

**IJECBE**

International Journal of Electrical, Computer and Biomedical Engineering

*IJECBE* (2023), 1, 2, 103–124  
Received (15 December 2023) / Revised (27 December 2023)  
Accepted (28 December 2023) / Published (30 December 2023)  
<https://doi.org/10.62146/ijecbe.v1i2.30>  
<https://ijecbe.ui.ac.id>  
ISSN 3026–5258

RESEARCH ARTICLE

# Advancing Network Infrastructure: Integrating VXLAN Technology with Automated Circuit Operations and NOS Configurations

Arfan Efendi, Diyanatul Husna, and I Gde Dharma Nugraha\*

Department of Electrical Engineering, Faculty of Engineering, Universitas Indonesia, Kampus UI Depok, West Java 16424 Indonesia

\*Corresponding author. Email: [i.gde@ui.ac.id](mailto:i.gde@ui.ac.id)

## Abstract

Enhancing network infrastructure is achieved through integrating VXLAN technology, Python-automated circuit operations, and Ansible-driven Network Operating System (NOS) configurations, complemented by GitHub for reliable configuration backups. VXLAN, a robust network virtualization protocol, addresses the challenges of managing extensive network segments. Python scripts facilitate the automated analysis, creation, and management of network circuits, significantly boosting efficiency and accuracy. Ansible, a powerful automation tool, is employed to streamline NOS configurations, ensuring consistency and reducing manual overhead in network settings. Concurrently, GitHub, working in tandem with crontab scheduling, offers a dependable platform for the automated, regular backup of configurations, thus enhancing network resilience and simplifying recovery processes. The collective implementation of VXLAN, Python, and Ansible automation, along with GitHub for configuration management, marks a notable advancement in operational efficiency, underscoring their importance as critical components in the modernization and security of network infrastructures.

**Keywords:** VXLAN, Automation, Python, Git, Ansible

## 1. Introduction

The need for robust, scalable, and efficient solutions is paramount in the ever-evolving network infrastructure domain. Virtual Extensible LAN (VXLAN) has emerged as a pivotal technology in addressing the complexities of modern network environments, particularly in data centers and cloud computing. VXLAN's ability to enable more

extensive scale network segmentation and overcome the limitations of traditional VLANs makes it an essential component in contemporary network architecture.

However, the benefits of VXLAN can be fully realized only when complemented by effective management and automation strategies. In this context, Python emerges as a powerful tool, offering flexibility and efficiency in automating network operations. Automating circuit operations, including analysis, creation, and management, becomes not just a possibility but a necessity to manage the intricate dynamics of VXLAN-enabled networks.

Moreover, reliable and accessible network configuration backups cannot be overstated in ensuring network resilience. In this vein, Ansible plays a crucial role. Its ability to automate the retrieval of network configurations and facilitate their seamless upload to a version control system like GitHub revolutionizes the way network health is maintained. By leveraging GitHub’s version control capabilities and integrating it with crontab for scheduled backups, network administrators can ensure that their configurations are secure, up-to-date, and easily recoverable in case of disruptions.

The fusion of VXLAN technology with Python-driven automation, Ansible for network configuration, and GitHub for configuration backups represents a holistic approach to network management. It addresses key challenges in network infrastructure, such as scalability, efficiency, and reliability. This article delves into these technologies’ synergies and implementation, demonstrating how they collectively enhance network infrastructure in today’s demanding and dynamic networking landscapes.

## 2. Theoretical Frameworks

### 2.1 VXLAN

#### 2.1.1 Overview of VLAN Technology

In RFC7348, VXLAN technology was formally defined and documented [1]. VXLAN functions as a Layer 2 overlay mechanism implemented over a Layer 3 core network [2]. This technology addresses the need for overlay networks in environments such as enterprise data centers and service provider settings. By providing isolation and extending its Layer 2 domains, it hopes to serve a large number of tenants [3]. VXLAN creates an overlay network by encasing a UDP datagram around a MAC frame [4]. VXLAN specifically achieves tunneling by adding extra headers, such as outer IP and UDP addresses and VxLAN headers, to encapsulate the original frames [5].

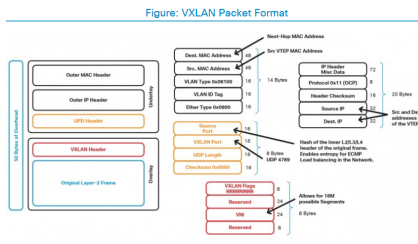


Figure 1. VXLAN Packet Format

VXLAN employs an 8-byte header featuring a 24-bit identifier (VNID) along with several reserved bits. This VXLAN header is incorporated into the UDP payload together with the original Ethernet frame. The 24-bit VNID plays a crucial role in identifying Layer 2 segments and maintaining isolation between these segments. Given the 24-bit allocation for the VNID, VXLAN can support up to 16 million logical segments. Key terminologies in the context of a VXLAN Fabric include [6]:

1. VTEP (Virtual Tunnel Endpoint): This is either a hardware or software component at the network's edge, responsible for establishing the VXLAN tunnel and handling the encapsulation and decapsulation processes of VXLAN.
2. VNI (Virtual Network Instance): This represents a logical network instance that provides Layer 2 or Layer 3 services and defines a Layer 2 broadcast domain.
3. VNID (Virtual Network Identifier): A 24-bit segment ID that addresses up to 16 million logical networks within the same administrative domain.
4. Bridge Domain: This encompasses a group of logical or physical ports that share similar flooding or broadcast characteristics.

### *2.1.2 VXLAN in Modern Networking Environment*

In modern networking environments, Virtual Extensible LAN (VXLAN) has emerged as a key technology in addressing the limitations of traditional network infrastructures, particularly in large-scale and cloud-based deployments. VXLAN enables the creation of overlay networks by encapsulating Layer 2 frames within Layer 3 packets, thus significantly expanding the scope of network segmentation and scalability. This technology is instrumental in facilitating multi-tenant architectures and providing the flexibility required in dynamic data center environments. The ability to span across various physical locations while maintaining network isolation positions VXLAN as a cornerstone technology in the evolution of network virtualization and cloud computing.

### *2.1.3 Challenges and Solutions with VXLAN Implementation*

Implementing VXLAN technology, while offering substantial benefits in network virtualization and scalability, presents distinct challenges, including complexity in network configuration and management and potential integration issues with existing network infrastructures. Addressing these challenges involves leveraging advanced network automation tools and strategies to simplify VXLAN deployment and configuration processes. Additionally, ensuring compatibility with existing VLAN setups necessitates meticulous planning and the use of transition mechanisms. Overcoming these hurdles also requires focusing on educating network teams about VXLAN's operational intricacies. Moreover, optimizing the use of network resources to manage the increased overhead due to VXLAN encapsulation is essential. Effective solutions lie in adopting a holistic approach that combines technical strategies with skill development, ensuring a smooth and efficient VXLAN integration into diverse network environments.

## **2.2 Automation**

### 2.2.1 *The Role of Automation in Network Management*

Currently, automation is a commonly heard concept in the field of computer technologies. Network architects and operators are confronted with a growing number of everyday obligations, such as implementing new services and improving network performance [7]. The primary objective of an automated system is to minimize the requirement for human intervention. Upon receiving information from a human or another system, it is programmed to operate autonomously without requiring additional external input. Although the intricacy involved in building such systems can be a barrier, their benefits frequently outweigh and surpass this concern. Enhancements are typically observed in both the efficacy of the activity and the caliber of the solutions generated. The primary function of the human operator is to initiate the system and provide assistance or maintenance when necessary throughout its autonomous operations rather than being responsible for the entire activity. Furthermore, this results in solutions being obtained with greater efficiency and precision [8]. Automation enables real-time network adjustments, proactive troubleshooting, and dynamic resource allocation, improving network performance and reliability. This shift towards automation is essential in modern network environments, where the complexity and scale of networks demand more agile and responsive management techniques.

### 2.2.2 *Automation Techniques and Tools*

**Python as a Scripting Code:** Python has emerged as a leading scripting language in network automation for its simplicity, versatility, and extensive library support. It allows for creating custom scripts to automate various network tasks, from configuration management to data analysis, providing a powerful tool for network engineers to streamline complex processes.

**Ansible as Network Operating System (NOS):** Infrastructure as Code (IaC) tools, including Ansible scripts, are employed in the large-scale deployment of computing infrastructure [9]. Ansible was chosen as the primary control method due to its extensive use in both the commercial and academic sectors for scripting infrastructure definitions and deploying them across various cloud providers [10][11][12]. Ansible offers a robust platform for automating network operating systems, enabling consistent and scalable configuration across diverse network devices. Its agentless architecture and use of simple, declarative language make it an effective tool for automating routine tasks and ensuring network configuration compliance. Ansible characterizes the IT infrastructure by emphasizing the interconnections among servers, rather than treating them as separate entities. An essential characteristic of Ansible is its dependence on inventory files, which are considered the primary authoritative source [13].

**Git as Version Control System (VCS):** Git plays a critical role in automation by providing a reliable system for version control of network configurations. It allows for tracking changes, collaborating on configurations, and maintaining a historical record of network states, enhancing the overall management and recovery processes.

### 2.2.3 *Integration of Automation with VXLAN*

Integrating automation tools and practices with VXLAN technology is key to managing the inherent complexity of large-scale virtual networks. Automation facilitates efficient deployment, configuration, and management of VXLAN networks, enabling rapid provisioning of network segments and streamlined management of virtual overlays. Network administrators can quickly adapt VXLAN configurations to changing network requirements through automation, reduce manual intervention, and ensure consistency across the network's virtual landscape.

### 2.2.4 *Addressing the Complexity of VXLAN Networks*

Addressing the complexity of VXLAN networks through automation involves leveraging tools and scripts to manage the extensive network segments and the associated encapsulation overhead. Automation provides the means to efficiently handle VXLAN-specific configurations, such as VTEP setup and VNI mappings, and to integrate these with existing network policies and structures. By automating these processes, network administrators can effectively manage the scalability and flexibility that VXLAN offers while maintaining oversight and control over the network's performance and security.

## 3. **Related Works**

The introduction of VXLAN technology in datacenter connection topologies represents a notable advancement in the domain of network architecture. This technology is wellacknowledged for its capacity to improve network scalability and flexibility, especially in intricate data center environments. The adoption of VXLAN is motivated by the need for more effective network segmentation solutions that can accommodate the growing demands of contemporary data traffic and cloud services. In this particular situation, automation is not merely a convenience but rather an essential requirement to diminish human mistakes and enhance operational effectiveness. This development highlights a significant change in the network management paradigm, with a growing emphasis on automated solutions for effectively managing intricate network architectures.

The use of automation in network operations, particularly in the context of VXLAN, has become a crucial area of concentration. The automation process primarily aims to streamline mundane and repetitive processes, enabling network managers to dedicate their attention to more strategic efforts. The increasing adoption of scripting languages and tools such as Ansible demonstrates the shift towards automation, facilitating the development of network systems that are more flexible and robust. Ansible has become well-known for its capacity to automate intricate network settings and streamline their incorporation with version control systems like Git. These advancements emphasize the industry's continuous endeavors to improve network dependability and efficiency through automated procedures, which is a crucial element of this research.

The choice of Python as the programming language for network automation activities is essential in this transformation. Python is widely recognized for its simplicity and adaptability, which makes it an excellent option for automating intricate network tasks. Python's extensive standard library and diverse third-party modules make it

highly suitable for duties such as verifying current configurations and dynamically modifying network settings. The utilization of Python in this role exemplifies a wider industry pattern of employing advanced scripting languages for intricate network automation assignments.

Moreover, the decision to incorporate Git as a Version Control System for configuration management and utilizing Ansible for automating configuration retrievals and changes is a significant choice in this research. The capacity of Git to monitor modifications, preserve a record of versions, and support collaborative endeavors, along with Ansible's effectiveness in managing configurations, renders them indispensable tools for network configuration management. This technique not only guarantees the consistency and restorability of network configurations but also adds a level of security and responsibility that is crucial in contemporary network administration. Integrating Git's resilient version control system with the automation capabilities of Python and Ansible offers a comprehensive method for effectively managing the intricacies associated with VXLAN-based network systems.

In the quest to safeguard the network configuration backups stored in the Git repository, particularly because these backups are hosted on an external platform, meticulous security protocols have been adopted. The repository is set to private, ensuring that access is strictly limited to authorized personnel who have been granted explicit invitations. This controlled access mechanism is crucial in preventing unauthorized disclosure and potential security breaches. By restricting repository visibility and interaction to a select group, the research upholds stringent security standards, thereby reinforcing the confidentiality and integrity of sensitive network configuration data. This proactive security measure reflects a comprehensive understanding of the risks associated with external data storage and underscores the research's commitment to maintaining robust network security in an era where data protection is paramount.

The author acknowledges that there is much to be developed for future progress. Therefore, the Python automation scripting files have been uploaded to a public repository for use in future developments. The repository URL is accessible at <https://github.com/arevhan/vxlan-circuit-management>.

## **4. Methodology**

### **4.1 Network Setup**

This chapter will specifically address the topology employed and the procedure for establishing a VXLAN circuit. The laboratory experiment employs Eve-NG as an emulation tool to support the use of VXLAN technology within a spine and leaf topology. Due to limited resources, a solitary spine router, specifically dc-spine, functions as the route reflector. The lab configuration consists of four leaf routers, each representing a distinct Point of Presence (POP) located in different locations and data centers. The names assigned to these are Bandung-leaf, Jakarta-leaf, Surabaya-leaf, and Bali-leaf.

A switch exclusively designated for administration is linked to every device to improve the network setup, guaranteeing smooth integration and control. In addition, the system incorporates a jump host that serves as a central controller for automated operations. The jump host serves as a crucial component in coordinating and optimizing

the network automation tasks, effectively consolidating control.

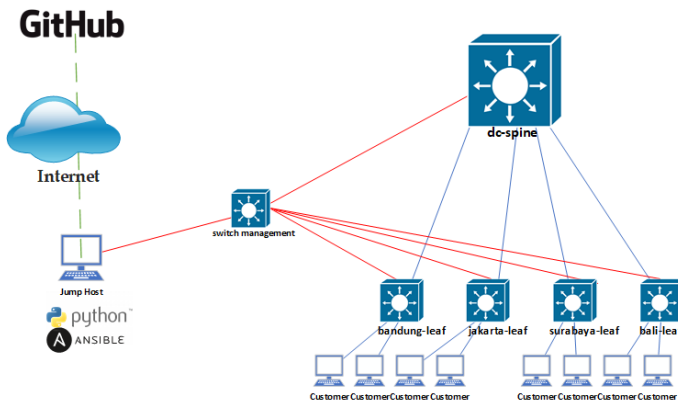


Figure 2. Automation operation with spine and leaf topology

Based on topology above, a crucial element of the network architecture is the jump host, which serves as a centralized server with Ansible and Python installed, playing a key role in network automation and acting as a scheduler for pushing updates to Git repositories. Jump host connects to the internet to upload to the GitHub repository. The network topology adopted is a spine and leaf configuration optimized for resource efficiency, with a single spine router functioning as a route reflector. To ensure streamlined management and oversight, the spine, leaf nodes, and the jump host are interconnected through a dedicated management switch, highlighting an integrated approach to network administration in modern, resource-constrained environments.

#### 4.1.1 VXLAN Implementation

The study involves deploying VXLAN on the Cisco Nexus 9000v in the Eve-NG virtual environment, which serves as the virtual infrastructure for the research. The Cisco Nexus 9000v series is selected for its sophisticated capabilities in managing VXLAN, which is essential for attaining network virtualization and segmentation.

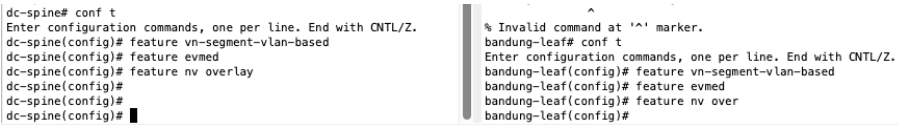
Before commencing the VXLAN configuration, it is imperative to activate particular functionalities on the Cisco Nexus 9000v devices. The major features include 'vn-segment', 'evmed', and 'nv overlay'. Including the 'vn-segment' function is crucial for facilitating VLAN to VXLAN mapping, enabling enhanced network segmentation with improved efficiency. The 'evmed' functionality is utilized to manage events within the network, offering improved monitoring and automation functionalities. Finally, the 'nv overlay' function is crucial for activating the network virtualization overlay, which is the fundamental VXLAN technology.

The configuration procedure is executed with great attention to detail on every device inside the spine and leaf structure. This entails configuring each spine and leaf device in the network to guarantee their ability to sustain and engage in the VXLAN overlay. The configuration process involves setting up VXLAN tunnel endpoints

(VTEPs), specifying the VXLAN network identities (VNIs), and configuring the required routing protocols to enable efficient data transmission across the VXLAN fabric.

The configuration of each spine and leaf device is customized to suit its specific role in the architecture. The spine devices serve as the primary infrastructure of the network, designed to ensure the most efficient routing and communication between the leaf devices. The leaf devices, which serve as the access layer of the network, are set up to establish connections between end devices and other network components within the VXLAN overlay.

This research intends to showcase the practical implementation of VXLAN by carefully configuring and setting up Cisco Nexus 9000v devices in the Eve-NG environment. The goal is to illustrate the efficacy of VXLAN in constructing a network infrastructure that is stable, scalable, and adaptable.



```

dc-spine# conf t
Enter configuration commands, one per line. End with CNTL/Z.
dc-spine(config)# feature vn-segment-vlan-based
dc-spine(config)# feature evmed
dc-spine(config)# feature nv overlay
dc-spine(config)#
dc-spine(config)#
dc-spine(config)#
bandung-leaf# conf t
% Invalid command at '^' marker.
Enter configuration commands, one per line. End with CNTL/Z.
bandung-leaf(config)# feature vn-segment-vlan-based
bandung-leaf(config)# feature evmed
bandung-leaf(config)# feature nv over
bandung-leaf(config)#

```

Figure 3. Enabling feature to implementation of VXLAN

#### 4.1.2 Network Device Configuration

The initial step in configuring network devices for this VXLAN implementation involves creating the Open Shortest Path First (OSPF) protocol as the underlying network. OSPF is an extensively utilized inner gateway protocol that plays a vital role in facilitating efficient and dynamic routing within a network. It establishes the fundamental routing architecture required to operate the VXLAN overlay efficiently.

Subsequently, the setup of VXLAN Tunnel Endpoints (VTEPs) is performed. VTEPs have a crucial function in the VXLAN architecture by encapsulating and decapsulating the network traffic. This procedure entails allocating IP addresses to VTEPs and guaranteeing seamless integration into the OSPF routing system, enabling uninterrupted connectivity throughout the network.

MP-BGP is configured as the control plane for VXLAN. MP-BGP is responsible for disseminating the routing and forwarding information among the VTEPs, allowing them to acquire and sustain knowledge of the network's topology. This configuration is crucial for establishing a cohesive, expandable, and effective overlay network that extends across multiple network segments.

After setting up the underlay and control plane, the next step is configuring the VXLAN network identifiers (VNIs). This entails configuring the 'vn-segment' on VLANs, establishing the 'nve' interface, and linking VNIs with the EVPN instances. Mapping the VLANs to VNIs is a crucial step that allows for the encapsulation of Layer 2 frames into Layer 3 packets, a fundamental requirement for VXLAN.

Layer 3 switches are also utilized as management switches, in addition to these setups. Every management switch is linked to the management ports of the spine and leaf devices. This configuration offers a specialized network for managing traffic, apart



from the traffic used for data transmission. It guarantees strong network management and monitoring capacities, making it easier to administer the spine-leaf design and simplify the management of the entire network infrastructure.

By carefully following these configuration steps, which include implementing OSPF as the underlying protocol and configuring MP-BGP, VTEPs, and VNIs, the network devices are optimized to operate using VXLAN. By including Layer 3 management switches, the network's management and operational efficiency are enhanced, establishing a strong foundation for a high-performing and resilient VXLAN network.

## **4.2 Automation Scripts Development**

This research utilizes Python as the primary coding framework in the field of script creation for network automation, together with Ansible for automating backup setups. The selection of Python is motivated by its extensive recognition as a robust and adaptable programming language, especially suitable for intricate network automation assignments. Python's vast library ecosystem and its capacity to seamlessly connect with diverse network devices and protocols make it an optimal selection for creating tailored automation scripts.

The Python scripts created in this research are intended to automate various processes within the VXLAN network. This encompasses the automation of network configuration analysis, the installation and administration of circuits, and the real-time modification of network settings. Python's scripting features enable the development of powerful, scalable, and adaptable automation solutions that can meet the evolving requirements of the network infrastructure.

In addition to Python, Ansible is employed as a crucial tool for automating the configuration of network backups. Ansible, renowned for its straightforwardness and lack of agents, excels at managing and configuring network devices. The declarative nature of the language enables precise and succinct declarations of configuration states, making it a powerful tool for network backup management. This study involves the creation of Ansible playbooks to automate the retrieval of network configurations and guarantee their uniformity by uploading them to a Git repository for version control.

Integrating Python for general network automation activities with Ansible for backup configuration automation constitutes a holistic approach to network administration. Python scripts manage the immediate operational activities in the network, guaranteeing efficiency and dependability in daily operations. Ansible playbooks prioritize the essential task of configuration management, ensuring that network configurations are securely backed up and easily recoverable when needed. The utilization of both approaches in automation script development demonstrates the capability of integrating diverse technologies to attain a robust and controllable network infrastructure.

### **4.2.1 Python Scripting**

Python scripting is essential for automating many aspects of network design and administration in this research. The utilization of the Netmiko module is essential in this methodology, facilitating smooth communication between Python scripts and

network devices. Netmiko is a Python library developed explicitly for managing SSH connections to network devices. It plays a crucial role in performing setup instructions and receiving data from these devices. The author has created a collection of Python files, with each file having a distinct purpose in the process of automation:

1. The file "devices\_leaf.py". This script is the fundamental element of the automation process. It functions as a repository for storing login credentials and host information, facilitating access to network devices. This script enhances the efficiency and security of the automation process by consolidating device access information, hence streamlining the management of various devices.
2. The file "add\_vlan.py": This script specifically addresses the VLAN configuration element. It allows for the automatic setup of VLANs on network devices according to user inputs. This function is crucial for preserving network segmentation and guaranteeing the optimal distribution of network resources. Since many revision for this script we will using add\_vlan3.py for final result.

```

Start
    Import Modules
        ConnectHandler from netmiko
        subprocess
        devices from devices_leaf

    Define Functions
        vlan_exists(connection, vlan)
        configure_vlan_on_device(device, vlan, interface)
        post_configuration_menu()

    Display Device List

    Get User Input for Device Selection
        If no valid devices selected, go to End

    Get VLAN ID from User
        Validate VLAN ID
        If invalid, go to End

    For Each Selected Device
        Configure VLAN

    Call post_configuration_menu()
        If 'Back to Main Menu', run check_vlan3.py
        If 'Exit', go to End

End

```

Figure 4. Add Vlan algorithm

3. The file "add\_vni.py": This script is designed to configure VXLAN on network devices. This tool streamlines the procedure of linking VLANs with VXLAN network identifiers (VNIs), which is a crucial stage in establishing a VXLAN overlay network. This script guarantees the uniformity and accuracy of VXLAN configuration throughout the network.
4. The file "add\_vni.py": This script is designed to configure VXLAN on network devices. This tool streamlines the procedure of linking VLANs with VXLAN network identifiers (VNIs), which is a crucial stage in establishing a VXLAN

overlay network. This script guarantees the uniformity and accuracy of VXLAN configuration throughout the network.

```

Start

Import Modules
    ConnectHandler from netmiko
    subprocess
    devices from devices_leaf

Define Functions
    vlan_exists(connection, vlan)
    configure_vni_on_device(device, vni)
    main_menu()

Display Device List for VNI Configuration
    Loop: For each device, display index. device name

Get User Input for Device Selection
    Validate user input for device selection
    If no valid devices selected, go to End

Get VNI Input from User
    Validate VNI input
    If invalid VNI, go to End

For Each Selected Device
    Call configure_vni_on_device with device and VNI

Call main_menu() Function
    Display menu options (Back to Main Menu, Go to Add VLAN, Exit)
    Get user choice
    If choice is "1" (Back to Main Menu), run check_vlan3.py and go to Start
    If choice is "2" (Go to Add VLAN), run add_vlan3.py and go to Start
    If choice is "3" (Exit), go to End

End

```

Figure 5. Add VNI algorithm

5. The file *check\_vlan.py*. This file functions as the primary menu executable script, offering a user-friendly interface for network managers. It enables them to verify VLAN IDs, VNI configurations on the nve interface, and EVPN settings. This script is essential for overseeing and verifying the network configuration, guaranteeing that the network functions as planned. Also for some revision we decided to using *check\_vlan3.py* for final script.

The collection of these Python scripts constitutes a comprehensive set of tools for automating network tasks. This research showcases the efficacy of Python in automating intricate network activities by utilizing Netmiko for device connectivity and developing specialized scripts for certain tasks. Scripts not only streamline mundane chores but also improve the precision and dependability of network setups, which is vital in VXLAN-enabled environments.

The flowchart depicts a structured network automation sequence that begins with the execution of the 'Check\_vlan.py' script to verify the existence of a specific VLAN across network devices. Upon entering the VLAN ID, the script determines whether the VLAN already exists; if it does not, the user selects the desired node(s) to configure and proceeds to run 'Add\_vlan.py' to add the VLAN. Subsequently, 'Add\_vni.py' is invoked to associate a VNI with the newly created VLAN, requiring the user to input the VNI number. This process iterates through a decision loop where the user can either continue adding VNIs to other nodes or conclude the session, ending with the 'Exit' state that terminates the automation process.

```

Start
Import Modules
ConnectHandler from netmiko
devices from devices_leaf
subprocess

Prompt User for VLAN Input
Get VLAN ID from the user
Validate VLAN ID (between 1 and 4095)
If invalid, Print "Invalid VLAN ID. Exiting." and go to End

Calculate VNI Based on VLAN
VNI = 100000 + VLAN

Iterate Over Each Device
For each device in devices:
    Establish connection
    Check for VLAN existence
    Check VNI existence on interface nve1
    Check VNI existence in EVPN configuration
    Disconnect from device
    Print an empty line

Display Post-Configuration Options
Print options: "1. Add VLAN", "2. Add VNI", "3. Exit"
Get user choice

Process User Choice
If choice is "1", run add_vlan3.py
If choice is "2", run add_vni.py
If choice is "3", Print "Exiting" and go to End
If invalid choice, Print "Invalid choice. Exiting." and go to End

End
    
```

Figure 6. Check Vlan algorithm

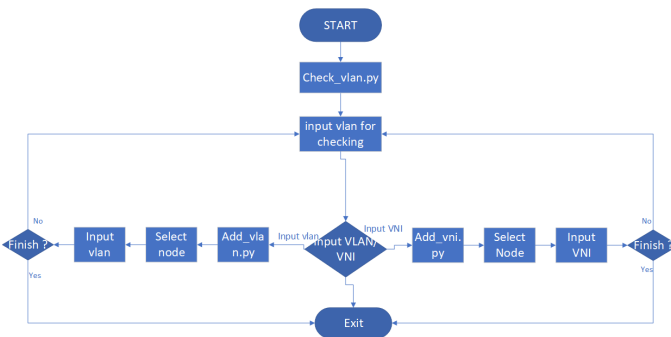


Figure 7. Flowchart python automation script

### 4.2.2 Ansible Playbooks

Ansible is crucial in automating configuration management and backup processes in this research. Ansible is deployed on a central management system to guarantee that all network device configurations are current and uniform. Ansible serves as a safeguard by automatically preserving configurations in situations where operators may overlook writing them during device setup. This functionality greatly mitigates the possibility of configuration divergence and improves the overall dependability of the network infrastructure.

```
root@junghost-22:/etc/ansible/backup_configuration# ls
2023-10-15 2023-10-17 2023-10-20 2023-11-17 2023-11-19 2023-11-21 2023-11-23 2023-12-13 2023-12-15 2023-12-17 2023-12-19 2023-12-21 2023-12-23 2023-12-25
2023-10-16 2023-10-18 2023-11-16 2023-11-18 2023-11-20 2023-11-22 2023-12-12 2023-12-14 2023-12-16 2023-12-18 2023-12-20 2023-12-22 2023-12-24 2023-12-26
root@junghost-22:/etc/ansible/backup_configuration#
```

Figure 8. Capture local drive backup configuration from ansible

In addition, Ansible is utilized to gather and store backup settings from every network device. The backups are stored on a local drive, guaranteeing the presence of a dependable duplicate of every device’s settings at all times. This strategy is crucial for expedited recovery in the event of a device malfunction or other network complications.

```
0 00 * * * /usr/bin/ansible-playbook -i /etc/ansible/hosts.yml /etc/ansible/backup_configuration.yml
5 00 * * * sh /etc/ansible/chown_from_root_to_arfan.sh
0 06 * * * /usr/bin/ansible-playbook -i /etc/ansible/hosts.yml /etc/ansible/backup_configuration.yml
0 12 * * * /usr/bin/ansible-playbook -i /etc/ansible/hosts.yml /etc/ansible/backup_configuration.yml
0 18 * * * /usr/bin/ansible-playbook -i /etc/ansible/hosts.yml /etc/ansible/backup_configuration.yml
```

Figure 9. Capture crontab -e scheduler for ansible playbook backup configuration

A crontab scheduler is used to automate the backup procedure. The scheduler is set to activate the Ansible playbook, which subsequently retrieves the configurations, four times daily. The process commences at 00:00 and recurs every six hours. The frequent and regular backup schedule guarantees that the latest settings are consistently backed up, hence reducing the risk of data loss in the event of sudden network changes or failures.

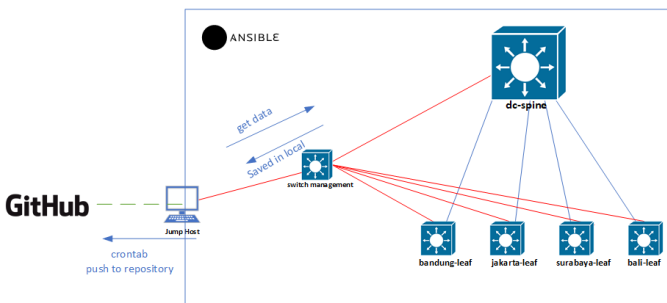


Figure 10. Flow Automation using ansible and github

At the specified time, Jump host takes configuration backup data to the network device. Once successful, it will be saved on the local driver server as a backup. To

continue saving to the repository, Jumphost will push to GitHub at the specified time using cronab.

### 4.3 Version Control Integration Git Configuration

We recognize that backup configuration files are inherently private. However, as a precautionary measure against potential internal failures, we have opted to upload them outside the internal network. To mitigate any associated risks, we have instituted restrictions on these files. We designate the files as private and accessible by invitation only.

The steps to make a repository private begin with navigating to the dashboard menu, entering the repository name, and setting it to private, followed by submitting the "create a new repository" form.

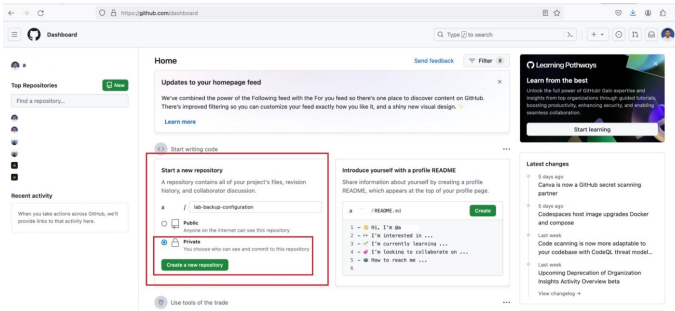


Figure 11. Create private repository from dashboard

Figure 8 illustrates the creation of a private repository from the dashboard. Once the repository is established, we access the previously created repository. As shown in Figure 9, the repository is confirmed to be private.

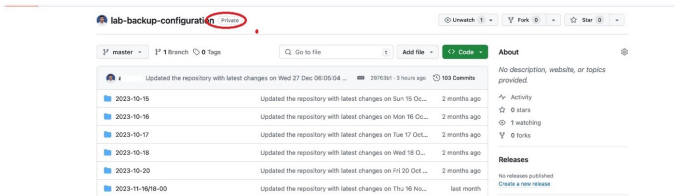


Figure 12. Private repository

Integrating Git into this network management framework is crucial for establishing a reliable version control system for the network configurations. The Git repository is hosted on a jumphost, an Ubuntu server that functions as a central hub for managing configuration version control.

To establish a connection between the Git repository on the jumphost and the network infrastructure, a series of actions are executed. Initially, Git is installed on the Ubuntu server, and a Git repository is created to hold the network configurations.

Subsequently, SSH keys are made on the jump host and then incorporated into the Git service (such as GitHub or a self-hosted Git server) to build a safe and protected connection.

```

281 apt-get install git
282 git config --global user.name arfan
283 git config --global user.email arfan@gmail.com
284 cd /etc/ansible/backup_configuration
285 git init
286 git add .
287 git commit -m "Initial Commit"
288 git remote add origin https://github.com/arfanelab-backup-configuration.git
289 git push -u origin master
290 ls -al ~/.ssh
291 ssh-keygen -t ed25519 -C arfan@gmail.com
292 exit
293 exit
294 git remote add origin https://github.com/arfanelab-backup-configuration.git

```

Figure 13. Step by step establish github to jump host server

The sequence of commands outlines the process for setting up Git for version control and connecting to a remote repository. It begins with the installation of Git on the system using “*apt-get install git*”. The user then configures Git with their username and email using *git config*. After initializing a new Git repository within the Ansible backup configuration directory using “*git init*”, all existing files are staged for the initial commit “*with git add .*”. A commit is then made with a descriptive message “*git commit -m “Initial Commit”*”. The next step is to link the local repository to a remote GitHub repository using “*git remote add origin*”, followed by pushing the commit to the remote repository on the ‘master’ branch with “*git push -u origin master*”. SSH keys are generated using *ssh-keygen* to establish a secure connection between the local machine and the GitHub repository.

## 4.4 Testing and Validation

### 4.4.1 Automated Operations Testing

1. After careful review and multiple modifications, the *check\_vlan3.py* script has been chosen to be executed during this testing phase. This test aims to create VLAN 999 and then integrate it into a VXLAN segment using VNI 100999. The process commences by executing *check\_vlan3.py* using the “*python3 check\_vlan3.py*” command. In this procedure, the VLAN ID is configured as 999, establishing the foundation for further operations in the test sequence.

```

| arfan@jumphost-22:~/automation$ python3 check_vlan3.py
| Enter the VLAN ID (1-4095) you want to check: 999

```

Figure 14. python3 executed check\_vlan3.py script

2. The Python script initiates a comprehensive check across all leaf nodes for VLAN 999 and VNI 100999 presence.
3. Upon completing the checking process, the script presents an option to either add the VLAN and VNI or exit. Selecting option 1 initiates the addition of the VLAN, leading the process to transition to “*add\_vlan.py*”. A menu then reappears, prompting the selection of the specific node for VLAN configuration.

```
VLAN 999 does not exist on bandung-leaf.
VNI 100999 is NOT configured as a member on interface nve1 of bandung-leaf.
VNI 100999 is NOT configured in the EVPN section of bandung-leaf.

VLAN 999 does not exist on jakarta-leaf.
VNI 100999 is NOT configured as a member on interface nve1 of jakarta-leaf.
VNI 100999 is NOT configured in the EVPN section of jakarta-leaf.

VLAN 999 exists on surabaya-leaf.
VNI 100999 is NOT configured as a member on interface nve1 of surabaya-leaf.
VNI 100999 is NOT configured in the EVPN section of surabaya-leaf.

VLAN 999 exists on bali-leaf.
VNI 100999 is NOT configured as a member on interface nve1 of bali-leaf.
VNI 100999 is NOT configured in the EVPN section of bali-leaf.
```

Figure 15. fpython3 executed check\_vlan3.py script

```
What would you like to do next?
1. Add VLAN
2. Add VNI
3. Exit
Enter your choice (1 , 2 or 3): 1
```

Figure 16. Next action after checking vlan from check\_vlan3.py

4. Nodes 1 and 2 are selected for this test, representing bandung-leaf and jakarta-leaf, respectively. The user inputs VLAN 999, then specifies the interface for assigning this VLAN to the customer. The script subsequently executes the creation of VLAN 999 and sets the appropriate port for node 1. A similar input prompt then facilitates the interface assignment for node 2, facing the customer.

```
Select devices to configure (e.g., 1 3 5):
1. bandung-leaf
2. jakarta-leaf
3. surabaya-leaf
4. bali-leaf
Enter the numbers of the devices separated by space: 1 2
Enter the VLAN ID (1-4095) you want to set: 999
Configuring bandung-leaf
Enter the interface for bandung-leaf you want to configure (e.g., Eth1/7): Eth1/2
VLAN 999 (VNI 100999) not found on bandung-leaf. Creating VLAN 999 and setting vn-segment to 100999.
Interface Eth1/2 on bandung-leaf already has VLAN 999 configured.
Configuring jakarta-leaf
Enter the interface for jakarta-leaf you want to configure (e.g., Eth1/7): Eth1/2
VLAN 999 (VNI 100999) not found on jakarta-leaf. Creating VLAN 999 and setting vn-segment to 100999.
Interface Eth1/2 on jakarta-leaf already has VLAN 999 configured.
```

Figure 17. create vlan into selected nodes.

5. Returning to the main menu, a verification check for VLAN 999 is explicitly performed for bandung-leaf and jakarta-leaf.
6. The results from this query confirm that both bandung-leaf and jakarta-leaf are configured with VLAN 999, as evidenced by the output. This configuration is further substantiated by reference to Figure 16 in the documentation.
7. The process then navigates back to the main menu, where option two is selected, leading to the execution of “*add\_vni.py.*” This script is run for devices 1 and 2, corresponding to bandung-leaf and jakarta-leaf, with the input 999 for VNI. This input is automatically translated into 100999 for the device configuration.
8. A subsequent verification is conducted from the main menu by selecting option one and inputting 999.
9. The output indicates that VLAN 999 and VNI 100999 are configured on both



```
What would you like to do next?
1. Back to Main Menu
2. Exit
Enter your choice (1 or 2): 1
Enter the VLAN ID (1-4095) you want to check: 999
VLAN 999 exists on bandung-leaf.
VNI 100999 is NOT configured as a member on interface nve1 of bandung-leaf.
VNI 100999 is NOT configured in the EVPN section of bandung-leaf.

VLAN 999 exists on jakarta-leaf.
VNI 100999 is NOT configured as a member on interface nve1 of jakarta-leaf.
VNI 100999 is NOT configured in the EVPN section of jakarta-leaf.

VLAN 999 exists on surabaya-leaf.
VNI 100999 is NOT configured as a member on interface nve1 of surabaya-leaf.
VNI 100999 is NOT configured in the EVPN section of surabaya-leaf.

VLAN 999 exists on bali-leaf.
VNI 100999 is NOT configured as a member on interface nve1 of bali-leaf.
VNI 100999 is NOT configured in the EVPN section of bali-leaf.

What would you like to do next?
1. Add VLAN
2. Add VNI
3. Exit
Enter your choice (1 , 2 or 3):
```

Figure 18. Back to main menu and re-check vlan 999 is exist based on input before

```
What would you like to do next?
1. Add VLAN
2. Add VNI
3. Exit
Enter your choice (1 , 2 or 3): 2
Select devices to configure VNI (e.g., 1 3 5):
1. bandung-leaf
2. jakarta-leaf
3. surabaya-leaf
4. bali-leaf
Enter the numbers of the devices separated by space: 1 2
Enter the VNI you want to set (e.g., 397): 999
Configuring VNI 100999 on bandung-leaf...
VNI 100999 configured on bandung-leaf.
Configuring VNI 100999 on jakarta-leaf...
VNI 100999 configured on jakarta-leaf.

What would you like to do next?
1. Back to Main Menu
2. Go to Add VLAN
3. Exit
Enter your choice (1, 2, or 3):
```

Figure 19. Input VNI to establish VXLAN

```

What would you like to do next?
1. Back to Main Menu
2. Go to Add VLAN
3. Exit
Enter your choice (1, 2, or 3): 1
Enter the VLAN ID (1-4095) you want to check: 999
VLAN 999 exists on bandung-leaf.
VNI 100999 is configured as a member on interface nve1 of bandung-leaf.
VNI 100999 is configured with 'l2' in the EVPN section of bandung-leaf.

VLAN 999 exists on jakarta-leaf.
VNI 100999 is configured as a member on interface nve1 of jakarta-leaf.
VNI 100999 is configured with 'l2' in the EVPN section of jakarta-leaf.

VLAN 999 exists on surabaya-leaf.
VNI 100999 is NOT configured as a member on interface nve1 of surabaya-leaf.
VNI 100999 is NOT configured in the EVPN section of surabaya-leaf.

VLAN 999 exists on bali-leaf.
VNI 100999 is NOT configured as a member on interface nve1 of bali-leaf.
VNI 100999 is NOT configured in the EVPN section of bali-leaf.

What would you like to do next?
1. Add VLAN
2. Add VNI
3. Exit
Enter your choice (1, 2 or 3): 3
Exiting
Going back to the main menu...
arfan@jumphost-22:~/automation$ █

```

**Figure 20.** Back to main menu and verify vlan 999 and VNI 100999 created on selected nodes.

bandung-leaf and jakarta-leaf, affirming the successful operation of the automation process.

#### 4.4.2 VXLAN Functionality Testing

Before the execution of `check_vlan3.py`, a verification is performed to ascertain the non-existence of the VLAN. Additionally, the configuration of VNI on interface `nve 1` under EVPN is also verified for completeness and accuracy.

```

bandung-leaf#
bandung-leaf#
bandung-leaf#
bandung-leaf# show vlan br | i 999
bandung-leaf#
bandung-leaf# show run | i 100999
bandung-leaf#
bandung-leaf# █

```

**Figure 21.** Verify before script `check_vlan3.py` executed

Initially, the VLAN is inputted into the Python script, yet the VNI remains unconfigured on the device. After entering the VNI in the script, we confirmed its

```

bandung-leaf# show vlan br | i 999
999 VLAN999                active   Eth1/2, Eth1/7
bandung-leaf#
bandung-leaf#
bandung-leaf# show run | i 100999
bandung-leaf#
bandung-leaf# █

```

**Figure 22.** Capture after input vlan from `check_vlan3.py`

validity by checking the devices. The output displayed "*member vni 100999*" under the interface *vni1* and "*vni 100999 l2*" in the EVPN configuration, which indicates that the VNI has been successfully integrated. To confirm the proper operation of

```

bandung-leaf#
bandung-leaf# show vlan br | i 999
999 VLAN999                               active   Eth1/2
bandung-leaf#
bandung-leaf# show run | i 100999
vn-segment 100999
member vni 100999
vni 100999 l2
bandung-leaf#
bandung-leaf#

```

Figure 23. Capture after input vlan and VNI from check\_vlan3.py

VXLAN, a testing procedure is executed on the leaf nodes. To determine connectivity and functionality, the command "*show bgp l2vpn evpn | begin 100999*" is executed, followed by completing ping tests from each node.

```

bandung-leaf# show bgp l2vpn evpn | begin 100999
Route Distinguisher: 172.16.100.102:33766 [L2PM 100999]
=>[12]:[0]:[0]:[40]:[0050.0100.8403]:[0]:[0.0.0.0]/216 100 0 1
=>[13]:[0]:[32]:[172.16.200.182] 100 32768 1
=>[13]:[0]:[32]:[172.16.200.181] 100 0 1
=>[13]:[0]:[32]:[172.16.200.182] 100 0 1
Route Distinguisher: 172.16.100.102:33766
=>[13]:[0]:[32]:[172.16.200.182]/0 100 0 1
Route Distinguisher: 172.16.100.102:33766
=>[12]:[0]:[0]:[40]:[0050.0100.8403]:[0]:[0.0.0.0]/216 100 0 1
=>[13]:[0]:[32]:[172.16.200.182]/0 100 0 1

jakarta-leaf# show bgp l2vpn evpn | begin 100999
Route Distinguisher: 172.16.100.102:33766 [L2PM 100999]
=>[12]:[0]:[0]:[40]:[0050.0100.8403]:[0]:[0.0.0.0]/216 100 32768 1
=>[13]:[0]:[32]:[172.16.200.182] 100 0 1
=>[13]:[0]:[32]:[172.16.200.181] 100 0 1
=>[13]:[0]:[32]:[172.16.200.182] 100 32768 1
Route Distinguisher: 172.16.100.102:33766
=>[13]:[0]:[32]:[172.16.200.182]/0 100 0 1
Route Distinguisher: 172.16.100.102:33766
=>[13]:[0]:[32]:[172.16.200.181] 100 0 1

```

Figure 24. Verify VXLAN is working on selected nodes

```

user@node-1i-5 ifconfig | grep 10.10.10.
    inet 10.10.10.1 netmask 255.255.255.0 broadcast 10.10.10.255
user@node-1i-4 ping 10.10.10.2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data:
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=0.97 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.63 ms
64 bytes from 10.10.10.2: icmp_seq=3 ttl=64 time=0.28 ms
64 bytes from 10.10.10.2: icmp_seq=4 ttl=64 time=0.72 ms
64 bytes from 10.10.10.2: icmp_seq=5 ttl=64 time=0.29 ms
^C
-- 10.10.10.2 ping statistics --
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 0.2867/0.572/0.2709/0.683 ms
user@node-1i-5

user@node-2i-5 ifconfig | grep 10.10.10.
    inet 10.10.10.2 netmask 255.255.255.0 broadcast 10.10.10.255
user@node-2i-8 ping 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data:
64 bytes from 10.10.10.1: icmp_seq=1 ttl=64 time=6.97 ms
64 bytes from 10.10.10.1: icmp_seq=2 ttl=64 time=5.82 ms
64 bytes from 10.10.10.1: icmp_seq=3 ttl=64 time=7.67 ms
64 bytes from 10.10.10.1: icmp_seq=4 ttl=64 time=6.39 ms
64 bytes from 10.10.10.1: icmp_seq=5 ttl=64 time=7.71 ms
^C
-- 10.10.10.1 ping statistics --
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 5.816/6.998/7.712/0.742 ms
user@node-2i-5

```

Figure 25. Verify VXLAN is working to between customer from each node proven by reachable ping

#### 4.4.3 Backup Process and Procedures

The network's configuration management is enhanced by an Ansible playbook called *backup\_configuration.yml*, specifically built to store and safeguard configurations. After the configuration process is complete, the collected configuration files are stored in a specified local directory. Furthermore, a *crontab* scheduler is set up to commence the backup procedure four times daily.

The ansible playbook, *backup\_configuration.yml*, is scheduled to execute at four predetermined intervals during the day: 00:00, 06:00, 12:00, and 18:00, guaranteeing systematic and uninterrupted backups. A specific 10-minute timeframe is allocated for the initial backup processing of the day, which involves generating a fresh folder for the backups of that particular day. After the initial backup, subsequent backups only need a 5-minute timeframe after executing the playbook. Therefore, the designated

```

- name: Copy run start
  ios_command:
    commands:
      - "copy run start"

- name: Show Running Config
  ios_command:
    commands:
      - "show running-config"
  register: config

- name: Save Output
  delegate_to: localhost
  copy:
    content: "{{ config.stdout[0] }}"
    dest: "/etc/ansible/backup_configuration/{{ current_date }}/{{ current_time }}/{{ i
nventory_hostname }}.config"
arfan@jumphost-22:/etc/ansible$ █

```

Figure 26. Ansible playbook task

```

0 00 * * * /usr/bin/ansible-playbook -i /etc/ansible/hosts.yml /etc/ansible/backup_configu
5 00 * * * sh /etc/ansible/chown_from_root_to_arfan.sh
0 06 * * * /usr/bin/ansible-playbook -i /etc/ansible/hosts.yml /etc/ansible/backup_configu
0 12 * * * /usr/bin/ansible-playbook -i /etc/ansible/hosts.yml /etc/ansible/backup_configu
0 18 * * * /usr/bin/ansible-playbook -i /etc/ansible/hosts.yml /etc/ansible/backup_configu

```

Figure 27. Crontab scheduler for ansible backup configuration task

```

10 0 * * * sh /etc/ansible/push_to_git.sh
5 6 * * * sh /etc/ansible/push_to_git.sh
5 12 * * * sh /etc/ansible/push_to_git.sh
5 18 * * * sh /etc/ansible/push_to_git.sh

```

Figure 28. Crontab scheduler for push to github repository

times for these backups are 00:10, 06:05, 12:05, and 18:05. After the day, we will validate our GitHub repository.

Name	Last commit message	Last commit date
..		
00-00	Updated the repository with latest changes on Wed 13 Dec 00:10:03 WIB...	2 days ago
06-00	Updated the repository with latest changes on Wed 13 Dec 06:05:03 WIB...	yesterday
12-00	Updated the repository with latest changes on Wed 13 Dec 12:05:04 WIB...	yesterday
18-00	Updated the repository with latest changes on Wed 13 Dec 18:05:04 WIB...	yesterday

Figure 29. Verification backup configuration uploaded to github repository

## 5. Conclusions

This project successfully enhanced network infrastructure by integrating VXLAN technology with advanced automation techniques using Python, Ansible, and Git for efficient configuration backups. By implementing a spine and leaf architecture within the Eve-NG environment, significant improvements were observed in network scalability and flexibility, essential attributes for the evolving demands of modern data centers. The application of Python in automating network operations notably streamlined the configuration and management of circuits, while Ansible's role in handling configuration backups added a layer of reliability. Additionally, the use of automated backups, paired with Git's version control system, ensured the security, timeliness, and accessibility of network configurations. Throughout the testing and validation phase, the system's durability stood out, with the comprehensive evaluation affirming its precision, reliability, and robustness. The integration of VXLAN with Python and Ansible represents a significant advancement in addressing the complexities and evolving needs of contemporary network infrastructures, delivering a scalable, efficient, and reliable framework for network management.

## References

- [1] Mallik Mahalingam et al. *Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*. RFC 7348. Aug. 2014. doi: 10.17487/RFC7348. URL: <https://www.rfc-editor.org/info/rfc7348>.
- [2] Chang-Gyu Lim et al. "Design and implementation of hardware accelerated VTEP in datacenter networks". In: *2015 17th International Conference on Advanced Communication Technology (ICACT)*. 2015, pp. 745–748. doi: 10.1109/ICACT.2015.7224894.
- [3] Joaquin Alvarez-Horcajo et al. "Scaling and Interoperability of All-Path with Bridged and SDN Domains using VXLANs". In: *2019 IEEE 44th Conference on Local Computer Networks (LCN)*. 2019, pp. 97–100. doi: 10.1109/LCN44214.2019.8990814.
- [4] Claudiu Trăistaru. "VXLAN - A practical approach to cloud computing scalability". In: *2023 22nd RoEduNet Conference: Networking in Education and Research (RoEduNet)*. 2023, pp. 1–4. doi: 10.1109/RoEduNet60162.2023.10274934.
- [5] Zhifeng Zhao, Feng Hong, and Rongpeng Li. "SDN Based VxLAN Optimization in Cloud Computing Networks". In: *IEEE Access* 5 (2017), pp. 23312–23319. doi: 10.1109/ACCESS.2017.2762362.
- [6] Brenden Buresh ; Dan Eline;David Jansen; Jason Gmitter; Jeff Ostermiller; Jose Moreno; Kenny Lei; Lilian Quan; Lukas Krattiger; Max Ardica; Rahul Parameswaran; Rob Tappenden; Satish Kondalam. *A modern, open and scalable fabric VXLAN EVPN*. Cisco, 2017.

- [7] Aptin Babaei, Parham M. Kebria, and Saeid Nahavandi. “A survey on Automation Technologies used in Network Control and Management”. In: *2022 15th International Conference on Human System Interaction (HSI)*. 2022, pp. 1–6. doi: 10.1109/HSI55341.2022.9869444.
- [8] Daniele Bringhenti; Guido Marchetto; Riccardo Sisto; Fulvio Valenza. “Automation for Network Security Configuration: State of the Art and Research Trends”. In: *ACM Comput. Surv.* (2023).
- [9] Mohammad Mehedi Hassan and Akond Rahman. “As Code Testing: Characterizing Test Quality in Open Source Ansible Development”. In: *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*. 2022, pp. 208–219. doi: 10.1109/ICST53961.2022.00031.
- [10] Lorin Hochstein; Rene Moser. *Ansible: Up and Running*. O’Reilly Media, 2017.
- [11] Jeff Geerling. *Ansible for DevOps: Server and Configuration Management for Humans*. Leanpub, 2015.
- [12] Julio Sandobalín, Emilio Insfran, and Silvia Abrahão. “On the Effectiveness of Tools to Support Infrastructure as Code: Model-Driven Versus Code-Centric”. In: *IEEE Access* 8 (2020), pp. 17734–17761. doi: 10.1109/ACCESS.2020.2966597.
- [13] Nishant Kumar Singh et al. “Automated provisioning of application in IAAS cloud using Ansible configuration management”. In: *2015 1st International Conference on Next Generation Computing Technologies (NGCT)*. 2015, pp. 81–85. doi: 10.1109/NGCT.2015.7375087.