ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA ELECTRÓNICA EN TELECOMUNICACIONES Y REDES

"IMPLEMENTACIÓN DEL PROVISIONAMIENTO AUTOMÁTICO DE CONFIGURACIONES (NETWORK AUTOMATION) EN INFRAESTRUCTURAS MULTIVENDOR CON ANSIBLE"

TRABAJO DE TITULACIÓN

TIPO: PROPUESTA TECNOLÓGICA

Presentado para optar al grado académico de:

INGENIERO EN ELECTRÓNICA, TELECOMUNICACIONES Y REDES

AUTOR: ALEX VICENTE YUNGA TOAQUIZA

TUTOR: Ing. ALBERTO LEOPOLDO ARELLANO AUCANCELA Msc.

Riobamba-Ecuador

2018

ESCUELA SUPERIOR POLITECNICA DE CHIMBORAZO

FACULTAD DE INFORMATICA Y ELECTRONICA

ESCUELA DE INGENIERIA ELECTRONICA EN TELECOMUNICACIONES Y REDES

El Tribunal de trabajo de titulación certifica que: El trabajo de titulación: "IMPLEMENTACIÓN DEL PROVISIONAMIENTO AUTOMÁTICO DE CONFIGURACIONES (NETWORK AUTOMATION) EN INFRAESTRUCTURAS MULTIVENDOR CON ANSIBLE", de responsabilidad del señor ALEX VICENTE YUNGA TOAQUIZA, ha sido minuciosamente revisado por los Miembros del Tribunal del trabajo de titulación quedando autorizado su presentación.

NOMBRE	FIRMA	FECHA
Dr. Julio Santillán **VICEDECANO FACULTAD DE INFORMÁTICA Y ELECTRÓNICA**	_____	_____
Ing. Patricio Romero **DIRECTOR DE ESCUELA DE INGENIERIA ELECTRÓNICA, TELECOMUNICACIONES Y REDES**	_____	_____
Ing. Alberto Arellano **DIRECTOR DE TESIS**	_____	_____
Ing. Raúl Lozada **MIEMBRO DEL TRIBUNAL**	_____	_____

Yo, Alex Vicente Yunga Toaquiza soy responsable de las ideas, doctrinas y resultados expuestos en este trabajo de titulación y el patrimonio intelectual del Trabajo de Titulación, y el patrimonio de la misma pertenece a la Escuela Superior Politécnica de Chimborazo.

_____

ALEX VICENTE YUNGA TOAQUIZA

DEDICATORIA

El presente trabajo de titulación va dedicado con mucho amor a mi madre María Transito Toaquiza que con sus consejos, regaños y enseñanzas me dio palabras de aliento para continuar y no desistir nunca en mis sueños, a mi padre Ángel Vicente Yunga que con su esfuerzo, dedicación y sacrificio me dio todo sin pedir nada a cambio, este logro no hubiera sido posible sin los dos pilares fundamentales que son mis padres ya que cada uno de ellos aporto con su granito de arena. A mis hermanos Paul y Patricio que con sus alegrías y tristezas hicieron de mi vida universitaria más llevadera, a mi hermana Heidy que me enseño que la distancia es solo un número y que el amor y respeto entre hermanos siempre permanecerá constante. A mi sobrino Lucas que llego en el momento exacto para llenar de alegría mi vida. A mis abuelos, tíos, primos y amigos que me han brindado su apoyo incondicional y han estado aquí en las buenas y malas.

Alex

AGRADECIMIENTO

Agradezco a DIOS por darme la oportunidad de terminar esta etapa en mi vida y por brindarme fuerza y sabiduría para superar todos los obstáculos que se me han presentado. A mi familia que por más complicadas que se veían las cosas nunca me dejaron solo. A mis amigos que me brindaron su amistad y me ayudaron en todo lo posible. A todos los docentes que formaron parte de esta larga travesía, quienes compartieron sus experiencias y conocimientos al momento de dictar sus clases. De manera especial a mi tutor Ing. Alberto Arellano por brindarme su apoyo y confianza para la realización de este trabajo de titulación y al Ing. Raúl Lozada por ayudarme con las correcciones necesarias. Gracias a todos, por tanto, no puedo responder otra cosa que gracias y mil gracias.

Alex

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE GRÁFICOS

ÍNDICE DE ABREVIATURAS

Abreviatura	Descripción
YAML	YAML no es un lenguaje de marcado (YAML Ain't Markup Language)
SDN	Redes definidas por software (Software Defined Networking)
NFV	Virtualización de funciones de red (Network Function Virtualization)
TI/IT	Tecnología de la información (Information Technology)
LAN	Red de área local (Local Area Network)
WAN	Red de área amplia (Wide Area Network)
CLI	Interfaz de línea de comandos (Command Line Interface)
GUI	Interfaz gráfica de usuario (Graphical User Interface)
API	Interfaz de programación de aplicaciones (Application Programming Interface)
XML	Lenguaje de Marcado Extensible (eXtensible Markup Language)
IoT	El internet de las cosas (Internet of Things)
SSH	Secure SHell
DEVOPS	Desarrollo y Operaciones (*Development* y *Operations*)
IP	Protocolo de internet (Internet Protocol)
CPU	Unidad central de procesamiento (Central Processing Unit)
GNS3	Simulación gráfica de redes (Graphic Network Simulation)
EVE-NG	Entorno Virtual Emulado - Próxima Generación (Emulated Virtual Environment - Next Generation)
RAM	Memoria de acceso aleatorio (Random Access Memory)
VLAN	Red de área local virtual (Virtual Local Area Network)
BGP	Protocolo de puerta de enlace de frontera (Border Gateway Protocol)

MPLS Conmutación de etiquetas multiprotocolo (Multiprotocol Label Switching)

PE Borde del proveedor (Provider Edge)

P Proveedor (Provider)

CE Borde del cliente (Customer Edge)

VPN Red privada virtual (Virtual Private Network)

SP Proveedor de Servicios (Service Provider)

NAT Traducción de direcciones de red (Network Address Translation)

OSPF Primer Camino Más Corto (Open Shortest Path First)

DHCP Protocolo de configuración dinámica de host (Dynamic Host Configuration Protocol

ASCII Código Estándar Estadounidense para el Intercambio de Información (American Standard Code for Information Interchange)

ÍNDICE DE ANEXOS

RESUMEN

El trabajo de titulación tuvo como objetivo implementar el provisionamiento automático de configuraciones (Network Automation) en infraestructuras multivendor con Ansible. Se utilizó GNS3 como herramienta de emulación para realizar la implementación de las distintas topologías, al utilizar GNS3 por defecto también se utiliza su máquina virtual a través de VM Ware; su uso es para instalar la herramienta de automatización y las plataformas de networking que se utilizan, como son CISCO, ARISTA Y JUNIPER, la selección de estas plataformas se hizo en base a una comparación de distintas características (Benchmarking). Al realizar la simulación, se emplean distintas topologías con configuraciones de BGP, VLANS, MPLS en los dispositivos de red como routers, switches y firewall. El nodo central, por medio de Ansible se encarga de aplicar todas las configuraciones a los dispositivos de red mediante los distintos directorios, ficheros, playbooks (que están escritos en YAML) y plantillas (que están escrito en JINJA2), cada uno de estos cumple un rol específico para aplicar las tareas de configuración al nodo secundario. Se concluye que Ansible es una herramienta de automatización "sin agentes" o "agentless" que no utiliza bases de datos, no utiliza demonios y ningún agente externo, por ende, no deja ninguna vulnerabilidad en los equipos, a diferencia de otras herramientas que utilizan agentes externos como Chef y Puppet. Al usar Ansible en las distintas topologías de red a más de miles de dispositivos de red se recomienda llevar una documentación bien estructurada y ordenada de todos los playbooks y plantillas para la red. Una de las herramientas que se utiliza para llevar dicha documentación es Sublime Text 3 que es un editor de texto y editor de código fuente que está escrito en C++ y Python que brinda la facilidad de soportar distintos lenguajes de programación.

PALABRAS CLAVE: <REDES>, <AUTOMATIZACION DE REDES>, <GNS3>, <ANSIBLE>, <PLAYBOOKS Y PLANTILLAS>, <PROVISIONAMIENTO AUTOMATICO >

SUMMARY

The aim of the titling work was to implement the automatic provisioning of configurations (Network Automation) in multivendor infrastructure with Ansible. GNS3 was used as an emulation tool to implement the different topologies. By using GNS3, by default, your virtual machine is also used through VM Ware; its use is to install the automation tool and the networking platforms that are used, such as CISCO, ARISTA and JUNIPER, the selection of these platforms was based on a comparison of different features (Benchmarking). When performing the simulation, different topologies are used with configurations of BGP, VLANS, MPLS in the network devices such as routers, switches and firewall. The central node, through Ansible is responsible for applying all configurations to network devices through the different directories, files, playbooks (which are written in YAML) and templates (which are written in JINJA2), each of these meets a specific role to apply the configuration tasks to the secondary node. It is concluded that Ansible is an "agentless" or "agentless" automation tool that does not use databases, does not use daemons and no external agent, therefore, it does not leave any vulnerability in the equipment, unlike other tools that use external agents such as Chef and Puppet. When using Ansible in the different network topologies to more than thousands of network devices it is recommended to have a well-structured and ordered documentation of all the playbooks and templates for the network. One of the tools used to carry such documentation is Sublime Text 3, which is a text editor and source code editor that is written in C ++ and Python that provides the ability to support different programming languages.

KEYWORDS: <NETWORKS>, <NETWORK AUTOMATION>, <GNS3>, <ANSIBLE>, <PLAYBOOKS AND TEMPLATES>, <AUTOMATIC PROVISIONING>

INTRODUCCIÓN

El Internet se define como una red de redes que permite conectar múltiples redes que se encuentran distribuidas por todo el mundo y que tienen distintas características, plataformas y ambientes. Esta conexión es posible porque todas utilizan el mismo protocolo de comunicación, TCP/IP. Todo este proceso, está regido por una serie de normas y estándares incluidas en dichos protocolos. El crecimiento de Internet ha ido a la par con las diferentes empresas que ofrecen una amplia gama de productos y soluciones para redes, diseñados para las pequeñas y las grandes empresas de diversos sectores, tanto para las redes públicas y para las redes privadas (Stallings, 2004).

Para que exista una conexión es necesario dos componentes: los componentes físicos que son el hardware y medios físicos necesarios para la comunicación entre computadoras y los componentes lógicos, que vienen a ser los protocolos de comunicación y el software que permite dicha comunicación. El tamaño de la red y las prestaciones que se ofrece influyen directamente en estos componentes ya que pueden aumentar en número y complejidad (Edelman, 2015).

Para realizar una configuración en cualquier dispositivo se emplea por lo general la interfaz de línea de comandos (CLI) que es un método que permite a los administradores de red realizar configuraciones por medio de una línea de texto y la interfaz gráfica de usuario (GUI) que ofrece una estética mejorada a costa de un mayor consumo de recursos computacionales. Todas estas configuraciones son hechas por los administradores de red que se enfrenta a cierto problema de actualización en los dispositivos de red, ya que existe una gran cantidad de equipos de diferentes marcas y que manejan distintos protocolos de enrutamiento. Este problema podría ocasionar pérdidas significativas para cualquier empresa tanto en la eficiencia operativa, la gestión y la prestación de sus servicios (Rivenes, 2016).

Las empresas buscan constantemente formas de mejorar su agilidad para mantener su competitividad. La TI y más concretamente, los dispositivos y servicios basados en IP juegan un papel fundamental en esta dinámica continua. En un mundo “todo IP”, la necesidad de suministrar infraestructuras de redes de gran calidad nunca ha sido mayor. Una empresa es tan ágil como los cimientos de sus redes (Efficient IP, 2016).

La solución es Network Automation que permite realizar una configuración automatizada de la red en todas sus plataformas físicas y virtuales, reduciendo el tiempo y los gastos en las tareas de configuración. En el mercado existen varias herramientas que ofrecen esta tecnología como: Chef, Puppet, Salt Stack y Ansible

FORMULACIÓN DEL PROBLEMA

¿Cuáles son los beneficios de Ansible al momento de utilizarlo para la configuración de los dispositivos de red en infraestructuras multivendor?

SISTEMATIZACIÓN DEL PROBLEMA

¿Cuáles son las características, ventajas y desventajas de las plataformas de Network Automation existentes?

¿Para qué sirve construir scripts que configuren las distintas tareas administrativas de la red?

¿Cuáles son los riesgos que están asociados a la seguridad de Network Automation?

¿Cuál es el grado de interoperabilidad enfocándose en las configuraciones de red mediante Ansible?

JUSTIFICACIÓN DEL TRABAJO DE TITULACIÓN

JUSTIFICACIÓN TEÓRICA

El uso de Ansible para realizar las acciones de configuración y solución de problemas reduce drásticamente la carga de trabajo del administrador de redes, estas acciones se realizan con la mínima intervención humana, por lo que hoy en día muchas de las empresas están implementando esta tecnología ya que permite mejorar los tiempos de configuración así optimizando la disponibilidad de la red.

Ansible brinda las siguientes características (Quilcate, 2016):

- Es libre. Los playbooks están basados en YAML y por lo tanto son muy fáciles de leer, entender y mantener.

- No requiere ningún agente. Utiliza una arquitectura “agentless” esto significa que no necesita un “ansible-client” en los nodos para ejecutar las diferentes tareas, solo necesitaría un “master” que diga que tareas ejecutar.
- Trabaja en SSH y mantiene un enfoque “push”, donde el master envía la configuración a los nodos.

Los criterios para elegir Ansible se encuentra en la investigación denominada “EXTENDING ANSIBLE: Discover how to efficiently deploy and customize Ansible in the way your platform demands” por autoría de Rishabh Das, donde se indica que Ansible al ser sin agente reduce la sobrecarga de la configuración de los agentes necesarios en los dispositivos de destino, reduciendo también los riesgos de seguridad, ya que no es necesario instalar ningún paquete o agente adicional, permitiéndole tomar ventaja sobre las demás soluciones (Das, 2016).

Y también en la investigación nombrada “ANSIBLE FOR DEVOPS: Server and configuration management for humans” desarrollada por el autor Jeff Geerling donde concluye que Ansible es una plataforma de automatización de TI de propósito general que utiliza una metáfora para describir sus archivos de configuración denominados ‘playbooks’, que listan un conjunto de tareas que se ejecutarán en un determinado dispositivos de red; que a través de ellos convierten a Ansible en una herramienta de gestión de configuración y aprovisionamiento de servidores y de dispositivos de red (Geerling, 2015).

Puesto que los equipos de operaciones de red, hoy en día requieren habilidades importantes para tomar ventaja de los avances en el análisis y la automatización de la red; se estima que para el 2020 haya un incremento del 16% en Network Automation, mediante el aprovechamiento de lenguajes de scripting como YAML y JINJA2 que permite cumplir las demandas y las necesidades de las empresas; reduciendo el tiempo de inactividad de la red cuando se produzca una actualización o un cambio en la misma (Ganguli, Bhalla y Lerner, 2017).

JUSTIFICACIÓN APLICADA

En todas las empresas de networking un administrador de red tiene como tarea desplegar y configurar los servicios y herramientas necesarias para el buen funcionamiento de la red pensando en temas

como la disponibilidad, escalabilidad, seguridad y redundancia. Permitiéndole a las empresas brindar sus servicios con un porcentaje mínimo de interrupción, optimizando los tiempos de configuración en cada uno de los dispositivos de red, mejorando la seguridad y permitiendo realizar actualizaciones en toda la topología de la red.

El proyecto pretende realizar una configuración automática por medio de scripts y cargarlos a un nodo central mediante una herramienta de emulación que permitan ejecutar las distintas tareas de automatización para los dispositivos de red a través de playbooks y plantillas que normalmente son comandos que ejecutan un conjunto de acciones en un orden preestablecido.

OBJETIVOS

OBJETIVO GENERAL

- Implementar el provisionamiento automático de configuraciones (network automation) en infraestructuras multivendor con Ansible.

OBJETIVOS ESPECÍFICOS

- Estudiar los diferentes parámetros de configuraciones establecidos en los módulos de red para Ansible y equipos a utilizar.
- Verificar el adecuado funcionamiento de Network Automation a través de una herramienta de emulación.
- Determinar el nivel de compatibilidad de Ansible con el equipamiento de networking existente en el mercado con respecto a las configuraciones de red.
- Elaborar una propuesta de implementación de Network Automation en infraestructuras multivendor con Ansible.

CAPÍTULO I

1. MARCO TEÓRICO REFERENCIAL

En este capítulo se describe las características y beneficios de Network Automation, las herramientas de automatización y además se detalla el funcionamiento de Ansible mediante el desarrollo de scripts a través de YAML y JINJA2.

1.1 Redes programables

El mundo de las redes está experimentando un cambio importante en los últimos años. Los microservicios y contenedores en la pila de aplicaciones han creado nuevos desafíos para los ingenieros de redes en todas las empresas para construir una pila de red más ágil y rápida. Una mayor visibilidad en el flujo de red de extremo a extremo y el control granular es otro requisito importante. Significa que las organizaciones deben mantenerse al día con la agilidad de un entorno de aplicaciones en constante cambio. En un entorno de ritmo rápido, la agilidad de la red se puede lograr a través de Redes definidas por software (SDN), Virtualización de funciones de red (NFV) y Redes programables. Si bien estos enfoques de agilidad de la red a veces se complementan entre sí, SDN requiere un cambio fundamental en la forma en que las redes se construyen y operan. Los proveedores de servicios han absorbido este concepto de una manera limitada, mientras que las empresas todavía les resulta difícil adoptar ya que sus necesidades son diferentes. Las empresas no venden redes como proveedores de servicios. La red programable es la respuesta a las necesidades de las empresas. La red de extremo a extremo se puede automatizar a través de la programabilidad sin cambiar la infraestructura. Es por eso que las empresas buscan la automatización de redes para administrar la infraestructura a través de un motor de automatización central (Venkatesh, 2018).

La automatización es el proceso que permite mejorar varias operaciones que controlan, regulan y administran máquinas, sistemas dispares o software con poca o ninguna intervención humana. Se espera que un sistema automatizado realice una función de manera más confiable, eficiente y precisa que un operador humano. La máquina automatizada realiza una función a un costo menor con una

mayor eficiencia que un operador humano, por lo tanto, la automatización se está volviendo cada vez más extendida en varias industrias de servicios, así como en la industria de TI y software. (Sharma y Soni, 2015).

La automatización básicamente ayuda a una empresa a:

- Reducir las complejidades de los procesos y pasos secuenciales.
- Reducir las posibilidades de error humano en tareas repetibles.
- Mejorar de forma consistente y predecible el rendimiento de un sistema.
- Aumentar productividad y el alcance de la innovación en un negocio.
- Mejorar la solidez, la agilidad del despliegue de aplicaciones en diferentes entornos y reduce el tiempo de comercialización de una aplicación.

La automatización es una palabra muy utilizada desde la prehistoria hasta las últimas décadas debido a que engloba un amplio abanico de sistemas y procesos en los cuales se requiere la mínima intervención del ser humano, además debe de ser un sistema "flexible" el cual se debe ajustar de distintas maneras a los posibles cambios en momentos puntuales (Martinez, 2017).

La ventaja de la automatización es que se adapta a cualquier tipo de industria ayudando a solucionar los problemas que en esta se presente, como es el caso de las redes que necesita de una automatización programable para una gran variedad de dispositivos de red a través de un medio de programación (software).

En términos generales, la automatización de la red (Network Automation), como la mayoría de los tipos de automatización, equivale a hacer las cosas más rápido. Si bien es más fácil hacerlo más rápido, reducir el tiempo para las implementaciones y los cambios en la configuración no siempre es un problema que deba resolverse para muchas organizaciones de TI (Edelman, 2016).

1.2 Automatización de red (network automation)

Es un proceso continuo de generación e implementación de cambios de configuración, administración y operaciones de dispositivos de red. A menudo implica cambios de configuración más rápidos en una cantidad significativa de dispositivos, pero no está limitado a grandes infraestructuras. Es igualmente importante cuando se administran implementaciones más pequeñas para garantizar la coherencia con otros dispositivos y reducir el factor de error humano. La automatización es más que solo administración de configuración; es un área amplia que también incluye la recopilación de datos de los dispositivos, resolución de problemas automática y capacidad de recuperación: la red puede ser lo suficientemente inteligente como para solucionar los problemas por sí misma, dependiendo de factores internos o externos (Ulinic y House, 2017).

1.2.1 Tipos de automatización

La automatización puede emplearse en cualquier tipo de red, incluidas las redes de área local (LAN), redes de área extensa (WAN), redes de centros de datos, redes en la nube y redes inalámbricas. En resumen, cualquier recurso de red controlado a través de la interfaz de línea de comando (CLI) o una interfaz de programación de aplicaciones (API) puede automatizarse (Rouse y Scarpati, 2017).

1.2.1.1 La automatización de red basada en scripts

La figura 1-1 indica un ejemplo de automatización basada en scripting y en lenguajes de programación para ejecutar tareas, idealmente aquellas con disparadores precisos y procedimientos consistentes. Los lenguajes heredados, como Perl y Tcl, prevalecen en la automatización de redes debido a su familiaridad. Pero, a medida que las redes se vuelven más complejas, los nuevos lenguajes de programación de código abierto, como Python y Ruby, han ganado popularidad por su facilidad de uso y flexibilidad.

Figura 1-1: Automatización de red basada en scripts
Fuente: YUNGA, Alex, 2018

1.2.1.2 La automatización de red basada en software

Denominada automatización inteligente de red se coordina a través de un portal administrativo que elimina la necesidad de ejecutar comandos manualmente. Estas plataformas suelen proporcionar plantillas para crear y ejecutar tareas basadas en políticas de lenguaje sencillo.

1.2.2 Beneficios de Network Automation

La automatización brinda los siguientes beneficios a la industria de TI: (Sharma y Soni, 2015)

- ***Agilidad:*** brinda prontitud y agilidad a su infraestructura de TI. La productividad y flexibilidad es la ventaja significativa de la automatización, que ayuda a competir con la situación económica ágil actual.
- ***Escalabilidad:*** mediante la automatización, se puede administrar las configuraciones de la infraestructura y aprovechar la escalabilidad de los recursos para satisfacer la demanda de los clientes. Ayudando a transformar la infraestructura en un código simple, lo que significa que construir, reconstruir, configurar y escalar la infraestructura es posible en solo unos minutos según las necesidades de los clientes en un entorno real.
- ***Eficiencia y consistencia:*** puede manejar todas las tareas repetidas con mucha facilidad, para que pueda concentrarse en negocios innovadores. Aumentando la agilidad y la eficiencia de la administración de un entorno de despliegue y la implementación.
- ***Gestión eficaz de los recursos:*** ayuda a mantener un modelo de infraestructura que debe ser coherente. Proporcionando un marco de diseño basado en código que lleva a una forma flexible y manejable de conocer todos los fundamentos de una red compleja.

- ***Precisión de implementación:*** el desarrollo y la entrega de aplicaciones es un esfuerzo multifacético, engorroso, repetitivo y limitado en el tiempo. Se necesita hacer uso de la automatización, la capacidad de prueba de un entorno de implementación y la disciplina de aplicación de una secuencia de comandos precisa de los cambios en un entorno, y la repetitividad de esos cambios se puede hacer muy rápidamente.

1.2.3 Herramientas de Network Automation

Hay varias categorías de interfaces, plataformas y protocolos utilizados para ejecutar la automatización de red basada en script o basada en software. El CLI es la forma más tradicional para implementar la automatización de red. En el mercado existen varias herramientas que ofrecen entornos de automatización de redes mediante los usos de una biblioteca de comandos comunes o flujos de trabajo que pueden repetirse fácilmente, a continuación, se describen las principales que se muestran en la figura 2-1.

Figura 2-1: Principales herramientas de Network Automation
Fuente: http://blog.kwnetapps.com/configuration-management/

1.2.3.1 Chef

Chef es una herramienta de administración de configuración de código abierto desarrollada por la comunidad Opscode en 2008. Lanzaron su primera edición en enero de 2009. Opscode es administrado por individuos de los equipos de centro de datos de Amazon y Microsoft. Chef es compatible con una variedad de sistemas operativos; normalmente se ejecuta en Linux, pero también es compatible con Windows 7 y Windows Server. Chef está escrito en Ruby y Erlang, ambos son lenguajes de programación en tiempo real (Sharma y Soni, 2015, p. 8).

La figura 3-1 indica los componentes de Chef tales como Workstation, Chef Server y los nodos son los tres componentes principales de esta plataforma. El Chef Server almacena los datos para configurar y administrar a los nodos de manera efectiva. Un workstation funciona como un repositorio local de Chef donde está instalado Knife, que permite cargar los cookbooks en un Chef Server. Los cookbooks son una colección de recipes (recetas). Las recipes ejecutan acciones que deben automatizarse. Un nodo se comunica con un Chef Server y obtiene los datos de configuración relacionados con el servidor y lo ejecuta para instalar paquetes o realizar cualquier otra operación para la administración de la configuración (Sharma y Soni, 2015).

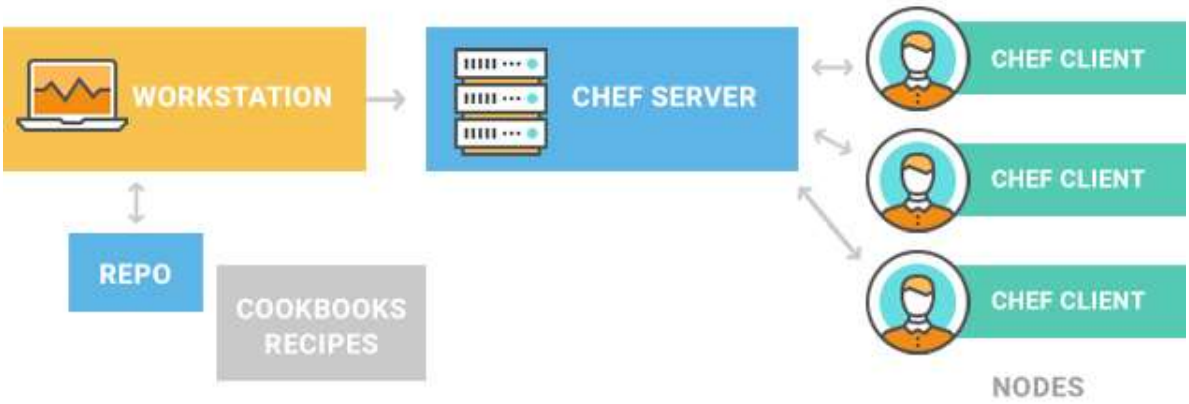


Figura 3-1: Arquitectura de Chef
Fuente: https://logz.io/blog/chef-vs-puppet/

Las características más destacadas de Chef son las siguientes (Sharma y Soni, 2015, p. 9-10) :

- Chef tiene diferentes flavors (sabores) para las soluciones automatizadas para las operaciones actuales de TI, como Open Source Chef, Hosted Chef y Private Chef.
- Chef habilita las funciones de capacidades de automatización altamente escalables, seguras y tolerantes a fallas de su infraestructura.
- Cada flavor tiene una solución específica para manejar diferentes tipos de necesidades de infraestructura. Por ejemplo, el servidor Open Source Chef está disponible gratuitamente para todos, pero admite funciones limitadas, mientras que el servidor Hosted Chef es administrado por Opscode como un servicio con tarifas de suscripción para soporte estándar y premium. El servidor de Private Chef proporciona una solución automatizada en el lugar con un precio de suscripción y planes de licencia.

- Chef tiene una comunidad muy fuerte. El sitio web, https://www.chef.io/, ayuda a comenzar con Chef y publicar cosas. Opscode ha organizado numerosos seminarios web, publica material de capacitación y facilita a los desarrolladores la contribución a nuevos parches y lanzamientos.
- Chef puede manejar rápidamente todo tipo de dependencias tradicionales y procesos manuales de toda la red.
- Chef tiene un sólido enfoque de administración de dependencias, lo que significa que solo la secuencia de orden importa, y todas las dependencias se cumplirán si se especifican en el orden correcto.

1.2.3.2 Puppet

Puppet Labs es uno de los líderes en la automatización de IT desde sus inicios en 2005, facilitando la automatización y gestión del host y su software tanto en máquinas físicas como virtuales en servidores dedicados o integrándose con servicios en la nube. Puppet aparece como proyecto de código abierto y a partir de este se distribuye Puppet Enterprise (PE). PE es una extensión de Puppet que incluye soporte profesional, un stack de Puppet listo para producción, una consola web para analizar reportes y controlar las infraestructuras, funcionalidades de orquestación y herramientas para provisionar en la nube. El ecosistema de Puppet incluye decenas de proyectos de código abierto que se emplean tanto en la versión de la comunidad como en la empresarial. Dentro de estos se encuentra Geppeto que es un IDE para facilitar el desarrollo en Puppet. Adicionalmente provee una interfaz: Forge para crear proyectos desde módulos preexistentes y facilitar su socialización en la comunidad (Carbonell y García, 2016).

Puppet está diseñado para funcionar en una arquitectura cliente/servidor o de forma independiente como se muestra en la figura 4-1. En la primera todos los ficheros de configuración con las "recetas" escritas en el lenguaje declarativo son almacenados en el nodo servidor denominado "Puppet Master". Este nodo central se encarga de muchas de las tareas como analizar las recetas y compilarlas para generar catálogos. Estos catálogos son conjuntos de ficheros XML que serán recogidos por los clientes o agentes. Los clientes solo se encargan de implementar la funcionalidad requerida al comparar los catálogos almacenados en su cache local con los del servidor y finalmente reporta al servidor los cambios realizados (Carbonell y García, 2016).

El Master también puede enviar notificaciones a los clientes en caso de que los archivos de configuración hayan cambiado para hacer que el cliente obtenga los nuevos catálogos. En la arquitectura independiente los nodos administrados contienen la copia completa de su información de configuración y compilan su propio catalogo para luego en demanda o por un proceso planificado aplicarlos (Carbonell y García, 2016).

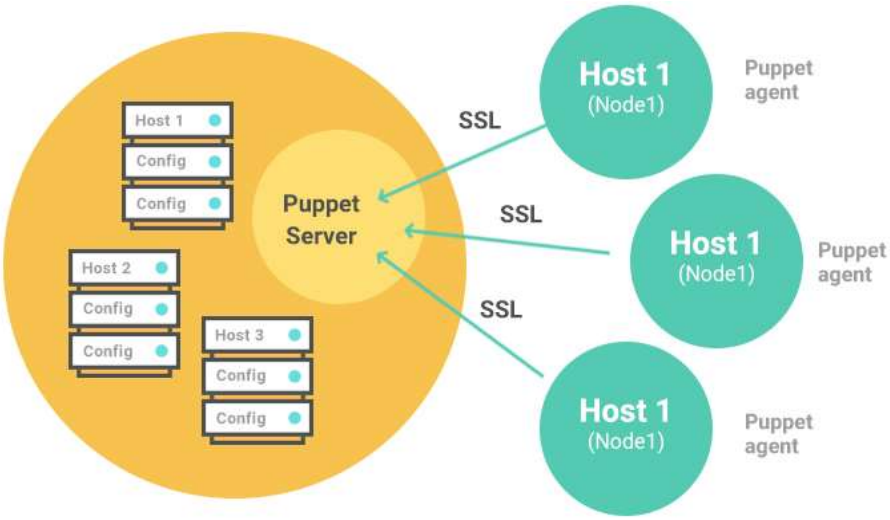


Figura 4-1: Configuración PUPPET cliente-servidor
Fuente: https://logz.io/blog/chef-vs-puppet/

Los datos que el agente de Puppet informa al maestro de Puppet se componen de dos elementos cruciales: registros y métricas. El agente de Puppet crea un registro de auditoría completo de los eventos durante cada ejecución, y cuando los informes están habilitados, esto se enviará al maestro de Puppet. Esto le permite ver si hubo algún problema durante la ejecución, y si es así, cuáles fueron; o, simplemente, le permite examinar qué operaciones realizó el agente de Puppet si todo transcurrió sin problemas. Las métricas que el agente de Puppet transmite al Puppet master son muy detalladas y ofrecen una visión fantástica de dónde y que está pasando en Puppet, ya sea buscando, procesando o aplicando cambios (Duffy, 2014, p. 8).

Las características de Puppet son las siguientes (Sharma y Soni, 2015, p. 15):

- Puppet proporciona automatización y orquestación continúa permitiendo resolver muchos desafíos de integración en tiempo real con diferentes tipos de implementación de servidores.
- Puppet se adapta rápidamente a los cambios y permite ampliar los servidores según demanda, con framework Puppet también se realiza la implementación en la nube.

- Puppet utiliza un enfoque basado en modelos declarativos para la automatización de TI. Tiene cuatro etapas principales: definir, simular, aplicar e informar.
- La comunidad de Puppet admite módulos de configuración reutilizables. Tiene más de 1,000 módulos de configuración precompilados y libremente descargables.
- Si se tiene un requisito específico se puede usar el lenguaje de configuración de Puppet, para construir un módulo propio y personalizado. Después de definir el módulo personalizado, se puede reutilizarlo para cualquier tipo de requisito, como físico, virtual o en la nube.

1.2.3.3 Salt Stack

Salt Stack es comúnmente empleado para desplegar, gestionar y automatizar infraestructuras y aplicaciones para big data, IoT, storages dinámicos, redes definidas por software, servidores de seguridad, etc. La arquitectura de Salt Stack que se muestra en la figura 4-1, se basa en la idea de ejecutar comandos de forma remota. Todas las redes están diseñadas en torno a algún aspecto de la ejecución remota. Esto podría ser tan simple como pedirle a un servidor web remoto que muestre una página web estática, o tan complejo como usar una sesión de shell para emitir comandos interactivamente contra un servidor remoto (Carbonell y García, 2016, p. 7).

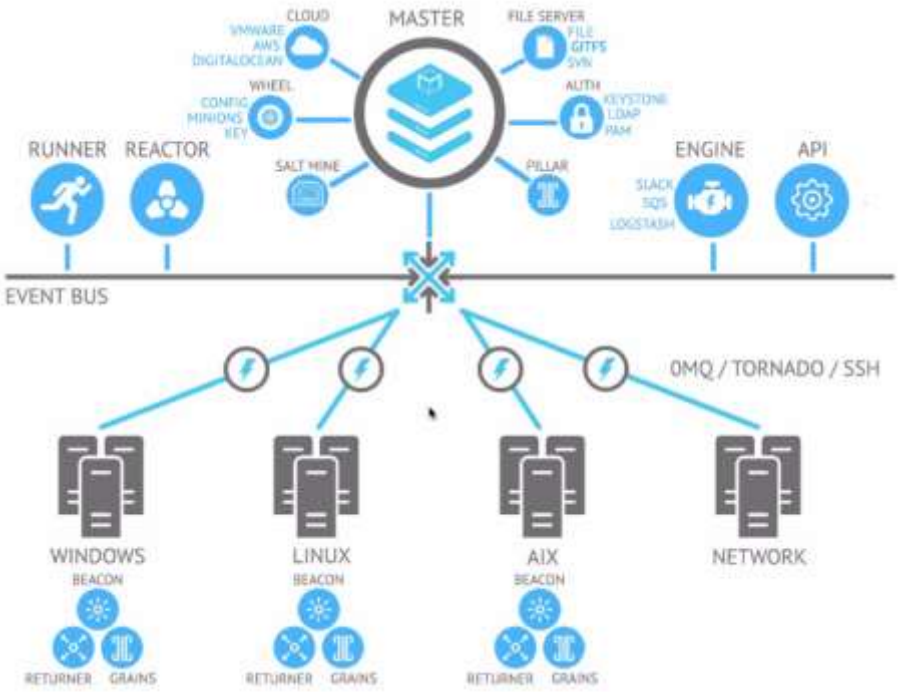


Figura 5-1: Arquitectura de SaltStack
Fuente: https://www.mirantis.com/blog/introduction-to-salt-and-saltstack/

Salt Stack está diseñado para permitir a los usuarios dirigirse explícitamente y emitir comandos a múltiples máquinas directamente. Salt se basa en la idea de un Master, que controla uno o más Minions. Los comandos normalmente se emiten desde el maestro a un grupo objetivo de subordinados, que luego ejecutan las tareas especificadas en los comandos y luego devuelven los datos resultantes al maestro. Las comunicaciones entre un maestro y Minions ocurren a través del bus de mensajes ZeroMQ. El Maestro de Salt y los Minions usan claves para comunicarse. Cuando un subordinado se conecta a un maestro por primera vez, automáticamente almacena las llaves en el maestro. SaltStack también ofrece Salt SSH, que proporciona una administración de sistemas "agentless" (Tutorials Points, 2017, p. 85, p. 3).

Las características de Salt son las siguientes (Sharma y Soni, 2015, p. 21-22):

- Es una de las comunidades de código abierto más activas y de más rápido crecimiento en el mundo.
- Ofrece un enfoque completamente diferente a las alternativas heredadas no creadas para la velocidad y la escala de una nube.
- El logro encomiable de SaltStack es que, con el fin de orquestar y controlar cualquier nube y proporcionar automatización para la cadena de herramientas DevOps, es utilizado por las empresas de TI y organizaciones DevOps más grandes del mundo.
- Proporciona soporte heterogéneo para la configuración e infraestructura de la nube o cualquier plataforma de software.
- SaltStack es principalmente conocido por la administración paralela. Además, proporciona automatización de datos en tiempo real. Es mucho menos lento, por lo que, si se quiere verificar las especificaciones de tiempo, en este caso la ejecución remota se realiza en segundos y no en minutos u horas.

1.2.3.4 Ansible

Ansible es una plataforma de automatización de TI que facilita la implementación de aplicaciones y sistemas. La plataforma es muy fácil de usar para cualquiera, incluidos los principiantes debido al uso de SSH para establecer una conexión con los dispositivos de red y luego ejecutar las tareas de configuración, dando como resultado una plataforma fácil de usar por medio de los scripts que son una forma muy común de realizar tareas de configuración (Hull, 2016, p. 8).

En Ansible, las tareas son idempotentes es decir que una tarea puede ser repetida tantas veces como sea necesario. Sin codificación adicional, se basa en hechos, que incluyen el sistema y la información del entorno que recopila antes de que pueda ejecutar dichas tareas. Ansible hace uso de tales hechos para verificar el estado y determinar si algo necesita ser cambiado para que pueda obtener el resultado que espera. Esto es lo que hace que sea seguro ejecutar Ansible contra un servidor en particular una y otra vez (Hull, 2016).

Las características de Ansible son las siguientes (Sharma y Soni, 2015, p. 21):

- Ansible es una herramienta de automatización de código abierto que se sirve para administrar tareas de configuración e implementación y también permite ejecutar tareas ad-hoc.
- Funciona con SSH, que es muy popular entre los usuarios y administradores de Linux.
- Es rápido y simple de instalar ya que no se necesita ningún archivo de configuración, daemon o base de datos.
- Ansible está diseñado de tal manera que no necesita nada más que una contraseña o clave SSH para comenzar a administrar sistemas y lo hace sin instalar ningún agente de software.
- Ansible configura sus tareas a través de playbooks que usa un lenguaje descriptivo simple, basado en YAML.

1.3 Conceptos y arquitectura de Ansible

Hay dos tipos de máquinas en la arquitectura Ansible: el nodo de control y los hosts administrados. El software Ansible está instalado en el nodo de control y todos sus componentes se mantienen en él. Los hosts gestionados se enumeran en un inventario de host, un archivo de texto en el nodo de control que incluye una lista de nombres de host administrados o direcciones IP, todos estos componentes se observa en la figura 6-1 (Chang et al., 2016, p. 2).

Los administradores del sistema inician sesión en el nodo de control e inician Ansible, proporcionándole un playbook y un host de destino para administrar. En lugar de un solo sistema para controlar, se puede especificar un grupo de hosts o un comodín. Ansible usa SSH como un transporte de red para comunicarse con los hosts administrados. Los módulos a los que se hace referencia en el playbook se copian en los hosts administrados. Luego se ejecutan, en orden, con los argumentos

especificados en los playbooks. Los usuarios de Ansible pueden escribir sus propios módulos personalizados, si es necesario, pero los módulos principales que vienen con Ansible pueden realizar la mayoría de las tareas de administración del sistema (Chang et al., 2016, p. 3).

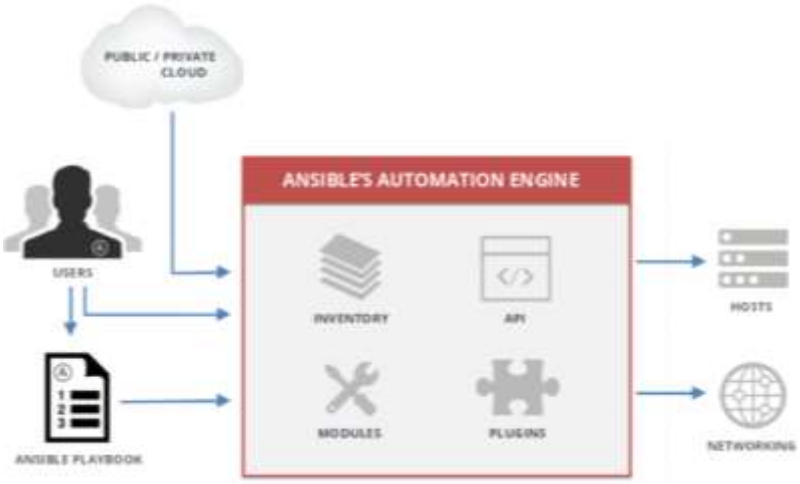


Figura 6-1: Arquitectura de Ansible
Fuente: (Chang et al., 2016)

La tabla 1-1 se enumera y se describe los componentes de Ansible que se mantienen en el nodo de control.

Tabla 1-1: Componentes del nodo de control Ansible

Componente	Descripción
Ansible configuración	Ansible tiene una configuración que define cómo se comporta. Estas configuraciones incluyen cosas tales como el usuario remoto para ejecutar comandos y contraseñas para proporcionar al ejecutar comandos remotos con sudo. Los valores de configuración predeterminados pueden ser reemplazados por variables de entorno o valores definidos en los archivos de configuración.
Host Inventory	El inventario del host Ansible define a qué grupos de configuración pertenecen los hosts.
Core modules	Los módulos principales son los módulos que vienen incluidos con Ansible, hay más de 400 módulos principales.

Custom modules	Los usuarios pueden ampliar la funcionalidad de Ansible escribiendo sus propios módulos y agregándolos a la biblioteca de Ansible. Los módulos generalmente se escriben en Python, pero se pueden escribir en cualquier lenguaje de programación interpretado (shell, Ruby, Python, etc.).
Playbooks	Son archivos escritos en sintaxis YAML que definen los módulos, con argumentos, para aplicar a los nodos administrados.
Connection Plugins	Complementos que permiten la comunicación con hosts gestionados o proveedores de la nube. Estos incluyen SSH nativo, paramiko SSH y local.
Plugins	Extensiones que mejoran la funcionalidad de Ansible. Los ejemplos incluyen notificaciones por correo electrónico y registro.

Realizado: YUNGA, Alex, 2018
Fuente: (Chang et al., 2016, p. 4)

1.3.1 Inventory (Inventario)

Un inventario de host define qué hosts maneja Ansible. Los hosts pueden pertenecer a grupos que normalmente se utilizan para identificar el rol de los hosts en el centro de datos. Un host puede ser miembro de más de un grupo. Hay dos formas en que se pueden definir los inventarios de host. Un inventario de host estático puede ser definido por un archivo de texto, o un inventario de host dinámico puede ser generado por proveedores externos (Heap, 2016, p. 19).

1.3.1.1 Inventario de host estático

Un inventario de host estático de Ansible que se muestra en la figura 7-1 define un grupo de hosts. Cada sección comienza con un nombre de grupo de host entre corchetes [ ]. A continuación, se enumeran las entradas de host para cada host gestionado en el grupo, cada uno en una sola línea, pueden tener los nombres de host o las direcciones IP de los hosts administrados que son miembros del grupo (Chang et al., 2016, p. 12).

```
[webservers]
localhost              ansible_connection=local
web1.example.com
web2.example.com:1234 ansible_connection=ssh ansible_user=ftaylor
192.168.3.7

[db-servers]
web1.example.com
db1.example.com
```

Figura 7-1: Inventario estático de Ansible
Fuente: (Chang et al., 2016, p. 12)

1.3.1.2 Inventario de host dinámico

La información de inventario de host también se puede generar dinámicamente. Las fuentes para la información de inventario dinámico incluyen proveedores de nube pública / privada, información del sistema Cobbler, una base de datos LDAP o una base de datos de administración de configuración (CMDB). Ansible incluye scripts que manejan información dinámica de host, grupo y variable de los proveedores más comunes, como Amazon EC2, Cobbler, Rackspace Cloud y OpenStack (Chang et al., 2016, p. 13-14).

1.3.2 Anatomía de un playbook

El playbook es el objeto de nivel superior que se ejecuta para automatizar los dispositivos de red. Un playbook usa YAML para definir el conjunto de tareas para automatizar, y cada playbook se compone de una o más plays (jugadas) (Edelman, 2016, p. 31).

Figura 8-1: Anatomía de un playbook
Fuente: (Chang et al., 2016, p. 4)

En la figura 10-1 se asigna un numero a cada una de la sección de un playbook a continuación en la tabla 2-1 se describe a que parte del playbook pertenece.

Tabla 2-1: Partes de un playbook

Numero	Sección
1	Plays
2	Tasks
3	Módulos

4	Variables

Realizado por: YUNGA, Alex, 2018

1.3.2.1 Plays (Jugadas)

Uno o más plays pueden existir dentro de un playbook de Ansible. Cada play consta de las siguientes dos partes importantes (Shah, 2015, p. 13):

- Qué configurar: se debe configurar un host o un grupo de hosts para ejecutar la jugada en contra. Además, se debe incluir información de conexión útil, como a qué usuario conectarse etc.,
- Qué ejecutar: esto incluye la especificación de las tareas que se ejecutarán, incluidos los componentes del sistema que se modificarán y el estado en el que deberían estar, por ejemplo, instaladas, iniciadas o más recientes.

1.3.2.2 Tasks

La sección de tareas es la última sección de cada play. Contiene una lista de acciones que va a realizar Ansible y el orden en que se va a ejecutar. Hay varios estilos en los que se puede expresar los argumentos de cada módulo. Las tareas representan lo que se automatiza de manera declarativa sin preocuparse por la sintaxis subyacente o "cómo" se realiza la operación. (Hall, 2015, p. 21).

Las tareas también pueden usar el parámetro de nombre al igual que los plays. Al igual que con los playbooks, el texto es arbitrario y se muestra cuando el playbook se ejecuta para mejorar la legibilidad durante la ejecución del playbook y los informes, es un parámetro opcional para cada tarea (Edelman, 2016, p. 32).

1.3.2.3 Módulos

Los módulos son scripts que vienen empaquetados con Ansible y realizan algún tipo de acción en un host. Es cierto que esa es una descripción bastante genérica, pero hay una gran variedad en los módulos de Ansible (Hochstein, 2014, p. 34).

Es fundamental comprender los módulos dentro de Ansible. Si bien cualquier lenguaje de programación se puede usar para escribir módulos Ansible, siempre y cuando devuelvan pares clave-valor JSON, casi siempre se escriben en Python (Edelman, 2016, p. 33).

La figura 9-1 describe la relación que existe entre cada uno de los componentes del playbook respetando la jerarquía que existe entre ellos.

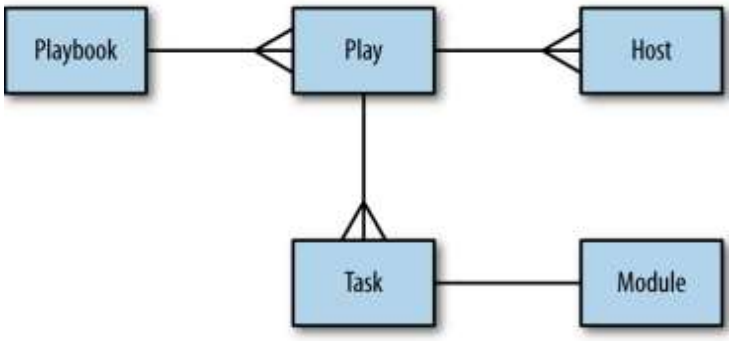


Figura 9-1: Diagrama de relación/entidad de un playbook
Fuente: (Hochstein, 2014, p. 35)

1.3.3 Variables

Las variables proporcionan una forma conveniente de administrar valores dinámicos para un entorno determinado en un proyecto Ansible. Las variables tienen nombres que consisten en una cadena que debe comenzar con una letra y que solo puede contener letras, números y guiones bajos. Al escribir un playbook, los administradores pueden usar sus propias variables y llamarlos en una tarea. Las variables se pueden definir en una desconcertante variedad de lugares dentro de un proyecto Ansible. Sin embargo, esto se puede simplificar a tres niveles básicos de alcance (Chang et al., 2016, p. 112-113):

- Alcance global: variables establecidas desde la línea de comando o la configuración Ansible.
- Alcance de juego: Variables establecidas en los plays y estructuras relacionadas.
- Alcance del host: variables establecidas en grupos de host y hosts individuales por el inventario, recopilación de datos o tareas registradas.

1.3.4 Facts (hechos)

Antes de ejecutar las tareas de un playbook se ejecuta primero GATHERING FACTS que recopila información, se conecta al host y consulta al host para obtener toda clase de detalles sobre el host: arquitectura de la CPU, sistema operativo, direcciones IP, información de memoria, información del disco y más. Esta información se almacena en variables que se llaman hechos, y se comportan como cualquier otra variable. Ansible implementa la recopilación de datos mediante el uso de un módulo especial denominado módulo de configuración. No necesita llamar a este módulo en su playbooks porque Ansible lo hace automáticamente cuando recopila hechos (Hochstein, 2014, p. 74-75).

1.3.5 Roles

Ansible proporciona mecanismos para descomponer trabajos complejos en piezas más pequeñas mediante el uso de roles que es el mecanismo principal para dividir un playbook en múltiples archivos como se observa en la figura 10-1. Esto simplifica la escritura de playbooks complejos, y también los hace más fáciles de reutilizar. Piense en un rol como algo que asigna a uno o más hosts. Cuando se utiliza roles se tiene una sección de roles en el playbook que espera una lista de roles (Hochstein, 2014, p. 147-149).

```
- name: deploy mezzanine on vagrant
  hosts: web
  vars_files:
    - secrets.yml
  roles:
    - role: database
      database_name: "{{ mezzanine_proj_name }}"
      database_user: "{{ mezzanine_proj_name }}"
    - role: mezzanine
      live_hostname: 192.168.33.10.xip.io
      domains:
        - 192.168.33.10.xip.io
        - www.192.168.33.10.xip.io
```

Figura 10-1: Ejemplo de roles en un playbook
Fuente: (Hochstein, 2014, p. 147)

Las funciones le permiten colocar sus variables, archivos, tareas, plantillas y manejadores en una carpeta, y luego incluirlos fácilmente. También puede incluir otros roles dentro de los roles, lo que efectivamente crea un árbol de dependencias. Similar a la tarea incluye, pueden tener variables

pasadas a ellos. Al usar estas características, debería ser capaz de crear roles independientes que sean fáciles de compartir con los demás (Hall, 2015, p. 66).

Los playbooks y los roles son muy similares, pero muy diferentes al mismo tiempo. Un playbook es un archivo independiente que Ansible puede ejecutar y que contiene toda la información necesaria para establecer el estado de una máquina según lo esperado. Esto significa que un playbook puede contener variables, tareas, handlers, roles e incluso otros playbooks, todos en el mismo archivo. No necesita ningún otro archivo para realizar su tarea (Heap, 2016, p. 49).

Se puede pensar en un rol como un playbook que se divide en múltiples archivos diferentes. En lugar de tener un único archivo que contiene todo lo que necesita, hay un archivo para variables, uno para tareas y otro para controladores. Sin embargo, no puedes ejecutar un rol por sí mismo; debe incluirlo dentro de un playbook junto con la información sobre qué hosts ejecutar (Heap, 2016, p. 49).

La figura 11-1 se muestra como ejecutar un playbook, para ello se utiliza el comando *ansible-playbook* que permite correr cualquier playbook.

```
ntc@ntc:~/ansible/multivendor$ ansible-playbook -i inventory
demo.yml

PLAY [PLAY 1 - CISCO NXOS]
****************************************************

TASK: [ENSURE VLAN 100 exists on Cisco Nexus switches]
*********************
changed: [nx1]
```

Figura 11-1: Ejecución de un playbook
Fuente: (Edelman, 2016, p. 42)

1.4 Introducción a YAML

YAML es un acrónimo recursivo que significa “YAML no es un lenguaje de marcado”, que hace hincapié en su diseño como formato de almacenamiento de datos. Es un lenguaje de serialización de lectura humana ligero, diseñado principalmente para ser fácil de leer y editar. Al agregar un sistema de tipado simple y un mecanismo de alias sobre las tres estructuras de datos más comunes utilizadas

al serializar (mapeos, arrays y cadenas), se forma un lenguaje que es fácil de usar, al tiempo que incluye funciones más complejas (Eriksson y Hallberg, 2011, p. 8).

Los archivos YAML opcionalmente comienzan con un inicio de tres guiones de marcador de documento y opcionalmente terminan con un marcador de fin de archivo de tres puntos. Entre los marcadores de documento inicial y final, las estructuras de datos se representan mediante un formato de esquema que utiliza caracteres espaciales para la sangría. No existe un requisito estricto con respecto a la cantidad de caracteres de espacio utilizados para la sangría, además de que los elementos de datos deben tener más sangría que sus padres para indicar relaciones anidadas (Chang et al., 2016, p. 62).

Las etiquetas se utilizan principalmente para asociar los metadatos a los nodos, por ejemplo, al decirle al analizador YAML qué tipo de usuario define el objeto que representa un nodo (en el que el analizador puede deserializar los datos). Los tipos de datos básicos son (Eriksson y Hallberg, 2011, p. 9-10):

- Números (hexadecimales / octales, enteros, números de coma flotante)
- Cadenas (con soporte Unicode)
- Boolean (verdadero / falso)
- Fechas y marcas de tiempo
- Mapas (matrices asociativas / objetos con pares clave-valor)
- Secuencias (matrices, listas ordenadas)
- Nulo

1.4.1 Las tres reglas de YAML

Regla # 1: YAML usa un esquema de espaciado fijo para representar las relaciones entre las capas de datos. No se debe usar tabulación (Ulinic y House, 2017, p. 9).

Regla # 2: Los dos puntos se usan en YAML para representar matrices asociativas; en otras palabras, asignaciones uno a uno entre una clave y un valor. La misma regla se puede extender a un nivel superior y usar pares clave-valor anidado donde se nota el uso de la sangría (Ulinic y House, 2017, p. 9).

Regla # 3: Los guiones se utilizan para representar una lista de elementos (Ulinic y House, 2017, p. 9).

En la figura 12-1 se observa cada una de las reglas importantes utilizadas en YAML.

Figura 12-1: Reglas de YAML
Fuente: (Hochstein, 2014, p. 9)

1.4.2 Usando YAML en playbooks de Ansible

Los playbooks de Ansible están escritos en un formato de lista. Los elementos en la lista son pares clave / valor. La composición de un playbook requiere solo un conocimiento básico de la sintaxis de YAML que se observa en la figura 13-1 y se listan a continuación (Chang et al., 2016, p. 524, p. 63-64):

- ***Inicio y final de los marcadores de archivos:*** Los archivos YAML se inician con un comienzo de marcador de documento formado por tres guiones y terminan con un marcador de fin de documento compuesto por tres puntos.
- ***Strings***: Las cadenas se pueden escribir usando el estilo estándar en línea (con o sin comillas) o con notación de bloque donde un símbolo inicial determina cómo se deben manejar las nuevas líneas en el documento.

- ***Diccionarios:*** Los pares de datos clave / valor utilizados en YAML también se conocen como diccionarios o matrices asociativas. En los pares clave / valor, las claves se separan de los valores usando una cadena delimitadora compuesta por dos puntos y un espacio.
- ***Listas:*** En YAML, las listas son como matrices en otros lenguajes de programación. Para representar una lista de elementos, se usa un solo guión seguido de un espacio para prefijar cada elemento de la lista.
- ***Comentarios:*** Los comentarios también se pueden usar para ayudar a la legibilidad. En YAML, los comentarios se inician con un símbolo de almohadilla (#) y pueden existir al final de cualquier línea, en blanco o no en blanco. Si se usa en una línea que no está en blanco, preceda al símbolo de almohadilla con un espacio.

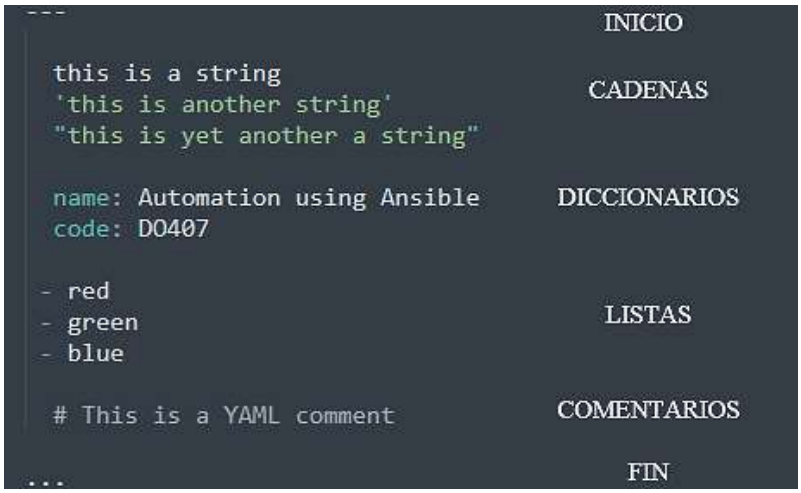


Figura 13-1: Estructura de un archivo YAML
Realizado por: YUNGA, Alex, 2018

1.4.3 YAML Lint

Para los administradores que no están familiarizados con Python, hay muchas herramientas de verificación de sintaxis YAML en línea disponibles. Un ejemplo es el sitio web YAML Lint que se muestra en la figura 14-1. Que permite copiar y pegar los contenidos YAML en un playbook en el formulario en la página de inicio. La página web informa los resultados de la verificación de sintaxis, y muestra una versión formateada del contenido enviado originalmente. El sitio web de YAML Lint informa el mensaje "¡YAML válido!" cuando se envían los contenidos correctos de YAML (Chang et al., 2016, p. 524, p. 65).

Figura 14-1: Página web de YAML Lint
Fuente: http://yamllint.com/

1.5 Introducción a Jinja2

El nombre Jinja se originó a partir de la palabra japonesa templo que se observa en la figura 15-1, que es similar en fonética a la plantilla de la palabra. Algunas de las características importantes de Jinja2 son (Shah, 2015, p. 40):

- Es rápido y compilado justo a tiempo con el código de byte de Python
- Tiene un entorno opcional de espacio aislado
- Es fácil depurar
- Es compatible con la herencia de plantillas

Figura 15-1: Origen de Jinja
Fuente: (Jinja, 2011)

Las plantillas le permiten construir dinámicamente un playbook y los datos relacionados, como variables y otros archivos de datos permitiendo a Ansible usar el lenguaje de plantillas Jinja2. Este tipo de archivos debe terminar con una extensión ".j2", pero se puede usar cualquier nombre. De

forma similar a los archivos, que será imposible encontrar el archivo “main.yml” en el directorio de plantillas (Mohaan y Raithatha, 2014, p. 144) (Hull, 2016).

Los filtros Jinja2 son funciones simples de Python que toman algunos argumentos, los procesan y devuelven el resultado. Por ejemplo, considere el siguiente comando: {{myvar | filter}} ya que, myvar es una variable; Ansible pasará myvar al filtro Jinja2 como argumento. El filtro Jinja2 lo procesará y devolverá los datos resultantes (Mohaan y Raithatha, 2014).

Ansible utiliza el sistema de plantillas Jinja2 para modificar los archivos antes de que se distribuyan a los hosts administrados. En términos generales, es preferible evitar modificar los archivos de configuración a través de la lógica en las plantillas. Sin embargo, las plantillas pueden ser útiles cuando los sistemas necesitan tener versiones ligeramente modificadas del mismo archivo (Chang et al., 2016, p. 224).

1.5.1 Unicode

Jinja2 está utilizando Unicode internamente, lo que significa que debe pasar un objeto Unicode a la función de renderizado o cadenas de bytes que solo consisten en caracteres ASCII. Las nuevas líneas de adición se normalizan en una secuencia de final de línea, que es por estilo UNIX predeterminado. Con Python 2.6 es posible hacer unicode por defecto por nivel de módulo y con Python 3 será el predeterminado (Jinja, 2011, p. 8).

Se recomienda utilizar utf-8 como módulos de codificación y plantillas de Python, ya que es posible representar cada carácter Unicode en utf-8 y porque es compatible con ASCII. Para Jinja2, se supone que la codificación predeterminada de plantillas es utf-8. No es posible usar Jinja2 para procesar datos que no sean Unicode. La razón de esto es que Jinja2 usa Unicode ya en el nivel de idioma. Por ejemplo, Jinja2 trata el espacio sin interrupción como espacios en blanco válidos dentro de expresiones que requieren conocimiento de la codificación u operación en una cadena Unicode (Jinja, 2011, p. 9).

1.5.2 Las tres reglas de Jinja2

Regla # 1: Dobles llaves significan la sustitución de una variable por su valor.

Regla #2: Un condicional dentro de una plantilla crea una ruta de decisión. El motor considerará el condicional y elegirá entre dos o más posibles bloques de código. Siempre hay un mínimo de dos: una ruta si se cumple el condicional (evaluado como verdadero) y una ruta implícita de un bloque vacío (Keating, 2015, p. 49).

Regla #3: Un bucle le permite crear secciones creadas dinámicamente en archivos de plantilla, y es útil para cuando se sabe que necesita operar en un número desconocido de elementos de la misma manera (Keating, 2015, p. 53).

Todas estas reglas se muestran en la figura 17-1, que describe los condicionales y los bucles que son de suma importancia para la estructura de una plantilla.

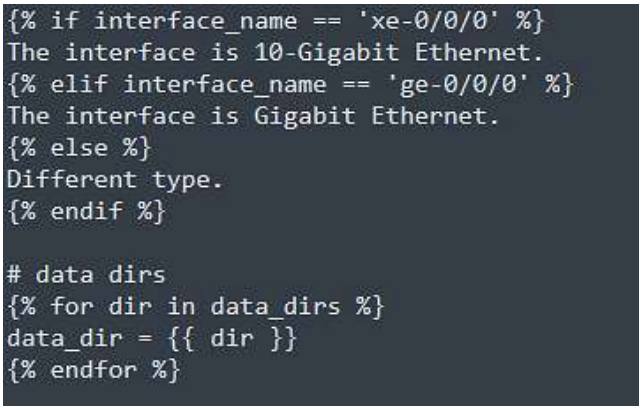


Figura 16-1: Condicionales y bucles de un archivo JINJA
Fuente: (Keating, 2015)

1.5.3 Delimitadores

Las variables o expresiones lógicas se colocan entre etiquetas o delimitadores. Por ejemplo, las plantillas Jinja2 usan {% EXPR%} para expresiones o lógica (por ejemplo, bucles), mientras que {{EXPR}} se usan para dar salida a los resultados de una expresión o una variable para el usuario

final. La última etiqueta, cuando se representa, se reemplaza con un valor o valores, y el usuario final la ve. Use la sintaxis {# COMMENT #} para adjuntar comentarios (Chang et al., 2016, p. 224).

1.6 Herramientas de emulación

A diferencia de los simuladores de red que brindan limitadas funcionalidades, según la implementación realizada por cada programador, un emulador permite cargar la imagen de uno o más sistemas operativos en una PC o servidor, lo cual brinda la posibilidad de implementar casi todas las funcionalidades de un equipo real en un entorno de laboratorio. En la actualidad, diversos fabricantes han implementado sus propios emuladores, tales como Junosphere de Juniper Networks y eNSP de Huawei Technologies (Ocampo Zuñiga, 2015).

Existen emuladores en el mercado que ofrece más funcionalidades y que no se limitan a una sola marca de fabricantes de productos para networking, tal es el caso de GNS3 y EVE-NG que se observa en la figura 18-1.

Figuran 17-1: Emuladores de red
Fuente: http://areaip.blogspot.com/2016/09/asav-en-gns3.html / https://twitter.com/eve_ng_team

1.6.1 GNS3

GNS3 usa las librerías de Dynagen para crearle una interfaz gráfica (GUI). Sus principales funciones son editar el archivo de texto .net y realizar las operaciones del CLI hechas por Dynagen y Dynamips. Adicionalmente incorpora la capacidad de simular PCs. Dicho de otro modo, GNS3 es un front-end gráfico de Dynamips y Dynagen, los cuales son las herramientas que realmente permiten la emulación

de IOS Cisco. Más, concretamente, Dynagen ofrece una interfaz de línea de comandos más simple a Dynamips, el cual es en última instancia, el responsable de la emulación de la IOS. Usando un simple editor de textos, un usuario podría crear su propio fichero de topología con la red a emular por Dynagen. Precisamente, GNS3 facilita este proceso creando para ello una sencilla interfaz gráfica que abstrae al usuario de los detalles de configuración del escenario (Neumann, 2015).

GNS3 se puede considerar como un lugar de encuentro para una variedad de emuladores de sistema operativo. El más conocido e importante de estos es Dynamips. Dynamips le permite emular los enrutadores de Cisco y proporciona una colección de dispositivos e interfaces genéricos. Otros emuladores compatibles con GNS3 son los siguientes (Welsh, 2013, p. 8):

- Qemu: proporciona la emulación de dispositivos ASA de Cisco, enrutadores Juniper, enrutadores Vyatta y hosts Linux.
- Pemu: esta es una variación de Qemu utilizada expresamente para cortafuegos Cisco PIX.
- VirtualBox: proporciona emulación de enrutadores Juniper, enrutadores Vyatta, hosts Linux y hosts de Windows.

Cada instancia de un enrutador o cualquier otro dispositivo que ejecute generará una copia de su propio sistema operativo que competirá por los ciclos de RAM y CPU de su computadora host. Ahora considere que dispositivos como enrutadores y cortafuegos requieren algún tipo de aplicación de terminal para darle acceso, tal es el caso de Gnome Terminal, iTerm2, Konsole, PuTTY, SecureCRT, SuperPutty, TeraTerm, Windows Telnet client o incluso Xterm. Finalmente, hay dos aplicaciones complementarias más que no son esenciales, pero que a menudo se usan junto con GNS3. Estas aplicaciones son las siguientes (Welsh, 2013, p. 9):

- Wireshark: es una popular aplicación de captura de paquetes de código abierto.
- Virtual PC Simulator (VPCS): Esto le permite simular hasta nueve PC que puede usar para hacer ping, traceroute y más.

1.6.2 EVE-NG

EVE-NG es Emulated Virtual Environment – Next Generation que en español es Entorno Virtual Emulado - Próxima Generación. En algunas palabras triviales, EVE-NG le ofrece herramientas para

usar en dispositivos virtuales e interconectarlos con otros dispositivos virtuales o físicos. Muchas de sus características simplifican enormemente las usabilidades, la capacidad de uso, la capacidad de administración, la interconexión, la distribución y, por lo tanto, la capacidad de comprender y compartir topologías, trabajos, ideas, conceptos o simplemente "laboratorios". Esto puede significar simplemente que reducirá el costo y el tiempo para configurar lo que necesita o le podría permitir realizar tareas que no hubiera pensado que se pudieran hacer así de simple (Dzerkals, 2017).

CAPÍTULO II

2. MARCO METODOLÓGICO

En la Fig. 1-2 se presenta un diagrama con la metodología que se aplicó al realizar el trabajo de titulación.

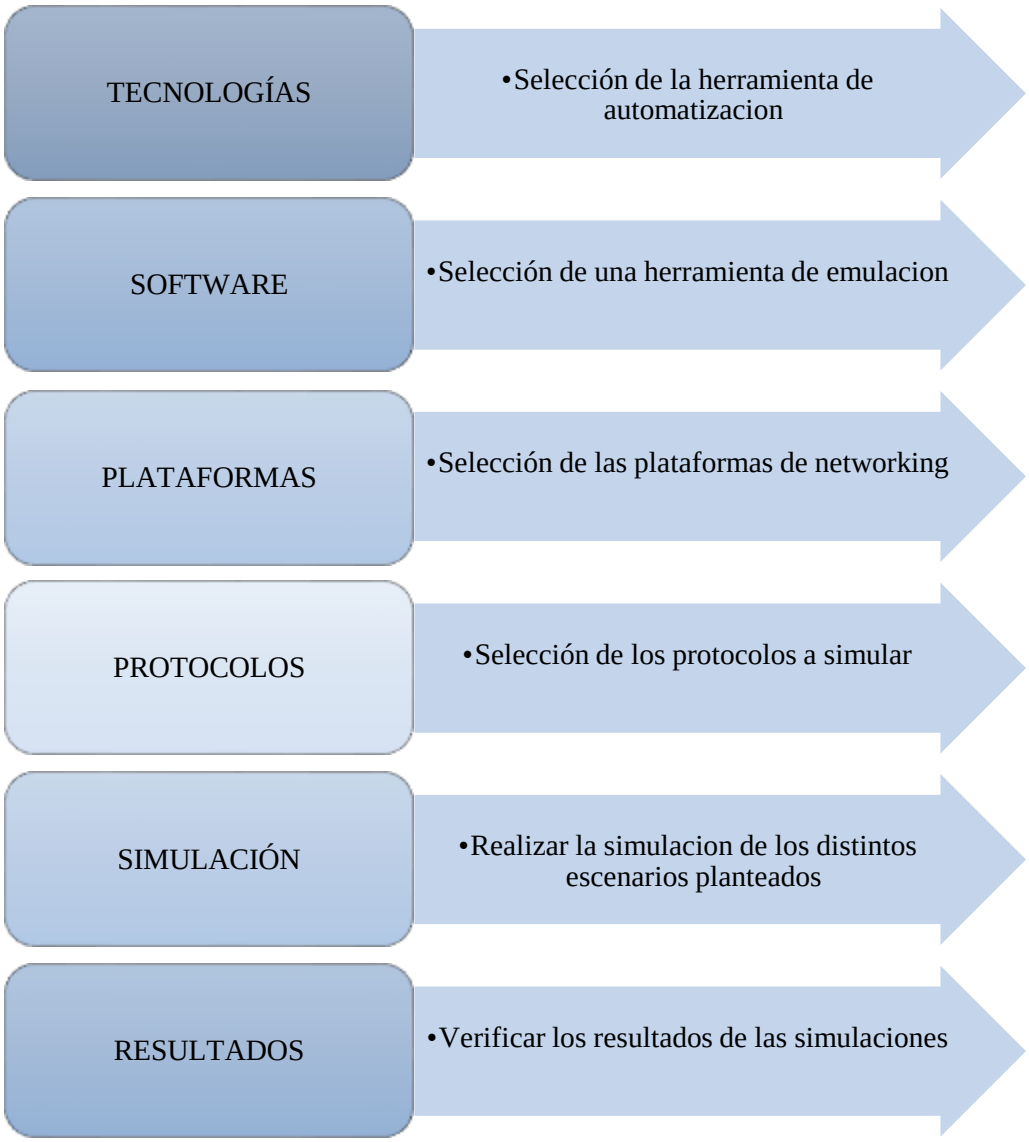


Figura 1-2: Metodología del trabajo de titulación
Realizado: YUNGA, Alex, 2018

En este capítulo se describe la justificación de la elección de la herramienta de automatización, las plataformas de networking y los protocolos empleados para la realización del trabajo de titulación mediante la simulación de 6 topologías que se describen a continuación:

En la topología 1, 2 y 3 se hace uso de Cisco y se va a comprobar la configuración de VLANS, BGP y MPLS respectivamente, en la topología 4 también se hace uso de Cisco y se va a mostrar la configuración de un firewall, en la topología 5 y 6 se utiliza Juniper y Arista respectivamente donde en cada una de ellas se muestra la configuración de BGP.

2.1 Selección de una herramienta de automatización

En el capítulo anterior se dio a conocer las distintas características que ofrece las herramientas de automatización disponibles en el mercado por lo que ahora se realizara una comparación de estas para determinar cuál es la que ofrece las mejores características al momento de efectuar el trabajo de titulación.

Tabla 1-2: Características de las herramientas de automatización

	ANSIBLE	SALTSTACK	CHEF	PUPPET
Escalabilidad	Pequeños-medios-grandes	Pequeños-medios-grandes	Pequeños-medios-grandes	Pequeños-medios-grandes
Facilidad de configuración	Sin agentes (SSH)	Master y Minions	Servidor y Clientes CHEF	Servidor y Clientes PUPPET
Disponibilidad	Instancia primaria y secundaria	Maestros múltiples	Arquitectura de varios maestros	Servidor primario, Servidor Chef y Servidor Backup
Interoperabilidad	Ansible también admite	Salt Master solo funciona	Chef Server solo funciona en	Puppet Master solo funciona

	máquinas Windows, pero el servidor Ansible debe estar en una máquina Linux / Unix	en Linux / Unix, pero Salt Minions también puede funcionar en Windows	Linux / Unix, pero Chef Client y Workstation también pueden estar en Windows	en Linux / Unix, pero Puppet Agent también funciona en Windows
Tipo de lenguaje	Declarativo	Declarativo	Declarativo	Imperativo
Agentes de traducción	Push	Push	Pull	Pull
Documentación	documentación, referencias, tutoriales, presentaciones, videos			
Lenguaje de desarrollo	Python	Python	Ruby	Ruby
Comunicación	SSH	ZeroMQ	SSL	SSL
Repositorio de la comunidad	Ejemplos de Ansible en Github	SaltStartes	Cookbooks	Puppet Forge
Interfaces de control	Playbooks: archivos de texto JSON, YAML	States: YAML y otras herramientas de plantilla estándar	Recipes: Ruby	Manifest: Lenguaje propietario

Fuente: (Carbonell y García, 2016), (Edureka, 2017) y (KWNETAPPS, 2017)

Realizado por: YUNGA, Alex, 2018

Para este proceso de comparación se ha empleado la escala de Likert que es una escala muy utilizada para medir variables cualitativas y porque se considera fácil de elaborar; además, permite lograr altos niveles de confiabilidad. Basándose en la información expresada en la tabla 1-2 se procede a determinar que herramienta de automatización es la más adecuada para el proyecto, mediante la asignación de valores de 1 a 3, en donde 3 representa *excelente*, 2 representa *normal* y 1 representa *deficiente* en las variables cualitativas para las herramientas de automatización.

Tabla 2-2: Asignación de valores para las características de las herramientas de automatización

	ANSIBLE	SALTSTACK	CHEF	PUPPET
Escalabilidad	Se asigna los siguientes valores: pequeños (1), medios (2) y grandes (3) y todas son altamente escalables.			
Facilidad de configuración	Se habla sobre la facilidad de instalación, Si utiliza algún agente externo para su configuración (1) y si es agentless (3).			
Disponibilidad	Todas las herramientas son altamente disponibles, lo que significa que hay varios servidores o varias instancias presentes cuando una de ellas falla.			
Interoperabilidad	En estas herramientas, el servidor principal o también puede decir que la máquina de control tiene que estar en Linux / Unix, pero sus esclavos o los nodos que tienen que configurar pueden estar en Windows.			
Tipo de lenguaje	La programación imperativa es cada vez más costosa (1) y la programación declarativa no (3)			
Agentes de traducción	Puppet y Chef siguen las configuraciones push (extracción) y Ansible y Saltstack siguen la configuración pull (inserción). En la configuración de pull, todas las configuraciones presentes en el servidor central se enviarán a los nodos mientras que, en la configuración push, los nodos esclavos extraerán automáticamente todas las configuraciones desde el servidor central sin ningún comando.			
Documentación	Todas disponen de mucha documentación (3)			
Lenguaje de desarrollo	La sintaxis de Ruby es similar a la de Python; por lo tanto, a Ansible y SaltStack (3) por su facilidad de aprendizaje, Chef (2) se debe ser un programador y Puppet (1) ya que cuenta con su propio lenguaje de programación llamado Puppet DSL y no es muy fácil de aprender.			
Comunicación	SSH y SSL tienen similitudes muy fuertes en sus atributos de seguridad (3) y ZeroMQ (2)			
Repositorio de la comunidad	Todos cuentan con repositorios donde se pueden encontrar ejemplos de configuración.			
Interfaces de control	Emplean sus propios mecanismos de configuración ya sea atraves de playbooks, states, recipes y manifest.			

Realizado por: YUNGA, Alex, 2018

Tabla 3-2: Calificación para la comparación de las herramientas de automatización

	ANSIBLE	SALTSTACK	CHEF	PUPPET
Escalabilidad	3	3	3	3
Facilidad de configuración	3	1	1	1
Disponibilidad	3	3	3	3
Interoperabilidad	3	3	3	3
Tipo de lenguaje	3	3	3	1
Agentes de traducción	3	3	3	3
Documentación	3	3	3	3
Lenguaje de desarrollo	3	3	2	1
Comunicación	3	1	3	3
Repositorio de la comunidad	3	3	3	3
Interfaces de control	3	3	3	3
TOTAL	**33**	**29**	**30**	**27**

Realizado por: YUNGA, Alex, 2018

En la tabla 3-2 se analizó las características propias de cada herramienta de automatización y se determina que ANSIBLE ofrece una gama más variada de parámetros, los cuales influyen directamente al momento de realizar la simulación.

Además, a diferencia de otras soluciones, que en su mayoría se basan en agentes, Ansible funciona exclusivamente en SSH. No se requiere agente. Por lo tanto, puede sentarse y relajarse, ya que no hay ningún paquete adicional en su sistema de producción. Por otra parte los playbooks de Ansible están basados en YAML ya que son muy fáciles de leer, comprender y mantener, lo que implica una curva de aprendizaje muy pequeña (Das, 2016).

2.2 Selección de una herramienta de emulación

Para poder realizar las simulaciones se deberá escoger una herramienta de emulación analizando sus características en la tabla 4-2.

Tabla 4-2: Características de las herramientas de emulación

	EVE-NG	GNS3
Escalabilidad	Alto	Alto
Facilidad de instalación	Regular	Fácil
Procesador	4 núcleos lógicos	2 núcleos lógicos
Memoria	8 GB RAM	4 GB RAM
Almacenamiento	40 GB	1 GB
Red	LAN	LAN
Sistema Operativo	Windows 7, 8, 10 o Linux Desktop	Windows 7, 8, 10
Tamaño de la aplicación para instalar	861.9 MB (ISO) 4.80 GB (OVA)	377.5 MB (Aplicación + VM)
Documentación	Poco	Mucho
Licencia	Pagada	Gratis
Interoperabilidad	Alto	Alto
Administración	Entornos empresariales más grandes y con mayores prestaciones	Interfaz amigable con el usuario

Fuente: http://www.eve-ng.net, https://gns3.com y (Dzerkals, 2017)
Realizado por: YUNGA, Alex, 2018

Basándose en la información expresada en la tabla 4-2 se procede a determinar que herramienta de automatización es la más adecuada para el proyecto, mediante la asignación de valores de 1 y 2, en donde 2 representa *excelente*, y 1 representa *deficiente* en las variables cualitativas para las herramientas de emulación.

Tabla 5-2: Calificación para la comparación de las herramientas de emulación

	EVE-NG	GNS3
Escalabilidad	2	2
Facilidad de instalación	1	2
Procesador	1	2
Memoria	1	2
Almacenamiento	1	2
Red	2	2
Sistema Operativo	2	2
Tamaño de la aplicación para instalar	1	2
Documentación	1	2
Licencia	1	2
Interoperabilidad	2	2
Administración	2	2
TOTAL	**17**	**24**

Realizado por: YUNGA, Alex, 2018

En la tabla 5-2 se analizó las características propias de las dos herramientas de emulación y se determinó que GNS3 ofrece un menor consumo de hardware (Memoria RAM, procesador). Otra de las características que también influye en la selección de GNS3 es porque ya se ha venido utilizado esta herramienta de emulación durante la carrera y se está muy familiarizada con la misma.

2.3 Plataformas de networking y VM GNS3

Al emplear GNS3 se nos limita el uso de las plataformas de networking que están listadas en la tabla 6-2 a aquellas que sean compatibles con dicha herramienta de emulación. En la página oficial de GNS3 (**https://gns3.com**) se muestra la información de las plataformas de networking disponibles, y para este análisis se utiliza las últimas versiones de cada plataforma.

Tabla 6-2: Características de las plataformas a analizar

	ARISTA	JUNIPER	CISCO	CUMULUS	F5	VYOS
RAM	2048 MB	1024 MB	512 MB	512 MB	4096 MB	512 MB
NOMBRE Y VERSIÓN	Arista vEOS 4.20.1F	Juniper vQFX RE 17.4R1	Cisco IOSv 15.6(2)T	Cumulus VX 3.6.2	F5 BIG-IP LTM VE 13.1.0 HF5	VyOS 1.2.0-beta1
TAMAÑO DE ARCHIVO	667 MB	554 MB	129 MB	1151 MB	4402 MB	243 MB
KMV en GNS3 VM	SI	SI	SI	SI	SI	NO

Realizado por: YUNGA, Alex, 2018

Fuente: https://docs.gns3.com/appliances

Se va a elegir las plataformas que cumplan todas las características requeridas en la tabla 7-2 para realizar la implementación del trabajo de titulación, una característica que no cumpla y la plataforma quedara automáticamente descartada. Estas características son variables cualitativas y cuantitativas que permiten conocer los parámetros de manejo e instalación de dichas plataformas.

Tabla 7-2: Comparación de los requisitos de las plataformas de networking

	Juniper	Cisco	Vyatta	Cumulus	F5	Arista
Acceso a la plataforma	✓	✓	✓	✓	X	✓
Configuración inicial (usuario, dirección IP)	✓	✓	X	✓	X	✓
Acceso por SSH	✓	✓	X	✓	X	✓
Consumo de RAM <2048 MB	✓	✓	✓	✓	X	✓
Tamaño del archivo <1500 MB	✓	✓	✓	✓	X	✓
Verificación de resultados	✓	✓	X	X	X	✓

Realizado por: YUNGA, Alex, 2018

Fuente: https://docs.gns3.com/appliances

Para la selección de dichas plataformas la característica mas relevante es el consumo de memoria RAM: para Arista el consumo de memoria es 2048 MB , Cisco de 512 MB y Juniper de 1024 MB; esta característica limita la cantidad de dispositivos que se vaya a emplear para realizar la implementación. Por ejemplo, en la topología donde se utiliza 3 dispositivos Arista el consumo de memoria RAM seria de 6 GB y se quiere utilizar un dispositivo más la VM de GNS3 nos da un mensaje indicando que no se puede administra ese dispositivo debido a que no se cuenta con la memoria RAM requerida por dicho dispositivo.

No se puede analizar las características de F5 debido a que la plataforma tiene licencia y no se tiene acceso a ella y también la plataforma de Vyatta ya que esta plataforma no guarda los cambios iniciales que se necesita para realizar la implementación.

En la tabla 7-2 se analizó los requisitos de las distintas plataformas y se determinó que CISCO, JUNIPER y ARISTA cumplen con los parámetros, los cuales influyen directamente con las exigencias del computador.

2.3.1 Características del computador y la VM de GNS3

Las características del computador a utilizar se muestran en la tabla 8-2, dichas características son indispensables debido al consumo de recursos de memoria RAM y del procesador para el uso de la máquina virtual de GNS3 que esta virtualizada en VM Ware y del mismo GNS3.

Tabla 8-2: Características del computador

Fabricante	ASUSTek Computer Inc.
Modelo	FX504GD-E4303T
Sistema Operativo	Microsoft Windows 10 de 64 bits
Procesador	Intel Core i7-8750H
Frecuencia de Procesador	2.2 GHz - 3.9 GHZ
Memoria RAM	16 GB
Capacidad de almacenamiento	1 TB

Fuente: YUNGA, Alex, 2018

Para la virtualización se utiliza 8 GB de RAM y 6 procesadores como se muestra en la figura 2-2.

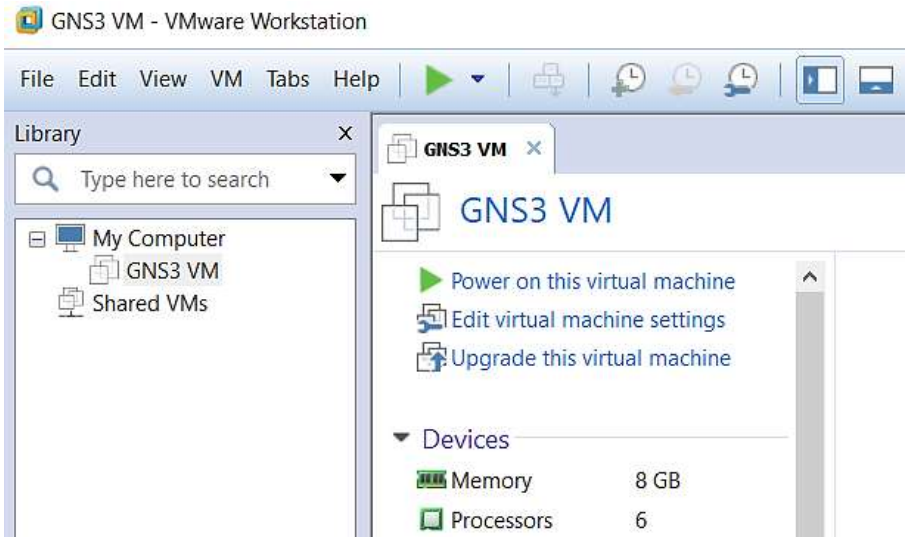


Figura 2-2: Recursos para la máquina virtual de GNS3
Fuente: YUNGA, Alex, 2018

2.4 Protocolos para la simulación

2.4.1 VLANS

Las razones del porque se utilizó VLANS en nuestra topología es por la fácil administración, seguridad y confidencialidad que brindan, limitando a un usuario no autorizado que no pueda acceder a información, recursos y archivos que no le correspondan y son muy empleadas en redes institucionales y corporativas tanto públicas como privadas ya que están configuradas de forma jerárquica dividiéndose en varios grupos de trabajo. Tanto es la importancia de las VLANS que están relacionadas con MPLS y otras importancias.

2.4.2 BGP

BGP es un protocolo extremadamente complejo utilizado a través de Internet y dentro de empresas multinacionales. La función de un protocolo de enrutamiento de pasarela externa, como BGP es el de

comunicar varios sistemas autónomos (AS), el motivo por el cual se escogió este protocolo es porque en Ecuador existen diferentes proveedores de internet que utilizan dicho protocolo para intercomunicarse entre sí.

2.4.3 MPLS

MPLS posibilita a los operadores de telecomunicaciones ofrecer múltiples servicios, en redes IP este servicio es Redes Privadas Virtuales (MPLS/VPN) que es un método flexible para transportar y enrutar varios tipos de tráfico de red utilizando un backbone MPLS. Se selecciona este protocolo porque los SP (Service Providers) del Ecuador utilizan este protocolo como mecanismo para las transmisiones de sus datos tales como CNT E.P, Transnexa y Telconet.

2.5 Configuraciones generales de las distintas topologías

2.5.1 Topología de red

En este apartado se da a conocer los componentes de una topología general de red con Ansible como se muestra en la figura 3-2. A continuación, se da a conocer la arquitectura general para todas las topologías. En el ***nodo central*** se encuentra todos los archivos de configuración que están escritas en YAML y JINJA2, en este nodo se encuentra Ansible que es el encargado de automatizar toda la red.

En el ***nodo secundario*** se encuentra todos los dispositivos de red que van a recibir las configuraciones del nodo central tales como routers, switches y firewalls.

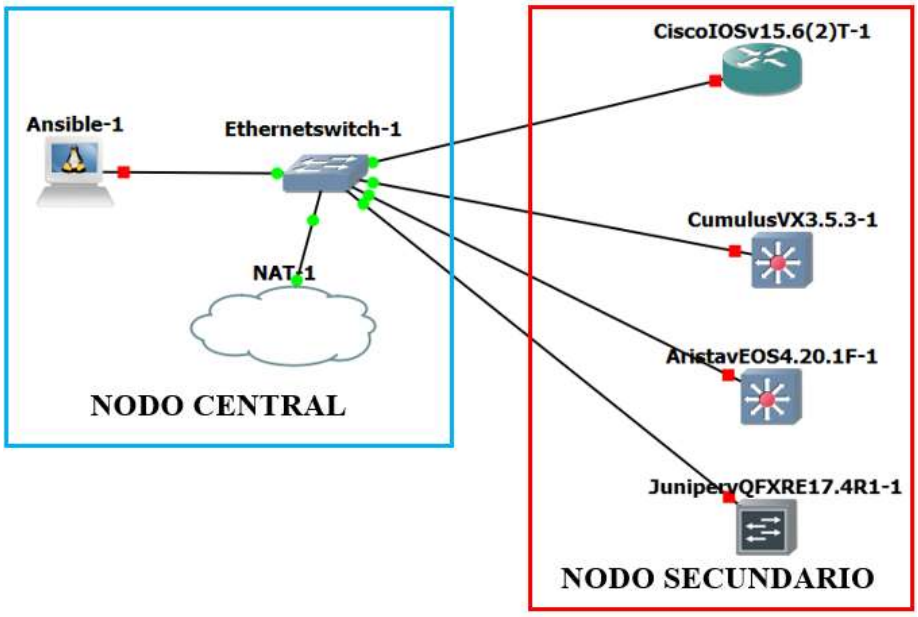


Figura 3-2: Topología general de una red con Ansible
Fuente: YUNGA, Alex, 2018

2.5.2 Diagrama de flujo de las configuraciones

No hay un orden específico para realizar la implementación, pero en la figura 4-2 se da a conocer los pasos que se siguió durante la configuración de los dispositivos del nodo secundario y del nodo principal.

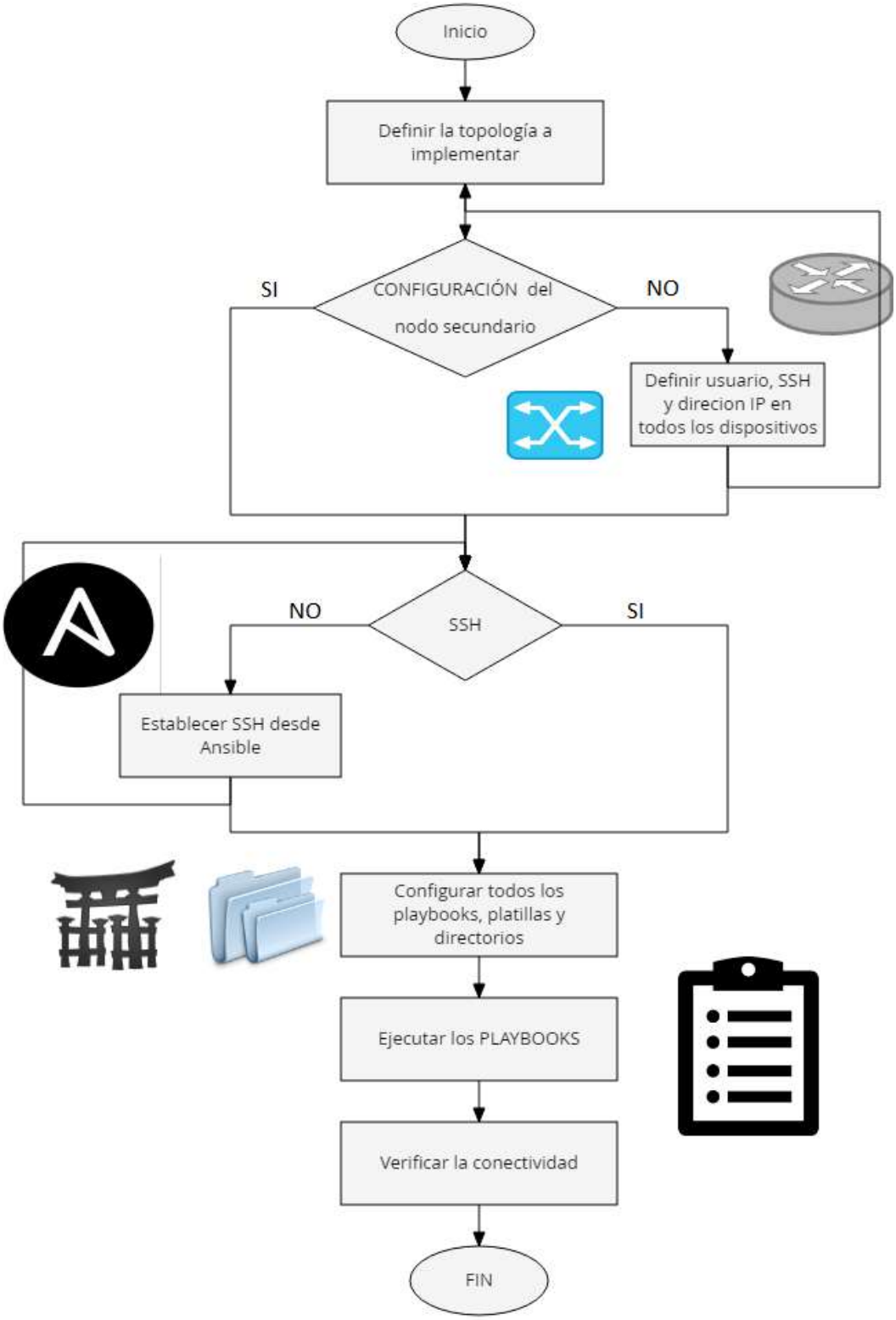


Figura 4-2: Pasos para realizar la implementación
Realizado por: YUNGA, Alex, 2018

2.5.3 Comandos de Linux empleados en el nodo central

Ansible es parte de la distribución de Linux Fedora, heredada de Red Hat Inc que está disponible para Red Hat Enterprise Linux, CentOS y Scientific Linux como también para otros sistemas operativos y utiliza los distintos comandos de Linux para la creación, eliminación y visualización de los ficheros, directorios y scripts que se describe en la tabla 9-2.

Tabla 9-2: Comandos de Linux utilizados en Ansible

COMANDO	DESCRIPCIÓN
mkdir	mkdir (de make directory o crear directorio), crea un directorio nuevo tomando en cuenta la ubicación actual.
Nano	nano es un editor de texto más básico, y cuyo uso es mucho más sencillo. Es el editor de texto nativo en sistemas como Ubuntu y es una alternativa a editores como emacs o vi, ya que opera más fácilmente y provee mayor interactividad.
Cat	cat (de concatenar), es una maravillosa utilidad que permite visualizar el contenido de un archivo de texto sin la necesidad de un editor.
rm	mm (de remove o remover), es el comando necesario para borrar un archivo o directorio.
rm –rf	Este comando también presenta varias opciones. La opción -r borra todos los archivos y directorios de forma recursiva. Por otra parte, -f borra todo sin pedir confirmación.
cd	cd (de change directory o cambiar directorio), es como su nombre lo indica el comando que necesitarás para acceder a una ruta distinta de la que te encuentras. Las opciones de cd varían: cd sin parámetros cambia al directorio de trabajo predeterminado del usuario y **cd ..** cambia al directorio un nivel más bajo que el actual.
Ls	ls (de listar), permite listar el contenido de un directorio o fichero.
Clear	clear (de limpiar), es un sencillo comando que limpiara nuestra terminal por completo dejándola como recién abierta.

Fuente: https://hipertextual.com/archivo/2014/04/comandos-basicos-terminal/

Realizado por: YUNGA, Alex, 2018

2.5.4 Configuración del nodo principal

En el nodo principal se realiza dos configuraciones importantes. La primera configuración es para que la interfaz eth0 de Ansible (maquina central) que tenga una dirección IP.

Con el comando ***nano /etc/network/interfaces*** se edita el script para habilitar las opciones de ***auto eth0*** *y* ***iface eth0 inet dhcp***, borrando el numeral (#) como se muestra en la figura 5-2. Ya que cualquier línea de comando con el símbolo # delante de esta automáticamente se convierte en un comentario.

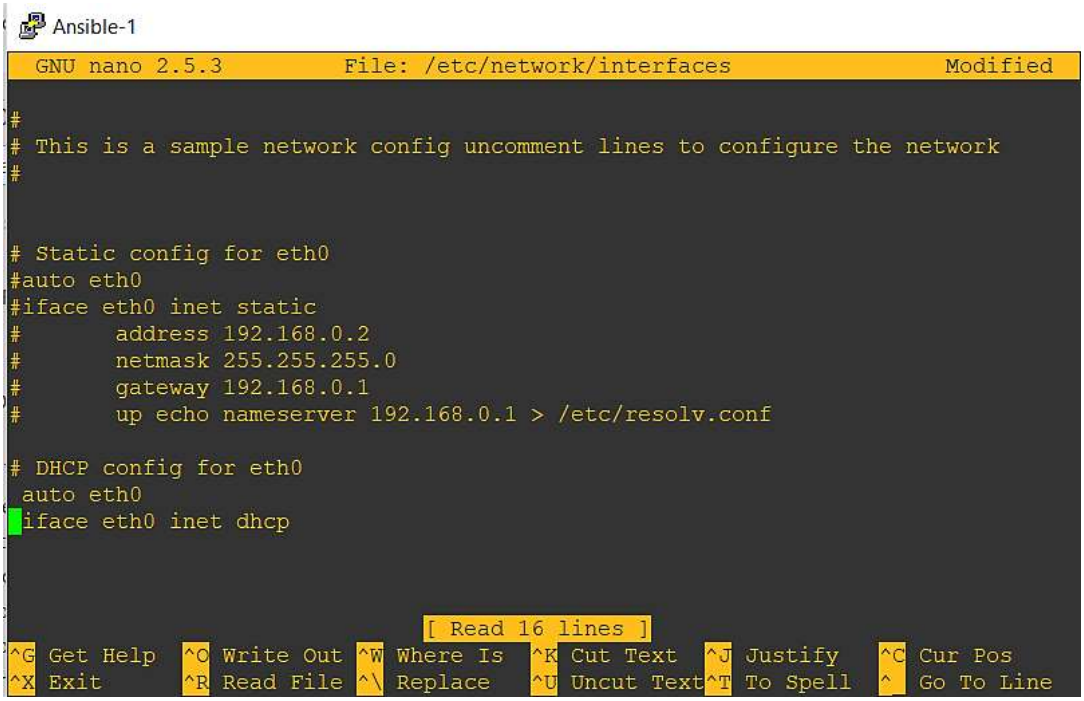


Figura 5-2: Configuración del direccionamiento IP de la maquina central
Fuente: YUNGA, Alex, 2018

En la figura 5-2 el símbolo ^ representa la tecla CTRL; así, si se presiona CTRL + X se sale de nano que permite guardar la configuración. Muchos de los comandos son interactivos, y este permite realizar atajos en la edición de scripts, en la tabla 10-2 se describe una lista de atajos.

Tabla 10-2: Lista de atajos de comandos básicos para el editor nano

COMANDOS	FUNCION
CTRL+G	Mostrar la pantalla de ayuda

CTRL+O	Guardar cambios
CTRL+X	Salir de nano. Al ejecutar este "atajo", nano le preguntará si desea o no guardar los cambios en su archivo, otra opción muy útil. Presione Y para guardar los cambios (Yes), N para descartarlos (No), o CTRL + C para cancelar la operación
CTRL+R	Inserta el contenido de otro archivo desde la posición actual, es un comando interactivo, por lo que se le solicitará colocar nombre y/o ruta del archivo cuyo contenido desea copiar
CTRL+W	Busca la cadena de caracteres y sitúa el cursor en la coincidencia
**CTRL+ **	Buscar y remplazar
CTRL+ /	Ir a la línea, columna
CTRL+Y	Ir a la página siguiente
CTRL+V	Ir a la página anterior
CTRL+K	Cortar el texto de la línea actual
CTRL+U	Pegar el texto desde la ubicación actual del cursor

Fuente: https://docs.bluehosting.cl/tutoriales/servidores/guia-practica-de-los-editores-de-texto-nano-y-vi-en-linux.html

Realizado por: YUNGA, Alex, 2018

Para que los cambios surjan efecto se apaga y se vuelve a encender la maquina central (Ansible). Al habilitar la función de DHCP para la interfaz eth0 a través de NAT se asignará una dirección IP automáticamente como se observa en la figura 6-2. Cuando en la máquina virtual se asigna una dirección IP igual a cualquier dispositivo del nodo secundario se vuelve a reiniciar la máquina para que no haya ningún problema al momento de realizar la ejecución de los playbooks.

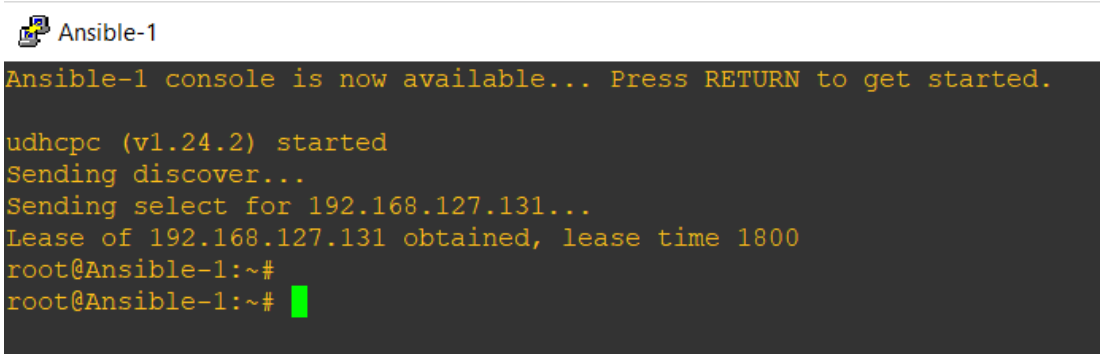


Figura 6-2: Dirección IP para la maquina central (Ansible)

Fuente: YUNGA, Alex, 2018

La segunda configuración es la de agregar los nombres de los dispositivos y sus direcciones IP correspondientes del nodo secundario para ello se utiliza el comando ***nano /etc/hosts*** que se muestra en la figura 7-2, se realiza este proceso porque permite modificar el fichero hosts y hace que la maquina central (Ansible) vea directamente las direcciones IP del nodo secundario.

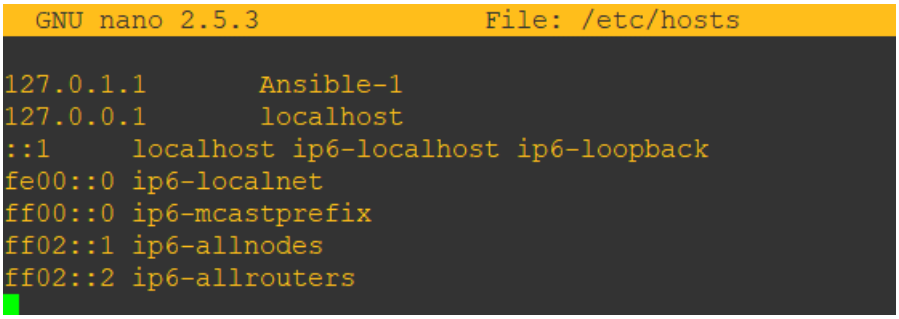


Figura 7-2: Fichero para añadir las direcciones IP de los dispositivos
Fuente: YUNGA, Alex, 2018

2.5.5 Configuración del nodo secundario

El nodo secundario es conocido como los clientes se caracteriza por recibir las configuraciones del nodo central. Los requisitos necesarios para las configuraciones iniciales en todas las topologías es que tenga una dirección IP, habilitar SSH, tener un usuario y una contraseña. Se puede agregar muchos usuarios a los dispositivos del nodo secundario, pero para fines prácticos solo se añade un usuario. Los comandos de configuración inicial varían dependiendo de qué plataforma de networking se utilice, las configuraciones iniciales para Arista se realizan como se muestra en la figura 8-2; para este ejemplo se utiliza el usuario que viene por defecto.

```
admin
ssh admin@leaf02


enable
configure

username admin privilege 15 secret arista
interface management 0
ip address 10.0.2.5/24
default management ssh
exit
write
```

Figura 8-2: Configuraciones iniciales para Arista
Fuente: YUNGA, Alex, 2018

La figura 9-2 muestra los comandos para realizar las configuraciones iniciales para Cisco.

```
enable
conf t
username alex privilege 15 secret cisco
line vty 0 4
login local
transport input all
ip domain-name gns3prueba.com
no ip domain-lookup
ip ssh version 2
crypto key generate rsa
1024

int fa0/0
ip add 192.168.122.45  255.255.255.0
no sh
```

Figura 9-2: Configuraciones iniciales para Cisco
Fuente: YUNGA, Alex, 2018

En la figura 10-2 se observa los comandos utilizados para la configuración inicial de Juniper.

```
cli
configure



set system root-authentication plain-text-password
set system host-name spine1
juniper123
juniper123
ssh-keygen -t rsa


set system login user alex class super-user
set system login user alex authentication plain-text-password
set system login user alex authentication load-key-file /root/.ssh/id_rsa.pub
set system services ssh
set system services ssh protocol-version v2
set system services netconf ssh
set interfaces em1 unit 0 family inet address 192.168.122.61/30
commit
```

Figura 10-2: Configuraciones iniciales para los dispositivos Juniper
Fuente: YUNGA, Alex, 2018

Como se va a utilizar el firewall de Cisco también se muestra la configuración inicial de este dispositivo en la figura 11-2.

```
conf t
interface Management0/0
 nameif management
 security-level 0
 ip address 192.168.122.41 255.255.255.0
 no shutdown
 exit

hostname asav1
domain-name gns3prueba.com
username alex password cisco privilege 15
aaa authentication ssh console LOCAL
aaa authorization exec LOCAL auto-enable
ssh version 2
ssh key-exchange group dh-group14-sha1
ssh 0 0 management
```

Figura 11-2: Configuración inicial para el firewall
Fuente: YUNGA, Alex, 2018

2.6 Implementación de las distintas topologías

2.6.1 Topología 1 – LAN, VLANS

En esta topología se emplea un switch capa 3 de Cisco que tiene un consumo de memoria RAM de 768 MB (figura 12-2) y tiene 4 adaptadores ethernet dando como resultado 16 interfaces.

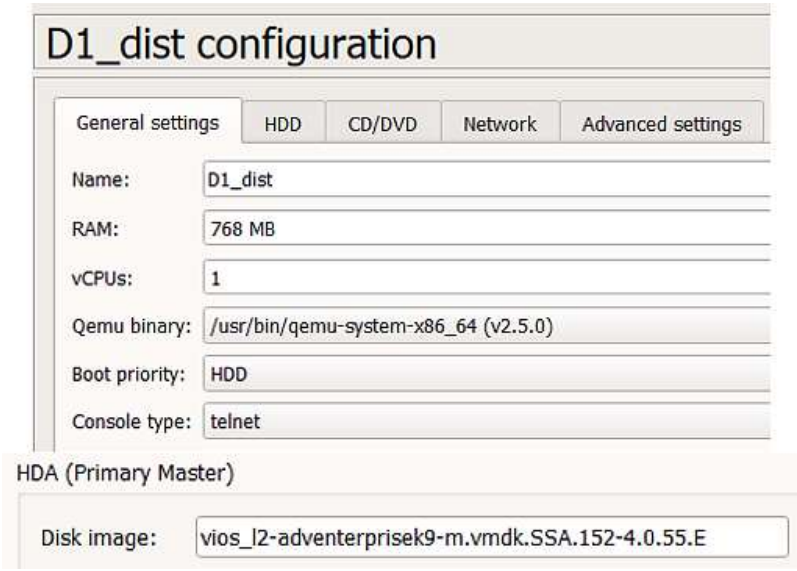


Figura 12-2: Consumo de memoria RAM - SW CISCO
Fuente: YUNGA, Alex, 2018

Se realiza el escenario de la figura 13-2 y se procede a configurar el usuario y su respectiva contraseña, se habilita SSH y se asigna una dirección IP en la interfaz Giga Ethernet 3/3 de los switches D1_dist, D2_dist, A1_acc y A2_acc. Al utilizar 4 switches se repetirá esta tarea el número de veces de los dispositivos a configurar.

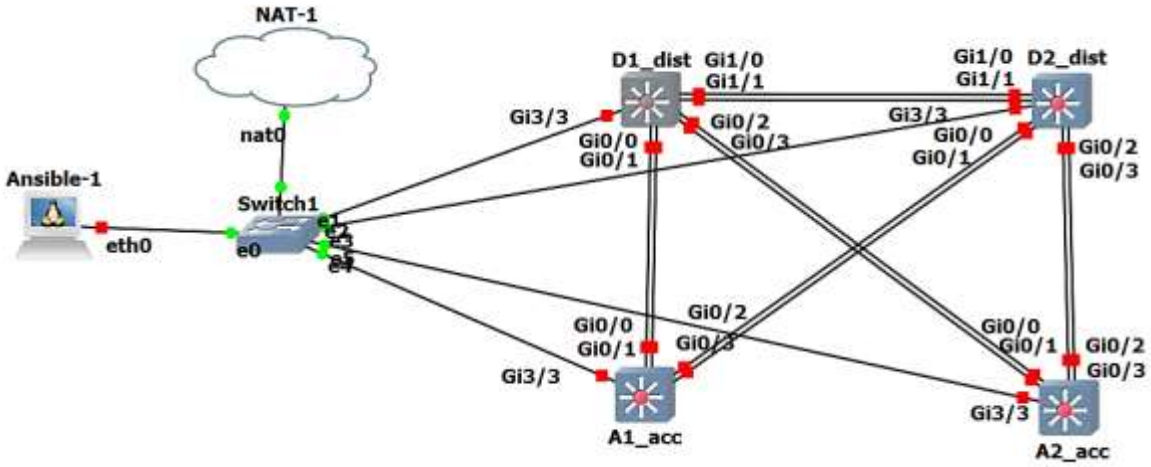


Figura 13-2: Escenario de VLANS
Fuente: YUNGA, Alex, 2018

En Ansible se crea el fichero **gns3hosts** que contiene el nombre y las direcciones IP de los switches y se ejecuta el comando ***cat gns3hosts >> /etc/hosts*** para anexarlos automáticamente como se observa en la figura 14-2, este comando debe ejecutarse cuando se inicie Ansible debido a que cuando se cierra la máquina virtual las direcciones IP y los hosts desaparecen.

Ansible-1

```
root@Ansible-1:~# ls
gns3hosts
root@Ansible-1:~# cat gns3hosts >> /etc/hosts
root@Ansible-1:~# cat /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
192.168.122.11 D1
192.168.122.12 D2
192.168.122.13 A1
192.168.122.14 A2
```

Figura 14-2: Agregación de los dispositivos del nodo secundario al nodo principal
Fuente: YUNGA, Alex, 2018

Desde Ansible se accede a todos los switches del nodo secundario por medio de SSH con el comando ***ssh alex@D1*** donde alex es el usuario y D1 es el nombre del dispositivo como se muestra en la figura 15-2. Al realizar SSH al dispositivo sale una respuesta de confirmación, se elige ***yes*** y se digita la respectiva contraseña, en el caso de que no se establezca SSH podría ser que el usuario o el nombre del switch este mal configurado como se muestra en la figura 21-2 donde R1 no es un host conocido para Ansible.

```
root@Ansible-1:~# ssh alex@R1
ssh: Could not resolve hostname r1: Name or service not known
root@Ansible-1:~# ssh alex@D1
The authenticity of host 'd1 (192.168.122.11)' can't be established.
RSA key fingerprint is SHA256:8fbTmjJU2Hn463Fp46Bmu2OOaKxREOjClK5LlrOa4W4.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'd1,192.168.122.11' (RSA) to the list of known hosts.
Password:

D1#
```

Figura 15-2: Acceso por SSH al switch D1
Fuente: YUNGA, Alex, 2018

Dentro del fichero ***known_hosts*** se almacena todas las llaves SSH (figura 16-2), a veces se desea borrar estas llaves y se hace mediante el siguiente comando ***rm /root/.ssh/known_hosts*** que elimina por completo el fichero ***known_hosts.***

```
root@Ansible-1:~# cat /root/.ssh/known_hosts
|1|7rEVsu/UK2USroLggx5fltx7Lus=|dujC9r31KtCu7wELm/1mNFzIUrE= ssh-rsa AAAAB3NzaC1y
n7080WE/6XjdlfqRKElUjpGHvhWWp6T28sLDUYWeUL9JvWUrmOL8VDflC3+I2YUcvuH8zWHY2w9Fc95Cv
fNHvbZmirWJ5y3BQInMAayvJ/t7xg0x1HW/4HujdYT7+hpdLQuw==
|1|OR2J54mZ6eT4XVa3hOpPSOWhDes=|7mJew3+CV0Egqm32mfkW6xZI04k= ssh-rsa AAAAB3NzaC1y
n7080WE/6XjdlfqRKElUjpGHvhWWp6T28sLDUYWeUL9JvWUrmOL8VDflC3+I2YUcvuH8zWHY2w9Fc95Cv
fNHvbZmirWJ5y3BQInMAayvJ/t7xg0x1HW/4HujdYT7+hpdLQuw==
root@Ansible-1:~# rm /root/.ssh/known_hosts
root@Ansible-1:~# cat /root/.ssh/known_hosts
cat: /root/.ssh/known_hosts: No such file or directory
```

Figura 16-2: Eliminación de las llaves SSH del fichero known_hosts
Fuente: YUNGA, Alex, 2018

Otra de las veces se necesita de un comando que elimine la llave SSH de un dispositivo en específico que es el siguiente ***rm ssh-keygen -f "/root/.ssh/known_hosts" -R D1*** donde D1 es el nombre del dispositivo, como se muestra en la figura 17-2.

```
root@Ansible-1:~# rm ssh-keygen -f "/root/.ssh/known_hosts" -R D1
root@Ansible-1:~# cat /root/.ssh/known_hosts
cat: /root/.ssh/known_hosts: No such file or directory
```

Figura 17-2: Eliminación de las llaves SSH del switch D1
Fuente: YUNGA, Alex, 2018

Se crea los directorios, playbooks y plantillas que se necesita para realizar la implementación en la topología 1 que se muestra en la figura 18-2. Dentro del directorio **groups_vars** y **host_vars** guarda las variables correspondientes a los hosts y en **templates** se guardan las plantillas.

```
root@Ansible-1:~# mkdir configs
root@Ansible-1:~# mkdir group_vars
root@Ansible-1:~# mkdir host_vars
root@Ansible-1:~# mkdir templates
root@Ansible-1:~# nano ansible.cfg
root@Ansible-1:~# nano devices
root@Ansible-1:~# ls
ansible.cfg  configs  devices  gns3hosts  group_vars  host_vars  templates
root@Ansible-1:~#
```

Figura 18-2: Ficheros y directorios de la topología 1
Fuente: YUNGA, Alex, 2018

El fichero ***ansible.cfg*** de la figura 19-2 guarda los parámetros de configuración para Ansible.

ansible.cfg

```
[defaults]

host_key_checking = false
inventory = devices
retry_files_enabled = false
display_skipped_hosts = false
deprecation_warnings = false
```

Figura 19-2: Configuración para Ansible
Fuente: YUNGA, Alex, 2018

La tabla 11-2 describe la función de cada parámetro de configuración. Todos estos parámetros se utilizan en cada una de las topologías.

Tabla 11-2: Descripción de los parámetros utilizados

PARÁMETRO	FUNCIÓN
host_key_checking	Si un host se reinstala y tiene una clave diferente en 'known_hosts', esto dará como resultado un mensaje de error hasta que se corrija. Si un host no está inicialmente en 'known_hosts', esto provocará que se solicite la confirmación de la clave, lo que da como resultado una experiencia interactiva si se usa Ansible

Inventory	Esta es la ubicación predeterminada del archivo de inventario, scripts o directorio que utilizará Ansible para determinar los hosts que tiene disponibles para realizar las tareas de configuración.
retry_files_enabled	Esto controla si un playbook de Ansible fallido debería crear un archivo .retry, la configuración predeterminada es true
display_skipped_hosts	Si se establece en false, Ansible no mostrará ningún estado para una tarea que se salta. El comportamiento predeterminado es mostrar las tareas omitidas.
deprecation_warnings	Las advertencias de depreciación indican el uso de funciones heredadas que están programadas para su eliminación en una versión futura de Ansible. Permite deshabilitar las advertencias a la salida de la ejecución de un playbook.

Fuente: https://docs.ansible.com/ansible/2.4/intro_configuration.html

Realizado por: YUNGA, Alex, 2018

En todas las topologías se guarda el inventario para Ansible en el archivo ***devices*** como se observa en la figura 20-2, que se caracteriza por guardar todos los dispositivos del nodo secundario que se vaya a configurar. Dentro de este archivo se puede clasificar los distintos dispositivos por grupos y dicho grupo se identifica mediante los [].

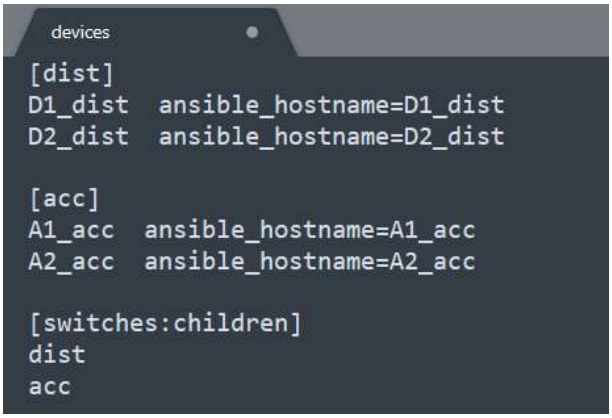


Figura 20-2: Inventario del nodo secundario para Ansible

Fuente: YUNGA, Alex, 2018

En el directorio de ***host_vars*** se crean los playbooks para cada dispositivo del nodo secundario como regla general estos deben llevar el mismo nombre que el dispositivo. Las configuraciones varían dependiendo de si es un switch de acceso (figura 21-2) o de distribución (figura 22-2).

```
root@Ansible-1:~/host_vars# ls
A1_acc.yml  A2_acc.yml  D1_dist.yml  D2_dist.yml
root@Ansible-1:~/host_vars# cat A1_acc.yml
---
vlan_01:
  accounting: { id: '30', subnet: 10.1.30.0/24, root: D1_dist }
  management: { id: '31',  subnet: 10.1.31.0/24, root: D1_dist }
  voice01: { id: '10', subnet: 10.1.10.0/24, root: D1_dist }

vlans: "{{ vlan_01|combine(vlan_02) }}"


access_ports:
  - ports:
      - 'Gi1/0 - 1'
      - 'Gi2/0'
    data: accounting
    voice: voice01
  - ports:
      - 'Gi1/2 - 3'
    data: management
    voice: voice01

trunk_ports:
  - { group: 1, ports: ['Gi0/0 - 1'], description: -> D1_dist }
  - { group: 2, ports: ['Gi0/2 - 3'], description: -> D2_dist }
```

Figura 21-2: playbook para el switch A1_acc
Fuente: YUNGA, Alex, 2018

```
root@Ansible-1:~/host_vars# cat D1_dist.yml
---

trunk_ports:
  - { group: 1, ports: ['Gi0/0', 'Gi0/1'], description: -> A1_acc }
  - { group: 2, ports: ['Gi0/2', 'Gi0/3'], description: -> A2_acc }

svi_id: 1

interfaces:
  Po40: { ip: 172.16.100.1/30, routed: true, ports: ['Gi1/0', 'Gi1/1'], description: ptp link to D2_dist }
  Gi1/2: { ip: 172.168.1.20/24, routed: true, description: -> wan-router1 }

loopbacks:
  lo0: { ip: 10.250.0.20/32 }
```

Figura 22-2: Playbook para un switch D1_dist
Fuente: YUNGA, Alex, 2018

Tanto para un switch de acceso y de distribución se crea la configuración necesaria. Cada plantilla cumple una tarea en específico y están relacionadas directamente con los playbooks del directorio ***host_vars***. En la figura 23-2 se observa las plantillas utilizadas para esta topología.

```
root@Ansible-1:~# cd templates/
root@Ansible-1:~/templates# ls
root@Ansible-1:~/templates# nano access-ports.j2
root@Ansible-1:~/templates# nano default-interface.j2
root@Ansible-1:~/templates# nano hsrp.j2
root@Ansible-1:~/templates# nano interface.j2
root@Ansible-1:~/templates# nano stp.j2
root@Ansible-1:~/templates# nano svi.j2
root@Ansible-1:~/templates# nano trunk-ports.j2
root@Ansible-1:~/templates# nano vlan.j2
```

Figura 23-2: Plantillas para la Topología 1
Fuente: YUNGA, Alex, 2018

A continuación, se muestra la configuración de las plantillas más relevantes de la topología 1. La figura 24-2 muestra la configuración para los puertos de acceso y la figura 25-2 se observa la configuración para las VLANS.

```
root@Ansible-1:~/templates# cat access-ports.j2
{% for item in access_ports -%}

{% for access in item.ports %}
{% if "-" in access|string %}
interface range {{ access }}
{% else %}
interface {{ access }}
{% endif %}
  switchport mode access
  switchport access vlan {{ vlans[item.data].id }}
  {% if item.voice is defined -%}
  switchport voice vlan {{ vlans[item.voice].id }}
  {% endif -%}
  {% if item.group is defined -%}
  channel-group {{ item.group }} mode on
  {% endif -%}
  no shutdown
{% endfor %}

{% endfor %}
```

Figura 24-2: Platilla para configurar los puertos de acceso
Fuente: YUNGA, Alex, 2018

```
root@Ansible-1:~/templates# cat vlan.j2
{% for key,value in vlans|dictsort %}
vlan {{ value.id }}
  name {{ key }}
{% endfor %}
```

Figura 25-2: Plantilla para configurar las vlans
Fuente: YUNGA, Alex, 2018

La figura 26-2 muestra la configuración para aplicar los cambios en las interfaces de los switches y en la figura 27-2 se observa la configuración para los puertos troncales.

```
root@Ansible-1:~/templates# cat interface.j2
{% for key,value in interfaces|dictsort + loopbacks|dictsort %}
interface {{ key }}
  {% if value.routed is defined -%}
  no switchport
  {% endif -%}
  {% if value.ip != 'dhcp' -%}
  ip address {{ value.ip|ipaddr('address') }} {{ value.ip|ipaddr('netmask') }}
  {% else -%}
  ip address dhcp
  {% endif -%}
  no shutdown
{% endfor %}

{% for key,value in interfaces|dictsort if value.ports is defined %}
{% for ports in value.ports %}
interface {{ ports }}
  no ip address
  {% if value.routed is defined -%}
  no switchport
  {% endif -%}
  no shutdown
  channel-group  {{ key|replace('Po', '') }} mode on
{% endfor %}
{% endfor %}
```

Figura 26-2: Platilla para configurar interfaces en los switches
Fuente: YUNGA, Alex, 2018

```
root@Ansible-1:~/templates# cat trunk-ports.j2
{% for item in trunk_ports -%}

{% for trunk in item.ports %}
{% if "-" in trunk|string %}
interface range {{ trunk }}
{% else %}
interface {{ trunk }}
{% endif %}
  {% if item.description is defined -%}
  description {{ item.description }}
  {% endif -%}
  switchport trunk encapsulation dot1q
  switchport mode trunk
  switchport trunk native vlan {{ vlan_02['native'].id }}
  switchport trunk allowed vlan {{ allowed_vlans|default('all') }}
  switchport trunk allowed vlan remove 1
  no shutdown
  {% if item.group is defined -%}
  channel-group {{ item.group }} mode on
  {% endif -%}
{% endfor %}

{% endfor %}
```

Figura 27-2: Plantilla para configurar los puertos troncales
Fuente: YUNGA, Alex, 2018

La ejecución de los playbooks se realiza mediante el comando ***ansible-playbook distribution.yml*** donde distribution.yml es el playbook que se va a ejecutar, este playbook se encarga de aplicar todas las tareas de configuración para los switches de distribución como se observa en la figura 28-2 donde:

hosts asigna el nombre de los dispositivos a configurar.

name indica el nombre de la tarea.

ios_config indica el módulo de red a utilizar en este caso el de CISCO,

src indica la ruta donde se almacena la plantilla.

tasks son las tareas que se van a ejecutar.

```
root@Ansible-1:~# cat distribution.yml
---
- name: distribution switch configs
  hosts: dist
  gather_facts: no
  connection: local

  tasks:

# lan switching
    - name: vlans config
      ios_config:
        src: templates/vlan.j2
      tags: vlan

    - name: spanning-tree config
      ios_config:
        src: stp.j2
      tags: stp

    - name: default interface config
      ios_config:
        src: default-interface.j2
      tags: default-interface

    - name: trunk port config
      ios_config:
        src: trunk-ports.j2
      tags: trunk-ports

# interfaces
    - name: switch vlan interface config
      ios_config:
        src: templates/svi.j2
      tags: svi
```

Figura 28-2: Playbook para los switches de distribución
Fuente: YUNGA, Alex, 2018

2.6.2 Topología 2 – BGP

Se utiliza un router CISCO que consume 256 MB de RAM (figura 29-2) con 4 adaptadores de red que da 16 interfaces ethernet.

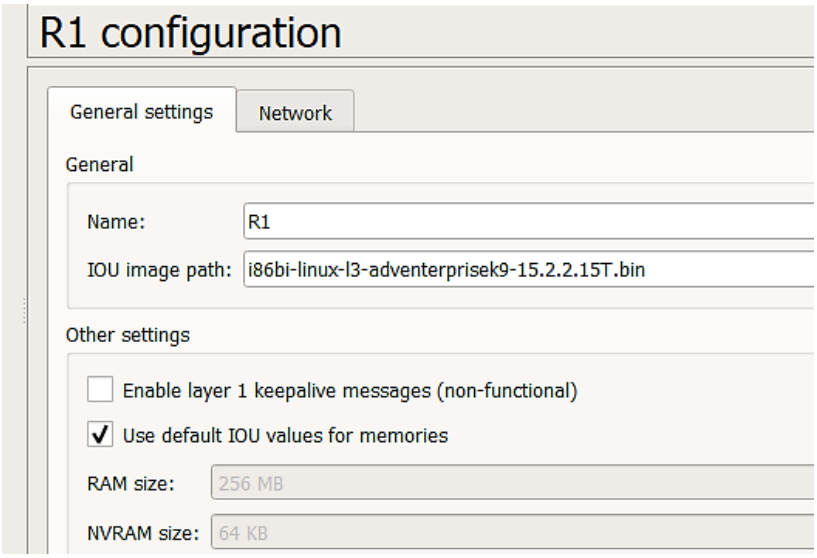


Figura 29-2: Consumo de memoria RAM del router CISCO de la topología 2
Fuente: YUNGA, Alex, 2018

Se realiza la configuración inicial de los routers R1, R2, R3 y R4 (SSH, dirección IP, usuario y contraseña). Para esta topología se utiliza 4 routers CISCO como se muestra en la figura 30-2.

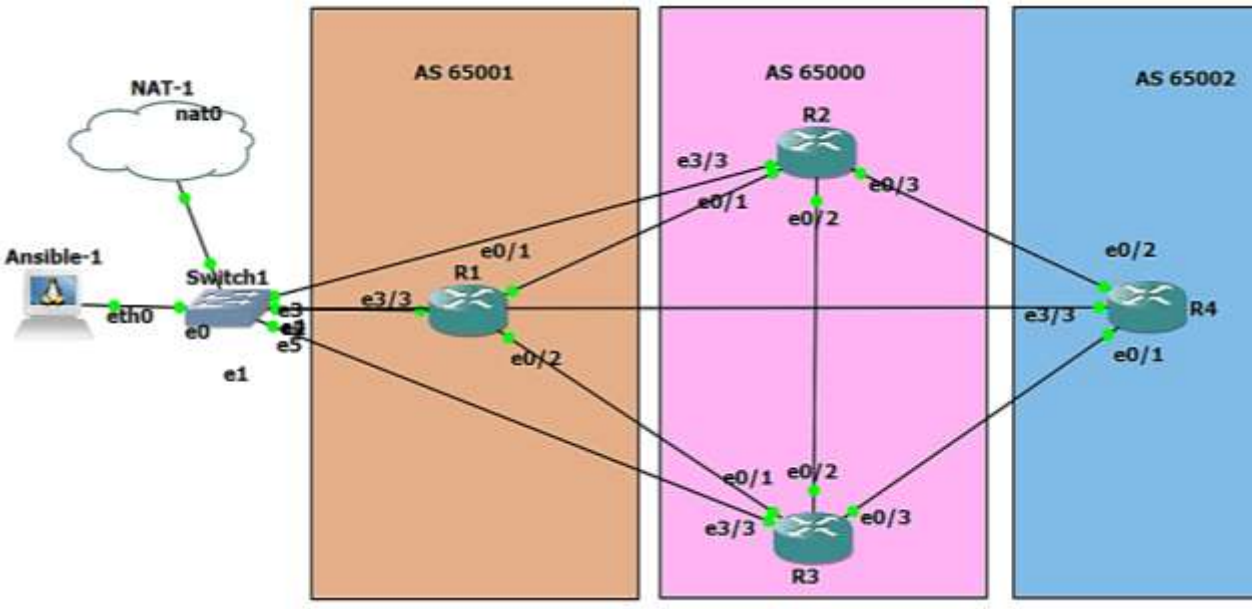


Figura 30-2: Escenario de BGP
Fuente: YUNGA, Alex, 2018

Desde el nodo principal o nodo central se accede a través de SSH a los routers del nodo secundario como se observa en la figura 31-2.

```
root@Ansible-1:~# ssh alex@R1
The authenticity of host 'r1 (192.168.122.21)' can't be established.
RSA key fingerprint is SHA256:NL/E1hABOl8PAKmpkCdZ/BhbNi37ewZyNHVWeYlZSJs.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'r1,192.168.122.21' (RSA) to the list of known hosts.
Password:

R1#exit
Connection to r1 closed by remote host.
Connection to r1 closed.
root@Ansible-1:~#
root@Ansible-1:~# ssh alex@R2
The authenticity of host 'r2 (192.168.122.22)' can't be established.
RSA key fingerprint is SHA256:7gWqvlQUdcymJ8H+m4Yu5HXjwqZkx/7AOOrLVFJDCa0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'r2,192.168.122.22' (RSA) to the list of known hosts.
Password:

R2#exit
Connection to r2 closed by remote host.
Connection to r2 closed.
```

Figura 31-2: Acceso por SSH a los routers R1 y R2
Fuente: YUNGA, Alex, 2018

Se crea todos los directorios, ficheros, playbooks y las plantillas necesarias en el nodo central para la topología 2 (figura 32-2).

```
root@Ansible-1:~# mkdir group_vars
root@Ansible-1:~# mkdir host_vars
root@Ansible-1:~# mkdir roles
root@Ansible-1:~# ls
ansible.cfg  devices1  gns3hosts1  group_vars  host_vars  roles
root@Ansible-1:~# nano check_icmp.yml
root@Ansible-1:~# nano site.yml
root@Ansible-1:~# cd group_vars/
root@Ansible-1:~/group_vars# ls
root@Ansible-1:~/group_vars# nano all.yml
root@Ansible-1:~/group_vars# cd ..
root@Ansible-1:~# cd host_vars/
root@Ansible-1:~/host_vars# ls
root@Ansible-1:~/host_vars# nano R1.yml
root@Ansible-1:~/host_vars# nano R2.yml
root@Ansible-1:~/host_vars# nano R3.yml
root@Ansible-1:~/host_vars# nano R4.yml
root@Ansible-1:~/host_vars# cd ..
root@Ansible-1:~# cd roles/
root@Ansible-1:~/roles# mkdir hostname
root@Ansible-1:~/roles# mkdir interfaces
root@Ansible-1:~/roles# mkdir routing
root@Ansible-1:~/roles# mkdir snmp
```

Figura 32-2: Directorios y playbooks para la topología 2
Fuente: YUNGA, Alex, 2018

En la figura 33-2 se observa la configuración para el router R1 donde se muestra la configuración de sus interfaces y el protocolo de enrutamiento.

```
root@Ansible-1:~/host_vars# cat R1.yml
---

hostname: R1
domain_name: gns3prueba.com

loopback:
  address: 10.255.0.1
  mask: 255.255.255.255

interfaces:
  0/1:
    alias: connection R2
    address: 10.0.255.1
    mask: 255.255.255.252

  0/2:
    alias: connection R3
    address: 10.0.255.5
    mask: 255.255.255.252

bgp:
  asn: 65001
  neighbor:
    - {address: 10.0.255.2, remote_as: 65000}
    - {address: 10.0.255.6, remote_as: 65000}
  networks:
    - {network: 10.0.255.0, mask: 255.255.255.252}
    - {network: 10.0.255.4, mask: 255.255.255.252}
    - {network: 10.255.0.1, mask: 255.255.255.255}
```

Figura 33-2: Playbook para el router R1
Fuente: YUNGA, Alex, 2018

En esta topología se hace el uso de roles, mediante la creación de los playbooks y plantillas en cada uno de sus respectivos directorios. La figura 34-2 muestra la ruta de donde se almacena la plantilla interfaces.j2 e indica la configuración para realizar los cambios en las interfaces.

La figura 35-2 muestra la ruta de la platilla routing.j2 y mediante los condicionales y bucles indica de como aplicar el enrutamiento en la topología 2.

```
root@Ansible-1:~/roles# cd interfaces/
root@Ansible-1:~/roles/interfaces# ls
tasks  templates
root@Ansible-1:~/roles/interfaces# cd templates/
root@Ansible-1:~/roles/interfaces/templates# ls
interfaces.j2
root@Ansible-1:~/roles/interfaces/templates# cat interfaces.j2
{% if loopback is defined %}
interface Loopback0
 ip address {{ loopback.address }} {{ loopback.mask }}
{% endif %}

{% if interfaces is defined %}
{% for port, value in interfaces.items() %}
interface Ethernet{{ port }}
{% if 'alias' in value %}
 description {{ value.alias }}
{% endif %}
 ip address {{ value.address }} {{ value.mask }}
{% endfor %}
{% endif %}
```

Figura 34-2: Plantilla para configurar las interfaces de la topología 2
Fuente: YUNGA, Alex, 2018

```
root@Ansible-1:~/roles# cd routing/
root@Ansible-1:~/roles/routing# ls
tasks  templates
root@Ansible-1:~/roles/routing# cd templates/
root@Ansible-1:~/roles/routing/templates# cat routing.j2
{% if bgp is defined %}
router bgp {{ bgp.asn }}
 bgp router-id {{ loopback.address }}
 bgp log-neighbor-changes
{% for item in bgp.neighbor %}
 neighbor {{ item.address }} remote-as {{ item.remote_as }}
{% endfor %}
 !
 address-family ipv4
{% for item in bgp.networks %}
  network {{ item.network }} mask {{ item.mask }}
{% endfor %}
{% for item in bgp.neighbor %}
  neighbor {{ item.address }} activate
{% endfor %}
{% if bgp.maxpath is defined %}
  maximum-paths {{ bgp.maxpath }}
{% endif %}
 exit-address-family
{% endif %}
```

Figura 35-2: Plantilla para configurar el enrutamiento en la topología 2
Fuente: YUNGA, Alex, 2018

Se crea un playbook principal que abarque todos los demás playbooks y plantillas, esto lo hace el playbook ***site,*** que ejecuta toda la configuración en los routers para la topología 2 (figura 36-2). Esta configuración es aplicada por medio de roles que se caracteriza por tener un directorio de tasks (tareas) que contiene el playbook en formato YAML y el directorio templates (plantillas) que tiene el archivo JINJA, los roles se ejecutan en el orden establecido.

```
root@Ansible-1:~# cat site.yml
---
- name: Ejecucion completa1
  hosts: routers
  connection: local
  gather_facts: 'no'

  roles:
    - hostname
    - interfaces
    - routing
    - snmp
```

Figura 36-2: Playbook site para la topología 2
Fuente: YUNGA, Alex, 2018

Se ejecuta el playbook site como se muestra en la figura 37-2 se ejecuta los roles de **hostname** e ***interfaces***

```
root@Ansible-1:~# ansible-playbook site.yml

PLAY [Ejecucion completa1] *****************************************************

TASK [interfaces : write interfaces config] ************************************
changed: [R1]
changed: [R2]
changed: [R3]
changed: [R4]

TASK [interfaces : enable interfaces] ******************************************
changed: [R4] => (item={u'0/2', {u'alias': u'connection R2', u'mask': u'255.255.255.252', u'address': u'10.0.2
54.1'}})
changed: [R1] => (item={u'0/2', {u'alias': u'connection R3', u'mask': u'255.255.255.252', u'address': u'10.0.2
55.5'}})
changed: [R2] => (item={u'0/2', {u'alias': u'connection R3', u'mask': u'255.255.255.252', u'address': u'10.0.2
53.1'}})
changed: [R3] => (item={u'0/2', {u'alias': u'connection R2', u'mask': u'255.255.255.252', u'address': u'10.0.2
53.2'}})
changed: [R4] => (item={u'0/1', {u'alias': u'connection R3', u'mask': u'255.255.255.252', u'address': u'10.0.2
54.5'}})
changed: [R1] => (item={u'0/1', {u'alias': u'connection R2', u'mask': u'255.255.255.252', u'address': u'10.0.2
55.1'}})
changed: [R2] => (item={u'0/3', {u'alias': u'connection R4', u'mask': u'255.255.255.252', u'address': u'10.0.2
54.2'}})
changed: [R3] => (item={u'0/3', {u'alias': u'connection R4', u'mask': u'255.255.255.252', u'address': u'10.0.2
54.6'}})
changed: [R2] => (item={u'0/1', {u'alias': u'connection R1', u'mask': u'255.255.255.252', u'address': u'10.0.2
55.2'}})
changed: [R3] => (item={u'0/1', {u'alias': u'conection R1', u'mask': u'255.255.255.252', u'address': u'10.0.25
5.6'}})
```

Figura 37-2: Ejecución del playbook site (Parte I)
Fuente: YUNGA, Alex, 2018

En la figura 38-2 se ejecutan los roles de ***routing*** y ***snmp***. Cuando todo está bien configurado da como resultado en PLAY RECAP todo en verde y si hay algún error se termina la ejecución del playbook y las letras son de color rojo.

```
TASK [routing : write routing config] ***************************************
changed: [R1]
changed: [R2]
changed: [R4]
changed: [R3]

TASK [snmp : deploys snmp configuration] ************************************
changed: [R2]
changed: [R3]
changed: [R1]
changed: [R4]

PLAY RECAP ******************************************************************
R1                         : ok=4    changed=4    unreachable=0    failed=0
R2                         : ok=4    changed=4    unreachable=0    failed=0
R3                         : ok=4    changed=4    unreachable=0    failed=0
R4                         : ok=4    changed=4    unreachable=0    failed=0
```

Figura 38-2: Ejecución del playbook site (Parte II)
Fuente: YUNGA, Alex, 2018

2.6.3 Topología 3 – MPLS

Se utiliza un router CISCO con un consumo de memoria RAM de 512 MB (figura 39-2), para esta topología se configura 10 routers; 4 routers PE (borde del proveedor), 4 routers CE (borde del cliente) y 2 routers P (proveedor) como se muestra en la figura 40-2.

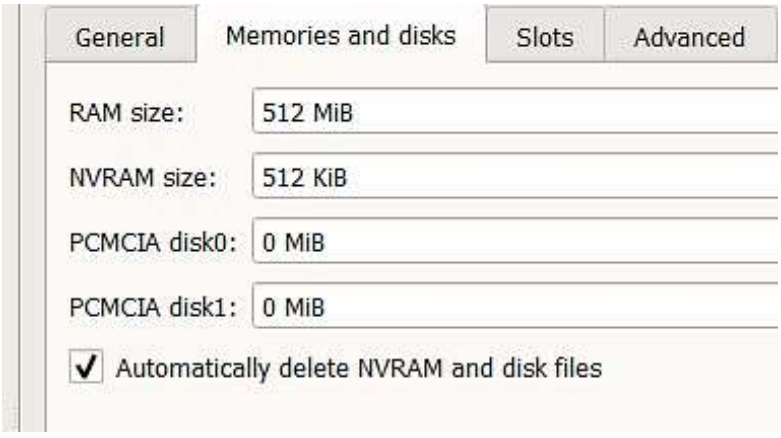


Figura 39-2: Consumo de memoria RAM del router para la topología 3
Fuente: YUNGA, Alex, 2018

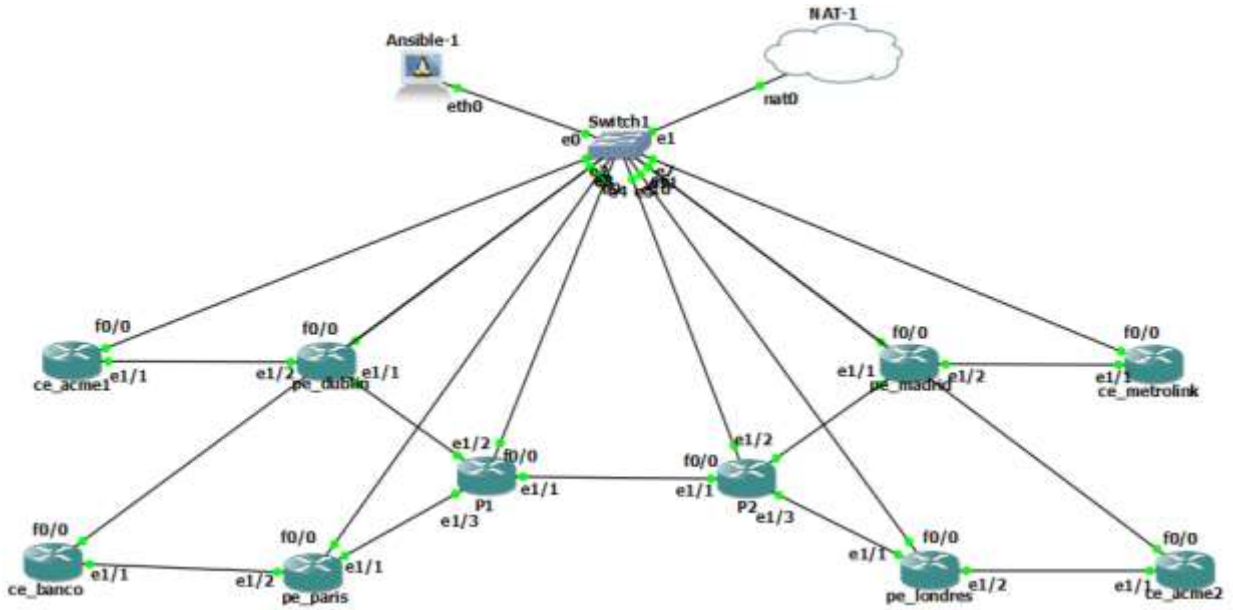


Figura 40-2: Escenario de MPLS
Fuente: YUNGA, Alex, 2018

Se accede a todos los routers desde Ansible por medio de SSH como se observa en la figura 41-2.

```
root@Ansible-1:~# ssh alex@ce_acme1
The authenticity of host 'ce_acme1 (192.168.122.31)' can't be established.
RSA key fingerprint is SHA256:3yGqzsY+JQSqWV2G7g0BdoN2dJBk1B6/PXm1aTVq3Kg.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ce_acme1,192.168.122.31' (RSA) to the list of known hosts.
Password:
```

Figura 41-2: Acceso por SSH a los router CE
Fuente: YUNGA, Alex, 2018

Se crean todos los directorios y ficheros necesarios para la implementación de la topología 3. Dentro del directorio host_vars se crean los playbooks para cada router ya que allí se guarda su respectiva configuración. Para los routers CE (borde del cliente) se tiene la configuración de una interfaz y del protocolo BGP, para los routers P (proveedor) se posee la configuración de interfaces, BGP, OSPF y VPN y por ultimo los routers PE (borde del proveedor) se realiza la configuración de los clientes, interfaces, BGP y OSPF (**ANEXO G**).

Se crea todos los directorios, ficheros, playbooks y plantillas necesarias para la topología 3 dentro del directorio roles como se muestra en la figura 42-2.

```
root@Ansible-1:~/roles# cd perouting/
root@Ansible-1:~/roles/perouting# mkdir tasks
root@Ansible-1:~/roles/perouting# mkdir templates
root@Ansible-1:~/roles/perouting# cd tasks/
root@Ansible-1:~/roles/perouting/tasks# nano main.yml
root@Ansible-1:~/roles/perouting/tasks# cd ..
root@Ansible-1:~/roles/perouting# cd templates/
root@Ansible-1:~/roles/perouting/templates# nano perouting.j2
root@Ansible-1:~/roles/perouting/templates# cd ..
root@Ansible-1:~/roles/perouting# ls
tasks  templates
root@Ansible-1:~/roles/perouting# cd ..
root@Ansible-1:~/roles# ls
cerouting  hostname  interfaces  perouting  snmp  vrf
root@Ansible-1:~/roles# cd snmp/
root@Ansible-1:~/roles/snmp# ls
root@Ansible-1:~/roles/snmp# mkdir tasks
root@Ansible-1:~/roles/snmp# mkdir templates
root@Ansible-1:~/roles/snmp# cd tasks/
root@Ansible-1:~/roles/snmp/tasks# nano main.yml
```

Figura 42-2: Creación de playbooks y platillas para el directorio roles
Fuente: YUNGA, Alex, 2018

A continuación, se muestra las plantillas más relevantes en la figura 43-2 se observa la configuración para aplicar la VRFs en los routers CE.

```
{% if customers is defined %}
{% for customervrf, value in customers.items() %}
ip vrf {{ customervrf }}
{% if 'rd' in value %}
 rd {{ value.rd }}
{% endif %}
{% if 'rt' in value %}
 route-target export {{ value.rt }}
{% endif %}
{% if 'rt' in value %}
 route-target import {{ value.rt }}
{# enable cef globally #}
ip cef
{% endif %}
{% endfor %}
{% endif %}
```

Figura 43-2: Configuración de las VRFs en los routers CE
Fuente: YUNGA, Alex, 2018

En la figura 44-2 se muestra la plantilla para configurar las interfaces de cada uno de los routers.

```
{% if peloopback is defined %}
interface Loopback0
 ip address {{ peloopback.address }} {{ peloopback.mask }}
{% endif %}
{% if celoopback is defined %}
interface Loopback0
 ip address {{ celoopback.address }} {{ celoopback.mask }}
{% endif %}
{% if interfaces is defined %}
{% for port, value in interfaces.items() %}
interface Ethernet{{ port }}
{% if 'vlan' in value %}
 encapsulation dot1Q {{ value.vlan }}
{% endif %}
{% if 'intvrf' in value %}
 ip vrf forwarding {{ value.intvrf }}
{% endif %}
{% if 'alias' in value %}
 description {{ value.alias }}
{% endif %}
{% if 'address', 'mask' in value %}
 ip address {{ value.address }} {{ value.mask }}
{% endif %}
{% endfor %}
{% endif %}
```

Figura 44-2: Plantilla para configurar las interfaces de la topología 3
Fuente: YUNGA, Alex, 2018

Se ejecutan los playbooks principales, para esta topología se tiene un playbook **deploy_pe** (figura 45-2) que aplica todas las configuraciones a los routers P y PE del escenario de MPLS. La ejecución de este playbook se realiza en dos partes que son mostradas en las figuras 46-2 y 47-2 respectivamente.

deploy_pe.yml

```
---

- hosts: pe

  connection: local
  gather_facts: 'no'

  roles:
    - hostname
    - vrf
    - interfaces
    - perouting
```

Figura 45-2: Configuración del playbook deploy_pe
Fuente: YUNGA, Alex, 2018

```
PLAY [pe] ***********************************************************************

TASK [hostname : set dns and hostname] ******************************************
ok: [pe_dublin]
ok: [pe_madrid]
ok: [pe_paris]
ok: [P1]
ok: [P2]
ok: [pe_londres]

TASK [vrf : write vrf config] ***************************************************
ok: [P1]
ok: [P2]
changed: [pe_madrid]
changed: [pe_paris]
changed: [pe_dublin]
changed: [pe_londres]

TASK [interfaces : write interfaces config] *************************************
changed: [pe_dublin]
changed: [pe_madrid]
changed: [pe_paris]
changed: [P2]
changed: [P1]
changed: [pe_londres]

TASK [interfaces : no shut physical interfaces] *********************************
changed: [pe_dublin]
changed: [P1]
changed: [P2]
changed: [pe_madrid]
changed: [pe_paris]
changed: [pe_londres]
```

Figura 46-2: Ejecución del playbook deploy_pe (Parte I)
Fuente: YUNGA, Alex, 2018

```
TASK [interfaces : no shut physical interfaces] *********************************
changed: [pe_paris]
changed: [P1]
changed: [pe_dublin]
changed: [pe_madrid]
changed: [P2]
changed: [pe_londres]

TASK [perouting : write routing config] *****************************************
changed: [pe_dublin]
changed: [pe_paris]
changed: [pe_madrid]
changed: [P1]
changed: [P2]
changed: [pe_londres]

PLAY RECAP **********************************************************************
P1                         : ok=7    changed=5    unreachable=0    failed=0
P2                         : ok=7    changed=5    unreachable=0    failed=0
pe_dublin                  : ok=7    changed=6    unreachable=0    failed=0
pe_londres                 : ok=7    changed=6    unreachable=0    failed=0
pe_madrid                  : ok=7    changed=6    unreachable=0    failed=0
pe_paris                   : ok=7    changed=6    unreachable=0    failed=0
```

Figura 47-2: Ejecución del playbook deploy_pe (Parte II)
Fuente: YUNGA, Alex, 2018

Se aplica las configuraciones a todos los routers CE mediante los roles de la figura 48-2 y se ejecuta en orden las siguientes tareas: hostname, snmp, interfaces y cerouting.

Figura 48-2: Configuración del playbook deploy_ce
Fuente: YUNGA, Alex, 2018

2.6.4 Topología 4 – Firewall

Para esta topología se utiliza dos routers y un firewall de CISCO que utiliza un consumo de memoria RAM de 2048 MB (figura 49-2) y tiene 8 interfaces Giga Ethernet, en la figura 50-2 se muestra el escenario para esta topología.

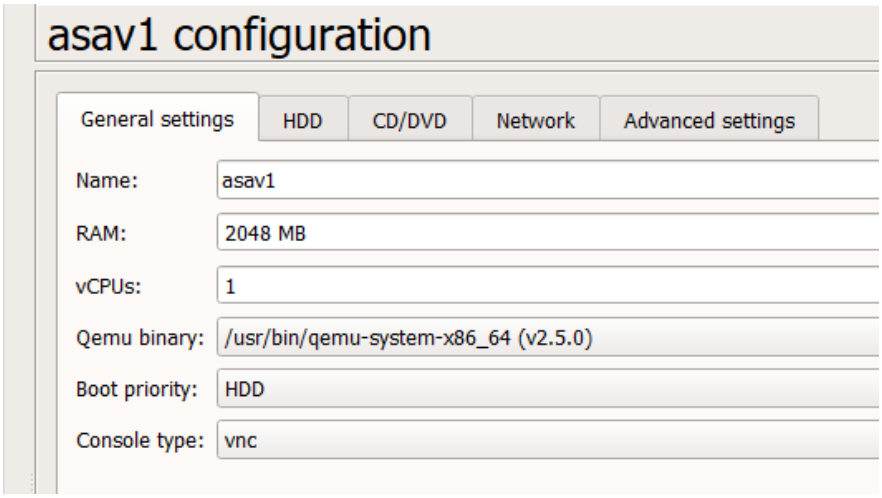


Figura 49-2: Consumo de memoria RAM de un firewall CISCO
Fuente: YUNGA, Alex, 2018

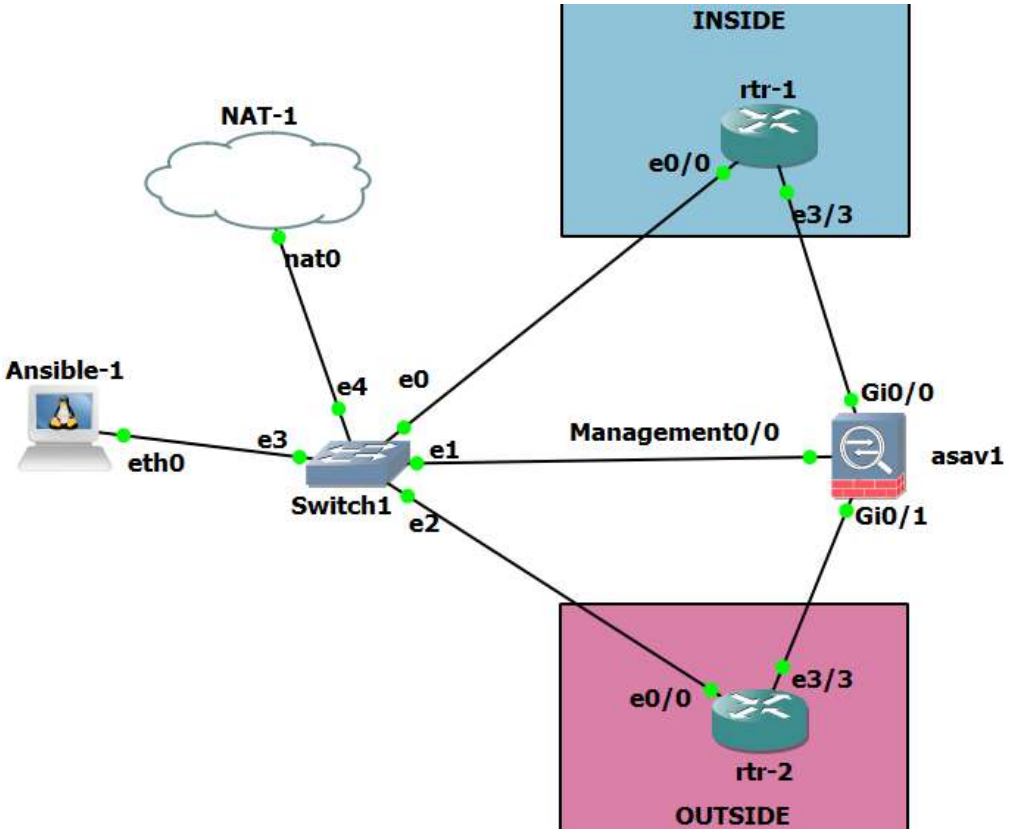


Figura 50-2: Escenario para la topología 4
Fuente: YUNGA, Alex, 2018

Se añade los dispositivos del nodo secundario a los hosts conocidos de Ansible mediante el comando ***cat gns3hosts >> /etc/hosts*** como se observa en la figura 51-2.

```
root@Ansible-1:~# cat gns3hosts
192.168.122.41 asav1
192.168.122.42 rtr-1
192.168.122.43 rtr-2

root@Ansible-1:~# cat gns3hosts >> /etc/hosts
root@Ansible-1:~# cat /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
192.168.122.41 asav1
192.168.122.42 rtr-1
192.168.122.43 rtr-2
```

Figura 51-2: Hosts de Ansible para la topología 4
Fuente: YUNGA, Alex, 2018

En la figura 52-2 se observa la creación de todos los ficheros, directorios, playbooks y plantillas necesarias para esta topología y también se muestra la configuración del playbook para el firewall.

```
root@Ansible-1:~# ls
ansible.cfg          config_firewall.yml  devices    group_vars  roles
asav1_check_icmp.yml  config_routers.yml   gns3hosts  host_vars
root@Ansible-1:~# cd host_vars/
root@Ansible-1:~/host_vars# cat asav1.yml
---

hostname: asav1
domain_name: gns3prueba.com

interfaces:
  0/0:
    alias: connection rtr-1 inside
    nameif: inside
    security_level: 100
    address: 10.0.255.1
    mask: 255.255.255.0

  0/1:
    alias: connection rtr-2 outside
    nameif: outside
    security_level: 0
    address: 217.100.100.1
    mask: 255.255.255.0

routes:
  - route outside 0.0.0.0 0.0.0.0 217.100.100.254 1
```

Figura 52-2: Ficheros, Directorios y Playbooks de la topología 4
Fuente: YUNGA, Alex, 2018

La configuración del firewall tiene diferentes objetos, listas de acceso y políticas de servicio que están dentro del directorio ***group_vars*** como se observa en la figura 53-2 aquí también se muestra la plantilla para las listas de acceso.

```
root@Ansible-1:~/group_vars# ls
access-lists.yml  nat.yml           objects.yml
all.yml           object-groups.yml  policy-framework.yml
root@Ansible-1:~/group_vars# cat access-lists.yml
---

override_acl: true

access_lists:
  - name: acl_inside
    interface: inside
    lines:
      - permit icmp object inside-subnet1 any
      - permit tcp object-group inside-server any eq www
      - deny ip any any

  - name: acl_outside
    interface: outside
    lines:
      - permit icmp any object-group inside-server
      - permit tcp any object-group inside-server eq www
      - deny ip any any
```

Figura 53-2: Playbooks del directorio group_vars para la topología 4
Fuente: YUNGA, Alex, 2018

Se ejecuta el playbook de la figura 54-2 para aplicar la configuración en los routers y en el firewall del nodo secundario.

```
TASK [policy-framework : write policy-map global_policy] ***********************
changed: [asav1] => (item={u'class': u'inspection_default'}, u'inspect icmp'})
changed: [asav1] => (item={u'class': u'inspection_default'}, u'inspect icmp error'})
changed: [asav1] => (item={u'class': u'inspection_default'}, u'inspect dns dns_map'})
changed: [asav1] => (item={u'class': u'inspection_default'}, u'inspect ftp'})
changed: [asav1] => (item={u'class': u'highweb-tcp-traffic'}, u'set connection embryonic-conn-max 4000 per-cl
ient-max 1000 per-client-embryonic-max 100'})
changed: [asav1] => (item={u'class': u'highweb-tcp-traffic'}, u'set connection timeout embryonic 0:00:10 idle
 1:00:00'})
changed: [asav1] => (item={u'class': u'medweb-tcp-traffic'}, u'set connection embryonic-conn-max 500 per-clie
nt-max 1000 per-client-embryonic-max 100'})
changed: [asav1] => (item={u'class': u'medweb-tcp-traffic'}, u'set connection timeout embryonic 0:00:10 idle
1:00:00'})
changed: [asav1] => (item={u'class': u'lowweb-tcp-traffic'}, u'set connection embryonic-conn-max 300 per-clie
nt-max 1000 per-client-embryonic-max 100'})
changed: [asav1] => (item={u'class': u'lowweb-tcp-traffic'}, u'set connection timeout embryonic 0:00:10 idle
1:00:00'})
changed: [asav1] => (item={u'class': u'all-tcp-traffic'}, u'set connection embryonic-conn-max 100 per-client-
max 500 per-client-embryonic-max 50'})
changed: [asav1] => (item={u'class': u'all-tcp-traffic'}, u'set connection timeout embryonic 0:00:10 idle 1:0
0:00'})

TASK [policy-framework : assign service global policy] *************************
changed: [asav1]

PLAY RECAP *********************************************************************
asav1                      : ok=22   changed=11   unreachable=0    failed=0
```

Figura 54-2: Ejecución del playbook config_firewall
Fuente: YUNGA, Alex, 2018

2.6.5 Topología 5 – Juniper

En esta topología se utiliza un switch capa 3 de Juniper que tiene un consumo de memoria RAM de 1024 MB (figura 55-2) para este escenario se emplea 7 dispositivos como se muestra en la figura 57-2.

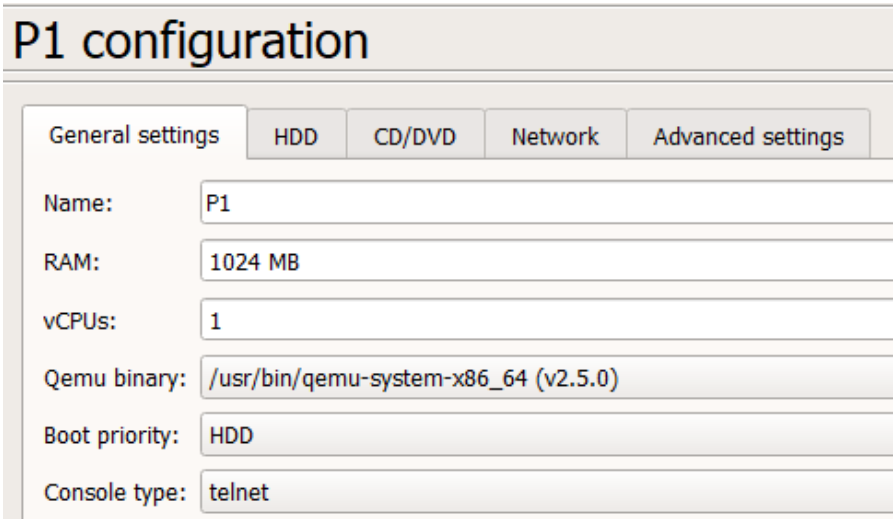


Figura 55-2: Consumo de Memoria RAM de un switch capa 3 de Juniper
Fuente: YUNGA, Alex, 2018

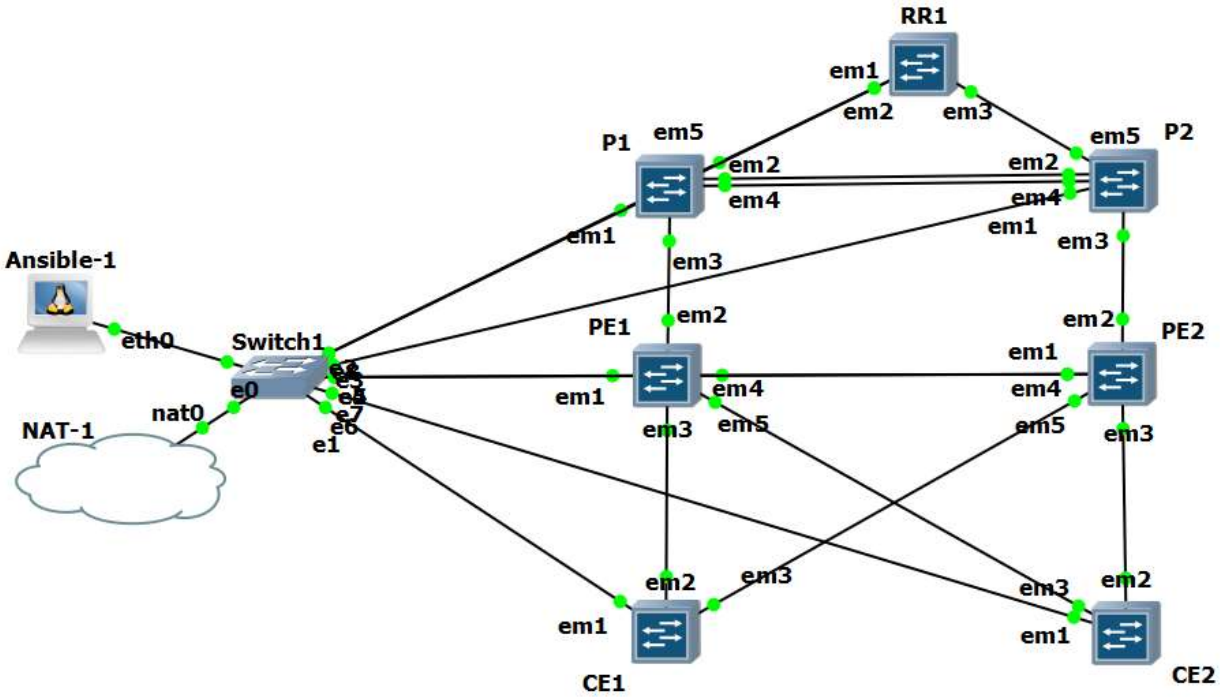


Figura 56-2: Escenario de la topología 5
Fuente: YUNGA, Alex, 2018

Se añade los dispositivos del nodo secundario en el inventario de Ansible para poder acceder a dichos dispositos a traves de SSH como se obserav en la figura 57-2.

```
Sending select for 192.168.122.182...
Lease of 192.168.122.182 obtained, lease time 3600
root@Ansible-1:~#
root@Ansible-1:~# cat gns3hosts >>/etc/hosts
root@Ansible-1:~# cat /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
192.168.122.61 P1
192.168.122.62 P2
192.168.122.63 PE1
192.168.122.64 PE2
192.168.122.65 CE1
192.168.122.66 CE2
192.168.122.67 RR1
```

Figura 57-2: Inventario de Ansible para la topología 5
Fuente: YUNGA, Alex, 2018

Se crea todos los ficheros, playbooks y plantillas para la configuración de esta topología para cada uno de los directorios host_vars, group_vars y templates.

El directorio tmp se caracteriza guardar los archivos .xml de todos los dispositivos. La figura 58-2 muestra el playbook del dispositivo PE1 donde indica las interfaces que se debe configurar, todos los playbooks del directorio host_vars varían en su configuración.

La figura 59-2 muestra la configuración que se va a aplicar a los dispositivos CE y la figura 60-2 muestra la configuración para el dispositivo RR1.

```
root@Ansible-1:~# ls
ansible.cfg     changes.log  gns3hosts   host_vars  tmp
baseconfig.yml  devices      group_vars  templates
root@Ansible-1:~# cd host_vars/
root@Ansible-1:~/host_vars# ls
CE1.yml  CE2.yml  P1.yml  P2.yml  PE1.yml  PE2.yml  RR1.yml
root@Ansible-1:~/host_vars# cat PE1.yml
host_name: PE1
host:
  loopback:
    local_ip: 172.16.0.11/32
    iso_net: 49.0000.0000.cccc.0003.00
  interfaces:
    - interface: em2
      description: to P1
      local_ip: 10.1.10.2/31
      metric: 10
    - interface: em3
      description: to CE1
      local_ip: 10.1.0.1/31
      metric: 10
```

Figura 58-2: Directorios, playbooks y platillas para la topología 5
Fuente: YUNGA, Alex, 2018

```
host_name: CE1
host:
  loopback:
    local_ip: 192.168.10.1/32
    iso_net: 49.0000.0000.cccc.0001.00
  interfaces:
    - interface: em2
      description: to PE1
      local_ip: 10.1.0.0/31
      metric: 10
    - interface: em3
      description: to PE2
      local_ip: 10.1.0.4/31
      metric: 10
  isis:
    enabled: false
    overload: false
  bgp:
    enabled: true
    as: 65001
    peers:
      external:
        65000:
          - ip: 10.1.0.1
          - ip: 10.1.0.5
      internal:
```

Figura 59-2: Playbook para el switch CE1
Fuente: YUNGA, Alex, 2018

```
host_name: RR1
host:
  loopback:
    local_ip: 172.16.0.201/32
    iso_net: 49.0000.0000.cccc.0005.00
  interfaces:
    - interface: em2
      description: to P1
      local_ip: 10.1.10.17/31
      metric: 10
    - interface: em3
      description: to P2
      local_ip: 10.1.10.19/31
      metric: 10
  isis:
    enabled: true
    overload: true
  bgp:
    enabled: true
    as: 65000
    peers:
      external:
      internal:
        - ip: 172.16.0.11
        - ip: 172.16.0.22
```

Figura 60-2: Playbook para el switch RR1
Fuente: YUNGA, Alex, 2018

En esta topología solo se va a aplicar las configuraciones mediante un playbook principal (figura 61-2) que abarca dos tareas de configuración para los switchs PE, P y CE como también para los routers RR. La primera tarea (**Build Base Configuration**) se encarga de construir la configuración para los dispositivos en formato xml y la segunda tarea (**Install Template**) se encarga de aplicar esa configuración mediante las plantillas de Jinja2 (Ver **ANEXO H**)

baseconfig.yml

```
- name: Build base configuration
  hosts: P_Nodes,PE_Nodes,CE_Nodes
  gather_facts: no
  roles:
    - Juniper.junos
  tasks:
    - name: Build Base Configuration
      template: src=templates/baseconfig.j2 dest=/tmp/{{ inventory_hostname  }}.xml

    - name: Install template
      junos_install_config:
        user: "alex"
        passwd: "juniper123"
        comment: "configured by ansible"
        file: "/tmp/{{ inventory_hostname  }}.xml"
        logfile: "changes.log"
        replace: true
```

Figura 61-2: Playbook principal baseconfig
Fuente: YUNGA, Alex, 2018

2.6.6 Topología 6 – Arista

En esta topología se utiliza un switch multicapa de ARISTA que tiene un consumo de memoria RAM de 2048 MB (figura 62-2) para este escenario se manipula 3 dispositivos como se muestra en la figura 74-2.

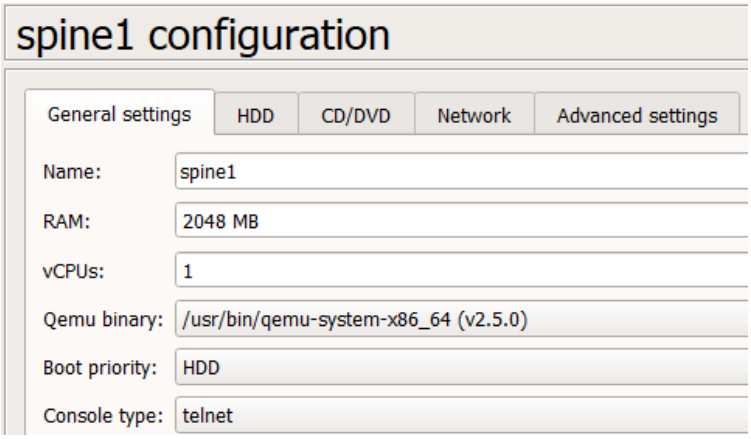


Figura 62-2: Consumo de RAM de un switch multicapa de Arista
Fuente: YUNGA, Alex, 2018

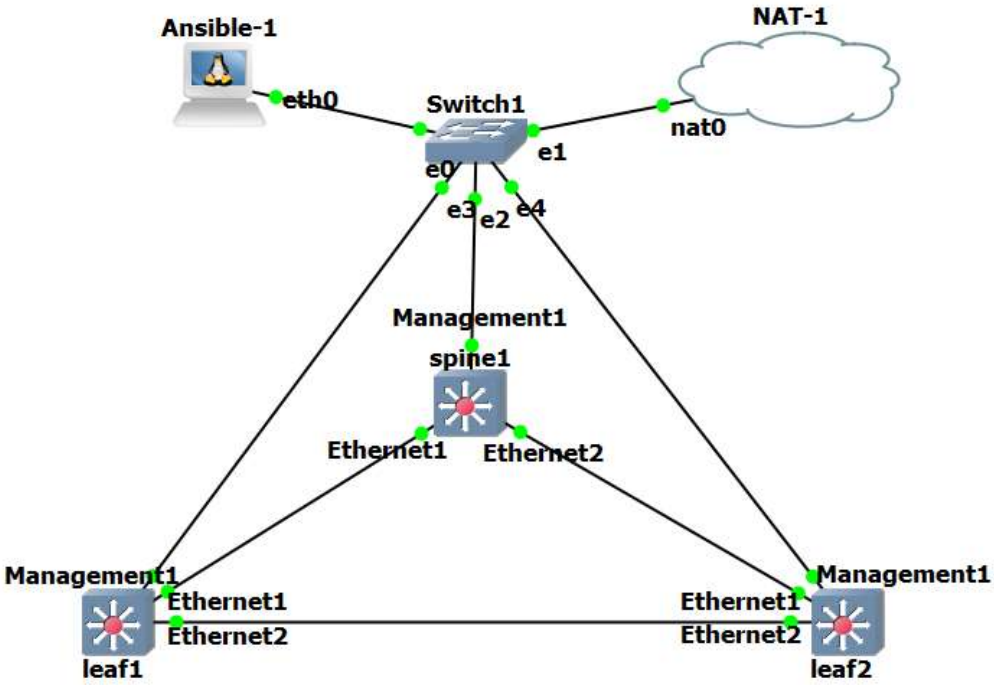


Figura 63-2: Escenario de la topología 6
Fuente: YUNGA, Alex, 2018

Se limita a esta cantidad de dispositivos debido a que cuando se quiere encender otro switch Arista la VM de GNS3 da un mensaje indicando el consumo de memoria RAM en porcentaje, en donde se indica cuáles son los motivos del porque no enciende ese dispositivo.

En todas las topologías se debe tomar en cuenta el consumo de la memoria RAM, para este ejemplo se intentó arrancar un dispositivo llamado "AristavEOS4.20.1F-1" como se observa en la figura 64-2 que da un mensaje de advertencia indicándonos que no se puede correr ese dispositivo porque no se cuenta con la memoria RAM requerida.

"AristavEOS4.20.1F-1" requires 2048MB of RAM to run but there is only 1462MB - 81.7% of RAM left on "gns3vm"
"AristavEOS4.20.1F-1" requires 2048MB of RAM to run but there is only 1462MB - 81.7% of RAM left on "gns3vm"
"AristavEOS4.20.1F-1" requires 2048MB of RAM to run but there is only 1497MB - 81.2% of RAM left on "gns3vm"
"AristavEOS4.20.1F-1" requires 2048MB of RAM to run but there is only 1497MB - 81.2% of RAM left on "gns3vm"

Figura 64-2: Mensaje de la máquina virtual de GNS3
Fuente: YUNGA, Alex, 2018

Se crea todos los playbooks, plantillas, directorios y ficheros para esta topología como se muestra en la figura 65-2.

```
Sending select for 192.168.122.48...
Lease of 192.168.122.48 obtained, lease time 3600
root@Ansible-1:~#
root@Ansible-1:~# ls
ansible.cfg            devices    group_vars  roles
arista_check_icmp.yml  gns3hosts  host_vars   site.yml
root@Ansible-1:~# cd host_vars/
root@Ansible-1:~/host_vars# ls
leaf1.yml  leaf2.yml  spine1.yml
root@Ansible-1:~/host_vars# cat spine1.yml
---

loopback: 10.255.0.1/32

bgp_fabric:
  asn: 65000
  router_id: 10.255.0.1
  neighbor:
    - {address: 10.0.255.2, remote_as: 65001}
    - {address: 10.0.255.10, remote_as: 65002}

  networks:
    - 10.255.0.1/32
    - 10.0.255.0/30
    - 10.0.255.8/30

interfaces:
  Ethernet1:
    alias: downlink-leaf1
    port_state: no switchport
    address: 10.0.255.1/30
  Ethernet2:
    alias: downlink-leaf2
    port_state: no switchport
    address: 10.0.255.9/30
```

Figura 65-2: Playbooks y directorios para la topología 6
Fuente: YUNGA, Alex, 2018

Antes de ejecutar el playbook principal se añade los dispositivos del nodo secundario al inventario de Ansible (figura 66-2). El playbook site se encarga de aplicar las configuraciones a los dispositivos.

```
root@Ansible-1:~# cat gns3hosts >> /etc/hosts
root@Ansible-1:~# cat /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
192.168.122.71 spine1
192.168.122.72 leaf1
192.168.122.73 leaf2
```

Figura 66-2: Inventario de Ansible para la topología 6
Fuente: YUNGA, Alex

CAPÍTULO III

3. ANÁLISIS Y RESULTADOS

Se emplea la ecuación de la figura 1-3 que sirve para determinar el tamaño de la muestra de una población de 27 estudiantes que aprobaron la asignatura de Conmutación y Ruteo II en el periodo académico Abril - Agosto 2018, se toma como muestra a esta población debido a que conocen de las configuraciones de los protocolos de BGP, MPLS y VLANS.

$$n = \frac{N \times Z_a^{\,2} \times p \times q}{d^2 \times (N-1) + Z_a^{\,2} \times p \times q}$$

Figura 1-3: Ecuación de muestreo para poblaciones conocidas

En base a los datos, se obtiene que la muestra de este estudio estará conformada por 25 estudiantes, por tener acceso a todo el grupo de estudiantes de dicha asignatura y porque en poblaciones pequeñas es recomendable trabajar con toda la población para obtener mejores resultados.

3.1 Resultados de la encuesta

Para determinar las mejores características de Ansible como herramienta de automatización, se han analizado los resultados obtenidos a través de una encuesta que consta de 8 preguntas a toda la población de 27 estudiantes con el fin de obtener información del grado de satisfacción, la facilidad de aprendizaje y el nivel de acuerdo con relación a dicha herramienta para medir el nivel de compatibilidad de Ansible con las plataformas de networking existentes en el mercado con respecto a las configuraciones de red.

Debido a que los estudiantes no están familiarizados con Ansible, se realizó una inducción de conocimientos para dar a conocer las características, manejo y los componentes de esta herramienta en el lapso de 1 hora y 30 minutos, y al término de la charla se procedió a aplicar la encuesta.

3.1.1 Facilidad de aprendizaje

Para evaluar esta característica se realizó la siguiente pregunta: ***Indique el nivel de dificultad para el aprendizaje de YAML que es usado para la construcción de playbooks y de JINJA2 que es usado para las plantillas.*** En el gráfico 1-3 se muestran los resultados para dichos lenguajes, YAML da como resultado que el 59.3% de los estudiantes consideran que es fácil su aprendizaje, el 29,6% indican que este lenguaje tiene un grado intermedio de dificultad es decir que no está fácil ni difícil y el restante con 11.1% indica que su aprendizaje es muy complejo.

Por otra parte, en JINJA2 se muestra que el 48.1% indica que este lenguaje es fácil en su aprendizaje, el 33.3% indican un grado intermedio de dificultad y el sobrante 18.5% indica que es muy difícil.

En si los dos lenguajes presentan niveles bajos de dificultad que no llegan al 25% debido a que la mayoría de los estudiantes están familiarizados con otros lenguajes de programación, obteniendo como resultado que los dos lenguajes brindan una mayor facilidad de comprensión en su sintaxis y estructura.

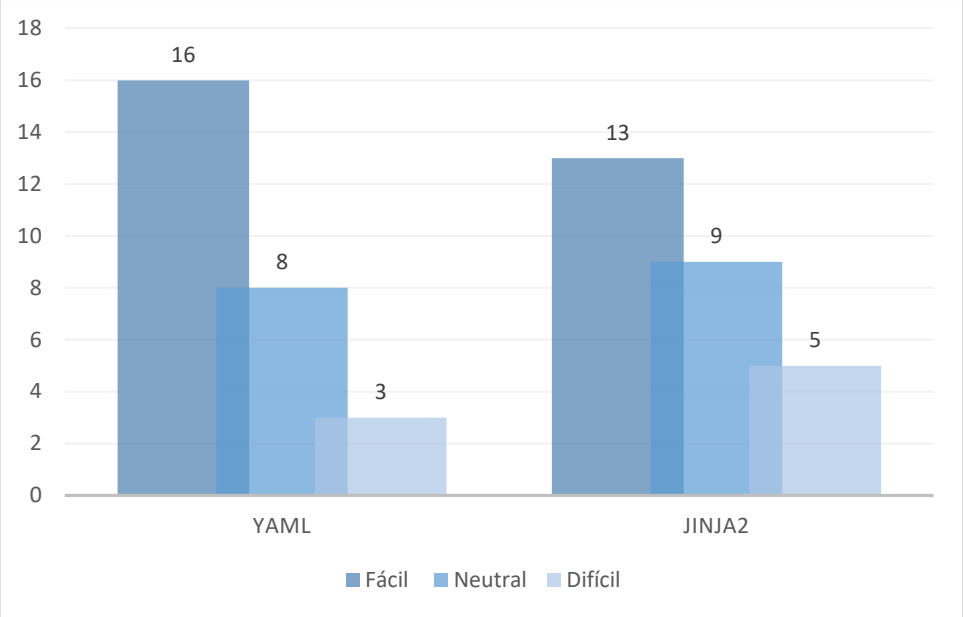


Gráfico 1-3: Evaluación de la facilidad de aprendizaje en Ansible
Realizado por: YUNGA, Alex

3.1.2 Administración de tareas y creación archivos

Aquí se evalúan dos características importantes, la primera se realiza mediante la pregunta: ***Indique el nivel de satisfacción sobre la administración de las distintas tareas a través de playbooks en Ansible.*** El grafico 2-3 muestra que el 51.9% de los estudiantes indican que la administración de tareas es excelente por lo que a través de un playbook se puede administrar varias tareas, un 37% indican que la administración es regular es decir presentan un pequeño grado de dificultad y el 11.1% califican como muy mala a la herramienta por la gran cantidad de tareas que administra.

Y la segunda característica se evalúa a través de la siguiente pregunta: ***Indique el nivel de satisfacción sobre la utilización de comandos básicos de Linux en Ansible para la creación de archivos en el nodo principal,*** el gráfico 2-3 indica que el 51.9% de los estudiantes saben de los comandos y están muy familiarizados, el 33.3% muestra que los estudiantes conocen de los comandos y el 14.8 % que no están familiarizados con dichos comandos.

Se evalúan estas dos características juntas porque están relacionadas entre sí debido a que no se puede administrar las tareas sin antes haber utilizado los comandos básicos de Linux para la creación de archivos (ficheros, directorios, playbooks y plantillas).

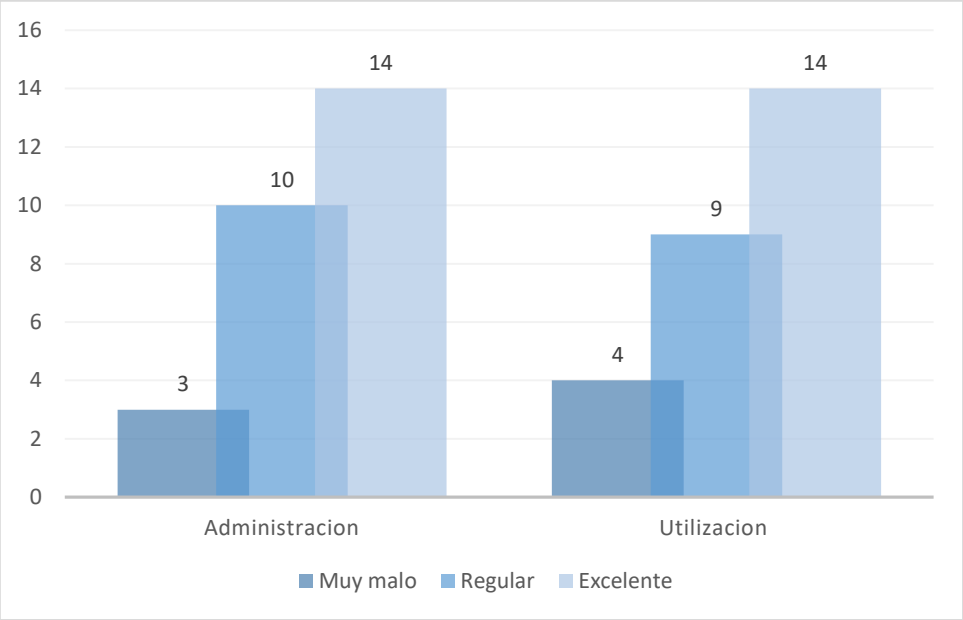


Gráfico 2-3: Evaluación para la administración de tareas y creación archivos
Realizado por: YUNGA, Alex

3.1.3 Seguridad

Al evaluar esta característica se planteó la siguiente pregunta: ***Como califica el nivel de seguridad que ofrece Ansible con la utilización del protocolo SSH.*** El gráfico 3-3 indica que un 66.7% de los estudiantes califican la seguridad de Ansible como excelente al tener una arquitectura "agentless" es decir sin agentes porque no utiliza ninguna base de datos, agente externo o demonio en el nodo secundario, mejorando la seguridad al no dejar ninguna vulnerabilidad en los dispositivos mientras que el 33.3% de participantes califican de regular porque consideran que puede haber vulnerabilidades en la seguridad.

Los estudiantes califican de excelente la seguridad de Ansible porque utiliza SSH mediante el uso de cifrado para asegurar la transferencia segura de información entre el nodo principal y los clientes del nodo secundario. Las claves SSH proporcionan una forma más segura de iniciar sesión en los dispositivos del nodo secundario con SSH que usar solo una contraseña ya que una contraseña se

puede descifrar con un ataque de fuerza bruta, las claves SSH son casi imposibles de descifrar solo con la fuerza bruta.

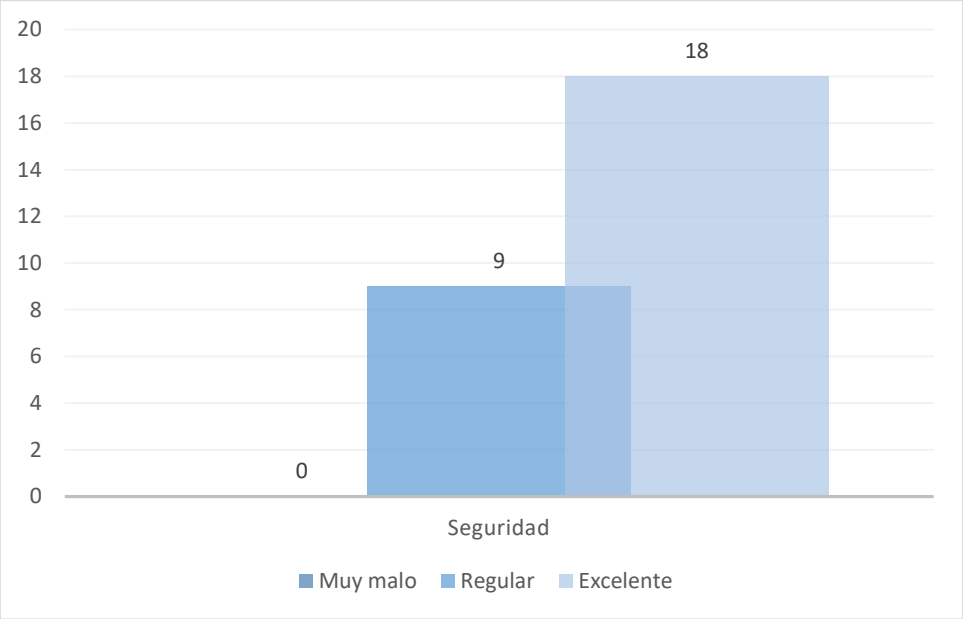


Gráfico 3-3: Evaluación de la seguridad de Ansible
Realizado por: YUNGA, Alex

3.1.4 Cantidad de dispositivos

Se evalúa esta característica a través de la siguiente pregunta: ***Está de acuerdo con la cantidad de dispositivos que Ansible puede administrar.*** El gráfico 4-3 indica que el 25.9% de estudiantes se muestran indecisos a esta característica y el 74.1% se encuentran totalmente de acuerdo con la configuración de varios dispositivos mediante la ejecución de un playbook.

Mas del 50% de los estudiantes están de acuerdo con la cantidad de 10 dispositivos utilizados para la topología 3. Esta seguía siendo una cantidad baja para las capacidades que representa esta herramienta, hasta que se dio con la conferencia presentada por Landon Holley y James Mighion de Red Hat que discuten sobre la arquitectura y las estrategias involucradas en la automatización de la red, en donde indican que se puede gestionar 15000 dispositivos de red con Ansible.

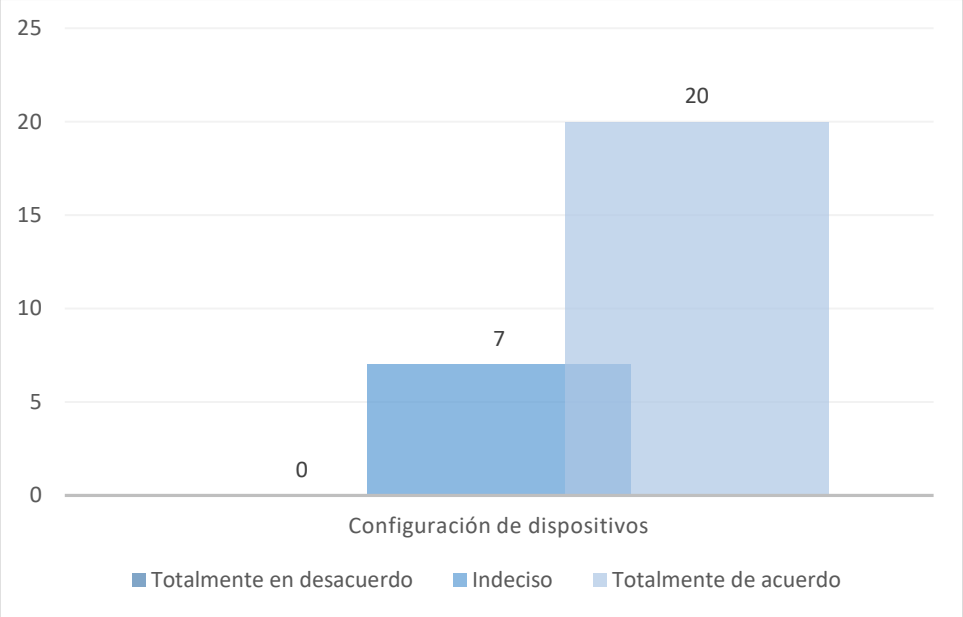


Gráfico 4-3: Evaluación para la cantidad de dispositivos
Realizado por: YUNGA, Alex

3.1.5 Tiempos y errores de configuración

Se evalúan dos características relevantes, la primera se realiza por medio de la siguiente pregunta: ***Está de acuerdo que Ansible reduce los tiempos de configuración para el nodo secundario.*** El gráfico 5-3 indica que el 85.2% de los estudiantes está de acuerdo con la reducción de los tiempos de configuración que presenta Ansible y el 14.8% restante se muestran indecisos.

La mayoría de los estudiantes están de acuerdo con la característica de reducir los tiempos de configuración porque conocen que es importante la reducción de estos tiempos en la configuración de los dispositivos permitiendo mejorar la velocidad para implementar nuevos cambios en la red.

Y la segunda característica se valora a través de la siguiente pregunta: ***Está de acuerdo que Ansible reduce los errores de configuración manual.*** En el grafico 5-3 muestra que el 66.7% de los estudiantes están de acuerdo en la reducción de errores de configuración y el 33.3% restante muestran que están indecisos con dicha característica.

Un porcentaje alto de los estudiantes están de acuerdo con la reducción de errores de configuración ya que conocen que el uso de Ansible como herramienta de automatización ayuda a tener un comportamiento más predecible, permitiendo a los administradores de red tengan una mayor seguridad de que la tarea se realizará correctamente en la primera vez sin la preocupación de que exista algún error humano.

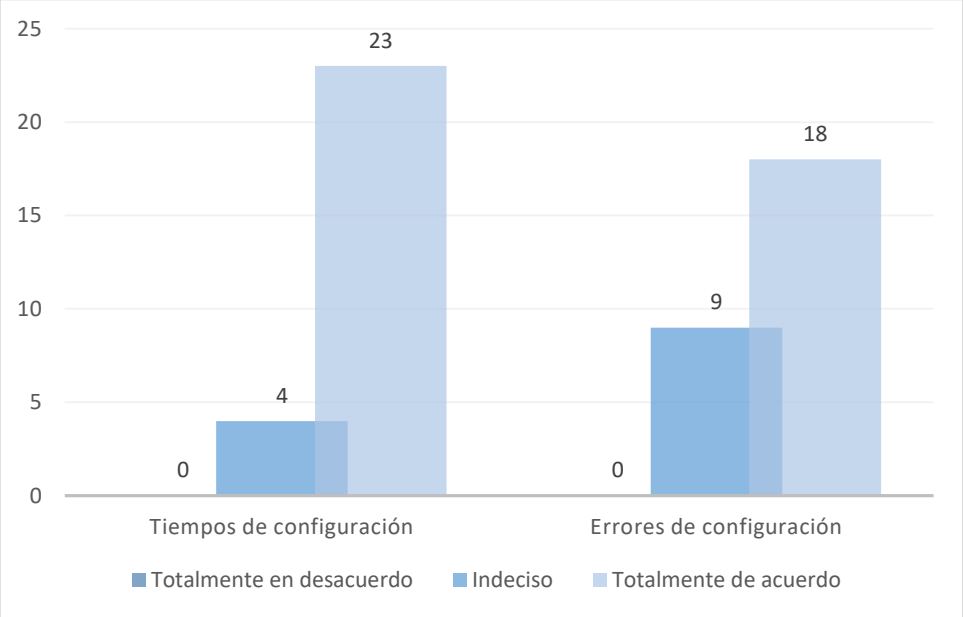


Gráfico 5-3: Evaluación para los tiempos y errores de configuración
Realizado por: YUNGA, Alex

3.2 Verificación de resultados en las topologías

3.2.1 Resultados de la topología 1

El playbook ***access*** de la figura 2-3 aplica las configuraciones a los switches de acceso, el playbook ***show*** muestra las configuraciones de los switches de distribución y acceso.

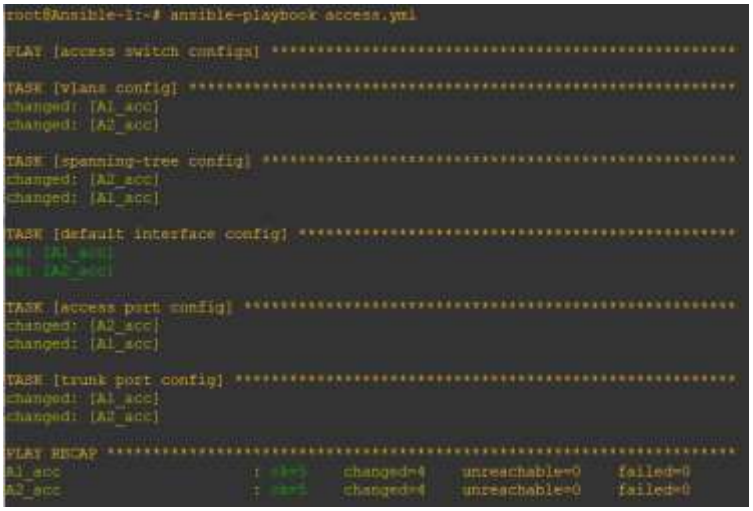


Figura 2-3: Ejecución del playbook access
Fuente: YUNGA, Alex, 2018

Se muestra la prueba de conectividad (figura 4-3) ya que cada switch de distribución asigna sus respectivas vlans con su dirección IP estas son accesibles dentro de su propio switch. En los dispositivos Cisco indican en tiempo real que cambio se han efectuado y de dónde como se observa en la figura 3-3.

Figura 3-3: Cambios efectuados en el switch D1_dist
Fuente: YUNGA, Alex, 2018

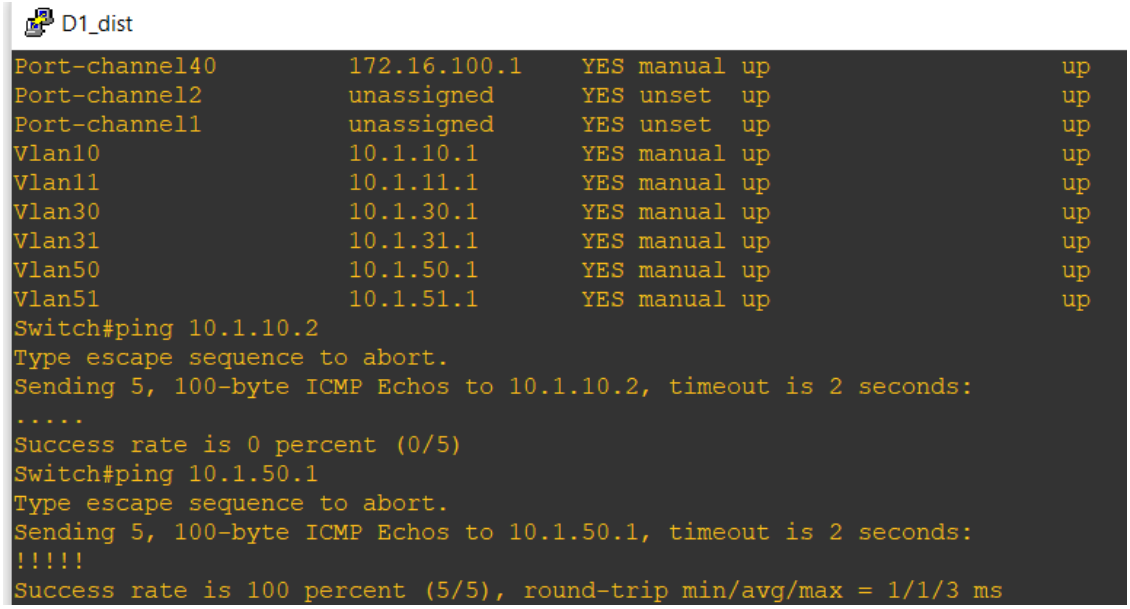


Figura 4-3: Prueba de conectividad entre las vlans de los switches de DISTRIBUCION
Fuente: YUNGA, Alex, 2018

3.2.2 Resultados de la topología 2

Se revisa la configuración aplicada en los routers de la topología 2, al abrir el router R4 se muestra lo que se configuro y a través de qué, también indica que dirección IP hizo el cambio (figura 5-3).

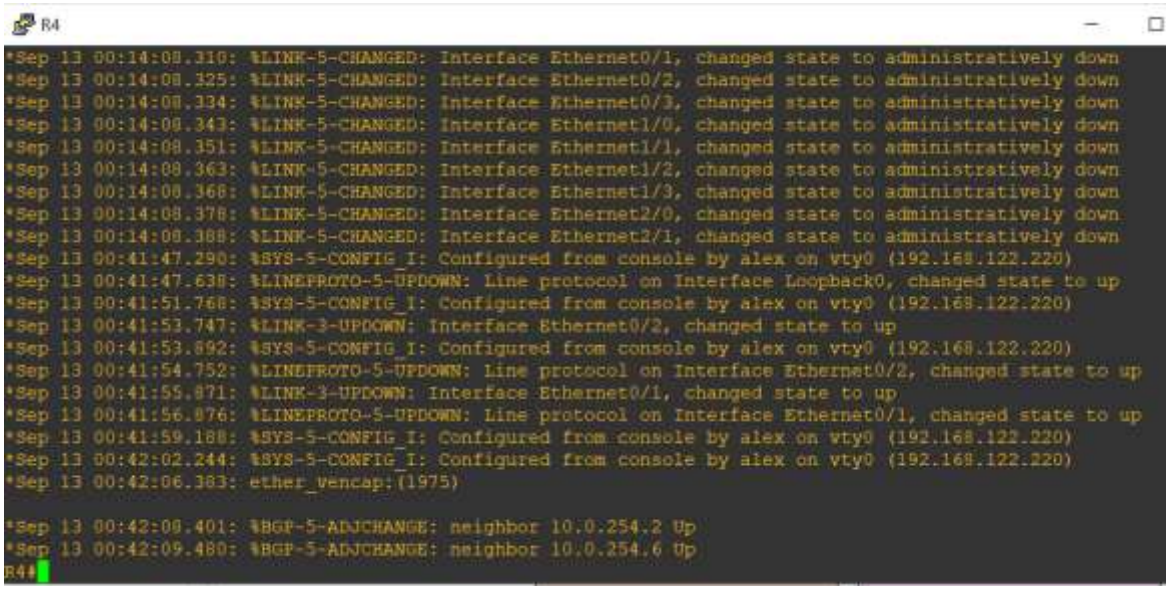


Figura 5-3: Configuración aplicada en el R4 por medio de SSH
Fuente: YUNGA, Alex, 2018

Con el playbook ***check_icmp*** se valida la configuración mediante el ping entre los routers como se observa en la figura 6-3.

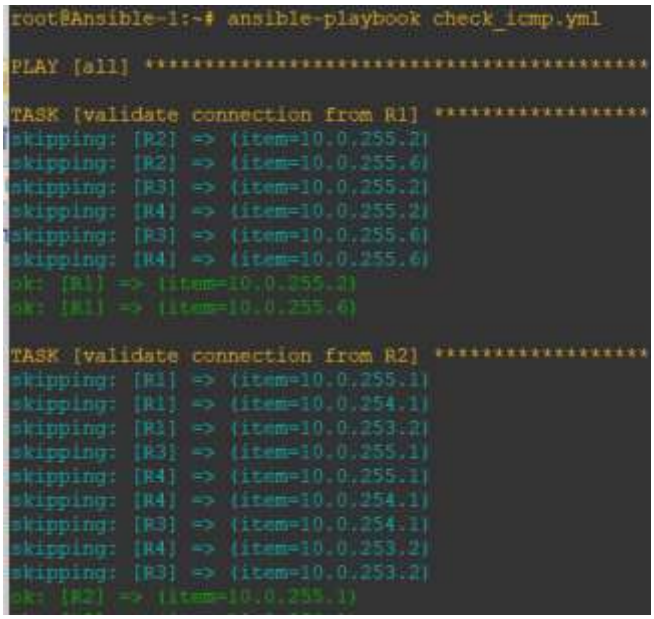


Figura 6-3: Ejecución del playbook check_icmp
Fuente: YUNGA, Alex, 2018

Se accede al router R4 y se procede a realizar pings a las distintas direcciones de esta topología como se muestra en la figura 7-3.

```
R4#ping 10.0.255.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.255.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/2 ms
R4#ping 10.0.255.5
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.255.5, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/2 ms
R4#ping 10.0.255.2
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.255.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms
```

Figura 7-3: Ejecución de pings desde R4
Fuente: YUNGA, Alex, 2018

3.2.3 Resultados de la topología 3

Los resultados al terminar la ejecución de cualquier playbook se muestran en PLAY RECAP (figura 8-3) que es un apartado que contiene información de todos los hosts a los cuales se aplicó las tareas de configuración. Cuando una tarea falla, todas las tareas posteriores no se ejecutarán en los hosts, en ese caso se solucionan los problemas y se vuelve a ejecutar las tareas que fallaron. El PLAY RECAP de los playbooks deploy_pe y deploy_ce dan como resultado todo de color verde que indica que no existe ningún error.

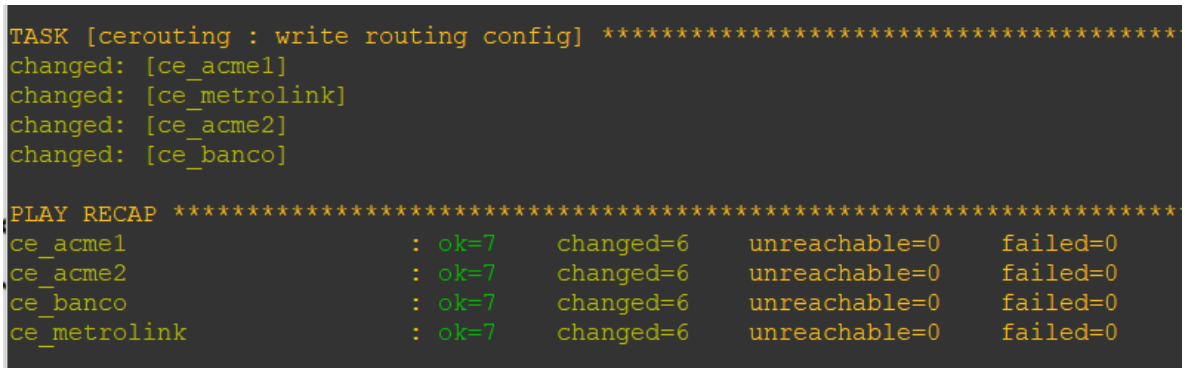


Figura 8-3: Play Recap del playbook deploy_ce
Fuente: YUNGA, Alex, 2018

Se ejecuta el playbook check de la figura 9-3 que permite validar la configuracion entre los routers, como resultado de la ejecución del playbook check da todo correcto en letras verdes.

En la figura 10-3 se realiza la validación de conectividad a través de ping desde el router ce_acme1.

```
TASK [validate connection from pe-paris] ************************************
ok: [pe_paris] => (item=10.255.0.1)
ok: [pe_paris] => (item=10.255.0.2)
ok: [pe_paris] => (item=10.255.0.4)

TASK [validate connection from pe-madrid] ***********************************
ok: [pe_madrid] => (item=10.255.0.1)
ok: [pe_madrid] => (item=10.255.0.2)
ok: [pe_madrid] => (item=10.255.0.3)

TASK [validate bgp connection from ce-acme1] ********************************
ok: [ce_acme1] => (item=172.2.0.222)

TASK [validate bgp connection from ce-acme2] ********************************
ok: [ce_acme2] => (item=172.1.0.111)

PLAY RECAP ******************************************************************
P1                         : ok=0    changed=0    unreachable=0    failed=0
P2                         : ok=0    changed=0    unreachable=0    failed=0
ce_acme1                   : ok=1    changed=0    unreachable=0    failed=0
ce_acme2                   : ok=1    changed=0    unreachable=0    failed=0
ce_banco                   : ok=0    changed=0    unreachable=0    failed=0
ce_metrolink               : ok=0    changed=0    unreachable=0    failed=0
pe_dublin                  : ok=1    changed=0    unreachable=0    failed=0
pe_londres                 : ok=1    changed=0    unreachable=0    failed=0
pe_madrid                  : ok=1    changed=0    unreachable=0    failed=0
pe_paris                   : ok=1    changed=0    unreachable=0    failed=0
```

Figura 9-3: Ejecución del playbook check
Fuente: YUNGA, Alex, 2018

```
ce_acme1# ping 192.168.101.2
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.101.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 12/26/40 ms
ce_acme1#trace
ce_acme1#traceroute 192.168.101.2
Type escape sequence to abort.
Tracing the route to 192.168.101.2
VRF info: (vrf in name/id, vrf out name/id)
  1 192.168.100.1 8 msec 16 msec 12 msec
  2 192.168.101.1 [AS 1] 40 msec 12 msec 20 msec
  3 192.168.101.2 [AS 1] 16 msec 28 msec 28 msec
```

Figura 10-3: Ping desde el router ce_acme1
Fuente: YUNGA, Alex, 2018

3.2.4 Resultados de la topología 4

Se verifica la conectividad entre los routers mediante el playbook ***asav1_check_icmp*** de la figura 11-3.

```
root@Ansible-1:~# ansible-playbook asav1_check_icmp.yml

PLAY [all] *********************************************************************

TASK [validate connection from rtr-1] ******************************************
skipping: [rtr-2] => (item=10.0.255.1)
skipping: [asav1] => (item=10.0.255.1)
ok: [rtr-1] => (item=10.0.255.1)

TASK [validate connection from rtr-2] ******************************************
skipping: [rtr-1] => (item=217.100.100.1)
skipping: [asav1] => (item=217.100.100.1)
ok: [rtr-2] => (item=217.100.100.1)

TASK [validate from rtr-1 to internet] *****************************************
skipping: [rtr-2] => (item=8.8.8.8)
skipping: [rtr-2] => (item=8.8.4.4)
skipping: [asav1] => (item=8.8.8.8)
skipping: [rtr-2] => (item=217.100.100.254)
skipping: [asav1] => (item=8.8.4.4)
skipping: [asav1] => (item=217.100.100.254)
ok: [rtr-1] => (item=8.8.8.8)
ok: [rtr-1] => (item=8.8.4.4)
ok: [rtr-1] => (item=217.100.100.254)

PLAY RECAP *********************************************************************
asav1                      : ok=0    changed=0    unreachable=0    failed=0
rtr-1                      : ok=2    changed=0    unreachable=0    failed=0
rtr-2                      : ok=1    changed=0    unreachable=0    failed=0
```

Figura 11-3: Prueba de conectividad para la topología 4
Fuente: YUNGA, Alex, 2018

3.2.5 Resultados de la topología 5

Se ejecuta el playbook baseconfig que se encarga de aplicar la configuración a todos switchs de dicha topología (Ver **ANEXO I**).

Se verifica los cambios aplicados en los dispositivos, para este ejemplo se verifica en el switch RR1 como se observa en la figura 12-3.

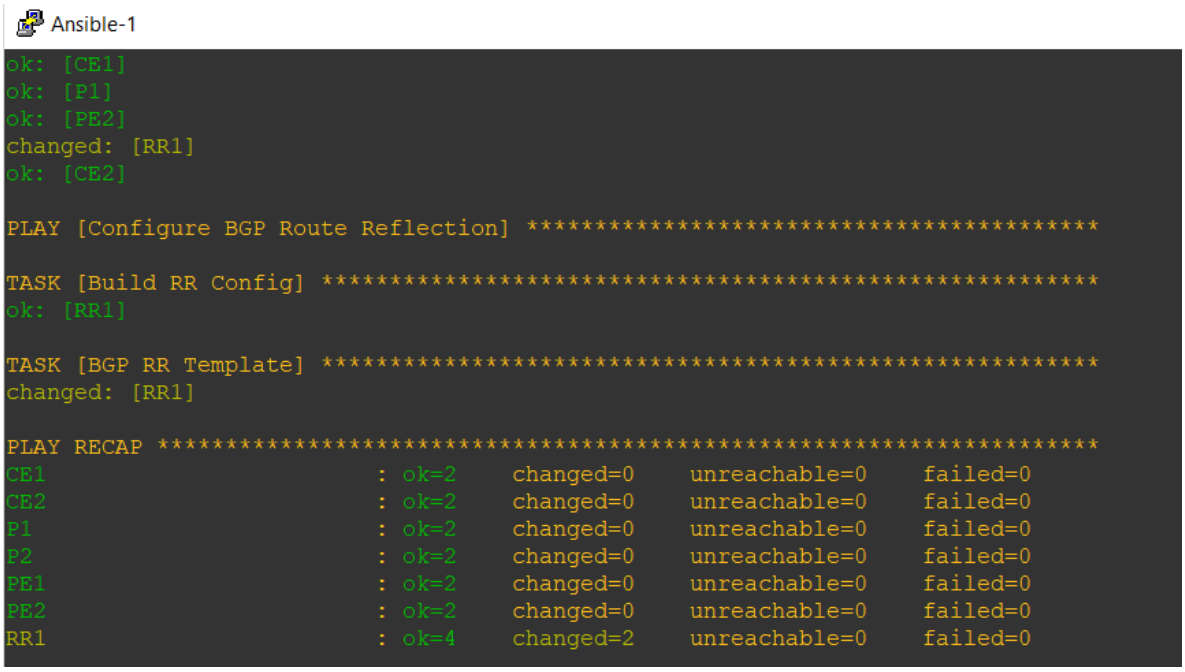


Figura 12-3: Ejecución del playbook baseconfig
Fuente: YUNGA, Alex, 2018

3.2.6 Resultados de la topología 6

Se ejecuta el playbook site de la figura 13-3 que se encarga de aplicar todas las configuraciones en los dispositivos ***spine1, leaf1 y leaf2.***

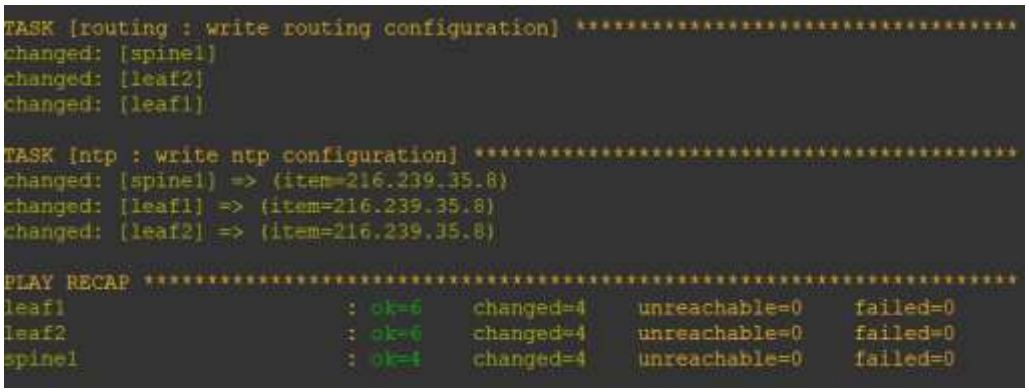


Figura 13-3: Playbook site para la topología 6
Fuente: YUNGA, Alex, 2018

Al aplicar los cambios se realiza la prueba de conectividad para esta topología como se observa en la figura 14-3.

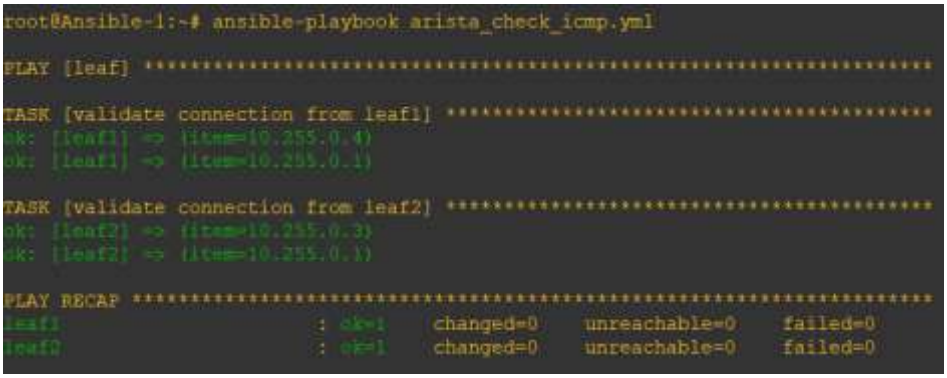


Figura 14-3: Verificación de conectividad
Fuente: YUNGA, Alex, 2018

CONCLUSIONES

La implementación de Network Automation permite mejorar los procesos actuales (configuraciones manuales) y proporciona un único lenguaje en toda la topología de la red mediante YAML, permitiendo así optimizar la gestión, la disponibilidad y los costos de la red, se deben considerar varios aspectos referentes al rendimiento de la red y de los equipos cuando se vaya a implementar en una infraestructura real.

Se comprobó que la tecnología de Network Automation realizada en GNS3 permitió configurar varios dispositivos de red al mismo tiempo, dando la posibilidad de reducir los errores y los tiempos de configuración en los equipos de las plataformas de networking utilizadas.

Ansible puede aplicar cualquier configuración a los dispositivos de red (router, switch y firewall) mediante el uso de playbooks (YAML) que se encargan de guardar la configuración para cada dispositivo en la máquina central y las plantillas (JINJA2) que son las encargadas del cómo se va a aplicar dicha configuración, el único requisito es que la plataforma de networking sea compatible con la herramienta de automatización.

La facilidad de aprendizaje que muestra Ansible con porcentajes altos indicando que no hay ninguna complejidad en el aprendizaje para los lenguajes de scripting como lo es YAML con un 59.3% y JINJA2 con un 48.1% y la administración de tareas y creación de archivos que muestra un porcentaje alto en la satisfacción con dichas características con un 51.9% se determina que la compatibilidad de Ansible con las diferentes plataformas de networking es óptimo ya que los estudiantes conocieron de cerca la manipulación de las configuraciones de red en Cisco, Juniper y Arista.

Se implementó la tecnología de Network Automation mediante las configuraciones de VLANS, BGP y MPLS en los dispositivos de red en diferentes infraestructuras tales como CISCO, JUNIPER y ARISTA, donde se seleccionó dichas plataformas en bases a las características Benchmarking, esta parte del estudio generó la propuesta de implementación presentada en el Capítulo 2.

RECOMENDACIONES

Al momento de implementar la tecnología de Network Automation en las distintas topologías de red, a más de miles de dispositivos de red se debería de llevar una documentación bien estructura y ordenada de todos los playbooks y plantillas para toda la red. Una de las herramientas que podría utilizar para llevar dicha documentación es Sublime Text 3 que es un editor de texto y editor de código fuente que está escrito en C++ y Python que brinda la facilidad de soportar distintos lenguajes de programación.

Para implementar Ansible en GNS3, es recomendable emplear una computadora con 16 GB de memoria RAM que para este caso se utilizó 8 GB para la máquina virtual y 6 núcleos para el procesador de esta estos requisitos de hardware influyen directamente en la cantidad de dispositivos que se vaya a utilizar.

Se debe considerar las características de la red en la que se vaya a implementar el provisionamiento automático ya que dichas características pueden influir en el desempeño en el rendimiento de Ansible.

Para el caso particular de este estudio, fue necesario administrar efectivamente los directorios y los scripts de configuración de Ansible en la maquina central que opera con un sistema operativo basado en Linux.

Es necesario que se realicen nuevos estudios que implementen Ansible en escenarios reales ya que, para el caso de este, no fue posible conseguir todos los IOS de las plataformas soportadas por Ansible.

Una vez iniciado el escenario de las distintas topologías en GNS3 se debe de añadir desde Ansible las direcciones IP y los hostname de cada uno de los dispositivos que se vayan a configurar ya que al momento de cerrar el emulador estas direcciones se borran del fichero /etc/hosts.

Asignar una dirección IP estática a Ansible, para que no exista solapamiento en la asignación automática de una IP entre Ansible y el nodo secundario.

Al utilizar GNS3 se debe de tener cuidado con el cierre de este software, para cerrarlo se debe detener todo el escenario caso contrario provocaría daños en la máquina virtual de GNS3 o en los equipos de emulación.

BIBLIOGRAFÍA

CARBONELL, E. y GARCÍA, A.M., Comparación de Herramientas de gestión de la configuración. [en línea], 2016. DOI 10.13140/RG.2.1.1387.4329. Disponible en: https://www.researchgate.net/publication/298793936_COMPARISON_OF_CONFIGURATION_MANAGEMENT_TOOLS%5Cnhttp://www.informaticahabana.cu/sites/default/files/ponencias/GES64.pdf%5Cnhttp://www.informaticahabana.cu/es/node/751.

CHANG, C., HACKER, G., MAHROUA, R. y VAZQUEZ, A., Automation with Ansible Student Workbook (ROLE). , 2016. p. 524. DOI DO407-A2.0-en-1-20160804.

DAS, R., *Extending Ansible* [en línea]. Birmingham: Packt Publishing. 2016. ISBN 9781782175001. Disponible en: http://www.packtpub.com.

DUFFY, M., *Puppet Reporting and Monitoring* [en línea]. First. Birmingham - Mumbai. 2014. ISBN 9781783981427. Disponible en: www.packtpub.com.

DZERKALS, U., EVE-NG Professional Cookbook. , 2017. p. 197.

EDELMAN, J., Network Automation with Ansible - Dynamically Configuring Interface Descriptions | Jason Edelman's Blog. [en línea]. 2015. [Consulta: 9 agosto 2018]. Disponible en: http://jedelman.com/home/network-automation-with-ansible-dynamically-configuring-interface-descriptions/.

EDELMAN, J., *Network Automation with Ansible* [en línea]. First. Boston: O'Reilly Media. 2016. ISBN 9781491937839. Disponible en: http://safaribooksonline.com.

EDUREKA, DevOps Tools: Configuration Management & Deployment. [en línea]. 2017. [Consulta: 31 agosto 2018]. Disponible en: https://www.slideshare.net/EdurekaIN/chef-vs-puppet-vs-ansible-vs-saltstack-configuration-management-tools-comparison-edureka.

EFFICIENT IP, Descubre la Solución de red de automatización de EfficientIP. [en línea]. 2016. [Consulta: 9 agosto 2018]. Disponible en: http://www.efficientip.com/es/soluciones/automatizacion/.

ERIKSSON, M. y HALLBERG, V., Comparison between JSON and YAML for data serialization. *Bachelor's thesis* [en línea], 2011. p. 24. Disponible en:

http://www.csc.kth.se/utbildning/kth/kurser/DD143X/dkand11/Group2Mads/Rapport_Malin_Eriksson_Viktor_Hallberg.pdf%5Cnpapers2://publication/uuid/30D544E7-D519-4F0B-A27A-77E0D6B8A995.

GANGULI, S., BHALLA, V. y LERNER, A., NetOps 2.0: Embrace Network Automation and Analytics to Stay Relevant in the Digital Business Era. [en línea]. 2017. [Consulta: 13 enero 2017]. Disponible en: https://www.gartner.com/doc/3597717/netops--embrace-network-automation.

GEERLING, J., *Ansible for DevOps* [en línea]. S.l.: Leanpub. 2015. ISBN 9780986393402. Disponible en: http://leanpub.com/ansible-for-devops.

HALL, D., *Ansible Configuration Management* [en línea]. Second. Birmingham - Mumbai: Packt Publishing. 2015. ISBN 9781783280810. Disponible en: www.packtpub.com.

HEAP, M., *Ansible: From Beginner to Pro*. New York: Apress. 2016. ISBN 9781484216606.

HOCHSTEIN, L., *Ansible Up and Running : Automating configuration management and deployment the easy way*. First Edit. Boston: O'Reilly Media. 2014. ISBN 9781491915325.

HULL, N., Starting Ansible Easy guide for beginners. , 2016. p. 83.

JINJA, *Jinja2 Documentation* [en línea]. 2011. Disponible en: https://media.readthedocs.org/pdf/jinja2/latest/jinja2.pdf

KEATING, J., *Mastering Ansible* [en línea]. Birmingham: Packt Publishing. 2015. ISBN 978-1-78439-548-3. Disponible en: www.packtpub.com.

KWNETAPPS, Configuration Management - Salt, Puppet, Ansible, Chef. [en línea]. 2017. [Consulta: 31 agosto 2018]. Disponible en: http://blog.kwnetapps.com/configuration-management/.

MARTINEZ, J.F., Qué es la Automatización | Blog SEAS. [en línea]. 2017. [Consulta: 9 agosto 2018]. Disponible en: https://www.seas.es/blog/automatizacion/que-es-la-automatizacion/.

MOHAAN, M. y RAITHATHA, R., *Learning Ansible*. Birmingham - Mumbai: Packt Publishing. 2014. ISBN 9781783550630.

NEUMANN, J.C., *The book of GNS3 : build virtual network labs using Cisco, Juniper, and more*. San Francisco: No Starch Press. 2015. ISBN 978-1-59327-554-9.

OCAMPO ZUÑIGA, A., Emuladores de Red | Networking. [en línea]. 2015. [Consulta: 31 agosto

2018]. Disponible en: https://aocampoz.wordpress.com/2015/05/02/emuladores-de-red/.

QUILCATE, J., Ansible - Provisionamiento sin agentes. [en línea]. 2016. [Consulta: 9 agosto 2018]. Disponible en: https://jeqo.github.io/es/posts/2016-03-30-ansible-agentless-provisioning/.

RIVENES, L., What is Network Automation? - Datapath.io. [en línea]. 2016. [Consulta: 9 agosto 2018]. Disponible en: https://datapath.io/resources/blog/what-is-network-automation/.

ROUSE, M. y SCARPATI, J., What is network automation? [en línea]. 2017. [Consulta: 17 julio 2018]. Disponible en: https://searchnetworking.techtarget.com/definition/network-automation.

SHAH, G., *Ansible Playbook Essentials: Design automation blueprints using Ansible's playbooks to orchestrate and manage your multitier infrastructure* [en línea]. First. Birmingham - Mumbai: Packt Publishing. 2015. ISBN 9781784398293. Disponible en: www.packtpub.com.

SHARMA, R. y SONI, M., *Learning Chef: Automate your infrastructure using code and leverage DevOps with Chef* [en línea]. First Edit. Birmingham: Packt Publishing. 2015. ISBN 9781783285211. Disponible en: www.packtpub.com.

STALLINGS, W., Comunicaciones y Redes de Computadores. *Pearson Prentice Hall*, 2004. p. 896.

TUTORIALS POINTS, SaltStack Tutorial. [en línea], 2017. p. 85. ISSN 18271855. DOI 10.1128/AAC.03728-14. Disponible en: contact@tutorialspoint.com.

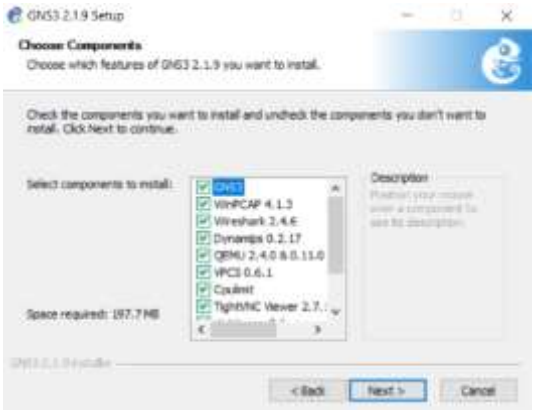
ULINIC, M. y HOUSE, S., *Network automation at scale Why us ?* [en línea]. First Edit. Boston: O'Reilly Media. 2017. ISBN 9781491992494. Disponible en: http://oreilly.com/safari.

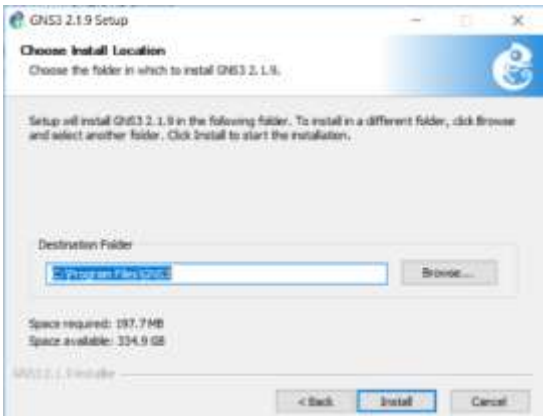
VENKATESH, P., Network Automation: Road to Programmable Networks | ONUG. [en línea]. 2018. [Consulta: 13 agosto 2018]. Disponible en: https://www.onug.net/blog/network-automation-road-to-programmable-networks/.

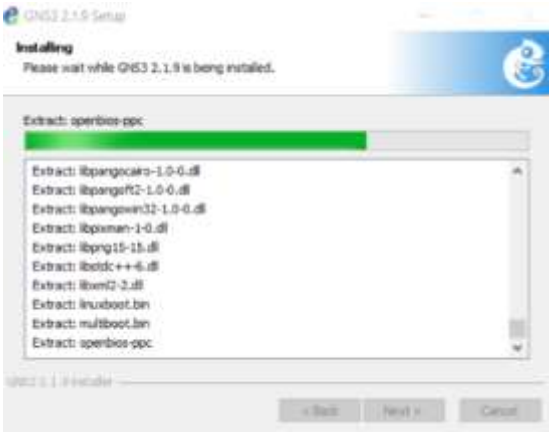
WELSH, C., *GNS3 Network Simulation Guide* [en línea]. Birmingham: Packt Publishing. 2013. ISBN 1782160809. Disponible en: http://cds.cern.ch/record/1633716.

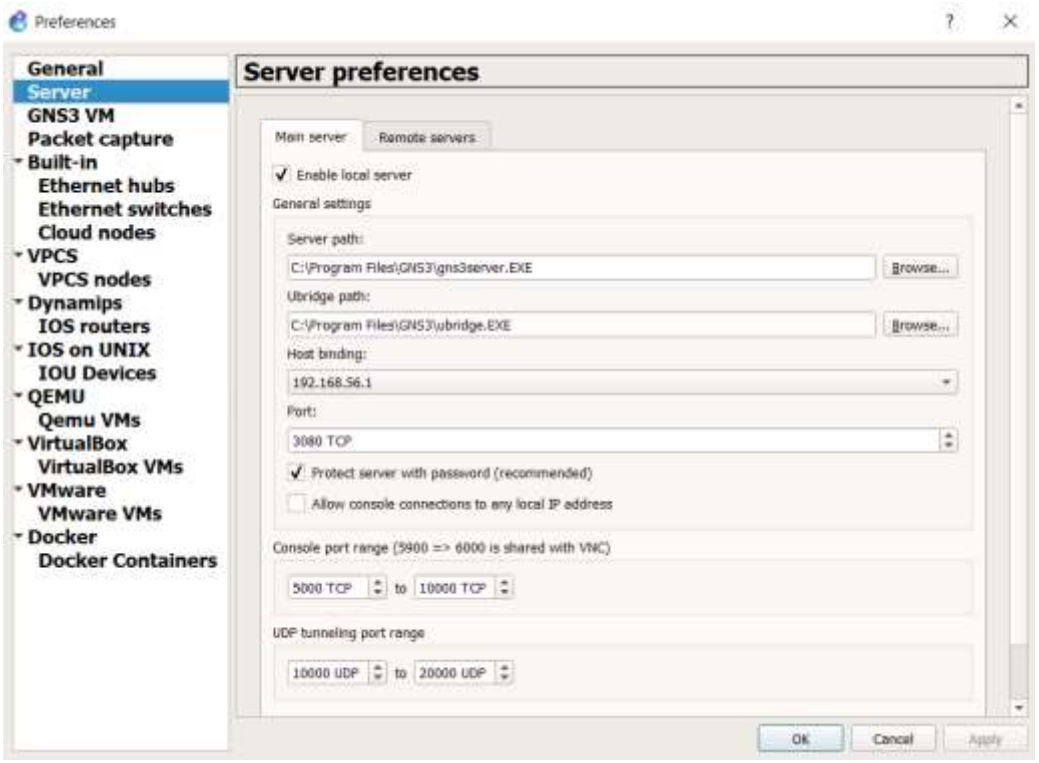
ANEXOS

ANEXO A. Instalación de GNS3

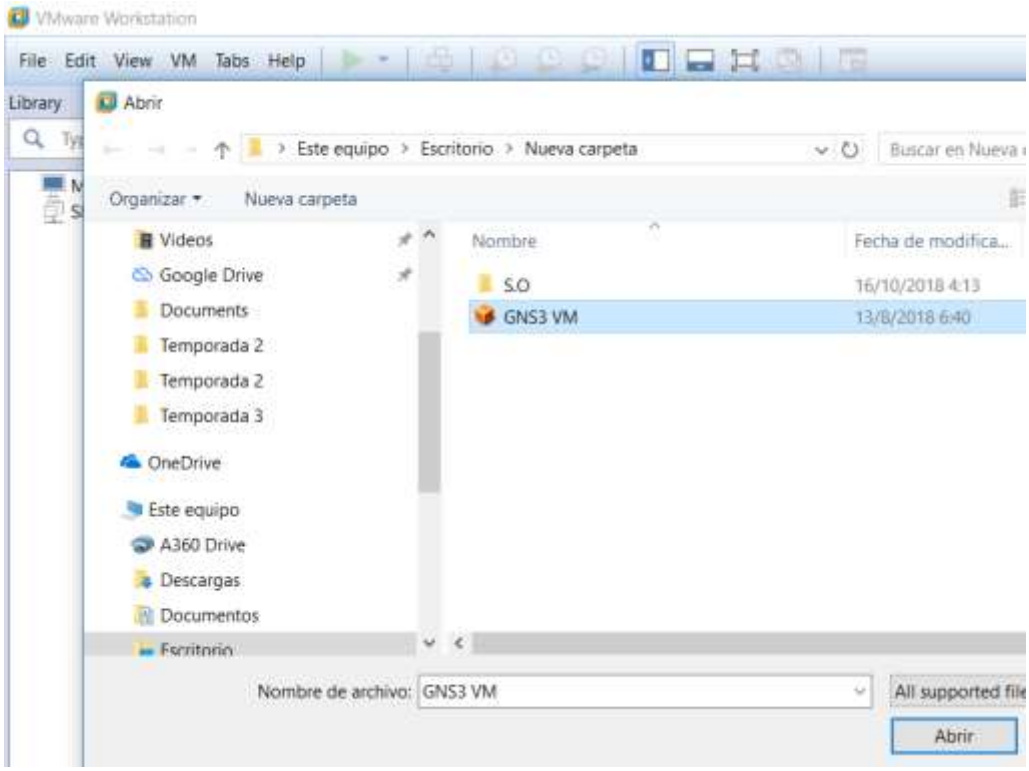


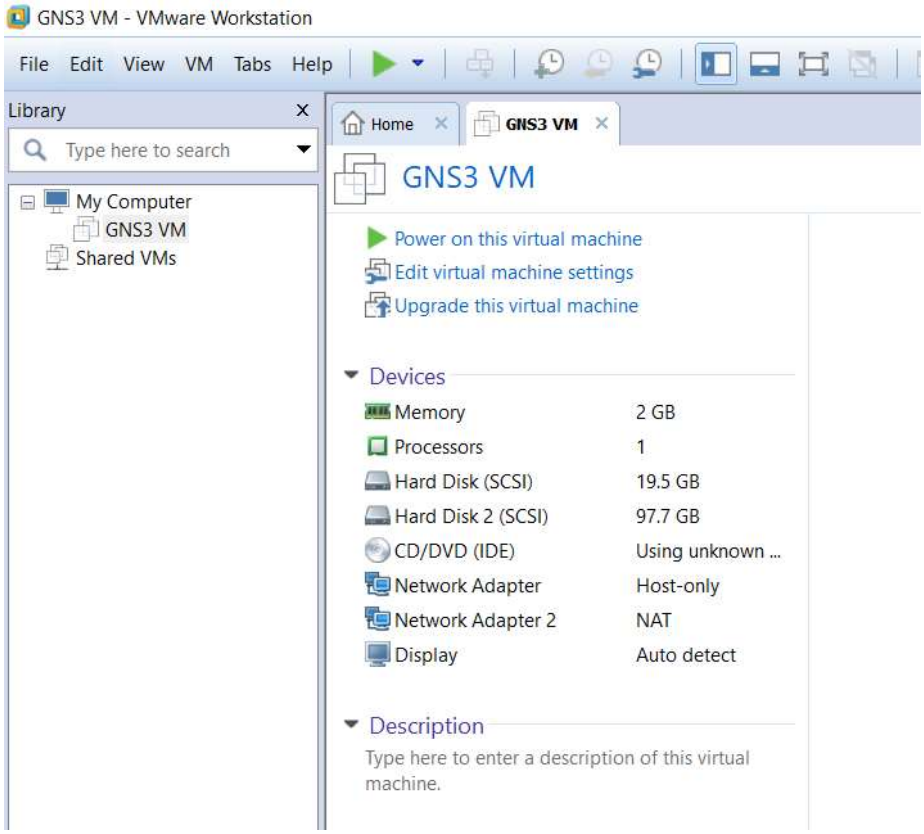


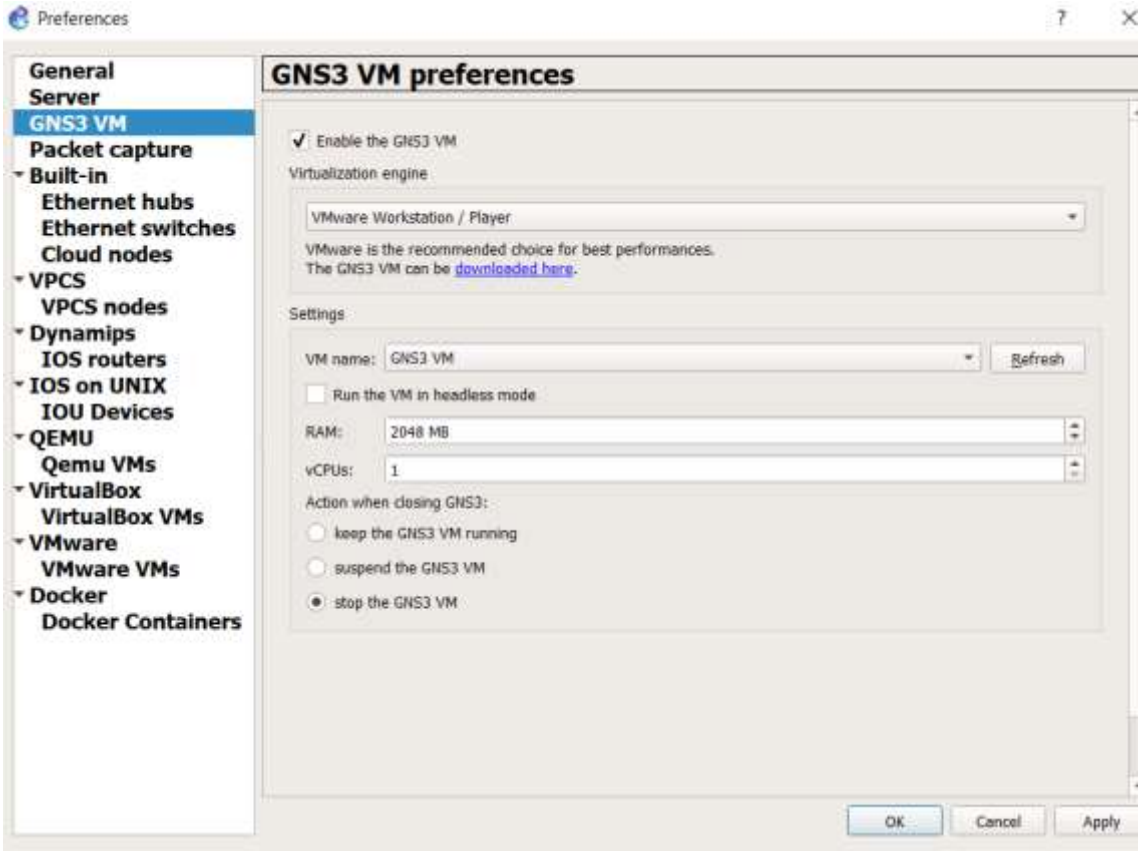


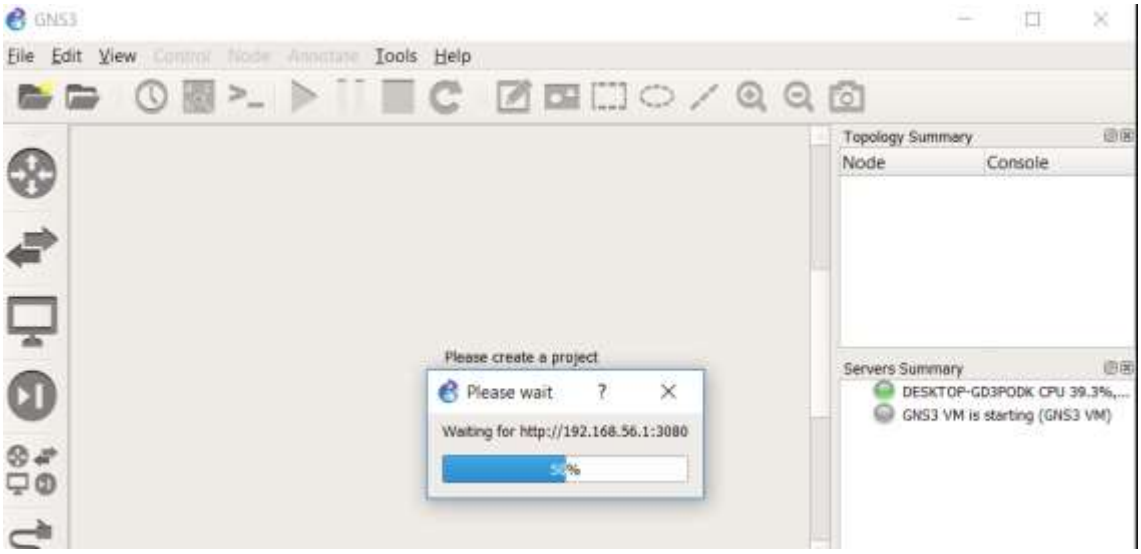


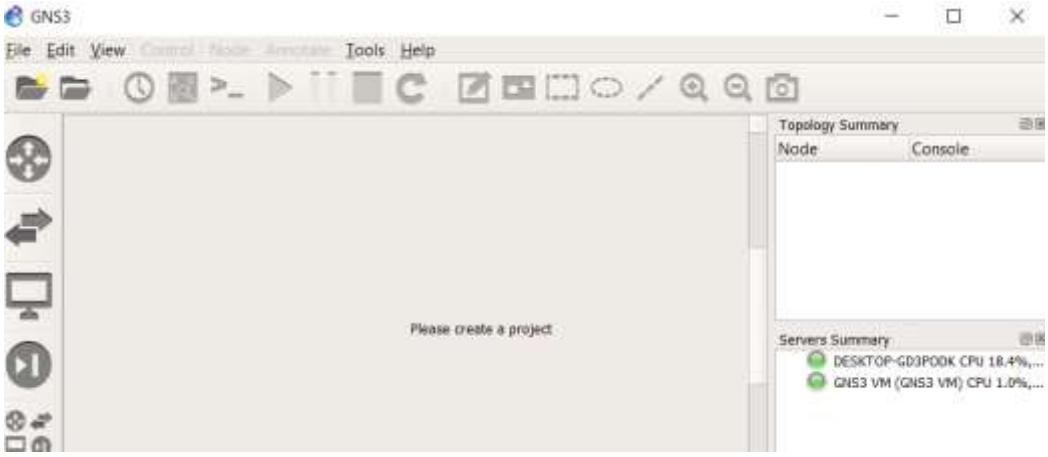
ANEXO B. Instalación de VM de GNS3 en VM Ware

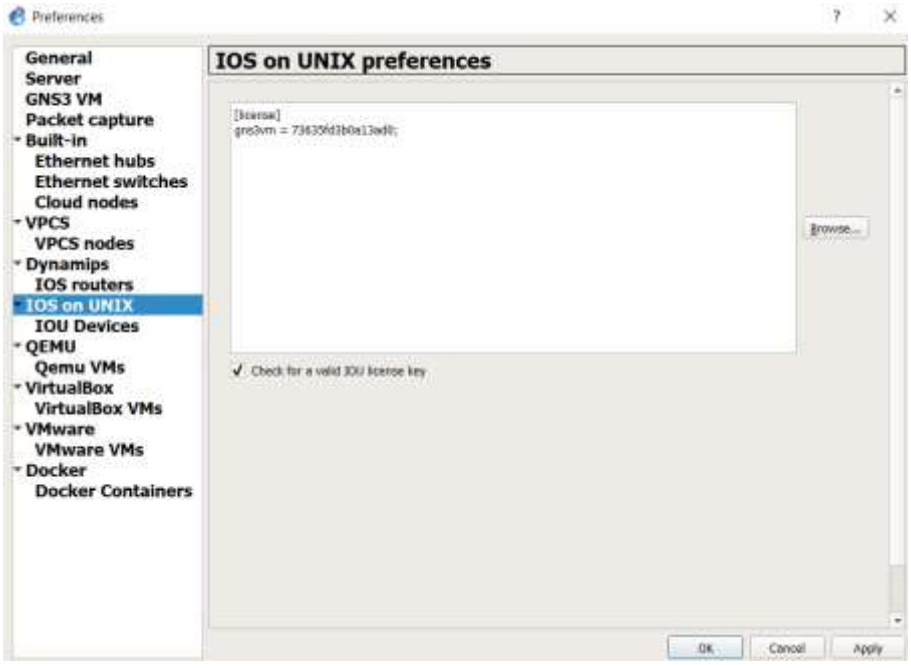



GNS3 VM - VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
GNS3 VM
Shared VMs
Home
GNS3 VM
GNS3 VM
Power on this virtual machine
Edit virtual machine settings
Upgrade this virtual machine
Devices
Memory 2 GB
Processors 1
Hard Disk (SCSI) 19.5 GB
Hard Disk 2 (SCSI) 97.7 GB
CD/DVD (IDE) Using unknown ...
Network Adapter Host-only
Network Adapter 2 NAT
Display Auto detect
Description
Type here to enter a description of this virtual machine.

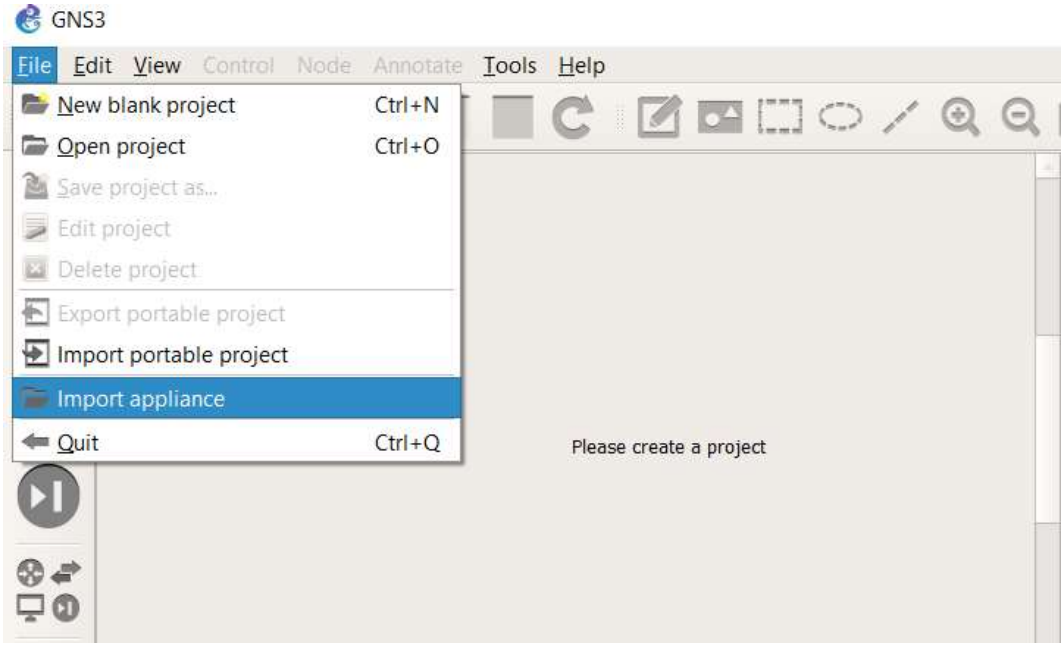

Preferences
General
Server
GNS3 VM
Packet capture
Built-in
Ethernet hubs
Ethernet switches
Cloud nodes
VPCS
VPCS nodes
Dynamips
IOS routers
IOS on UNIX
IOU Devices
QEMU
Qemu VMs
VirtualBox
VirtualBox VMs
VMware
VMware VMs
Docker
Docker Containers
GNS3 VM preferences
Enable the GNS3 VM
Virtualization engine
VMware Workstation / Player
VMware is the recommended choice for best performances.
The GNS3 VM can be downloaded here.
Settings
VM name: GNS3 VM
Refresh
Run the VM in headless mode
RAM: 2048 MB
vCPUs: 1
Action when closing GNS3:
keep the GNS3 VM running
suspend the GNS3 VM
stop the GNS3 VM
OK
Cancel
Apply

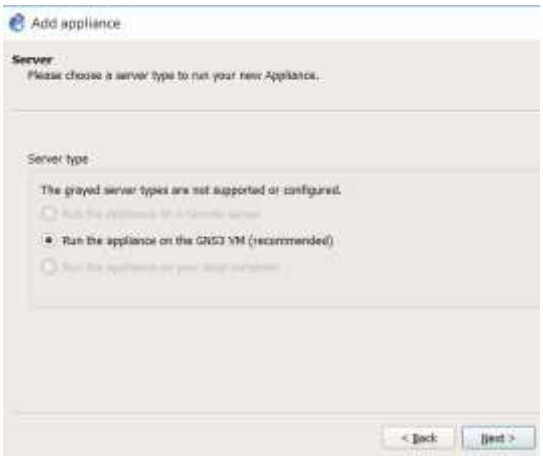

GNS3
File Edit View Control Node Annotate Tools Help
Topology Summary
Node Console
Please create a project
Please wait
Waiting for http://192.168.56.1:3080
50%
Servers Summary
DESKTOP-GD3PODK CPU 39.3%,...
GNS3 VM is starting (GNS3 VM)

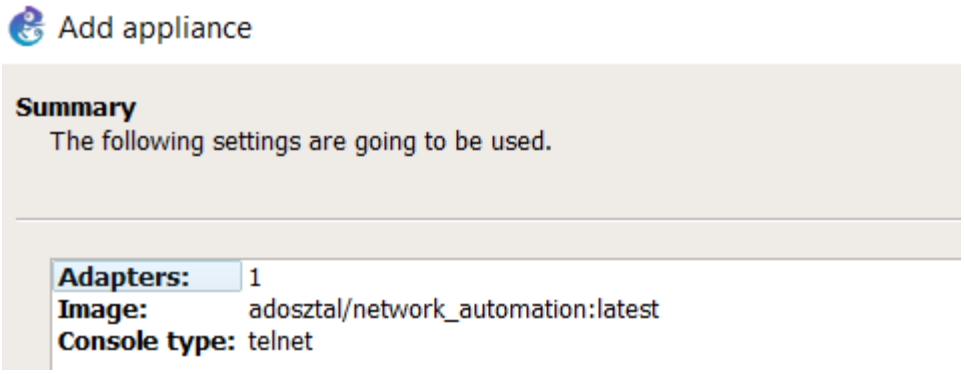

GNS3
File Edit View Control Node Annotate Tools Help
Topology Summary
Node Console
Please create a project
Servers Summary
DESKTOP-GD3PODK CPU 18.4%,...
GNS3 VM (GNS3 VM) CPU 1.0%,...


Preferences
General
Server
GNS3 VM
Packet capture
Built-in
Ethernet hubs
Ethernet switches
Cloud nodes
VPCS
VPCS nodes
Dynamips
IOS routers
IOS on UNIX
IOU Devices
QEMU
Qemu VMs
VirtualBox
VirtualBox VMs
VMware
VMware VMs
Docker
Docker Containers
IOS on UNIX preferences
[license]
gns3vm = 73635fd1b0a13ad0;
Browse...
Check for a valid IOU license key
OK
Cancel
Apply

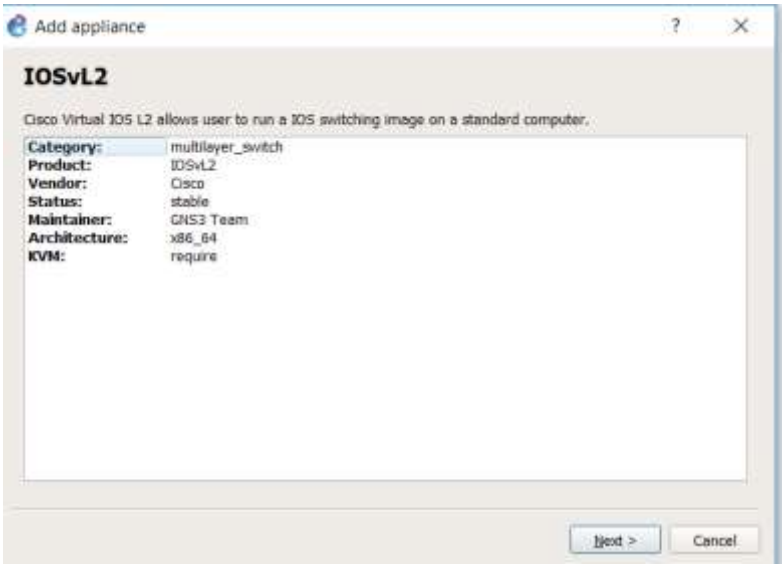
ANEXO C. Instalación de Ansible

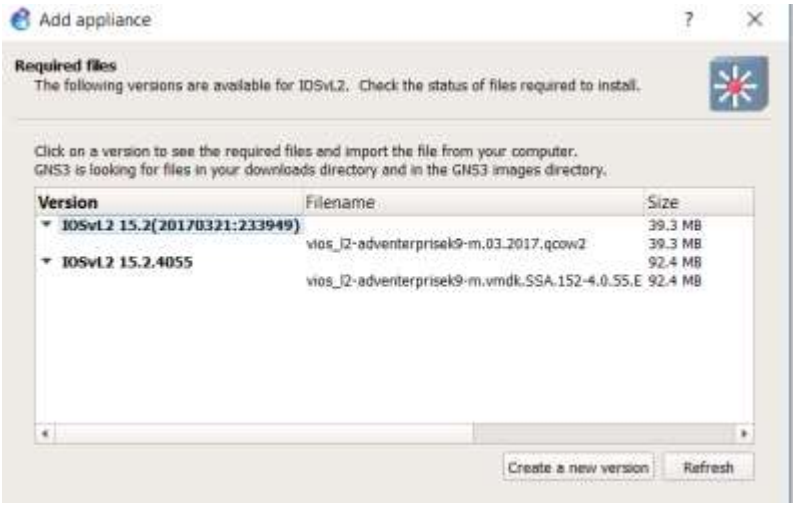


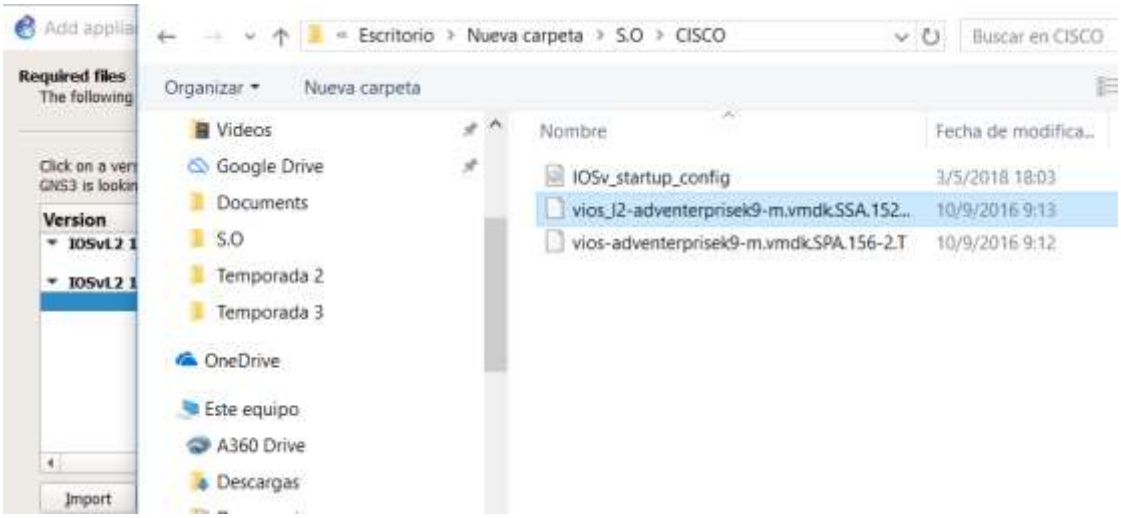


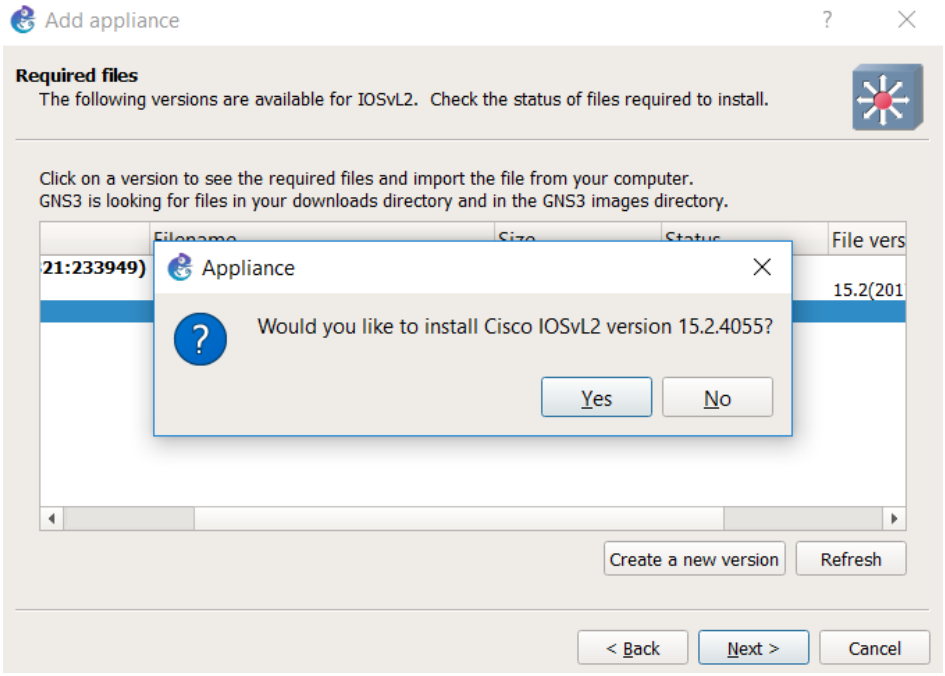


ANEXO D. Instalación de CISCO

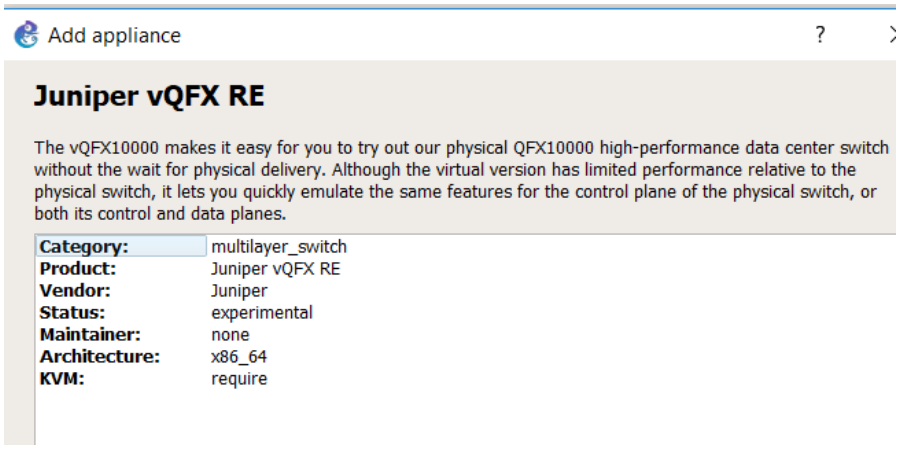


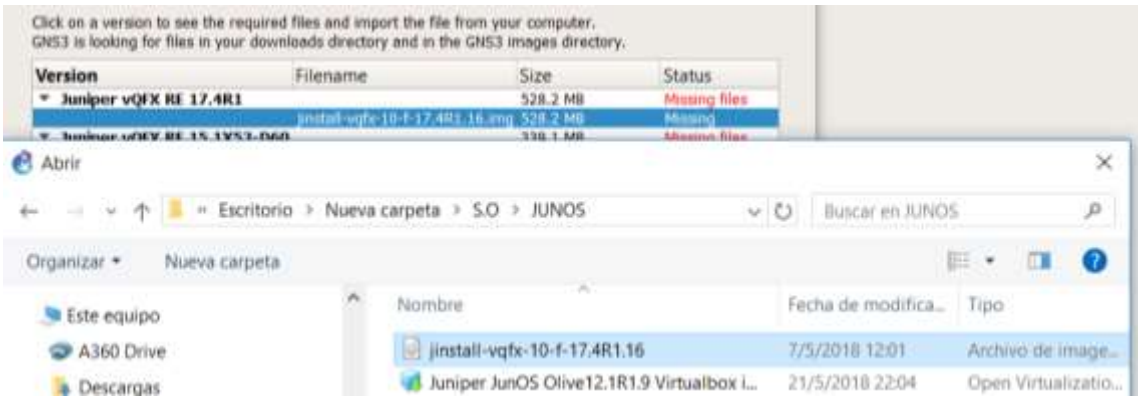


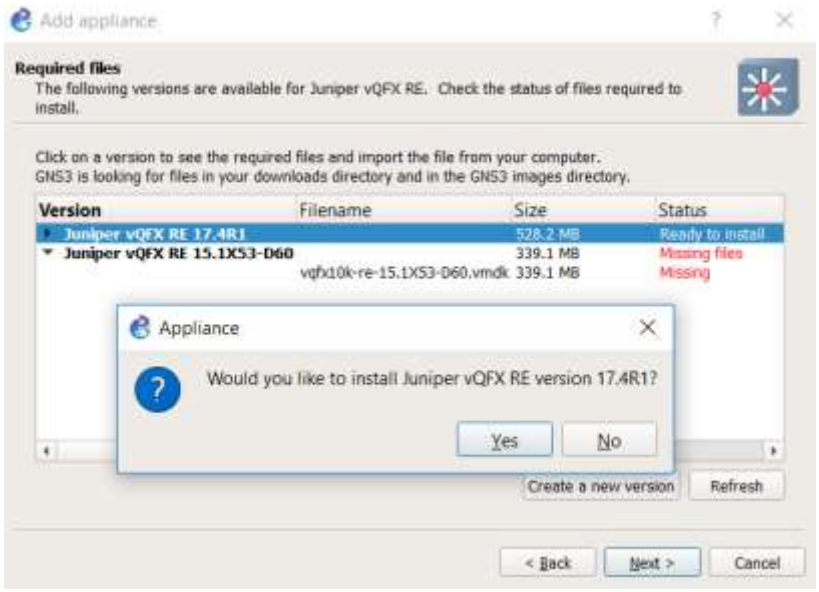




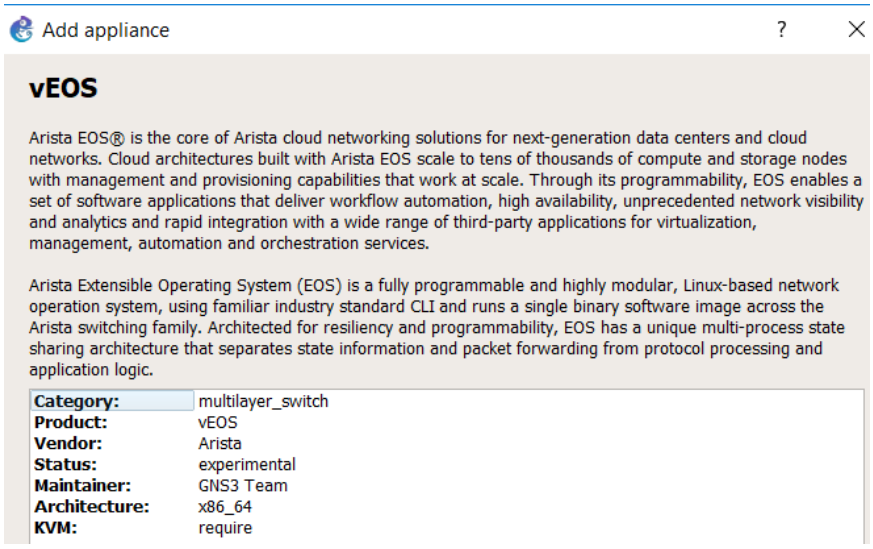
ANEXO E. Instalación de JUNIPER

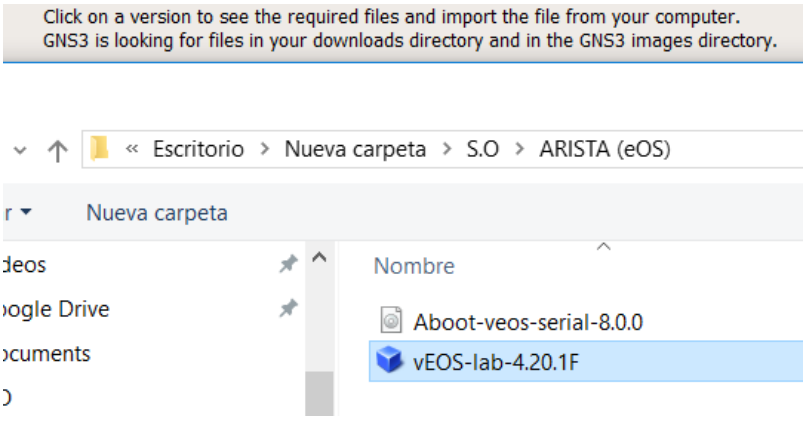


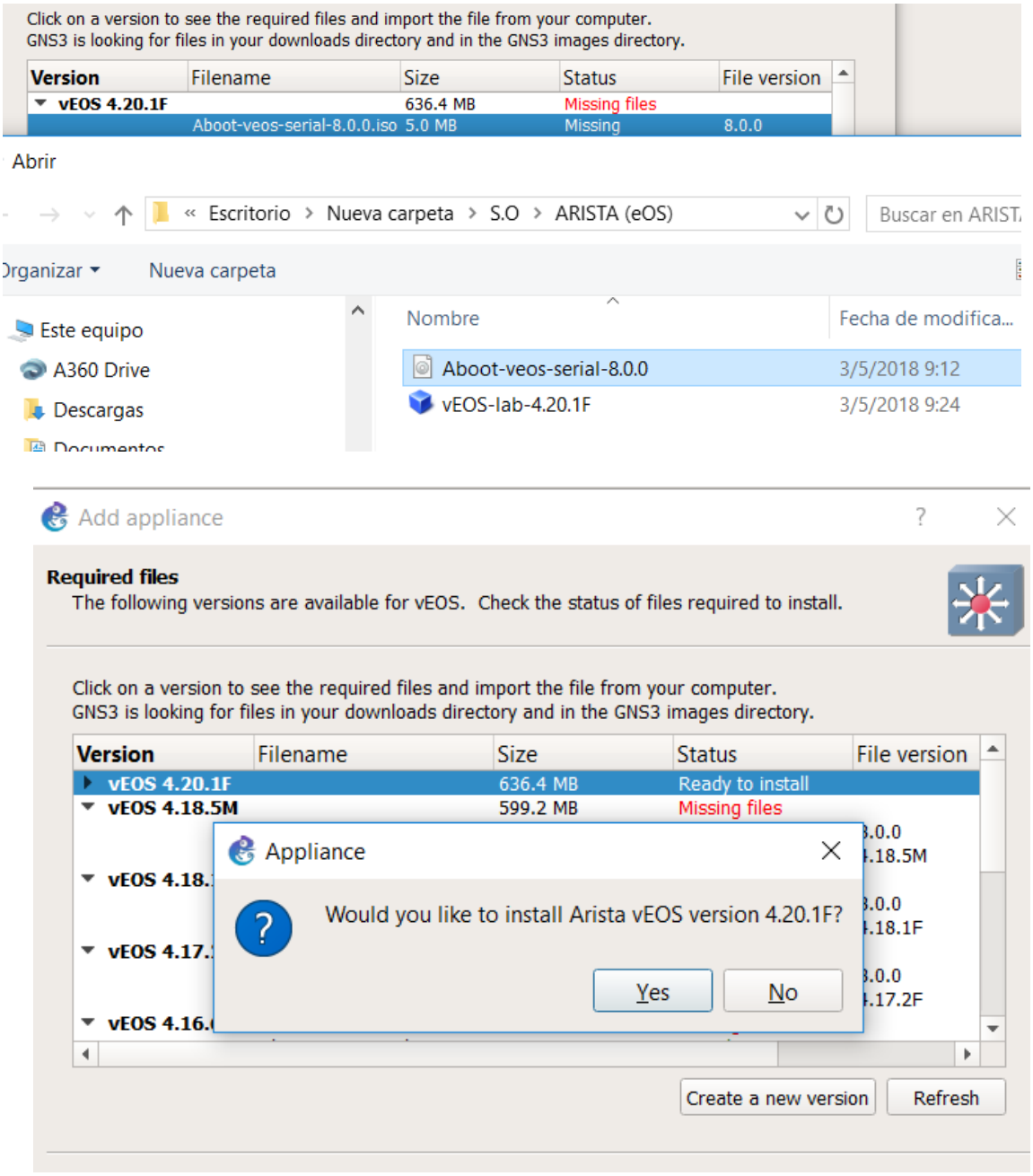




ANEXO F. Instalación de ARISTA







ANEXO G. Playbooks para los routers ce_acme1, P1 y pe_dublin

---

hostname: ce_acme1
domain_name: gns3prueba.com

celoopback:
  address: 172.1.0.111
  mask: 255.255.255.255

```
interfaces:


  1/1.100:
    alias: connection pe_dublin
    vlan: 100
    address: 192.168.100.2
    mask: 255.255.255.252


bgp:
  asn: 65111
  neighbor:
    - {address: 192.168.100.1, remote_as: 1}


---

hostname: P1
domain_name: gns3prueba.com

peloopback:
  address: 10.255.0.100
  mask: 255.255.255.255


interfaces:
  1/1.122:
    alias: connection p2
    vlan: 122
    address: 10.0.122.1
    mask: 255.255.255.252

  1/2.12:
    alias: connection pe_dublin
    vlan: 12
    address: 10.0.255.2
    mask: 255.255.255.252


  1/3.34:
    alias: connection pe_paris
    vlan: 34
    address: 10.0.254.2
    mask: 255.255.255.252



ospf:
   proc: 1
```

```
   ldp: mpls ldp autoconfig
   networks:
    - {network: 0.0.0.0, mask: 255.255.255.255, area: 0}


bgp:
  asn: 1
  neighbor:
    - {address: 10.255.0.1, remote_as: 1}
    - {address: 10.255.0.2, remote_as: 1}
    - {address: 10.255.0.3, remote_as: 1}
    - {address: 10.255.0.4, remote_as: 1}
  ups:
    - {address: 10.255.0.1, loopback: Loopback0}
    - {address: 10.255.0.2, loopback: Loopback0}
    - {address: 10.255.0.3, loopback: Loopback0}
    - {address: 10.255.0.4, loopback: Loopback0}

  vpnv4:
    - {address: 10.255.0.1}
    - {address: 10.255.0.2}
    - {address: 10.255.0.3}
    - {address: 10.255.0.4}

rrc:
  client:
    - {address: 10.255.0.1, policy: route-reflector-client}
    - {address: 10.255.0.2, policy: route-reflector-client}
    - {address: 10.255.0.3, policy: route-reflector-client}
    - {address: 10.255.0.4, policy: route-reflector-client}



---

hostname: pe_dublin
domain_name: gns3prueba.com

peloopback:
  address: 10.255.0.1
  mask: 255.255.255.255


customers:
  acme:
    rd: 10.255.0.1:10
    rt: "10:10"


  banco:
    rd: 10.255.0.1:30
```

```
    rt: "10:30"


  corpnet:
    rd: 10.255.0.1:20
    rt: "10:20"


  metrolink:
    rd: 10.255.0.1:40
    rt: "10:40"


interfaces:
  1/1.12:
    alias: connection p1
    vlan: 12
    address: '10.0.255.1'
    mask: '255.255.255.252'

  1/2.100:
    alias: connection ce_acme
    vlan: 100
    address: '192.168.100.1'
    mask: '255.255.255.252'
    intvrf: 'acme'


ospf:
   proc: 1
   ldp: mpls ldp autoconfig
   networks:
    - {network: 0.0.0.0, mask: 255.255.255.255, area: 0}


bgp:
  asn: 1
  neighbor:
    - {address: 10.255.0.100, remote_as: 1}
    - {address: 10.255.0.200, remote_as: 1}

  ups:
    - {address: 10.255.0.100, loopback: Loopback0}
    - {address: 10.255.0.200, loopback: Loopback0}

  ipv4:
    - {ce: 192.168.100.2, remote_as: 65111, cvrf: acme}

  vpnv4:
    - {address: 10.255.0.100, remote_as: 1}
    - {address: 10.255.0.200, remote_as: 1}
```

```
policy:
  outbound:
    - {ce: 192.168.100.2, asover: as-override}
```

ANEXO H. Plantilla para aplicar la configuración en la topología 5

```
<configuration>
    <system>
        <host-name>{{ host_name }}</host-name>
        <services>
            <ssh>
                <root-login>allow</root-login>
            </ssh>
            <netconf>
                <ssh>
                </ssh>
            </netconf>
            <web-management>
                <http>
                    <interface>ge-0/0/0.0</interface>
                </http>
            </web-management>
        </services>
    </system>
    <interfaces>
        {% for i in host.interfaces %}
        <interface replace="replace">
            <name>{{ i.interface }}</name>
            <unit>
                <name>0</name>
                <description>{{ i.description }}</description>
                <family>
                    <inet>
                        <address>
                            <name>{{ i.local_ip }}</name>
                        </address>
                    </inet>
                    <iso>
                    </iso>
                </family>
            </unit>
        </interface>
        {% endfor %}
        <interface replace="replace">
            <name>lo0</name>
            <unit>
                <name>0</name>
                <description>peering loopback</description>
```

```
            <family>
               <inet>
                  <address>
                     <name>{{ host.loopback.local_ip }}</name>
                  </address>
               </inet>
               <iso>
                  <address>
                     <name>{{ host.loopback.iso_net }}</name>
                  </address>
               </iso>
            </family>
         </unit>
      </interface>
   </interfaces>
   {% if host.bgp.enabled == true %}
   <routing-options>
      <autonomous-system replace="replace">
         <as-number>{{ host.bgp.as }}</as-number>
      </autonomous-system>
   </routing-options>
   {% endif %}
   <protocols>
      {% if host.isis.enabled == true %}
      <isis replace="replace">
         {% if host.isis.overload == true %}
         <overload>
         </overload>
         {% endif %}
         <level>
            <name>2</name>
            <disable/>
         </level>
         {% for i in host.interfaces %}
         <interface>
            <name>{{ i.interface }}</name>
         </interface>
         {% endfor %}
         <interface>
            <name>lo0.0</name>
         </interface>
      </isis>
      {% endif %}
      {% if host.bgp.enabled == true %}
      <bgp replace="replace">
         {% if not host.bgp.peers.external|e %}
         {% for k,v in host.bgp.peers.external.iteritems() %}
         <group>
            <name>eBGP-{{ k }}</name>
            <family>
               <inet>
```

```
                <unicast>
                </unicast>
              </inet>
            </family>
            <peer-as>{{ k }}</peer-as>
            {% for p in v %}
            <neighbor>
              <name>{{ p.ip }}</name>
            </neighbor>
            {% endfor %}
          {% endfor %}
          </group>
          {% endif %}
          <group>
            <name>iBGP</name>
            <type>internal</type>
            <local-address>{{ host.loopback.local_ip[:-3] }}</local-address>
            <family>
              <inet>
                <unicast>
                  <add-path>
                    <receive/>
                    <send>
                      <path-count>6</path-count>
                    </send>
                  </add-path>
                </unicast>
              </inet>
            </family>
            {% if host.bgp.peers.internal %}
            {% for p in host.bgp.peers.internal %}
            <neighbor>
              <name>{{ p.ip }}</name>
            </neighbor>
            {% endfor %}
            {% endif %}
          </group>
        </bgp>
        {% endif %}
        <lldp>
          <interface>
            <name>all</name>
          </interface>
        </lldp>
      </protocols>
    </configuration>
```

ANEXO I. Cambios realizados en el switch RR1 de la topología 5

```
    em1 {
        unit 0 {
            family inet {
                address 169.254.0.2/24;
                address 192.168.122.67/24;
            }
        }
    }
    em2 {
        unit 0 {
            description "to P1";
            family inet {
                address 10.1.10.17/31;
            }
            family iso;
        }
    }
    em3 {
        unit 0 {
            description "to P2";
            family inet {
                address 10.1.10.19/31;
            }
            family iso;
        }
    }
    lo0 {
        unit 0 {
            description "peering loopback";
            family inet {
                address 172.16.0.201/32;
            }
            family iso {
                address 49.0000.0000.cccc.0005.00;
            }
        }
    }
forwarding-options {
    storm-control-profiles default {
        all;
    }
routing-options {
    autonomous-system 65000;
protocols {
    bgp {
        group iBGP {
            type internal;
            local-address 172.16.0.201;
```

```
            family inet {
                unicast {
                    add-path {
                        receive;
                        send {
                            path-count 6;
                        }
                    }
                }
            }
            cluster 172.16.0.201;
            neighbor 172.16.0.11;
            neighbor 172.16.0.22;
            neighbor 172.16.0.33;
            neighbor 172.16.0.44;
        }
    }
    isis {
        overload;
        level 2 disable;
        interface em2.0;
        interface em3.0;
        interface lo0.0;
    }
    lldp {
        interface all;
    }
    igmp-snooping {
        vlan default;
    }
vlans {
    default {
        vlan-id 1;
    }

{master:0}
```