBorde1
        route_targets:
          export: "65000:200" # Exporta rutas con este RT
          import: "65000:200" # Importa rutas que llevan este RT
        interfaces:
          - name: f0/1
            ipv4: 192.168.50.5/30
        ebgp_peers:
          - neighbor: 192.168.50.6
            remote_as: 65002

```

A continuación, se muestra en la Figura 60, el siguiente fragmento del archivo `pe.yml` correspondiente a las variables del router `PBorde2`, las cuales mantienen la misma estructura usada en el `PBorde1`. Por tanto, se indica que ambos routers de borde comparten el mismo ASN (65000) para establecer sesión iBGP y sea posible intercambiar rutas de los clientes, además se define las mismas dos VRFs (`red1` y `red2`), pero en cada una con su respectivo Route Distinguer (RD) único, mientras que los Route Targets (RT) son iguales a los definidos en el `PBorde1` para que sea posible exportar e importar las rutas entre VRFs que pertenecen a la

misma VPN. Por último, se establece las sesiones eBGP entre el PE con los routers clientes vecinos correspondientes a BCliente3 y BCliente4, donde cada uno tiene un ASN distinto que representa una red independiente para cada cliente.

Figura 60

Definición de variables para router de borde2 en escenario 2

```
GNU nano 4.8                                pe.yml
PBorde2:
  asn: 65000
  loopbacks:
    - name: loopback0
      ipv4: 6.6.6.6/32
  physical:
    - name: f0/0
      ipv4: 192.168.0.10/30
  ospf:
    process: 1
    interfaces:
      - f0/0
  mpls_ldp_interfaces:
    - f0/0
  ibgp_peers:
    - neighbor: 3.3.3.3
      remote_as: 65000

  vifs:
    - name: red1
      rd: "6.6.6.6:1" # Route Distinguisher unico para red1 en PBorde2
      route_targets:
        export: "65000:100" # Exporta rutas con este RT (coincide con PBorde1 red1)
        import: "65000:100" # Importa rutas que llevan este RT (coincide con PBorde1 red1)
      interfaces:
        - name: f0/1
          ipv4: 192.168.50.9/30
      ebgp_peers:
        - neighbor: 192.168.50.10
          remote_as: 65003

    - name: red2
      rd: "6.6.6.6:2" # Route Distinguisher unico para red2 en PBorde2
      route_targets:
        export: "65000:200" # Exporta rutas con este RT (coincide con PBorde1 red2)
        import: "65000:200" # Importa rutas que llevan este RT (coincide con PBorde1 red2)
      interfaces:
        - name: f1/0
          ipv4: 192.168.50.13/30
      ebgp_peers:
        - neighbor: 192.168.50.14
          remote_as: 65004
```

El último archivo correspondiente a ce.yml contiene las variables para el grupo de routers clientes (ce) conformado por BCliente1, BCliente2, BCliente3 y BCliente4.

La primera parte del archivo se muestra en la Figura 61, donde inicialmente se define la variable principal 'ce_routers' la cual agrupa las siguientes subvariables internas que contienen valores específicos de los routers BCliente1 y BCliente2:

- **asn:** define el número del sistema autónomo (ASN) al que pertenece cada router cliente (BCliente). Dicho valor es diferente para cada router cliente, para diferenciarlos como redes independientes al establecer sesiones eBGP con los routers de borde del proveedor (PE).
- **loopbacks:** se coloca como valor la interfaz Loopback del router, mediante la definición de su nombre y dirección IP.
- **physical:** lista todas las interfaces físicas del router, cada una con su respectivo nombre y dirección IP.
- **ebgp_peers:** permite definir las sesiones eBGP entre el CE y PE, por ende se coloca la IP y el ASN (65000) correspondiente al router de borde del proveedor vecino.

Figura 61

Definición de variables para routers clientes en escenario 2- Parte 1

```

GNU nano 4.8                                ce.yml
ce_routers:
  BCliente1:
    asn: 65001
    loopbacks:
      - name: lo
        ipv4: 1.1.1.1/32
    physical:
      - name: ether2
        ipv4: 172.16.0.2/25
      - name: ether3
        ipv4: 192.168.100.1/26
      - name: ether1
        ipv4: 192.168.50.2/30
    ebgp_peers:
      - neighbor: 192.168.50.1
        remote_as: 65000
  BCliente2:
    asn: 65002
    loopbacks:
      - name: lo
        ipv4: 2.2.2.2/32
    physical:
      - name: ether2
        ipv4: 172.16.0.3/25
      - name: ether3
        ipv4: 192.168.100.65/26
      - name: ether1
        ipv4: 192.168.50.6/30
    ebgp_peers:
      - neighbor: 192.168.50.5
        remote_as: 65000

```

Seguidamente, se termina de definir las variables para los routers clientes restantes que son BCliente3 y BCliente4 dentro de mismo archivo ‘ce.yml’, como se observa en la Figura 62. Dichas variables mantienen la misma estructura empleada en los routers BCliente1 y BCliente2, ya que están agrupadas bajo la clave principal ‘ce_routers’, sin embargo para cada enrutador se debe colocar los valores correspondientes a sus variables asn, loopbacks, physical y ebgp.

Figura 62

Definición de variables para routers clientes en escenario 2- Parte 2

```
GNU nano 4.8                                ce.yml                                M
BCliente3:
  asn: 65003
  loopbacks:
    - name: lo
      ipv4: 7.7.7.7/32
  physical:
    - name: ether2
      ipv4: 172.16.0.8/25
    - name: ether3
      ipv4: 192.168.100.129/26
    - name: ether1
      ipv4: 192.168.50.10/30
  ebgp_peers:
    - neighbor: 192.168.50.9
      remote_as: 65000
BCliente4:
  asn: 65004
  loopbacks:
    - name: lo
      ipv4: 8.8.8.8/32
  physical:
    - name: ether2
      ipv4: 172.16.0.9/25
    - name: ether3
      ipv4: 192.168.100.193/26
    - name: ether1
      ipv4: 192.168.50.14/30
  ebgp_peers:
    - neighbor: 192.168.50.13
      remote_as: 65000
```

Al terminar de definir los hosts y las variables, se procede a detallar las instrucciones establecidas dentro de cada playbook, que permitirán la configuración automatizada de los dispositivos de red en el escenario 2.

En total se han desarrollado nueve playbooks, donde cada uno tiene una función específica para la configuración, además están estructurados con un único PLAY, el cual, agrupa varias tareas individuales que serán ejecutadas secuencialmente.

El primer playbook a desarrollar se denomina ‘01-configure_ipv4.yml’ que tiene como objetivo configurar el direccionamiento IPv4, usando las instrucciones presentadas en la Tabla 31, las cuales son propias de Ansible para escribir el contenido del archivo de manera jerárquica.

Tabla 31

Instrucciones del playbooks para configuración IPv4 del escenario 2

Instrucción	Descripción
vars	Crea una sección para declarar variables localmente en el playbook.
debug:	Imprime durante la ejecución del playbook datos o resultados de comandos de verificación en las tareas.
var	Permite mostrar el valor de una variable
when	Condición que evalúa una expresión.
block	Permite, agrupar varias tareas para ejecutarlas en conjunto bajo una condición.
cisco.ios.ios_config	Módulo ios_config que pertenece a la colección de cisco.ios para ejecutar las líneas de las tareas en equipos Cisco.
ansible.netcommon.cli_command	Módulo genérico para ejecutar comandos en equipos, en este caso en dispositivos Mikrotik.
commands	Permite listar los comandos para dispositivos Mikrotik.
parents	Define donde ejecutar las líneas de comandos en el CLI.
loop	Repite la tarea para configurar cada elemento de una lista.
register	Guarda en una variable el resultado de la configuración de una tarea.
split('/')	Separa una cadena texto en dos partes usando el carácter (/)

El playbook llamado ‘01-configure_ipv4.yml’, se inicia con un encabezado, que se define mediante palabras claves de Ansible, el título del PLAY (Configurar IPv4 en interfaces), los hosts sobre los que se ejecutará las tareas (all) y desactivar la recopilación de hechos o información del sistema del host remoto (gather_facts: no), como se muestra en la Figura 63.

Además, se usa la palabra clave de Ansible ‘vars’ para definir una sección donde se declara localmente en el playbook las siguientes variables:

- **prefix_to_netmask:** se declara un diccionario para traducir el formato de representación de una máscara de subred, por tanto, se asocia la longitud del prefijo CIDR (Enrutamiento entre dominios sin clases) con la máscara de

subred equivalente en formato decimal. Esto es necesario al momento de configurar los dispositivos Cisco, ya que no aceptan notación CIDR.

- `device_map`: esta variable tiene un nombre personalizado y tiene la función de obtener de forma automática los datos de configuración del dispositivo que se está procesando en ese momento, dichos datos son obtenidos de los archivos de variables.

Para ello, se utiliza una expresión condicional, donde primero se identifica mediante `inventory_hostname` el nombre del dispositivo que se está procesando, para así evaluar a que grupo del inventory pertenece (core, pe y ce), y posteriormente poder relacionarlo con las variables declaradas para cada grupo (`core_routers`, `pe_routers` o `ce_routers`), con el fin de seleccionar y cargar los datos definidos en las variables de cada host.

Figura 63

Establecimiento del bloque PLAY para IPv4 en escenario 2

```
GNU nano 4.8 01-configure_ipv4.yml
- name: Configurar IPv4 en interfaces
  hosts: all
  gather_facts: no

  vars:
    prefix_to_netmask:
      8: "255.0.0.0"
      16: "255.255.0.0"
      24: "255.255.255.0"
      30: "255.255.255.252"
      32: "255.255.255.255"

    device_map: >
    {{
      (core_routers[inventory_hostname] if 'core' in group_names else
      pe_routers[inventory_hostname] if 'pe' in group_names else
      ce_routers[inventory_hostname])
    }}

  tasks:
```

Posteriormente, dentro del mismo playbook '01-configure_ipv4.yml' se procede a listar las acciones agrupadas en la sección `tasks`. Un fragmento de las primeras tareas se muestra en la Figura 64, las cuales corresponden a la configuración de direccionamiento IPv4 para equipos Cisco y se detallan a continuación:

- Debug `device_map`: imprime o muestra el contenido de la variable `device_map` al momento de ejecutar el playbook, con el fin de verificar que las variables a cargar en los dispositivos son las correctas.
- Configurar Loopback en Cisco IOS: asigna la dirección IP con su respectiva máscara a la interfaz Loopback de cada router Cisco.

Por tanto, mediante el *loop* se recorre las interfaces Loopback definidas en la variable `device_map.loopbacks`, ya que, contiene los datos de las variables del dispositivo que se está configurando en ese momento. Para luego, dentro del contexto correspondiente al nombre de la interfaz (`interface {{ item.name }}`) se asigna las líneas de configuración.

La primera línea indica una evaluación de las variables para construir el comando final. En la primera variable, se extrae la dirección IP, tal como se encuentra en el archivo de variables (ejemplo: '4.4.4.4/32') pero con la función `split('/')` se separa en dos cadenas diferentes, la dirección IP (`[0]`) y el prefijo (`[1]`) (ejemplo: '4.4.4.4' , '32'), esto permite seleccionar únicamente la primera parte de la cadena con `[0]`, es decir, la dirección IP limpia que se asignará a la interfaz.

La segunda variable se evalúa para obtener la máscara de subred en formato decimal, para ello, primero se obtiene la dirección IP completa, luego se separa la IP de su prefijo. Con el fin de obtener la segunda parte (`[1]`) correspondiente al prefijo (ejemplo: '32'), el cual se convierte en un número entero con la función `int` (ejemplo: 32). Una vez identificado el valor del prefijo, se procede a consultar en el diccionario definido en la variable `prefix_to_netmask` para devolver el valor equivalente en formato decimal de la máscara de subred (ejemplo: 255.255.255.255).

Esta tarea está condicionada, ya que solo se ejecutará la acción si el dispositivo tiene sistema operativo Cisco, finalmente el resultado de la ejecución de la tarea será guardado en la variable *loopback_config_result*.

- Mostrar resultado Loopback IOS: esta tarea usa la palabra clave de Ansible *debug* para imprimir el resultado de la tarea anterior, que se encuentra guardado en la variable *loopback_config_result*, además se ejecuta solo si el sistema operativo del equipo es Cisco.
- Configurar interfaces físicas en Cisco IOS: se establece esta tarea para asignar una dirección IP con su respectiva máscara de subred en todas las interfaces físicas de los routers con sistema operativo Cisco.

Para ello, primero se extrae la IP en formato CIDR de la variable *item.ipv4*, la cual, se divide en dos partes, el primer elemento es la IP ([0]) y la segunda parte es el prefijo ([1]). En el caso de la primera variable a evaluar, es para seleccionar con [0] la primera parte de la cadena, correspondiente a la IP. Mientras que la segunda variable a evaluar es para obtener la máscara de subred en formato decimal equivalente al prefijo ([1]), que se encuentran situados en el diccionario establecido en la variable *prefix_to_netmask*.

Esta evaluación de variables para la asignación de IP se realiza para cada interfaz listada en la variable *device_map.physical* del dispositivo que está siendo procesado. Por último, el resultado de la tarea se guarda en la variable *physical_config_result*.

- Mostrar resultado interfaces físicas IOS: mediante la palabra clave *debug*, imprime el valor guardado en la variable *physical_config_result* con el fin de verificar sus cambios.

Figura 64

Configuración de IPv4 en equipos Cisco en escenario 2

```

GNU nano 4.8                                01-configure_ipv4.yml
tasks:
  - name: Debug device_map
    debug:
      var: device_map

  # Cisco IOS - Configurar interfaces Loopback
  - name: Configurar Loopback en Cisco IOS
    cisco.ios.ios_config:
      lines:
        - "ip address {{ item.ipv4.split('/')[0] }} {{ prefix_to_netmask[item.ipv4.split('/')[1]|int] }}"
        - no shutdown
      parents: "interface {{ item.name }}"
      loop: "{{ device_map.loopbacks }}"
      when: platform == 'ios'
      register: loopback_config_result

  - name: Mostrar resultado Loopback IOS
    debug:
      var: loopback_config_result
      when: platform == 'ios'

  # Cisco IOS - Configurar interfaces fisicas
  - name: Configurar interfaces fisicas en Cisco IOS
    cisco.ios.ios_config:
      lines:
        - "ip address {{ item.ipv4.split('/')[0] }} {{ prefix_to_netmask[item.ipv4.split('/')[1]|int] }}"
        - no shutdown
      parents: "interface {{ item.name }}"
      loop: "{{ device_map.physical }}"
      when: platform == 'ios'
      register: physical_config_result

  - name: Mostrar resultado interfaces fisicas IOS
    debug:
      var: physical_config_result
      when: platform == 'ios'

```

A continuación, se detallan las siguientes tareas del playbook '01-configure_ipv4.yml', que se muestran en la Figura 65, las cuales se enfocan solo a dispositivos con sistema operativo routers que corresponde a dispositivos Mikrotik.

- Crear interfaz loopback (bridge) en RouterOs: asegura la creación de la interfaz bridge en equipos Mikrotik, para que simule una loopback, ya que Mikrotik no tiene una interfaz loopback nativa como en equipos Cisco.
- Por tanto, el comando que se envía es para crear una interfaz *bridge*, con el nombre extraído de la variable *item.name*, el cual corresponde a la loopback (*lo*) del dispositivo que está siendo configurado en ese momento. Esta tarea se repite para la lista de loopbacks definidas en el archivo de variables del dispositivo actual.

Por último, el resultado de la ejecución de esta tarea se guarda en la variable *bridge_result*.

- Mostrar resultado de creación bridge RouterOs: imprime la información guardada en la variable *bridge_result* mediante el módulo *debug*, además se ejecutará bajo la condición que el sistema operativo sea routers.
- Asignar IP a interfaz loopback en RouterOs: la tarea tiene el fin de asignar la IP a cada interfaz loopback del dispositivo, por tanto se obtiene mediante la extracción de la IP desde la variable *item.ipv4* y se aplica a la interfaz con su respectivo nombre que se encuentra en la variable *item.name*, donde los datos de ambos elementos están alojados en *device_map.loopbacks*.

Esta tarea se repite para las interfaces loopbacks listadas del dispositivo actual y se guarda el resultado en la variable *loopback_ip_result*.

- Mostrar resultado asignación IP loopback RouterOS: esta tarea imprime el resultado de la asignación de direcciones IP a interfaces loopbacks, que se encuentra guardado en la variable *loopback_ip_result*. Los resultados serán mostrados, solo si el sistema operativo es de equipos Mikrotik.
- Configurar interfaces físicas en RouterOS: se asigna una dirección IP a las interfaces físicas del dispositivo que se está configurando en ese momento.

Para ello, mediante *loop* se itera sobre la lista de interfaces físicas del dispositivo actual, que están alojadas sus datos en la variable *device_map.physical*, por tanto, las variables *item.ipv4* y *item.name* toman el valor de la interfaz que se está procesando. Por último, el resultado obtenido de la ejecución de esta tarea será guardado en la variable *physical_ip_result*.

- Mostrar resultado configuración física RouterOS: mediante el módulo *debug* es posible imprimir el resultado guardado en la variable *physical_ip_result*, sin embargo, se mostrará el resultado solo si el equipo es Mikrotik.

Figura 65

Configuración de IPv4 en equipos Mikrotik en escenario 2

```
GNU nano 4.8 01-configure_ipv4.yml

# Mikrotik RouterOS - Crear interfaz loopback (bridge)
- name: Crear interfaz loopback (bridge) en RouterOS
  ansible.netcommon.cli_command:
    command: "/interface bridge add name={{ item.name }}"
  loop: "{{ device_map.loopbacks }}"
  when: platform == 'routeros'
  register: bridge_result

- name: Mostrar resultado creacion bridge RouterOS
  debug:
    var: bridge_result
  when: platform == 'routeros'

# Mikrotik RouterOS - Asignar IP a interfaz loopback
- name: Asignar IP a interfaz loopback en RouterOS
  ansible.netcommon.cli_command:
    command: "/ip address add address={{ item.ipv4 }} interface={{ item.name }}"
  loop: "{{ device_map.loopbacks }}"
  when: platform == 'routeros'
  register: loopback_ip_result

- name: Mostrar resultado asignacion IP loopback RouterOS
  debug:
    var: loopback_ip_result
  when: platform == 'routeros'

# Mikrotik RouterOS - Configurar interfaces fisicas
- name: Configurar interfaces fisicas en RouterOS
  ansible.netcommon.cli_command:
    command: "/ip address add address={{ item.ipv4 }} interface={{ item.name }}"
  loop: "{{ device_map.physical }}"
  when: platform == 'routeros'
  register: physical_ip_result

- name: Mostrar resultado configuracion fisica RouterOS
  debug:
    var: physical_ip_result
  when: platform == 'routeros'
```

El siguiente playbook a definir, corresponde al archivo '02-configure_ospf.yml', destinado a realizar la configuración del enrutamiento OSPF, usando una serie de instrucciones mostradas en la Tabla 32, sin embargo se listan solo las instrucciones nuevas, ya que las demás se ha abordado en anteriores playbooks, las cuales ya fueron explicadas.

Tabla 32

Instrucciones del playbook para configuración OSPF del escenario 2

Instrucción	Descripción
set_fact	Módulo de Ansible que permite actualizar o definir de manera dinámica, el valor de una variable.

<code>selectattr</code>	Selecciona el elemento de una lista que cumpla con ciertos criterios especificados por el usuario.
<code>map</code>	Permite extraer o aplicar alguna función a valores de una lista previamente definida.

El playbook ‘02-configure_ospf.yml’ inicia con el nombre descriptivo del PLAY llamado, *Configurar OSPF en dispositivos Cisco IOS*, el cual, contiene las tareas para aplicarlas a todos los dispositivos listados en el grupo *core* y *pe*, especificados en el archivo ‘hosts.yml’, como indica la Figura 66. Además se define las siguientes variables de forma local en el playbook que están alojadas dentro de la variable *vars*:

- `ospf_process_id`: corresponde al valor que tomará el identificador (ID) del proceso OSPF, el cual se define el valor 1 para todos los dispositivos.
- `ospf_area`; se especifica el valor del área OSPF para todos los dispositivos, la cual, corresponde al área 0, es decir el área principal.
- `prefix_to_netmask`: esta variable personalizada, declara un diccionario de los prefijos que representa las máscaras de subred, para su traducción de formato, por tanto asocia cada prefijo con su máscara de subred equivalente en formato decimal.
- `prefix_to_wildcard`: esta variable personalizada, declara un diccionario de los prefijos de las máscaras de subred para la traducción a una máscara wildcard, por tanto asocia cada prefijo con su máscara wildcard equivalente en formato decimal.
- `device_map`: esta variable permite obtener los datos de configuración del dispositivo que se está configurando. Para ello, primero mediante *inventory_hostname* se identifica el nombre del dispositivo y posteriormente asociarlo al grupo de hosts y variables correspondientes.

Figura 66

Establecimiento del bloque PLAY para OSPF en escenario 2

```
GNU nano 4.8                                02-configure_ospf.yml
--
- name: Configurar OSPF en dispositivos Cisco IOS
  hosts: core:pe
  gather_facts: no

  vars:
    ospf_process_id: 1
    ospf_area: 0

    prefix_to_netmask:
      8: "255.0.0.0"
      16: "255.255.0.0"
      24: "255.255.255.0"
      30: "255.255.255.252"
      32: "255.255.255.255"

    prefix_to_wildcard:
      8: "0.255.255.255"
      16: "0.0.255.255"
      24: "0.0.0.255"
      30: "0.0.0.3"
      32: "0.0.0.0"

    device_map: >
      {{
        (core_routers[inventory_hostname] if 'core' in group_names else
        pe_routers[inventory_hostname] if 'pe' in group_names else
        ce_routers[inventory_hostname])
      }}

  tasks:
    - name: Crear sentencias de red OSPF con mascara wildcard
```

Una vez definidas las variables, se procede a listar las tareas del playbook ‘02-configure_ospf.yml’ como se muestra en la Figura 67, dichas tareas se detallan a continuación, las cuales, están destinadas a dispositivos de core y borde del proveedor que corresponden a sistema operativo únicamente Cisco.

- Crear sentencias de red OSPF con máscara wildcard: tiene la finalidad de definir las redes que van a ser parte del proceso OSPF, por tanto se asocia la dirección IP con su respectiva wildcard.

Para ello, se usa la función de Ansible *set_fact* para actualizar el valor de forma dinámica la variable *ospf_networks*, la cual almacena la lista de redes que se

van formando en cada iteración de las interfaces loopbacks y físicas del dispositivo actual.

La lista de las redes que se almacenan en ‘ospf_networks’ se inicia vacía, es decir en default ([]), donde se va a ir añadiendo valores a la lista que está compuesto por una dirección IP concatenada a su respectiva wildcard.

Ambas direcciones se extraen desde la variable *item.ipv4* (ejemplo: 192.168.100.3/24), las cuales mediante el uso del separador *split (/)* la cadena se divide en dos partes (ejemplo: ‘192.168.100.3’ ‘24’), y finalmente seleccionar mediante los índices [0] o [1] una de las dos partes, ya sea la dirección o el prefijo de red. En caso, de seleccionar el prefijo (ejemplo: 24) es necesario convertirlo a número entero para ser buscado en el diccionario *prefix_to_wildcard* y obtener el valor equivalente en formato decimal la máscara de wildcard.

- Obtener el router-id desde la interfaz Loopback: esta tarea obtiene el ID del enrutador desde la Loopback, con el fin de identificar al router de manera única en el enrutamiento. Para ello, se define la variable ‘router_id’, la cual irá actualizando sus valores que almacena en su lista, correspondiente a la Loopback del dispositivo, para ello se usa el filtro *selectattr* que permite seleccionar el elemento de la lista cuyo nombre de la interfaz inicie con ‘Lo’, además se extrae el primer valor encontrado en *ipv4* correspondiente a la IP de la interfaz Loopback, en caso de no existir dicho elemento devuelve el valor 0.0.0.0.
- Configurar el proceso OSPF en Cisco IOS: tiene el propósito de designar el proceso OSPF a los dispositivos, reutilizando el valor de la variable *ospf_process_id*

- Configurar el router-id dentro del proceso OSPF: con el fin de identificar al router en la red OSPF, se configura dentro del proceso OSPF definido en la variable *ospf_process_id*, el identificador único ID del enrutador usando los valores de la variable *router_id*, ya que almacena las IP de la Loopback de los dispositivos.
- Aplicar sentencias de red OSPF en Cisco IOS: esta tarea se aplica dentro del contexto *router ospf*, donde se asocia cada red OSPF almacenada en *ospf_networks* con el área especificada en la variable *ospf_area*, para que cada router anuncie sus redes disponibles, correspondientes a sus interfaces físicas.

Figura 67

Configuración de OSPF en equipos Cisco en escenario 2

```

GNU nano 4.8                                02-configure_ospf.yml
tasks:
- name: Crear sentencias de red OSPF con mascara wildcard
  set_fact:
    ospf_networks: "{{ ospf_networks | default([]) + [ item.ipv4.split('/')[0] ~ ' ' ~ prefix_to_wildcard[item.ipv4.split('/')[1] | int] ] }}"
  loop: "{{ device_map.physical + device_map.loopbacks }}"
  when: item.ipv4 is defined

- name: Obtener el router-id desde la interfaz Loopback
  set_fact:
    router_id: >-
    {{ (device_map.loopbacks
      | selectattr('name', 'match', '^Lo')
      | map(attribute='ipv4')
      | map('split', '/')
      | map('first')
      | first) | default('0.0.0.0') }}

- name: Configurar el proceso OSPF en Cisco IOS
  cisco.ios.ios_config:
    lines:
      - "router ospf {{ ospf_process_id }}"

- name: Configurar el router-id dentro del proceso OSPF
  cisco.ios.ios_config:
    lines:
      - "router-id {{ router_id }}"
    parents:
      - "router ospf {{ ospf_process_id }}"

- name: Aplicar sentencias de red OSPF en Cisco IOS
  cisco.ios.ios_config:
    lines:
      - "network {{ item }} area {{ ospf_area }}"
    parents:
      - "router ospf {{ ospf_process_id }}"
  loop: "{{ ospf_networks }}"
  when: ospf_networks is defined

```

A continuación, se escribe el siguiente playbook correspondiente a ‘03-configure_mpls_ldp.yml’, para configurar de manera automatizada MPLS. Para ello, inicialmente, se denomina su PLAY con el nombre *Configurar MPLS LDP en routers Cisco IOS*, el cual contendrá las acciones destinadas a los dispositivos del grupo de *core* y *pe*, tal

como se muestra en la Figura 68, además se determina la variable *device_map*, para acceder información del router que se esté configurando en ese momento, desde las variables del grupo de equipos correspondiente.

Figura 68

Establecimiento del bloque PLAY para MPLS en escenario 2

```
GNU nano 4.8                                03-configure_mpls_ldp.yml
- name: Configurar MPLS LDP en routers Cisco IOS
  hosts: core:pe
  gather_facts: no

  vars:
    device_map: >-
      {{
        core_routers[inventory_hostname] if 'core' in group_names else
        pe_routers[inventory_hostname] if 'pe' in group_names else
        ce_routers[inventory_hostname]
      }}

  tasks:
    - name: Configurar MPLS en dispositivos Cisco IOS
```

Seguidamente, se lista las acciones dentro del bloque *tasks*, donde existe una tarea principal llamada *Configurar MPLS en dispositivos Cisco IOS*, la cual, agrupa subtareas con la palabra clave de Ansible *block* y serán ejecutadas bajo la condición de que, el sistema operativo de los dispositivos sea IOS, ya que son los que pertenecen a la nube MPLS. El primer fragmento de las tareas se muestra en la Figura 69, que incluyen las siguientes:

- Habilitar IP CEF globalmente (requisito para MPLS): de manera global en todos los dispositivos se habilita el requisito *ip cef* (Cisco Express Forwarding) para que funcione MPLS, ya que es un mecanismo de reenvío eficiente de paquetes que permite al router construir tanto la tabla FIB (Forwarding Information Base) para reenvío de paquetes IP y la tabla LFIB (Label Forwarding Information Base) para el reenvío de paquetes etiquetados.
- Habilitar MPLS IP globalmente: la presente subtask habilita MPLS IP mediante el comando *mpls ip* a todos los router pertenecientes a la Nube MPLS, para procesar el tráfico MPLS.

- Configurar router-id para MPLS LDP en Loopback0: esta subtarea tiene la finalidad de configurar el identificador (ID) para cada router, usando la IP de la interfaz llamada *Loopback0*. Sin embargo, se ejecutará bajo la condición ‘when’, indicando que en *device_map* exista datos, también que las interfaces estén definidas bajo la clave *loopbacks* y que haya una interfaz llamada *loopback0*.

Esto se empleará en sesiones LDP para intercambiar etiquetas entre los routers vecinos de la red MPLS, con la finalidad de identificarse al momento de intercambiar etiquetas.

Figura 69

Configuración de MPLS en equipos Cisco en escenario 2- Parte 1

```
GNU nano 4.8                                03-configure_mpls_ldp.yml
tasks:
  - name: Configurar MPLS en dispositivos Cisco IOS
    when: ansible_network_os == "ios"
    block:

      - name: Habilitar IP CEF globalmente (requisito para MPLS)
        cisco.ios.ios_config:
          lines:
            - ip cef

      - name: Habilitar MPLS IP globalmente
        cisco.ios.ios_config:
          lines:
            - mpls ip

      - name: Configurar router-id para MPLS LDP en Loopback0
        cisco.ios.ios_config:
          lines:
            - mpls ldp router-id Loopback0 force
        when:
          - device_map is defined
          - device_map.loopbacks is defined
          - device_map.loopbacks | selectattr('name', 'equalto', 'loopback0') | list | length > 0
```

Posteriormente, dentro del mismo playbook ‘03-configure_mpls_ldp.yml’ se indica las siguientes subtarear restantes para completar la configuración MPLS, las cuales se muestran en la Figura 70:

- Habilitar MPLS en interfaces principales: dentro del modo de configuración de cada interfaz física del router que se está configurando, se aplica los comandos para habilitar MPLS y configurar el protocolo de etiquetado LDP, para que

participen en el intercambio de etiquetas. Esta tarea recorre tanto las interfaces físicas como las loopbacks del dispositivo actual, sin embargo se ejecutará bajo las siguientes condiciones: la interfaz debe tener una IP y nombre definido, el cual debe ser diferente al nombre de loopback0 u otro tipo de interfaces administrativas, sin importar que el nombre este en minúsculas o mayúsculas (ignorecase=True; lower), esto debido a que MPLS opera en interfaces físicas.

- Verificar configuración MPLS: valida la configuración MPLS, ejecutando los comandos *show mpls interfaces* y *show mpls ldp neighbor* en todos los dispositivos, para verificar las interfaces que se encuentra habilitado MPLS y las sesiones LDP con los vecinos. Estos resultados obtenidos de la verificación, se almacena en la variable *mpls_verification*.
- Mostrar resultados de verificación MPLS: imprime los resultados de la verificación de MPLS, en forma de lista (stdout_lines) que están almacenados en la variable *mpls_verification*. Esto se ejecuta si dicha variable esta definida.

Figura 70

Configuración de MPLS en equipos Cisco en escenario 2- Parte 2

```
GNU nano 4.8                                03-configure_mpls_ldp.yml

- name: Habilitar MPLS en interfaces principales
  cisco.ios.ios_config:
    lines:
      - mpls ip
      - mpls label protocol ldp
    parents: "interface {{ item.name }}"
    loop: "{{ (device_map.physical | default([])) + (device_map.loopbacks | default([])) }}"
    when:
      - item.name is defined
      - item.name | lower != "loopback0"
      - not item.name | regex_search('^(\Mgmt|Management|Vlan1)', ignorecase=True)
      - item.ipv4 is defined

- name: Verificar configuracion MPLS
  cisco.ios.ios_command:
    commands:
      - show mpls interfaces
      - show mpls ldp neighbor
    register: mpls_verification
    ignore_errors: yes

- name: Mostrar resultados de verificacion MPLS
  debug:
    var: mpls_verification.stdout_lines
  when: mpls_verification is defined and mpls_verification.stdout_lines is defined
```

El siguiente playbook a desarrollar es el '04-configure_ibgp.yml', el cual, esta compuesto por un solo PLAY que aloja a varias tareas para configurar iBGP en los routers de borde del proveedor (pe) de manera automatizada.

El inicio del playbook se muestra en la Figura 71, donde indica: el nombre del bloque PLAY llamado *Configurar iBGP para MPLS-VPN en Routers Cisco IOS PE*; el grupo de hosts sobre los que se ejecutará las acciones; y la desactivación de la recopilación previa de información de los dispositivos. Además, se define las siguientes variables de manera local en el bloque *vars* del archivo de configuración:

- *device_data*: variable personalizada para acceder la información del router que se está procesando desde los archivos de variables de cada grupo de dispositivos.
- *bgp_asn*: variable personalizada para extraer el número del sistema autónomo del router desde la información recopilada en la variable *device_data*, específicamente de la clave *asn*. En caso de no existir el valor, se usará por defecto un *asn 65000*.
- *loopback_ip*: es una variable personalizada para extraer la dirección IP de la *loopback* desde la información almacenada en la variable *device_data*, específicamente de la clave *loopbacks*, donde se tomará la primera dirección de la lista (*loopbacks[0]*), y la primera parte de la dirección IP, es decir sin el prefijo (ejemplo: 192.168.100.2)

Figura 71

Establecimiento del bloque PLAY para iBGP en escenario 2

```
GNU nano 4.8                                04-configure_ibgp.yml
---
- name: Configurar iBGP para MPLS-VPN en Routers Cisco IOS PE
  hosts: pe
  gather_facts: no
  connection: network_cli

  vars:
    device_data: >-
      {{
        (core_routers[inventory_hostname] if 'core' in group_names else
        pe_routers[inventory_hostname] if 'pe' in group_names else
        ce_routers[inventory_hostname])
      }}

    bgp_asn: "{{ device_data.asn | default(65000) }}"
    loopback_ip: "{{ device_data.loopbacks[0].ipv4.split('/')[0] }}"

  tasks:
```

Seguidamente, se indica en la Figura 72, las tareas para la configuración de iBGP entre los routers de borde del proveedor, las cuales se detallan a continuación:

- Configurar router-id y vecinos iBGP: esta tarea que ejecuta bajo la condición que exista la clave *ibgp_peers* en la información recopilada en la variable *device_data*, ya que dicha clave contiene la lista de iBGP vecinos.

Además, la tarea se repite para cada vecino listado en la clave *ibgp_peers*, donde las líneas de configuración se escriben dentro del contexto BGP, las cuales permiten definir lo siguiente: el ID del router actual usando la dirección IP de la loopback, que se extrae de la variable *loopback_ip* para su identificación; también, se declara el vecino tomando la IP del *neighbor* definida en la lista *ibgp_peers* y se especifica que pertenece al mismo sistema autónomo; por último se especifica el origen de la sesión BGP (update-source), que corresponde a la IP de la loopback0 del vecino definida en las variables, ya que las sesiones iBGP se establecen entre las IP loopbacks porque los routers PE no están conectados directamente.

Por tanto, la tarea establece la relación iBGP entre los routers PE, los cuales pertenecen al mismo sistema autónomo.

- Activar vecino en `address-family vpnv4`: esta tarea tiene el objetivo de activar la comunicación VPNv4 para transportar las rutas con información adicional de los clientes como RT y RD que pertenecen a la misma familia de direcciones.

Dicha acción itera sobre los vecinos listados en la clave *ibgp_peers* con el fin de aplicar las siguientes configuraciones en el router actual, dentro del contexto BGP y de la familia de direcciones vpnv4:

- Se activa el vecino para establecer la sesión iBGP, en la familia de direcciones VPNv4.
- Permitir el envío de rutas con información extendida de RT y RD, desde el router local hacia el router vecino, para poder identificar a que VRF se debe importar o exportar. A esto se conoce como comunidad extendida BGP.
- Se anuncia el siguiente salto (next-hop-self), tomando el valor de la IP del router vecino escrita en la variable *neighbor*, con el propósito de que anuncie las rutas aprendidas de sus CE hacia el router vecino (PBorde2).

Figura 72*Configuración de iBGP entre routers PE en escenario 2*

```

GNU nano 4.8                                04-configure_ibgp.yml
tasks:
- name: Configurar router-id y vecinos iBGP (remote-as, update-source)
  when: device_data.ibgp_peers is defined
  ios_config:
    parents: "router bgp {{ bgp_asn }}"
    lines:
      - "bgp router-id {{ loopback_ip }}"
      - "neighbor {{ item.neighbor }} remote-as {{ bgp_asn }}"
      - "neighbor {{ item.neighbor }} update-source Loopback0"

    loop: "{{ device_data.ibgp_peers }}"

- name: Activar vecino en address-family vpnv4
  when: device_data.ibgp_peers is defined
  ios_config:
    parents:
      - "router bgp {{ bgp_asn }}"
      - "address-family vpnv4"
    lines:
      - "neighbor {{ item.neighbor }} activate"
      - "neighbor {{ item.neighbor }} send-community extended"
      - "neighbor {{ item.neighbor }} next-hop-self"
    loop: "{{ device_data.ibgp_peers }}"

```

El próximo playbook a desarrollarse corresponde a '05-configure_vfrs.yml' con el fin de crear las instancias VRF, para ello, el archivo de configuración se compone por un único PLAY descrito como *Configurar VRFs e interfaces en routers PE Cisco IOS*, el cual se aplicará sobre el grupo de dispositivos PE (routers de borde del proveedor), además se define las variables: *prefix_to_netmask* para escribir un diccionario que permita convertir los prefijos a máscaras de subred en formato decimal; y también la variable *device_data* para recopilar información del dispositivo que se esté procesando en ese momento, como se muestra en la Figura 73. Además, dentro del bloque PLAY se escribe las siguientes tareas:

- Crear VRFs con rd: en el router que está siendo procesado, se crea las VRFs listadas en la variable *vrf*s, la cual, está localizada en el archivo de variables de los dispositivos *pe.yml*. Para ello, se ejecuta los siguientes comandos:
 - `ip vrf {{ item.name }}`: crea la VRF con su respectivo nombre que se extrae la variable *item.name*.

- `rd {{ item.rd }}`: se asigna el RD (Route Distinguisher) de la VRF correspondiente, ya que se extrae su valor desde la variable *item.rd*.
- Configurar interfaces en VRFs con IP + máscara decimal: esta tarea asigna la interfaz física con su respectiva IP a su VRF correspondiente. Para ello, se va a recorrer con la instrucción *loop* la lista de VRFs del dispositivo actual (*device_data.vrfs*) y también recorrerá el subelemento situado dentro de la VRF, correspondiente a las interfaces físicas listadas en la variable *interfaces* del router actual (*subelements('interfaces')*).

Al recorrer dos elementos, se usa las variables *item.0* e *item.1*, las cuales, alojara temporalmente los valores de la instancia que se está procesando, que es el nombre de la vrf (*item.0*) y la interfaz de esa vrf (*item.1*). Sin embargo, para mejorar su legibilidad se las representa con los nombres de *vrf* para *item.0* y con *iface* para *item.1*.

Entonces, las líneas de configuración que se aplicaran para esta tarea son: acceder a la interfaz para su configuración mediante su nombre que se extrae de la variable *iface.item*; limpiar la dirección IP; asociar la interfaz con la VRF usando el nombre de la vrf que se extrae de la variable *vrf.item*; asignar la dirección IP extraída de la variable *iface.ipv4* pero usando la máscara de subred en formato decimal; por último se debe habilitar la interfaz.

Figura 73*Configuración de VRFs en routers PE en escenario 2*

```

GNU nano 4.8                                05-configure_vrfs.yml
- name: Configurar VRFs e interfaces en routers PE Cisco IOS
  hosts: pe
  gather_facts: no

  vars:
    prefix_to_netmask:
      8: "255.0.0.0"
      16: "255.255.0.0"
      24: "255.255.255.0"
      30: "255.255.255.252"
      32: "255.255.255.255"

    device_data: "{{ pe_routers[inventory_hostname] }}"

  tasks:

    - name: Crear VRFs con rd
      ios_config:
        lines:
          - "ip vrf {{ item.name }}"
          - " rd {{ item.rd }}"
        loop: "{{ device_data.vrfs }}"

    - name: Configurar interfaces en VRFs con IP + mascara decimal
      ios_config:
        lines:
          - "interface {{ iface.name }}"
          - "no ip address"
          - "ip vrf forwarding {{ vrf.name }}"
          - "ip address {{ iface.ipv4.split('/')[0] }} {{ prefix_to_netmask[iface.ipv4.split('/')[1] | int] }}"
          - "no shutdown"
        loop: "{{ device_data.vrfs | subelements('interfaces') }}"
      loop_control:
        label: "{{ item.1.name }}"
      vars:
        vrf: "{{ item.0 }}"
        iface: "{{ item.1 }}"

```

El siguiente playbook a definir, corresponde al archivo ‘06-configure_ebgp_pe.yml’, destinado a realizar la configuración de VRFs y la habilitación de sesiones eBGP en routers de borde del proveedor (pe).

El fragmento de inicio del playbook se muestra en la Figura 74, donde se indica la definición del bloque PLAY llamado *Configurar VRFs, RTs y eBGP por VRF en routers PE* y de la variable *device_data*, la cual, permite acceder a los valores de las variables del dispositivo que se está procesando para configurar. Seguidamente, se establece las siguientes tareas en la sección *tasks*:

- Procesar VRFs con route targets: esta tarea tiene el fin de organizar los datos de las variables de las VRFs para aplicarlas posteriormente en los routers.

Entonces, en la variable *processed_vrfs* contiene un diccionario de las VRFs, el cual, inicia vacío (`default({})`), pero se va actualizando en cada iteración de la lista de VRFs con los siguientes parámetros:

- `name`: nombre de la VRF
 - `rd`: el valor del Route Distinguisher
 - `rt_import`: el valor del Route Target para identificar las rutas a importar, en caso de no existir o estar vacía la variable, se designa como valor por defecto el asn (65000) concatenado al último elemento del RD (`[-1]`), es decir a su sufijo.
 - `rt_export`: se extrae el valor del Route Target para identificar las rutas a exportar, en caso de no existir se realiza el mismo proceso del *rt_import*.
 - `interfaces`: se extrae los valores de la interfaz asociada a la VRF, que corresponden al nombre e IP de la interfaz.
 - `ebgp_peers`: se indica los vecinos asociados a dicha VRF, en caso de no existir se toma una lista vacía *default([])*.
- **Debug - Mostrar VRFs procesadas**: imprime el resultado del diccionario creado en la variable *processed_vrfs*, el cual, contiene la lista de VRFs con sus respectivos parámetros. Además, dicha tarea está condicionada a ejecutarse, si la variable *processed_vrfs* está definida.

Figura 74*Configuración de eBGP en routers PE en escenario 2- Parte 1*

```

GNU nano 4.8                                06-configure_ebgp_pe.yml
--
- name: Configurar VRFs, RTs y eBGP por VRF en routers PE
  hosts: pe
  gather_facts: no
  connection: network_cli

  vars:
    device_data: "{{ pe_routers[inventory_hostname] }}"

  tasks:
    - name: Procesar VRFs con route targets
      set_fact:
        processed_vrfs: >-
          {{
            (processed_vrfs | default([])) + [
              {
                'name': item.name,
                'rd': item.rd,
                'rt_import': item.route_targets.import | default('65000:' ~ (item.rd.split(':')[1])),
                'rt_export': item.route_targets.export | default('65000:' ~ (item.rd.split(':')[1])),
                'interfaces': item.interfaces | default([]),
                'ebgp_peers': item.ebgp_peers | default([])
              }
            ]
          }}
      loop: "{{ device_data.vrfs | default([]) }}"
      when: device_data.vrfs is defined

    - name: Debug - Mostrar VRFs procesadas
      debug:
        var: processed_vrfs
      when: processed_vrfs is defined

```

Las siguientes tareas definidas, se muestran en el fragmento extraído del playbook ‘06-configure_ebgp_pe.yml’ en la Figura 75, que tratan sobre configurar las VRFs con sus respectivos vecinos eBGP, las cuales, se detallan a continuación:

- Configurar VRFs y RTs: esta tarea ya se aplica a los routers PE, para crear las VRFs con sus respectivos parámetros alojados en el diccionario de la variable *processed_vrfs*. Por lo tanto, esta acción se repite para cada VRF del dispositivo actual, a través de la instrucción *loop* y se creará con los siguientes elementos: nombre de la vrf, su Route Distinguisher y los Route Targets de importación y exportación de rutas.

Además, se controla lo que se va a mostrar a la salida de la ejecución del playbook, específicamente de esta tarea, ya que mediante *loop_control* se evidencia el progreso del bucle, para mejorar la legibilidad y entender la VRF que se configura.

- Configurar vecinos eBGP por VRF: para configurar los vecinos eBGP por cada VRF, se usa un bucle combinado que repite la tarea para cada VRF listada en la variable *processed_vrfs* y también tomando como subelemento la lista de los vecinos eBGP alojados en la variable *ebgp_peers* del dispositivo actual.

Una vez que se identifica la VRF a configurar, se procede a definir en el contexto jerárquico que se aplicarán los comandos, que corresponden, al modo de configuración BGP indicando el ASN del router actual (*router bgp {{ device_data.asn }}*) y posteriormente al modo de configuración de familia de direcciones BGP (*address-family ipv4 vrf*) con el fin de indicar los vecinos que pertenecen a la VRF actual.

Después de entrar en el contexto de configuración, se aplican las líneas de comandos para definir el vecino con su respectivo ASN y activarlo dentro de la familia de direcciones.

Figura 75

Configuración de eBGP en routers PE en escenario 2- Parte 2

```
GNU nano 4.8                                06-configure_ebgp_pe.yml
- name: Configurar VRFs y RTs
  ios_config:
    lines:
      - "ip vrf {{ item.name }}"
      - " rd {{ item.rd }}"
      - " route-target import {{ item.rt_import }}"
      - " route-target export {{ item.rt_export }}"
    loop: "{{ processed_vrfs | default([]) }}"
    loop_control:
      label: "Configurando VRF {{ item.name }} con RD {{ item.rd }}"
      when: processed_vrfs is defined

- name: Configurar vecinos eBGP por VRF
  ios_config:
    parents:
      - "router bgp {{ device_data.asn | default(65000) }}"
      - "address-family ipv4 vrf {{ item.0.name }}"
    lines:
      - "neighbor {{ item.1.neighbor }} remote-as {{ item.1.remote_as }}"
      - "neighbor {{ item.1.neighbor }} activate"
    loop: "{{ processed_vrfs | default([]) | selectattr('ebgp_peers', 'defined') | subelements('ebgp_peers') }}"
    loop_control:
      label: "VRF {{ item.0.name }} ^f^r vecino {{ item.1.neighbor }}"
      when: processed_vrfs is defined
```

El siguiente paso, es desarrollar el playbook '07-configure_ebgp_ler.yml', el cual, tiene como objetivo automatizar la configuración del proceso eBGP específicamente en los routers

de borde del cliente (CE) que tienen sistema operativo RouterOS. Este playbook está compuesto por un único PLAY llamado *Configurar eBGP en CE RouterOS*, que agrupa un conjunto de tareas.

Al inicio del archivo de configuración se define las siguientes variables dentro del bloque *vars*, las cuales se muestra en la Figura 76:

- *device_data*: variable que se encarga de recopilar la información desde el archivo de variables del dispositivo *ce* que se está procesando para su configuración.
- *router_id*: variable para extraer la dirección IP de la loopback desde la información recopilada en *device_data*, ya que la loopback será el router-id del router actual.

Sin embargo, la dirección loopback en el archivo de variables está en notación CIDR (ejemplo: 1.1.1.1/32), por tanto se transforma en una lista conformada por dos partes mediante el filtro *split* para extraer solo la dirección IP sin el prefijo de red (ejemplo: 1.1.1.1).

- *bgp_template_name*: variable que define el nombre de la plantilla BGP que se creará posteriormente para escribir los parámetros de las conexiones eBGP de manera centralizada.

Seguidamente, se establece las tareas dentro de la sección *tasks*, para configurar las conexiones eBGP entre los router PE y CE. Dichas acciones, se detallan a continuación:

- Crear plantilla BGP en RouterOS: esta tarea se ejecuta si el sistema operativo del dispositivo es RouterOS, la cual consiste en crear la plantilla BGP, usando el nombre de la variable *bgp_template_name* y el ASN con el router-id correspondiente al dispositivo CE actual.

- Eliminar peer BGP existente: la tarea se encarga de eliminar una conexión eBGP previamente creada, para evitar duplicaciones, para ello se busca con *find* y se recorre todas las conexiones que coincidan con el nombre generado, ejemplo: *peer_192.168.100.0*.
- Configurar conexiones eBGP con vecinos PE: para crear la conexión eBGP de los vecinos CE, se debe recorrer la lista de la variable *device_data.ebgp_peers* del router actual, para aplicar los comandos por cada vecino de cada CE. Las líneas de configuración que se aplican son las siguientes:
 - name=peer: nombre de la conexión, usando la dirección IP del router vecino (*item.neighbor*), donde cada octeto se separa por un guion (_) en lugar de punto (.), para que el nombre sea válido, ya que Mikrotik no permite nombres con puntos.
 - remote.address: toma el valor de la clave *item.neighbor*, que corresponde a la dirección IP del router vecino PE.
 - remote.as: valor del ASN del router vecino PE, el cual se extrae mediante la variable *item.remote_as*.
 - template: se usa la plantilla BGP creada previamente llamada *bgp_template_name*, donde se aloja los valores del ASN y router-id del dispositivo CE, ya que es necesario para la conexión con los PE.
 - local.role: se define el rol que cumple el router CE actual en la conexión BGP, la cual corresponde a una sesión eBGP, ya que son diferentes sistemas autónomos.
 - connect: indica que se establecerá la sesión eBGP con el vecino, es decir desde el CE al PE.

- listen: indica que estará escuchando conexiones eBGP entrantes, es decir aceptará sesiones que inicien desde el PE hacia el CE. Por tanto, podrá existir una conexión en ambos sentidos.

Figura 76

Configuración de eBGP en routers CE en escenario 2

```
GNU nano 4.8                                playbooks/07-configure_ebgp_1er.yml
- name: Configurar eBGP en CE RouterOS
  hosts: ce
  gather_facts: no
  connection: network_cli

  vars:
    device_data: "{{ ce_routers[inventory_hostname] }}"
    router_id: "{{ device_data.loopbacks[0].ipv4.split('/')[0] }}"
    bgp_template_name: "ebgp-template"

  tasks:
    - name: Crear plantilla BGP en RouterOS
      community.routeros.command:
        commands:
          - /routing bgp template add name={{ bgp_template_name }} as={{ device_data.asn }} router-id={{ router_id }}
        when: platform == 'routeros'

    - name: Eliminar peer BGP existente (evita duplicados)
      community.routeros.command:
        commands:
          - "/routing bgp connection remove [find name=peer_{{ item.neighbor | replace('.', '_') }}"
        loop: "{{ device_data.ebgp_peers }}"
        when: platform == 'routeros'

    - name: Configurar conexiones eBGP con vecinos PE
      community.routeros.command:
        commands:
          - >
            /routing bgp connection add
            name=peer_{{ item.neighbor | replace('.', '_') }}
            remote.address={{ item.neighbor }}
            remote.as={{ item.remote_as }}
            template={{ bgp_template_name }}
            local.role=ebgp
            connect=yes
            listen=yes
        loop: "{{ device_data.ebgp_peers }}"
        when: platform == 'routeros'
```

A continuación, se desarrolla la configuración para la redistribución de rutas conocidas de cada VRF entre los routers PE en el playbook llamado '08-configure_mpls_vpn.yml', donde cuyo contenido se presenta en la Figura 77. Este archivo de configuración se compone de un PLAY llamado *Configurar PE para MPLS-VPN sobre iBGP en Cisco IOS*, el cual, está destinado a los dispositivos listados en el grupo *pe* (routers de borde del proveedor), además se añade dos variables de manera local en la sección *vars*, que son:

- `device_data`: permite el acceso a los datos del grupo de variables al que pertenece el dispositivo procesado, en este caso se buscará la información en el diccionario `pe_routers`.
- `bgp_asn`: variable que tomará el valor de `asn` correspondiente al router actual, el cual, se extrae desde la información de la variable `device_data.asn`, en caso de no existir el campo `asn`, se procede a usar un valor por defecto de 65000.

Una vez definida las variables, se escribe la tarea en el bloque `tasks`, la cual tiene el nombre *Redistribuir rutas conectadas y estáticas en VRFs* con el objetivo describir la acción a realizar.

Para esta tarea, se usa el módulo `ios_config` que permite el envío de configuraciones de manera jerárquica, ya que los comandos se aplican dentro del contexto BGP y de la familia de direcciones de la VRF correspondiente (`address-family ipv4 vrf`), además la tarea será ejecutada para cada VRF definida en el PE.

Los comandos por aplicar en el proceso BGP son: *redistribute connected* para anunciar a través de BGP las rutas conectadas de manera directa; y *redistribute static*, el cual anunciará rutas estáticas, esto con el fin de anunciar rutas locales de cada cliente en la red MPLS-VPN mediante iBGP y exista conectividad entre los clientes.

Figura 77

Configuración de distribución de rutas en routers PE en escenario 2

```
GNU nano 4.8                                08-configure_mpls_vpn.yml
- name: Configurar PE para MPLS-VPN sobre iBGP en Cisco IOS
  hosts: pe
  gather_facts: no
  connection: network_cli

  vars:
    device_data: "{{ pe_routers[inventory_hostname] }}"
    bgp_asn: "{{ device_data.asn | default(65000) }}"

  tasks:
    - name: Redistribuir rutas conectadas y estaticas en VRFs
      ios_config:
        parents:
          - "router bgp {{ bgp_asn }}"
          - "address-family ipv4 vrf {{ item.name }}"
        lines:
          - "redistribute connected"
          - "redistribute static"
      loop: "{{ device_data.vrfs | default([]) }}"
```

El ultimo playbook desarrollado es ‘09-configure-share-nets.yml’, el cual tiene como objetivo anunciar las redes locales (LANs y su loopback) de los routers CE hacia su vecino PE mediante eBGP, para que sean alcanzadas desde otras sucursales a través de la red MPLS-VPN. Este playbook está compuesto por un solo PLAY llamado *Anunciar Redes (lo y ether3) por BGP en CE RouterOS*, que agrupa un conjunto de tareas, destinadas al grupo de dispositivos ‘ce’ (routers de borde del cliente). Entonces, el inicio del archivo de configuración trata de la definición de las siguientes variables, las cuales se muestra en el fragmento extraído del playbook en la **Figura 78**:

- `device_data`: variable que extrae datos del dispositivo actual, desde el archivo de variables correspondiente al grupo *ce*.
- `lan_prefix`: dicha variable va a almacenar el valor de *lan_network* del dispositivo *ce* procesado en dicho momento, donde *lan_network* está definida en el archivo de variables de equipos *ce*.
- `loopback_entry`: variable que extrae la información completa del campo *loopbacks* del dispositivo actual, donde dichos valores se extraen de la variable *device_data.loopbacks* y tomará el primer elemento de la lista *loopbacks* que coincida con el nombre *lo*.
- `loopback_prefix`: esta variable toma el valor de *ipv4* del elemento extraído en *loopback_entry*.

Figura 78

Establecimiento del PLAY para anunciar rutas CE en escenario 2

```
GNU nano 4.8                                playbooks/09-configure-share-nets.yml
- name: Anunciar LAN y Loopback por BGP (RouterOS v7)
  hosts: ce
  gather_facts: no
  connection: network_cli

  vars:
    device_data: "{{ ce_routers[inventory_hostname] }}"

    lan_prefix: "{{ device_data.lan_network }}"

    loopback_entry: "{{ device_data.loopbacks | selectattr('name', 'equalto', 'lo') | first }}"
    loopback_prefix: "{{ loopback_entry.ipv4 }}"
```

Una vez establecido el bloque PLAY y las variables, se procede a escribir las siguientes tareas dentro de la sección *tasks*, como se muestra en el fragmento extraído del playbook en la en la Figura 79:

- Limpiar reglas existentes del filtro *bgp-out*: la tarea elimina reglas de filtrado específicas de la cadena (chain) *bgp-out* antes de crear nuevas, con el fin de controlar las rutas a anunciar para evitar duplicaciones y fugas de rutas.
- Permitir LAN del cliente por BGP: se crea una regla en la cadena *bgp-out* para permitir que BGP anuncie el prefijo de la red LAN del cliente, ya que representa toda la red (ejemplo: 192.168.100.0/26). El prefijo esta almenado en *lan_prefix* y permite el anuncio con *accept*.
- Permitir loopback por BGP: crea la regla en la cadena *bgp-out* para anunciar por BGP el prefijo de la loopback (ejemplo: 1.1.1.1/32), la cual representa una dirección única como su identificador.
- Rechazar cualquier otro prefijo: se crea una regla en la cadena *bgp-out* para rechazar (*reject*) cualquier otro prefijo que no se acepto anteriormente, y así evitar el anuncio de otras rutas. Ya que, si no se crea reglas se anuncia todas las rutas conectadas.
- Redistribuir rutas conectadas hacia BGP: esta tarea modifica una conexión BGP ya existente, mediante el comando *set*, ya que la conexión entre CE y PE fue creada en el playbook '07-configure_ebgp_ler.yml'.

Para ello, se especifica el nombre de la conexión BGP previamente creada, para distribuir las rutas conectadas y así BGP pueda considerarlas para anunciarlas, siempre y cuando cumplan con las reglas de filtración anteriores.

- Aplicar filtro BGP OUT: se identifica la conexión con el vecino BGP para aplicar las reglas de filtrado de la cadena *bgp-out*, y se pueda controlar el anuncio de prefijos, los cuales, representan las rutas.

Figura 79

Configuración para anunciar rutas CE en escenario 2

```
GNU nano 4.8                                playbooks/09-configure-share-nets.yml
tasks:
  - name: Limpiar reglas existentes del filtro bgp-out
    community.routeros.command:
      commands:
        - "/routing filter rule remove [find chain=bgp-out]"
    when: platform == 'routeros'

  - name: Permitir LAN del cliente por BGP
    community.routeros.command:
      commands:
        - "/routing filter rule add chain=bgp-out rule=\"if (dst == {{ lan_prefix }}) { accept }\""
    when: platform == 'routeros'

  - name: Permitir loopback por BGP
    community.routeros.command:
      commands:
        - "/routing filter rule add chain=bgp-out rule=\"if (dst == {{ loopback_prefix }}) { accept }\""
    when: platform == 'routeros'

  - name: Rechazar cualquier otro prefijo
    community.routeros.command:
      commands:
        - "/routing filter rule add chain=bgp-out rule=\"reject\""
    when: platform == 'routeros'

  - name: Redistribuir rutas conectadas hacia BGP
    community.routeros.command:
      commands:
        - "/routing bgp connection set peer_{{ item.neighbor | replace('.', '_') }} output.redistribute=connected"
    loop: "{{ device_data.ebgp_peers }}"
    when: platform == 'routeros'

  - name: Aplicar filtro BGP OUT (RouterOS v7)
    community.routeros.command:
      commands:
        - "/routing bgp connection set peer_{{ item.neighbor | replace('.', '_') }} output.filter-chain=bgp-out"
    loop: "{{ device_data.ebgp_peers }}"
    when: platform == 'routeros'
```

- **Validación de configuraciones para escenario 2**

La validación de las configuraciones aplicadas en los dispositivos se realiza mediante la ejecución de los playbooks mencionados, con el comando *ansible-playbook* seguido tanto del nombre del archivo de inventario y del nombre del playbook, con su respectiva ruta relativa hacia cada archivo, ya que, Ansible debe ubicar el archivo de los dispositivos (inventory) y los archivos de configuración (playbooks). Ya que, para el escenario 2, se organizó los archivos

por subdirectorios y también las configuraciones en diferentes playbooks según su función, como se hizo referencia en la estructura del proyecto Ansible mostrada en la Figura 53.

Además, se debe considerar que los playbooks se ejecutan individualmente para aplicar las tareas definidas en cada archivo, los cuales están enfocados a funciones específicas como el direccionamiento IPv4, protocolo de enrutamiento OSPF, habilitación de MPLS, configuración de VRFs, enrutamiento iBGP y eBGP.

Para validar la configuración, se presenta solo la parte inicial y final (PLAY RECAP) de la ejecución del playbook, ya que, debido a la alta cantidad de tareas, la salida que se genera en la consola sobre el estado de cada acción es extensa.

Primeramente, se ejecuta el playbook '01-configure_ipv4.yml', el cual se encarga de asignar las direcciones IPv4 a las interfaces de cada dispositivo. Por lo tanto, el comando con el que se ejecuta es *ansible-playbook -i inventory/hosts playbook/01-configure_ipv4.yml*, permitiendo mostrar en pantalla el estado de ejecución de las tareas por cada dispositivo, como se muestra en la Figura 80.

Figura 80

Ejecución del playbook IPv4 referente al escenario 2

```
PLAY [Configurar IPv4 en interfaces] *****

TASK [Debug device_map] *****
ok: [PBorde1] => {
  "device_map": {
    "asn": 65000,
    "ibgp_peers": [
      {
        "neighbor": "6.6.6.6",
        "remote_as": 65000
      }
    ],
    "loopbacks": [
      {
        "ipv4": "3.3.3.3/32",
        "name": "loopback0"
      }
    ],
    "mpls_ldp_interfaces": [
      "f1/0"
    ]
  }
}
1.
```

Al finalizar la ejecución del playbook ‘01-configure_ipv4.yml’, se muestra un resumen en el *PLAY RECAP* sobre el estado de las tareas ejecutadas, como se muestra en la Figura 81. Por tanto, se verifica que todas las tareas fueron aplicadas correctamente (OK) y con reporte de la cantidad de cambios (changed) que se aplicaron sobre todos los dispositivos.

Figura 81
Validación de configuración IPv4 en escenario 2

```
PLAY RECAP *****
BCliente1      : ok=7    changed=0    unreachable=0    failed=0    skipped=4    rescued=0    ignored=0
BCliente2      : ok=7    changed=0    unreachable=0    failed=0    skipped=4    rescued=0    ignored=0
BCliente3      : ok=7    changed=0    unreachable=0    failed=0    skipped=4    rescued=0    ignored=0
BCliente4      : ok=7    changed=0    unreachable=0    failed=0    skipped=4    rescued=0    ignored=0
PBorde1        : ok=5    changed=2    unreachable=0    failed=0    skipped=6    rescued=0    ignored=0
PBorde2        : ok=5    changed=2    unreachable=0    failed=0    skipped=6    rescued=0    ignored=0
PNucleo1       : ok=5    changed=2    unreachable=0    failed=0    skipped=6    rescued=0    ignored=0
PNucleo2       : ok=5    changed=2    unreachable=0    failed=0    skipped=6    rescued=0    ignored=0
```

Después, se ejecuta el segundo playbook correspondiente a ‘02-configure.ospf.yml’ con el comando *ansible-playbook -i inventory/hosts playbook/02-configure_ospf.yml*, para automatizar el enrutamiento OSPF. Por tanto, mientras se ejecuta el playbook, se muestra en la salida de la consola, el estado de cada tarea de manera detallada, tal como se muestra en la Figura 82.

Figura 82
Ejecución del playbook OSPF referente al escenario 2

```
PLAY [Configure OSPF on Cisco IOS devices] *****

TASK [Build OSPF network statements with wildcard mask] *****
ok: [PNucleo1] => (item={'name': 'f0/0', 'ipv4': '192.168.0.1/30'})
ok: [PNucleo2] => (item={'name': 'f0/0', 'ipv4': '192.168.0.6/30'})
ok: [PBorde1] => (item={'name': 'f1/0', 'ipv4': '192.168.0.2/30'})
ok: [PBorde2] => (item={'name': 'f0/0', 'ipv4': '192.168.0.10/30'})
ok: [PNucleo1] => (item={'name': 'f0/1', 'ipv4': '192.168.0.5/30'})
ok: [PNucleo2] => (item={'name': 'f0/1', 'ipv4': '192.168.0.9/30'})
ok: [PBorde1] => (item={'name': 'loopback0', 'ipv4': '3.3.3.3/32'})
ok: [PBorde2] => (item={'name': 'loopback0', 'ipv4': '6.6.6.6/32'})
ok: [PNucleo2] => (item={'name': 'loopback0', 'ipv4': '5.5.5.5/32'})
ok: [PNucleo1] => (item={'name': 'loopback0', 'ipv4': '4.4.4.4/32'})

TASK [Get router-id from Loopback interface] *****
ok: [PNucleo2]
ok: [PBorde2]
ok: [PNucleo1]
ok: [PBorde1]

TASK [Configure OSPF process on Cisco IOS] *****
changed: [PNucleo1]
changed: [PBorde1]
```

El resumen de la ejecución del playbook ‘02-configure.ospf.yml’ se ilustra en la Figura 83, dando como resultado tareas que produjeron cambios en la configuración de los dispositivos, además indica que se aplicaron correctamente sin la existencia de tareas fallidas (failed=0) o dispositivos inalcanzables por problemas de conexión (unreachable=0).

Figura 83

Validación de configuración OSPF en escenario 2

```
PLAY RECAP *****
PBorde1      : ok=5    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PBorde2      : ok=5    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PNucleo1     : ok=5    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PNucleo2     : ok=5    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

El siguiente playbook a ejecutar corresponde a ‘03-configure_mpls_ldp.yml’ a través, del comando *ansible-playbook -i inventory/hosts playbook/03-configure_mpls_ldp.yml*, con el fin de verificar la implementación de las tareas de configuración MPLS en la consola de salida, tal como se muestra en la Figura 84.

Figura 84

Ejecución del playbook MPLS referente al escenario 2

```
PLAY [Configurar MPLS LDP en routers Cisco IOS] *****

TASK [Habilitar IP CEF globalmente (requisito para MPLS)] *****
ok: [PBorde1]
ok: [PNucleo1]
ok: [PNucleo2]
ok: [PBorde2]

TASK [Habilitar MPLS IP globalmente] *****
changed: [PNucleo2]
changed: [PNucleo1]
changed: [PBorde1]
changed: [PBorde2]
```

Al final de la ejecución del playbook ‘03-configure_mpls_ldp.yml’, se muestra en la Figura 85 el resumen del estado de las tareas sobre los routers pertenecientes a la nube MPLS, por tanto se muestra el número total de las tareas que produjeron cambios en la configuración del dispositivo y fueron aplicadas correctamente, sin ningún fallo registrado.

Figura 85*Validación de configuración MPLS en escenario 2*

```
PLAY RECAP *****
PBorde1      : ok=6    changed=2    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
PBorde2      : ok=6    changed=2    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
PNucleo1     : ok=6    changed=2    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
PNucleo2     : ok=6    changed=2    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
```

A continuación, se ejecuta el playbook ‘04-configure_ibgp.yml’ para automatizar la configuración de iBGP entre los routers de borde del proveedor (pe), usando el comando *ansible-playbook -i inventory/hosts playbook/04-configure_ibgp.yml*.

Durante la ejecución, se muestra en la consola el estado de las tareas que se van aplicando para cada dispositivo, en este caso únicamente para los routers PE (PBorde1 y PBorde2). Además, al finalizar se presenta el resumen (PLAY RECAP) donde indica el total de tareas que fueron procesadas y aplicadas con éxito, tal como se muestra en la Figura 86.

Figura 86*Validación de configuración iBGP en routers PE en escenario 2*

```
PLAY [Configurar iBGP para MPLS-VPN en Routers Cisco IOS PE] *****

TASK [Configurar router-id y vecinos iBGP (remote-as, update-source)] *****
changed: [PBorde2] => (item={'neighbor': '3.3.3.3', 'remote_as': 65000})
changed: [PBorde1] => (item={'neighbor': '6.6.6.6', 'remote_as': 65000})

TASK [Activar vecino en address-family vpnv4] *****
changed: [PBorde2] => (item={'neighbor': '3.3.3.3', 'remote_as': 65000})
changed: [PBorde1] => (item={'neighbor': '6.6.6.6', 'remote_as': 65000})

PLAY RECAP *****
PBorde1      : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PBorde2      : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Posteriormente, se ejecuta el playbook ‘05-configure_vrfs.yml’ mediante el comando *ansible-playbook -i inventory/hosts playbook/05-configure_vrfs.yml*, el cual, permite validar las configuraciones para la creación de VRFs en los routers PE, ya que en consola se muestra a detalle la aplicabilidad de cada tarea definida en el PLAY. Una vez finalizada la ejecución, se presenta el PLAY RECAP, como se muestra en la Figura 87, para indicar el total de tareas que han producido cambios y fueron aplicadas de manera correcta, sin ningún fallo.

Figura 87
Validación de configuración VRFs en escenario 2

```
PLAY [Configurar VRFs e interfaces en routers PE Cisco IOS] *****
*****

TASK [Crear VRFs con rd] *****
*****
changed: [PBorde1] => (item={'name': 'red1', 'rd': '3.3.3.3:1', 'route_targets': {'export': '65000:100', 'import': '65000:100'}, 'interfaces':
[{'name': 'f0/0', 'ipv4': '192.168.50.1/30'}], 'ebgp_peers': [{'neighbor': '192.168.50.2', 'remote_as': 65001}]})
changed: [PBorde2] => (item={'name': 'red1', 'rd': '6.6.6.6:1', 'route_targets': {'export': '65000:100', 'import': '65000:100'}, 'interfaces':
[{'name': 'f0/1', 'ipv4': '192.168.50.9/30'}], 'ebgp_peers': [{'neighbor': '192.168.50.10', 'remote_as': 65003}]})
changed: [PBorde1] => (item={'name': 'red2', 'rd': '3.3.3.3:2', 'route_targets': {'export': '65000:200', 'import': '65000:200'}, 'interfaces':
[{'name': 'f0/1', 'ipv4': '192.168.50.5/30'}], 'ebgp_peers': [{'neighbor': '192.168.50.6', 'remote_as': 65002}]})
changed: [PBorde2] => (item={'name': 'red2', 'rd': '6.6.6.6:2', 'route_targets': {'export': '65000:200', 'import': '65000:200'}, 'interfaces':
[{'name': 'f1/0', 'ipv4': '192.168.50.13/30'}], 'ebgp_peers': [{'neighbor': '192.168.50.14', 'remote_as': 65004}]})

TASK [Configurar interfaces en VRFs con IP + máscara decimal] *****
*****
changed: [PBorde2] => (item=f0/1)
changed: [PBorde1] => (item=f0/0)
changed: [PBorde2] => (item=f1/0)
changed: [PBorde1] => (item=f0/1)

PLAY RECAP *****
*****
PBorde1                : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PBorde2                : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

A continuación, se ejecuta el playbook ‘06-configure_ebgp_pe.yml’ mediante el comando `ansible-playbook -i inventory/hosts playbook/06-configure_ebgp_pe.yml`, que permite validar las configuraciones de ebgp en routers de borde del proveedor (pe), los cuales corresponden a un sistema operativo IOS. Por tanto, en la consola de Ansible se muestra la aplicabilidad de cada tarea definida en el PLAY, como se indica en la Figura 88.

Figura 88
Ejecución del playbook eBGP en routers PE referente al escenario 2

```
PLAY [Configurar VRFs, RTs y eBGP por VRF en routers PE] *****
*****

TASK [Procesar VRFs con route targets] *****
*****
ok: [PBorde1] => (item={'name': 'red1', 'rd': '3.3.3.3:1', 'route_targets': {'export': '65000:100', 'import': '65000:100'}, 'interfaces':
[{'name': 'f0/0', 'ipv4': '192.168.50.1/30'}], 'ebgp_peers': [{'neighbor': '192.168.50.2', 'remote_as': 65001}]})
ok: [PBorde2] => (item={'name': 'red1', 'rd': '6.6.6.6:1', 'route_targets': {'export': '65000:100', 'import': '65000:100'}, 'interfaces':
[{'name': 'f0/1', 'ipv4': '192.168.50.9/30'}], 'ebgp_peers': [{'neighbor': '192.168.50.10', 'remote_as': 65003}]})
ok: [PBorde1] => (item={'name': 'red2', 'rd': '3.3.3.3:2', 'route_targets': {'export': '65000:200', 'import': '65000:200'}, 'interfaces':
[{'name': 'f0/1', 'ipv4': '192.168.50.5/30'}], 'ebgp_peers': [{'neighbor': '192.168.50.6', 'remote_as': 65002}]})
ok: [PBorde2] => (item={'name': 'red2', 'rd': '6.6.6.6:2', 'route_targets': {'export': '65000:200', 'import': '65000:200'}, 'interfaces':
[{'name': 'f1/0', 'ipv4': '192.168.50.13/30'}], 'ebgp_peers': [{'neighbor': '192.168.50.14', 'remote_as': 65004}]})

TASK [Debug - Mostrar VRFs procesadas] *****
*****
ok: [PBorde1] => {
  "processed_vrfs": [
    {
      "ebgp_peers": [
        {
          "neighbor": "192.168.50.2",
          "remote_as": 65001
        },
        {
          "neighbor": "192.168.50.10",
          "remote_as": 65003
        }
      ]
    },
    {
      "ebgp_peers": [
        {
          "neighbor": "192.168.50.6",
          "remote_as": 65002
        },
        {
          "neighbor": "192.168.50.14",
          "remote_as": 65004
        }
      ]
    }
  ]
}
```

El resumen general de la ejecución del playbook ‘06-configure_ebgp_pe.yml’ se presenta en la Figura 89, indicando el total de tareas que han producido cambios y que fueron aplicadas correctamente sin errores.

Figura 89*Validación de configuración eBGP en router PE en escenario 2*

```
PLAY RECAP *****
PBorde1      : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PBorde2      : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Seguidamente, se ejecuta el playbook ‘07-configure_ebgp_ler.yml’ mediante el comando *ansible-playbook -i inventory/hosts playbook/07-configure_ebgp_ler.yml*, el cual, permite validar las configuraciones de conexión eBGP en los routers CE hacia sus vecinos PE, ya que en consola se muestra la aplicación de cada tarea definida en el PLAY, para verificar que las instrucciones son procesadas de forma correcta, como se indica la Figura 90.

Al finalidad la ejecución en el PLAY RECAP se muestra el resumen de la aplicación y cambios realizados de las tareas por cada dispositivo CE.

Figura 90*Validación de configuración eBGP en router CE en escenario 2*

```
PLAY [Configurar eBGP en CE Router05] *****

TASK [Crear plantilla BGP en Router05] *****
changed: [BCliente3]
changed: [BCliente2]
changed: [BCliente4]
changed: [BCliente1]

TASK [Configurar conexiones eBGP con vecinos PE] *****
changed: [BCliente1] => (item={'neighbor': '192.168.50.1', 'remote_as': 65000})
changed: [BCliente2] => (item={'neighbor': '192.168.50.5', 'remote_as': 65000})
changed: [BCliente4] => (item={'neighbor': '192.168.50.13', 'remote_as': 65000})
changed: [BCliente3] => (item={'neighbor': '192.168.50.9', 'remote_as': 65000})

PLAY RECAP *****
BCliente1      : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
BCliente2      : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
BCliente3      : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
BCliente4      : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Después, se ejecuta el playbook ‘08-configure_mpls_vpn.yml’ mediante el comando *ansible-playbook -i inventory/hosts playbook/08-configure_mpls_vpn.yml*, para automatizar la redistribución de rutas locales de cada VRF entre los routers PE, por ende en la Figura 91, se muestra la aplicación de la tarea para verificar que las instrucciones son procesadas de forma correcta. Además, en el resumen de la ejecución (PLAY RECAP) se registra el total de la única tarea que presento cambios y fue aplicada correctamente sin fallos a los routers PE.

Figura 91
Validación de distribución de rutas PE en escenario 2

```
PLAY [Configurar PE para MPLS-VPN sobre iBGP en Cisco IOS] *****
*****

TASK [Redistribuir rutas conectadas y estáticas en VRFs] *****
*****
changed: [PBorde2] => (item={'name': 'red1', 'rd': '6.6.6.6:1', 'route_targets': {'export': '65000:100', 'import': '65000:100'},
'interfaces': [{'name': 'f0/1', 'ipv4': '192.168.50.9/30'}], 'ebgp_peers': [{'neighbor': '192.168.50.10', 'remote_as': 65003}]})
changed: [PBorde1] => (item={'name': 'red1', 'rd': '3.3.3.3:1', 'route_targets': {'export': '65000:100', 'import': '65000:100'},
'interfaces': [{'name': 'f0/0', 'ipv4': '192.168.50.1/30'}], 'ebgp_peers': [{'neighbor': '192.168.50.2', 'remote_as': 65001}]})
changed: [PBorde2] => (item={'name': 'red2', 'rd': '6.6.6.6:2', 'route_targets': {'export': '65000:200', 'import': '65000:200'},
'interfaces': [{'name': 'f1/0', 'ipv4': '192.168.50.13/30'}], 'ebgp_peers': [{'neighbor': '192.168.50.14', 'remote_as': 65004}]})
changed: [PBorde1] => (item={'name': 'red2', 'rd': '3.3.3.3:2', 'route_targets': {'export': '65000:200', 'import': '65000:200'},
'interfaces': [{'name': 'f0/1', 'ipv4': '192.168.50.5/30'}], 'ebgp_peers': [{'neighbor': '192.168.50.6', 'remote_as': 65002}]})

PLAY RECAP *****
*****
PBorde1                : ok=1    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PBorde2                : ok=1    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Finalmente, se ejecuta el playbook '09-configure-share-nets.yml', para automatizar en los dispositivos CE la configuración de anunciar sus redes locales (LANs y su loopback) hacia su vecino PE mediante eBGP. El comando para ejecutar es *ansible-playbook -i inventory/hosts playbook/09-configure-share-nets.yml*, con el que es posible verificar que las instrucciones son procesadas de forma correcta, tal como se muestra en la Figura 92, además en el PLAY RECAP se evidencia las tareas aplicadas de manera exitosa, sin presentar errores.

Figura 92
Validación de anuncio de rutas CE en escenario 2

```
TASK [Permitir loopback por BGP] *****
changed: [BCliente1]
changed: [BCliente2]
changed: [BCliente3]
changed: [BCliente4]

TASK [Rechazar cualquier otro prefijo] *****
changed: [BCliente1]
changed: [BCliente2]
changed: [BCliente4]
changed: [BCliente3]

TASK [Redistribuir rutas conectadas hacia BGP] *****
changed: [BCliente1] => (item={'neighbor': '192.168.50.1', 'remote_as': 65000})
changed: [BCliente4] => (item={'neighbor': '192.168.50.13', 'remote_as': 65000})
changed: [BCliente2] => (item={'neighbor': '192.168.50.5', 'remote_as': 65000})
changed: [BCliente3] => (item={'neighbor': '192.168.50.9', 'remote_as': 65000})

TASK [Aplicar filtro BGP OUT (RouterOS v7)] *****
changed: [BCliente1] => (item={'neighbor': '192.168.50.1', 'remote_as': 65000})
changed: [BCliente3] => (item={'neighbor': '192.168.50.9', 'remote_as': 65000})
changed: [BCliente2] => (item={'neighbor': '192.168.50.5', 'remote_as': 65000})
changed: [BCliente4] => (item={'neighbor': '192.168.50.13', 'remote_as': 65000})

PLAY RECAP *****
BCliente1                : ok=6    changed=6    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
BCliente2                : ok=6    changed=6    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
BCliente3                : ok=6    changed=6    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
BCliente4                : ok=6    changed=6    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

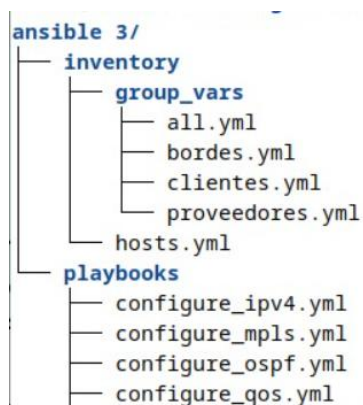
3.5.4. Desarrollo de *playbook* para escenario 3

Inicialmente, se debe realizar las configuraciones descritas en el apartado 3.5.1. correspondiente a la preparación del entorno, en el cual se detalla el proceso para asignar una dirección IP a la interfaz de gestión y habilitar el protocolo SSH en cada dispositivo, con el fin de establecer conexión desde Ansible hacia los equipos a configurar.

Una vez preparado el entorno de trabajo, a continuación, se establece la estructura del proyecto para la automatización, por ende, en la Figura 93, indica la arquitectura establecida para escribir los archivos de configuración que son, el inventario donde guarda las variables y la lista de hosts, mientras que los playbooks se dividen según sus funciones específicas a automatizar, esto permite tener una buena organización, mejor control de errores, facilidad de entender y permitir una ejecución selectiva de las configuraciones en lugar de forma conjunta.

Figura 93

Estructura del proyecto Ansible para escenario 3



Primero, se procede con la definición de los dispositivos en el archivo `hosts.yml`, ya que, las configuraciones serán aplicados sobre ellos. Este archivo inicia especificando un grupo principal llamado 'all', el cual aloja a todos los dispositivos, seguido se coloca la entrada 'children', para que Ansible pueda interpretar la existencia de subgrupos que pertenecen a 'all'. Los subgrupos se establecieron por bloques, que son separados en base a la función que desempeña los routers que son: proveedores, bordes y clientes, además se define un subgrupo llamado servidor.

Entonces, en la Figura 94, indica los subgrupos de proveedores y bordes, donde cada uno tiene una lista de hosts definidos individualmente con un nombre que represente al equipo y con las variables respectivas de ese host que son:

- `ansible_host`: nombre del host o dirección IP del equipo remoto.
- `ansible_network_os`: sistema operativo del equipo remoto, en este caso es 'ios'.

Figura 94

Definición de hosts para equipos Cisco en escenario 3

```
GNU nano 4.8                               hosts.yml
all:
  children:
#Equipos de core LSR
  proveedores:
    hosts:
      P1:
        ansible_host: 10.10.10.1
        ansible_network_os: ios
      P2:
        ansible_host: 10.10.10.2
        ansible_network_os: ios
      P3:
        ansible_host: 10.10.10.3
        ansible_network_os: ios

      P4:
        ansible_host: 10.10.10.4
        ansible_network_os: ios

# Equipos de borde LER
  bordes:
    hosts:
      PE1:
        ansible_host: 10.10.10.5
        ansible_network_os: ios
      PE2:
        ansible_host: 10.10.10.6
        ansible_network_os: ios
      PE3:
        ansible_host: 10.10.10.7
        ansible_network_os: ios
```

De manera similar, en el archivo `host.yml`, se define los hosts para el subgrupo 'clientes' como muestra en la Figura 95, que se asigna un nombre lógico y las variables para colocar la IP o nombre de host del equipo remoto y el sistema operativo, el cual corresponde a 'routers' ya que, son equipos Mikrotik.

Figura 95*Definición de hosts para equipos Mikrotik en escenario 3*

```

GNU nano 4.8                               hosts.yml
# Equipos Mikrotik CLIENTES
clientes:
  hosts:
    CE1:
      ansible_host: 10.10.10.8
      ansible_network_os: routeros
    CE2:
      ansible_host: 10.10.10.9
      ansible_network_os: routeros
    CE3:
      ansible_host: 10.10.10.11
      ansible_network_os: routeros

```

Al final del archivo de los hosts, se establecen variables de conexión en común para todos los dispositivos de red, como indica la Figura 96, los cuales son aplicadas a los equipos que pertenecen a los grupos de routers de proveedores, bordes y clientes. Estas variables en común son:

- `ansible_connection`: El tipo de conexión desde Ansible hacia los dispositivos.
- `ansible_user`: nombre del usuario privilegiado creado previamente en el dispositivo para establecer el inicio de sesión SSH.
- `ansible_password`: contraseña del usuario para la conexión.

Figura 96*Definición de variables en archivo hosts del escenario 3*

```

GNU nano 4.8                               hosts.yml
vars:
  ansible_connection: network_cli
  ansible_user: ansible
  ansible_password: ansible

```

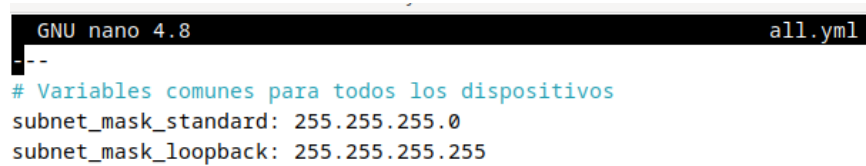
Después, se define las variables para los dispositivos listados en el inventario de hosts, las cuales, están organizados por archivos separados según el grupo de equipos al que pertenece, ya sea proveedores, bordes y clientes.

En el caso de la Figura 97, indica variables comunes para todos los dispositivos del grupo 'all', como la máscara de subred para las interfaces físicas (`subnet_mask_standard`) y la máscara de subred para interfaz loopback (`subnet_mask_loopback`), las cuales están

almacenadas en el archivo `all.yml`, esto evita repetir dichos valores en todas las interfaces definidas en los grupos.

Figura 97

Variables en común de los dispositivos para escenario 3



```
GNU nano 4.8 all.yml
--
# Variables comunes para todos los dispositivos
subnet_mask_standard: 255.255.255.0
subnet_mask_loopback: 255.255.255.255
```

Por otro lado, los archivos de la carpeta `'group_vars'`, definen las variables para cada grupo de dispositivos, las cuales permitirán detallar las interfaces con su respectivas direcciones IP. Por tanto, las variables usadas para la estructura de datos de los archivos son:

- `router_type`: nombre personalizado para identificación del tipo de grupo de dispositivos.
- `devices`: variable personalizada que permite enlistar los dispositivos para cada grupo.
- `interfaces`: variable personalizada que contiene la lista de las interfaces del dispositivo.

Ahora, se especifica el contenido de los archivos de las variables, para el caso de la Figura 98, corresponde al archivo `'proveedores.yml'`, que organiza al grupo de routers designados como proveedores o núcleo de la red MPLS, el cual se identifica con el nombre `'provider'` y enlistar a los dispositivos pertenecientes a dicho grupo para detallar sus interfaces con su respectivo nombre, dirección IP y la máscara de subred que se obtendrán reutilizando las variables `subnet_mask_standard` y `subnet_mask_loopback`, definida en el archivo `'all.yml'`.

Figura 98*Variables para definir interfaces en Routers Proveedor-P1*

```

GNU nano 4.8 proveedores.yml
--
router_type: provider
devices:
  P1:
    interfaces:
      - name: f0/1
        ip: 172.16.0.1
        mask: "{{ subnet_mask_standard }}"
      - name: f1/0
        ip: 172.16.3.1
        mask: "{{ subnet_mask_standard }}"
      - name: f1/1
        ip: 172.16.4.1
        mask: "{{ subnet_mask_standard }}"
      - name: f0/0
        ip: 10.10.10.1
        mask: "{{ subnet_mask_standard }}"
      - name: Lo0
        ip: 1.1.1.1
        mask: "{{ subnet_mask_loopback }}"

```

De manera similar se realiza la definición de las variables para enlistar los dispositivos del grupo, con sus interfaces correspondientes para los archivos de ‘bordes.yml’ y ‘clientes.yml’.

Para el grupo de dispositivos que pertenecen al archivo proveedores.yml, corresponden a los router cisco con la función de proveedor o núcleo de la red MPLS, por tanto, se termina de detallar las interfaces del dispositivo P2 con su dirección IP y mascara de subred, en la Figura 99. Así mismo, se evidencia la especificación de las interfaces para el router proveedor (P3) en la Figura 100 y para el router proveedor (P4) en la Figura 101.

Figura 99*Variables para definir interfaces en Routers Proveedor-P2*

```

GNU nano 4.8 proveedores.yml
P2:
  interfaces:
    - name: f0/1
      ip: 172.16.0.2
      mask: "{{ subnet_mask_standard }}"
    - name: f1/0
      ip: 172.16.1.2
      mask: "{{ subnet_mask_standard }}"
    - name: f1/1
      ip: 172.16.5.2
      mask: "{{ subnet_mask_standard }}"
    - name: f2/0
      ip: 172.16.8.2
      mask: "{{ subnet_mask_standard }}"
    - name: f0/0
      ip: 10.10.10.2
      mask: "{{ subnet_mask_standard }}"
    - name: Lo0
      ip: 2.2.2.2
      mask: "{{ subnet_mask_loopback }}"

```

Figura 100*Variables para definir interfaces en Routers Proveedor-P3*

```

GNU nano 4.8 proveedores.yml
P3:
  interfaces:
    - name: f0/0
      ip: 172.16.3.3
      mask: "{{ subnet_mask_standard }}"
    - name: f0/1
      ip: 172.16.2.3
      mask: "{{ subnet_mask_standard }}"
    - name: f1/0
      ip: 172.16.5.3
      mask: "{{ subnet_mask_standard }}"
    - name: f1/1
      ip: 172.16.6.3
      mask: "{{ subnet_mask_standard }}"
    - name: f2/0
      ip: 10.10.10.3
      mask: "{{ subnet_mask_standard }}"
    - name: Lo0
      ip: 3.3.3.3
      mask: "{{ subnet_mask_loopback }}"

```

Figura 101*Variables para definir interfaces en Routers Proveedor-P4*

```

P4:
  interfaces:
    - name: f0/0
      ip: 172.16.2.4
      mask: "{{ subnet_mask_standard }}"
    - name: f0/1
      ip: 172.16.1.4
      mask: "{{ subnet_mask_standard }}"
    - name: f1/0
      ip: 172.16.4.4
      mask: "{{ subnet_mask_standard }}"
    - name: f1/1
      ip: 172.16.10.4
      mask: "{{ subnet_mask_standard }}"
    - name: f2/0
      ip: 10.10.10.4
      mask: "{{ subnet_mask_standard }}"
    - name: Lo0
      ip: 4.4.4.4
      mask: "{{ subnet_mask_loopback }}"

```

El siguiente archivo para definir las variables es ‘bordes.yml’, el cual sigue la misma estructura de datos con el grupo identificado como ‘edge’ para routers de borde en una red MPLS. Además, se define las variables para definir las interfaces del router de borde (PE1), como indica la Figura 102.

También, se incluye la variable `borde_interfaces`: que permite guardar una lista de interfaces que se conecta desde el PE hacia el router cliente CE, ya que se puede utilizar posteriormente en configuraciones del playbook para aplicar políticas QoS.

Figura 102*Variables para definir interfaces en Routers Bordes-PE1*

```

GNU nano 4.8                               inventory/group_vars/bordes.yml
---
router_type: edge
devices:
  PE1:
    interfaces:
      - name: f0/0
        ip: 172.16.7.5
        mask: "{{ subnet_mask_standard }}"
      - name: f0/1
        ip: 172.16.6.5
        mask: "{{ subnet_mask_standard }}"
      - name: f2/0
        ip: 10.10.10.5
        mask: "{{ subnet_mask_standard }}"
      - name: Lo0
        ip: 5.5.5.5
        mask: "{{ subnet_mask_loopback }}"
    borde_interfaces:
      - fastEthernet0/0

```

Usando la misma estructura de las variables, se muestra en la Figura 103, la definición de las interfaces para los routers de borde PE2 y PE3.

Figura 103*Variables para definir interfaces en Routers Bordes-PE2 y PE3*

```

GNU nano 4.8                               inventory/group_vars/bordes.yml
PE2:
  interfaces:
    - name: f1/0
      ip: 172.16.8.6
      mask: "{{ subnet_mask_standard }}"
    - name: f0/1
      ip: 172.16.9.6
      mask: "{{ subnet_mask_standard }}"
    - name: f0/0
      ip: 10.10.10.6
      mask: "{{ subnet_mask_standard }}"
    - name: Lo0
      ip: 6.6.6.6
      mask: "{{ subnet_mask_loopback }}"
  borde_interfaces:
    - fastEthernet0/1
PE3:
  interfaces:
    - name: f0/0
      ip: 172.16.10.7
      mask: "{{ subnet_mask_standard }}"
    - name: f0/1
      ip: 172.16.11.7
      mask: "{{ subnet_mask_standard }}"
    - name: f2/0
      ip: 10.10.10.7
      mask: "{{ subnet_mask_standard }}"
    - name: Lo0
      ip: 7.7.7.7
      mask: "{{ subnet_mask_loopback }}"
  borde_interfaces:
    - fastEthernet0/1

```

Por último, el archivo ‘clientes.yml’ contiene los routers Mikrotik que tienen la función de clientes (CE) en la red MPLS. De igual forma se especifica el nombre ‘customer’ para la variable ‘router_type’ como muestra la Figura 104, y se procede a enlistar los dispositivos, además se detalla las interfaces del router cliente (CE1).

También, se indica las interfaces para el router cliente 2 (CE2) y router cliente 3 (CE3) se detallan en la Figura 105.

Figura 104

Variables para definir interfaces en Routers Clientes-CE1

```
GNU nano 4.8 clientes.yml
---
router_type: customer
devices:
  CE1:
    interfaces:
      - name: ether1
        ip: 172.16.7.8
        mask: "{{ subnet_mask_standard }}"
      - name: ether2
        ip: 192.168.10.1
        mask: "{{ subnet_mask_standard }}"
      - name: ether3
        ip: 10.10.10.8
        mask: "{{ subnet_mask_standard }}"
      - name: lo
        ip: 8.8.8.8
        mask: "{{ subnet_mask_loopback }}"
```

Figura 105

Variables para definir interfaces en Routers Clientes-CE2 y CE3

```
GNU nano 4.8 clientes.yml
CE2:
  interfaces:
    - name: ether1
      ip: 172.16.9.9
      mask: "{{ subnet_mask_standard }}"
    - name: ether2
      ip: 192.168.11.1
      mask: "{{ subnet_mask_standard }}"
    - name: ether3
      ip: 10.10.10.9
      mask: "{{ subnet_mask_standard }}"
    - name: lo
      ip: 9.9.9.9
      mask: "{{ subnet_mask_loopback }}"
CE3:
  interfaces:
    - name: ether1
      ip: 172.16.11.10
      mask: "{{ subnet_mask_standard }}"
    - name: ether2
      ip: 192.168.12.1
      mask: "{{ subnet_mask_standard }}"
    - name: ether3
      ip: 10.10.10.11
      mask: "{{ subnet_mask_standard }}"
    - name: lo
      ip: 10.10.10.10
      mask: "{{ subnet_mask_loopback }}"
```

Una vez establecido los hosts y sus variables, se procede a detallar la configuración de cada playbook en el orden que serán ejecutados para el funcionamiento del escenario 3. Además, cada playbook está constituido por un único bloque de configuraciones es decir, un solo PLAY que aloja las tareas (TASKS) para la automatización.

Por tanto, el primer playbook a desarrollar es la configuración del direccionamiento IPv4, llamado ‘configure_ipv4.yml’, el cual sigue una estructura jerárquica que organiza los datos, utilizando instrucciones para las configuraciones, las cuales se presentan en la Tabla 33.

Tabla 33

Instrucciones del playbook para configuración IPv4 del escenario 3

Instrucción	Descripción
when	Condición que evalúa una expresión.
block	Permite, agrupar varias tareas para ejecutarlas en conjunto bajo una condición.
cisco.ios.ios_config	Módulo ios_config que pertenece a la colección de cisco.ios para ejecutar las líneas de las tareas en equipos Cisco.
community.routeros.command	Módulo para ejecutar comandos en equipos Mikrotik.
commands	Lista comandos para dispositivos Mikrotik.
parents	Define donde ejecutar las líneas de comandos en el CLI.
loop	Repite la tarea para configurar cada elemento de una lista.
register	Guarda en una variable el resultado de la configuración de una tarea.

El contenido del playbook ‘configure_ipv4.yml’ se compone de un PLAY llamado *Configure IPv4 addressing on network devices* el cual, está destinado a todos los dispositivos listados en el grupo principal ‘all’ en el inventario ‘hosts.yml’, además dentro de este bloque se consideran dos TASKS.

Una de las tareas se muestra en la Figura 106, denominada como *Configurar direcciones ipv4 para equipos Cisco*, posteriormente se coloca el condicional ‘when’ para definir el sistema operativo de los equipos y poder aplicar las tareas agrupadas en ‘block’, las cuales son, asignar la dirección IP con su respectiva máscara en cada interfaz del dispositivo

mediante un ‘loop’ ya que permite recorrer la lista de interfaces del dispositivo que está procesando Ansible para aplicar la configuración automáticamente.

Figura 106

Configuración de IPv4 en equipos Cisco en escenario 3

```
GNU nano 4.8                                configure_ipv4.yml
- name: Configure IPv4 addressing on network devices
  hosts: all
  gather_facts: no

  tasks:
    ## Para todos lo equipos cisco LSR, LER

    - name: Configurar direcciones ipv4 para equipos Cisco

      when: ansible_network_os == "ios"
      block:
        - name: Configurar direcciones IPv4 en interfaces
          cisco.ios.ios_config:
            lines:
              - "ip address {{ item.ip }} {{ item.mask }}"
              - no shutdown
            parents: "interface {{ item.name }}"
            loop: "{{ devices[inventory_hostname]['interfaces'] }}"
            register: config_result
```

La siguiente tarea del playbook ‘configure_ipv4.yml’, se define como *Configurar direcciones ipv4 para equipos Mikrotik*, la cual, contiene subtareas para agregar una dirección IP con su respectiva máscara de subred ya sea de /32 para interfaz que tenga el nombre ‘lo’ (Loopback), o caso contrario serán de /24 para las interfaces físicas, las cuales, serán asignadas para cada interfaz listada de los dispositivos en el archivo ‘hosts.yml’, además, realiza una backup de la tarea en caso de que si haya producido cambios en la misma, como se muestra en la Figura 107.

Figura 107

Configuración de IPv4 en equipos Mikrotik en escenario 3

```
GNU nano 4.8                                configure_ipv4.yml                                Modified
## Para todos los equipos de acceso MIKROTIK

- name: Configurar direcciones ipv4 para equipos Mikrotik
  when: ansible_network_os == "routeros"
  block:
    - name: Configurar direcciones ipv4 en interfaces
      community.routeros.command:
        commands:
          - /ip address add address={{ item.ip }}/{{ '32' if item.name == 'lo' else '24' }} interface={{ item.name }}
      loop: "{{ devices[inventory_hostname]['interfaces'] }}"
      register: config_result

    - name: Guardar configuraciones en Mikrotik
      community.routeros.command:
        commands:
          - /system backup save name=ipv4-config
      when: config_result.changed
```

El siguiente playbook a definir, corresponde al archivo ‘configure_ospf.yml’, destinado a realizar la configuración del enrutamiento OSPF, usando una serie de instrucciones mostradas en la Tabla 34, sin embargo se listan solo las instrucciones nuevas, ya que las demás se ha abordado en anteriores playbooks, las cuales ya fueron explicadas.

Tabla 34

Instrucciones del playbook para configuración OSPF del escenario 3

Instrucción	Descripción
set_fact	Módulo de Ansible que permite actualizar o definir de manera dinámica, el valor de una variable.
selectattr	Selecciona el elemento de una lista que cumpla con ciertos criterios especificados por el usuario.
map	Extrae atributos o valores de una lista previamente definida.
rejectattr	Excluye el elemento de una lista que cumple con los criterios definidos.
regex_replace	Reemplaza los valores de una cadena de texto.

El playbook ‘configure_ospf’ inicia con el nombre descriptivo del PLAY llamado, *Configurar OSPF en todos los dispositivos*, el cual, contiene las tareas para aplicarlas a todos los dispositivos listados en el grupo ‘all’, especificados en el archivo ‘hosts.yml’, como indica la Figura 108. Además se añade tres variables en común dentro del playbook que son:

- **ospf_process_id:** corresponde al identificador (ID) del proceso OSPF, el cual se define el valor 1 para todos los dispositivos

- **ospf_area:** se especifica el área de OSPF para los dispositivos, configurada con la dirección 0.0.0.0, la cual corresponde al área 0, es decir el área principal.
- **netmask_to_cidr:** es una variable que almacena un diccionario para convertir máscaras de subred a una notación CIDR (enrutamiento entre dominios sin clases), entonces este directorio está compuesto por pares de clave: valor, donde la clave es la máscara de subred en formato decimal y el valor es el prefijo equivalente a la máscara.

Figura 108

Establecimiento del bloque PLAY para configuración OSPF en escenario 3

```
GNU nano 4.8                                configure_ospf.yml
- name: Configurar OSPF en todos los dispositivos
  hosts: all
  gather_facts: no
  vars:
    ospf_process_id: 1
    ospf_area: 0.0.0.0
    # Diccionario para convertir mascarar de subred a CIDR
    netmask_to_cidr:
      "255.255.255.255": "32"
      "255.255.255.254": "31"
      "255.255.255.252": "30"
      "255.255.255.248": "29"
      "255.255.255.240": "28"
      "255.255.255.224": "27"
      "255.255.255.192": "26"
      "255.255.255.128": "25"
      "255.255.255.0": "24"
      "255.255.254.0": "23"
      "255.255.252.0": "22"
      "255.255.248.0": "21"
      "255.255.240.0": "20"
```

En el playbook ‘configure_ospf’ contiene dos tareas principales dentro del bloque (PLAY), que corresponden a la configuración de OSPF en dispositivos Cisco y la configuración de OSPF en dispositivos Mikrotik.

La primera tarea, se ilustra en la Figura 109, con el nombre *Configurar OSPF en equipos Cisco*, la cual, agrupa subtareas que serán ejecutadas únicamente si el sistema operativo del equipo es Cisco, que son las siguientes:

- **Crear sentencias de red OSPF para Cisco:** tiene la finalidad de definir las redes que van a ser parte del proceso OSPF, por tanto se debe asociar la

dirección IP con su wildcard. Para esta subtarea se usa la función de Ansible ‘set_fact’ para actualizar el valor de manera dinámica de la variable ‘ospf_networks’, la cual almacena la lista de redes que se van formando en cada iteración de las interfaces del dispositivo.

La lista de las redes que se almacenan en ‘ospf_networks’ se inicia vacía, es decir en default ([]), donde se va a ir añadiendo valores a la lista que está compuesto por una dirección IP concatenada a su respectiva wildcard, la cual será calculada en base a la máscara de subred, usando el filtro ‘regex_replace’ ya que, captura los 4 octetos definidos en la máscara de subred (ejemplo: 255.255.255.0) y aplica una resta a cada uno (255- octeto) para obtener el valor de la wildcard resultante. Además, se garantiza que la tarea se ejecute solo si las variables de IP y mask están definidas en la interfaz.

- **Obtener el ID del enrutador desde la Loopback:** se establece esta subtarea para identificar al router de manera única en el enrutamiento. Para ello, se define la variable ‘router_id’, la cual irá actualizando sus valores que almacena en su lista, correspondiente a la Loopback del dispositivo, para ello se usa el filtro ‘selectattr’ que permite seleccionar el elemento de la lista que sea igual a ‘Lo0’ y extraer el primer valor encontrado que es la IP de la interfaz Loopback, en caso de no existir dicho elemento devuelve el valor 0.0.0.0.
- **Configurar el proceso OSPF en Cisco:** tiene el propósito de designar el proceso OSPF a los dispositivos, reutilizando el valor de la variable ‘ospf_process_id’.
- **Configure el ID del router en contexto OSPF:** tiene la finalidad de asignar el identificador único, usando los valores de la variable ‘router_id’ que almacena

las IP de la Loopback, sobre el proceso OSPF definido en la variable ‘ospf_process_id’, con la finalidad de identificar al router en la red OSPF.

- **Aplicar las sentencias de red OSPF en Cisco:** esta subtarea se configura dentro del proceso OSPF, para asociar cada red OSPF almacenada en ‘ospf_networks’ con el área especificada en la variable ‘ospf_area’, para que cada router anuncie sus redes disponibles, correspondientes a sus interfaces físicas.

Figura 109

Configuración OSPF para dispositivos Cisco en escenario 3

```
GNU nano 4.8                                configure_ospf.yml
tasks:
  - name: Configurar OSPF en equipos Cisco
    when: ansible_network_os == "ios"
    block:
      - name: Crear sentencias de red OSPF para Cisco
        set_fact:
          ospf_networks: "{{ ospf_networks | default([]) + [ item.ip ~ ' ' ~ item.mask |
regex_replace('^(\d+)\.(\d+)\.(\d+)\.(\d+)$', '{ 255 - \1 }.{ 255 - \2 }.{ 255 - \3 }.{ 255 - \4 }') ] }}"
        loop: "{{ devices[inventory_hostname]['interfaces'] }}"
        when: item.ip is defined and item.mask is defined

      - name: Obtener el ID del enrutador desde la Loopback
        set_fact:
          router_id: "{{ devices[inventory_hostname]['interfaces'] | selectattr('name', 'equalto', 'Lo0') | map(attribute='ip')
| first | default('0.0.0.0') }}"

      - name: Configurar el proceso OSPF en Cisco
        cisco.ios.ios_config:
          lines:
            - "router ospf {{ ospf_process_id }}"

      - name: Configurar el ID del router en OSPF
        cisco.ios.ios_config:
          lines:
            - "router-id {{ router_id }}"
          parents:
            - "router ospf {{ ospf_process_id }}"

      - name: Aplicar las sentencias de red OSPF en Cisco
        cisco.ios.ios_config:
          lines: "network {{ item }} area {{ ospf_area }}"
          parents: "router ospf {{ ospf_process_id }}"
          loop: "{{ ospf_networks }}"
          when: ospf_networks is defined
```

La siguiente tarea principal del playbook ‘configure_ospf’ corresponde a *Configurar OSPF en equipos Mikrotik*, como se muestra en la Figura 110, la cual se enfoca solo a dispositivos con sistema operativo routers que corresponde a dispositivos Mikrotik, por tanto las siguientes subtarear se ejecutarán bajo dicha condición:

- **Habilitación de instancias OSPF:** hace referencia a la creación y habilitación de la instancia OSPF denominada ‘default’, donde se establece el identificador

(router-id) de cada dispositivo a partir de la interfaz Loopback que es extraída su IP mediante el filtro ‘map’, luego se lo asocia con el área OSPF creada con el nombre ‘backbone’ utilizando el valor de la variable ‘ospf_area’, con el fin de identificar al router de forma única.

- **Anunciar la red OSPF:** esta subtarea permite al router anunciar sus redes, ya que se añade una red por cada interfaz física para que OSPF funcione, excepto la interfaz Loopback que se excluye con el filtro ‘rejectattr’. Dichas redes se asocian al área backbone, las cuales son calculadas usando la IP de cada interfaz y su máscara de subred, donde será sustituida por el prefijo CIDR situado en el diccionario netmask_to_cidr, en caso de no estar en el diccionario se usará por defecto /24.

Figura 110

Configuración OSPF para dispositivos Mikrotik en escenario 3

```
GNU nano 4.8                                configure_ospf.yml
- name: Configurar OSPF en equipos Mikrotik
  when: ansible_network_os == "routeros"
  block:
    - name: Habilitacion de instancias OSPF
      community.routeros.command:
        commands:
          - "/routing/ospf instance add name=default router-id={{ devices[inventory_hostname]['interfaces'] |
selectattr('name', 'equalto', 'lo') | map(attribute='ip') | first | default('0.0.0.0') }} disabled=no"
          - "/routing ospf area add instance=default name=backbone area-id={{ ospf_area }}"

    - name: Anunciar la red OSPF
      community.routeros.command:
        commands: "routing/ospf/interface-template/ add area=backbone
networks={{ item.ip }}/{{ netmask_to_cidr[item.mask] | default('24') }}"
      loop: "{{ devices[inventory_hostname]['interfaces'] | rejectattr('name', 'equalto', 'lo') | list }}"
      when: item.ip is defined and item.mask is defined
```

Luego, se define el siguiente playbook llamado ‘configure_mpls.yml’, el cual contiene las tareas de automatización para aplicar configuraciones de MPLS, a continuación se presenta nuevas instrucciones usadas en este playbook, en la Tabla 35.

Tabla 35

Instrucciones del playbook para configuración MPLS del escenario 3

Instrucción	Descripción
regex_search	Busca coincidencia de una cadena de texto con una expresión regular.

debug:	Imprime durante la ejecución del playbook datos o resultados de comandos de verificación en las tareas.
var	Permite mostrar el valor de una variable

En el playbook ‘configure_mpls.yml’ se denomina a su principal PLAY como *Configurar MPLS en equipos Cisco*, tal como se muestra en la Figura 111, el cual, contiene una única tarea principal destinada los dispositivos listados en el grupo ‘all’. Sin embargo, la tarea principal está condicionada a aquellos dispositivos con sistema operativo ‘IOS’, ya que son los que pertenecen a la red MPLS, en caso de cumplir dicha condición se ejecutará secuencialmente las subtareas agrupadas en el bloque ‘block’, las cuales se detallan a continuación para el proceso de configuración MPLS:

- **Habilitación global de IP CEF:** la habilitación global de IP CEF (Cisco Express Forwarding), por tanto en todos los dispositivos que pertenecen a la Nube MPLS, se activa la función ‘ip cef’, el cual es un requerimiento previo para el funcionamiento de MPLS.
- **Habilitación global de MPLS IP:** esta subtask habilita MPLS IP mediante el comando ‘mpls ip’ a todos los router pertenecientes a la Nube MPLS, para procesar el tráfico MPLS.
- **Configurar ID en routers de nube MPLS:** permite configurar el identificador único (ID) en cada router cisco, usando la IP de la interfaz virtual Loopback, para ser empleado en sesiones LDP para intercambiar etiquetas entre los routers vecinos de la red MPLS, con la finalidad de identificarse al momento de intercambiar etiquetas.

En caso de tener una sesión LDP activa, con el parámetro ‘force’ obliga a cambiar el router ID si es necesario, además esta subtask se ejecutará bajo la condición ‘when’, indicando que debe existir una interfaz llamada ‘Lo0’ dentro

de la lista generada de las interfaces, con el fin de evitar errores en caso de no tener definida la interfaz Loopback.

Figura 111

Configuración de MPLS para dispositivos Cisco en escenario 3

```

GNU nano 4.8                                configure_mpls.yml                                Modified
- name: Configurar MPLS en equipos Cisco
  hosts: all
  gather_facts: no
  tasks:
    - name: Configurar MPLS en equipos Cisco
      when: ansible_network_os == "ios"
      block:
        - name: Habilitacion global de IP CEF (prerequisito para MPLS)
          cisco.ios.ios_config:
            lines:
              - ip cef

        - name: Habilitacion global de MPLS IP
          cisco.ios.ios_config:
            lines:
              - mpls ip

        - name: Configurar ID en routers de nube MPLS
          cisco.ios.ios_config:
            lines:
              - mpls ldp router-id Loopback0 force
          when:
            - devices[inventory_hostname]['interfaces'] | selectattr('name', 'equalto', 'Lo0') | list | length > 0

```

Seguidamente, en la Figura 112 se indica las subtareas restantes para completar la configuración de MPLS, las cuales se detallan a continuación:

- **Habilitar MPLS en interfaces físicas:** permite habilitar MPLS y configurar el protocolo LDP en las interfaces físicas que forman parte de la nube MPLS de cada router, para que participen en el intercambio de etiquetas. Sin embargo, esta subtarea se aplica bajo condiciones, ya que se selecciona las interfaces que tengan definido su nombre y una IP asignada, pero excluyendo las interfaces que tengan el nombre 'Lo0' (Loopback) y las de gestión que empiecen con la expresión regular 'Mgmt|Management|Vlan1'.
- **Verificación de configuración MPLS:** valida la configuración MPLS, ejecutando los comandos 'show mpls interfaces' y 'show mpls ldp neighbor' en todos los dispositivos, para verificar las interfaces que se encuentra habilitado MPLS y las sesiones LDP con los vecinos. **c**

Además, se usa ‘ignore_errors’ para evitar que la ejecución del playbook se detenga por errores de verificación, ejemplo cuando MPLS no está habilitado o no hay sesiones LDP. Entonces, se usa dicha instrucción ya que son comandos unicamente de verificación y no afecta a la configuración del equipo.

- **Imprimir resultados de configuración MPLS:** imprime los resultados de la verificación de MPLS, en forma de lista (stdout_lines) que están almacenados en la variable ‘mpls_verification’. Su ejecución está condicionada a que dicha variable este correctamente definida, es decir que contenga datos.

Figura 112

Subtareas para configuración de MPLS en escenario 3

```
GNU nano 4.8      configure_mpls.yml
- name: Habilitar MPLS en interfaces fisicas (skip loopback and management interfaces)
  cisco.ios.ios_config:
    lines:
      - mpls ip
      - mpls label protocol ldp
    parents: "interface {{ item.name }}"
    loop: "{{ devices[inventory_hostname]['interfaces'] }}"
    when:
      - item.name is defined
      - item.name != "Lo0"
      - not item.name | regex_search('^(Mgmt|Management|Vlan1)')
      - item.ip is defined # Solo configurar MPLS en interfaces con IP asignada

- name: Verificacion de configuracion MPLS
  cisco.ios.ios_command:
    commands:
      - show mpls interfaces
      - show mpls ldp neighbor
  register: mpls_verification
  ignore_errors: yes

- name: Imprimir resultados de configuracion MPLS
  debug:
    var: mpls_verification.stdout_lines
  when: mpls_verification is defined and mpls_verification.stdout_lines is defined
```

El siguiente paso, es desarrollar el playbook ‘configure_qos.yml’, para automatizar la configuración de políticas de calidad de servicio (QoS), específicamente en los routers de borde. Este playbook está compuesto por un solo PLAY, llamado *Aplicar políticas de QoS a routers en ambas direcciones*, que agrupa un conjunto de tareas, orientadas a la clasificación, marcaje y priorización del tráfico. Al inicio del archivo de configuración se define variables

para establecer los nombres de las políticas QoS tanto de entrada como de salida, y de las ACLs para clasificar el tráfico, como se indica en la Figura 113.

Esto permite establecer en el fragmento mostrado del playbook ‘configure_qos’ las tareas para la clasificación de tráfico, mediante la configuración de ACLs extendidas, para identificar los diferentes tipos de servicios a partir de sus puertos TCP, tales como: tráfico web que emplea puertos 80 para HTTP y 443 para HTTPS; FTP que utiliza el puerto 21; y el tráfico de base de datos (DB) se considera el puerto 3306.

Dichas ACLs clasifican las solicitudes que envía los clientes hacia el servidor, por ello, la estructura de la regla (ACL) indica que permite el tráfico TCP desde cualquier origen (any) hacia cualquier destino (any), especificando el puerto destino correspondiente al tipo de servicio.

Figura 113

Configuración de listas de control de acceso (ACL) en escenario 3

```
GNU nano 4.8                                configure_qos.yml
---
- name: Aplicar politicas QoS en ambas direcciones
  hosts: bordes
  connection: network_cli
  gather_facts: no
  vars:
    input_policy_name: QoS-INPUT-POLICY
    output_policy_name: QoS-OUTPUT-POLICY
    acl_web: WEBACL
    acl_ftp: FTPACL
    acl_db: DBACL

  tasks:

    # Definir ACLs

    - name: Configurar ACL para trafico web
      ios_config:
        parents: ["ip access-list extended {{ acl_web }}"]
        lines:
          - permit tcp any any eq 80
          - permit tcp any eq 80 any
          - permit tcp any any eq 443
          - permit tcp any eq 443 any

    - name: Configurar ACL para trafico FTP
      ios_config:
        parents: ["ip access-list extended {{ acl_ftp }}"]
        lines:
          - permit tcp any any eq 21
          - permit tcp any eq 21 any

    - name: Configurar ACL para trafico DB
      ios_config:
        parents: ["ip access-list extended {{ acl_db }}"]
        lines:
          - permit tcp any any eq 3306
          - permit tcp any eq 3306 any
```

Las siguientes tareas definidas, se muestran en el fragmento extraído del playbook ‘configure_qos’ en la Figura 114, que tratan sobre la creación de class-map para agrupar el tráfico previamente clasificado en las ACLs para cada tipo de servicio . Por tanto se establece estas clases de tráfico diferenciadas para: servicio web, ftp y base de datos (BD), con el objetivo de distinguir su tráfico de cada servicio y poder aplicar posteriormente las políticas de QoS.

Figura 114

Creación de class-map para cada tráfico en escenario 3

```
GNU nano 4.8                                configure_qos.yml
#Definir class-maps de la clasificacion de trafico

- name: Crear class-map para trafico web
  ios_config:
    parents: ["class-map match-any WEB-TRAFFIC"]
    lines:
      - match access-group name {{ acl_web }}

- name: Crear class-map para trafico FTP
  ios_config:
    parents: ["class-map match-any FTP-TRAFFIC"]
    lines:
      - match access-group name {{ acl_ftp }}

- name: Crear class-map para trafico DB
  ios_config:
    parents: ["class-map match-any DB-TRAFFIC"]
    lines:
      - match access-group name {{ acl_db }}
```

Seguidamente, en la Figura 115, se muestra un fragmento del playbook ‘configure_qos’ sobre la configuración de policy-map, para definir la política de QoS para la salida del tráfico, mediante la asignación de porcentajes de ancho de banda a cada clase de tráfico.

En primer lugar, se crea el ‘policy-map’ con el nombre extraído de la variable ‘output_policy_name’ para luego configurar los recursos asignados para cada clase de tráfico, detallados a continuación:

- WEB-TRAFFIC: a esta clase de tráfico se le asigna un porcentaje de prioridad del 30% del ancho de banda, por tanto, su tráfico será atendido primero en situaciones de congestión de la red.

- FTP-TRAFFIC: para la clase de tráfico FTP, se garantiza un porcentaje mínimo del 20% de ancho de banda, por ende en caso de congestión se garantiza un rendimiento óptimo.
- DB-TRAFFIC: para el tráfico que pertenece a la clase DB (base de datos), se asigna el 20% como porcentaje mínimo de ancho de banda que debe estar disponible para esta clase.
- class-default: esta clase es para el tráfico que no se encuentra clasificado en ninguna de las clases anteriores, el cual, se configura como política 'fair-queue' para gestionar este tráfico restante de manera equitativa, evitando que ocupe todo el ancho de banda de la red.

Figura 115

Configuración de la política QoS de salida en escenario 3

```
GNU nano 4.8                                configure_qos.yml
# Policy-map OUTPUT

- name: Crear policy-map de salida
  ios_config:
    lines:
      - policy-map {{ output_policy_name }}

- name: Configurar clases en policy-map de salida
  ios_config:
    parents: ["policy-map {{ output_policy_name }}", "class WEB-TRAFFIC"]
    lines:
      - priority percent 30

- name: Clase FTP salida
  ios_config:
    parents: ["policy-map {{ output_policy_name }}", "class FTP-TRAFFIC"]
    lines:
      - bandwidth percent 20

- name: Clase DB salida
  ios_config:
    parents: ["policy-map {{ output_policy_name }}", "class DB-TRAFFIC"]
    lines:
      - bandwidth percent 20

- name: Clase default salida
  ios_config:
    parents: ["policy-map {{ output_policy_name }}", "class class-default"]
    lines:
      - fair-queue
```

El siguiente fragmento del playbook ‘configure_qos’ mostrado en la Figura 116, indica las tareas para la configuración de la política QoS del tráfico de entrada, mediante el marcaje de paquetes con valores DSCP (Punto de Código de Servicios Diferenciados) para cada clase de tráfico.

Por tanto, se inicia creando la ‘policy-map’ con el nombre definido en la variable ‘input_policy_name’, para luego asociarles a cada clase de tráfico y aplicar para los paquetes de entrada los valores DSCP, los cuales definen el nivel de prioridad y la probabilidad de descarte (dropeo) en condiciones de congestión. Además, los valores DSCP permite identificar el tráfico a lo largo de la red, a través del fomado AFxy, donde ‘x’ representa la clase de prioridad tomando un valor del 1 al 4 (más alto el número, mayor prioridad) mientras que ‘y’ es el nivel de descarte tomando valores del 1 al 3 (cuanto mayor es el número, mayor posibilidad de dropeo).

Las clases previamente definidas se les asigna los siguientes valores DSCP:

- WEB-TRAFFIC: para paquetes de tráfico web se asigna un dscp af31, otorgándole un nivel de prioridad alta.
- FTP-TRAFFIC: para la clase que agrupa el tráfico FTP se le asigna una prioridad baja con dscp af11.
- DB-TRAFFIC: se establece el valor dscp af21 para el tráfico de base de datos, el cual, tiene una prioridad media y con baja posibilidad de descarte.
 - class-default: en esta clase se aplica el valor ‘default’ para el tráfico que no pertenece a ninguna de las clases anteriores, por tanto no tendrá un tratamiento especial.

Figura 116*Configuración de la política QoS de entrada en escenario 3*

```

GNU nano 4.8                                configure_qos.yml
# Policy-map INPUT

- name: Crear policy-map de entrada
  ios_config:
    lines:
      - policy-map {{ input_policy_name }}

- name: Clase WEB en entrada
  ios_config:
    parents: ["policy-map {{ input_policy_name }}", "class WEB-TRAFFIC"]
    lines:
      - set dscp af31

- name: Clase FTP entrada
  ios_config:
    parents: ["policy-map {{ input_policy_name }}", "class FTP-TRAFFIC"]
    lines:
      - set dscp af11

- name: Clase DB entrada
  ios_config:
    parents: ["policy-map {{ input_policy_name }}", "class DB-TRAFFIC"]
    lines:
      - set dscp af21

- name: Clase default entrada
  ios_config:
    parents: ["policy-map {{ input_policy_name }}", "class class-default"]
    lines:
      - set dscp default

```

Como ultima tarea del playbook ‘configure_qos’ es aplicar tanto las políticas de entrada como de salida en las interfaces de cada router de borde, como se muestra en la Figura 117, por ello se usa la instrucción ‘loop’ que recorre las interfaces listadas en la variable ‘borde_interfaces’, permitiendo que reciban la configuración, con la finalidad que todo el tráfico que sale y entra de dichas interfaces sea tratado según las políticas QoS establecidas.

Figura 117*Aplicación de políticas QoS en escenario 3*

```

GNU nano 4.8                                configure_qos.yml
# Aplicar ambas politicas en las interfaces

- name: Aplicar ambas QoS policies (input/output) en interfaces de borde
  ios_config:
    parents: "interface {{ item }}"
    lines:
      - service-policy input {{ input_policy_name }}
      - service-policy output {{ output_policy_name }}
    loop: "{{ devices[inventory_hostname]['borde_interfaces'] }}"

```

- **Validación de configuraciones para escenario 3**

Para validar las configuraciones automatizadas en los dispositivos de red, se deben ejecutar los playbooks creados, mediante el comando `ansible-playbook` seguido del nombre del archivo de inventario y el nombre del playbook correspondiente, sin embargo, en este escenario se organizó los archivos por subcarpetas, como se hizo referencia en la Figura 93.

Por lo tanto, se debe incluir en el comando la ruta relativa hacia cada archivo, ya que, Ansible debe ubicar de forma correcta el archivo de los dispositivos (inventory) y los archivos de configuración (playbooks).

Además, al tener varios playbooks, estos deben ser ejecutados individualmente para aplicar las configuraciones definidas en las tareas de cada archivo de configuración, los cuales están enfocados en funciones específicas como el direccionamiento IPv4, protocolo de enrutamiento OSPF, habilitación de MPLS, configuración QoS y levantamiento servicios de red.

Sin embargo en la documentación, se presenta sola la parte inicial y final (PLAY RECAP) de la ejecución del playbook, ya que, debido a la alta cantidad de tareas, la salida que se genera en la consola sobre el estado de cada acción es extensa, pero es suficiente para validar el funcionamiento de la automatización de configuraciones.

En primer lugar, se ejecuta el playbook `'configure_ipv4.yml'` encargado de la asignación de direcciones IPv4 a las interfaces de red de cada dispositivo, ya que es fundamental para el funcionamiento posterior de OSPF y MPLS. Por lo tanto, la ejecución se realiza con el siguiente comando `ansible-playbook -i inventory/hosts playbook/configure_ipv4.yml`, permitiendo mostrar en pantalla la ejecución de las tareas por cada dispositivo tal como se muestra en la Figura 118.

Figura 118

Ejecución del playbook IPv4 referente al escenario 3

```
PLAY [Configure IPv4 addressing on network devices] *****

TASK [Configure IPv4 addresses on interfaces] *****
changed: [P1] => (item={'name': 'f0/1', 'ip': '172.16.0.1', 'mask': '255.255.255.0'})
changed: [PE1] => (item={'name': 'f0/0', 'ip': '172.16.7.5', 'mask': '255.255.255.0'})
changed: [P2] => (item={'name': 'f0/1', 'ip': '172.16.0.2', 'mask': '255.255.255.0'})
changed: [P3] => (item={'name': 'f0/0', 'ip': '172.16.3.3', 'mask': '255.255.255.0'})
changed: [P4] => (item={'name': 'f0/0', 'ip': '172.16.2.4', 'mask': '255.255.255.0'})
changed: [P1] => (item={'name': 'f1/0', 'ip': '172.16.3.1', 'mask': '255.255.255.0'})
changed: [PE1] => (item={'name': 'f0/1', 'ip': '172.16.6.5', 'mask': '255.255.255.0'})
changed: [P2] => (item={'name': 'f1/0', 'ip': '172.16.1.2', 'mask': '255.255.255.0'})
changed: [P3] => (item={'name': 'f0/1', 'ip': '172.16.2.3', 'mask': '255.255.255.0'})
changed: [P4] => (item={'name': 'f0/1', 'ip': '172.16.1.4', 'mask': '255.255.255.0'})
changed: [P1] => (item={'name': 'f1/1', 'ip': '172.16.4.1', 'mask': '255.255.255.0'})
changed: [PE1] => (item={'name': 'f2/0', 'ip': '10.10.10.5', 'mask': '255.255.255.0'})
changed: [P4] => (item={'name': 'f1/0', 'ip': '172.16.4.4', 'mask': '255.255.255.0'})
changed: [P2] => (item={'name': 'f1/1', 'ip': '172.16.5.2', 'mask': '255.255.255.0'})
changed: [P3] => (item={'name': 'f1/0', 'ip': '172.16.5.3', 'mask': '255.255.255.0'})
changed: [P1] => (item={'name': 'f0/0', 'ip': '10.10.10.1', 'mask': '255.255.255.0'})
changed: [P4] => (item={'name': 'f1/1', 'ip': '172.16.10.4', 'mask': '255.255.255.0'})
changed: [PE1] => (item={'name': 'Lo0', 'ip': '5.5.5.5', 'mask': '255.255.255.255'})
changed: [P2] => (item={'name': 'f2/0', 'ip': '172.16.8.2', 'mask': '255.255.255.0'})
changed: [P3] => (item={'name': 'f1/1', 'ip': '172.16.6.3', 'mask': '255.255.255.0'})
changed: [P4] => (item={'name': 'f2/0', 'ip': '10.10.10.4', 'mask': '255.255.255.0'})
changed: [P2] => (item={'name': 'f0/0', 'ip': '10.10.10.2', 'mask': '255.255.255.0'})
changed: [P1] => (item={'name': 'Lo0', 'ip': '1.1.1.1', 'mask': '255.255.255.255'})
changed: [P3] => (item={'name': 'f2/0', 'ip': '10.10.10.3', 'mask': '255.255.255.0'})
changed: [P4] => (item={'name': 'Lo0', 'ip': '4.4.4.4', 'mask': '255.255.255.255'})
changed: [P2] => (item={'name': 'Lo0', 'ip': '2.2.2.2', 'mask': '255.255.255.255'})
skipping: [CE1] => (item={'name': 'ether1', 'ip': '172.16.7.8', 'mask': '255.255.255.0'})
skipping: [CE1] => (item={'name': 'ether2', 'ip': '192.168.10.1', 'mask': '255.255.255.0'})
skipping: [CE1] => (item={'name': 'ether3', 'ip': '10.10.10.8', 'mask': '255.255.255.0'})
```

Al final de la ejecución del playbook ‘configure_IPv4.yml’, se presenta un resumen sobre el estado final de las tareas ejecutadas (PLAY RECAP) tal como se muestra en la Figura 119.

Por tanto, se verifica que todas las tareas fueron ejecutadas correctamente (OK) sin reporte de errores (failed=0) ni problemas de conexión (unreachable=0).

Figura 119

Validación de configuración IPv4 en escenario 3

```
PLAY RECAP *****
CE1                : ok=2    changed=2    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
CE2                : ok=2    changed=2    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
CE3                : ok=2    changed=2    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
P1                 : ok=1    changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
P2                 : ok=1    changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
P3                 : ok=1    changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
P4                 : ok=1    changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
PE1                : ok=1    changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
PE2                : ok=1    changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
PE3                : ok=1    changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
```

Posteriormente, se ejecuta el segundo playbook llamado ‘configure_ospf.yml’ mediante el comando `ansible-playbook -i inventory/hosts playbook/configure_ospf.yml`, con el fin de automatizar la configuración del enrutamiento OSPF. Durante la ejecución, se muestra en

consola una salida detallada de las tareas, reflejando la aplicación de cada una, tal como muestra la Figura 120.

Figura 120

Ejecución del playbook OSPF referente al escenario 3

```
PLAY [Configure OSPF on all network devices] *****
TASK [Build OSPF network statements for Cisco] *****
ok: [P1] => (item={'name': 'f0/1', 'ip': '172.16.0.1', 'mask': '255.255.255.0'})
ok: [P4] => (item={'name': 'f0/0', 'ip': '172.16.2.4', 'mask': '255.255.255.0'})
ok: [P2] => (item={'name': 'f0/1', 'ip': '172.16.0.2', 'mask': '255.255.255.0'})
ok: [PE1] => (item={'name': 'f0/0', 'ip': '172.16.7.5', 'mask': '255.255.255.0'})
ok: [P3] => (item={'name': 'f0/0', 'ip': '172.16.3.3', 'mask': '255.255.255.0'})
ok: [P1] => (item={'name': 'f1/0', 'ip': '172.16.3.1', 'mask': '255.255.255.0'})
ok: [P3] => (item={'name': 'f0/1', 'ip': '172.16.2.3', 'mask': '255.255.255.0'})
ok: [P2] => (item={'name': 'f1/0', 'ip': '172.16.1.2', 'mask': '255.255.255.0'})
ok: [P4] => (item={'name': 'f0/1', 'ip': '172.16.1.4', 'mask': '255.255.255.0'})
ok: [PE1] => (item={'name': 'f0/1', 'ip': '172.16.6.5', 'mask': '255.255.255.0'})
ok: [P1] => (item={'name': 'f1/1', 'ip': '172.16.4.1', 'mask': '255.255.255.0'})
ok: [P3] => (item={'name': 'f1/0', 'ip': '172.16.5.3', 'mask': '255.255.255.0'})
ok: [P4] => (item={'name': 'f1/0', 'ip': '172.16.4.4', 'mask': '255.255.255.0'})
ok: [P2] => (item={'name': 'f1/1', 'ip': '172.16.5.2', 'mask': '255.255.255.0'})
ok: [PE1] => (item={'name': 'f2/0', 'ip': '10.10.10.5', 'mask': '255.255.255.0'})
ok: [P1] => (item={'name': 'f0/0', 'ip': '10.10.10.1', 'mask': '255.255.255.0'})
ok: [P2] => (item={'name': 'f2/0', 'ip': '172.16.8.2', 'mask': '255.255.255.0'})
ok: [P4] => (item={'name': 'f1/1', 'ip': '172.16.10.4', 'mask': '255.255.255.0'})
ok: [PE1] => (item={'name': 'Lo0', 'ip': '5.5.5.5', 'mask': '255.255.255.255'})
ok: [P3] => (item={'name': 'f1/1', 'ip': '172.16.6.3', 'mask': '255.255.255.0'})
ok: [P1] => (item={'name': 'Lo0', 'ip': '1.1.1.1', 'mask': '255.255.255.255'})
```

Al terminar con la ejecución del archivo ‘configure_ospf.yml’, se evidencia el resumen final del playbook (PLAY RECAP), donde se observa en la Figura 121, que las tareas fueron procesadas y aplicadas con éxito en todos los dispositivos, sin presentarse errores en la configuración.

Figura 121

Validación de configuración OSPF en escenario 3

```
PLAY RECAP *****
CE1      : ok=2    changed=2    unreachable=0    failed=0    skipped=5    rescued=0    ignored=0
CE2      : ok=2    changed=2    unreachable=0    failed=0    skipped=5    rescued=0    ignored=0
CE3      : ok=2    changed=2    unreachable=0    failed=0    skipped=5    rescued=0    ignored=0
P1       : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
P2       : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
P3       : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
P4       : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
PE1      : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
PE2      : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
PE3      : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
```

El siguiente playbook a ejecutar es ‘configure_mpls.yml’ a través del comando `ansible-playbook -i inventory/hosts playbook/configure_mpls.yml` para verificar el proceso de aplicación de todas las tareas de manera detallada correspondientes a la configuración

automatizada de MPLS, como se muestra en la Figura 122, ya que en la consola se indica si se realiza los cambios o si existen errores que puedan detener la ejecución del playbook.

Figura 122

Ejecución del playbook MPLS referente al escenario 3

```
PLAY [Configure MPLS on Cisco IOS devices] *****

TASK [Enable IP CEF globally (prerequisite for MPLS)] *****
ok: [P4]
ok: [P1]
ok: [P2]
skipping: [CE1]
skipping: [CE2]
skipping: [CE3]
skipping: [multi_server]
ok: [P3]
ok: [PE1]
ok: [PE2]
ok: [PE3]

TASK [Enable MPLS IP globally] *****
changed: [P2]
changed: [P1]
changed: [P3]
changed: [PE1]
skipping: [CE1]
skipping: [CE2]
skipping: [CE3]
```

Al final de la ejecución del playbook ‘configure_mpls.yml’, se presenta el resumen (PLAY RECAP), el cual proporciona información sobre las tareas aplicadas, tal como indica la Figura 123. En este caso, se refleja los cambios y aplicación correcta de las tareas en los routers de núcleo y borde, con estados de ‘changed’ y ok’, mientras que para el resto de dispositivos fueron omitidas dichas tareas (skipped), ya que no estaban contemplados en la configuración MPLS.

Figura 123

Validación de configuración MPLS en escenario 3

```
PLAY RECAP *****
CE1      : ok=0    changed=0    unreachable=0    failed=0    skipped=6    rescued=0    ignored=0
CE2      : ok=0    changed=0    unreachable=0    failed=0    skipped=6    rescued=0    ignored=0
CE3      : ok=0    changed=0    unreachable=0    failed=0    skipped=6    rescued=0    ignored=0
P1       : ok=6    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
P2       : ok=6    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
P3       : ok=6    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
P4       : ok=6    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PE1      : ok=6    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PE2      : ok=6    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PE3      : ok=6    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

A continuación, se ejecuta el playbook ‘configure_qos.yml’ mediante el comando `ansible-playbook -i inventory/hosts playbook/configure_qos.yml` el cual permite validar las

configuraciones de calidad de servicio, ya que en consola se muestra la aplicabilidad de cada tarea definida en el PLAY, como se indica en la Figura 124.

Figura 124

Ejecución del playbook QoS referente al escenario 3

```
PLAY [Aplicar politicas QoS en ambas direcciones] *****

TASK [Configurar ACL para trafico web] *****
changed: [PE3]
changed: [PE2]
changed: [PE1]

TASK [Configurar ACL para trafico FTP] *****
changed: [PE1]
changed: [PE3]
changed: [PE2]

TASK [Configurar ACL para trafico DB] *****
changed: [PE2]
changed: [PE3]
changed: [PE1]

TASK [Crear class-map para trafico web] *****
changed: [PE1]
changed: [PE2]
changed: [PE3]

TASK [Crear class-map para trafico FTP] *****
changed: [PE2]
changed: [PE1]
changed: [PE3]

TASK [Crear class-map para trafico DB] *****
changed: [PE2]
changed: [PE3]
changed: [PE1]
```

El resumen general del playbook (PLAY RECAP) se presenta al final de la ejecución, indicando el total de tareas que han producido cambios y que fueron aplicadas correctamente, sin ninguna detección de errores, tal como se indica en la Figura 125.

Figura 125

Validación de configuración QoS en escenario 3

```
PLAY RECAP *****
PE1           : ok=17  changed=17  unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PE2           : ok=17  changed=17  unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PE3           : ok=17  changed=17  unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Capítulo IV: Evaluación de Resultados

Este último capítulo consiste en evaluar los distintos aspectos resultantes del proceso de automatización mediante la herramienta de Infraestructura como Código (IaC) Ansible, los cuales se presentan en los escenarios de simulación, además, se aborda la fase de optimización con el fin de mejorar la eficiencia operativa y mantener la estabilidad de la gestión centralizada durante la aplicación de configuraciones a través de playbooks.

4.1. Análisis de la gestión de configuraciones centralizada

En este apartado se considera el análisis de aspectos que pueden suscitarse en el proceso de automatización de las configuraciones en equipos de red con Ansible, los cuales incluyen la conexión SSH, manejo de las llaves RSA, las advertencias que surgen durante la ejecución de playbooks y la verificación de configuraciones aplicadas en los dispositivos de red, esto con el fin de evidenciar la efectividad frente posibles situaciones y mejorar la comprensión de procedimientos iniciales para la gestión centralizada, lo cual es la base para aplicar en todos los escenarios escalables.

4.1.1. Autenticación para conexiones SSH

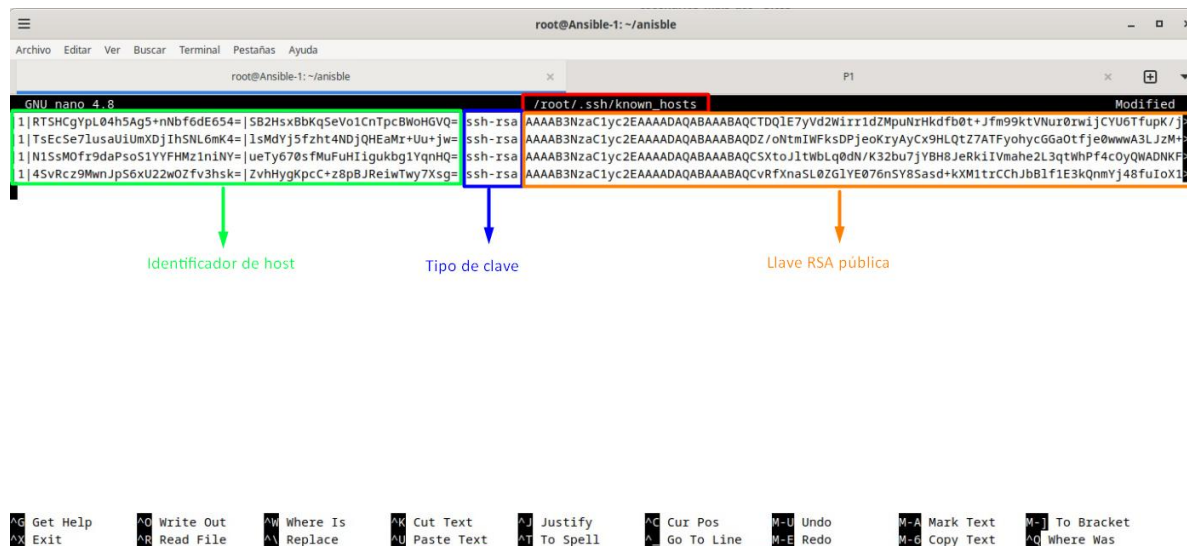
En el proceso de la gestión centralizada de configuraciones, el primer paso es establecer conexión SSH con cada uno de los dispositivos, para ello, es necesario autenticarse usando su dirección IP o nombre distintivo establecido previamente, los cuales puede cambiar en futuras configuraciones de la red por el administrador. Esto puede ocasionar algún fallo, por tanto se analizará dicho comportamiento en el sistema de Ansible, para evitar inconvenientes proceso de automatización.

Al establecer la primera conexión desde el nodo de control hacia cada uno de los equipos gestionados, se registra en Ansible la identidad de cada host, mediante su respectivo identificador (nombre de hosts o IP) con su correspondiente llave pública RSA en el archivo `~/.ssh/known_hosts`, como se muestra en la Figura 126.

Una vez realizada la autenticación en la primera conexión SSH, ya no es necesario repetir el proceso en posteriores conexiones, ya que, la clave registrada en el cliente Ansible va a coincidir con la que vuelve a recibir del dispositivo remoto.

Figura 126

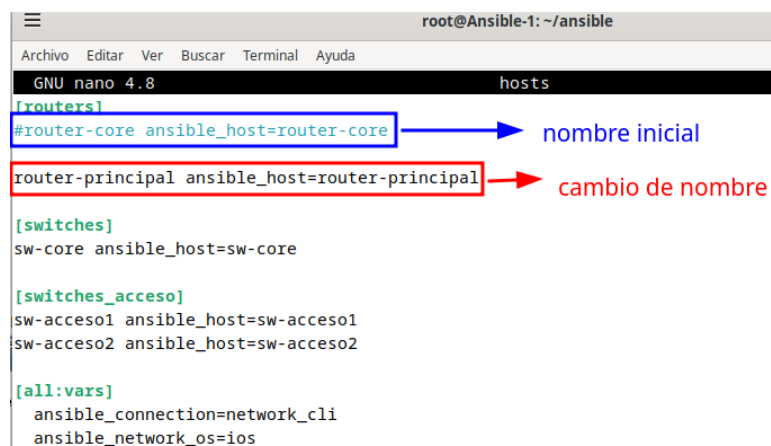
Lista de hosts autenticados



Sin embargo, existe la posibilidad de que el administrador de red modifique el nombre o dirección IP de cualquier host, el cual ya fue previamente autenticado, por tanto en la Figura 127 se muestra un ejemplo para renombrar un equipo tipo router desde el inventario del proyecto, donde inicialmente se lo identificó como *router-core* y posterior se cambia con el nombre a *router-principal*.

Figura 127

Modificación de nombre del host



Al modificar de hostname o IP de un equipo previamente autenticado, el cliente Ansible lo interpreta como nuevo host, ya que la nueva identidad del equipo no coincide con ningún registro guardado en el archivo `~/.ssh/known_hosts`, por ello, al momento de ejecutar un playbook se produce un fallo en la comunicación SSH con dicho dispositivo, debido a la falta de verificación en la autenticidad, como se muestra en la Figura 128.

Figura 128

Error por falta de autenticación en host

```

root@Ansible-1: ~/ansible
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
root@Ansible-1:~/ansible# ansible-playbook -i hosts playbook.yml

PLAY [Configurar router Cisco] *****

TASK [Cambiar de nombre al equipo] *****
fatal: [router-principal]: FAILED! => {"changed": false, "msg": "paramiko: The authenticity of host 'router-principal' can't be established.\\n\\nThe ssh-rsa key fingerprint is b'dfafb2472f2627c90b4663d36982678f'."}

PLAY RECAP *****
router-principal : ok=0  changed=0  unreachable=0  failed=1  skipped=0  rescued=0  ignored=0

```

Por tanto, para evitar el error de conexión SSH desde Ansible hacia el dispositivo renombrado, es necesario autenticar el host con el nuevo identificador, que en este caso corresponde al nombre *router-principal*, tal como se indica en la Figura 129.

Figura 129

Autenticación de host con nuevo identificador

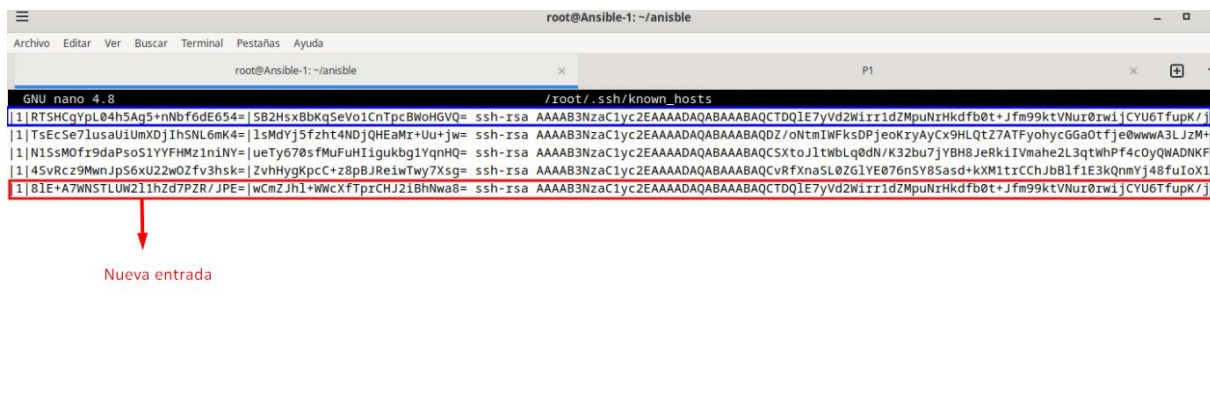
```

root@Ansible-1:~/ansible# ssh ansible@router-principal
The authenticity of host 'router-principal (10.0.0.12)' can't be established.
RSA key fingerprint is SHA256:UGwGOAC26HpdMbsmgprH5JwmW+ThssnH4lfGShrfcog.
Are you sure you want to continue connecting (yes/no/[fingerprint])?

```

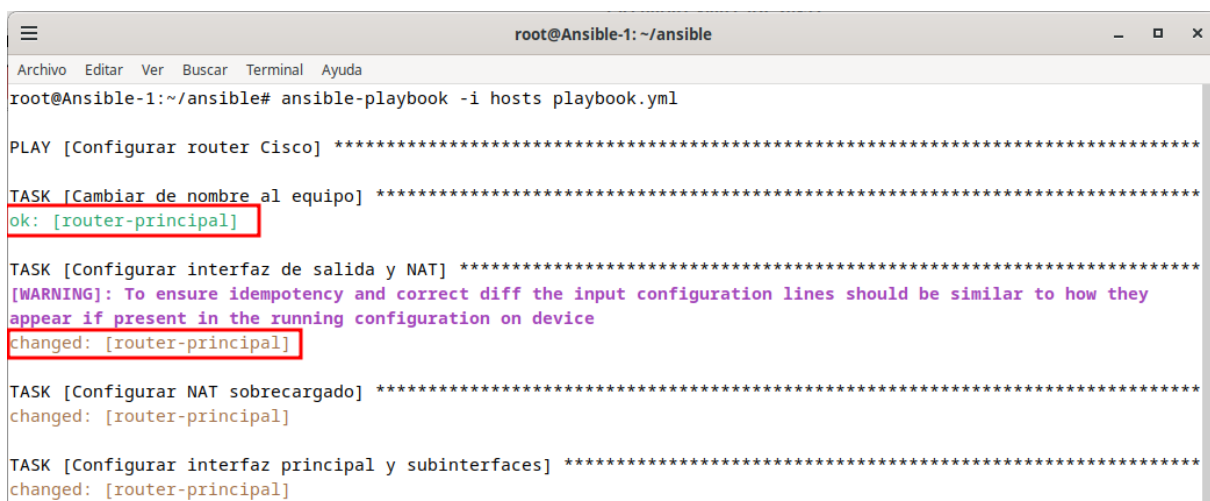
Una vez que se autentica la identidad del dispositivo de red con el nuevo nombre, en el archivo `~/.ssh/known_hosts` se registra una nueva entrada, como se indica en la Figura 130, la cual corresponde a la identidad del mismo equipo (router), pero con nuevo identificador (router-principal) asociado a la misma clave pública RSA generada cuando se habilitó SSH.

Es decir, la clave pública RSA no cambia, a menos que se la vuelva a regenerar, ya que internamente se lo considera como el mismo equipo físico.

Figura 130*Nueva entrada registrada por renombre de host*

Por último, se vuelve a ejecutar el playbook, para comprobar que se establece satisfactoriamente la comunicación entre Ansible y los nodos secundarios registrados, principalmente con el equipo que se cambió su identificador con el nombre *router-principal*, y con la finalidad que sea posible la aplicación de configuraciones, tal como se muestra en la Figura 131.

Por tanto, con este proceso se demuestra que la comunicación con el protocolo SSH garantiza conexiones seguras entre Ansible y los nodos gestionados, evitando que equipos no registrados tengan acceso o exista suplantación de identidad, ya que cada host verifica su autenticidad con claves criptográficas RSA.

Figura 131*Aplicación de configuraciones al equipo renombrado*

4.1.2. Advertencias durante la ejecución de playbooks

En Ansible, durante la ejecución del playbook, pueden surgir advertencias, los cuales no son errores, por tanto no impiden la aplicación de configuraciones, sin embargo, pueden indicar un diagnóstico o información del entorno de trabajo, sobre condiciones relacionadas con: la conexión SSH entre el nodo de control y los nodos gestionados; mal manejo de claves RSA, uso de módulos obsoletos, variables o librerías sin definir, dispositivos sin privilegios administrativos, entre otros. Por tanto, se va a evaluar el impacto de las advertencias presentadas en los escenarios de simulación y si es necesario implementar acciones correctivas.

Con la ejecución de los playbooks en los escenarios de simulación, se aplicó correctamente las configuraciones en los dispositivos de red, sin embargo, se presentaron tres advertencias, las cuales se señalan en la Figura 132 y se detallan a continuación:

Figura 132

Advertencias durante la ejecución del playbook en Ansible

```

root@Ansible-1: ~/ansible
PBorder1

root@Ansible-1:~/ansible# ansible-playbook -i inventory/hosts.yml playbooks/04-configure_ibgp.yml
[WARNING]: Collection cisco.ios does not support Ansible version 2.13.13

PLAY [Configurar iBGP para MPLS-VPN en Routers Cisco IOS PE] *****

TASK [Configurar router-id y vecinos iBGP (remote-as, update-source)] *****
[WARNING]: Collection ansible.netcommon does not support Ansible version 2.13.13
[WARNING]: Collection ansible.netcommon does not support Ansible version 2.13.13
[WARNING]: Collection ansible.utils does not support Ansible version 2.13.13
[WARNING]: Collection ansible.utils does not support Ansible version 2.13.13
[WARNING]: ansible-pylibssh not installed, falling back to paramiko
[WARNING]: ansible-pylibssh not installed, falling back to paramiko
changed: [PBorder2] => (item={'neighbor': '3.3.3.3', 'remote_as': 65000})
[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: [PBorder1] => (item={'neighbor': '6.6.6.6', 'remote_as': 65000})

TASK [Activar vecino en address-family vpnv4] *****
[WARNING]: Collection ansible.netcommon does not support Ansible version 2.13.13
[WARNING]: Collection ansible.netcommon does not support Ansible version 2.13.13
[WARNING]: Collection ansible.utils does not support Ansible version 2.13.13
[WARNING]: Collection ansible.utils does not support Ansible version 2.13.13
changed: [PBorder1] => (item={'neighbor': '6.6.6.6', 'remote_as': 65000})
changed: [PBorder2] => (item={'neighbor': '3.3.3.3', 'remote_as': 65000})

PLAY RECAP *****
PBorder1      : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PBorder2      : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

```

- Advertencia por validación de colecciones: la advertencia se evidencia en el recuadro rojo, esta surge porque las colecciones usadas no están validadas por la versión de Ansible utilizada. Pero en caso de eliminar o modificar algún

parámetro para versiones futuras, va a existir una nueva advertencia para prevenir al administrador y tomar sus medidas.

- Advertencia por librería no instalada: dicha notificación se identifica en el recuadro de color azul, la cual aparece debido a la falta de la librería *pylibssh*, por tanto recae en la librería predeterminada *paramiko*. Dichas librerías se utilizan para establecer la conexión SSH, y no existe inconveniente en usar cualquier de las dos. Sin embargo, puede aportar con conexiones más eficientes la librería *pylibssh* pero por estabilidad se recomienda la librería *paramiko*.
- Advertencia de idempotencia: esta recomendación se encuentra señalada en el recuadro de color verde, indicando que la configuración escrita en los playbooks debe coincidir exactamente con la sintaxis de las configuraciones del equipo, esto con la finalidad de garantizar idempotencia, es decir, que no se vuelvan a aplicar cambios repetidos en las configuraciones del equipo a pesar de ejecutar varias veces el mismo playbook.

En otras palabras, si la configuración ya fue aplicada correctamente, Ansible no la vuelve aplicar

Por lo tanto, se puede decir que las advertencias no impiden la aplicación de configuraciones, ya que el sistema sigue funcionando correctamente, más bien se busca tener un entorno con gestión automatizada de manera estable, confiable y compatible, por ello no es obligatorio tomar medidas correctivas.

4.1.3. Evidenciar cambios aplicados en los dispositivos

Con la ejecución correcta de los playbooks en los tres escenarios de simulación, posteriormente se puede comprobar que las configuraciones fueron aplicadas en los dispositivos de red, para garantizar la gestión automatizada.

Para evidenciar dichos cambios, existe dos maneras de verificación, las cuales depende del fabricante del equipo de red, sin embargo en este proyecto se usaron dispositivos Cisco y Mikrotik, permitiendo validar la interoperabilidad de Ansible con diferentes proveedores.

En caso de dispositivos Mikrotik, se comprueba las configuraciones aplicadas, mediante la ejecución directa de comandos en cada equipo, según el parámetro a verificar.

A continuación, en la Figura 133 se presenta un ejemplo, la cual consiste en ejecutar la tarea de *Configurar conexiones eBGP con vecinos PE*, en dispositivos Mikrotik nombrados como *BCliente*. Para verificar, tomamos de referencia el equipo *BCliente1*, donde se configura la dirección IP (192.168.50.1) y valor del sistema autónomo (6500) del vecino eBGP.

Figura 133

Ejemplo de aplicación de configuración eBGP en equipos Mikrotik

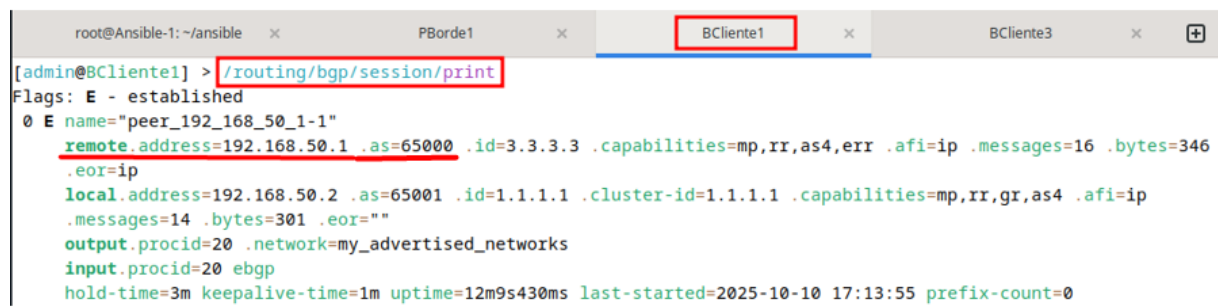
```
TASK [Configurar conexiones eBGP con vecinos PE] *****
[WARNING]: Collection ansible.netcommon does not support Ansible version 2.13.13
[WARNING]: Collection ansible.netcommon does not support Ansible version 2.13.13
[WARNING]: Collection ansible.utils does not support Ansible version 2.13.13
[WARNING]: Collection ansible.utils does not support Ansible version 2.13.13
[WARNING]: Collection ansible.netcommon does not support Ansible version 2.13.13
[WARNING]: Collection ansible.utils does not support Ansible version 2.13.13
[WARNING]: Collection ansible.netcommon does not support Ansible version 2.13.13
[WARNING]: Collection ansible.utils does not support Ansible version 2.13.13
changed: [BCliente2] => (item={'neighbor': '192.168.50.5', 'remote_as': 65000})
changed: [BCliente1] => (item={'neighbor': '192.168.50.1', 'remote_as': 65000})
changed: [BCliente3] => (item={'neighbor': '192.168.50.9', 'remote_as': 65000})
changed: [BCliente4] => (item={'neighbor': '192.168.50.13', 'remote_as': 65000})

PLAY RECAP *****
BCliente1      : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
BCliente2      : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
BCliente3      : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
BCliente4      : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Una vez ejecutada satisfactoriamente la tarea de configuración de sesiones eBGP, se verifica directamente los resultados en el equipo. En este caso, se toma de ejemplo el *BCliente1*, para ejecutar el comando específico */routing/gbp/sesión/print* con el fin de evidenciar la sesión eBGP establecida con el vecino, el cual se confirma con los parámetros configurados correspondiente a la dirección IP (192.168.50.1) y número del sistema autónomo (6500) del vecino PE, como se muestra en la Figura 133.

Figura 134

Verificación directa en el equipo Mikrotik de la configuración eBGP



```

root@Ansible-1: ~/ansible  x  PBorde1  x  BCliente1  x  BCliente3  x  +
[admin@BCliente1] > /routing/bgp/session/print
Flags: E - established
0 E name="peer_192_168_50_1-1"
  remote.address=192.168.50.1 .as=65000 .id=3.3.3.3 .capabilities=mp,rr,as4,err .afi=ip .messages=16 .bytes=346
  .eor=ip
  local.address=192.168.50.2 .as=65001 .id=1.1.1.1 .cluster-id=1.1.1.1 .capabilities=mp,rr,gr,as4 .afi=ip
  .messages=14 .bytes=301 .eor=""
  output.procid=20 .network=my_advertised_networks
  input.procid=20 ebgp
  hold-time=3m keepalive-time=1m uptime=12m9s430ms last-started=2025-10-10 17:13:55 prefix-count=0
  
```

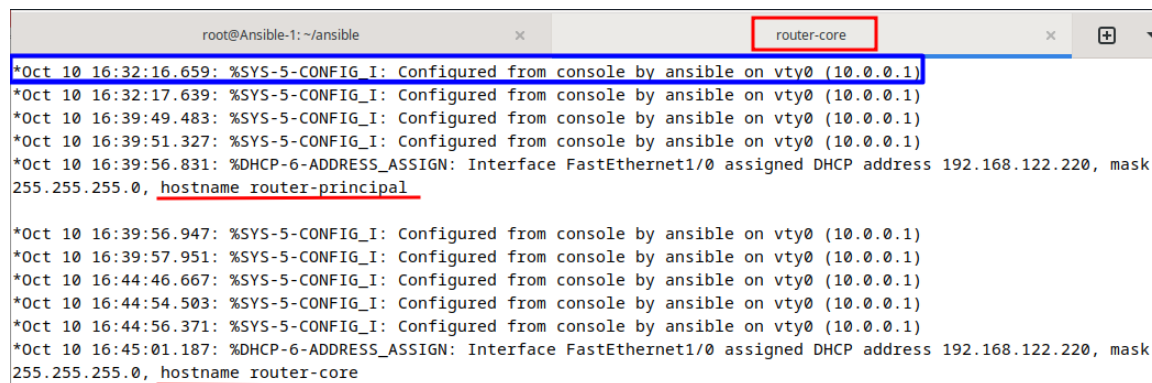
Por otro lado, se verifica la aplicación de las configuraciones en equipos Cisco mediante la ejecución directa de los comandos en cada dispositivo, sin embargo, también es posible visualizar en tiempo real desde la consola del dispositivo las configuraciones que se van aplicando durante la ejecución del playbook.

En este caso, se ejemplifica la modificación del hostname en el equipo, el cual pasa de *router-principal* a *router-core* sin intervención manual, donde únicamente se visualiza el mensaje sobre el cambio realizado en la configuración del equipo *router-core*, como se muestra en la Figura 134.

Además, en los equipos Cisco registran mensajes del sistema, uno de ellos se encuentra señalado en el recuadro azul, indicando que se realizó algún tipo de cambio en la configuración del dispositivo por el usuario *ansible*, el cual accedió por una línea virtual vty0, ya que estableció conexión remota por SSH desde el nodo de control donde se ejecutan los playbooks.

Figura 135

Verificación de configuraciones en la consola de equipo Cisco



```

root@Ansible-1: ~/ansible  x  router-core  x  +
*Oct 10 16:32:16.659: %SYS-5-CONFIG_I: Configured from console by ansible on vty0 (10.0.0.1)
*Oct 10 16:32:17.639: %SYS-5-CONFIG_I: Configured from console by ansible on vty0 (10.0.0.1)
*Oct 10 16:39:49.483: %SYS-5-CONFIG_I: Configured from console by ansible on vty0 (10.0.0.1)
*Oct 10 16:39:51.327: %SYS-5-CONFIG_I: Configured from console by ansible on vty0 (10.0.0.1)
*Oct 10 16:39:56.831: %DHCP-6-ADDRESS_ASSIGN: Interface FastEthernet1/0 assigned DHCP address 192.168.122.220, mask
255.255.255.0, hostname router-principal
*Oct 10 16:39:56.947: %SYS-5-CONFIG_I: Configured from console by ansible on vty0 (10.0.0.1)
*Oct 10 16:39:57.951: %SYS-5-CONFIG_I: Configured from console by ansible on vty0 (10.0.0.1)
*Oct 10 16:44:46.667: %SYS-5-CONFIG_I: Configured from console by ansible on vty0 (10.0.0.1)
*Oct 10 16:44:54.503: %SYS-5-CONFIG_I: Configured from console by ansible on vty0 (10.0.0.1)
*Oct 10 16:44:56.371: %SYS-5-CONFIG_I: Configured from console by ansible on vty0 (10.0.0.1)
*Oct 10 16:45:01.187: %DHCP-6-ADDRESS_ASSIGN: Interface FastEthernet1/0 assigned DHCP address 192.168.122.220, mask
255.255.255.0, hostname router-core
  
```

4.2. Tiempo de ejecución de los playbooks con Ansible

Este apartado tiene el propósito de medir el tiempo de ejecución de los playbooks usados en los escenarios simulados así como el tiempo requerido para habilitar SSH y asignar IP para la red de gestión, con el objetivo de indicar el tiempo total que tarda en configurar el escenario completo mediante una gestión automatizada. Esto permite demostrar el rendimiento de Ansible al desplegar las configuraciones en topologías más grandes, además se evidencia la eficiencia operativa, ya que, se acelera la administración de redes bajo el enfoque de Infraestructura como Código (IaC).

Primero, se calcula el tiempo promedio que tarda en configurar manualmente la habilitación de SSH y la asignación de la IP de gestión, considerando la cantidad de dispositivos por escenario, como se muestra en la Tabla 36. Estas configuraciones son indispensables, ya que forman parte del proceso inicial para establecer la conexión de Ansible con los nodos secundarios, y posteriormente permitir la gestión automatizada.

Los resultados obtenidos en la Tabla 36, serán sumados al tiempo de ejecución de los playbooks de su respectivo escenario, con el fin de calcular un tiempo total más preciso requerido para la configuración completa de la topología de manera automatizada.

Tabla 36
Tiempo promedio de configuraciones iniciales de forma manual

Escenario	Actividad	Comandos	Tiempo promedio por equipo	Número de equipos	Tiempo total promedio de SSH + IP gestión
Escenario	-Habilitar	conf t	1 min 30	4	6 min
1	SSH	ip domain-name	seg		

Escenario	crypto key generate	8	12 min
2	rsa		
Escenario	2048	10	15 min
3	username ansible		
	privilege 15 secret		
	ansible		
	ip ssh version 2		
	line vty 0 4		
	login local		
	transport input ssh		
	no ip domain-name		
	do wr		
- Asignar IP de gestión	-configure terminal		
	- interface g0/0		
	-ip address 10.0.0.2		
	255.255.255.252		
- Guardar cambios	- no shutdown		
	- copy running-		
	config startup-		
	config		

Por otro lado, para registrar el tiempo total de ejecución de los playbooks, se crea un archivo de script Bash llamado *run_all_playboooks.sh* en cada escenario dentro de la carpeta del proyecto de Ansible, al cual se le debe dar permisos previamente antes de ejecutarlo. Este script va a permitir la ejecución secuencial de los playbooks empleados en la topología

simulada para obtener en segundos el tiempo total del proceso de automatización de configuraciones.

A continuación, se muestra en la Figura 136, el contenido del script Bash denominado *run_all_playboooks.sh*, el cual inicia declarando la variable de tiempo *START_ALL* para guardar el momento exacto en que comienza la ejecución de los playbooks, posteriormente se coloca la secuencia de comandos para ejecutar cada playbook (ejemplo: *ansible-playbook -i inventory/hosts.yml playbooks/configure_ipv4.yml*). Una vez terminado el proceso de automatización, se procede a definir la variable *END_ALL* para registrar el momento final de la ejecución que captura el comando *date +%s*, por último se declara la variable *TOTAL_TIME*, la cual, almacena el tiempo calculado por la resta de *START_ALL - END_ALL*, por lo tanto, dicho resultado corresponde al tiempo que tarda en ejecutar todos los scripts de configuración automatizada con Ansible.

Figura 136

Archivo Bash para medir el tiempo total de ejecución de playbooks



```

root@Ansible-1: ~/ansible
GNU nano 4.8 run_all_playboooks.sh
#!/bin/bash

echo "=====
echo " INICIO DE EJECUCION DE TODOS LOS PLAYBOOKS "
echo "=====

START_ALL=$(date +%s)

# --- PLAYBOOK 1 ---
echo "[1/9] Ejecutando 01-configure_ipv4.yml ..."
ansible-playbook -i inventory/hosts.yml playbooks/01-configure_ipv4.yml

# --- PLAYBOOK 2 ---
echo "[2/9] Ejecutando 02-configure_ospf.yml ..."
ansible-playbook -i inventory/hosts.yml playbooks/02-configure_ospf.yml

# --- PLAYBOOK 3 ---
echo "[3/9] Ejecutando 03-configure_mpls_ldp.yml ..."
ansible-playbook -i inventory/hosts.yml playbooks/03-configure_mpls_ldp.yml

# --- PLAYBOOK 4 ---
echo "[4/9] Ejecutando 04-configure_ibgp.yml ..."
ansible-playbook -i inventory/hosts.yml playbooks/04-configure_ibgp.yml

# --- PLAYBOOK 5 ---
echo "[5/9] Ejecutando 05-configure_vrfs.yml ..."
ansible-playbook -i inventory/hosts.yml playbooks/05-configure_vrfs.yml

# --- PLAYBOOK 6 ---
echo "[6/9] Ejecutando 06-configure_ebgp_pe.yml ..."
ansible-playbook -i inventory/hosts.yml playbooks/06-configure_ebgp_pe.yml

END_ALL=$(date +%s)
TOTAL_TIME=$((END_ALL - START_ALL))

echo "=====
echo " TIEMPO TOTAL DE EJECUCION $TOTAL_TIME segundos"
echo "=====

^G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify
^X Exit          ^R Read File    ^N Replace      ^U Paste Text   ^T To Spell

```

El archivo Bash de cada escenario será ejecutado 20 veces para registrar el tiempo que tarda en ejecutar los playbooks, entonces, a partir de los 20 valores obtenidos se calcula métricas estadísticas como media y desviación estándar, con el fin de calcular un tiempo promedio más preciso de la configuración automatizada por escenario y analizar la dispersión de los tiempos de ejecución para identificar el rendimiento de Ansible.

Para calcular la métrica de promedio o media se considera la (Ec. 1, que consiste en sumar todos los tiempos registrados en cada ejecución (t_i) y dividir para el número total de iteraciones (n).

$$\bar{t} = \frac{1}{n} \sum_{i=1}^n t_i \quad (\text{Ec. 1})$$

Donde:

- $\bar{t}$: es el tiempo promedio del conjunto de datos
- n : el número total de iteraciones
- t_i : es el tiempo obtenido de cada iteración (i)

Por otro lado, a partir de la (Ec. 2 se calcula la desviación estándar muestral para medir la variabilidad de los tiempos, para ello, primero se realiza una diferencia entre el tiempo de cada iteración y el promedio, donde dicha operación se eleva al cuadrado $((t_i - \bar{t})^2)$, posteriormente se suma los resultados de la diferencia cuadrática de todas las iteraciones y este total se divide para la cantidad de iteraciones menos 1 ($n - 1$), finalmente el valor obtenido de todo este proceso se aplica la raíz cuadrada.

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (t_i - \bar{t})^2} \quad (\text{Ec. 2})$$

Donde:

- s : es la desviación estándar muestral del conjunto de datos
- t_i : es el tiempo obtenido de cada iteración (i)

- $\bar{t}$: es el tiempo promedio del conjunto de datos previamente calculado
- n : el número total de iteraciones

A continuación, se realiza la ejecución del respectivo archivo Bash en cada escenario, para presentar los resultados del tiempo total que tarda Ansible en aplicar las configuraciones automatizadas a dispositivos de red, correspondientes a switches y routers.

Al medir la duración del proceso de automatización se genera 3 valores de tiempos, como se demuestra en la Figura 137:

- **real:** tiempo total que tardó el proceso de automatización, incluyendo el tiempo en la aplicación de todas las configuraciones y el tiempo de espera en conexiones SSH.
- **user:** tiempo que usa el CPU para la ejecución de tareas con Ansible.
- **sys:** tiempo que usa el CPU en operaciones del sistema, por ejemplo establecer conexiones SSH y abrir archivos para ejecutar las tareas de configuración.

En el primer escenario, corresponde a una topología sencilla que se configura VLANs y STP en cuatro dispositivos de red, como evidencia de la ejecución del playbook, se muestra en la Figura 137 un resultado de referencia, donde obtiene el tiempo total de 1 minuto con 33 segundos (93 segundos) para configurar el escenario completamente, lo cual refleja el inicio de una configuración eficiente debido a la gestión centralizada.

Figura 137

Tiempo total referente de ejecución del playbook - Escenario 1

```
PLAY RECAP *****
router-core      : ok=5    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
sw-acceso1       : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
sw-acceso2       : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
sw-core          : ok=6    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

=====
TIEMPO TOTAL DE EJECUCION: 93 segundos
=====

real    1m32.985s
user    0m23.827s
sys     0m2.662s
root@Ansible-1: ~/ansible#
```

Luego, en la Tabla 37 se evidencia los tiempos registrados en las 20 iteraciones de la ejecución del archivo Bash del escenario 1, en dicha tabla se detalla el tiempo en minutos con segundos y su equivalente solo en segundos. Por tanto, con dichos datos, se calcula un tiempo promedio de 93.25 segundos (1 min 33 segundos) y una desviación estándar de 1.86 segundos (≈ 2 segundos), es decir que en cada ejecución del playbook va a tardar aproximadamente 2 segundos más o menos respecto al tiempo promedio, lo cual indica una dispersión de tiempo bajo al momento de ejecutar la configuración automatizada.

Tabla 37

Tiempo total promedio de la ejecución de playbook - Escenario 1

Iteración	Tiempo (minutos/segundos)	Tiempo en segundos	$(t_i - \bar{t})^2$
1	1 min 32 seg	92	1.56
2	1 min 34 seg	94	0.56
3	1 min 33 seg	93	0.06
4	1 min 33 seg	93	0.06
5	1 min 33 seg	93	0.06
6	1 min 35 seg	95	3.06
7	1 min 33 seg	93	0.06
8	1 min 30 seg	90	10.56
9	1 min 32 seg	92	1.56
10	1 min 34 seg	94	0.56
11	1 min 33 seg	93	0.06
12	1 min 33 seg	93	0.06
13	1 min 32 seg	92	1.56

14	1 min 31 seg	91	5.06
15	1 min 33 seg	93	0.06
16	1 min 32 seg	92	1.56
17	1 min 32 seg	92	1.56
18	1 min 38 seg	98	22.56
19	1 min 35 seg	95	3.06
20	1 min 37 seg	97	14.06
Sumatoria del tiempo		1865	
TIEMPO PROMEDIO ($\bar{t}$)		$\bar{t} = 93.25$	
Sumatoria de $(t_i - \bar{t})^2$			67.75
Desviación estándar			$s = 1.86$

El resultado del promedio mostrado en la Tabla 37 se obtiene aplicando la (Ec. 1:

$$\bar{t} = \frac{\text{sumatoria del tiempo}}{\text{cantidad de iteraciones}}$$

$$\bar{t} = \frac{1865}{20}$$

$$\bar{t} = 93.25 \text{ segundos} \approx 1 \text{ min } 33 \text{ segundos}$$

Mientras que la desviación estándar, se obtuvo por la aplicación de la (Ec. 2:

$$s = \sqrt{\frac{\text{sumatoria de } (t_i - \bar{t})^2}{\text{cantidad de iteraciones} - 1}}$$

$$\sigma = \sqrt{\frac{67.75}{19}}$$

$$s = 1.86 \text{ segundos} \approx 2 \text{ min}$$

Para el segundo escenario, se configura un total de ocho dispositivos que desempeñan diferentes roles en una red MPLS-VPN, y como evidencia del proceso de ejecución, se registra en una iteración de la ejecución de los playbooks un tiempo total de referencia de 2 minutos con 9 segundos (129 segundos), tal como se muestra en la Figura 138.

Figura 138

Tiempo total referente de ejecución de playbooks - Escenario 2

```
PLAY RECAP *****
BCliente1      : ok=6    changed=6    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
BCliente2      : ok=6    changed=6    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
BCliente3      : ok=6    changed=6    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
BCliente4      : ok=6    changed=6    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

=====
TIEMPO TOTAL DE EJECUCION 128 segundos
=====

real    2m7.463s
user    2m48.690s
sys      0m16.384s
```

A continuación, se presenta los tiempos registrados en las 20 iteraciones de la ejecución de los playbooks del escenario 2, indicando el tiempo expresado en minutos y segundos, junto a su equivalente en segundos, como se muestra en la Tabla 38.

A partir de los 20 tiempos registrados se obtiene un tiempo total promedio de 120.05 segundos (≈ 2 min 01 segundos) y una desviación estándar de 2.13 segundos (≈ 2 segundos). Entonces, en cada ejecución secuencial de los playbooks tiende a tardar aproximadamente 2 segundos más o menos respecto al tiempo promedio.

Por tanto, los tiempos de ejecución no tienen variaciones significativas, indicando un buen rendimiento y estabilidad de Ansible. Sin embargo, se demuestra un incremento pequeño en el tiempo respecto al anterior escenario, debido al aumento de la cantidad de equipos y tareas a configurar.

Tabla 38

Tiempo total promedio de la ejecución de playbooks - Escenario 2

Iteración	Tiempo	Tiempo en	$(t_i - \bar{t})^2$
	(minutos/segundos)	segundos	

1	2 min 07 seg	127	36
2	2 min 00 seg	120	1
3	2 min 00 seg	120	1
4	1 min 59 seg	119	4
5	2 min 00 seg	120	1
6	2 min 03 seg	123	4
7	2 min 05 seg	125	16
8	2 min 00 seg	120	1
9	2 min 01 seg	121	0
10	2 min 03 seg	123	4
11	2 min 02 seg	122	1
12	2 min 00 seg	120	1
13	2 min 00 seg	120	1
14	1 min 59 seg	119	4
15	1 min 59 seg	119	4
16	2 min 00 seg	120	1
17	1 min 59 seg	119	4
18	2 min 02 seg	122	1
19	2 min 01 seg	121	0
20	2 min 00 seg	120	1
Sumatoria del tiempo		2420	
TIEMPO PROMEDIO ($\bar{t}$)		$\bar{t} = 121$	
Sumatoria de $(t_i - \bar{t})^2$			86
Desviación estándar			$s = 2.13$

Por último, en la Figura 139 se presenta un valor de referencia del tiempo total que tarda en una de las iteraciones de ejecución de los playbooks, para configurar el tercer escenario, el cual, se implementa MPLS y QoS en 10 routers en 2 minutos con 56 segundos (176 segundos), esta topología corresponde a la más extensa por lo tanto el tiempo es mayor.

Figura 139

Tiempo total referente de ejecución de playbooks - Escenario 3

```
PLAY RECAP *****
PE1      : ok=17  changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PE2      : ok=17  changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PE3      : ok=17  changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

=====
TIEMPO TOTAL DE EJECUCION: 176 segundos
=====

real    2m56.055s
user    3m38.722s
sys     0m20.986s
```

Posteriormente, se presenta los valores obtenidos en la ejecución de los playbooks en cada iteración del escenario 3, como el tiempo en minutos y segundos con su equivalente en segundos, tal como se muestra en la Tabla 39.

Además, con los 20 datos registrados, correspondientes al tiempo que tarda en aplicar los playbooks, se obtiene el tiempo promedio de 177.4 segundos (≈ 2 min 57 segundos) y la desviación estándar 3.78 segundos (≈ 4 segundos), esto indica que Ansible se demora aproximadamente 4 segundos más o menos respecto al tiempo promedio para completar la automatización de configuraciones, por tanto, se evidencia estabilidad y rendimiento alto por parte de la herramienta IaC.

Tabla 39

Tiempo total promedio de la ejecución de playbooks - Escenario 3

Iteración	Tiempo (minutos/segundos)	Tiempo en segundos	$(t_i - \bar{t})^2$
1	3 min 11 seg	191	184.96
2	2 min 57 seg	177	0.16

3	2 min 56 seg	176	1.96
4	2 min 54 seg	174	11.56
5	2 min 56 seg	176	1.96
6	2 min 56 seg	176	1.96
7	3 min 04 seg	184	43.56
8	2 min 57 seg	177	0.16
9	2 min 58 seg	178	0.36
10	2 min 57 seg	177	0.16
11	2 min 57 seg	177	0.16
12	2 min 56 seg	176	1.96
13	2 min 56 seg	176	1.96
14	2 min 55 seg	175	5.76
15	2 min 56 seg	176	1.96
16	2 min 55 seg	175	5.76
17	2 min 55 seg	175	5.76
18	2 min 58 seg	178	0.36
19	2 min 56 seg	176	1.96
20	2 min 58 seg	178	0.36
Sumatoria del tiempo		3548	
TIEMPO PROMEDIO ($\bar{t}$)		$\bar{t} = 177.4$	
Sumatoria de $(t_i - \bar{t})^2$			272.8
Desviación estándar			$s = 3.78$

Una vez obtenidos la desviación estándar y el tiempo promedio de la ejecución de los playbooks en cada escenario, en la Tabla 40 se procede a calcular el tiempo total promedio

mediante configuración automatizada en cada escenario, para ello, se suma el tiempo de las configuraciones preparatorias registradas en la Tabla 36 con el tiempo promedio de la ejecución de los playbooks.

Tabla 40

Calcular el tiempo total promedio del proceso de configuración automatizada

Escenario	Tiempo total promedio de configuraciones manuales (SSH + IP gestión)	Tiempo total promedio de ejecución de playbooks	Tiempo total promedio de configuración automatizada
Escenario 1	360 segundos (6 min)	93 segundos = 1 min 33 seg	453 seg = 7 min 33 seg
Escenario 2	720 segundos (12 min)	121 segundos = 2 min 01 seg	841 seg = 14 min 01 seg
Escenario 3	900 segundos (15 min)	177 segundos = 2 min 57 seg	1077 seg = 17 min 57 seg

Los datos obtenidos del tiempo total promedio de configuración automatizada, se presenta a continuación la Tabla 41, la cual recopila información de cada escenario, como las configuraciones aplicadas, el número de dispositivos y de tareas configuradas, ya que, son factores que influyen en el tiempo total que tarda para configurar el escenario completo.

Por lo tanto, se evidencia que conforme aumenta la cantidad de dispositivos y tareas a configurar también incrementa proporcionalmente el tiempo total de configuración automatizada, por ello la diferencia entre los tiempos registrados de los escenarios no es abrupto, demostrando que Ansible mantiene escalabilidad, ya que no se degrada el rendimiento al gestionar mayores cargas.

Además se demuestra estabilidad en la gestión automatizada, ya que la variación del tiempo de ejecución de los playbooks en cada escenario no supera los 5 segundos, indicando tiempos consistentes entre iteraciones de ejecución.

Por último, ese evidencia un buen rendimiento en la automatización de configuraciones, dado que los tiempos de configuración son bajos respecto a la cantidad de configuraciones aplicadas, esto ocurre porque Ansible crea varios procesos de trabajo en paralelo y aprovecha los recursos del procesador.

Tabla 41

Tiempo total promedio de la configuración automatizada mediante Ansible de los tres escenarios

Escenario	Configuraciones generales	Cantidad de dispositivos	Cantidad de tareas de configuración	Tiempo total promedio de configuración automatizada	Variación de tiempo de ejecución
Escenario 1	VLANs y STP	4	14	453 seg = 7 min 33 seg	1.86 seg
Escenario 2	MPLS-VPN	8	38	841 seg =14 min 01 seg	2.13 seg
Escenario 3	MPLS y QoS	10	40	1077 seg = 17min 57 seg	3.78 seg

4.2.1. Comparación del tiempo entre configuración automatizada y manual

En esta sección se compara el tiempo que tarda en configurar los escenarios mediante la gestión automatizada con Ansible y el método tradicional. Esta comparativa evidencia la diferencia de los tiempos de configuración entre ambos métodos, además de demostrar la

eficiencia operativa de Ansible, ya que reduce el tiempo de implementación de configuraciones y la intervención humana.

Para estimar el tiempo del método tradicional, se debe considerar que el administrador de red aplica los comandos de forma manual e individual en cada dispositivo, además el tiempo que tarde en el proceso de configuración depende de la experticia del administrador. Por lo tanto, se considera un tiempo promedio para la configuración manual de cada escenario.

A continuación, se presenta la Tabla 42, la cual indica el tiempo promedio requerido para aplicar comandos manualmente en el primer escenario, para ello se detalla la actividad realizada, los comandos aplicados, el tiempo promedio por dispositivo y el tiempo total por actividad según la cantidad de dispositivos configurados.

Tabla 42

Tiempo total promedio de configuraciones manuales- Escenario 1

Actividad	Comandos	Tiempo promedio de equipo	Numero de dispositivos	Tiempo total promedio por actividad
Cambiar nombre	-hostname R1	10 seg	4	40 seg
Habilitar SSH	conf t ip domain-name crypto key generate rsa 2048 username ansible privilege 15 secret ansible ip ssh version 2	1 min	4	4 min

	line vty 0 4			
	login local			
	transport input ssh			
	no ip domain-name			
	do wr			
Configurar	- interface f1/0	1 min	1	1 min
NAT	- ip address dhcp			
sobrecargado	- ip nat outside			
	- access-list 1 permit any			
	- ip nat inside source list			
	1 interface f1/0 overload			
Configuración	- interface f0/0.10	3 min	1	3 min
de subinterfaces	- encapsulation dot1Q 10			
en router	- ip address 192.168.10.1			
	-ip nat inside			
Configurar	- interface f0/0	2 min 30	3	7 min 30 seg
interfaces	- switchport trunk	seg		
troncales	encapsulation dot1q			
	- switchport mode trunk			
	- no shutdown			
Crear VLANs	- vlan 10	2 min	1	2 min
10-90	-name <i>Datos</i>			
	-exit			

Configurar	- vtp domain tesis	1 min	3	3 min
VTP server y cliente	- vtp mode server/client - vtp version 2			
Configurar STP	-spanning-tree vlan 10 priority 0	2 min	1	2 min
Guardar configuraciones	- copy running-config startup-config	10 seg	4	40 seg
Tiempo total promedio de configuración				23 min 30 seg

Para el segundo escenario (MPLS-VPN), se presenta el tiempo total promedio que requiere para configurar la topología de manera manual, el cual se calcula en base a los comandos aplicados para equipos Cisco y Mikrotik, el tiempo estimado para configurar por equipo y tiempo total que tarda en configurar toda la actividad como se muestra en la Tabla 43.

Tabla 43
Tiempo total promedio de configuraciones manuales- Escenario 2

Actividad	Fabricante	Comandos	Tiempo promedio por equipo	Numero de equipos	Tiempo total por actividad
Cambiar nombre	Cisco	- hostname PE1	10 seg	8	1 min 20 seg
	Mikrotik	- /system identity set name=CE1			
Habilitar SSH	Cisco	conf t ip domain-name crypto key generate rsa	1 min	8	8 min

		2048			
		username ansible			
		privilege 15 secret			
		ansible			
		ip ssh version 2			
		line vty 0 4			
		login local			
		transport input ssh			
		no ip domain-name			
		do wr			
Configurar	Cisco	- interface g0/0	1 min	8	8 min
todas las		-ip address 10.0.0.2			
interfaces		255.255.255.252			
IPv4		- no shutdown			
	Mikrotik	- /ip address add			
		address=192.168.10.			
		1/30			
		interface=ether1			
Configurar	Cisco	- interface	1 min	8	8 min
interfaces		Loopback0			
Loopback		- ip address 1.1.1.1			
		255.255.255.255			
		- no shutdown			
	Mikrotik	- /interface bridge			
		add name=lo			

		- /ip address add address=7.7.7.7/32 interface=lo			
Configurar OSPF	Cisco	- router ospf 1 - network 10.0.0.0 0.0.0.3 area 0 - router-id 2.2.2.2	1min 30 seg	4	6 min
Configurar MPLS	Cisco	- ip cef - mpls ip - mpls ldp router-id Loopback0 force - interface g0/1 - mpls ip	2 min	4	6 min
Configurar iBGP en routers de Borde (PE)	Cisco	- router bgp 65000 - neighbor 1.1.1.2 remote-as 65000 - neighbor 1.1.1.2 update-source Loopback0	2 min	2	4 min
Crear VFRs y asignar interfaces en (PE)	Cisco	- ip vrf <i>red1</i> - rd 65000:1 - interface g0/1 - ip vrf forwarding <i>red1</i>	2 min	2	4 min

Configurar	Cisco	- router bgp 65000	2 min	2	4 min
eBGP por		- address-family			
VRFs en		ipv4 vrf <i>red1</i>			
(PE)		- neighbor			
		192.168.10.2			
		remote-as 65010			
Configurar	Mikrotik	- /routing bgp	3 min	4	12 min
eBGP en		template add			
routers de		name=ebgp-template			
borde de		as=65010 router-			
cliente		id=10.10.10.2			
(CE)		- /routing bgp			
		connection add			
		remote.as=65000			
		remote.address=10.0			
		.0.2 template=ebgp-			
		template			
Anunciar	Cisco	- router bgp 65000	1 min 30	2	3 min
rutas en		- address-family	seg		
VRFs en		ipv4 vrf <i>red1</i>			
(PE)		- redistribute			
		connected			
		- redistribute static			
Anunciar	Mikrotik	- /ip firewall	1 min 30	4	6 min
redes por		address-list add	seg		

BGP en (CE)		list=my_advertised_ networks address=192.168.10. 0/24 - /routing bgp connetion ser peer_10_0_0_1 output.network=my_ advertised _netorwoks			
Guardar configuraciones	Cisco	- copy running- config startup-config	10 seg	8	1 min 40 seg
Tiempo total promedio de configuración					73 min $\approx$ 1 hora 13 min

De igual manera, se considera los criterios mencionados en las anteriores tablas para calcular el tiempo promedio total para la configuración manual del tercer escenario (MPLS-QoS), el cual se presenta en la Tabla 44.

Tabla 44
Tiempo total promedio de configuraciones manuales- Escenario 3

Actividad	Fabrica nte	Comandos	Tiempo promedio por equipo	Numero de equipos	Tiempo total por actividad
	Cisco	- hostname R1	10 seg	10	

Cambiar	Mikrotik	- /system identity set			1 min 40
nombre		name=C1			seg
Habilitar	Cisco	conf t	1 min	10	10 min
SSH		ip domain-name			
		crypto key generate rsa			
		2048			
		username ansible			
		privilege 15 secret			
		ansible			
		ip ssh version 2			
		line vty 0 4			
		login local			
		transport input ssh			
		no ip domain-name			
		do wr			
Configurar	Cisco	- interface g1/0	1 min	10	10 min
todas las		-ip address 10.0.0.5			
interfaces		255.255.255.252			
IPv4		- no shutdown			
	Mikrotik	- /ip address add			
		address=10.10.10.9/30			
		interface=ether3			
Configurar	Cisco	- interface Loopback0	1 min	10	10 min
interfaces		- ip address 2.2.2.2			
Loopback		255.255.255.255			

		- no shutdown			
	Mikrotik	- /interface bridge add name=lo - /ip address add address=7.7.7.7/32 interface=lo			
Configurar	Cisco	- router ospf 1	1min 30 seg	10	15 min
OSPF		- router-id 2.2.2.2 - network 10.0.0.0 0.0.0.3 area 0			
	Mikrotik	-/routing ospf instance add name=default router-id=1.1.1.1 - /routing ospf area add name=backbone area- id=0.0.0.0 - /routing ospf interface-template add area=backbone networks=10.10.10.0/3 0			
Configurar	Cisco	- ip cef	2 min	7	14 min
MPLS		- mpls ip - mpls ldp router-id Loopback0 force			

		- interface g1/1			
		- mpls ip			
Definir	Cisco	- ip access-list extended	3 min	3	9 min
ACLs para		WEBACL			
clasificar el		- permit tcp any any eq			
tráfico		80			
		- ip access-list extended			
		FTPACL			
		- permit tcp any any eq			
		21			
		- ip access-list extended			
		DBACL			
		- permit tcp any any eq			
		3306			
Crear	Cisco	- class-map match-any	3 min	3	9 min
class-map		WEB-TRAFFIC			
		- match access-group			
		name WEBACL			
		- class-map match-any			
		FTP-TRAFFIC			
		- match access-group			
		name FTPACL			
		- class-map match-any			
		DB-TRAFFIC			

		- match access-group			
		name DBACL			
Crear	Cisco	- policy-map QoS-	3 min	3	9 min
policy-map		OUTPUT-POLICY			
de salida y		- class WEB-TRAFFIC,			
asignar		priority percent 30			
clases		- class FTP-TRAFFIC			
		- bandwidth percent 20			
		- class DB-TRAFFIC			
		- bandwidth percent 20			
Crear	Cisco	-policy-map QoS-	3 min	3	9 min
policy-map		INPUT-POLICY			
de entrada		- class WEB-TRAFFIC			
y asignar		- set dscp af31			
marcación		- class FTP-TRAFFIC			
DSCP		- set dscp af11			
		- class DB-TRAFFIC			
		- set dscp af21			
Aplicar	Cisco	- interface g0/0	2 min	3	6 min
políticas		- service-policy input			
Qos		QoS-INPUT-POLICY			
(entrada y		- service-policy output			
salida) en		QoS-OUTPUT-			
interfaces		POLICY			
de borde					

Guardar	Cisco	- copy running-config	10 seg	10	1 min 40
configuraci		startup-config			seg
ón		- /system backup save			
		name=config_mpls_qos			
Tiempo total promedio de configuración					104 min 20
					seg $\approx$ 1hora
					44min

Por último, se presenta la Tabla 45, la cual indica los tiempos promedios obtenidos mediante la configuración automatizada a través de playbooks y el tiempo estimado de la configuración manual en los tres escenarios. A partir de estos resultados, se calcula el porcentaje de reducción del tiempo de cada escenario, con el fin de evidenciar la eficiencia en procesos de configuración con Ansible en comparación de métodos tradicionales.

El porcentaje de optimización del tiempo de configuración se calcula con la Ec. 3, que consiste en la diferencia entre el tiempo de configuración manual y el tiempo de configuración automatizado, el resultado de dicha diferencia se divide para el tiempo de configuración manual y por último se multiplica por 100 para que el resultado se exprese en porcentaje.

$$\text{Optimización (\%)} = \frac{T_{\text{manual}} - T_{\text{ansible}}}{T_{\text{manual}}} * 100 \quad (\text{Ec. 3})$$

Como ejemplo de cálculo se reemplaza los valores del tiempo de configuración manual y automatizada del escenario 1, obteniendo un ahorro de tiempo en la configuración del 67.87%. Mientras que el resto de los resultados se detalla en la tabla.

$$\text{Optimización (\%)} = \frac{1410 - 453}{1410} * 100$$

$$\text{Optimización (\%)} = 0.6787 * 100$$

$$\text{Optimización (\%)} = 67.87 \%$$

Tabla 45
Comparación de tiempos de configuración automatizada y manual

Escenario	Tiempo total de configuración con Ansible	Tiempo total promedio con configuración manual	Porcentaje de optimización del tiempo
Escenario 1	453 seg = 7 min 33 seg	1410 seg = 23 min 30 seg	67.87 %
Escenario 2	841 seg =14 min 01 seg	4393 seg =73 min 13 seg	80.85 %
Escenario 3	1077 seg =17 min 57 seg	6260 seg =104 min 20 seg	82.79 %

Por lo tanto, se demuestra que el tiempo de configuración mediante Ansible es notablemente menor a comparación de aplicar comandos manualmente, alcanzando reducciones de tiempo de configuración entre un 67 % y 82% según el escenario, ya que mientras más extensa es la red existe mayor ahorro de tiempo, esto ocurre porque al realizar configuraciones manuales en redes más complejas requiere mayor esfuerzo y el tiempo se triplica, sin embargo Ansible incrementa el tiempo pero de manera moderada.

Por ende, la herramienta de automatización proporciona escalabilidad y mayor eficiencia operacional en procesos de configuración de baja o alta complejidad, ya que reduce la intervención manual y errores humanos.

4.3. Optimización para automatización de configuraciones

En esta sección se presentan dos mecanismos implementados en el proyecto de Ansible para optimizar el despliegue de configuraciones en dispositivos de red, como la agrupación de hosts por rol y la definición de playbooks según su función. Esto con la finalidad de tener una

estructura de proyecto más ordenada que permita mejorar el rendimiento y la escalabilidad en topologías extensas mediante una gestión centralizada.

4.3.1. Agrupación de hosts por roles

El mecanismo de agrupación de dispositivos por roles aporta a la optimización de las configuraciones, ya que, en el inventario se organiza los dispositivos según la función que desempeña dentro de la topología de red. Por tanto, al tener una estructura organizada, se evita definir tareas repetitivas por dispositivo individual.

Como referencia se considera el inventario del escenario 2, correspondiente a la topología MPLS-VPN, donde se optó una estructura que clasifica los dispositivos por grupos, según el rol operativo que desempeña los enrutadores, ya sea core (núcleo), borde (pe) y clientes (ce), como se muestra en la Figura 140. Esta organización evita tener una lista plana de equipos que perjudique la gestión y claridad en la arquitectura del proyecto.

Figura 140

Agrupación de hosts por rol en inventario de Ansible

```

GNU nano 4.8 hosts.yml
all:
  children:
    core:
      hosts:
        PNucleo1:
          ansible_host: PNucleo1
          platform: ios
          ansible_connection: network_cli
          ansible_network_os: ios
        PNucleo2:
          ansible_host: PNucleo2
          platform: ios
          ansible_connection: network_cli
          ansible_network_os: ios
    pe:
      hosts:
        PBorde1:
          ansible_host: PBorde1
          platform: ios
          ansible_connection: network_cli
          ansible_network_os: ios
        PBorde2:
          ansible_host: PBorde2
          platform: ios
          ansible_connection: network_cli
          ansible_network_os: ios
    ce:
      hosts:
        BCliente1:
          ansible_host: BCliente1
          platform: routers
          ansible_connection: network_cli
          ansible_network_os: routers
  
```

Routers de Núcleo

Routers de Borde

Routers Cliente

La agrupación de hosts según el rol que desempeña en la red permite una estructura ordenada en el inventario, lo cual, facilita agregar nuevos dispositivos en redes en crecimiento, además se puede definir y ejecutar las configuraciones a un grupo de dispositivos para evitar la duplicación de tareas. Por tanto, se optimiza la integración de configuraciones automatizadas a una red más compleja, asegurando la correcta aplicación de tareas desde Ansible que permitan la interconectividad entre los nodos independientemente del fabricante de los dispositivos.

4.3.2. Definir playbooks por función

Otro mecanismo que aporta en la optimización de configuraciones automatizadas es la definición de playbooks por función, es decir, cada archivo de configuración (playbook) se conforma por un conjunto de tareas enfocadas a una configuración específica, para tener un despliegue de configuraciones ordenado, adaptable y eficiente en escenarios más complejos.

De este modo, se medirá el tiempo que tarda en ejecutar el playbook con configuraciones específicas, los cuales, fueron aplicados en los escenarios, como IPv4, OSPF, MPLS y QoS.

Sin embargo, para registrar el tiempo que tarda en aplicar las tareas de manera más precisa, cada playbook por función se ejecutará 20 veces, para calcular el tiempo promedio de ejecución y desviación estándar muestral.

Por tanto, con los valores de tiempo obtenidos se puede analizar si ejecutando por funciones mejora la estabilidad, rendimiento y eficiencia operativa en el proyecto de Ansible con mayor cantidad de tareas y dispositivos.

Inicialmente, para evidenciar la ejecución del playbook por función IPv4 se registra un tiempo de referencia en la Figura 141, donde indica que tarda 45.62 segundos (46 segundos), lo cual es un tiempo muy bajo, ya que aplica tareas específicas para cumplir la configuración IPv4.

Figura 141*Tiempo de referencia en ejecución del playbook por función IPv4*

```

PLAY RECAP *****
CE1      : ok=3    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
CE2      : ok=3    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
CE3      : ok=3    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
P1       : ok=2    changed=1    unreachable=0    failed=0    skipped=3    rescued=0    ignored=0
P2       : ok=2    changed=1    unreachable=0    failed=0    skipped=3    rescued=0    ignored=0
P3       : ok=2    changed=1    unreachable=0    failed=0    skipped=3    rescued=0    ignored=0
P4       : ok=2    changed=1    unreachable=0    failed=0    skipped=3    rescued=0    ignored=0
PE1      : ok=2    changed=1    unreachable=0    failed=0    skipped=3    rescued=0    ignored=0
PE2      : ok=2    changed=1    unreachable=0    failed=0    skipped=3    rescued=0    ignored=0
PE3      : ok=2    changed=1    unreachable=0    failed=0    skipped=3    rescued=0    ignored=0
multi_server : ok=0    changed=0    unreachable=0    failed=0    skipped=5    rescued=0    ignored=0

=====
TIEMPO TOTAL EJECUCION DE FUNCION IPV4: 46 segundos
=====

real    0m45.620s
user    0m48.657s
sys     0m4.891s
root@Ansible-1:~/anisble#

```

A continuación, en la Tabla 46, se muestran los tiempos obtenidos en las 20 iteraciones de ejecución del playbook por función, dichos valores son usados en la (Ec. 1 y (Ec. 2 para calcular el tiempo promedio y desviación estándar, donde se obtiene 46.25 segundos y aproximadamente 0.85 segundos, respectivamente, por tanto se demuestra buen rendimiento y baja dispersión del tiempo al ejecutar tareas específicas de forma automatizada.

Tabla 46*Tiempo total promedio de ejecución del playbook por función IPv4*

Iteración	Tiempo en segundos	$(t_i - \bar{t})^2$
1	46 seg	0.0625
2	47 seg	0.5625
3	47 seg	0.5625
4	46 seg	0.0625
5	46 seg	0.0625
6	48 seg	3.0625
7	46 seg	0.0625

8	45 seg	1.5625
9	46 seg	0.0625
10	46 seg	0.0625
11	46 seg	0.0625
12	46 seg	0.0625
13	46 seg	0.0625
14	45 seg	1.5625
15	45 seg	1.5625
16	46 seg	0.0625
17	48 seg	3.0625
18	47 seg	0.5625
19	47 seg	0.5625
20	46 seg	0.0625
- Sumatoria del tiempo		925 / 20
- Tiempo Promedio ($\bar{t}$)		$\bar{t} = 46.25$
- Sumatoria de $(t_i - \bar{t})^2$		$\sqrt{13.75/19}$
- Desviación estándar (σ)		$s = 0.85$

A continuación, se toma un tiempo referente que tarda en aplicar las configuraciones el playbook por función OSPF, el cual registra aproximadamente 55 segundos como se muestra en la Figura 142. Destacando el rápido despliegue de configuraciones para el enrutamiento OSPF.

Figura 142*Tiempo de referencia en ejecución del playbook por función OSPF*

```

PLAY RECAP *****
CE1      : ok=2    changed=2    unreachable=0    failed=0    skipped=5    rescued=0    ignored=0
CE2      : ok=2    changed=2    unreachable=0    failed=0    skipped=5    rescued=0    ignored=0
CE3      : ok=2    changed=2    unreachable=0    failed=0    skipped=5    rescued=0    ignored=0
P1       : ok=5    changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
P2       : ok=5    changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
P3       : ok=5    changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
P4       : ok=5    changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
PE1      : ok=5    changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
PE2      : ok=5    changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
PE3      : ok=5    changed=1    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
multi_server : ok=0    changed=0    unreachable=0    failed=0    skipped=7    rescued=0    ignored=0

=====
TIEMPO TOTAL EJECUCION DE FUNCION OSPF: 55 segundos
=====

real    0m54.887s
user    1m17.449s
sys     0m7.473s

```

Luego, se presenta los valores registrados en las 20 iteraciones de ejecución del playbook por función OSPF para calcular a partir de la (Ec. 1) y (Ec. 2) el promedio y desviación estándar respectivamente, como se muestra en la Tabla 47.

Como resultado se obtiene que en cada ejecución secuencial del playbook tiende a tardar aproximadamente 1 segundos más o menos respecto al tiempo promedio de 55 segundos.

Por lo tanto, los tiempos de ejecución no tienen variaciones significativas, sin embargo, se demuestra un incremento pequeño en el tiempo respecto al anterior, debido al aumento de tareas a configurar.

Tabla 47*Tiempo total promedio de ejecución del playbook por función OSPF*

Iteración	Tiempo en segundos	$(t_i - \bar{t})^2$
1	55	0.0025
2	54	1.1025
3	55	0.0025
4	58	8.7025
5	55	0.0025

6	55	0.0025
7	55	0.0025
8	56	0.9025
9	57	3.8025
10	56	0.9025
11	55	0.0025
12	54	1.1025
13	54	1.1025
14	54	1.1025
15	55	0.0025
16	55	0.0025
17	54	1.1025
18	55	0.0025
19	55	0.0025
20	54	1.1025
-Sumatoria del tiempo		1101 / 20
- Tiempo Promedio ($\bar{t}$)		$\bar{t} = 55.05$
- Sumatoria de $(t_i - \bar{t})^2$		$\sqrt{20.95/19}$
- Desviación estándar (σ)		$s = 1.05$

Posteriormente, en la Figura 143 se indica un tiempo de referencia registrado de 35.80 segundos (36 segundos aproximadamente), el cual corresponde al tiempo de ejecución del playbook por función MPLS. Por tanto, al tener tareas específicas de conmutación de etiquetas existe una menor carga de procesamiento en Ansible.

Figura 143*Tiempo de referencia en ejecución del playbook por función MPLS*

```

PLAY RECAP *****
CE1      : ok=0    changed=0    unreachable=0    failed=0    skipped=6    rescued=0    ignored=0
CE2      : ok=0    changed=0    unreachable=0    failed=0    skipped=6    rescued=0    ignored=0
CE3      : ok=0    changed=0    unreachable=0    failed=0    skipped=6    rescued=0    ignored=0
P1       : ok=6    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
P2       : ok=6    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
P3       : ok=6    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
P4       : ok=6    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PE1      : ok=6    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PE2      : ok=6    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PE3      : ok=6    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
multi_server : ok=0    changed=0    unreachable=0    failed=0    skipped=6    rescued=0    ignored=0

=====
TIEMPO TOTAL EJECUCION DE FUNCION MPLS: 36 segundos
=====

real    0m35.805s
user    0m46.980s
sys      0m4.705s
root@Ansible-1:~/anisble#

```

Los tiempos registrados en las 20 iteraciones de la ejecución del playbook por función MPLS se muestra en la Tabla 48, a partir de aquello se calcula con la (Ec. 1) y (Ec. 2) el tiempo promedio de 35.65 segundos y la desviación estándar de 1.07 segundos, es decir, en cada ejecución secuencial tiende a tardar aproximadamente 1 segundo más o menos respecto al tiempo promedio.

Tabla 48*Tiempo total promedio de ejecución del playbook por función MPLS*

Iteración	Tiempo en segundos	$(t_i - \bar{t})^2$
1	35	0.4225
2	35	0.4225
3	39	11.2225
4	36	0.1225
5	35	0.4225
6	35	0.4225
7	34	2.7225

8	36	0.1225
9	36	0.1225
10	37	1.8225
11	36	0.1225
12	36	0.1225
13	35	0.4225
14	36	0.1225
15	35	0.4225
16	35	0.4225
17	36	0.1225
18	35	0.4225
19	35	0.4225
20	36	0.1225
-Sumatoria del tiempo	713 / 20	
- Tiempo Promedio ($\bar{t}$)	$\bar{t} = 35.65$	
- Sumatoria de $(t_i - \bar{t})^2$		$\sqrt{21.9/19}$
- Desviación estándar (σ)		$s = 1.07$

Por último, se considera el playbook con configuraciones específicas de QoS, el cual tarda un tiempo de referencia de 48 segundos en aplicar las tareas particulares sobre políticas de calidad de servicio, reflejando una mayor eficiencia, como se indica en la Figura 150.

Figura 144*Tiempo de referencia en ejecución del playbook por función QoS*

```

PLAY RECAP *****
PE1      : ok=17  changed=17  unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PE2      : ok=17  changed=17  unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
PE3      : ok=17  changed=17  unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

=====
TIEMPO TOTAL EJECUCION DE FUNCION QoS: 48 segundos
=====

real    0m48.021s
user    0m37.874s
sys     0m3.709s
root@Ansible-1:~/ansible# █

```

A continuación, se indica en la Tabla 49 el tiempo registrado en las 20 iteraciones de la ejecución del playbook por función QoS, con estos valores se calcula el tiempo promedio y la desviación estándar, considerando la (Ec. 1) y (Ec. 2) respectivamente.

Por tanto se indica que Ansible se demora aproximadamente 4 segundos más o menos respecto al tiempo promedio de 43.9 segundos para completar la automatización de configuraciones específicas de calidad de servicio, entonces, se evidencia estabilidad y rendimiento alto por parte de la herramienta IaC.

Tabla 49*Tiempo total promedio de ejecución del playbook por función QoS*

Iteración	Tiempo en segundos	$(t_i - \bar{t})^2$
1	48	16.81
2	41	8.41
3	48	16.81
4	43	0.81
5	41	8.41
6	41	8.41
7	42	3.61
8	40	15.21

9	41	8.41
10	41	8.41
11	43	0.81
12	41	8.41
13	40	15.21
14	41	8.41
15	42	3.61
16	41	8.41
17	40	15.21
18	41	8.41
19	41	8.41
20	42	3.61
-Sumatoria del tiempo	878 / 20	
- Tiempo Promedio ($\bar{t}$)	$\bar{t} = 43.9$	
- Sumatoria de $(t_i - \bar{t})^2$	$\sqrt{181.4/19}$	
- Desviación estándar (σ)	$s = 3.09$	

Los valores obtenidos del tiempo de ejecución de playbooks según su función, son recopilados en la Tabla 50, también se detalla la cantidad de tareas de cada archivo de configuración y el número de equipos. En dicha tabla se evidencia que el tiempo de ejecución varía según la cantidad de tareas y el número de dispositivos a configurar, por tanto, tomará más tiempo aplicar mayor cantidad de configuraciones, sin embargo el rendimiento de Ansible es estable, ya que la diferencia y variación de los tiempos es proporcional a la cantidad de tareas y equipos configurados en cada función.

Además, se analiza si el despliegue de configuraciones por función presenta mayor rendimiento y eficiencia operativa respecto a los resultados de la Tabla 40 correspondiente al tiempo promedio de la ejecución de los playbooks por escenario para configurar de forma conjunta.

Tabla 50

Tiempo total promedio de ejecución de playbooks por función

Playbook por función	Número de equipos	Número de tareas	Tiempo de ejecución de playbook por función	Variación de tiempo de ejecución
IPv4	10	7 tareas	46.25 segundos	0.85 seg
OSPF	10	9 tareas	55.05 segundos	1.05 seg
MPLS	7	7 tareas	35.65 segundos	1.07 seg
QoS	3	17 tareas	43.9 segundos	3.09 seg

Por otro lado, al analizar los valores obtenidos de la Tabla 50 y compararlos con los datos de la Tabla 40, se evidencia que es posible mejorar el rendimiento y mantener estabilidad en el proceso de automatización mediante la configuración de playbooks separados por funciones, ya que su tiempo de ejecución es menor y con bajas variaciones debido al número de tareas reducido por operación.

Además, al tener archivos de configuración separados, Ansible puede ejecutar únicamente el playbook que contiene el cambio a aplicar, lo cual aporta eficiencia operativa dado que, evita ejecutar tareas ya aplicadas, reduciendo el procesamiento.

Finalmente, otro aspecto que mejora la configuración mediante playbooks por función, es la mantenibilidad del proyecto, ya que, facilita la gestión, identificación y corrección de errores, además es posible modificar o corregir las tareas de configuración, por tanto todos los

aspectos mencionados, contribuye a la optimización del proceso de automatización especialmente en redes con mayor cantidad de dispositivos.

Conclusiones y Recomendaciones

Conclusiones

La implementación de una Infraestructura como Código (IaC) mediante el uso de una herramienta de automatización de configuraciones de red en infraestructuras tradicionales, ha demostrado ser eficiente para la administración de redes mediante código estructurado, ya que se gestiona a través de automatización centralizada las configuraciones de múltiples dispositivos de red, lo cual reduce la intervención manual, tiempo de configuración, probabilidad de errores y el esfuerzo del administrador, por tanto, se obtiene infraestructuras de red confiables, consistentes, organizadas y escalables, que optimizan la administración operativa en redes de datos tradicionales.

La herramienta de Infraestructura como Código elegida fue Ansible, ya que es ideal para la automatización de configuraciones en dispositivos de red, debido a su alta compatibilidad con protocolos y tecnologías implementados en los escenarios como OSPF, MPLS, VPNs, BGP, VLANs, entre otros, así como la capacidad de gestionar equipos de diferentes fabricantes como Cisco y Mikrotik, además garantiza seguridad, ya que usa protocolos seguros como SSH que permite autenticar el acceso de dispositivos y transmitir de manera confiable la información. También, Ansible tiene una curva de aprendizaje sencilla, ya que se escribe los playbooks con formato YAML y plantilla jinja2 para generar configuraciones dinámicas, por último, otro factor, es que no requiere de configuraciones adicionales como instalación de agentes en los nodos a gestionar, por tanto Ansible ayuda a comprender el funcionamiento de una herramienta IaC y obtener una infraestructura de red basada en código.

La herramienta de simulación seleccionada para alojar infraestructuras de red tanto pequeñas como robustas, corresponde a GNS3 debido a su versatilidad para simular escenarios de estudio y asemejarlos a entornos reales, asimismo posee alta compatibilidad para ejecutar imágenes reales de dispositivos de diferentes fabricantes, además es posible integrar la

herramienta de automatización Ansible, ya sea instalada en una máquina virtual o alojada en el contenedor Network Automation para ofrecer una infraestructura como código en un entorno controlado.

En el entorno de simulación GNS3 se plantearon tres topologías de red, las cuales se basan en arquitecturas comúnmente usadas en entornos empresariales pequeños y medianos, dichos escenarios de simulación tienen niveles de complejidad progresivos, ya que se aumenta la cantidad de dispositivos y las tareas de configuración, lo que permitió estudiar y evaluar el comportamiento de Ansible con cargas operativas diferentes.

La configuración de los tres escenarios de simulación se requiere inicialmente una intervención manual, que corresponde con asignar la dirección IP perteneciente a una red de gestión y habilitar SSH en cada equipo, y así garantizar que Ansible alcance a los nodos secundarios. Posteriormente, se estructura el proyecto de automatización, el cual se constituye por inventario, archivos de variables y playbooks en formatos YAML, por lo tanto se consigue un proyecto ordenado, reutilizable y escalable que asegura la gestión centralizada de configuraciones desde el nodo de control hacia los distintos dispositivos de red.

Se evalúa los resultados de la gestión centralizada mediante la ejecución de los playbooks desde Ansible en los tres escenarios, donde se analizó que el tiempo de configuración automatizada es significativamente bajo respecto a configuraciones manuales, ya que, con Ansible optimiza el tiempo de configuración entre un 67 % y 82% según el escenario, debido a que mientras más extensa es la red existe mayor ahorro de tiempo, lo cual aporta eficiencia operativa en el proceso. Además, la variación del tiempo durante la ejecución de los playbooks es inferior al 4%, por tanto demuestra buen uso de recursos, rendimiento, escalabilidad y estabilidad al implementar una herramienta IaC para la gestión de infraestructuras de redes.

La ejecución de playbooks separados por funciones optimiza el proceso de automatización, especialmente en redes con mayor cantidad de equipos, ya que reduce tanto el tiempo de configuración a valores inferiores de 1 minuto y la variación de ejecución se reduce al 3% porque se aplica tareas específicas por operación, por lo tanto se obtiene mayor rendimiento y estabilidad en el sistema, además aporta mantenibilidad al proyecto porque facilita la gestión, modificación y corrección de tareas de configuración, por ende los playbooks segmentados se considera como una estrategia para la optimización de configuraciones en Infraestructuras como Código (IaC).

Recomendaciones

Con los resultados obtenidos en el presente trabajo, se recomienda el uso de Infraestructura como Código (IaC) mediante la integración de una herramienta de automatización como Ansible en infraestructuras de datos tradicionales, sobre todo en entornos reales de redes empresariales de baja y mediana extensión, así como en redes de crecimiento, ya que, pueden reducir configuraciones manuales repetitivas que ocasionan alta demanda de tiempo en la administración, por ello, los resultados demuestran que la gestión automatizada optimiza de manera significativa los tiempos de configuración en dispositivos de red, también garantiza estabilidad en sus operaciones y minimiza errores operativos.

En una Infraestructura como Código, la herramienta Ansible no se limita únicamente a la configuración de dispositivos, por ende se recomienda expandir la gestión automatizada hacia otros elementos que conforma una infraestructura de red, como: servidores para configurar su estado y agregar servicios; firewalls para automatizar sus políticas de seguridad; plataformas de monitoreo para configurar agentes, alertas, umbrales, entre otros para controlar el estado de la red. Además, se debe considerar integrar un sistema de control de versiones como Git, para llevar un historial de cambios, ya que, aporta al control de configuraciones, recuperación de versiones estables ante fallos con el fin de aplicar los playbooks correctos de configuración.

Se recomienda, en escenarios empresariales reales el uso de Ansible comercial, llamado Red Hat Ansible Automation Platform, ya que ofrece avanzadas funcionalidades, como disponibilidad de una interfaz gráfica que permite el manejo de los playbooks, proporciona el control de acceso de usuarios, el registro y auditoría de ejecuciones, además ofrece soporte técnico oficial de Red Hat. Esto aporta para mejorar la administración de redes de gran tamaño, trazabilidad y gestión centralizada desde el nodo de control para la automatización de configuraciones.

Para casos de estudio o laboratorios que requieran implementar infraestructuras como código, la plataforma de simulación GNS3 es recomendable, ya que es posible trabajar con herramientas de automatización para aplicar configuraciones dinámicas y no usar solo métodos convencionales, además, tiene alta compatibilidad con diferentes componentes de red, no requiere de alto consumo de recursos, y simula redes complejas, las cuales pueden ser mixtas con el fin de probar configuraciones de forma segura antes de ser aplicadas a entornos en producción.

Se recomienda, realizar validaciones previas a la ejecución de los playbooks de configuración como, verificar conexión del nodo de control con todos los nodos secundarios, comprobar sintaxis en archivos yaml y revisar la definición correcta de variables tanto generales como específicas, con esto es posible la detección temprana de errores y así obtener estabilidad y confiabilidad en el proceso de configuración automatizada.

Por último, se sugiere mantener una estructura organizada en el proyecto de automatización con Ansible, con la definición de grupos de dispositivos en los inventarios, creación de archivos de variables y playbooks separados por funciones específicas, sin importar el tamaño de la infraestructura de red, ya que, así garantiza la facilidad de interpretar las configuraciones, la escalabilidad, mantenibilidad y reducir la carga de procesamiento del proyecto, sin afectar la gestión centralizada de configuraciones en Infraestructuras como Código.

Bibliografía

- Agapidis, L. (2024). *Comparación de GNS3 vs EVE-NG vs Packet Tracer para simulación de redes*. Networks Training. <https://www.networkstraining.com/gns3-vs-eve-ng-vs-cisco-packet-tracer/>
- Aguilar, R. (2022). *RED CONVERGENTE DE DATOS, VOZ Y VIDEO VIGILANCIA BAJO EL PROTOCOLO IP PARA EL HOSPITAL IESS – IBARRA ADMINISTRADO CON SOFTWARE LIBRE* [INGENIERÍA EN ELECTRÓNICA Y REDES DE COMUNICACIÓN, UNIVERSIDAD TÉCNICA DEL NORTE].
<https://repositorio.utn.edu.ec/bitstream/123456789/11930/2/04%20RED%20280%20TRABAJO%20GRADO.pdf>
- Ansible. (2025, January 10). *Installing Ansible — Ansible Community Documentation*.
Installing Ansible.
https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html
- Arista Networks. (2015). *REPORTE TECNICO DE ARISTA*.
<https://www.arista.com/assets/data/pdf/Whitepapers/AristaAdvantage-Spanish.pdf>
- Arzo, S. T., Naiga, C., Granelli, F., Bassoli, R., Devetsikiotis, M., & Fitzek, F. H. P. (2021). A Theoretical Discussion and Survey of Network Automation for IoT: Challenges and Opportunity. *IEEE Internet of Things Journal*, 8(15), 12021–12045.
<https://doi.org/10.1109/JIOT.2021.3075901>
- AWS CloudFormation. (2023). *Aprovisionamiento de infraestructura como código - AWS CloudFormation - AWS*. Características AWS CloudFormation.
<https://aws.amazon.com/es/cloudformation/>
- Ben-Kiki, O., Evans, C., & Ingerson, B. (2009). *YAML Ain't Markup Language (YAML™) Version 1.1 Working Draft 2004-12-28*. 5(11). <https://yaml.org/spec/history/2004-12-28/2004-12-28.pdf>

- Blanchet, R., Pérez, S., & Facchini, H. (2021). Estudio y simulación de redes definidas por software y automatización de red. *XXIII Workshop de Investigadores En Ciencias de La Computación, December*, 18–23. <http://sedici.unlp.edu.ar/handle/10915/120006>
- Blauw, F. (2018). The Use of Git as Version Control in the South African Software Engineering Classroom | IEEE Conference Publication | IEEE Xplore. *Conferencia de La Semana IST-África 2018 (IST-África)*, 1–8.
<https://ieeexplore.ieee.org/document/8417196/authors#authors>
- Bone Andrade, M. F., Rodríguez Vizuite, J. D., Sosa Calero, S. M., & Freire Núñez, L. A. (2021). Perspectivas de aplicación e investigación en Software Defined Networking SDN. *ConcienciaDigital*, 4(1), 121–133.
<https://doi.org/10.33262/concienciadigital.v4i1.1538>
- Bosch, R. (2020). *Proyecto de automatización de una infraestructura web* [Grado de Ingeniería Informática, Universitat de les Illes Balears].
<http://hdl.handle.net/11201/152109>
- Bruschetti, F. S., Guevara, J., Abeledo, M. C., & Priano, D. A. (2023). An Empirical Evaluation of Automated Configuration Tools for Software-Defined Networking: A Usability and Performance Perspective. *Ingenierie Des Systemes d'Information*, 28(5), 1127. <https://doi.org/10.18280/ISI.280502>
- Bustos, S. (2023). “DESARROLLO DE UN PLAN DE MIGRACIÓN DE UNA RED DE ÁREA EXTENSA DE UN PROVEEDOR DE SERVICIOS DE INTERNET A UNA RED DEFINIDA POR SOFTWARE SD-WAN” [INGENIERO EN ELECTRÓNICA Y REDES DE COMUNICACIÓN]. UNIVERSIDAD TÉCNICA DEL NORTE.
- Caisalitin, L., & Quillupangui, J. (2015). *USO DE LAS NUEVAS TECNOLOGÍAS DE LA INFORMACIÓN POR PARTE DE LAS SECRETARIAS DE LAS EMPRESAS*

PÚBLICAS Y PRIVADAS DEL SECTOR URBANO DEL CANTÓN SALCEDO.

UNIVERSIDAD TÉCNICA DE COTOPAXI.

Caisatoa, M. (2023). *DISEÑO Y SIMULACIÓN DE UNA RED LOCAL CONECTADA A PROVEEDORES DE SERVICIOS DE INTERNET (ISP) CON POLÍTICAS DE SEGURIDAD Y AUTOMATIZACIÓN DE SUS CONFIGURACIONES* [PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR].

<https://repositorio.puce.edu.ec/handle/123456789/43273>

Camas Mainato, W. X., & Riera Taza, H. X. (2022). *Diseño e implementación de un proveedor de infraestructura en la nube INP, mediante segmentación de recursos.*

<http://dspace.ups.edu.ec/handle/123456789/23990>

Cisco. (2005). *Enterprise QoS Solution Reference Network Design Guide.*

<https://community.cisco.com/legacyfs/online/legacy/3/2/6/64623-qossrnd.pdf>

Cisco. (2020). *Design Zone - Campus LAN and Wireless LAN Solution Design Guide - Cisco.* Solutions Design Zone.

<https://www.cisco.com/c/en/us/td/docs/solutions/CVD/Campus/cisco-campus-lan-wlan-design-guide.html?dtid=ossdc000283&linkclickid=srch>

Cisco. (2021). Small Enterprise Design Profile (SEDP)-Network Foundation Design. *Cisco.*

http://www.cisco.com/en/US/docs/solutions/Enterprise/Campus/HA_recovery_DG/ca

Cisco. (2025). *Cisco: Software, Network, and Cybersecurity Solutions - Cisco.* Cisco - Global Home Page. <https://www.cisco.com/>

Collahuazo, V. D. C. (2016). *DISEÑO DE UN SISTEMA DE TELEFONÍA IP BASADO EN SOFTWARE LIBRE ENTRE LA COOPERATIVA DE AHORRO Y CRÉDITO SAN ANTONIO LTDA Y SUS SUCURSALES* [INGENIERA EN ELECTRÓNICA Y REDES DE COMUNICACIÓN, UNIVERSIDAD TÉCNICA DEL NORTE].

<https://repositorio.utn.edu.ec/handle/123456789/7015>

- Conde Gil, A. M. (2021). *Automatización de redes informáticas con Python* [Universidad Oberta de Catalunya]. <http://hdl.handle.net/10609/126926>
- Cuzme-Rodríguez, F., Umaquina-Criollo, A., Suárez-Zambrano, L., Farinango-Endara, H., Domínguez-Limaico, H., & Mediavilla-Valverde, M. (2020). Simulation Tools for Solving Engineering Problems. Case Study. *Communications in Computer and Information Science*, 1193 CCIS, 271–285. https://doi.org/10.1007/978-3-030-42517-3_21
- Dominguez-Quintero, L., & Vargas-Lombardo, M. (2021). Herramientas de infraestructura como código: ansible, terraform, chef, puppet. *Periodicidad: Semestral*, 17(2), 1–6. <http://portal.amelica.org/ameli/jatsRepo/339/3392429002/index.html>
- Dutta, A. (2023, March 16). *Ansible Architecture | Simple Architecture of Ansible*. EDUCBA. <https://www.educba.com/ansible-architecture/>
- Edelman, J. (2015). Network Automation with Ansible - Dynamically Configuring Interface Descriptions. *JDELMAN*. <https://jedelman.com/home/network-automation-with-ansible-dynamically-configuring-interface-descriptions/>
- Edelman, J., Lowe, S., & Oswalt, M. (2018). *Network programmability and automation : skills for the next-generation network engineer*. (Courtney Allen, Ed.; O'Reilly Media). https://www.google.com.ec/books/edition/Network_Programmability_and_Automation/QjJKDwAAQBAJ?hl=es&gbpv=1
- Enciso, L., & Morales-Iñiguez, C. (2021). Ansible una estrategia de administración y configuración automatizada sobre GNS3 con OSPF para redes empresariales medianas. *Revista Ibérica de Sistemas e Tecnologías de Informação*, 239–252.
- Eriksson, M., & Hallberg, V. (2011). *Comparison between JSON and YAML for data serialization*.

- EVE-NG. (2025). *Documentation - EVE-NG*. Documentation - EVE-NG. <https://www.eve-ng.net/index.php/documentation/>
- Froehlich, A., & Scarpatti, J. (2022). *What is network automation?* TechTarget. <https://www.techtarget.com/searchnetworking/definition/network-automation>
- GNS3. (2016). *Getting Started with GNS3*. GNS3 Documentation. <https://docs.gns3.com/docs/getting-started/installation/windows>
- GNS3. (2025). *Appliances | Marketplace | GNS3*. <https://gns3.com/marketplace/appliances>
- Greenfield, P., Droettboom, M., & Bray, E. (2015). ASDF: A new data format for astronomy. *Astronomy and Computing*, 12, 240–251. <https://doi.org/10.1016/J.ASCOM.2015.06.004>
- Hernández, M., Vargas, I., & Díaz, A. (2017). Transferencia y validación de datos utilizando el formato XML. Aplicación en la Web Azucarera. *Instituto Cubano de Investigaciones de Los Derivados de La Caña de Azúcar*, 51(3), 13–18. <http://www.redalyc.org/articulo.oa?id=223158039003>
- Huawei. (2024). *Instalación de eNSP - Comunidad Huawei Enterprise*. Foro de Tecnología-Instalación de ENSP. <https://forum.huawei.com/enterprise/intl/es/thread/Instalaci%C3%B3n-de-eNSP/829642255819472896?blogId=829642255819472896>
- Huerlo, J. (2021). *Terraform Como Herramienta Para Automatizar La Creación De Infraestructuras Siguiendo El Concepto “Infraestructura Como Código”* [PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR SEDE ESMERALDAS]. <https://repositorio.puce.edu.ec/handle/123456789/38008>
- IPCisco. (2020). *Puppet Software | Network Automation Tool | How Puppet Works ★ IpCisco*. IPCisco. https://ipcisco.com/lesson/puppet-overview/#Puppet_Software

ISO 25010. (2024). *ISO/IEC 25000*. <https://iso25000.com/index.php/normas-iso-25000/iso-25010>

Jackson, C., Gooley, J., Iliesiu, A., & Malegaonkar, A. (2020). *Cisco Certified DevNet Associate DEVASC 200-901 Official Cert Guide*. Cisco.
https://www.google.com.ec/books/edition/Cisco_Certified_DevNet_Associate_DEVASC/wNfPEAAAQBAJ?hl=es&gbpv=0

Jacobs, I., & Walsh, N. (2004). Architecture of the World Wide Web, Volume One. *W3C*, 1.
<https://www.w3.org/TR/2004/REC-webarch-20041215/#doc-audience>

Jingo Cevallos, I. M. (2024). Desarrollo de un diseño de red de área extensa basada en una arquitectura definida por software Sd-Wan adaptada a un modelo MPLS e implementación de políticas de calidad de servicio qos para el análisis del rendimiento de una red híbrida [Tesis Ingeniería, Universidad Técnica del Norte]. In *Universidad Técnica del Norte*. <http://repositorio.utn.edu.ec/handle/123456789/15698>

Josheff, D., & Márquez, J. (2015). EXPERIMENTAL DESIGN OF A QUALITY OF SERVICE MODEL FOR NEXT GENERATION NETWORKS. *Investigación e Innovación En Ingenierías*, 2(3), 46–54. <http://www.opnet.com>

Juniper Networks. (2024, March 19). *Hipervisores y versiones compatibles | Juniper Networks*. Documentation .
<https://www.juniper.net/documentation/us/en/software/apstra4.2/apstra-install-upgrade/topics/ref/apstra-server-platforms.html>

Kumaran, N., Ramya, H., & Apoorva, V. (2022). Chef Automation on Google Cloud. *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, 10(III), 2321–9653.
<https://doi.org/https://doi.org/10.22214/ijraset.2022.40909>

- Lozada, J. (2022). *Implementación de una red híbrida SDN/MPLS para la gestión de servicios diferenciados mediante simulación de tiempo real GNS3* [Tesis de Ingeniería, Universidad Técnica del Norte]. <http://repositorio.utn.edu.ec/handle/123456789/13203>
- Luna, S. (2024). *04 RED 376 TRABAJO GRADO* [TÍTULO DE INGENIERÍA EN ELECTRÓNICA Y REDES DE COMUNICACIÓN, UNIVERSIDAD TÉCNICA DEL NORTE].
<https://repositorio.utn.edu.ec/bitstream/123456789/15580/2/04%20RED%20376%20TRABAJO%20GRADO.pdf>
- Malin, V. (2011). *Comparison between JSON and YAML for data serialization* [School of Computer Science and Engineering].
<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=636a2b04d98c0af8e9d6f59148352dd63af4f0c1>
- Martel, L. (2021). *Automatización de procesos para el registro de facturas en formato XML en una base de datos mediante Python* [Universidad Nacional Mayor de San Marcos].
<https://hdl.handle.net/20.500.12672/17709>
- Mikrotik. (2025). *RouterOS*. RouterOS - MikroTik Documentation.
<https://help.mikrotik.com/docs/spaces/ROS/pages/328059/RouterOS>
- MINTEL. (2022). *AGENDA DE TRANSFORMACIÓN DIGITAL DEL ECUADOR 2022-2025. Ministerio de Telecomunicaciones y de La Sociedad de La Información*.
- Morris, K. (2016). *Infrastructure as code : managing servers in the cloud*. O'Reilly Media.
<https://books.google.com.ec/books?id=BIhRDAAAQBAJ>
- Morris, K. (2021). *Infrastructure as Code Dynamic Systems for the Cloud Age* (2 nd Edition). O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.
<https://dl.ebooksworld.ir/books/Infrastructure.as.Code.2nd.Edition.Kief.Morris.OReilly.9781098114671.EBooksWorld.ir.pdf>

ONU. (2023). The-Sustainable-Development-Goals-Report-2023_Spanish. *Naciones Unidas*.

Oswalt, M., Adell, C., Lowe, S. S., & Edelman, J. (2022). *Network Programmability and Automation* (O'Reilly Media, Ed.; 2nd ed.).

https://www.google.com.ec/books/edition/Network_Programmability_and_Automation/cEnREAAQBAJ?hl=es&gbpv=0

Pacheco, X. (2022). *Análisis de herramientas de software para la simulación y emulación de redes de computadores para aprendizaje en entornos de laboratorio* [Ingeniería en Sistemas, Universidad Técnica de Machala].

<http://repositorio.utmachala.edu.ec/handle/48000/18760>

Pérez, S., Facchini, H., Dantiacq, A., Roberti, B., & Hidalgo, F. (2021). *Plataformas de Automatización de Red*. <https://sedici.unlp.edu.ar/handle/10915/120006>

Pompa, F. (2022). XML, JSON y el intercambio de información. *Anuario Ciencia En La UNAH*, 18(3). <https://revistas.unah.edu.cu/index.php/ACUNAH/article/view/1532>

Progress Chef. (2025). *Herramientas de automatización de infraestructura - Chef Infrastructure Management | Chef*. Gestión de Infraestructura Del Chef.

<https://www.chef.io/products/chef-infrastructure-management>

Rahman, A., & Parnin, C. (2023). Detecting and Characterizing Propagation of Security Weaknesses in Puppet-Based Infrastructure Management. *IEEE Transactions on Software Engineering*, 49(6), 3536–3553. <https://doi.org/10.1109/TSE.2023.3265962>

RedHat. (2022, June 21). *¿Qué es Ansible y cómo funciona?* RedHat.

<https://www.redhat.com/es/topics/automation/learning-ansible-tutorial#funcionamiento>

Ríos, R. A., & Ríos, R. A. (2016). Conceptualización de SDN y NFV. *Maskay*, 6(1), 29–34.

http://scielo.senescyt.gob.ec/scielo.php?script=sci_arttext&pid=S1390-67122016000100029&lng=es&nrm=iso&tlng=es

Romero, D. (2021). *Infraestructura como código* [Universidad Oberta de Catalunya].

<http://hdl.handle.net/10609/132647>

Santoso, B., & Sari, M. (2022). IMPROVEMENT OF SETUP TIME ON SERVER INFRASTRUCTURE AUTOMATION USING ANSIBLE FRAMEWORK. *Setup Time on Server Infrastructure Automation Using. ... 3661 Journal of Engineering Science and Technology*, 17(5), 3660–3671.

Silva, K. (2021). XML. El lenguaje de comunicación universal. *Nextia*, 6, 26–29.

<https://doi.org/10.1145/3132269>

Spinellis, D. (2012). Git. *IEEE Software*, 29(3), 100–101.

<https://doi.org/10.1109/MS.2012.61>

Srinivash, Y., & Sivapunniyam, R. (2016). *Implementation of MPLS L2.5 VPN in an Enterprise Network Environment*.

Tandazo, J. C. (2022). *AUTOMATIZACIÓN DE REDES UTILIZADAS PARA EOT AUTOMATIZACIÓN DE REDES CON NETMIKO* [ESCUELA POLITÉCNICA NACIONAL]. <https://bibdigital.epn.edu.ec/handle/15000/21581>

Terraform Community. (2025). *Infrastructure as code*. Terraform by HashiCorp.

<https://www.terraform.io/use-cases/infrastructure-as-code>

Venkatesh, P. (2018). *Network Automation: Road to Programmable Networks - ONUG*.

ONUG. <https://onug.net/blog/network-automation-road-to-programmable-networks/>

Vuorre, M., & Curley, J. P. (2018). Curating Research Assets: A Tutorial on the Git Version Control System. *Advances in Methods and Practices in Psychological Science*, 1(2),

219–236. <https://doi.org/10.1177/2515245918754826>

Wågbrant, S., & Dahlén Radic, V. (2022). *Automated Network Configuration: A Comparison Between Ansible, Puppet, and SaltStack for Network Configuration* [Mälardalen

University]. <https://urn.kb.se/resolve?urn=urn:nbn:se:mdh:diva-58886>

Walton, A. (2020, November 9). *Formato de Datos*. CCNA Desde Cero.

<https://ccnadesdecero.es/formato-datos/>

Yang, Y. (2012). Impact data-exchange based on XML. *ICCSE 2012 - Proceedings of 2012 7th International Conference on Computer Science and Education*, 1147–1149.

<https://doi.org/10.1109/ICCSE.2012.6295268>

Yunga, A. (2018). *Implementación del provisionamiento automático de configuraciones (Network Automation) en infraestructuras multivendor con Ansible*. [Tesis de Ingeniería, Escuela Superior Politécnica de Chimborazo].

<http://dspace.esPOCH.edu.ec/handle/123456789/10935>

Zammar, C. (2024, April 22). *Transitioning from Chef Cookbooks to Ansible Playbooks: A seamless journey*. Red Hat. <https://www.redhat.com/es/blog/transitioning-chef-cookbooks-ansible-playbooks-seamless-journey>

Zhang, Y., Rahman, M., Wu, F., & Rahman, A. (2023). Quality Assurance for Infrastructure Orchestrators: Emerging Results from Ansible. *Proceedings - IEEE 20th International Conference on Software Architecture Companion, ICSA-C 2023*, 322–324.

<https://doi.org/10.1109/ICSA-C57050.2023.00073>

Zolkifli, N. N., Ngah, A., & Deraman, A. (2018). Version Control System: A Review. *Procedia Computer Science*, 135, 408–415.

<https://doi.org/10.1016/J.PROCS.2018.08.191>

Anexos

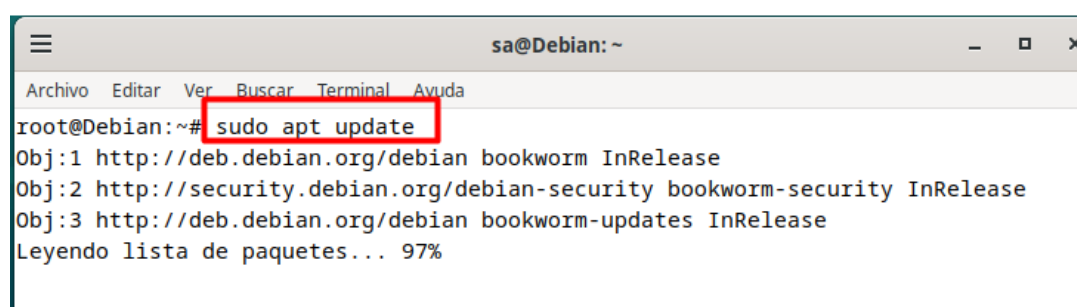
Anexo 1. Proceso de instalación de GNS3 en sistema operativo Debian

El software de simulación Graphical Network Simulator-3 (GNS3), tiene compatibilidad con varios sistemas operativos como Linux, sin embargo, requiere cumplir ciertos requerimientos en la máquina anfitriona, para el funcionamiento adecuado que permita la simulación estable de topologías, ya que, virtualizar los dispositivos consume varios recursos. Por tanto, los requerimientos recomendados que menciona (GNS3, 2016) son: 4 núcleos lógicos de serie ADM o Intel, soporte de virtualización, memoria RAM de 16 GB y almacenamiento de 35 GB en unidad de estado sólido (SDD).

Considerando los requerimientos mencionados, la instalación de GNS3 en un sistema operativo Debian 12 se realiza desde la terminal, donde se comienza con el comando `sudo apt update`, el cual, se encarga de la actualización de paquetes disponibles y más actuales en repositorios del sistema. A continuación, en la Figura 145 se evidencia el proceso al ejecutar el comando.

Figura 145

Actualización de paquetes en el sistema



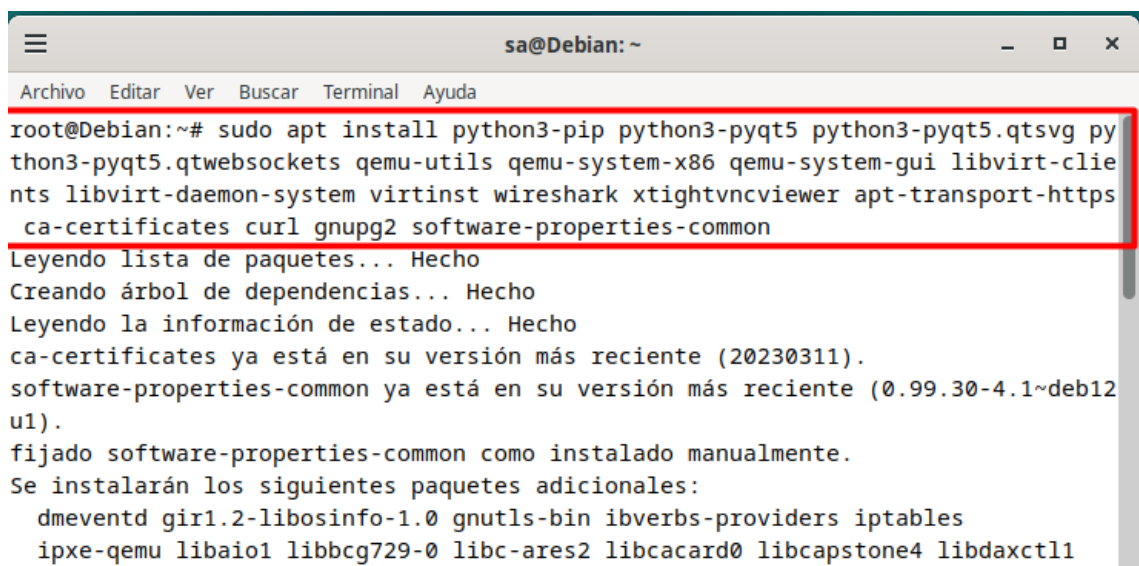
```
sa@Debian: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
root@Debian:~# sudo apt update  
Obj:1 http://deb.debian.org/debian bookworm InRelease  
Obj:2 http://security.debian.org/debian-security bookworm-security InRelease  
Obj:3 http://deb.debian.org/debian bookworm-updates InRelease  
Leyendo lista de paquetes... 97%
```

Posteriormente, se ilustra en la Figura 146, el comando a ejecutarse en el sistema Debian es: `sudo apt install python3-pip python3-pyqt5 python3-pyqt5.qtsvg python3-pyqt5.qtwebsockets qemu-utils qemu-system-x86 qemu-system-gui libvirt-clients libvirt-daemon-system virtinst wireshark xtightvncviewer apt-transport-https ca-certificates curl gnupg2 software-properties-common`, el cual, instala herramientas en conjunto. (Juma, 2025)

Entre las herramientas y paquetes necesarios para las simulaciones incluye bibliotecas de Python como PyQt5 que permite la creación de interfaces gráficas, herramientas de virtualización para la gestión de máquinas virtuales o recursos virtualizados mediante QEMU o libvirt, también instala herramientas de red para el análisis de tráfico como Wireshark e integra utilidades de administración para la gestión de repositorios, con la finalidad de obtener un entorno virtualizado completo en el sistema.

Figura 146

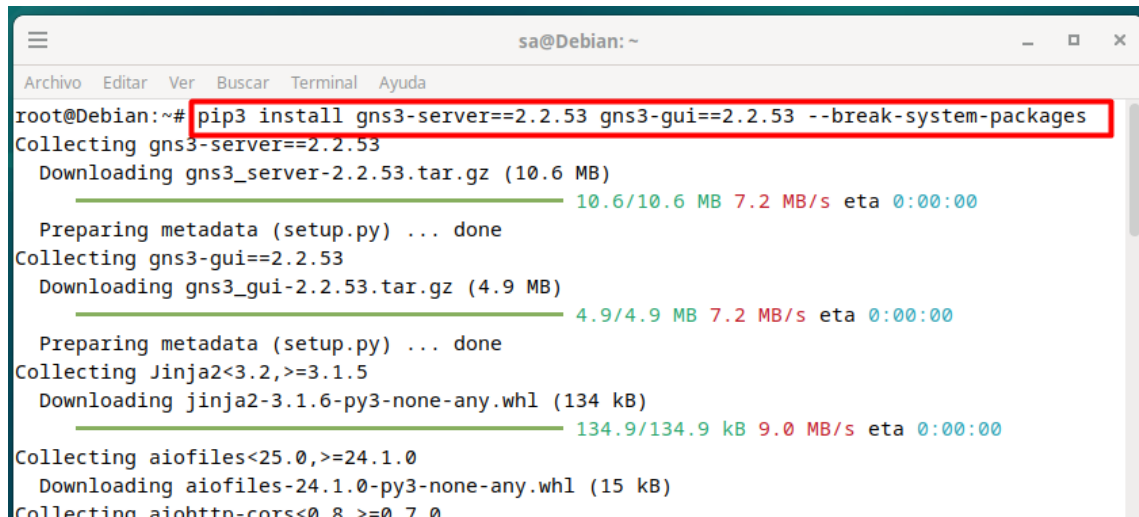
Instalación de herramientas requeridas por GNS3



```

sa@Debian: ~
Archivo Editar Ver Buscar Terminal Ayuda
root@Debian:~# sudo apt install python3-pip python3-pyqt5 python3-pyqt5.qtsvg python3-pyqt5.qtwebsockets qemu-utils qemu-system-x86 qemu-system-gui libvirt-clients libvirt-daemon-system virtinst wireshark xtightvncviewer apt-transport-https ca-certificates curl gnupg2 software-properties-common
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
ca-certificates ya está en su versión más reciente (20230311).
software-properties-common ya está en su versión más reciente (0.99.30-4.1~deb12u1).
fijado software-properties-common como instalado manualmente.
Se instalarán los siguientes paquetes adicionales:
  dmeventd gir1.2-libosinfo-1.0 gnutls-bin ibverbs-providers iptables
  ipxe-qemu libaio1 libbcg729-0 libc-ares2 libcacard0 libcapstone4 libdaxctl1
  
```

A continuación, el comando mostrado en la Figura 147, es: `pip3 install gns3-server==2.2.53 gns3-gui==2.2.53 --break-system-packages`, que es necesario para instalar GNS3, específicamente la versión 2.2.53, esto incluye la interfaz gráfica y servidor GNS3.

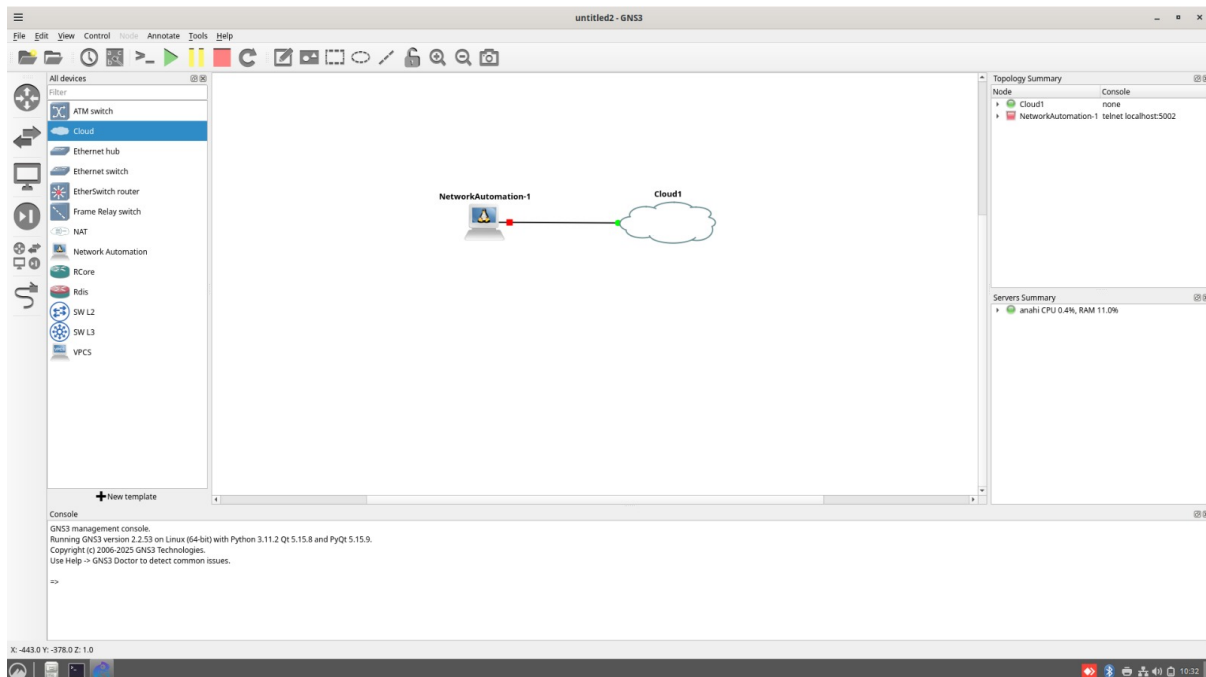
Figura 147*Instalación del entorno GNS3*


```

sa@Debian: ~
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
root@Debian:~# pip3 install gns3-server==2.2.53 gns3-gui==2.2.53 --break-system-packages
Collecting gns3-server==2.2.53
  Downloading gns3_server-2.2.53.tar.gz (10.6 MB)
    10.6/10.6 MB 7.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting gns3-gui==2.2.53
  Downloading gns3_gui-2.2.53.tar.gz (4.9 MB)
    4.9/4.9 MB 7.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting Jinja2<3.2,>=3.1.5
  Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)
    134.9/134.9 kB 9.0 MB/s eta 0:00:00
Collecting aiofiles<25.0,>=24.1.0
  Downloading aiofiles-24.1.0-py3-none-any.whl (15 kB)
Collecting aiohttp-cors<0.8.>=0.7.0

```

Al concluir con la instalación de herramientas, paquetes y dependencias necesarias para GNS3, se puede comprobar el proceso mediante el acceso al software, donde se inicia su interfaz gráfica como muestra la Figura 148, para el desarrollo y simulación de topologías.

Figura 148*Ingreso a la interfaz gráfica de GNS3*

Anexo 2. Instalación del servicio de automatización Ansible en máquina virtual

La primera alternativa para la instalación de Ansible es en una máquina virtual, la cual puede ser implementada en escenarios de simulación independientemente del software de simulación empleado.

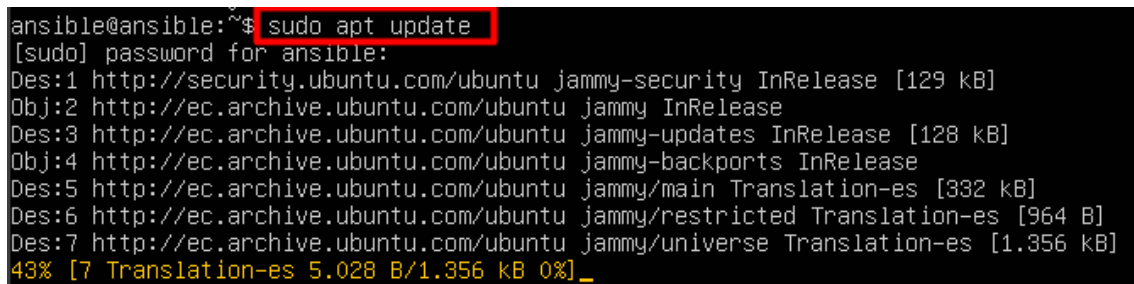
También, para implementarse en estructuras de redes reales, se recomienda instalar Ansible en una máquina física, máquina virtual o un servidor ubicado en el centro de datos, los cuales, deben ser accesibles para gestionar a todos los dispositivos de la red.

Para ejemplificar dicha alternativa, se considera la instalación de Ansible en una máquina virtual de sistema operativo Ubuntu Server versión 22.04, la cual, no posee una interfaz gráfica, reduciendo la cantidad de recursos. Sin embargo, los comandos deben ser adaptados según el sistema operativo a utilizar.

Su proceso inicia con la actualización de paquetes disponibles en el sistema con el comando `sudo apt update` indicado en la Figura 149.

Figura 149

Actualización de paquetes en sistema



```

ansible@ansible:~$ sudo apt update
[sudo] password for ansible:
Des:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Obj:2 http://ec.archive.ubuntu.com/ubuntu jammy InRelease
Des:3 http://ec.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Obj:4 http://ec.archive.ubuntu.com/ubuntu jammy-backports InRelease
Des:5 http://ec.archive.ubuntu.com/ubuntu jammy/main Translation-es [332 kB]
Des:6 http://ec.archive.ubuntu.com/ubuntu jammy/restricted Translation-es [964 B]
Des:7 http://ec.archive.ubuntu.com/ubuntu jammy/universe Translation-es [1.356 kB]
43% [7 Translation-es 5.028 B/1.356 kB 0%]_

```

Luego, se ejecuta el comando `sudo apt upgrade` mostrado en la Figura 150, para instalar las versiones actuales de los paquetes ya instalados en el sistema, con la finalidad de tener estabilidad y evitar errores por antiguas versiones que ya no pueden ser compatibles.

Figura 150*Instalación de paquetes de última versión*

```

ansible@ansible:~$ sudo apt upgrade
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Calculando la actualización... Hecho
Se instalarán los siguientes paquetes NUEVOS:
python3-packaging
Se actualizarán los siguientes paquetes:
cloud-init cryptsetup cryptsetup-bin cryptsetup-initramfs distro-info-data dmeventd dmidecode
dmsetup gir1.2-packagekit-glib-1.0 landscape-common libcryptsetup12 libdevmapper-event1.02.1
libdevmapper1.02.1 libldap-2.5-0 libldap-common liblvm2cmd2.03 libmbim-glib4 libmbim-proxy
libmm-glib0 libnss-sysd libnss3 libnss3-extra libpackagekit-glib2-18 libpam-modules

```

A continuación, la Figura 151, indica la ejecución del comando `sudo apt-add-repository ppa:ansible/ansible`, el cual permite añadir el repositorio de Ansible al sistema y poder acceder a las últimas versiones de la herramienta de automatización.

Figura 151*Agregar el repositorio de ansible al sistema*

```

ansible@ansible:~$ sudo apt-add-repository ppa:ansible/ansible
Repository: 'deb https://ppa.launchpadcontent.net/ansible/ansible/ubuntu/ jammy main'
Description:
Ansible is a radically simple IT automation platform that makes your applications and systems easier
to deploy. Avoid writing scripts or custom code to deploy and update your applications- automate in
a language that approaches plain English, using SSH, with no agents to install on remote systems.

http://ansible.com/

If you face any issues while installing Ansible PPA, file an issue here:
https://github.com/ansible-community/ppa/issues
More info: https://launchpad.net/~ansible/+archive/ubuntu/ansible
Adding repository.
Press [ENTER] to continue or Ctrl-c to cancel.

```

Después, se instala el paquete completo de Ansible, que incluye las dependencias necesarias como módulos de Python y conexión remota SSH para el funcionamiento de la herramienta, utilizando el comando `sudo apt install ansible`, indicado en la Figura 152.

Figura 152*Instalación de Ansible*

```

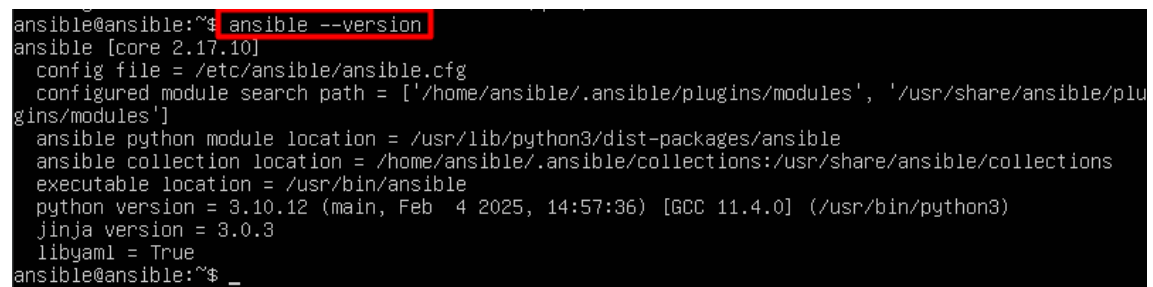
ansible@ansible:~$ sudo apt install ansible
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
ansible-core python3-jmespath python3-kerberos python3-nacl python3-ntlm-auth python3-paramiko
python3-requests-kerberos python3-requests-ntlm python3-resolvelib python3-winrm
python3-xmltodict sshpass
Paquetes sugeridos:
python-nacl-doc python3-gssapi python3-invoke
Se instalarán los siguientes paquetes NUEVOS:
ansible ansible-core python3-jmespath python3-kerberos python3-nacl python3-ntlm-auth
python3-paramiko python3-requests-kerberos python3-requests-ntlm python3-resolvelib
python3-winrm python3-xmltodict sshpass
0 actualizados, 13 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 19,2 MB de archivos.
Se utilizarán 213 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n]

```

Por último, se ingresa en la terminal el comando `ansible --version`, como indica en la Figura 153, para verificar la instalación correcta de la herramienta Ansible, este comando detalla la versión instalada, la ruta del archivo de configuración de ansible, versión de Python utilizada, ya que Ansible se basa en este lenguaje para su funcionamiento, además, detalla la instalación de librerías adicionales como `libyaml` y `jinja2` para el desarrollo de archivos de configuración.

Figura 153

Verificación de la instalación de Ansible



```
ansible@ansible:~$ ansible --version
ansible [core 2.17.10]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/ansible/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/ansible/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.10.12 (main, Feb  4 2025, 14:57:36) [GCC 11.4.0] (/usr/bin/python3)
  jinja version = 3.0.3
  libyaml = True
ansible@ansible:~$ _
```