Pontificia Universidad Católica del Ecuador

FACULTAD DE INGENIERÍA
COORDINACIÓN DE POSGRADO

PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

Trabajo de Titulación como requisito previo para la obtención del título de Magíster en Tecnologías de la Información Mención en Gestión y Administración de TI

TEMA:

USO DE GIT COMO HERRAMIENTA DE VERSIONAMIENTO DE REDES PROGRAMÁTICAS

Autor: JUAN CARLOS GONZALEZ ORTIZ

Director: Dr. GUSTAVO DAVID SALAZAR CHACON. (PHD)

Quito, Septiembre 2022

PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

DECLARACIÓN Y AUTORIZACIÓN

Yo, **JUAN CARLOS GONZALEZ ORTIZ,** con C.I 1104136625**,** autor del trabajo de graduación intitulado: **"USO DE GIT COMO HERRAMIENTA DE VERSIONAMIENTO DE REDES PROGRAMÁTICAS"**, previa a la obtención del grado académico de **MAGISTER EN TECNOLOGÍAS DE LA INFORMACIÓN MENCIÓN GESTIÓN Y ADMINISTRACIÓN DE TI** en la Facultad de **INGENIERÍA**:

1.- Declaro tener pleno conocimiento de la obligación que tiene la Pontificia Universidad Católica del Ecuador, de conformidad con el artículo 144 de la Ley Orgánica de Educación Superior, de entregar a la SENESCYT en formato digital una copia del referido trabajo de graduación para que sea integrado al Sistema Nacional de Información de la Educación Superior del Ecuador para su difusión pública respetando los derechos de autor.
2.- Autorizo a la Pontificia Universidad Católica del Ecuador a difundir a través de sitio web de la Biblioteca de la PUCE el referido trabajo de graduación, respetando las políticas de propiedad intelectual de Universidad.

Quito, Septiembre 2022

_____
JUAN CARLOS GONZALEZ ORTIZ
C.I. 1104136625

Av. 12 de octubre 1076 y Ramón Roca
Apartado postal 17-01-2184
Telf.: (593) 2 2991647
Quito – Ecuador www.puce.edu.ec

APROBACIÓN DEL TUTOR

En mi carácter de Director (a) – Tutor (a) del Trabajo de Posgrado Titulado: "USO DE GIT COMO HERRAMIENTA DE VERSIONAMIENTO DE REDES PROGRAMÁTICAS", presentado por el maestrante JUAN CARLOS GONZALEZ ORTIZ, titular de la Cédula de Identidad N.º 1104136625 para optar al Grado de Magíster en Tecnologías de la Información mención en Gestión y Administración de TI, considero que dicho Trabajo de Investigación reúne los requisitos y méritos suficientes para ser sometido a la evaluación por parte de los Lectores – Evaluadores que se designen para tal fin por parte de las autoridades de la Facultad de Ciencias de la Educación.

En la ciudad de Quito, a los 15 días de Septiembre del 2022

Firmado electrónicamente por:
GUSTAVO DAVID SALAZAR CHACON

GUSTAVO DAVID SALAZAR CHACÓN C.I. 1716104797

gsalazar787@puce.edu.ec

NOTA:

Se comunica que en el servicio de análisis Turnitin, el referido trabajo de titulación alcanzó el siguiente resultado: 6% índice de similitud con otras fuentes.

Av. 12 de octubre 1076 y Ramón Roca
Apartado postal 17-01-2184
Telf.: (593) 2 2991647
Quito – Ecuador www.puce.edu.ec

Visualizador de documentos

Turnitin Informe de Originalidad

Procesado el: 15-sept.-2022 15:03 -05
Identificador: 1900708320
Número de palabras: 16016
Entregado: 1

Tesis-Final-MaestríaTI Por Juan Carlos González Ortíz

Índice de similitud	Similitud según fuente	
6%	Internet Sources:	6%
	Publicaciones:	1%
	Trabajos del estudiante:	2%

incluir citas incluir bibliografía excluyendo las coincidencias < 5 de las palabras modo:
ver informe en vista quickview (vista clásica) Change mode imprimir actualizar descargar

1% match (Internet desde 08-sept.-2022)
https://www.arubanetworks.com/latam/faq/que-es-la-arquitectura-spine-leaf/

<1% match ()
Espinoza López, Juan Luis. "Property Management System Web: análisis y ampliación de la implementación del PMS Web de Tesipro", 2017

<1% match ()
Capcha Carhajal, Hernán William. "Informe de suficiencia y competencias desarrolladas en el proceso de fabricación de software empresarial CONTASIS - CONTASIC SA", 'Baishideng Publishing Group Inc.', 2021

<1% match ()
Mahiques Álamo, Jordi. "Explotación de servicios gratuitos Cloud Computing", 'Universitat Politecnica de Valencia', 2015

<1% match (Internet desde 08-sept.-2022)
https://empresas.blogthinkbig.com/netdevops-nuevo-ingeniero-de-red/

<1% match (Internet desde 05-mar.-2022)
https://cambiodigital-ol.com/2022/01/que-es-la-infraestructura-como-codigo/

<1% match (Internet desde 13-oct.-2021)
https://programarfacil.com/blog/arduino-blog/git-y-github/

<1% match ()
López Chamorro, Ignacio. "Gestión de reservas online del servicio de deportes del campus de la Universidad Francisco de Vitoria.", Universidad Francisco de Vitoria, 2015

<1% match (Internet desde 13-abr.-2022)
https://www.juniper.net/documentation/mx/es/software/junos/evpn-vxlan/topics/concept/vxlan-evpn-integration-overview.html

<1% match (publicaciones)
Delgado-Hernández David Joaquín, Romero-Ancira Liliana. "Satisfacción de las necesidades del cliente en el sector vivienda: el cas del Valle de Toluca", Ingeniería, Investigación y Tecnología, 2013

<1% match (trabajos de los estudiantes desde 09-dic.-2019)
Submitted to UNIV DE LAS AMERICAS on 2019-12-09

<1% match (Internet desde 30-mar.-2022)
https://www.coursehero.com/file/76227768/Jhernandezpatpdf/

<1% match (Internet desde 09-may.-2022)
https://www.coursehero.com/file/145745983/Conclusi%C3%B3ndocx/

<1% match (Internet desde 29-sept.-2021)
https://www.coursehero.com/file/87486749/ITP342-W15-APIspdf/

<1% match (Internet desde 12-jun.-2021)
https://1library.co/document/g2ml8ejy-prototipo-sistema-control-automatico-temperatura-humedad-invernadero.html

<1% match (Internet desde 07-mar.-2022)
https://1library.co/document/god7rm5z-07-8384.html

<1% match (Internet desde 20-sept.-2021)
https://www.json.org/json-es

<1% match (Internet desde 15-ene.-2020)
https://issuu.com/cavieres/docs/actual_ti

<1% match (Internet desde 04-ago.-2019)
https://issuu.com/acemi/docs/conexxion-revista4

<1% match (trabajos de los estudiantes desde 20-feb.-2021)
Submitted to Universidad Carlos III de Madrid on 2021-02-20

<1% match (Internet desde 02-jul.-2020)
https://upcommons.upc.edu/bitstream/handle/2117/191946/Memoria%20TFM%20Jorge%20Sanango.pdf?isAllowed=y&sequence=1

<1% match (Internet desde 22-may.-2014)

Av. 12 de octubre 1076 y Ramón Roca
Apartado postal 17-01-2184
Telf.: (593) 2 2991647
Quito – Ecuador www.puce.edu.ec

DECLARACIÓN DE AUTENTICIDAD Y RESPONSABILIDAD

Yo, Juan Carlos Gonzalez Ortiz portador de la cédula de ciudadanía No.1104136625, declaro bajo juramento que la presente investigación es de total responsabilidad del autor, y que se ha respetado las diferentes fuentes de información realizando las citas correspondientes. Esta investigación no contiene plagio alguno y es resultado de un trabajo serio desarrollado en su totalidad por mi persona.

JUAN CARLOS GONZALEZ ORTIZ
C.I. 1104136625

DEDICATORIA

A mi amada madre Lupita, a mis hermanos: Carlos, Joan, Jimmy y Rusbel, este logro es dedicado a todos ustedes, porque siempre han creído en mí.

AGRADECIMIENTOS

Al terminar este objetivo tan importante, que hace un año me planteé como un reto profesional para conmigo mismo, no me queda nada más que agradecer a Dios y a la Virgen del Cisne por permitirme poder cumplirlo, por todas las veces en las que en la oración encontré la fortaleza y la voluntad para no darme por vencido.

Agradezco profundamente a mi madre Lupita, en toda mi vida tu guía basada en el amor, comprensión y sabiduría me han servido para ir por el camino del bien y ser una buena persona. Gracias, porque todo lo que me has exigido y enseñado siempre ha sido con el ejemplo, siempre me dijiste que la única manera de pagarte todo lo que has hecho por mí era estudiando. ¡Finalmente soy Magister!

A mis cinco hermanos menores: Carlos, Joan, Jimmy y Rusbel, les estaré eternamente agradecido, porque me alentaron a que siga la maestría y siempre estuvieron ahí para mí.

Quiero agradecer a mis jefes y amigos del trabajo especialmente a Mónica Estrella, Edwin Herrera, Sixto Rueda y Alejandro Masabanda, porque siempre me han motivado a que siga estudiando y preparándome, ustedes con su ética, trabajo y liderazgo, me inspiran.

Finalmente, un agradecimiento a la Pontificia Universidad Católica del Ecuador por haberme aceptado al programa de maestría y en especial a mi director de tesis, Dr. Gustavo Salazar por confiarme el desarrollo del presente trabajo de titulación y guiarme en su ejecución. En pregrado nunca me gustó nada referente a las redes, sin embargo, disfruté y aprendí con cada una de sus clases en la maestría. |Gracias por su pasión a la hora de enseñar!

Av. 12 de octubre 1076 y Ramón Roca
Apartado postal 17-01-2184
Telf.: (593) 2 2991647
Quito – Ecuador www.puce.edu.ec

ÍNDICE DE CONTENIDOS

INTRODUCCIÓN 14
CAPÍTULO I: PLANTEAMIENTO DEL PROBLEMA 15
1.1. Formulación del problema 15
1.2. Objetivos de la Investigación 16
Objetivo General 16
Objetivos Específicos 16
1.3. Justificación de la Investigación 17
CAPÍTULO II: FUNDAMENTACIÓN TEÓRICA 18
2.1. Infraestructura como código 18
2.2. API 20
2.3. Formatos de Datos 21
2.3.1. JSON 21
2.3.2. XML 22
2.3.3. YAML 23
2.4. Python 23
2.5. Google Cloud 24
2.6. VXLAN 25
2.7. Arquitectura Spine-Leaf 27
2.8. NetDevOps 28
2.8.1. Pipelines CI-CD 30
2.8.2. Herramientas de NetDevOps 31
CAPÍTULO III: METODOLOGÍA 38
3.1. Software EVE-NG 38
3.2. Gitlab CI-CD 38
3.3. Topología de la Red 38
3.4. Integración con Git y GitLab CI-CD 40
CAPÍTULO IV: EMULACION 42
4.1. Configuración de VXLAN 42
4.2. Instalación de Git 51
4.3. Configuración de Gitlab CI/CD 53
CAPÍTULO V: ANÁLISIS DE RESULTADOS 56
CONCLUSIONES Y RECOMENDACIONES 64

Av. 12 de octubre 1076 y Ramón Roca
Apartado postal 17-01-2184
Telf.: (593) 2 2991647
Quito – Ecuador www.puce.edu.ec

ESTUDIOS FUTUROS 66
REFERENCIAS 67
ANEXOS 69
ANEXO 1 – INSTALACIÓN DE LA MÁQUINA VIRTUAL EN GOOGLE CLOUD ... 70
ANEXO 2 – INSTALACIÓN DE EVE-NG 73
ANEXO 3 - CONFIGURACIÓN DE EVE-NG 76

ÍNDICE DE TABLAS

Tabla 1. Direccionamiento IP de la topología SPINE-LEAF 39

ÍNDICE DE GRÁFICOS

Figura 1. Flujo de trabajo de la IaC 18
Figura 2. Interfaz de comunicación de una API 20
Figura 3. Formato JSON 22
Figura 4. Formato XML 23
Figura 5. Formato YAML 23
Figura 6. Visión Global de Google Cloud 25
Figura 7. Formato de Encapsulación VXLAN 26
Figura 8. Capas de conmutación en Redes Tradicionales 27
Figura 9. Arquitectura Spine-Leaf 28
Figura 10. Fases de NetDevOps 29
Figura 11. Cultura del Cambio con NetDevOps 29
Figura 12. Flujo CI/CD en un repositorio 31
Figura 13. Herramientas de DevOps 31
Figura 14. Ecosistema NetDevOps de CISCO 32
Figura 15. Almacenamiento de Datos en un VCS Tradicional 32
Figura 16. Almacenamiento de Datos como instantáneas en Git 33
Figura 17. Secciones principales de un proyecto en Git 33
Figura 18. Workflow en las etapas del ciclo de vida DevOps 35
Figura 19. Logo de EVE-NG 35
Figura 20. Arquitectura de Ansible 36
Figura 21. Topología PoC-VXLAN 39
Figura 22. Directorio de Ansible dentro de Ubuntu 40
Figura 23. Repositorio remoto en Gitlab 40
Figura 25. Diagrama de secuencia del escenario a emular 41
Figura 26. Arquitectura SPINE-LEAF emulada 42
Figura 27. Configuración IP del nodo Linux (red administrable) 43
Figura 28. Configuración de direccionamiento en nodos Cumulus Linux (red administrable) 43
Figura 29. Prueba de ping desde Ansible hacia LEAF-1 y SPINE 43
Figura 30. Generación de llaves SSH en Equipos LEAF-1, LEAF-2, SPINE 44
Figura 31. Configuración del archivo hosts 45
Figura 32. Configuración del archivo ansible.cfg 45
Figura 33. Activación de la relación de llaves SSH entre los nodos Cumulus y Ansible 46
Figura 34. Ejecución del comando “ansible –m ping all” 46
Figura 35. Configuración de direccionamiento IP 46
Figura 36. Configuración de enrutamiento Underlay (OSPF de una sola área) 47
Figura 37. Configuración de VXLAN 47
Figura 38. Configuración de VLAN 10 en los LEAF 47
Figura 39. Configuración del Mapeo de VXLAN con VLAN Y VTEP (VNID 1010) 48
Figura 40. Edición del fichero daemons para activar OSPF 48
Figura 41. Activación de OSPF y VXLAN 48
Figura 42. Ejecución del comando “ansible-playbook vxlan.yml” 49
Figura 43. Reporte de ejecución del fichero playbook vxlan.yml 49
Figura 44. Configuración de direccionamiento ip en las PC y prueba de ping 49
Figura 45. Tabla de enrutamiento del nodo SPINE 50
Figura 46. Tabla de aprendizaje MAC en LEAF-1 50
Figura 47. Captura de tráfico con Wireshark 50
Figura 48. Instalación de Git 51
Figura 49. Llave SSH dentro del nodo Linux 51
Figura 50. Llave SSH dentro de GitLab 51
Figura 51. Verificación de funcionamiento SSH 52
Figura 52. Directorio de instalación de Ansible 52
Figura 53. Subida a Git del directorio Ansible 52
Figura 54. Subida a repositorio remoto Fuente: Autor 53

Figura 55. Repositorio remoto en GitLab con la NaC de VXLAN 53
Figura 56. Instalación de Gitlab Runner en nodo Linux 53
Figura 57. Obtención del token del Runner 54
Figura 58. Registro del GitLab Runner 54
Figura 59. Verificación de Runner activo dentro de GitLab 55
Figura 60. Creación e integración de la rama “pruebas” dentro del repositorio Git 56
Figura 61. Subida a repositorio remoto de la rama de pruebas 56
Figura 62. Historial de commits del repositorio 57
Figura 63. Topología VXLAN Spine-Leaf con dos VLAN 57
Figura 64. Creación de VLAN 10 Y VLAN 20 58
Figura 65. Mapeo de VNI 2020 en la red Overlay 58
Figura 67. Subida a repositorio de pruebas remotos de los cambios 59
Figura 68. Repositorio remoto en Gitlab 59
Figura 69. Fichero “. gitlab-ci.yml” 60
Figura 70. Reporte de tiempo de ejecución del pipeline 60
Figura 71. Ejecución del job test 61
Figura 72. Ejecución del job deploy 61
Figura 73. Verificación de ping entre equipos de VLAN 10 y 20 62
Figura 74. Visualización de la configuración en la interface del Leaf-2 62
Figura 75. Captura de paquetes con VNI 1010 en WireShark 63
Figura 76. Captura de paquetes con VNI 2020 en WireShark 63
Figura 77. Interfaz de Compute Engine en Google Cloud 70
Figura 78. Configuración de la máquina virtual parte 1 70
Figura 79. Configuración de la máquina virtual parte 2 71
Figura 80. Configuración de la máquina virtual parte 3 71
Figura 81. Activación de la virtualización anidada 71
Figura 82. Creación e inicialización de la VM 72
Figura 83. Acceso por SSH a la VM 72
Figura 84. Creación de regla en Firewall para permitir tráfico TCP 73
Figura 85. Instalación de EVE-NG 74
Figura 86. Finalización de la instalación de EVE-NG 74
Figura 87. Wizard de configuración de EVE-NG 75
Figura 88. Acceso http a EVE-NG 75
Figura 89. Directorio qemu de la emulación 76
Figura 90. Edición del fichero /etc/network/interfaces 77
Figura 91. Activación de IPV4 Forwarding 77

PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR
FACULTAD DE INGENIERÍA
MAESTRIA EN TECNOLOGÍAS DE LA INFORMACIÓN MENCIÓN GESTIÓN Y ADMINISTRACIÓN DE TI

USO DE GIT COMO HERRAMIENTA DE VERSIONAMIENTO DE REDES PROGRAMÁTICAS

Autor: Juan Carlos Gonzalez Ortiz

Director -Tutor: Dr. Gustavo Salazar Chacón

RESUMEN

El presente trabajo de titulación analiza la factibilidad de implementar el control de versionamiento de código a través de Git en una red moderna programática de topología VXLAN Spine-Leaf, por medio del uso de la API de Ansible como herramienta de Network as Code y en un ecosistema de Open Networking por medio de Cumulus Linux como NOS (Network Operating System). Toda esta infraestructura se emula de manera innovadora en un entorno en la nube con Google Cloud y utilizando el conjunto de prácticas que trae consigo la metodología NetDevops.

Palabras clave: Git, GitLab CI/CD, Ansible, VXLAN, NetDevOps, Google Cloud

PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR
FACULTAD DE INGENIERÍA
MAESTRIA EN TECNOLIGÍAS DE LA INFORMACIÓN MENCIÓN GESTIÓN
Y ADMINISTRACIÓN DE TI

USE OF GIT AS A PROGRAMMATIC NETWORK VERSIONING TOOL

Author: Juan Carlos Gonzalez Ortiz

Director -Tutor: Dr. Gustavo Salazar Chacón

ABSTRACT

This degree work analyzes the feasibility of implementing code versioning control through Git in a modern programmatic network of VXLAN Spine-Leaf topology, through the use of the Ansible API as a Network as Code tool and in an Open Networking ecosystem through Cumulus Linux as NOS (Network Operating System). All this infrastructure is emulated in an innovative way in a cloud environment with Google Cloud and using the set of practices brought by the NetDevops methodology.

Keywords: Git, GitLab CI/CD, Ansible, VXLAN, NetDevOps, Google Cloud

INTRODUCCIÓN

La innovación en las redes de datos no ha ido a la par del ritmo vertiginoso de crecimiento que ha tenido la evolución de la informática, Centros de Datos y aplicaciones en los últimos años, y es que aún es muy común administrar una red de datos con un enfoque tradicional de manejo por consola y con silos departamentales, donde lo que menos quieren los miembros encargados de su administración, es efectuar algún tipo de cambio en la red. Sin embargo, en los últimos años se ha sucedido una revolución en el campo del networking, donde surgen conceptos como programabilidad, automatización, API´s (Application Programming Interfaces), orquestadores de red, IaC (Infraestructure as Code), NetDevOps, etc, que en conjunto permiten optimizar la implementación de las redes bajo una indispensable filosofía de cultura colaborativa.

Por todo lo anteriormente expuesto, este trabajo de titulación indaga y promueve el uso de una de las herramientas más importantes de este nuevo paradigma NetDevOps como lo es Git para el control de versionamiento de la infraestructura. Para ello se trata los siguientes temas:

El capítulo I realiza la formulación del problema, plantea los objetivos de la investigación y la justificación, con lo que se busca dar un contexto en el cual se encuentra el Networking en la actualidad.

El capítulo II hace énfasis en la conceptualización teórica de todas las técnicas y herramientas que se integran a NetDevOps como lo es la Infraestructura como Código, API´s, tipos de formatos de datos, Python, Google Cloud, VXLAN, Git, Gitlab y pipelines CI/CD.

En el capítulo III se detalla la metodología utilizada para la puesta en marcha de esta investigación, a fin de dar una visión global de cómo se va a implementar la PoC (Proof of Concept), misma que va a demostrar la viabilidad y potencialidad del uso de Git como herramienta de versionamiento en una red programática.

El capítulo IV presenta el proceso a seguir para la configuración de la emulación diseñada, esto en un entorno en la nube donde se configura la red VXLAN por medio de Ansible, se instala Git para el versionamiento de la IaC y Gitlab Runner como servidor CI/CD; esto con el objetivo de poner en práctica todos los conceptos anteriormente descritos y así demostrar en un entorno de pruebas emulado la viabilidad de su implementación al lograr una aproximación lo más de cercana al comportamiento que se podría llegar a tener en el entorno de producción.

En el capítulo V se analizan los resultados obtenidos de la simulación, las cuales determinan la posibilidad de integración de las herramientas Ansible, Git, Gitlab para ejecutar cambios en la red, pero por medio de pipelines CI/CD que automatizan y aceleran los procesos de implementación.

Se termina el trabajo con conclusiones y recomendaciones que se generan luego de haber culminado el desarrollo del presente trabajo.

CAPÍTULO I: PLANTEAMIENTO DEL PROBLEMA

1.1. Formulación del problema

Un ingeniero de red conoce el por qué y cómo de las configuraciones de cada elemento que compone la infraestructura, las cuales se implementan a fin de solventar o depurar un determinado requerimiento o problema, ya sea un cambio de interfaz para el protocolo de enrutamiento, rediseño del direccionamiento IP, seguridad, etc. Prácticamente ningún equipo físico de la red es estático en su configuración y esta varía de acuerdo a cómo evolucionan los requisitos del cliente u organización a través del tiempo. Según la recopilación de (Shah & Dubaria, 2019), obtenida de varios blogs de CISCO, el 74% de los operadores reportan que los cambios de red han impactado significativamente en su negocio, el 97% de los operadores admite al factor humano como causante de las caídas de red y el 22% de las interrupciones no planificadas son causadas por algún tipo de error humano. Lo anteriormente expuesto deriva en una cultura de miedo al cambio en el personal que implementa y administra las redes de datos, así como una metodología de trabajo individualista donde se evita colaborar con los demás a fin de evitar al máximo los errores de terceros.

Con estos antecedentes, ha surgido la imperiosa necesidad de cambiar este enfoque tradicional hacia una nueva cultura de trabajo y es ahí donde se desarrolla el concepto de NetDevOps, que toma la cultura DevOps pero aplicada al campo de las redes, implementando las metodologías de Agile, pipelines CI-CD, automatización e infraestructura como código (IaC), motivando a que el ingeniero de red no tenga miedo al cambio en las redes de datos sino más bien que se adapte al mismo y pueda trabajar de manera colaborativa con su equipo y con un conjunto de herramientas que le brinden facilidades para la administración de este tipo de redes que pasan a denominarse como redes programables.

En DevOps se tenía claro la importancia del versionamiento y control de scripts en los pipelines CI-CD, no obstante, con NetDevops se dio un mayor enfoque a la parte funcional de automatización y productividad dejando de lado esta herramienta que permite inicializar y cerrar correctamente el ciclo, ya que está asociada con la gestión de cambios incrementales, compartir el trabajo con los demás miembros del equipo, conciliar y combinar los cambios, poder volver a una versión anterior de un archivo, proporcionar capacidades de comprobación de sintaxis y pruebas automatizadas, etc.

Ante toda la problemática anteriormente expuesta se plantea como solución a dichos inconvenientes, investigar si es factible la integración de Git en un pipeline CI/CD dentro de un entorno de infraestructura de red programática.

1.2. Objetivos de la Investigación

Objetivo General

Realizar una Prueba de Concepto de una infraestructura de red de nueva generación basada en el concepto de NetDevOps que tenga el control de versionamiento usando Git en la ejecución de pipelines CI/CD.

Objetivos Específicos

- Implementar una infraestructura de red de nueva generación en la nube con el protocolo de comunicación VXLAN.
- Verificar el correcto funcionamiento de la conectividad y análisis de encabezados en la infraestructura.
- Implementar el uso de GIT para el control de versionamiento del IaC diseñado y ejecución de pipelines CI/CD.

1.3. Justificación de la Investigación

El cambiante mundo en el que vivimos ha ido de la mano con la necesidad de desarrollar redes modernas que permitan soportar las cada vez más exigentes necesidades de conectividad de nuestra sociedad, lo que ha generado que a lo largo de los años surjan diferentes etapas de transición en el mundo del Networking. Hoy se habla que nos encontramos en la "Era programable" de las redes (Preston & Sullivan, 2017), donde se pasa del concepto típico de las mismas, basadas en routing y switching, a un modelo moderno, más escalable, fácil de implementar, monitorear y automatizar (Salazar, Naranjo & Marrone, 2018) cuyo enfoque se denomina NetDevOps, en donde se orienta el concepto DevOps, pero al campo de las redes de datos, es decir la combinación de programabilidad en la infraestructura utilizando API´s y una cultura de trabajo colaborativa.

Con NetDevOps se pueden desarrollar infraestructuras de red basadas en pruebas de concepto, que simulan un entorno real en producción y en donde se prueban los cambios antes de implementarlos en un ambiente en producción, minimizando así los errores y promoviendo la colaboración entre los miembros de un equipo; todo este concepto dentro de NetDevOps se lo conoce como pipelines CI-CD (Integración continua-Desarrollo continuo). Al ser tratada la red como código dentro de NetDevOps, los cambios en la configuración se realizan tal cual como lo hacen lo desarrolladores de código en DevOps, en repositorios de versionamiento, de esta manera se facilita la implementación de cambios de software, la extracción de cambios de software, la administración del control de versiones y el trabajo en equipo, etc.

Si bien el concepto de NetDevOps inició en el 2018 aún es un campo poco explorado y divulgado en la comunidad, basta con buscar publicaciones relevantes dentro de Google Schoolar o en bases de datos científicas como la IEEE, para encontrarnos con solo un par de papers sobre este tema, cuyos autores si bien han indagado sobre esta temática, no profundizan de manera práctica en el control de versionamiento. Por todo lo anteriormente expuesto en la presente investigación se busca demostrar la factibilidad y vialidad en la integración de todo un entorno de red programable dentro del pipeline CI-CD a través del uso de GIT.

CAPÍTULO II: FUNDAMENTACIÓN TEÓRICA

2.1. Infraestructura como código

La IaC (Infraestructure as code) es un enfoque de la automatización de la infraestructura basado en prácticas del desarrollo de software en el que se enfatiza el uso de rutinas coherentes y repetibles para el aprovisionamiento o cambio en los sistemas y su configuración (Morris Kief, 2021). Se puede realizar cambios en el código, y luego utilizar la automatización para probar y aplicar esos cambios en el sistema con un proceso de validación continua que comprueba en todo momento que lo definido se puede aprovisionar permitiendo así gestionar todo el ciclo de vida del entorno (Romero Sánchez, 2021).

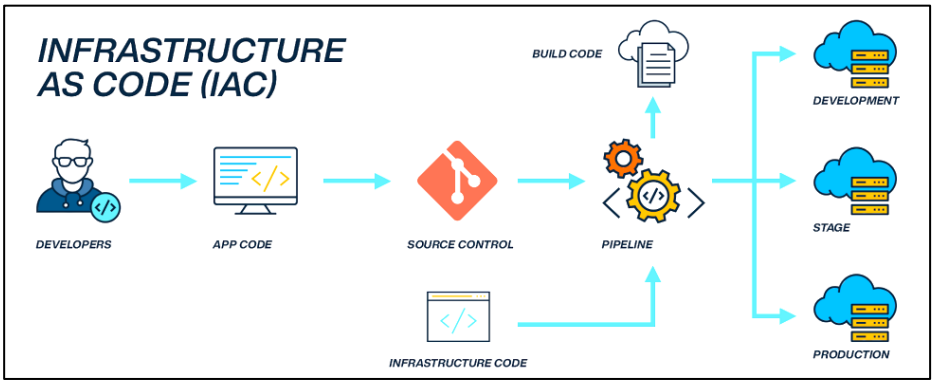


Figura 1. Flujo de trabajo de la IaC
Recuperado de (SparkFabrik Team, 2022)

Según (Morris Kief, 2021), existen tres prácticas básicas para implementar infraestructura como código:

- **Definir todo como código:** es una práctica fundamental para realizar cambios de una manera rápida y segura, ayudando en términos de reusabilidad, consistencia y transparencia.
- **Probar y entregar continuamente todo el trabajo en progreso:** Los equipos de infraestructura efectivos son rigurosos con las pruebas, mientras trabajan, prueban, en lugar de esperar hasta que hayan terminado; la idea es construir calidad en lugar de intentar probar la calidad. CI (Continuous Integration) implica fusionar y probar el código de todos durante el desarrollo. CD (Continuous Development) lleva esto más allá, manteniendo el código fusionado siempre listo para la producción.
- **Construir piezas pequeñas y sencillas que se puedan cambiar de forma independiente:** Cuanto más grande es un sistema, más difícil es cambiarlo y más fácil es romperlo, bajo esa premisa en DevOps al usar IaC, el sistema se compone de partes pequeñas y simples, cada pieza es fácil de entender y tiene interfaces claramente definidas. El equipo puede cambiar fácilmente cada componente por separado y puede implementarlo y probarlo de forma aislada.

Otro factor de importancia a tomar en cuenta dentro de la ingeniería de software es la calidad del código. Los siguientes principios de implementación son pautas para diseñar y organizar IaC a fin de apoyar este objetivo:

- **Código declarativo e imperativo separados:** El combinar estos diferentes tipos de código genera un Design Smell, es decir un problema de estructura que, si bien no produce errores de compilación o de ejecución, afecta negativamente el factor de calidad del software (Alkharabsheh et al., 2016).
- **Tratar al código de infraestructura como código real:** Es necesario mantener la misma disciplina de ingeniería del software al código de infraestructura a fin de que el mismo pueda ser fácil de entender y mantener, complementando a la par con prácticas de revisión de código, programación en pareja, pruebas automatizadas, etc.

Una organización que adopte la IaC para gestionar su infraestructura obtendrá beneficios tales como:

- Utilizar la infraestructura de TI como un facilitador de transformación digital para la entrega de valor.
- Reducir el esfuerzo y riesgo que normalmente se emplearía a la hora de realizar un cambio.
- Construcción de sistemas fiables, seguros y mantenibles que permitan regresar a un estado anteriormente conocido.
- Mejorar la velocidad para solucionar problemas y resolver fallos.
- Proveer herramientas centralizadas donde los cambios están disponibles a los usuarios de los departamentos de desarrollo, operaciones u otras partes interesadas.

Red como código (Network as Code)

Los conceptos de IaC se pueden adaptar al Networking y es lo que origina un nuevo término denominado "Network as Code" o NaC. La idea principal es almacenar todas las configuraciones de red en un repositorio centralizado basado en un Sistema de Control de Versiones (VCS) que gestiona y realiza un seguimiento de los cambios en la red. Este sistema que almacena todas las configuraciones de la infraestructura de la red en archivos JSON, YAML o XML, se considera la fuente única de verdad para la configuración de todo lo relacionado a la red (Shah & Dubaria, 2019).

NaC despliega la configuración utilizando APIs programáticas y herramientas SDK (Software Development Kit) con el objetivo de promover la inmutabilidad, esto significa mantener los sistemas completamente como código, sin que se realicen operaciones manuales en ellos y la idempotencia que es producir el mismo resultado deseable cada vez que se realiza una ejecución.

Herramientas de Infraestructura como código

Existen varias alternativas dentro del mercado de TI, sin embargo, según (Domínguez-Quintero & Vargas-Lombardo, 2021) , existen algunas características esenciales que debe presentar estas plataformas a fin de cumplir de la mejor manera el objetivo de dinamismo y administración de la infraestructura y que deben ser tomadas en cuenta a la hora de seleccionar una herramienta de IaC:

- **Programable:** La plataforma debe tener acceso a API´s, SDK´s, así como admitir scripting de codificación basado en la sintaxis de algún lenguaje de programación o formato de dato (YAML, JSON, XML). Algunas herramientas que cumplen estos requisitos son: Ansible, Terrafom, Chef, Puppet.
- **Demandable:** Debe permitir crear, modificar o eliminar cualquier componente de la infraestructura en tiempos de respuesta muy cortos.

- **Autoadministrable:** Los usuarios deben poder acceder directamente a la gestión de recursos de la plataforma a través de API's programáticas sin la necesidad de recurrir previamente a un departamento de IT o usuario externo, excepto en casos muy puntuales.

Las herramientas de IaC: Ansible, Puppet y Chef mantienen entornos sin-estado lo cual es una ventaja importantísima, puesto que en base al análisis y pruebas de concepto realizadas en la investigación de (G. D. Salazar, 2021) "Las API's que mantengan el estado (mantengan en el servidor el estado) son inconvenientes para una automatización de tipo *full-stack* (infraestructura y servicios/aplicaciones en su conjunto) principalmente por que el estado será destruido por un rebuild típico de la infraestructura, además de que estos entornos son difíciles de actualizar y ejecutar migraciones".

2.2. API

En la era actual de las redes programables, los ingenieros de redes han aprendido a programar a fin de llegar a implementar herramientas de software que les permitan conectarse con las plataformas de infraestructura como código. El concepto de API (Application Programming Interfaces) hace referencia a una interfaz para la comunicación entre aplicaciones que facilita la interacción con los datos a los desarrolladores que programan dichas aplicaciones; bajo el contexto detallado, los fabricantes de equipos de red han desarrollado interfaces de programación de aplicaciones o API´s que permiten la programabilidad y automatización de la red, independientemente del sistema operativo de red (NOS) que se utilice.

El no estar atado a la sintaxis CLI de un determinado proveedor, es sumamente beneficioso para el ingeniero de red, ya que reduce el tiempo de la curva de aprendizaje que normalmente utilizaría para poder diseñar e implementar una solución de red, tomando en cuenta que en un entorno real, especialmente si se trata de una arquitectura empresarial, se utilizarán numerosos equipos de red y de diversos proveedores, donde cada cual posee su propia sintaxis.

Estilo arquitectónico REST

Según (Jin et al., 2018), REST (Representational State Transfer) es la opción más popular para el desarrollo de API's últimamente. El paradigma REST es utilizado por proveedores como Google, Stripe, Twitter y GitHub. Las API's de REST exponen los datos como recursos y utilizan métodos HTTP estándar para representar transacciones CRUD (Create, Read, Update, Delete).

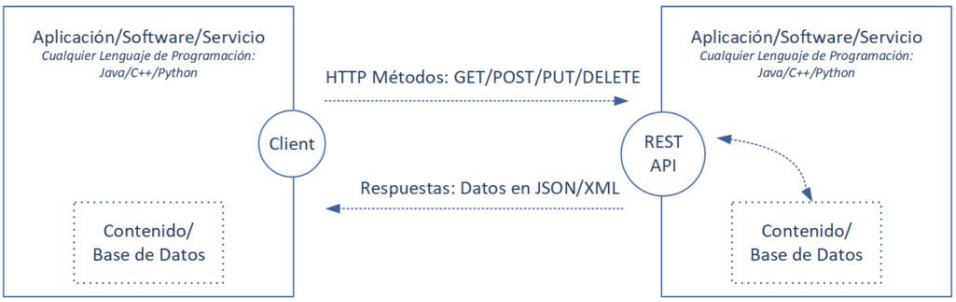


Figura 2. Interfaz de comunicación de una API
Recuperado de *(*Alvarez, 2019)

Hay que tener claro que REST no es un estándar o protocolo, sino más bien un estilo arquitectónico compuesto de acuerdos y restricciones. Algunas restricciones que se impone

son (Alvarez, 2019):

- Los recursos forman parte de las URL, como /users
- Para cada recurso, generalmente se implementan dos URLs: una para la colección, como /users, y una para un elemento específico como /users/U123.
- Para los recursos se utilizan sustantivos en lugar de verbos. Por ejemplo, en lugar de /getUserInfo/U123, se utiliza /users/U123.
- Los métodos HTTP como GET, POST, UPDATE y DELETE informan al servidor sobre la acción a realizar. Los diferentes métodos HTTP se describen a continuación:
    - Create: Utiliza POST para crear nuevos recursos.
    - Read: Utilice GET para leer recursos. Las peticiones GET nunca cambian el estado del recurso. El método GET tiene una semántica de sólo lectura. GET es idempotente, en consecuencia, las llamadas al método se pueden almacenar en caché.
    - Update: Utiliza PUT para reemplazar un recurso y PATCH para actualizaciones parciales de recursos ya existentes.
    - Delete: Utiliza DELETE para borrar recursos existentes.

    Una vez que se ejecute alguno de estos métodos, el servidor nos devolverá códigos de estado de la respuesta HTTP, indicando el éxito o el fracaso. Generalmente la nomenclatura de rango de códigos viene especificada de la siguiente forma:

    - Rango 2xx: éxito.
    - Rango 3xx: que un recurso se ha movido.
    - Rango 4xx: error del lado del cliente.
    - Rango 5xx: errores del lado del servidor.

- Las API's REST pueden devolver respuestas JSON o XML, debido a su simplicidad y facilidad de uso con JavaScript, JSON se ha convertido en el estándar para las API's modernas.

2.3. Formatos de Datos

Basados en (Cisco Devnet-Programming Fundamentals, 2020) existen tres formatos estándar para intercambiar información con API´s dentro del Networking:

- **JSON:** JavaScript Object Notation
- **XML:** Extensible Markup Language
- **YAML:** Ain't Markup Language

La elección específica de alguno de estos tipos de formatos de datos depende mucho de la aplicación, entorno y contexto donde se lo requiera implementar, ya que cada uno tiene sus propias características específicas.

2.3.1. JSON

JSON (JavaScript Object Notation) es un formato ligero para almacenar, transferir o leer datos, el cuál es fácil de leer y escribir para los humanos y a la par fácil para las máquinas de analizar y generar. Las convenciones que utiliza son familiares para los programadores de la familia de lenguajes en C, incluidos C ++, C #, Java, JavaScript, Perl, Python y muchos otros lo que lo

convierte en un lenguaje de intercambio de datos ideal(*JSON*, s/f).

Reglas de Sintaxis JSON

- Un objeto JSON está rodeado por llaves "{ }".
- Los pares nombre-valor se agrupan por dos puntos ":" y se separan por una coma ","
- Las matrices empiezan con corchetes " [ ] ".
- Las comas finales y los ceros iniciales en los números están prohibidos.
- Cada clave dentro del JSON debe ser única y encerrada entre comillas dobles.
- El tipo booleano coincide solo con dos valores especiales (*true*, *false*) y los valores NULL se representan por con la palabra reservada *null*.
- JSON no admite una sintaxis para la inclusión de comentarios en la codificación.
- El espacio en blanco no tiene relevancia y se puede aplicar sangría usando identación o espacios.

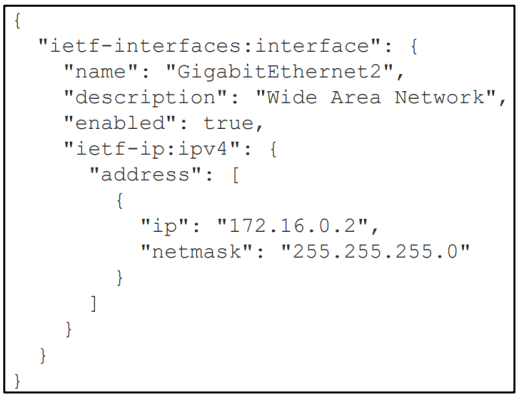


Figura 3. Formato JSON
Recuperado de (Cisco Devnet-Programming Fundamentals, 2020)

2.3.2. XML

XML (Extensible Markup Language) es un lenguaje de marcado, similar a HTML ya que usa etiquetas, fue diseñado con el fin de almacenar, transportar y leer datos. Un documento de XML tiene una estructura de árbol, comenzando con un nodo raíz que contiene nodos secundarios (también conocidos como elementos secundarios). Cada elemento puede contener datos, pero también 1…n atributos (Patni, 2017). En los últimos años ha ido en desuso debido a la simplicidad que ofrecen JSON o YAML.

Reglas de Sintaxis XML

- La primera línea del documento se conoce como prolog:

  <?xml version="1.0" encoding="UTF-8" ?>

  Si bien su declaración no es obligatoria, el utilizarla requiere que se la defina siempre al inicio.
- Cada una de las etiquetas debe tener su apertura y respectivo cierre:

  <tag>dato</tag>
- Las etiquetas son sensibles a minúsculas y mayúsculas.
- Los atributos XML deben ir en comillas.
- Un objeto en XML se compone de un par clave/valor, siendo el nombre de la etiqueta la clave:

  <clave>valor</clave>

- La identación o anidación es una buena práctica a utilizar a fin de facilitar la lectura del script.

```
<?xml version="1.0" encoding="UTF-8" ?>
<interface xmlns="ietf-interfaces">
  <name>GigabitEthernet2</name>
  <description>
    Wide Area Network
  </description>
  <enabled>true</enabled>
  <ipv4>
    <address>
      <ip>172.16.0.2</ip>
      <netmask>255.255.255.0</netmask>
    </address>
  </ipv4>
</interface>
```

Figura 4. Formato XML

Recuperado de (Cisco Devnet-Programming Fundamentals, 2020)

2.3.3. YAML

Es un formato de serialización de datos, es decir no es un lenguaje marcado, ampliamente utilizado para archivos de configuración dentro de herramientas como Ansible, Docker, Kubernetes por nombrar algunas. Si se lo compara con JSON y especialmente con XML, este formato estándar es considerablemente más simple y entendible para humanos, lo que lo hace más fácil para leer y codificar en un script.

Reglas de Sintaxis YAML

- Los atributos dentro de un objeto deben estar al mismo nivel de identación (basado en espacios en blanco) para formar parte del mismo objeto y ser un documento YAML válido.
- No se permite el uso de caracteres de tabulación para las sangrías de identación.
- Usa tres tipos de estructuras: Objeto, array, escalar.
- Un objeto en YAML es una colección de pares de clave: valor que se constituye en el bloque de construcción básico y en donde cada objeto es miembro de al menos un diccionario.
- Cada YAML comienza con --- que denota el inicio de un archivo YAML, del mismo modo si se lo escribe al final denota el fin del archivo.
- El guion medio se lo puede usar para separar un elemento del array o lista de objetos
- YAML distingue entre mayúsculas y minúsculas.

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

Figura 5. Formato YAML

Recuperado de (Cisco Devnet-Programming Fundamentals, 2020)

2.4. Python

Python es un lenguaje de programación de alto nivel, interpretado, orientado a objetos con una comunidad muy extensa involucrada en su desarrollo. Sus principales características son:

- Es un lenguaje sencillo e intuitivo fácil de aprender, enseñar, utilizar y obtener.
- Es gratuito, OpenSource, multiplataforma.
- Ideal para tareas cotidianas y scripting lo que permite tiempos de desarrollo cortos.

Este lenguaje se ha popularizado en el mundo del NetDevOps debido a su amplio repertorio de bibliotecas y frameworks como Netmiko, Paramiko, NAPALM y Nornir, que facilitan un sinfín de posibilidades para las interacciones de los dispositivos de red de diferentes proveedores. Gracias al soporte de la comunidad de Python, la lista de dispositivos soportados no ha parado de expandirse, añadiendo soporte para vendedores adicionales a medida que se introducen en el mercado. Así mismo todos los archivos de Python son módulos que pueden ser reutilizados o importados en otro programa Python, esto hace que sea fácil compartir programas entre ingenieros y fomentar la reutilización del código.

La flexibilidad que nos provee Python permite que se pueda empezar en un script de tamaño relativamente pequeño con unas pocas líneas de código y crecer hasta convertirse en un sistema en producción completo lo que es muy práctico en el mundo del NetDevOps ya que el código crece a la par del tamaño de la red. Todo lo anteriormente expuesto ha generado que Python se encuentre en uso para sistemas de empresas de vanguardia del sector (Python Wiki, 2018).

Tipos de datos en Python

Python implementa tipificación dinámica es decir a medida que se declara objetos automáticamente determina el tipo del mismo. Los tipos de datos estándar incorporados en el intérprete son:

- Numéricos: entero, flotante, complejo, y booleano
- Secuencias: cadenas, listas, tuplas
- Mapeos: diccionarios
- Conjuntos: set y frozenset
- Ninguno: El objeto null

2.5. Google Cloud

La computación en la nube consiste en abstraer la infraestructura informática ya sea software o hardware, para ofrecerlos como un servicio de pago por uso a través del internet. En el mercado actual, Google Cloud Platform es una de entre varias opciones que ofrecen este servicio a los usuarios, con la gran ventaja de que posee una de las redes más extendidas geográficamente y avanzadas en el mundo, siendo de las pocas empresas que posee un cable de fibra óptica privado bajo el Océano Pacífico (Krishnan & Gonzalez, 2015). Lo expuesto anteriormente ha permitido que Google Cloud aloje sus recursos en varios lugares del mundo, dando la posibilidad a que un cliente pueda seleccionar el Centro de Datos con la región y zona dentro de esta, más cercana a su ubicación, para desplegar su infraestructura de manera escalable y fiable basado en dos criterios sumamente importantes que son la redundancia de fallas y baja latencia.

“La entidad organizadora de lo que se está compilando y donde se asigna y usa los recursos en Google Cloud se conoce como proyecto, el cual contiene la configuración, los permisos y otros metadatos para describir las aplicaciones. Los recursos de un mismo proyecto pueden trabajar en conjunto mediante la comunicación de una red interna, sujetos a las reglas de las regiones y las zonas, pero un proyecto no puede acceder a los recursos de otro proyecto, a menos que se

use una VPC compartida o el intercambio de tráfico entre redes de VPC ” (Google Cloud, 2022).

Un servicio informático que hará posible la creación y ejecución de máquinas virtuales en la infraestructura es Compute Engine y gracias a la oferta de $300 en crédito gratuito para evaluar el entorno Google Cloud, se la puede utilizar para la implementación de un proyecto NetDevOps a fin de no tener limitaciones de recursos.

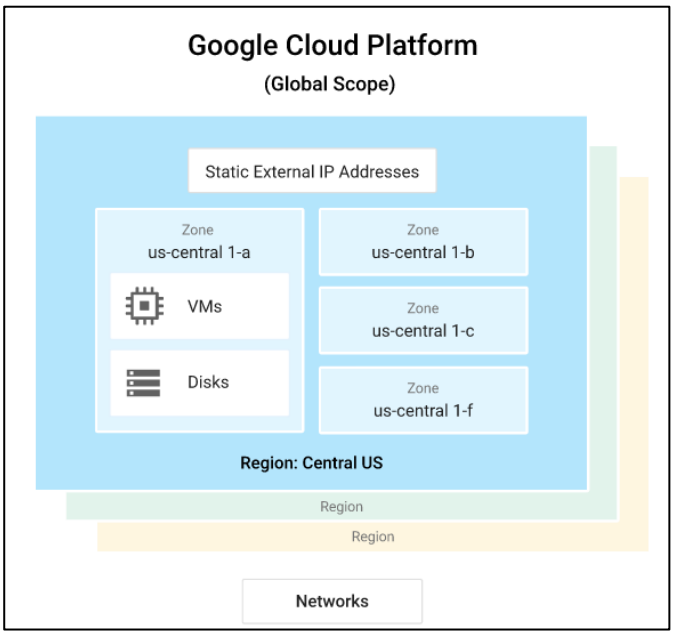


Figura 6. Visión Global de Google Cloud
Recuperado de (Google Cloud, 2022)

2.6. VXLAN

VXLAN (Virtual Extensible LAN), definida en el RFC 7348, es una tecnología de red superpuesta(overlay), diseñada para proporcionar conectividad de nivel 2 y 3 a través de la encapsulación inteligente de tráfico sobre una red IP tradicional(Vaca P. & Salazar-Chacón., 2020).

El estándar VXLAN define la encapsulación a través de una cabecera de 8 bytes compuesta por un identificador de 24 bits (VNID) y varios bits reservados. La cabecera VXLAN, a lo largo de la trama Ethernet original se coloca como una carga UDP. El VNID de 24 bits se utiliza para identificar los segmentos L2(red de capa 2) y mantener el aislamiento entre ellos. Con el VNID de 24 bits, VXLAN puede soportar 224 segmentos locales (Naranjo & Salazar, 2018), es decir hasta 16 millones de segmentos de L2 en la misma red, adecuándose a los requerimientos actuales que exige una red moderna.

La principal terminología utilizada al describir los componentes clave de la tecnología VXLAN es la siguiente (Naranjo & Salazar, 2018):

- **VTEP (Virtual Tunnel Endpoint):** Elemento hardware o software (Hipervisor) encargado de instanciar la tunelización VXLAN y realizar el proceso de encapsulación y desencapsulación de las tramas ethernet en paquetes UDP/IP. También conocido como LEAF.

- **VNI (Virtual Network Instance):** Instancia de red lógica que proporciona servicios L2 o L3 y define un dominio de difusión de capa 2.
- **VNID (Virtual Network Identifier):** identificador de 24 bits que permite 16 millones de redes lógicas.
- **Bridge-Domain:** Conjunto de puertos físicos o lógicos que comparten el mismo dominio de difusión.
- **SPINE:** Dispositivo que interconecta los VTEP.

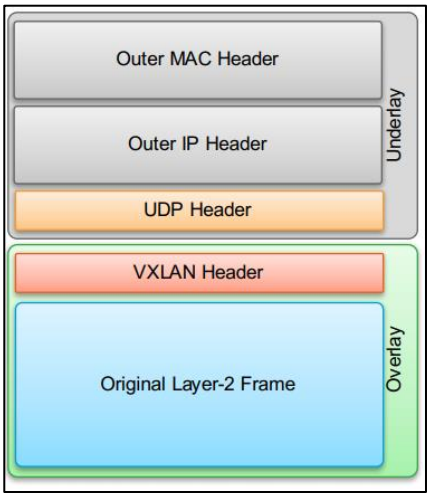


Figura 7. Formato de Encapsulación VXLAN
Recuperado de (Naranjo & Salazar, 2018)

Los segmentos VXLAN se crean entre los dos puntos finales del túnel VXLAN que son los VTEP, estos se encargan de realizar la encapsulación de la trama capa 2 con un encabezado UDP y un encabezado IP y enviarlo a través del túnel hacia su otro extremo VTEP que se encargará de la desencapsulación del paquete. Algunos ejemplos de VTEP son hipervisores, máquinas virtuales y conmutadores compatibles con VXLAN (Citrix Systems, 2022).

Entre las razones para la adopción de VXLAN tenemos:

- VXLAN soluciona el problema de escalabilidad debido a la baja cantidad de segmentación que se tiene con los tags VLAN(VID), los cuales son hasta 4094; esto se solventa al incrementar la cantidad de tags a más de 16 millones usando un identificador de segmento de 24-bits.
- La misma VLAN no puede extenderse a través de un dominio de Capa 3, VXLAN sí puede hacerlo, al tratar dos dominios L2 separados y haciéndolos parecer como uno.
- No depende de STP (Spanning-tree Protocol) para converger, pues emplea protocolos de Capa 3.
- El balanceo de carga se realiza a través de todos los links activos, evitando así el desuso de recursos en la infraestructura.
- Con VXLAN en un Data Center (DC) se puede alojar múltiples inquilinos manteniendo el aislamiento entre sus redes, es decir, es posible que la máquina virtual (VM) de cada inquilino comparta el servidor físico con otros inquilinos distribuidos en servidores físicos dentro o en distintos centros de datos y poder aislar el tráfico de cada inquilino para la seguridad y posibles superposiciones de dirección MAC, además de permitir realizar la conexión entre Data Centers remotos (DCI).

2.7. Arquitectura Spine-Leaf

En un modelo de red tradicional típico, se manejan tres niveles (Aruba Networks, 2022):

- Switches de acceso: que se conectan a los servidores.
- Switches de distribución: proveen conexiones redundantes a los switches de acceso.
- Switches de núcleo: brindan un transporte rápido entre los switches de distribución y que se conectan a un par redundante de alta disponibilidad.

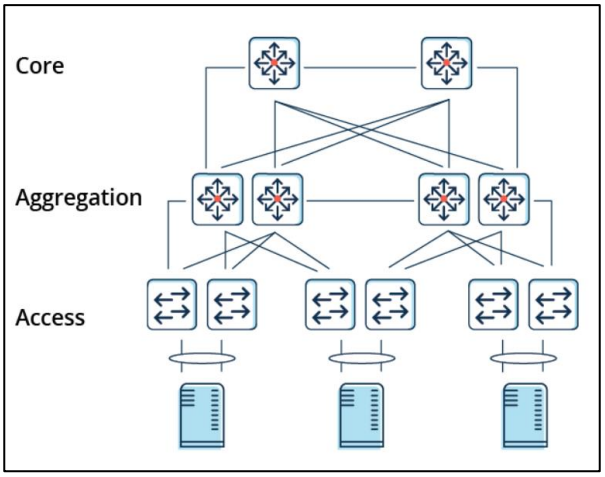


Figura 8. Capas de conmutación en Redes Tradicionales
Recuperado de (Aruba Networks, 2022)

Sin embargo, el auge de la infraestructura en la nube donde se mantienen recursos distribuidos en varios servidores o máquinas virtuales ubicados en distintas regiones ha generado en un aumento del tráfico de extremo a extremo. Es ahí donde con la arquitectura Spine-Leaf, podemos garantizar que este tráfico siempre mantenga la misma cantidad de saltos hacia el destino reduciendo de esta manera la latencia; adicionalmente podemos tener redundancia, escalabilidad y alto rendimiento, por lo que es ideal para ser implementada en redes de centros de datos.

Este tipo de arquitectura Spine-Leaf, elimina el protocolo de árbol de expansión STP, hace uso de switchs de puerto fijo y se diseña con una infraestructura de escalabilidad horizontal que en su nivel más básico consta de dos capas de conmutación:

- Nivel Spine: Los switches de este nivel son la columna vertebral de la red e interconectan todos los switches de Leaf en una topología de malla completa.
- Nivel Leaf: Se compone de switchs de acceso que acumulan el tráfico de servidores y se conectan al nivel Spine o el núcleo de la red de manera directa.

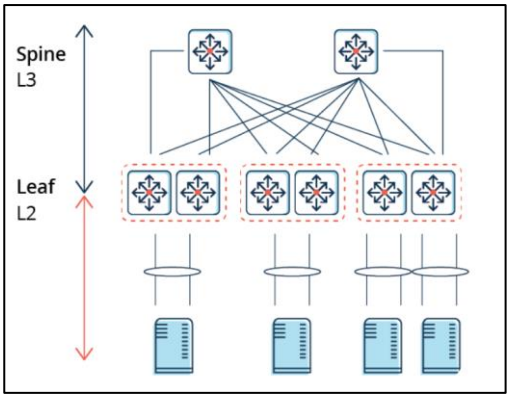


Figura 9. Arquitectura Spine-Leaf
Recuperado de (Aruba Networks, 2022)

2.8. NetDevOps

Todo necesita conectividad, por lo que la red es sin lugar a dudas un activo fundamental de cualquier empresa en la actualidad, sin embargo, las mismas se han vuelto grandes y complejas. Según datos de la revista Network World (Bednarz, 2016), basados en una encuesta a 315 profesionales en redes de empresas medianas y grandes de California, se llega a concluir los siguientes resultados:

- El 97% de los encuestados considera al error humano como un causante de las interrupciones de la red.
- El 44% de encuestados cree que los cambios en la red conducen a interrupciones o problemas de rendimiento.
- El 69% de participantes prefiere verificar de manera manual un cambio de red luego de su implementación ya sea mediante la inspección de los dispositivos por CLI o por comandos de testeo de red como ping, traceroute.
- Para el 60% de la población el promedio de tiempo para resolver un problema de red es de una a cinco horas.

Los datos anteriormente expuestos denotan la imperante necesidad de converger en el networking hacia la cuarta era del networking, es decir, “La Era Programable”. Para dicho fin ha surgido el NetDevOps (Networking Development Operations) que es un término que hace referencia al uso de DevOps aplicado a las redes de datos, esto implica la combinación de programabilidad en la infraestructura usando API´s y herramientas de automatización con el fin de cumplir el ciclo cultural de DevOps y así generar una sinergia entre desarrollo e implementación.

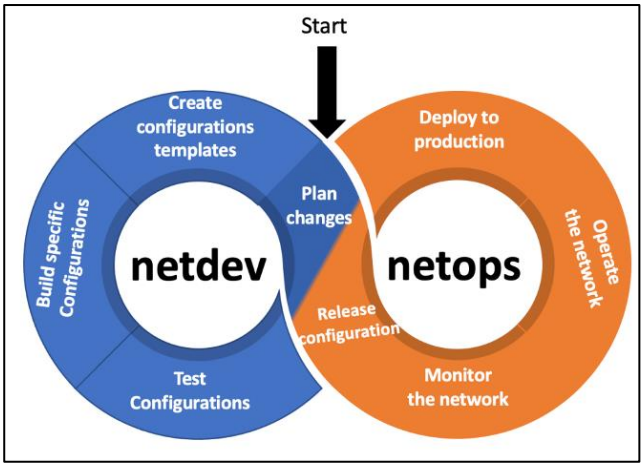


Figura 10. Fases de NetDevOps
Recuperado de (Pinto & Chauhan, 2022)

El principal objetivo que se persigue al utilizar NetDevOps es buscar la agilidad completa a la hora de diseñar, desarrollar, integrar, probar e implementar una red de datos y que los equipos de desarrollo y operaciones no estén separados, sino que trabajen de manera conjunta involucrándose en todo el ciclo de vida de una infraestructura de red, permitiéndoles: actuar con velocidad, mantener altos índices de calidad y controlar los cambios con suma rapidez. Esto necesariamente requiere un cambio en la filosofía y cultura de una organización en donde se deje de tener miedo al cambio y se lo acepte como eje fundamental de sus operaciones.

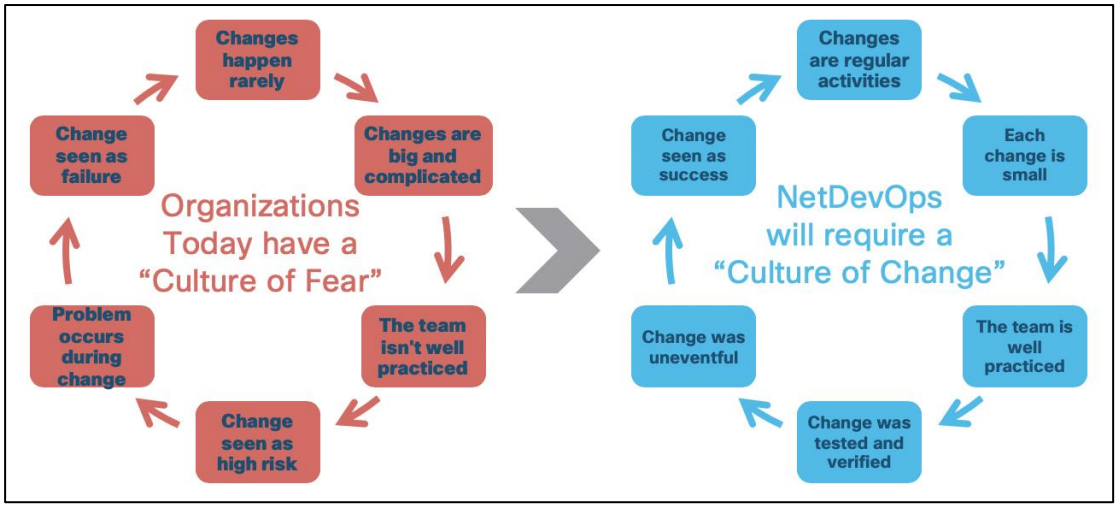


Figura 11. Cultura del Cambio con NetDevOps
Recuperado de (Gomez, 2021)

Lo anteriormente expuesto requiere que el ingeniero de red que desee adaptar NetDevops, deba adquirir nuevas habilidades y conocimientos como (Pavón, 2020) :

- Comprensión de las fases que componen el ciclo de vida del software.
- Capacidad para operar los equipos de red a través de NETCONF, RESTCONF y REST API.
- Conocimiento de los entornos de red en plataformas cloud como AWS, Azure o Google Cloud.
- Habilidades de programación, que permitan automatizar tareas en la red "Network as code".

- Familiaridad con herramientas que se emplean en desarrollo de software como control de versiones y aplicarlo a las configuraciones de la red.

Según (Agudo, 2018) entre los métodos y estrategias que se debe adoptar para implementar NetDevOps, tenemos:

- **Automatización:** Reducir la intervención manual directa cuando se tenga una gran cantidad de equipos de red en donde se deba ejecutar cambios, minimizando por ejemplo las tareas repetitivas de modo que se mitiguen los posibles errores humanos.
- **Infraestructura como código:** Definir un repositorio central de configuración con políticas unificadas para todos los elementos de la red.
- **Integración continua:** Ejecutar pruebas y despliegues automatizados.
- **Verificación continua:** Implementar comprobaciones periódicas del estado de la infraestructura, así como una monitorización automatizada que permita validar los cambios ligados al estado de la red.
- **Evolución hacía una red basada en la intención (Intent-Based Networking):** A fin de definir políticas generales a ser aplicadas en el equipamiento de la red por medio de las herramientas de automatización como por ejemplo: roles con políticas para el acceso a recursos o recursos globales necesarios para el equipamiento de firewalls, conmutadores, etc.

2.8.1. Pipelines CI-CD

Un pipeline es una consecución de fases en donde, partiendo de un código fuente, se llega a una versión reléase (build particular de lanzamiento). En un entorno DevOps los pipelines son el componente de nivel superior de la integración, entrega e implementación continuas y se definen en un archivo de configuración dentro del proyecto que se esté desarrollando, el cual mediante un servidor de CI/CD permite que los pipelines sean ejecutados de manera automatizada a través de la generación de commits o pushes.

En el modelo NetDevOps al igual que DevOps, existen tres prácticas habituales en el desarrollo, implementación y despliegue de una red, las cuales se pueden integrar en un flujo de pipeline CI-CD de la siguiente forma:

- **Continuous Integration (CI):** Implica la integración continua de código nuevo en el repositorio de Git existente mediante la fusión automática de cambios y la ejecución de pruebas.CI aplica automáticamente el control de versiones y la gestión de cambios para garantizar que nadie rompa o escriba accidentalmente sobre el código de otra persona. Además, se ejecutan pruebas unitarias automatizadas en el código para comprobar si hay errores.
- **Continuous Delivery (CD):** En la fase de entrega continua, la nueva configuración de la red se pasa a un entorno de prueba o PoC (Proof-Of-Concept), normalmente con dispositivos virtualizados en una red privada, desde la cual se puede realizar pruebas automatizadas. Por ejemplo, las pruebas de carga comprobarán si hay problemas de rendimiento y las pruebas de seguridad garantizarán que el archivo de definición no introduzca ninguna vulnerabilidad en la red de producción.
- **Continuous Deployment (CD):** La última fase es la implementación continua donde se implementará automáticamente los cambios ya probados en la red de producción. Debido a que las modificaciones en la red han sido testeadas a fondo en las dos etapas anteriores del pipeline, la puesta en producción generará un

impacto mínimo en las operaciones de red de la organización.

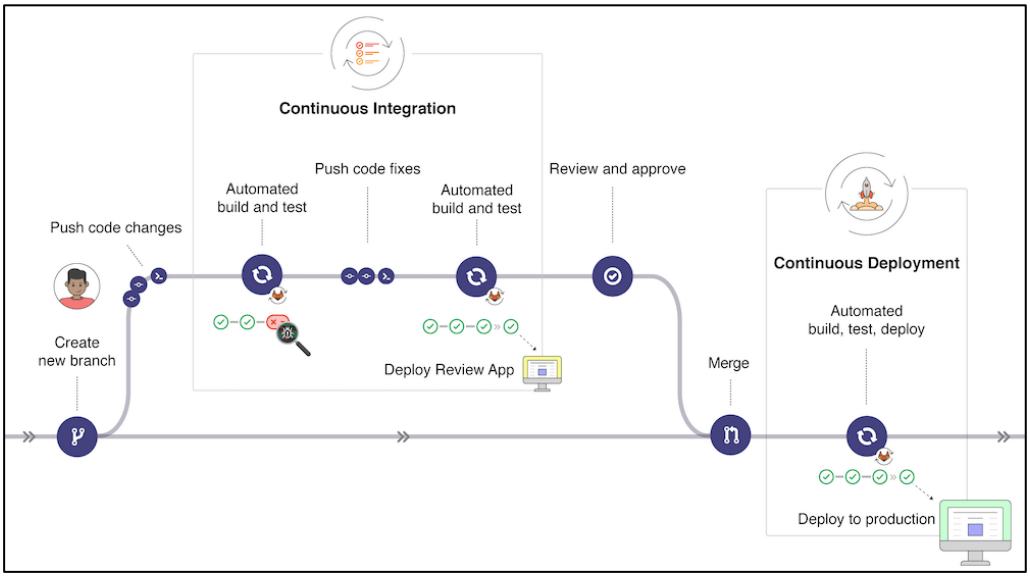


Figura 12. Flujo CI/CD en un repositorio
Recuperado de (Gitlab, 2022)

2.8.2. Herramientas de NetDevOps

Dentro de NetDevOps se hace uso de las herramientas que en DevOps se utilizan para optimizar y agilizar los procesos de implementación de infraestructuras o aplicaciones, adaptándolas al campo del networking, sea esta una red tradicional o moderna. Un entorno NetDevops puede hacer uso de un conjunto muy variado de herramientas que se acoplan a cada una de las diferentes fases de la metodología DevOps.

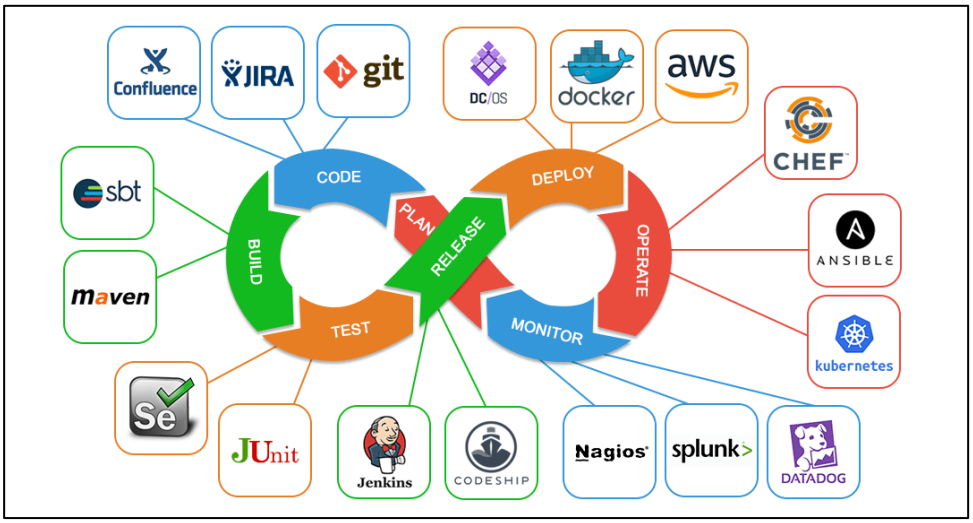


Figura 13. Herramientas de DevOps
Recuperado de (Genius IT Training, 2018)

Algunos proveedores lideres a nivel mundial en equipos de redes de telecomunicaciones como CISCO, por ejemplo, han desarrollado sus propios ecosistemas a fin de poder impulsar este nuevo paradigma tal como se muestra en la *figura 14*:

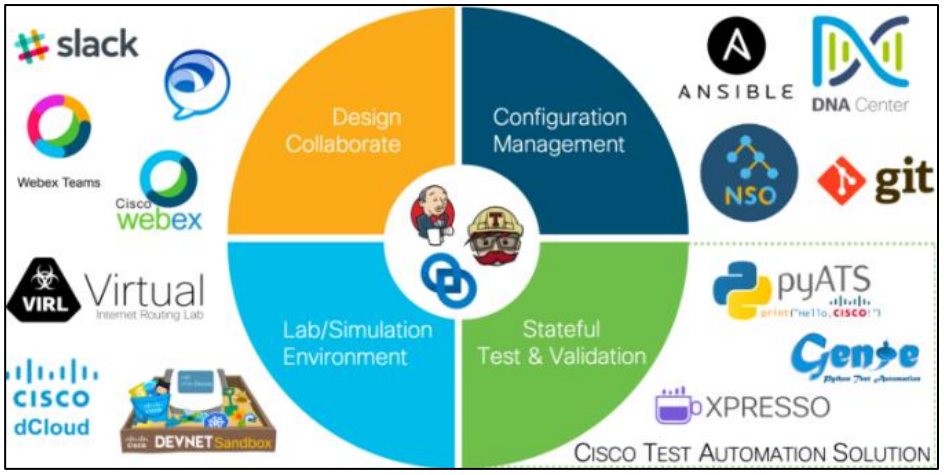


Figura 14. Ecosistema NetDevOps de CISCO
Recuperado de (Yuan, 2019)

Git & Gitlab

Git es un sistema de control de versiones (VCS) Opensource, desarrollado por Linus Torvalds en el 2005 cuya función es registrar los cambios en los archivos a lo largo del tiempo, permitiendo recuperar las versiones anteriores, ver el historial de cambios de tal manera que se facilite el trabajo colaborativo por medio de repositorios remotos alojados en plataformas como Gitlab. El concepto es sencillo, sin embargo, las cosas se complican cuando se trabaja con otros usuarios en el mismo conjunto de archivos.

Tradicionalmente, un VCS almacena su información como una lista de cambios en los archivos y manejan la información que almacenan como un conjunto de archivos y las modificaciones hechas a cada uno de ellos a través del tiempo:

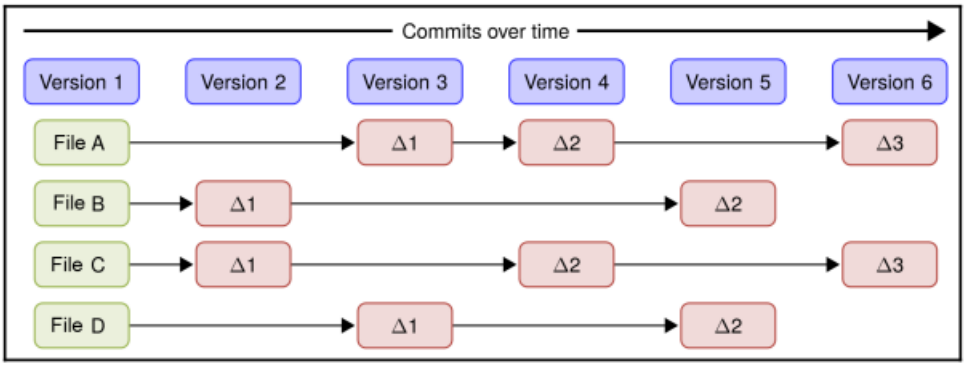


Figura 15. Almacenamiento de Datos en un VCS Tradicional
Recuperado de (Geisshirt et al., 2018)

Git es diferente, ya que registra una copia instantánea de todos los archivos rastreados y sus rutas relativas a la raíz del repositorio, es decir, los archivos rastreados por Git en el árbol del sistema de archivos. Cada confirmación en Git registra el estado completo del árbol. Si un archivo no cambia entre confirmaciones no se almacenará el archivo de nuevo. En su lugar, Git almacena un enlace al archivo (Geisshirt et al., 2018). Esto se muestra en el diagrama de abajo, donde se ve cómo quedarán los archivos después de cada commit.

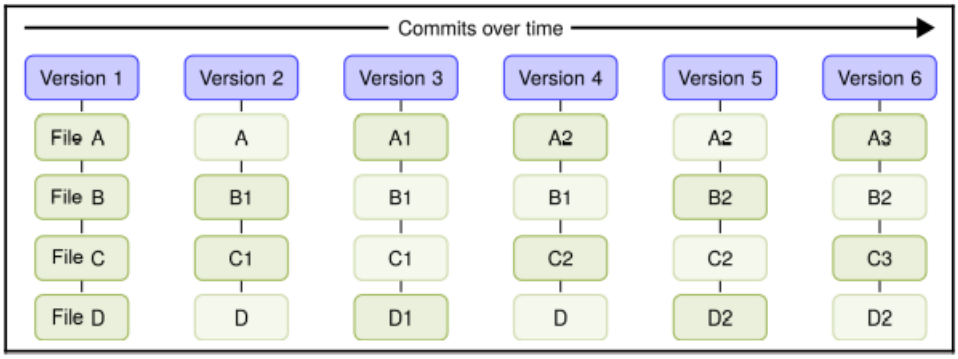


Figura 16. Almacenamiento de Datos como instantáneas en Git
Recuperado de (Geisshirt et al., 2018)

El primer paso para inicializar un nuevo repositorio de Git es hacer el comando: **git init**. El comando creará un área conocida como Staging que representa un lugar temporal donde se guardan los archivos y un repositorio local.

En Git los archivos pueden vivir y moverse entre 3 diferentes estados: confirmado (committed), modificado (modified), y preparado (staged):

- Confirmado: significa que los datos están almacenados de manera segura en la base de datos local.
- Modificado: significa que se ha modificado el archivo, pero todavía no está confirmado a la base de datos.
- Preparado: se ha marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

Existen tres secciones principales de un proyecto de Git: El directorio de Git (Git directory), el directorio de trabajo (working directory), y el área de preparación (staging area).

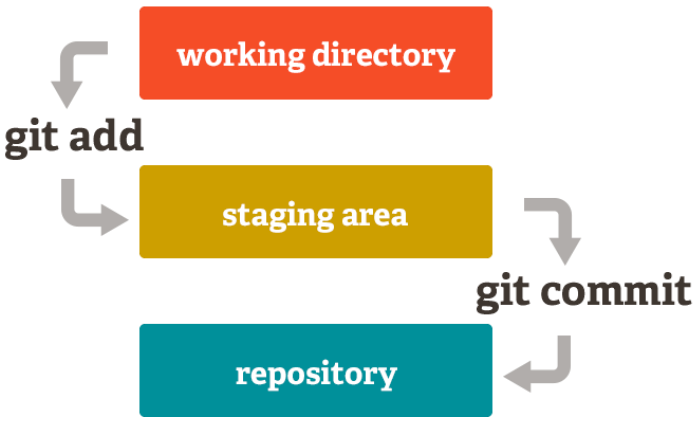


Figura 17. Secciones principales de un proyecto en Git
Recuperado de (de la Cruz, 2017)

Todos los commits se aplican sobre una rama que por defecto es la master o main dependiendo de la versión de Git que se utilice. Pero también se pueden crear más ramas a fin de que un determinado commit de una rama pase a otra de modo que se pueda trabajar con una versión del repositorio sin afectar el flujo de trabajo principal que se ubicaría en la rama máster.

Las operaciones de Git son locales, lo que significa que no necesita interactuar con un repositorio remoto o central limitando la posibilidad de compartir los repositorios y desarrollar

trabajo colaborativo, de ahí que surgen los repositorios remotos de código Git como Gitlab.

Gitlab es un software para el control de versiones de código y desarrollo colaborativo basado en Git y escrito en el lenguaje de programación Ruby. Posee una versión OpenSource: Gitlab CE (Comunity Edition) y otra de tipo empresarial con licenciamiento: Gitlab EE (Enterprise Edition). Cabe destacar que Gitlab no es solo un repositorio web remoto sino una plataforma Devops completa que permite al ingeniero de software o de red la implementación de pipelines CI (Continuous Integration) y CD (Continouos Delivery).

Gitlab CI/CD

GitLab CI/CD es la herramienta de la suite DevOps Gitlab para los métodos de: integración continua (CI), entrega continua (CD), implementación continua (CD). Para que las fases del pipeline CI/CD se realicen de manera ágil, Gitlab ejecuta estas tareas por medio de runners, los cuales se pueden compilar en ejecutores de tipo Shell, SSH, VirtualBox o en tecnologías de contenedores como Docker y Kubernetes.

Con GitLab CI/CD se puede generar un flujo de trabajo de desarrollo colaborativo donde se puede comenzar discutiendo una implementación de código para solventar un requerimiento y trabajando localmente en los cambios propuestos. A continuación, se puede enviar las confirmaciones a una rama de pruebas en un repositorio remoto alojado en GitLab. La inserción activa automáticamente el pipeline de CI/CD para el proyecto. Luego, GitLab (Gitlab, 2022):

- Ejecuta scripts automatizados (secuencialmente o en paralelo) para:
    - Crear y probar la aplicación.
    - Obtener una vista previa de los cambios en la aplicación.

Después de que la implementación funcione como se esperaba:

- Se solicita que el código sea revisado y aprobado.
- Se combina los cambios de la rama de pruebas en la rama principal o master
    - GitLab CI/CD implementa los cambios automáticamente en un entorno de producción.

Finalmente, y uno de los aspectos más importante al adoptar este paradigma de trabajo, es que, si algo sale mal, se tiene la posibilidad de revertir los cambios, ya que al trabajar con versionamiento, se puede regresar a una versión anterior de la IaC donde todo funcionaba correctamente.

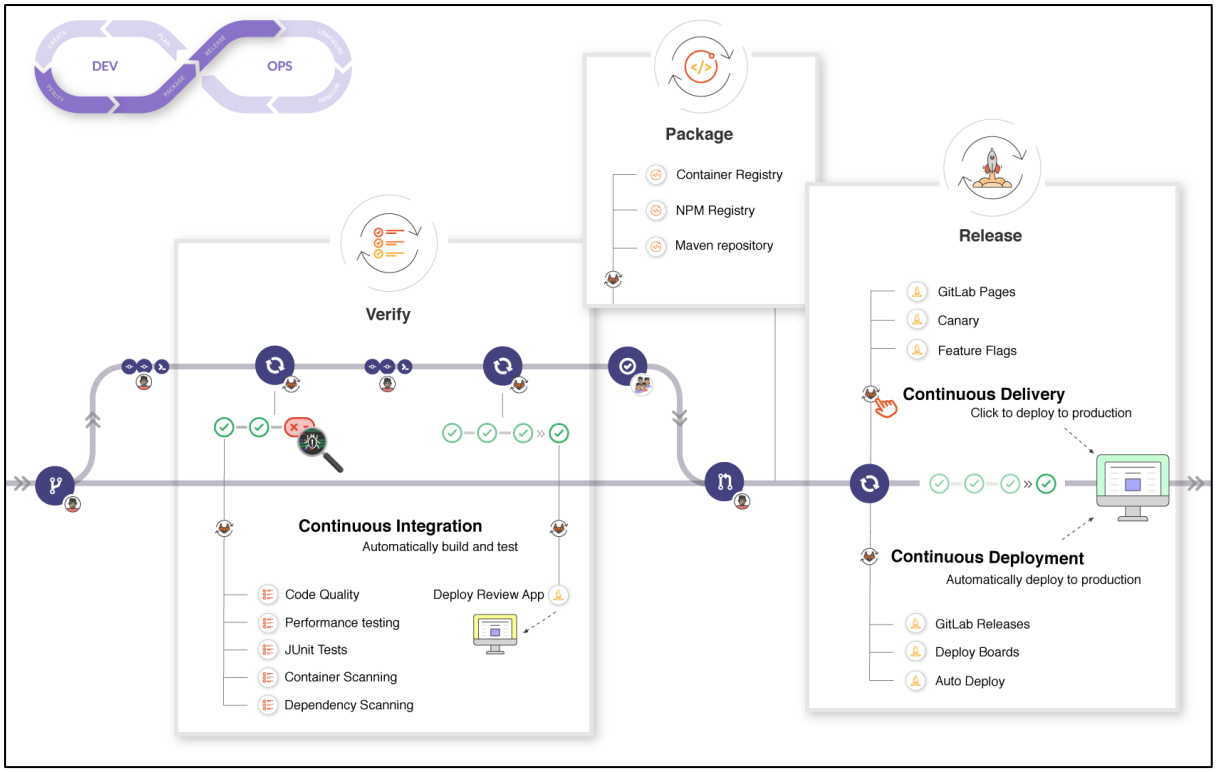


Figura 18. Workflow en las etapas del ciclo de vida DevOps
Recuperado de (Gitlab, 2022)

EVE-NG

EVE-NG (Emulated Virtual Environment – Next Generation) es un emulador de infraestructuras de red de TI sin cliente, que permite realizar pruebas de conceptos, soluciones y ambientes de entrenamiento a empresas o personas que buscan simular redes lo más cercanas posibles a lo que se encontrarían en un ambiente real. Esta herramienta cuenta con dos versiones: la Profesional que es de pago anualmente y cuenta con soporte al cliente y una serie de funcionalidades extendidas y la versión Comunitaria, que es una versión dirigida a estudiantes y desarrolladores, con una parte reducida de las mejoras existentes en la versión Profesional.

Figura 19. Logo de EVE-NG
Recuperado de (EVE-NG, 2022)

Para el profesional de TI brinda facilidades de (EVE-NG, 2022):

- Aprendizaje, al permitir entrenarse en el manejo de NOS (Network Operating System) de marcas como Cisco, Juniper, Cumulus Linux, PaloAlto y más.
- Diseño, al poder construir redes basadas en requerimientos a fin de validar que un prototipo de diseño sea la solución óptima.

- Eficiencia, ya que se puede montar la arquitectura real en un ambiente seguro para pruebas, sin el riesgo de provocar daños en el mundo real.
- Flexibilidad, al poder trabajar con varias marcas y hacerlas interactuar a la disposición del diseñador.

EVE-NG es un Linux que emula los sistemas operativos utilizando los motores llamados Quemu y Dynamips.

Ansible

Ansible es una herramienta Opensource de automatización y programabilidad basada en Python con una gran acogida en ambientes de networking de próxima generación. Sus principales objetivos son la simplicidad y la facilidad de uso, así como su fuerte enfoque en la seguridad y la confiabilidad, con un mínimo de partes móviles, el uso de OpenSSH para el transporte (con otros transportes y modos de extracción como alternativas), y un lenguaje que está diseñado en torno a la auditabilidad por parte de humanos, incluso aquellos que no están familiarizados con el programa (RedHat, 2022).

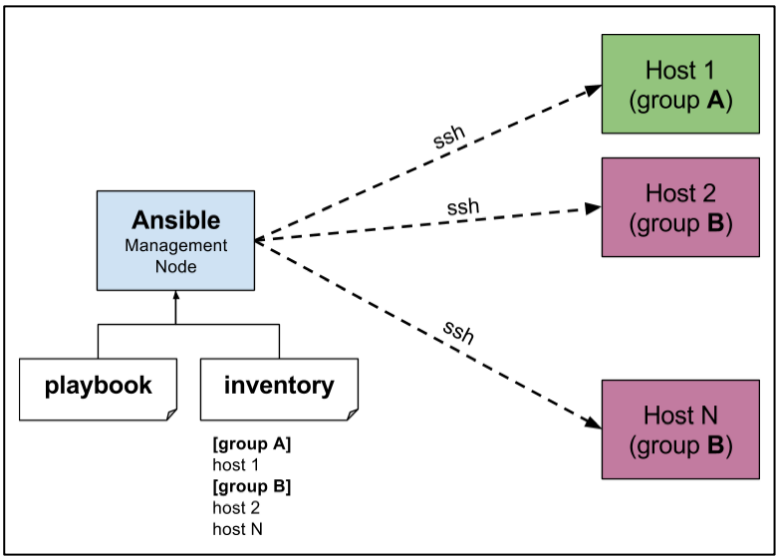


Figura 20. Arquitectura de Ansible
Recuperado de (Kumar, 2019)

Por medio de un nodo de control o administración que hace uso de SSH y API´S para poderse conectar a los equipos hosts, servidores, routers, switches, etc., se puede simplificar la automatización de tareas repetibles y tratar a la infraestructura de red como código. Siendo de este modo adecuado para administrar desde pequeñas configuraciones con pocas instancias hasta entornos empresariales con muchos miles de instancias de red.

Debido a que Ansible una gran compatibilidad para la interconexión con equipos de red sean estos privativos o abiertos, se lo puede utilizar para poder implementar redes más flexibles y con interoperabilidad, adaptadas a las necesidades específicas de cada usuario, lo que al final promueve y facilita la adaptación del concepto de Open Networking.

Ansible para su adecuado funcionamiento, se compone de tres tipos de archivos importantes:

- **Inventario:** Archivo también conocido como hosts, donde se enlista los equipos y se los agrupa en base a su función, tipo o tareas que van a realizar. En este archivo se

especifican las direcciones de administración de cada equipo, además de la forma de envío de las configuraciones automáticas, por lo general mediante SSH.

- **Playbook File**: Un playbook es un archivo de estructura YAML que permite combinar tareas y dar algún control de acción al flujo de ejecución.
- **Roles:** Los roles permiten reutilizar variables, tareas y controladores sobre un grupo de hosts, mediante la agrupación de ficheros en una estructura de directorio que se puede compartir con otros usuarios de Ansible. Una carpeta de rol contiene una carpeta /tasks con un archivo de tareas main.yml. Otras carpetas importantes son: handlers, library, templates, vars, defaults.
- **Módulos:** Son librerías o plugins que utiliza Ansible para controlar servicios y que se pueden usar desde la línea de comandos o haciendo llamadas en un playbook. Estos módulos se alojan en colecciones y como respuesta a su ejecución devuelven datos en formato JSON.

CAPÍTULO III: METODOLOGÍA

En este capítulo se presenta un análisis del funcionamiento de la prueba de concepto (PoC) que se ha diseñado siguiendo la metodología NetDevOps, esto con el objetivo de comprender como va a funcionar la red moderna automatizada y programable propuesta, antes de ser emulada. En primer lugar, se define la topología de la red VXLAN que se va a implementar de forma programática a través de Ansible en un entorno OpenNetworking con Cumulus Linux y su posterior integración con Git como herramienta de versionamiento de la red en conjunto con Gitlab para la ejecución de los ciclos CI-CD.

3.1. Software EVE-NG

Se describe los recursos utilizados para la instalación del software emulador EVE-NG dentro de la instancia de Google Cloud aprovisionada. Adicional se adjunta un manual de instalación y configuración en el Anexo 2, en el cual se detalla el proceso a seguir para su funcionamiento.

EVE-NG es un emulador de red de tipo multiproveedor que posee una versión community edition(gratuita), que para los fines prácticos de la prueba de concepto que se requiere implementar es más que suficiente. La elección de la misma se debe a que comparado con su principal alternativa GNS3 es más flexible y con una capacidad de emulación más amplia de fabricantes de equipos de red, sin tanto consumo de recursos.

Los recursos utilizados para la instalación de EVE-NG son los siguientes: una instancia en Google Cloud de tipo n1-standard, con Ubuntu 16.04 LTS Pro Server, 8 núcleos, 30 GB de RAM, Disco SSD persistente de 60 GB, acceso a internet y las imágenes qemu de los nodos de red Cumulus Linux, así como un cliente Ubuntu Desktop 18.04.

3.2. Gitlab CI-CD

Es la herramienta empleada para la implementación de los ciclos CI-CD de NetDevops que compila, prueba e implementa código automáticamente. Desde el repositorio de código almacenado en Gitlab se administra el código la IaC definida para la red SPINE-LEAF, así como la ejecución de cambios en la red automatizables dentro de un pipeline CI-CD automatizado. Al ser una prueba de concepto nos limitamos a representar un escenario de pruebas donde un cambio se ejecuta y verifica primero en un entorno emulado de pruebas, antes de que el mismo pueda pasar al entorno de producción.

Se ha optado por el uso de esta herramienta debido a que es un ecosistema muy maduro y con muchas herramientas para implementaciones CI-CD, es OpenSource, usa archivos de tipo YAML para la configuración y las canalizaciones CI-CD vienen incorporadas dentro de la misma plataforma de repositorio de código sin la necesidad de tener que recurrir a un servidor externo.

3.3. Topología de la Red

La arquitectura de la red está formada por 2 equipos LEAF que representan dos sedes que comparten el mismo dominio de capa 2, esto a pesar de estar separados por un equipo de capa 3, permitiendo superar la desventaja que se tiene con una VLAN de tipo tradicional. Adicionalmente se tiene una red administrable que sirve de puente entre los equipos SPINE-

LEAF y el equipo Linux que contiene: Ansible para la orquestación programática de la red, Git como herramienta de versionamiento de la IaC y Gitlab Runner para la automatización de los pipelines CI-CD.

La topología tipo Spine-Leaf emulada es la siguiente:

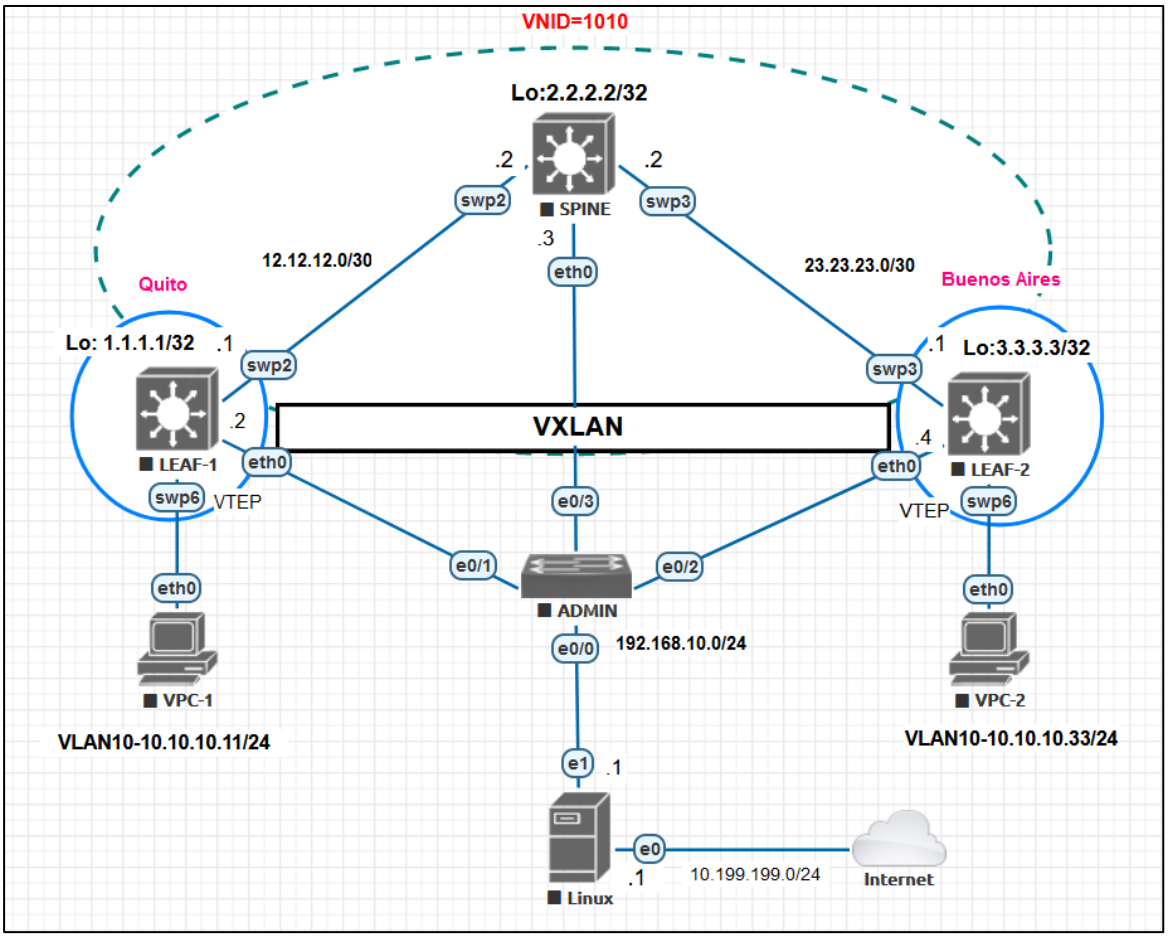


Figura 21. Topología PoC-VXLAN
Fuente: *Autor*

Direccionamiento IP

En la tabla adjunta se detalla el direccionamiento ip utilizado para el funcionamiento de la prueba de concepto diseñada.

Equipo	Red	Ip	
LEAF-1	12.12.12.0/30	swp2:12.12.12.1/30	eth0=192.168.10.2
LEAF-2	23.23.23.0/30	swp3:23.23.23.1/30	eth0=192.168.10.4
SPINE	12.12.12.0/30 23.23.23.0/30	swp2:12.12.12.2/30 swp3:23.23.23.2/30	eth0=192.168.10.3
VPC-1	10.10.10.0/24	10.10.10.11/24	
VPC-2	10.10.10.0/24	10.10.10.33/24	
SW-ADMIN	192.168.10.0/24		
Linux-Ansible	10.199.199.0/24	e0=10.199.199.1/24	e1=192.168.10.1

Tabla 1. Direccionamiento IP de la topología SPINE-LEAF
Fuente: *Autor*

3.4. Integración con Git y GitLab CI-CD

Para la integración de la red programable con Git se suben al repositorio web remoto en Gitlab los archivos que componen el directorio de Ansible, mismo que se encuentra instalando dentro del equipo Linux emulado con Ubuntu Desktop 18.04. Estos archivos son: hosts, ansible.cfg, playbooks, roles. Hacer esto nos permitirá tener el control de versionamiento de la red programable.

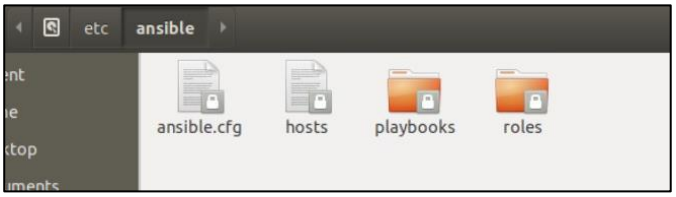


Figura 22. Directorio de Ansible dentro de Ubuntu
Fuente: *Autor*

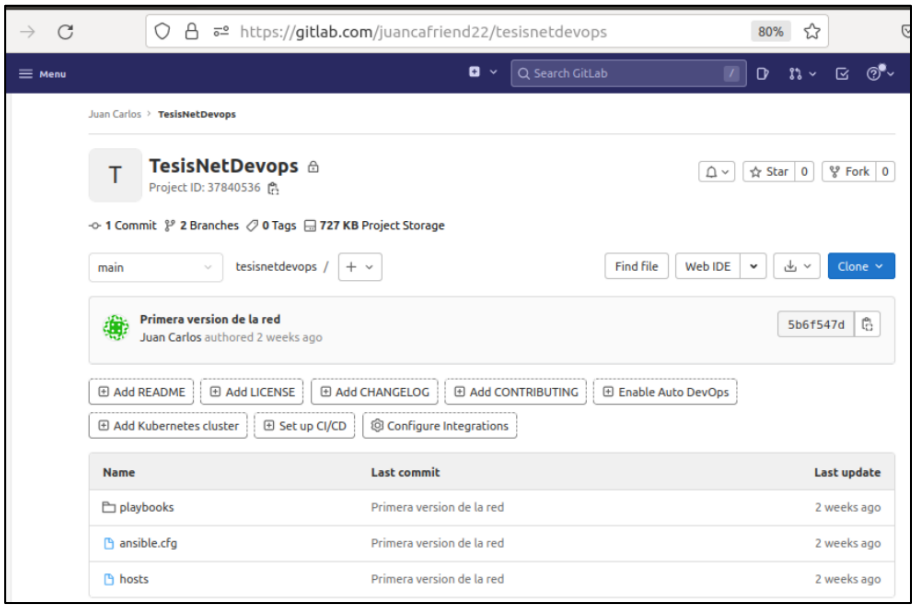


Figura 23. Repositorio remoto en Gitlab
Fuente: *Autor*

Para la parte de la integración con Gitlab CI-CD se instala dentro del equipo Linux, la aplicación Gitlab Runner que nos permite ejecutar los pipelines que se define dentro de un archivo "gitlab-ci.yml".

El pipeline comprende:

- Jobs: que definen qué hacer. Por ejemplo, realizar pruebas de conectividad, revisión de sintaxis yaml y tareas de ejecución de playbooks.
- Stages, que definen cuándo ejecutar los trabajos. Por ejemplo, etapas que ejecutan playbooks tras etapas que revisan la sintaxis.

Los cambios en la configuración de la red se proponen como código al sistema de control de versiones y se suben al repositorio remoto que es Gitlab por medio de un comando "Git Push". Una vez que se insertan los cambios, automáticamente por medio del runner de Gitlab CI-CD se inicia el pipeline. La herramienta de IaC, en este caso Ansible, ejecutará las tareas contenidas dentro del playbook hacia los equipos de red emulados dentro de EVE-NG que será nuestro

entorno de pruebas.

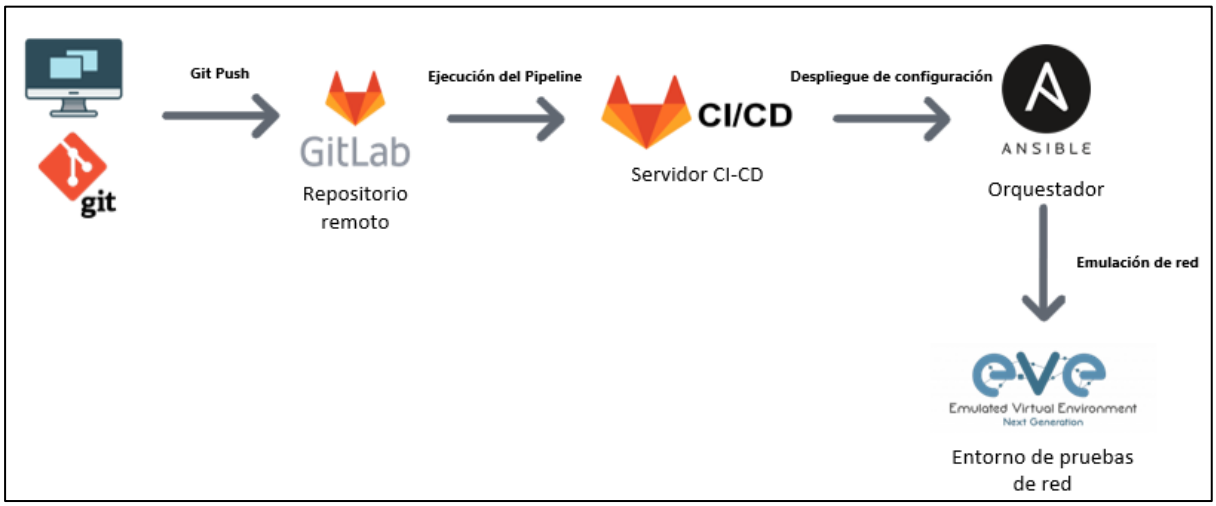


Figura 24. Workflow a ejecutarse en la PoC
Fuente: *Autor*

Para el caso práctico de ejecución del pipeline CI-CD, se simulará un escenario donde se requiere añadir una VLAN adicional a la topología de red VXLAN ya existente para lo cual se seguirá el siguiente flujo:

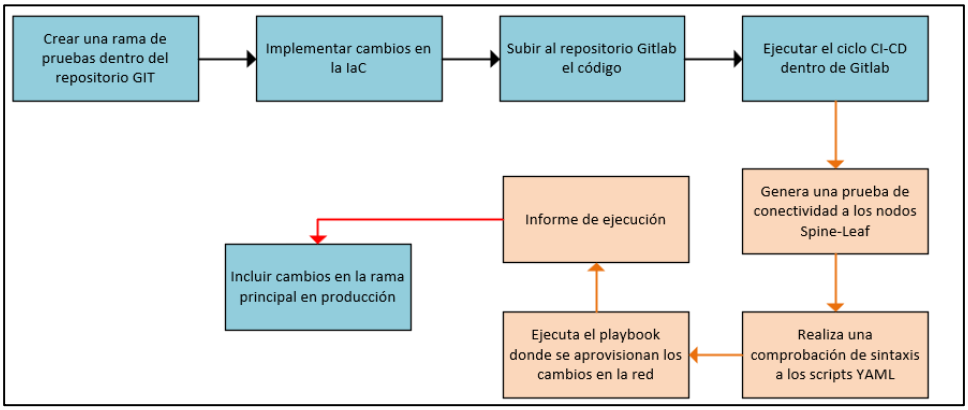


Figura 25. Diagrama de secuencia del escenario a emular
Fuente: *Autor*

CAPÍTULO IV: EMULACION

En este capítulo se especifica la configuración de cada equipo para que cumpla con el funcionamiento detallado en el capítulo III de metodología. Cabe destacar que esta arquitectura se despliega sobre una máquina virtual en Google Cloud para lo cual se adjunta el ***ANEXO 1 – INSTALACIÓN DE LA MÁQUINA VIRTUAL EN GOOGLE CLOUD,*** en donde se explica el proceso a seguir para su creación. Así mismo se parte de un emulador EVE-NG ya instalado y configurado en base al ***ANEXO 2 – INSTALACIÓN DE EVE-NG*** y el ***ANEXO 3 - CONFIGURACIÓN DE EVE-NG.*** Adicionalmente se toma como referencia para la configuración de VXLAN en Ansible el trabajo de (G. Salazar et al., 2020).

4.1. Configuración de VXLAN

Definimos una topología SPINE-LEAF de VXLAN con 3 equipos L2/L3 que emula el NOS Cumulus Linux y un nodo donde se virtualizará un Linux con Ansible de la siguiente manera:

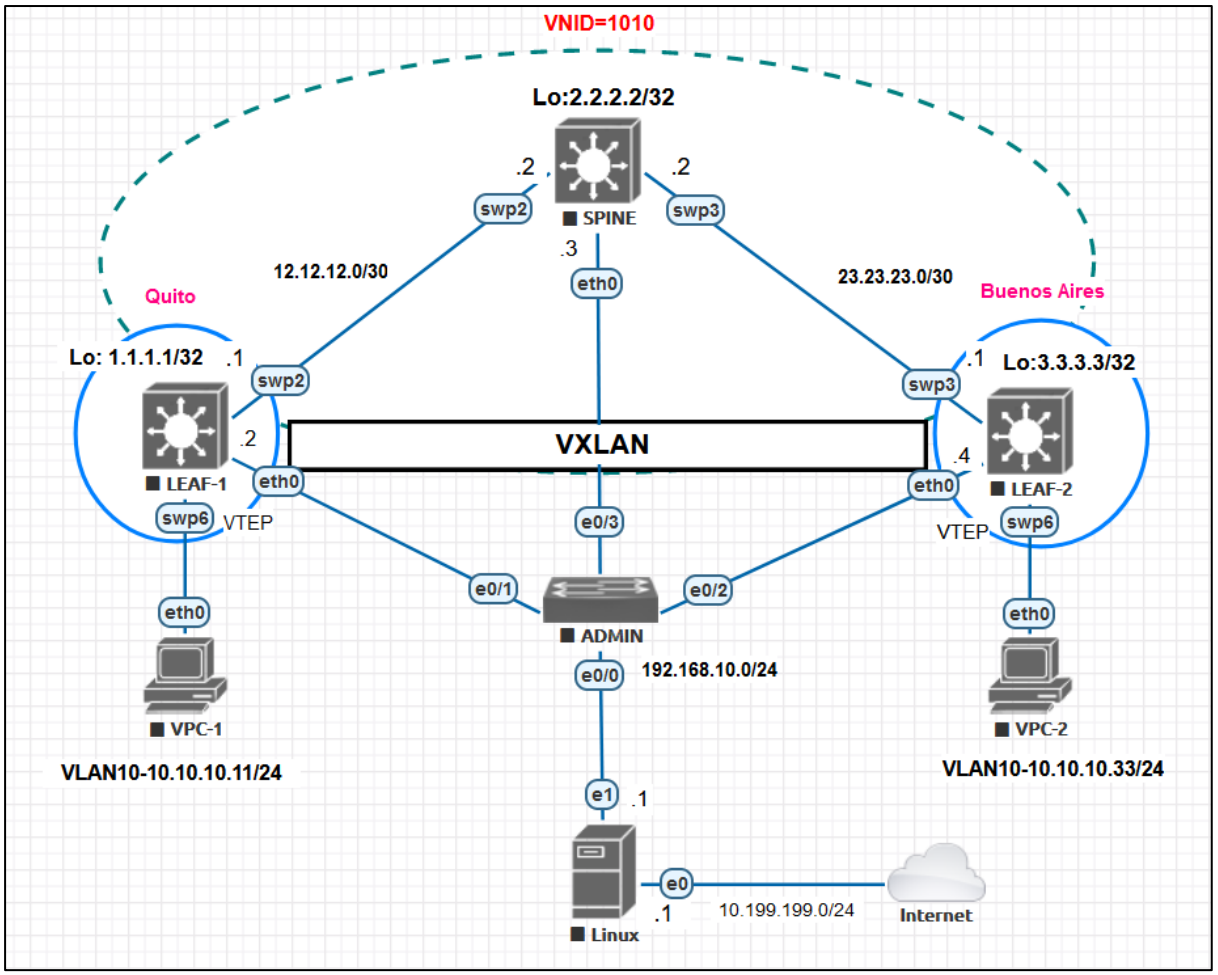


Figura 26. Arquitectura SPINE-LEAF emulada
Fuente: *Autor*

Existirá la red 192.168.10.0/24, la cual es la red de administración. Esta red conectará el Linux con la infraestructura VXLAN mediante un Switch Normal (Ethernet Switch) a través de sus interfaces eth0. Corremos la simulación y configuramos la red administrable en los 3 equipos Cumulus, así como en el nodo Linux que tiene el Ansible.

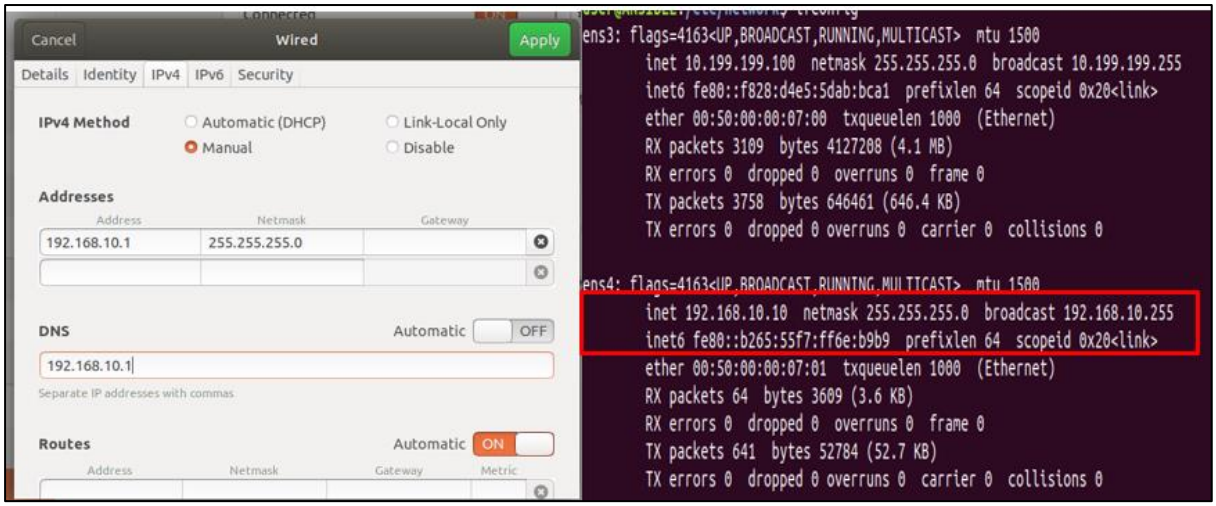


Figura 27. Configuración IP del nodo Linux (red administrable)
Fuente: *Autor*

```
cumulus@SPINE:mgmt:~$ net add interface eth0 ip address 192.168.10.3/24
cumulus@SPINE:mgmt:~$ net commit
--- /etc/network/interfaces      2022-06-22 16:34:55.630000000 +0000
+++ /run/nclu/ifupdown2/interfaces.tmp  2022-06-22 16:40:59.825000000 +0000
@@ -2,18 +2,19 @@
 # and how to activate them. For more information, see interfaces(5).

cumulus@LEAF-1:mgmt:~$ net add interface eth0 ip address 192.168.10.2/24
cumulus@LEAF-1:mgmt:~$ net commit
--- /etc/network/interfaces      2022-06-22 16:34:12.642000000 +0000
+++ /run/nclu/ifupdown2/interfaces.tmp  2022-06-22 16:42:17.787000000 +0000
cumulus@LEAF-2:mgmt:~$ net add interface eth0 ip address 192.168.10.4/24
cumulus@LEAF-2:mgmt:~$ net commit
--- /etc/network/interfaces      2022-06-22 16:35:50.361000000 +0000
+++ /run/nclu/ifupdown2/interfaces.tmp  2022-06-22 16:43:52.719000000 +0000
@@ -2,18 +2,19 @@
 # and how to activate them. For more information, see interfaces(5).
```

Figura 28. Configuración de direccionamiento en nodos Cumulus Linux (red administrable)

A fin de verificar conectividad realizamos una prueba de conexión entre Ansible y los equipos LEAF-1, SPINE, LEAF-2:

```
user@ANSIBLE: ~
File  Edit  View  Search  Terminal  Help
user@ANSIBLE:~$ ping 192.168.10.2
PING 192.168.10.2 (192.168.10.2) 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=64 time=3.12 ms
64 bytes from 192.168.10.2: icmp_seq=2 ttl=64 time=0.992 ms
64 bytes from 192.168.10.2: icmp_seq=3 ttl=64 time=1.20 ms
64 bytes from 192.168.10.2: icmp_seq=4 ttl=64 time=0.980 ms
^C
--- 192.168.10.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 0.980/1.576/3.124/0.898 ms
user@ANSIBLE:~$ ping 192.168.10.3
PING 192.168.10.3 (192.168.10.3) 56(84) bytes of data.
64 bytes from 192.168.10.3: icmp_seq=1 ttl=64 time=3.09 ms
64 bytes from 192.168.10.3: icmp_seq=2 ttl=64 time=0.966 ms
64 bytes from 192.168.10.3: icmp_seq=3 ttl=64 time=0.764 ms
64 bytes from 192.168.10.3: icmp_seq=4 ttl=64 time=0.960 ms
64 bytes from 192.168.10.3: icmp_seq=5 ttl=64 time=0.863 ms
^C64 bytes from 192.168.10.3: icmp_seq=6 ttl=64 time=1.00 ms
64 bytes from 192.168.10.3: icmp_seq=7 ttl=64 time=0.907 ms
^C
--- 192.168.10.3 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6038ms
rtt min/avg/max/mdev = 0.764/1.223/3.097/0.768 ms
user@ANSIBLE:~$
```

Figura 29. Prueba de ping desde Ansible hacia LEAF-1 y SPINE
Fuente: *Autor*

Se generan las llaves SSH en cada equipo, a fin de habilitar la conexión entre Ansible y la red (omitimos el cambio de ubicación y el passphrase, dando enter cuando aparezcan los mensajes):

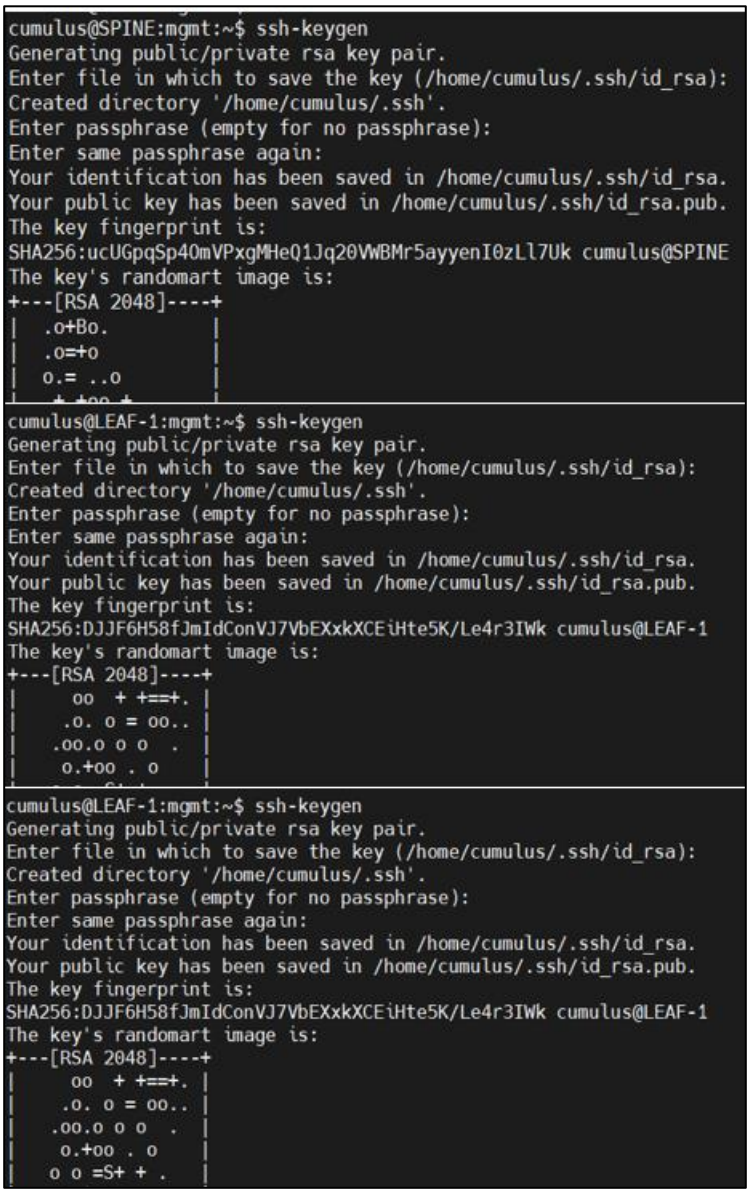


Figura 30. Generación de llaves SSH en Equipos LEAF-1, LEAF-2, SPINE
Fuente: *Autor*

Procedemos a configurar Ansible, empezaremos por el archivo hosts que se encuentra dentro del directorio del Ansible. No es recomendable dejar configurado el usuario y pass, ssh, dentro del archivo hosts, sin embargo, se lo realiza netamente para fines de la emulación por cuanto si hiciéramos uso del encriptado ssh, consumiría muchos recursos al generarnos un tamaño de clave de hasta 2048 bits.

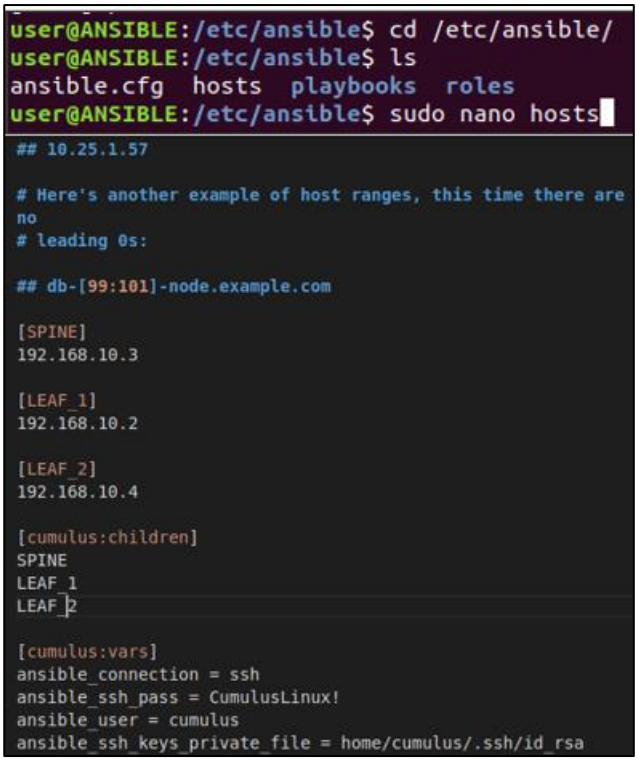


Figura 31. Configuración del archivo hosts
Fuente: *Autor*

Como no vamos a usar la autenticación con claves RSA por lo expuesto anteriormente(nota), es necesario también editar el archivo “ansible.cfg” de la siguiente manera:

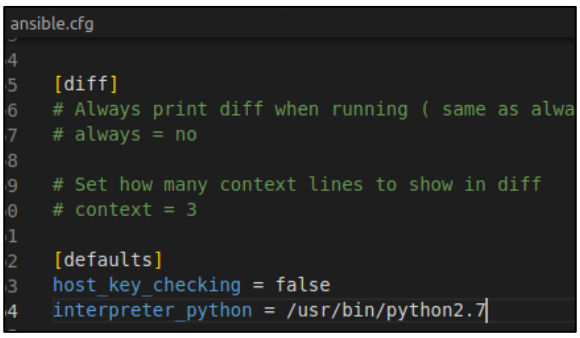


Figura 32. Configuración del archivo ansible.cfg
Fuente: *Autor*

Para que Ansible pueda enviar las configuraciones a cada equipo usa SSH, por ese motivo se generaron las llaves en cada Cumulus, pero también es necesario que Ansible tenga configurada de manera activa dicha relación de llaves público-privadas activa. Para dicho fin, enviamos los siguientes comandos:

```
user@ANSIBLE:/etc/ansible$ ssh-keyscan -H 192.168.10.2 >> ~/.ssh/known_host
# 192.168.10.2:22 SSH-2.0-OpenSSH_7.9p1 Debian-10+deb10u2
# 192.168.10.2:22 SSH-2.0-OpenSSH_7.9p1 Debian-10+deb10u2
# 192.168.10.2:22 SSH-2.0-OpenSSH_7.9p1 Debian-10+deb10u2
user@ANSIBLE:/etc/ansible$ ssh-keyscan -H 192.168.10.3 >> ~/.ssh/known_host
# 192.168.10.3:22 SSH-2.0-OpenSSH_7.9p1 Debian-10+deb10u2
# 192.168.10.3:22 SSH-2.0-OpenSSH_7.9p1 Debian-10+deb10u2
# 192.168.10.3:22 SSH-2.0-OpenSSH_7.9p1 Debian-10+deb10u2
user@ANSIBLE:/etc/ansible$ ssh-keyscan -H 192.168.10.4 >> ~/.ssh/known_host
# 192.168.10.4:22 SSH-2.0-OpenSSH_7.9p1 Debian-10+deb10u2
# 192.168.10.4:22 SSH-2.0-OpenSSH_7.9p1 Debian-10+deb10u2
# 192.168.10.4:22 SSH-2.0-OpenSSH_7.9p1 Debian-10+deb10u2
user@ANSIBLE:/etc/ansible$
```

Figura 33. Activación de la relación de llaves SSH entre los nodos Cumulus y Ansible
Fuente: *Autor*

Testeamos la conexión de Ansible hacia SPINE y LEAF-1, LEAF-2 con el comando: **ansible -m ping all** (no es necesario estar ubicados dentro del directorio de Ansible):

```
user@ANSIBLE:/etc/ansible/playbooks$ ansible -m ping all
192.168.10.2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
192.168.10.3 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
192.168.10.4 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
user@ANSIBLE:/etc/ansible/playbooks$
```

Figura 34. Ejecución del comando "ansible –m ping all"
Fuente: *Autor*

Se procede a configurar un playbook en Ansible, en donde se definirán las tareas a realizar de forma automatizada en cada equipo. Para este fin se crea un archivo "**vxlan.yml**" dentro del directorio: /etc/ansible/playbooks. Por temas de organización se ha creado una carpeta playbooks que en un proyecto real contendrá todas las tareas, sin embargo, estos ficheros también pueden estar ubicados dentro del path /etc/ansible/. La configuración queda de la siguiente manera:

```
#Configuracion de direccionamiento ip
---
- hosts: SPINE
  tasks:
    - name: Direccionamiento IP SPINE
      nclu:
        commands:
        - add loopback lo ip address 2.2.2.2/32
        - add interface swp2 ip address 12.12.12.2/30
        - add interface swp3 ip address 23.23.23.2/30
        - commit

- hosts: LEAF_1
  tasks:
      - name: Direccionamiento IP LEAF_1
        nclu:
            commands:
            - add loopback lo ip address 1.1.1.1/32
            - add interface swp2 ip address 12.12.12.1/30
            - commit

- hosts: LEAF_2
  tasks:
      - name: Direccionamiento IP LEAF_2
        nclu:
            commands:
            - add loopback lo ip address 3.3.3.3/32
            - add interface swp3 ip address 23.23.23.1/30
            - commit
```

Figura 35. Configuración de direccionamiento IP
Fuente: *Autor*

```
#Configuracion de enrutamiento Underlay(OSPF de una sola area)
- hosts: SPINE
  tasks:
    - name: Configuracion OSPF SPINE
      nclu:
        commands:
        - add ospf router-id 2.2.2.2
        - add loopback lo ospf area 0
        - add interface swp2 ospf area 0
        - add interface swp3 ospf area 0
        - add interface swp2 ospf network broadcast
        - add interface swp3 ospf network broadcast
        - commit

- hosts: LEAF_1
  tasks:
    - name: Configuracion OSPF LEAF_1
      nclu:
        commands:
        - add ospf router-id 1.1.1.1
        - add loopback lo ospf area 0
        - add interface swp2 ospf area 0
        - add interface swp2 ospf network broadcast
        - commit

- hosts: LEAF_2
  tasks:
    - name: Configuracion OSPF LEAF_2
      nclu:
        commands:
        - add loopback lo ospf area 0
        - add interface swp3 ospf area 0
        - add interface swp3 ospf network broadcast
        - add ospf router-id 3.3.3.3
        - commit
```

Figura 36. Configuración de enrutamiento Underlay (OSPF de una sola área)
Fuente: *Autor*

```
#Configuracion de VXLAN
- hosts: SPINE
  tasks:
    - name: Configuracion Service Node Functionality,junto con An
      nclu:
        commands:
        - add lnv service-node source 2.2.2.2
        - add lnv service-node anycast-ip 2.2.2.2
        - commit

- hosts: LEAF_1
  tasks:
    - name: Configuracion VTEP y Service Node IP en Leaf 1
      nclu:
        commands:
        - add loopback lo vxrd-src-ip 1.1.1.1
        - add loopback lo vxrd-svcnode-ip 2.2.2.2
        - commit

- hosts: LEAF_2
  tasks:
    - name: Configuracion VTEP y Service Node IP en Leaf 2
      nclu:
        commands:
        - add loopback lo vxrd-src-ip 3.3.3.3
        - add loopback lo vxrd-svcnode-ip 2.2.2.2
        - commit
```

Figura 37. Configuración de VXLAN
Fuente: *Autor*

```
#Creacion de VLAN10 en los LEAF
- hosts: LEAF_1
  tasks:
    - name: host VLAN configs
      nclu:
        commands:
        - add vlan 10 ip address 10.10.10.1/24
        - add interface swp6 bridge access 10
        - commit

- hosts: LEAF_2
  tasks:
    - name: host VLAN configs
      nclu:
        commands:
        - add vlan 10 ip address 10.10.10.3/24
        - add interface swp6 bridge access 10
        - commit
```

Figura 38. Configuración de VLAN 10 en los LEAF
Fuente: *Autor*

```
#Mapeo de VXLAN con VLAN y VTEP (VNID 1010)
- hosts: LEAF_1
  tasks:
      - name: VLAN-VxLAN Mapping
        nclu:
            commands:
            - add vxlan vni1010 vxlan id 1010
            - add vxlan vni1010 vxlan local-tunnelip 1.1.1.1
            - add vxlan vni1010 vxlan remoteip 3.3.3.3
            - add vxlan vni1010 bridge access 10
            - commit

- hosts: LEAF_2
  tasks:
      - name: VLAN-VxLAN Mapping
        nclu:
            commands:
            - add vxlan vni1010 vxlan id 1010
            - add vxlan vni1010 vxlan local-tunnelip 3.3.3.3
            - add vxlan vni1010 vxlan remoteip 1.1.1.1
            - add vxlan vni1010 bridge access 10
            - commit
```

Figura 39. Configuración del Mapeo de VXLAN con VLAN Y VTEP (VNID 1010)
Fuente: *Autor*

Antes de proceder con la ejecución del playbook, es muy importante inicializar los servicios frr. service (previamente activando OSPF dentro del archivo daemons) y el vxrd.service (activar VXLAN) en los LEAF y SPINE:

```
cumulus@LEAF-2:~$ sudo nano /etc/frr/daemons
[sudo] password for cumulus:
  GNU nano 2.2.6            File: /etc/frr/daemons

#
# ATTENTION:
#
# When activation a daemon at the first time, a config file, even if it is
# empty, has to be present *and* be owned by the user and group "frr", else
# the daemon will not be started by /etc/init.d/frr. The permissions should
# be u=rw,g=r,o=.
# When using "vtysh" such a config file is also needed. It should be owned by
# group "frrvty" and set to ug=rw,o= though. Check /etc/pam.d/frr, too.
#
# The watchfrr daemon is always started. Per default in monitoring-only but
# that can be changed via /etc/frr/daemons.conf.
#
zebra=yes
bgpd=no
ospfd=yes
ospf6d=no
ripd=no
ripngd=no
```

Figura 40. Edición del fichero daemons para activar OSPF
Fuente: *Autor*

Utilizamos los siguientes comandos para inicializar OSPF y VXLAN en los equipos SPINE, LEAF-1 y LEAF-2:

```
cumulus@LEAF-2:~$ sudo systemctl start frr.service
cumulus@LEAF-2:~$ sudo systemctl start vxrd.service
```

Figura 41. Activación de OSPF y VXLAN
Fuente: *Autor*

Se procede a correr el playbook de Ansible con el comando **ansible-playbook vxlan.yml**, lo que nos permitirá monitorear y observar la ejecución de las tareas configuradas:

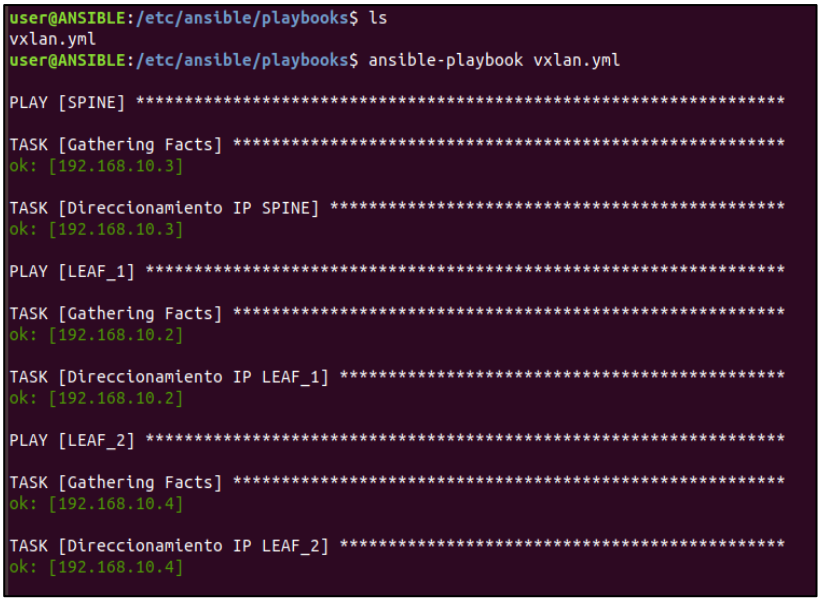


Figura 42. Ejecución del comando "ansible-playbook vxlan.yml"

Fuente: *Autor*

Si todo está bien al finalizar la ejecución se debe mostrar un reporte con OK sin estados de notificación failed.

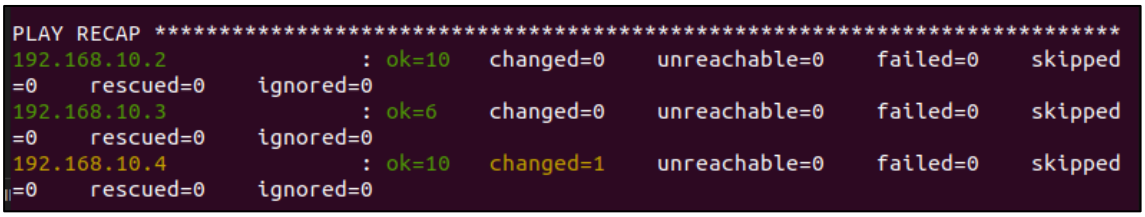


Figura 43. Reporte de ejecución del fichero playbook vxlan.yml

Fuente: *Autor*

Se configura los hosts y hacemos prueba de conectividad de ping de extremo a extremo entre los PC(VPC):

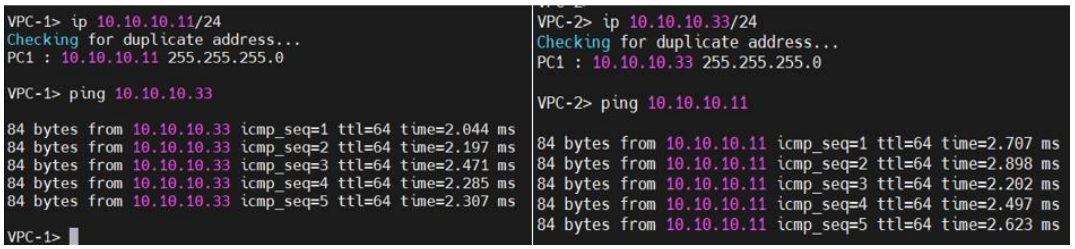


Figura 44. Configuración de direccionamiento ip en las PC y prueba de ping

Fuente: *Autor*

Verificamos los protocolos OSPF y VXLAN en SPINE y LEAF1 respectivamente:

```
cumulus@SPINE:/$ net show route
show ip route
==============
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR,
       > - selected route, * - FIB route

O>* 1.1.1.1/32 [110/100] via 12.12.12.1, swp2, 01:27:15
O   2.2.2.2/32 [110/0] is directly connected, lo, 01:28:20
C>* 2.2.2.2/32 is directly connected, lo, 01:28:21
O>* 3.3.3.3/32 [110/100] via 23.23.23.1, swp3, 00:19:40
O   12.12.12.0/30 [110/100] is directly connected, swp2, 01:28:20
C>* 12.12.12.0/30 is directly connected, swp2, 01:28:21
O   23.23.23.0/30 [110/100] is directly connected, swp3, 01:28:20
C>* 23.23.23.0/30 is directly connected, swp3, 01:28:21
C>* 192.168.10.0/24 is directly connected, eth0, 01:28:21
```

Figura 45. Tabla de enrutamiento del nodo SPINE
Fuente: *Autor*

Comprobamos el aprendizaje de direcciones MAC´s, por ejemplo, se puede ver en el VTEP del LEAF-1:

```
cumulus@LEAF-1:~$ net show bridge macs

VLAN      Master  Interface  MAC                TunnelDest  State      Flags  LastSeen
--------  ------  ---------  -----------------  ----------  ---------  -----  --------
10        bridge  bridge     50:00:00:01:00:06              permanent          00:27:55
10        bridge  swp6       00:50:79:66:68:04                                 00:01:52
10        bridge  vni1010    00:50:79:66:68:05                                 00:03:22
10        bridge  vni1010    50:00:00:02:00:06                                 00:00:23
untagged          vni1010    00:00:00:00:00:00  3.3.3.3     permanent  self   00:27:31
untagged          vni1010    00:50:79:66:68:05  3.3.3.3                self   00:25:31
untagged          vni1010    50:00:00:02:00:06  3.3.3.3                self   00:22:56
untagged  bridge  swp6       50:00:00:01:00:06              permanent          00:27:56
untagged  bridge  vni1010    aa:7f:10:6f:79:e9              permanent          00:27:48
cumulus@LEAF-1:~$
```

Figura 46. Tabla de aprendizaje MAC en LEAF-1
Fuente: *Autor*

En LEAF-1, se ha aprendido la MAC de LEAF-2 mediante el VNI1010, a pesar de estar en otro sector geográfico, separado por otras redes IP. Capturamos el tráfico de red con Wireshark mientras se ejecuta un ping continuo entre vPC1 y vPC2.

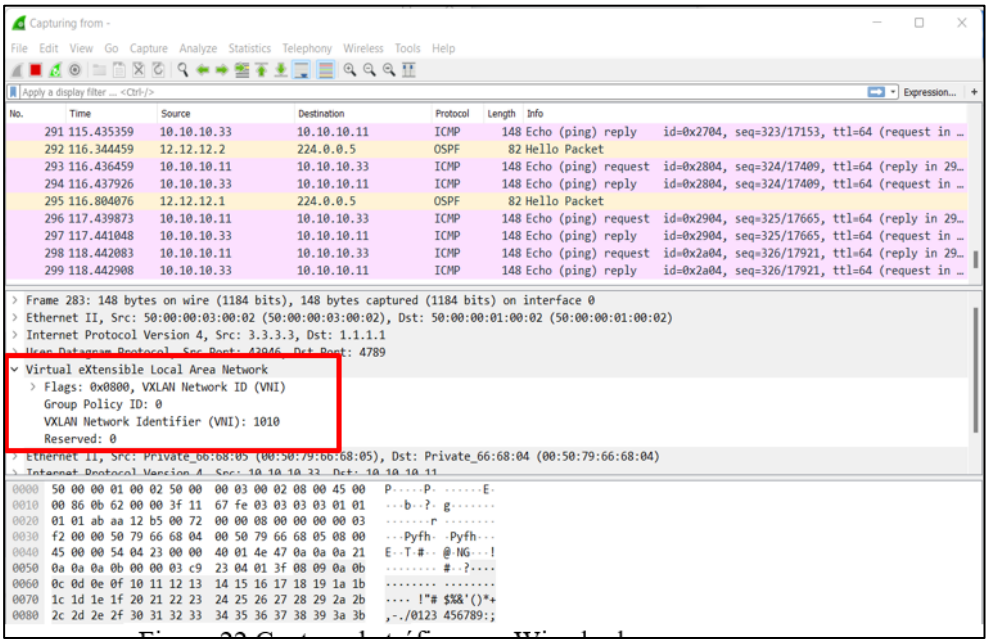


Figura 47. Captura de tráfico con Wireshark
Fuente: *Autor*

En las peticiones ARP o en ICMP, se observará la encapsulación adicional de VXLAN con el VNID 1010.

4.2. Instalación de Git

El primer paso es la instalación de Git en el equipo Linux (Ubuntu 16.04) para el versionamiento de la NaC que implementa la red VXLAN, luego de haber realizado un update:

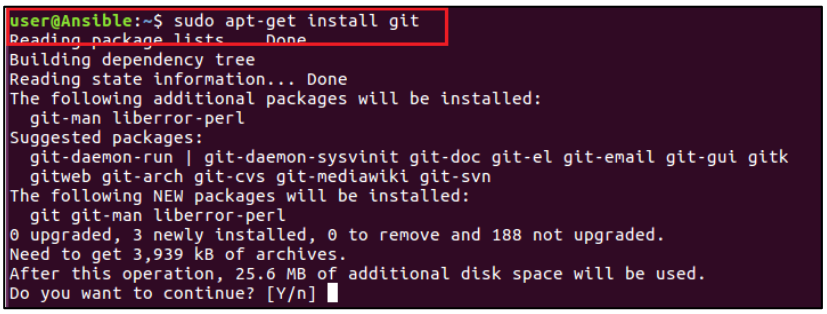


Figura 48. Instalación de Git
Fuente: *Autor*

Es necesario configurar la autenticación ssh entre el nodo Linux y el repositorio de Gitlab, esto con el fin de poder ejecutar los pushes remotos de manera segura. Para lo que copiamos la llave ssh del nodo Linux dentro de la interfaz de Gitlab en la opción de SSH Keys

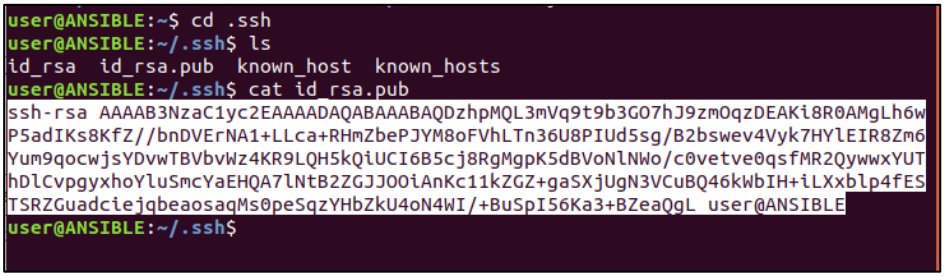


Figura 49. Llave SSH dentro del nodo Linux
Fuente: *Autor*

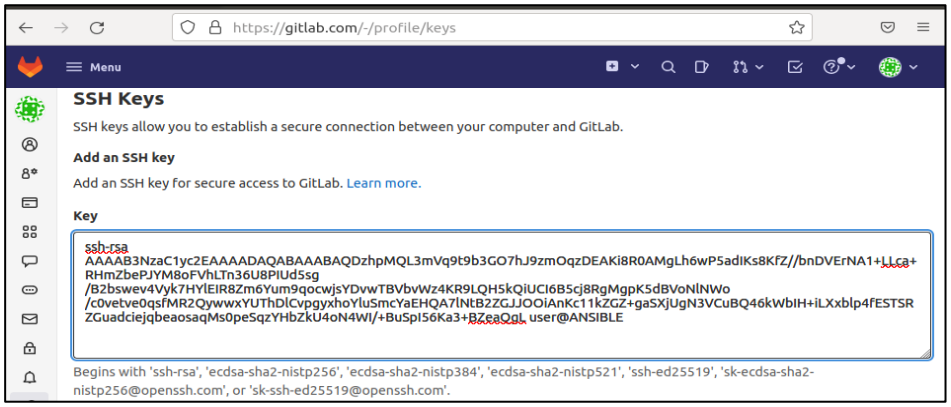


Figura 50. Llave SSH dentro de GitLab
Fuente: *Autor*

Para verificar que funciona la conexión ssh utilizamos el siguiente comando:

```
user@ANSIBLE:~/.ssh$ ssh -t git@gitlab.com
The authenticity of host 'gitlab.com (172.65.251.78)' can't be established.
ECDSA key fingerprint is SHA256:HbW3g8zUjNSksFbqTiUWPWg2Bq1x8xdGUrliXFzSnUw.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'gitlab.com,172.65.251.78' (ECDSA) to the list of kno
wn hosts.
```

Figura 51. Verificación de funcionamiento SSH
Fuente: *Autor*

Es normal que al ser la primera vez que se conecta por ssh, se vea el mensaje de la figura, solo se debe escribir ***yes*** y el nodo queda agregado a la lista de host conocidos.

Realizado este paso, es necesario inicializar el repositorio local de Git con el comando:

$ git init

Luego se debe configurar un nombre de usuario y correo electrónico de la cuenta de GitLab que se asociarán a todas las confirmaciones que se realicen, mediante los siguientes comandos:

$ git config --global user.name "xxxx"
$ git config --global user.email "xxxx"

Agregamos a Git los ficheros que se encuentran dentro del directorio /etc/ansible:

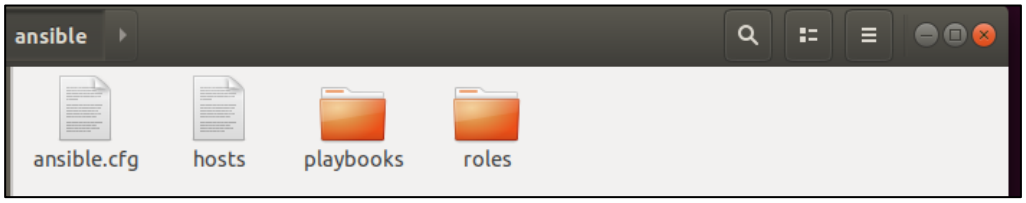


Figura 52. Directorio de instalación de Ansible
Fuente: *Autor*

```
root@ANSIBLE:/etc/ansible# git add .
root@ANSIBLE:/etc/ansible# git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   ansible.cfg
        new file:   hosts
        new file:   playbooks/vxlan.yml

root@ANSIBLE:/etc/ansible# git commit -m "Primera version de la red"
[master (root-commit) 5b6f547] Primera version de la red
 3 files changed, 271 insertions(+)
 create mode 100644 ansible.cfg
 create mode 100644 hosts
 create mode 100644 playbooks/vxlan.yml
root@ANSIBLE:/etc/ansible#
```

Figura 53. Subida a Git del directorio Ansible
Figura: *Autor*

Se sube al repositorio remoto en Gitlab los archivos que están en local en Git:

```
root@ANSIBLE:/etc/ansible# git branch -m main
root@ANSIBLE:/etc/ansible# git remote add origin git@gitlab.com:juancafriend22/t
esisnetdevops.git
root@ANSIBLE:/etc/ansible# git push -u origin main
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 2.48 KiB | 2.48 MiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To gitlab.com:juancafriend22/tesisnetdevops.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
root@ANSIBLE:/etc/ansible#
```

Figura 54. Subida a repositorio remoto
Fuente: *Autor*

Dentro del servidor web de Gitlab se crea el repositorio con la infraestructura de la red VXLAN:

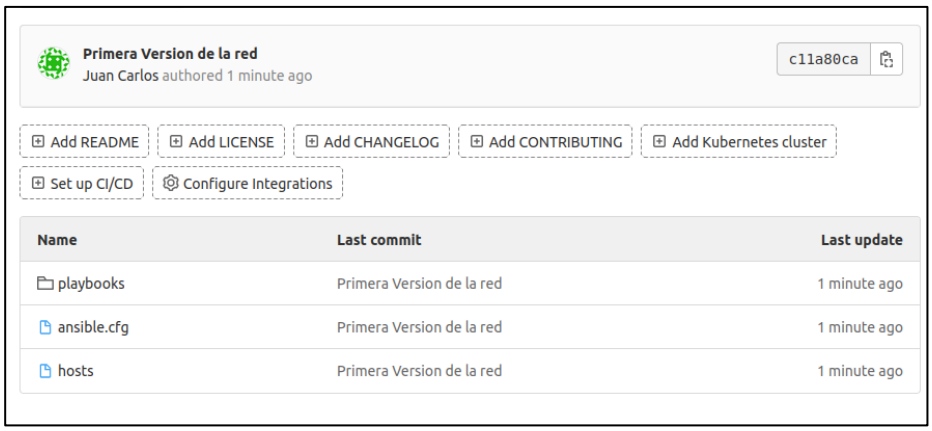


Figura 55. Repositorio remoto en GitLab con la NaC de VXLAN
Fuente: *Autor*

4.3. Configuración de Gitlab CI/CD

Para poder utilizar la herramienta Gitlab CI/CD necesitamos instalar Gitlab Runner, que es la instancia que compila los jobs definidos dentro de los pipelines.

```
user@ANSIBLE:~/ansible/playbooks$ sudo curl -L --output /usr/local/bin/gitlab-ru
nner "https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-ru
nner-linux-amd64"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 48.1M  100 48.1M    0     0  40.6M      0  0:00:01  0:00:01 --:--:-- 40.6M
user@ANSIBLE:~/ansible/playbooks$ sudo chmod +x /usr/local/bin/gitlab-runner
user@ANSIBLE:~/ansible/playbooks$ sudo useradd --comment 'GitLab Runner' --creat
e-home gitlab-runner --shell /bin/bash
user@ANSIBLE:~/ansible/playbooks$
user@ANSIBLE:~/ansible/playbooks$ sudo gitlab-runner install --user=gitlab-runne
r --working-directory=/home/gitlab-runner
Runtime platform                                    arch=amd64 os=linux pid=5296
 revision=76984217 version=15.1.0
user@ANSIBLE:~/ansible/playbooks$ sudo gitlab-runner start
Runtime platform                                    arch=amd64 os=linux pid=5351
 revision=76984217 version=15.1.0
user@ANSIBLE:~/ansible/playbooks$
user@ANSIBLE:~/ansible/playbooks$ sudo gitlab-runner status
Runtime platform                                    arch=amd64 os=linux pid=5372
 revision=76984217 version=15.1.0
gitlab-runner: Service is running
user@ANSIBLE:~/ansible/playbooks$
```

Figura 56. Instalación de Gitlab Runner en nodo Linux
Fuente: *Autor*

Es también necesario obtener el token del Runner dentro del repositorio web remoto de Gitlab, mismo que luego se configurará en el equipo Linux:

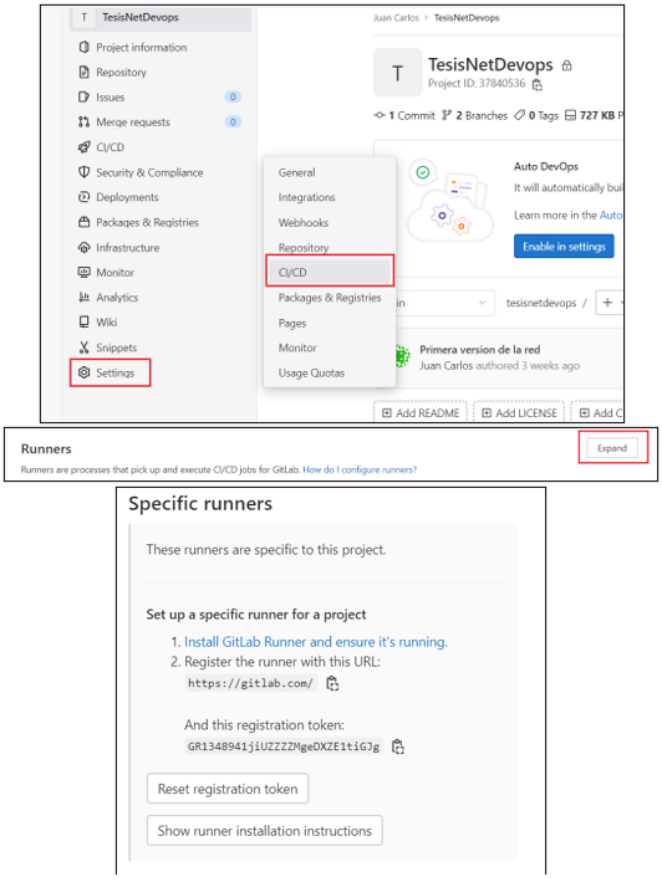


Figura 57. Obtención del token del Runner
Fuente: *Autor*

Se procede a registrar el Runner dentro del nodo Linux, con el token, descripción, tags, así como el tipo de ejecutor en este caso Shell y la configuración SSH.

```
root@ANSIBLE:/etc/gitlab-runner# gitlab-runner register
Runtime platform                                    arch=amd64 os=linux pid=17822 revisi
on=76984217 version=15.1.0
Running in system-mode.

Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com/
Enter the registration token:
GR1348941jiUZZZZMgeDXZE1tiGJg
Enter a description for the runner:
[ANSIBLE]: runner ci/cd
Enter tags for the runner (comma-separated):
deploy,build,demo,test,ansible
Enter optional maintenance note for the runner:

Registering runner... succeeded                     runner=GR1348941jiUZZZZM
Enter an executor: docker-ssh, parallels, virtualbox, docker+machine, custom, docker, do
cker-ssh+machine, kubernetes, shell, ssh:
Enter the SSH server address (for example, my.server.com):
https://gitlab.com/
Enter the SSH server port (for example, 22):
20
Enter the SSH user (for example, root):
root
Enter the SSH password (for example, docker.io):
root
Enter the path to the SSH identity file (for example, /home/user/.ssh/id_rsa):

Runner registered successfully. Feel free to start it, but if it's running already the c
onfig should be automatically reloaded!
root@ANSIBLE:/etc/gitlab-runner#
```

Figura 58. Registro del GitLab Runner
Fuente: *Autor*

De estar correctamente configurado el runner, dentro de la web de Gitlab verificaremos inicializado el mismo y activo para el proyecto.

Figura 59. Verificación de Runner activo dentro de GitLab
Fuente: *Autor*

CAPÍTULO V: ANÁLISIS DE RESULTADOS

El tener subida la NaC a un repositorio de Git, nos permite tener versionamiento de código de la infraestructura de la red, tal cual, como si estuviéramos trabajando con código de software. En tal virtud, se puede crear una rama dedicada a la ejecución de pruebas donde todo cambio se implemente primeramente en un entorno emulado (Continuos Development) antes de pasar al entorno de producción.

```
root@ANSIBLE:/etc/ansible# git branch
* main
root@ANSIBLE:/etc/ansible#
root@ANSIBLE:/etc/ansible# git checkout -b pruebas
Switched to a new branch 'pruebas'
root@ANSIBLE:/etc/ansible# git branch
  main
* pruebas
root@ANSIBLE:/etc/ansible# git merge main -m "Creacion de la rama pruebas con el
 codigo base de la red VXLAN inicial"
Already up to date.
root@ANSIBLE:/etc/ansible#
root@ANSIBLE:/etc/ansible#
```

Figura 60. Creación e integración de la rama "pruebas" dentro del repositorio Git
Fuente: *Autor*

Ahora la rama pruebas está actualizada con todos los cambios en la rama principal main y sobre la misma se puede trabajar en el desarrollo de nuevos requerimientos o testing.

Al utilizar una metodología NetDevOps, la cultura de colaboración es de vital importancia, de ahí la necesidad de trabajar con repositorios remotos de Git como GitLab, donde todos los miembros del equipo encargado de administrar la red puedan participar en el desarrollo de funcionalidades o cambios a la red, de tal manera que estos se puedan integrar de manera continua. Para subir al repositorio remoto la rama pruebas y el contenido asociado al **merge** ejecutado se realiza lo siguiente:

```
root@ANSIBLE:/etc/ansible# git push -u origin pruebas
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: To create a merge request for pruebas, visit:
remote:   https://gitlab.com/juancafriend22/tesisnetdevops/-/merge_requests/new?
merge_request%5Bsource_branch%5D=pruebas
remote:
To gitlab.com:juancafriend22/tesisnetdevops.git
 * [new branch]      pruebas -> pruebas
Branch 'pruebas' set up to track remote branch 'pruebas' from 'origin'.
root@ANSIBLE:/etc/ansible#
root@ANSIBLE:/etc/ansible#
root@ANSIBLE:/etc/ansible#
```

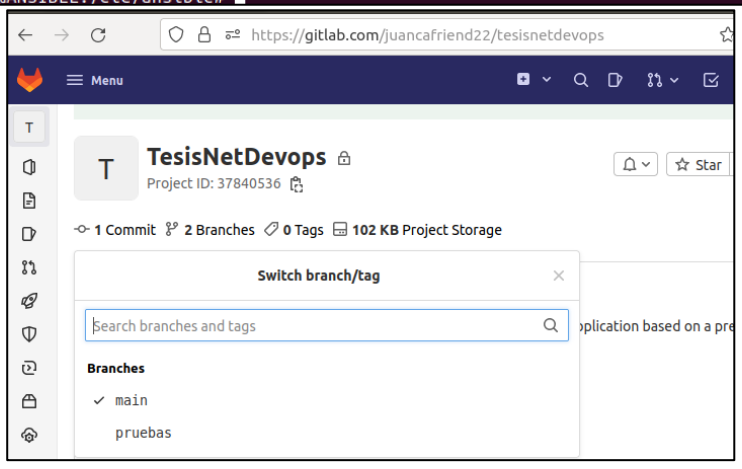


Figura 61. Subida a repositorio remoto de la rama de pruebas
Fuente: *Autor*

El resultado de trabajar con Git como herramienta de versionamiento es que brinda la posibilidad, de ser necesario, de regresar un commit anterior de versionamiento de la NaC. Con la ejecución del comando **git log –oneline** se puede acceder al historial de commits.

```
user@ANSIBLE:/etc/ansible$ git log --oneline
1adab26 (HEAD -> pruebas, origin/pruebas) Primera version de la red
124d097 modificacion del VNI
4203349 Update .gitlab-ci.yml
40a262a merge
970adef Fusion local y remota
7ba4244 Actualizacion .gitlab-ci.yml
dd56976 actualizacion de directorio
b55a351 Se agrega un DNS
98f260f 7to intento
3bf0a9b 6ta prueba
0ebaf7e 5to intento
15dac7f tercer intento
5eab8f7 segunda prueba
5ecfa06 actualizacion de DNS
625a3fa actualizacion de DNS
1b71692 Actualizacion de DNS
558d08a Se agregan los DNS a los equipos Spine y Leaf
9c2d3a8 Update .gitlab-ci.yml file
d16e1a2 Update .gitlab-ci.yml file
8528a2c Configuracion de DNS en equipos
5b6f547 (origin/main, main) Primera version de la red
user@ANSIBLE:/etc/ansible$
```

Figura 62. Historial de commits del repositorio
Fuente: *Autor*

La red moderna VXLAN Spine-Leaf desarrollada, al tener un enfoque NetDevOps, automatiza los requerimientos de cambio en la infraestructura por medio de pipelines CI-CD implementados en GitLab CI-CD. Por ejemplo, en un escenario real, se puede tener un requerimiento donde sea necesario agregar una VLAN 20 adicional:

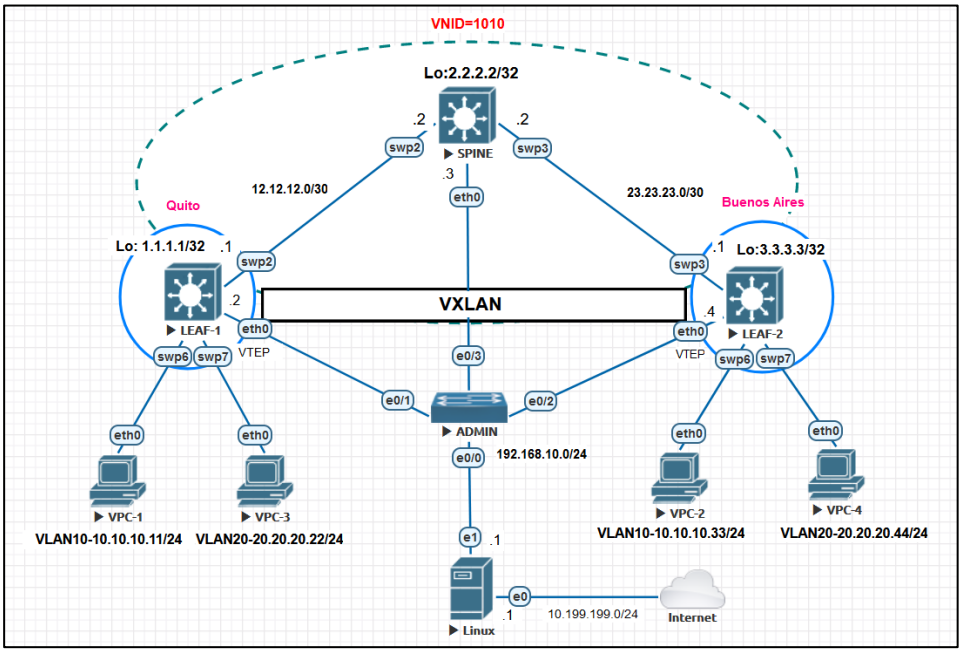


Figura 63. Topología VXLAN Spine-Leaf con dos VLAN
Fuente: *Autor*

El ingeniero de red, acude al repositorio de pruebas de Git y edita la NaC, a fin de añadir la VLAN 20 a la topología, no es necesario ingresar a la CLI de cada equipo de red, puesto que utiliza Ansible que por medio de sus API´s nos permite la manipulación masiva a los hosts que sean necesarios configurar para este requerimiento.

```
#Creacion de VLAN10 y VLAN20 en los LEAF
- hosts: LEAF_1
  tasks:
      - name: host VLAN configs
        nclu:
            commands:
            - add vlan 10 ip address 10.10.10.1/24
            - add interface swp6 bridge access 10
            - add vlan 20 ip address 20.20.20.1/24
            - add interface swp7 bridge access 20
            - commit

- hosts: LEAF_2
  tasks:
      - name: host VLAN configs
        nclu:
            commands:
            - add vlan 10 ip address 10.10.10.3/24
            - add interface swp6 bridge access 10
            - add vlan 20 ip address 20.20.20.3/24
            - add interface swp7 bridge access 20
            - commit
```

Figura 64. Creación de VLAN 10 Y VLAN 20
Fuente: *Autor*

Se mapea un nuevo VNI, en este caso el 2020 y se lo asocia a la VLAN 20 para que permita el flujo de datos a través de la red overlay de VXLAN:

```
#Mapeo de VXLAN con VLAN y VTEP (VNID 1010-VNID 2020)
- hosts: LEAF_1
  tasks:
      - name: VLAN-VxLAN Mapping
        nclu:
            commands:
            - add vxlan vni1010 vxlan id 1010
            - add vxlan vni1010 vxlan local-tunnelip 1.1.1.1
            - add vxlan vni1010 vxlan remoteip 3.3.3.3
            - add vxlan vni1010 bridge access 10
            - commit
            - add vxlan vni2020 vxlan id 2020
            - add vxlan vni2020 vxlan local-tunnelip 1.1.1.1
            - add vxlan vni2020 vxlan remoteip 3.3.3.3
            - add vxlan vni2020 bridge access 20
            - commit

- hosts: LEAF_2
  tasks:
      - name: VLAN-VxLAN Mapping
        nclu:
            commands:
            - add vxlan vni1010 vxlan id 1010
            - add vxlan vni1010 vxlan local-tunnelip 3.3.3.3
            - add vxlan vni1010 vxlan remoteip 1.1.1.1
            - add vxlan vni1010 bridge access 10
            - commit
            - add vxlan vni2020 vxlan id 2020
            - add vxlan vni2020 vxlan local-tunnelip 3.3.3.3
            - add vxlan vni2020 vxlan remoteip 1.1.1.1
            - add vxlan vni2020 bridge access 20
            - commit
```

Figura 65. Mapeo de VNI 2020 en la red Overlay
Fuente: *Autor*

Terminada la implementación de código, se usa Git y Gitlab para actualizar el repositorio y

tener control de versionamiento de las nuevas funcionalidades implementadas y que, al trabajar colaborativamente, los demás miembros puedan verificar el usuario que desarrolló los nuevos cambios en la red o que puedan participar en la implementación de los mismos. Se ejecuta un comando **git add playbooks/vxlan.yml** para agregar a etapa de stage(preparación) los cambios realizados y luego un **git commit** para guardar los mismos en el repositorio local.

```
root@ANSIBLE:/etc/ansible# git status
On branch pruebas
Your branch is up to date with 'origin/pruebas'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   playbooks/vxlan.yml

no changes added to commit (use "git add" and/or "git commit -a")
root@ANSIBLE:/etc/ansible#
root@ANSIBLE:/etc/ansible#
root@ANSIBLE:/etc/ansible# git add playbooks/vxlan.yml
root@ANSIBLE:/etc/ansible# git commit -m "Se adiciona VLAN 20 y VNI 2020"
[pruebas 52eaeac] Se adiciona VLAN 20 y VNI 2020
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Figura 66. Actualización de repositorio Git con las nuevas modificaciones
Fuente: *Autor*

Posteriormente se sube a la rama de pruebas del repositorio los cambios guardados a nivel local, lo que ejecutará automáticamente debido a la configuración que se realiza en el archivo .**gitlab-ci.yml** la ejecución del pipeline.

```
root@ANSIBLE:/etc/ansible# git push -u origin pruebas
Counting objects: 4, done.
Delta compression using up to 6 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 352 bytes | 352.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0)
remote:
remote: To create a merge request for pruebas, visit:
remote:   https://gitlab.com/juancafriend22/tesisnetdevops/-/merge_requests/new?
merge_request%5Bsource_branch%5D=pruebas
remote:
To gitlab.com:juancafriend22/tesisnetdevops.git
   4362075..52eaeac  pruebas -> pruebas
Branch 'pruebas' set up to track remote branch 'pruebas' from 'origin'.
root@ANSIBLE:/etc/ansible#
```

Figura 67. Subida a repositorio de pruebas remotos de los cambios
Fuente: *Autor*

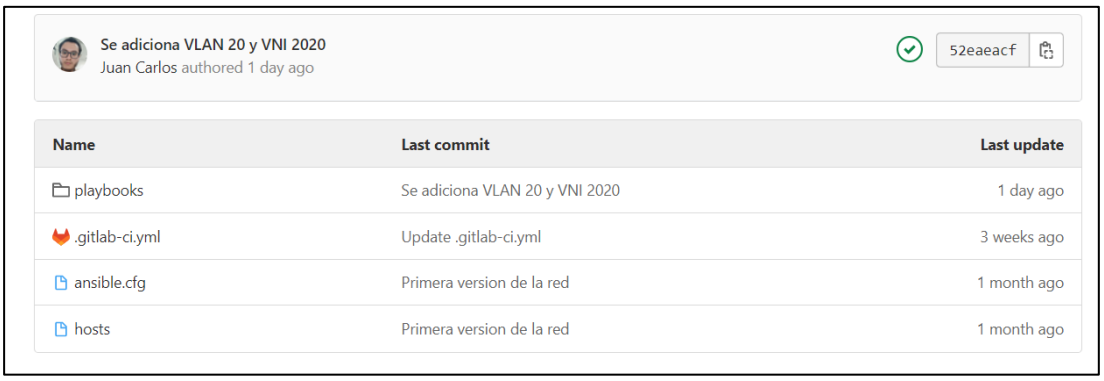


Figura 68. Repositorio remoto en Gitlab
Fuente: *Autor*

Se tiene dos ***jobs*** que es como Gitlab CI/CD denomina a las fases del pipeline:

- **Test:** es dónde se realiza comandos de ping hacia los hosts para verificar conectividad y una revisión de sintaxis del playbook que se debe ejecutar y
- **Deploy:** es donde se despliega la ejecución del playbook para el aprovisionamiento del nuevo código dentro de los equipos que conforman la red de pruebas emulada.

.gitlab-ci.yml 572 bytes

```
workflow:
    rules:
    - if: $CI_PIPELINE_SOURCE == "push"
validation:
    stage: test
    tags:
        - ansible
    script:
        - echo "Comprobacion de ping a equipos de red"
        - 'ansible -m ping all'
        - echo "Comprobacion de sintaxis del playbook"
        - 'ansible-playbook --syntax-check playbooks/vxlan.yml'
    environment: pruebas
deploy_to_test:
    stage: deploy
    tags:
        - ansible
    script:
        - echo "Ejecucion del playbook"
        - 'ansible-playbook playbooks/vxlan.yml'
    environment: pruebas
   # when: manual
```

Figura 69. Fichero ". gitlab-ci.yml"
Fuente: *Autor*

Se ejecutan secuencialmente estas fases dentro del pipeline cuyo tiempo de ejecución demora 56 segundos, lo que permite determinar que la automatización de una red con Ansible y Gitlab CI/CD es una buena opción para la implementación de programabilidad en NetDevOps.

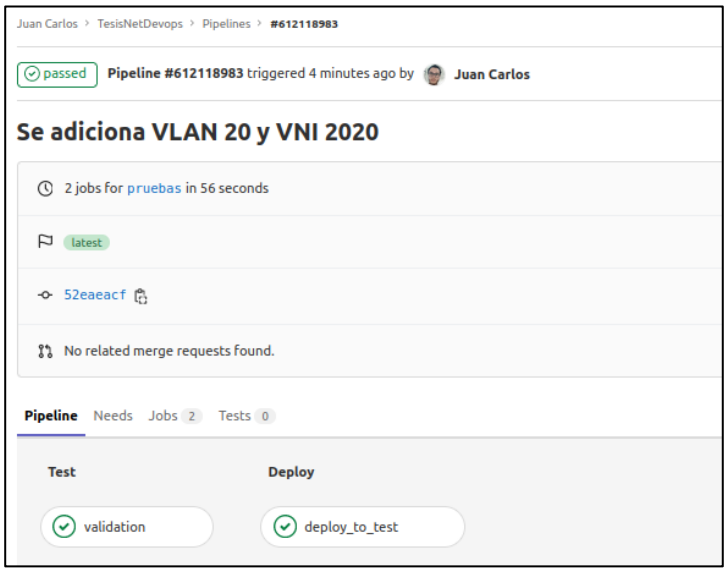


Figura 70. Reporte de tiempo de ejecución del pipeline
Fuente: *Autor*

La herramienta GitLab CI/CD integra una consola donde se puede verificar la ejecución de cada uno de los ***jobs***.

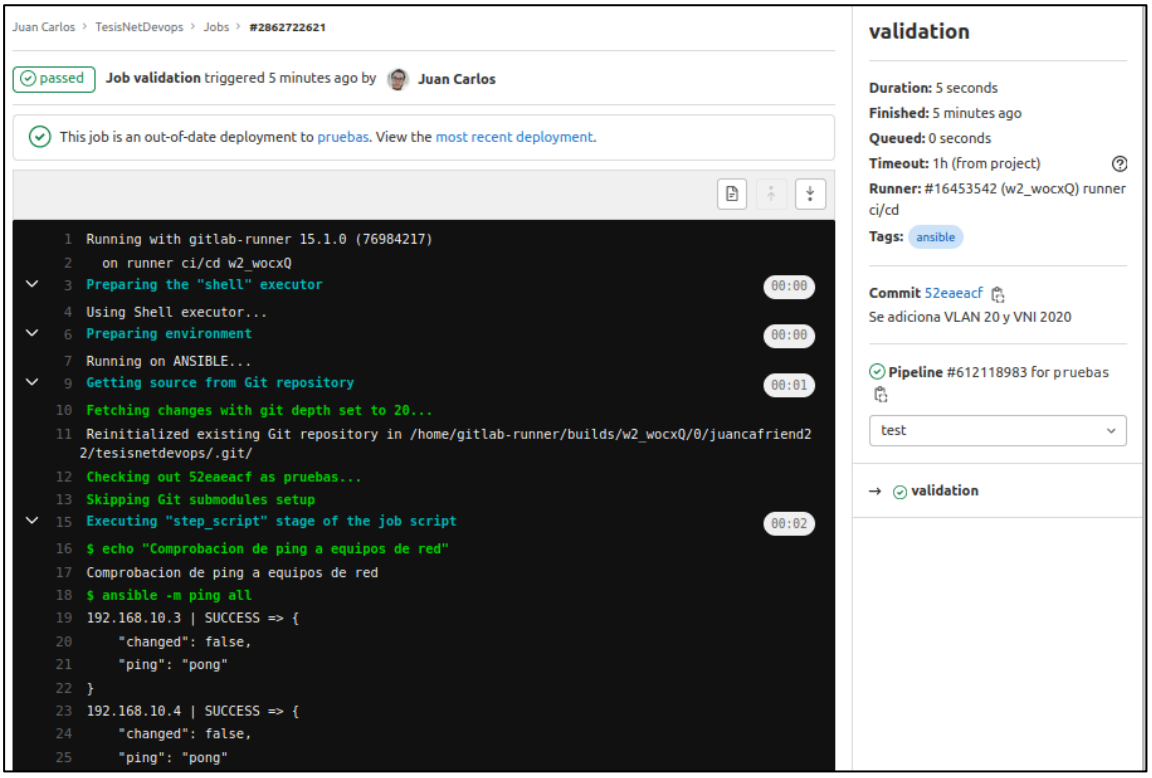


Figura 71. Ejecución del job test
Fuente: *Auto*

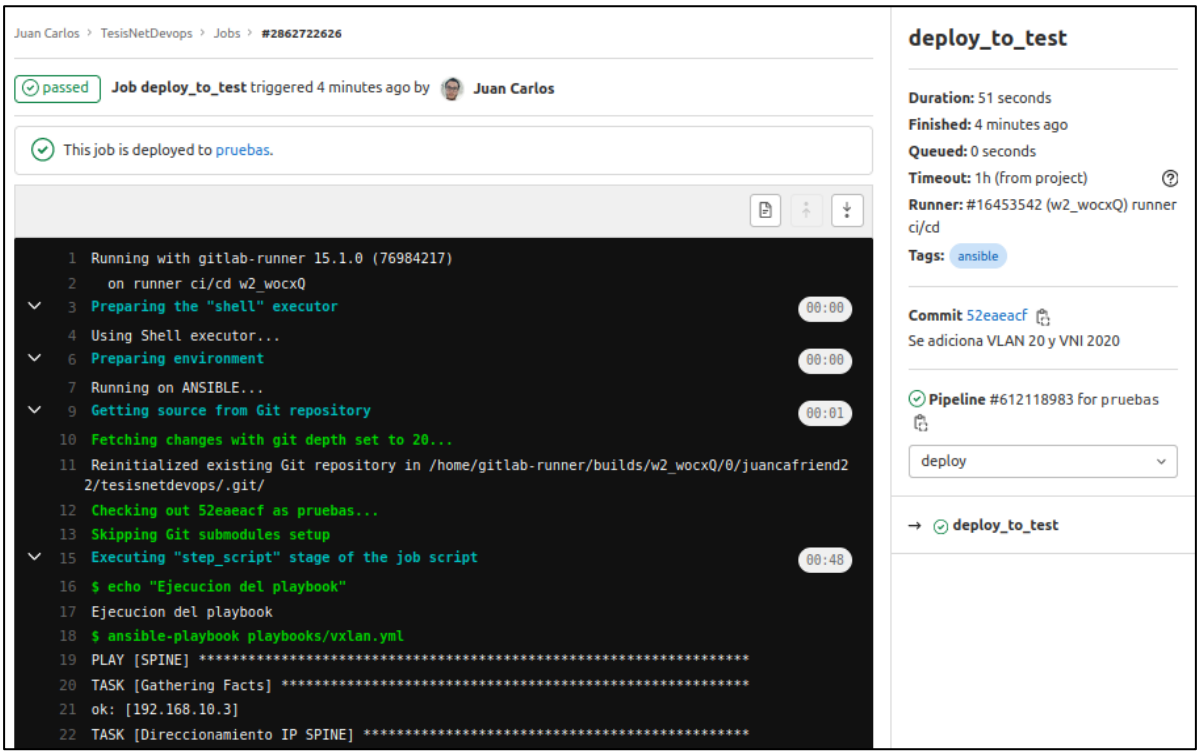


Figura 72. Ejecución del job deploy
Fuente: *Autor*

Como la ejecución del pipeline fue exitosa podemos validar dentro del emulador EVE-NG que se tiene ya implementada la nueva VLAN 20, habiendo conexión entre los hosts que se ubican en esta VLAN, pero en diferente LEAF.

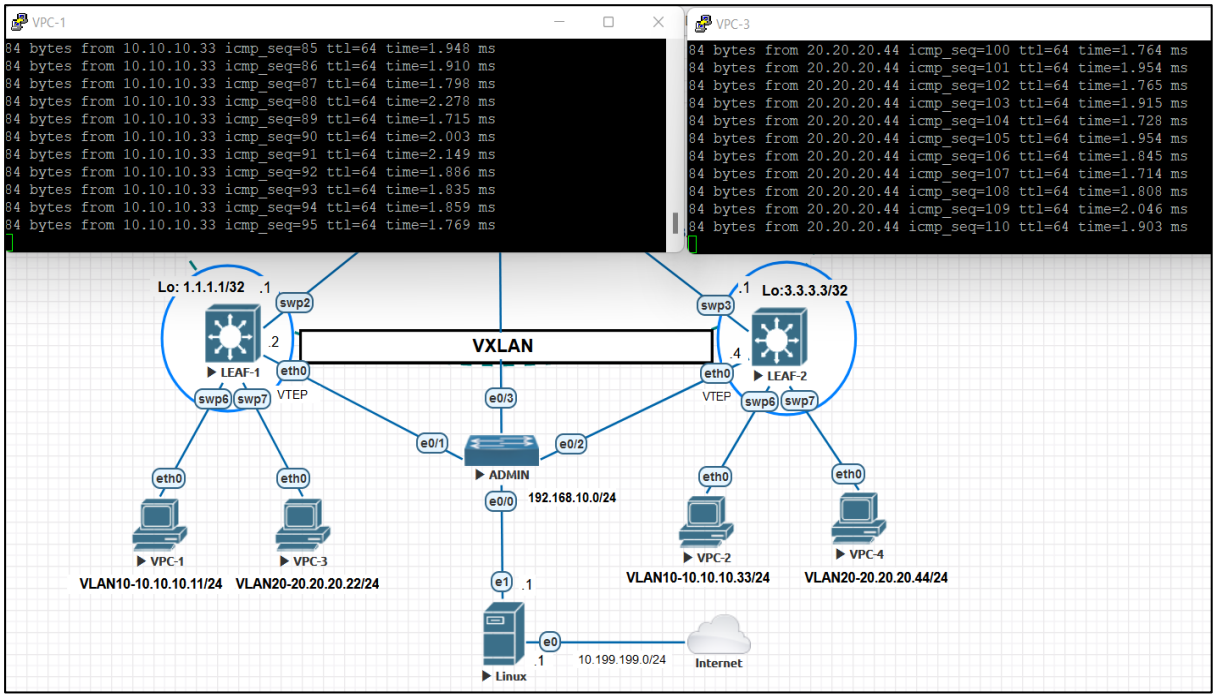


Figura 73. Verificación de ping entre equipos de VLAN 10 y 20
Fuente: *Autor*

Dentro del equipo LEAF-2, se puede visualizar mediante el comando **net show configuration interface,** que se tiene implementada la configuración de la nueva VLAN 20, así como el VNI 2020 y la tunelización.

```
interface vlan20
  address 20.20.20.3/24
  vlan-id 20
  vlan-raw-device bridge

interface vni1010
  bridge-access 10
  mstpctl-bpduguard yes
  mstpctl-portbpdufilter yes
  vxlan-id 1010
  vxlan-local-tunnelip 3.3.3.3
  vxlan-remoteip 1.1.1.1

interface vni2020
  bridge-access 20
  mstpctl-bpduguard yes
  mstpctl-portbpdufilter yes
  vxlan-id 2020
  vxlan-local-tunnelip 3.3.3.3
  vxlan-remoteip 1.1.1.1
```

Figura 74. Visualización de la configuración en la interface del Leaf-2
Fuente: *Autor*

Se realiza una captura de paquetes por medio de wireshark, donde se puede constatar que los mismos están llevando VXLAN en sus cabeceras tanto del VNI 1010 como del VNI 2020.

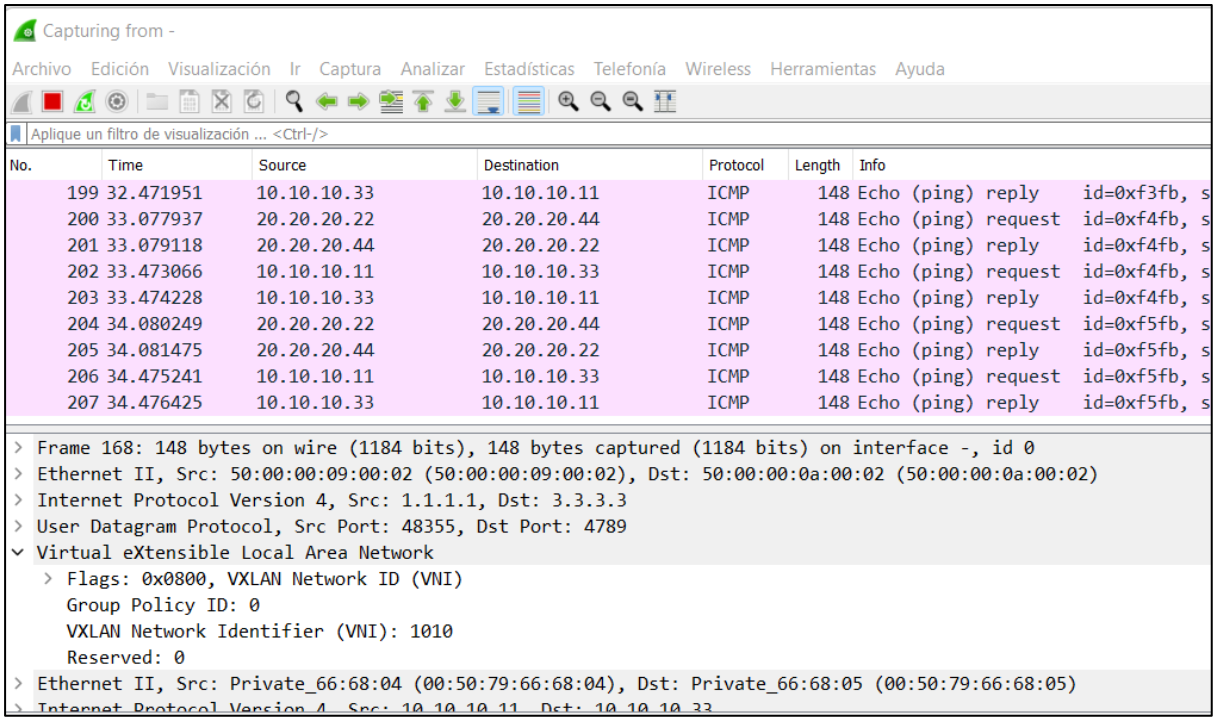


Figura 75. Captura de paquetes con VNI 1010 en WireShark
Fuente: *Autor*

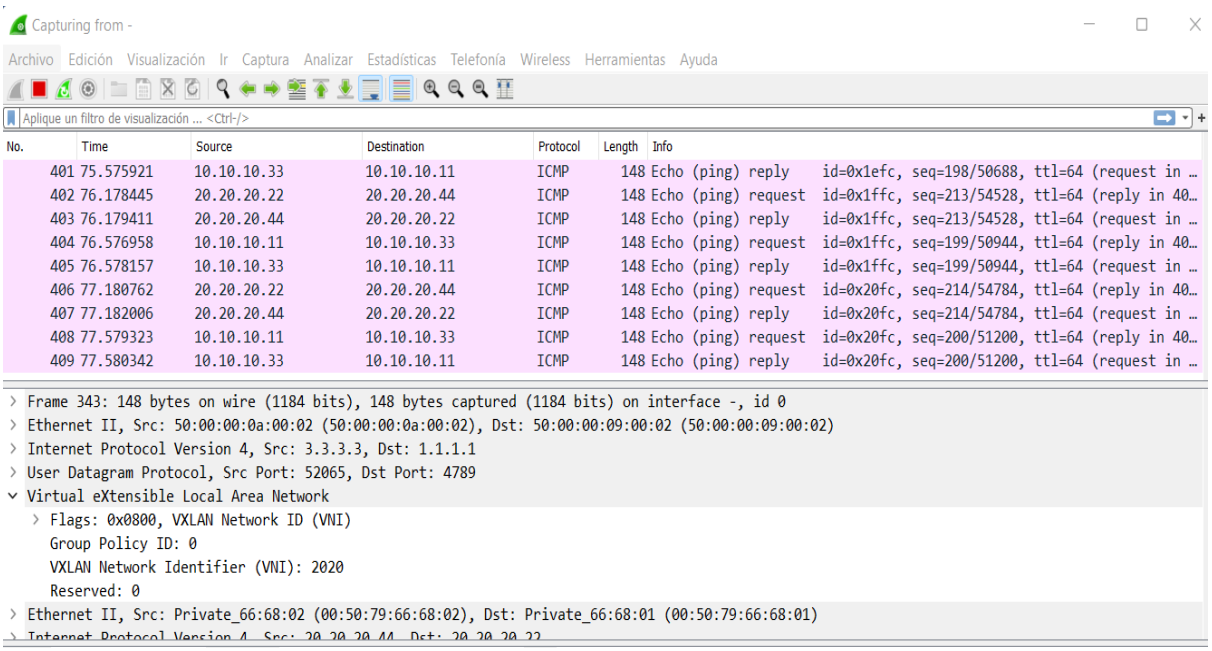


Figura 76. Captura de paquetes con VNI 2020 en WireShark
Fuente: *Autor*

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

Una vez culminado el presente trabajo de investigación, se concluye que, la emulación de una red programática de nueva generación de tipo VXLAN implementada de modo programático a través de API Ansible, con control de versionamiento por medio de GIT, un servidor para la ejecución de pipelines como GitLab CI/CD y todo esto bajo una metodología NetDevops es totalmente factible, puesto que facilita y optimiza el trabajo del ingeniero de red, por cuanto se simplifica la ejecución de tareas para la configuración de la red, se reducen los puntos de fallo y se promueve el trabajo colaborativo con los demás miembros del equipo, pasando de una cultura de miedo al cambio, a una donde el cambio es parte fundamental y continua de los procesos de gestión de la red.

Luego de un proceso de análisis de los encabezados y la encapsulación de VXLAN, se logró la conectividad de la red, pudiendo constatar que con esta tecnología de tunelización overlay es posible unir VTPE's entre sí para que puedan trabajar en la misma subred, a pesar de que sus dominios de capa 2 estén separados físicamente por uno dispositivo de capa 3; permitiendo de este modo resolver problemas de escalabilidad en las infraestructuras modernas y de Data Centers.

Se ha podido evidenciar y concluir que trabajar con el paradigma NetDevOps implica que todo lo que se desarrolla, integra y despliega en ciclos CI/CD, debe pasar necesariamente a un entorno denominado "prueba de concepto" en donde se emula y prueba la red, lo que nos permitirá tener una aproximación muy cercana y real al comportamiento que se tendría en el ambiente en producción. Específicamente en este trabajo, el uso de una plataforma Cloud para el despliegue de la prueba de concepto desarrollada, ha permitido evidenciar su potencialidad y flexibilidad de uso, ya que las emulaciones consumen un significativo uso de RAM y procesamiento, lo cual es normalmente una limitante del uso de entornos locales por la alta inversión que requieren, sin embargo, con Google Cloud se ha podido desplegar una instancia con altas prestaciones de hardware por medio de una cuenta gratuita de prueba de tres meses, lo que ha posibilitado dar viabilidad a la ejecución del presente trabajo.

Una vez realizada la PoC (Proof of Concept) se puede determinar que el uso del control de versionamiento de código por medio de Git en la NaC (Network as Code) es parte esencial de la cuarta generación de las redes conocida como "Era programable", ya que esta práctica es la base fundamental de la integración de la infraestructura, convirtiendo a los repositorios en la fuente de verdad donde se almacena toda la configuración de la red. Sin embargo, Git, trabaja de manera local y los ingenieros de redes nunca trabajan solos, sino en equipos colaborativos por ello los repositorios web remotos cobran vital importancia en el versionamiento, adicional a que permiten el despliegue automatizado en pipelines del flujo CI/CD hacia los ambientes de pruebas o en producción, acelerando los procesos de implementación. En ese sentido, el uso de GitLab CI/CD como herramienta para la ejecución de pipelines siguiendo la metodología NetDevOps supone una enorme ventaja de funcionalidad sobre otras alternativas, esto por cuanto viene integrada dentro de la misma suite donde está el repositorio de código, no siendo necesaria una integración adicional externa como sería en el caso de Jenkins, además todo el conjunto de herramientas que trae consigo GitLab están completamente orientadas a DevOps.

Si bien, la programabilidad y automatización de la red son mejoras traídas al Networking por medio de un conjunto muy amplio de herramientas que se acoplan a todas las fases de la

metodología DevOps y que en el presente trabajo investigativo se ha hecho uso de algunas de ellas como Git, Ansible, GitLab, EVE-NG, hay que considerar que estas, solo son unas de las tantas herramientas, eso sí indispensables, que nos permiten trabajar bajo este paradigma, pero usar solo estas herramientas no es NetDevops.

RECOMENDACIONES

Este proyecto tiene como una de sus finalidades incentivar el estudio y aplicabilidad del NetDevOps en el Networking por todas las ventajas anteriormente expuestas y si bien en países desarrollados existe ya un uso arraigado en su implementación, a nivel de Latinoamérica es un campo muy poco explorado y utilizado, de ahí que sea necesario seguir profundizando en este tema a nivel práctico a fin de promover a nivel educativo y empresarial la aplicabilidad de este nuevo paradigma que no es solo una capacidad de TI sino un proceso esencial de negocio por cuanto permite obtener un retorno significativo de la inversión.

Al manejar la infraestructura como código, es normal que el código asociado al desarrollo de la red tenga que sufrir variaciones ya sea porque aumento el tamaño de la red o por la necesidad de implementar nuevas funcionalidades, de ahí que sea muy recomendable el uso de herramientas de versionamiento como GitLab, que nos permiten administrar en un repositorio el control de versionamiento de la red y así poder supervisar de una manera óptima los cambios que se realicen en la misma así como acelerar los procesos de implementación por medio de pipelines automatizados.

Se recomienda a los ingenieros de redes, ya sea que implementen una red tradicional o moderna, a adoptar este nuevo enfoque de programabilidad y automatización que tienen las redes, puesto que es el futuro, sin embargo esto implica necesariamente la adquisición de nuevas destrezas, así como a aprender nuevas herramientas y a adoptar una cultura de trabajo llamada NetDevOps, por lo que es muy conveniente obtener certificaciones como la de Cisco Devnet a fin de profundizar en conocimientos teóricos y prácticos.

El trabajar bajo una metodología como NetDevOps involucra personas, que necesariamente deben adoptar una cultura colaborativa pero que, en algunas organizaciones al no estar acostumbrados a trabajar bajo este paradigma, puede generar dificultades en su adaptación, por lo que se sugiere a los líderes de las organizaciones a hacer mucho énfasis en tener al personal comprometido en la consecución de objetivos, así como integrados con los procesos y herramientas de automatización.

ESTUDIOS FUTUROS

Debido a que los entornos de TI, manejan múltiples servidores donde la configuración varía de manera regular, tener playbooks de configuración estáticos no es una opción muy viable, por lo que, de ahí surge la primera propuesta como trabajo futuro, que es la adopción de plantillas Jinja2 con Ansible ya que permite definir variables dinámicas ofreciendo así, flexibilidad en los archivos de configuración.

El siguiente punto planteado es investigar sobre la usabilidad de una herramienta que permita complementar con gestión visual, capacidades de monitorización y una mayor facilidad de automatización a Ansible como es el caso de Ansible AWX.

Finalmente, para futuros casos de estudio basados en este trabajo se puede realizar un análisis comparativo sobre el uso de Ansible, Puppet y Terraform como herramientas de IaC a fin de determinar las potencialidades de cada una de ellas aplicadas al Networking.

REFERENCIAS

Agudo, J. M. (2018). NetDevOps en la Red de la Universidad de Salamanca. *Jornadas Técnicas de RedIRIS*.

Alkharabsheh, K., Crespo, Y., Manso, E., & Taboada, J. A. (2016). Comparación de herramientas de Detección de Design Smells. En J. (Ed.) García Molina (Ed.), *JISBD2016*. SISTEDES. http://hdl.handle.net/11705/JISBD/2016/012

Alvarez, M. (2019). *API REST Para Ingenieros de Redes*. https://codingnetworks.blog/es/que-es-un-api-rest-fundamentos-para-ingenieros-de-redes/

Aruba Networks. (2022). *¿Qué es la arquitectura spine-leaf?* https://www.arubanetworks.com/latam/faq/que-es-la-arquitectura-spine-leaf/

Bednarz, A. (2016, noviembre 18). *Network outages linked to human error, incompatible changes, greater complexity*. https://www.networkworld.com/article/3142838/top-reasons-for-network-downtime.html

Cisco Devnet-Programming Fundamentals. (2020). *Data Formats: Understanding and using JSON, XML and YAML*.

Citrix Systems. (2022). *Citrix ADC 13.1*. https://docs.citrix.com/en-us/citrix-adc/current-release/citrix-adc-13.1.pdf

de la Cruz, M. (2017, junio 9). *Tutorial básico de Git y GitHub*. https://moisesdelacruz.medium.com/tutorial-b%C3%A1sico-de-git-y-github-42e46ff41194

Dominguez-Quintero, L., & Vargas-Lombardo, M. (2021). Herramientas de infraestructura como código: ansible, terrafom, chef, puppet. *I+ D Tecnológico*, *17*(2), 2021.

EVE-NG. (2022). *EVE-The Emulated Virtual Environment for Network, Security and DevOps professionals*. https://www.eve-ng.net/

Geisshirt, Kenneth., Zattin, Emanuele., Voss, Rasmus., & Olsson, Aske. (2018). *Git Version Control Cookbook : Leverage Version Control to Transform Your Development Workflow and Boost Productivity, 2nd Edition.* Packt Publishing Ltd.

Genius IT Training. (2018, julio 27). *DevOps: La Evolución de IT para la entrega continua*. https://geniusitt.com/blog/devops-la-evolucion-de-it/

Gitlab. (2022). *CI/CD concepts*. https://docs.gitlab.com/ee/ci/introduction/

Gomez, J. (2021, mayo 11). *Hands-on with NetDevOps*. https://github.com/juliogomez/netdevops#netdevops

Google Cloud. (2022). *Descripción general de Google Cloud | Overview*. https://cloud.google.com/docs/overview?hl=es-419

Jin, B., Sahni, S., & Shevat, A. (2018). *Designing Web APIs BUILDING APIS THAT DEVELOPERS LOVE*. O'Reilly Media, Inc.

JSON. (s/f). Recuperado el 10 de abril de 2022, a partir de https://www.json.org/json-en.html

Krishnan, S. P. T., & Gonzalez, J. L. U. (2015). Building Your Next Big Thing with Google Cloud Platform. En *Building Your Next Big Thing with Google Cloud Platform*. Apress Berkeley, CA. https://doi.org/https://doi.org/10.1007/978-1-4842-1004-8

Kumar, R. (2019, julio 4). *Understanding Ansible Architecture using diagram*. https://www.devopsschool.com/blog/understanding-ansible-architecture-using-diagram/

Morris Kief. (2021). *Infrastructure as Code Dynamic Systems for the Cloud Age*. O'Reilly Media.

Naranjo, E. F., & Salazar, G. D. (2018). Underlay and overlay networks: The approach to solve addressing and segmentation problems in the new networking era: VXLAN encapsulation with Cisco and open source networks. *2017 IEEE 2nd Ecuador Technical Chapters Meeting, ETCM 2017*, *2017-January*, 1–6. https://doi.org/10.1109/ETCM.2017.8247505

Patni, S. (2017). Pro RESTful APIs: Design. En *Build and Integrate with REST, JSON, XML and JAX-RS*. Springer.

Pavón, L. (2020, mayo 19). *NetDevOps: la transformación necesaria del ingeniero de red tradicional*. https://empresas.blogthinkbig.com/netdevops-nuevo-ingeniero-de-red/

Pinto, I., & Chauhan, S. (2022, julio 19). *NetDevOps: A modern approach to AWS networking deployments*. https://aws.amazon.com/es/blogs/networking-and-content-delivery/netdevops-a-

modern-approach-to-aws-networking-deployments/
Python Wiki. (2018, marzo 6). *Organizations Using Python*. https://wiki.python.org/moin/OrganizationsUsingPython
RedHat. (2022, agosto 5). *Ansible Documentation*. https://docs.ansible.com/ansible/latest/index.html
Romero, D. (2021). Infraestructura como código. *Tesis de Grado, Universitat Oberta de Catalunya*. http://hdl.handle.net/10609/132647
Salazar, G. D. (2021). Hybrid Networking SDN y SD-WAN: Interoperabilidad de Arquitecturas de Redes Tradicionales y Redes definidas por Software en la era de la digitalización. *Disertación Doctoral, Universidad Nacional de La Plata*. https://doi.org/10.35537/10915/129910
Salazar, G., Marrone, L., & Naranjo, E. (2020). *Open networking programmability for VXLAN Data Centre infrastructures: Ansible and Cumulus Linux feasibility study*. https://www.researchgate.net/publication/345139981
Shah, J., & Dubaria, D. (2019). NetDevOps: A New Era Towards Networking & DevOps. *Ieeexplore.Ieee.Org*, 0775–0779. https://ieeexplore.ieee.org/abstract/document/8992969/
SparkFabrik Team. (2022, febrero 10). *Infrastructure as code: cos'è e vantaggi*. https://blog.sparkfabrik.com/it/infrastructure-as-code-cosa-e-vantaggi
Vaca, J., & Salazar, G. (2020). VXLAN-IPSec dual-overlay as a security technique in virtualized datacenter environments. *2020 IEEE ANDESCON, ANDESCON 2020*. https://doi.org/10.1109/ANDESCON50619.2020.9272160
Yuan, S. (2019, mayo 22). *The Anatomy of NetDevOps*. https://blogs.cisco.com/developer/anatomy-of-netdevops

ANEXOS

ANEXO 1 – INSTALACIÓN DE LA MÁQUINA VIRTUAL EN GOOGLE CLOUD

A continuación, se describen los pasos necesarios para la creación de la instancia virtual dentro de Google Cloud, para este propósito utilizamos el servicio de **Compute Engine**. Con fines de estudio se utiliza una cuenta de prueba gratuita de 3 meses con $300 de créditos que ofrece esta plataforma.

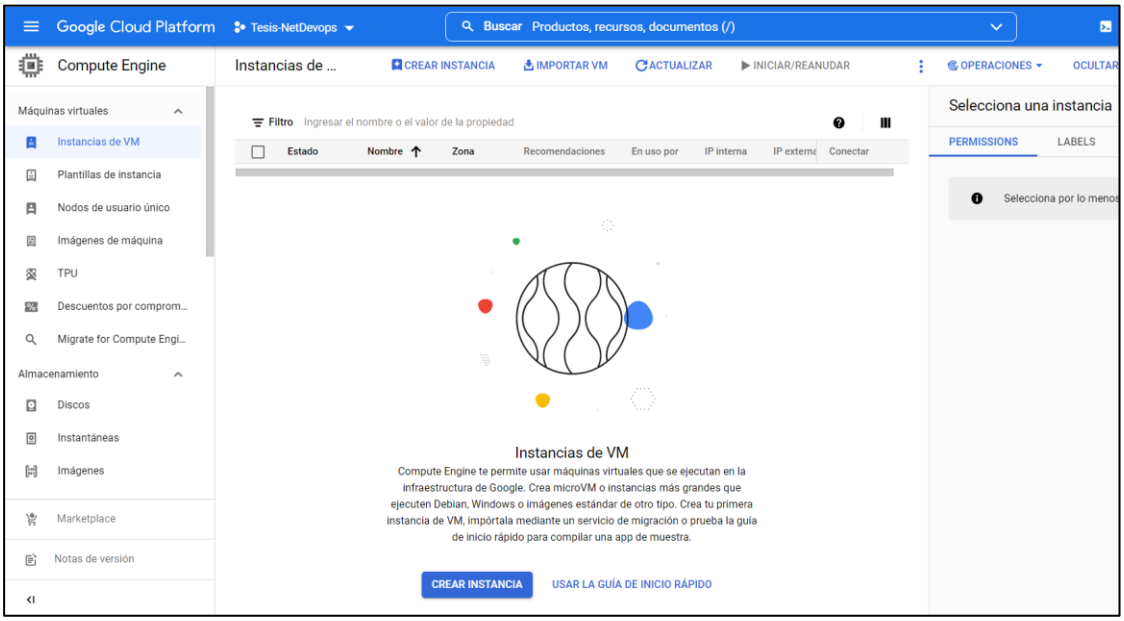


Figura 77. Interfaz de Compute Engine en Google Cloud
Fuente: *Autor*

Se configura la instancia virtual definiendo una región y zona, así como el tipo de máquina virtual que para el escenario de emulación es una n1-standard-8 de 8 CPU´s y 30 GB de memoria RAM de serie N1 y una plataforma de CPU Intel Haswell o posterior:

Figura 78. Configuración de la máquina virtual parte 1
Fuente: *Autor*

Se selecciona el S.O para el montaje de la máquina virtual, que en este caso es un Ubuntu Pro

16.04 LTS Pro Server con un disco SSD persistente de 80 GB.

Figura 79. Configuración de la máquina virtual parte 2
Fuente: *Autor*

Se configura la identidad y acceso a la API, así como los permisos del Firewall:

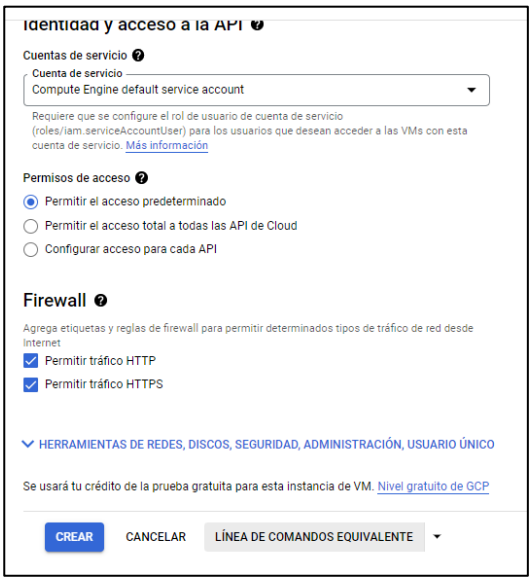


Figura 80. Configuración de la máquina virtual parte 3
Fuente: *Autor*

Un paso fundamental antes de crear la máquina virtual es activar la virtualización anidada dentro de la instancia, por lo que se selecciona la opción de **Línea de Comandos Equivalente** que se aprecia en la *Figura 80*, y se añade la siguiente línea en el cloud shell:

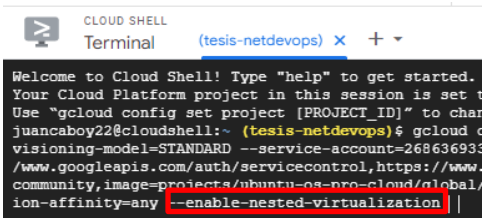


Figura 81. Activación de la virtualización anidada
Fuente: *Autor*

Tras esta edición en la Shell se da un enter y se procederá a crear e inicializar la máquina virtual, del mismo modo en la interfaz gráfica se tendrá una representación de la instancia:

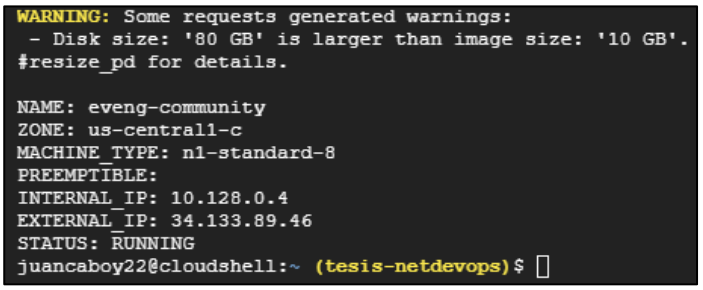


Figura 82. Creación e inicialización de la VM
Fuente: *Autor*

Como se puede observar en la *Figura 82*, la instancia está asociada a una ip interna y externa que van a variar cada vez que se inicie la instancia salvo que paguemos por un servicio de ip pública estática. Por medio de la dirección IP externa y una conexión SSH se puede acceder a la VM.

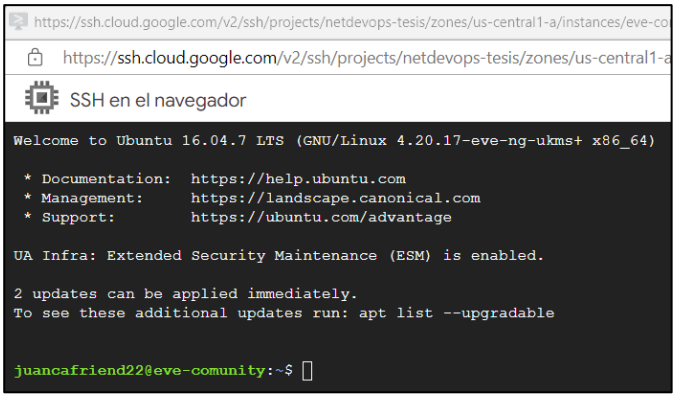


Figura 83. Acceso por SSH a la VM
Fuente: *Auto*

ANEXO 2 – INSTALACIÓN DE EVE-NG

Antes de proceder con la instalación de EVE-NG se deben crear dos reglas de firewall en la instancia de Google Cloud a fin de permitir el tráfico por los puertos TCP 0-65535(ingreso y egreso):

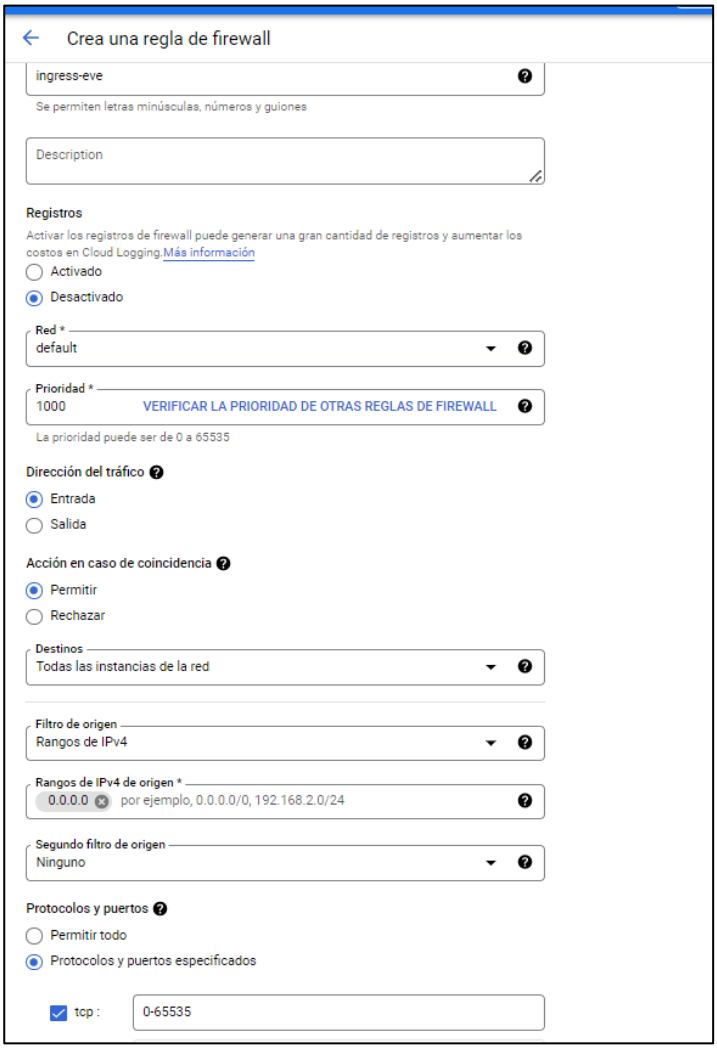


Figura 84. Creación de regla en Firewall para permitir tráfico TCP
Fuente: *Autor*

Se inicializa por consola en modo superusuario y se empieza con la instalación de EVE-NG community edition:

```
juancaboy22@eveng-community:~$ sudo -i
root@eveng-community:~# wget -O - https://www.eve-ng.net/focal/install-eve.sh | bash -i
```

Figura 85. Instalación de EVE-NG
Fuente: *Autor*

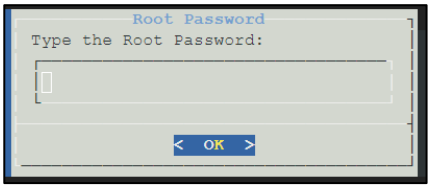
Terminada la instalación se ejecuta un update, un upgrade y un reboot a la VM:

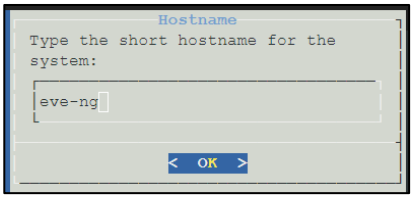
```
root@eveng-community:~# apt update
Hit:1 http://us-central1.gce.archive.ubuntu.com/ubuntu xenial InRelease
Hit:2 http://us-central1.gce.archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:3 http://us-central1.gce.archive.ubuntu.com/ubuntu xenial-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu xenial-security InRelease
Hit:5 http://www.eve-ng.net/focal focal InRelease
Hit:6 https://esm.ubuntu.com/apps/ubuntu xenial-apps-security InRelease
Hit:7 https://esm.ubuntu.com/apps/ubuntu xenial-apps-updates InRelease
Hit:8 https://esm.ubuntu.com/infra/ubuntu xenial-infra-security InRelease
Hit:9 https://esm.ubuntu.com/infra/ubuntu xenial-infra-updates InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
106 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@eveng-community:~# apt upgrade
```

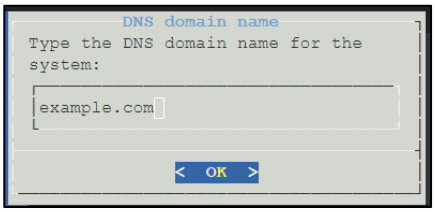
```
root@eveng-community:~# reboot
```

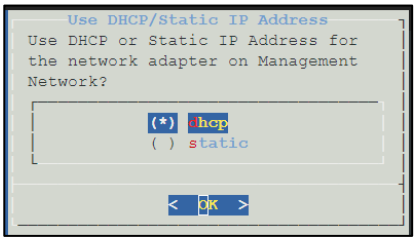
Figura 86. Finalización de la instalación de EVE-NG
Fuente: *Autor*

Luego del reboot se debe ejecutar el wizard de configuración de EVE-NG y definimos secuencialmente los siguientes parámetros:









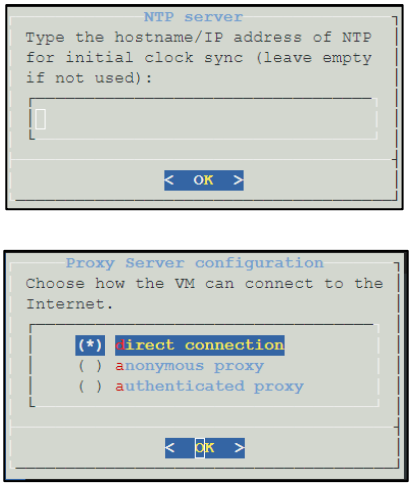


Figura 87. Wizard de configuración de EVE-NG
Fuente: *Autor*

Una vez concluida la ejecución del wizard se podrá usar la IP Externa de la instancia de Google Cloud para acceder vía http hacia EVE-NG:

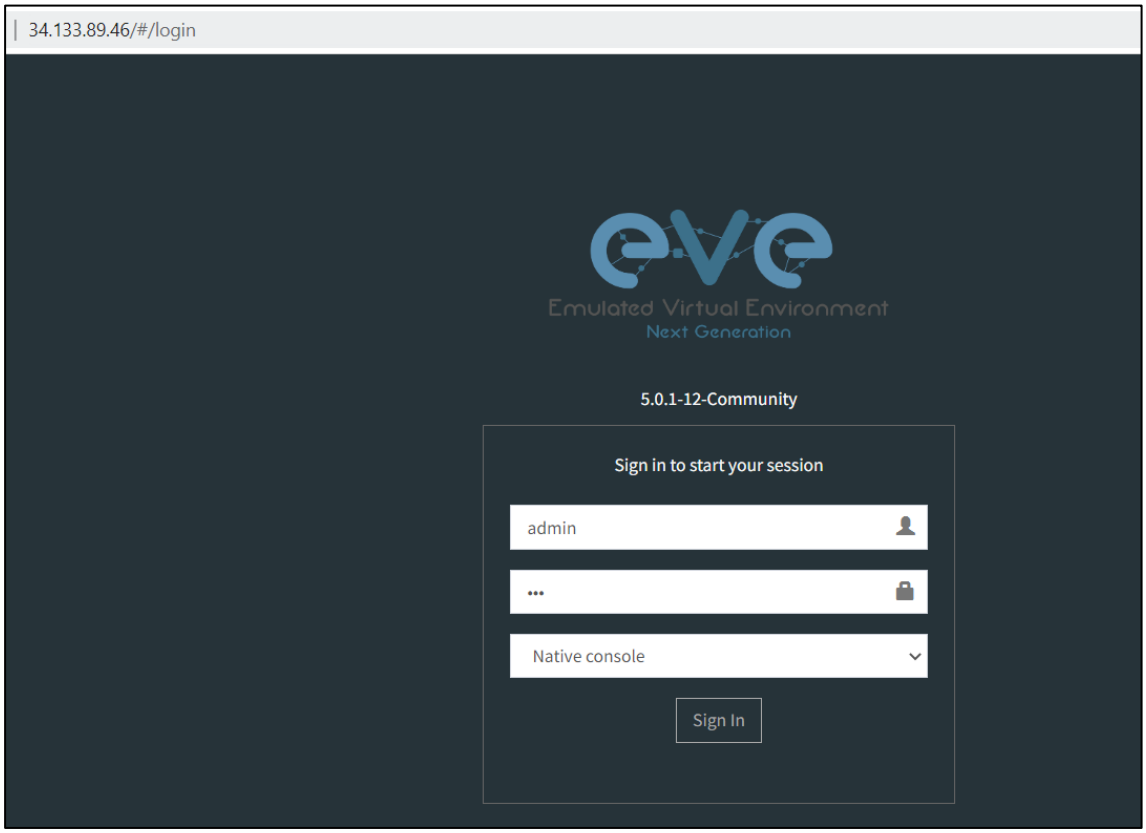


Figura 88. Acceso http a EVE-NG
Fuente: *Autor*

El procedimiento descrito se puede realizar por medio del Community Cookbook provisto en la página oficial de EVE-NG:

https://www.eve-ng.net/index.php/documentation/community-cookbook/

ANEXO 3 - CONFIGURACIÓN DE EVE-NG

Carga de NOS y S.O a los equipos de red

Una vez levantado el acceso http de EVE-NG por medio de la VM en Google Cloud, se deben agregar dentro de EVE-NG las quemu Cumulus Linux para emular los nodos SPINE, LEAF y la de Ubuntu desktop 18.04 para la instancia Linux donde se instalará Ansible, Git y Gitlab Runner. Estos archivos se obtienen de la página oficial de EVE-NG.

Para cargar la qemu de Cumulus Linux se inicia sesión como root en el EVE y se crea la carpeta donde almacenará el archivo qcow2 con el comando:

mkdir /opt/unetlab/addons/qemu/cumulus-vx-3.7.6

Se debe transferir hacia la instancia de VM el archivo q.cow para lo cual se puede hacer uso de WinSCP y luego cambiar el nombre de la imagen original de Cumulus VX a virtioa.qcow2:

cd /opt/unetlab/addons/qemu/cumulus-vx-3.7.3
mv cumulus-linux-3.7.6-vx-amd64-qemu.qcow2 /opt/unetlab/addons/qemu/cumulus-vx-3.7.6/virtioa.qcow2

Es importante corregir permisos con el comando:

/opt/unetlab/wrappers/unl_wrapper -a fixpermissions

En el caso de la carga del quemu de Ubuntu, se acude al repositorio oficial de EVE-NG:

https://mega.nz/folder/30p3TKob#42_S__9wwPVO0zHIfC4xow

Aquí se encontrarán las imágenes de los S.O, se descarga el tar.gz de Ubuntu 16.04, se crea una carpeta dentro del directorio qemu con el nombre del S.O a emular y se la descomprime:

tar xzvf linux-ubuntu-desktop-16.04.4.tar.gz

Del mismo modo se corrigen permisos con:

/opt/unetlab/wrappers/unl_wrapper -a fixpermissions

El directorio qemu final debe quedar así:

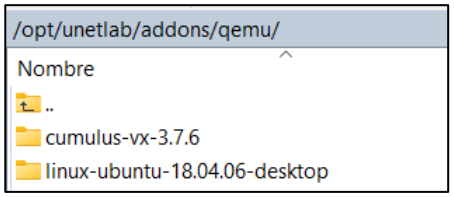


Figura 89. Directorio qemu de la emulación
Fuente: *Autor*

Salida a Internet de la PoC

Es necesario que el nodo emulado con Ubuntu Desktop tenga salida al internet para poder conectarnos en la PoC hacia el repositorio remoto. Con este objetivo basados en el direccionamiento IP propuesto en la *Figura 21*, se da direccionamiento ip estático a la interfaz pnet1 para lo que se edita el fichero **/etc/network/interfaces,** debiendo quedar de la siguiente forma:

```
iface eth1 inet manual
auto pnet1
iface pnet1 inet static
    bridge_ports eth1
    bridge_stp off
    address 10.199.199.1
    netmask 255.255.255.0
```

Figura 90. Edición del fichero /etc/network/interfaces
Fuente: *Autor*

Se reinicia la red con el comando:

systemctl restart networking

Ahora se debe activar el reenvío de tráfico IP de modo que cuando reciba un paquete que no está destinado a sí mismo, buscará el destino en su propia tabla de enrutamiento y lo reenviará según como sea necesario. Para ello, se debe descomentar una línea en el archivo **/etc/sysctl.conf**:

```
# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
```

Figura 91. Activación de IPV4 Forwarding
Fuente: *Autor*

Es necesario forzar a que sysctl lea este archivo por lo que se debe enviar el siguiente comando:

sysctl -p /etc/sysctl.conf

Finalmente es necesario crear una regla de iptables de NATEO de salida con el comando:

iptables -t nat -A POSTROUTING -s 10.199.199.0/24 -0 pnet0 -j MASQUERADE

La configuración de reglas de iptables realizadas se mantiene solo mientras no se reinicie la instancia de la VM, por ello se puede seguir el siguiente procedimiento encontrando en un blog a fin de tener reglas persistentes:

https://blog.showipintbri.com/blog/eve-ng-internet-access