International Journal of Electronics and Communications System
Volume, Issue, 11-20.
ISSN: 2798-2610
http://ejournal.radenintan.ac.id/index.php/IJECS/index
DOI: *10.24042/ijecs.v3i1.16524*

Automation of Open VSwitch-Based Virtual Network Configuration Using Ansible on Proxmox Virtual Environment

Khairan Marzuki *
Universitas Bumigora Mataram, INDONESIA

Muhammad Idham Kholid
Universitas Bumigora Mataram, INDONESIA

I Putu Hariyadi
Universitas Bumigora Mataram, INDONESIA

Lalu Zazuli Azhar Mardedi
Universitas Bumigora Mataram, INDONESIA

Article Info

Article history:

Received: April 18, 2023
Revised: June 7, 2023
Accepted: June 28, 2023

Keywords:

Ansible;
Automation;
Open Vswitch;
Proxmox;
Virtualization;
VLAN;
Vxlan.

Abstract

Proxmox has a feature that can build a private network in it. Each host on a private network on Proxmox generally shares physical resources, including network connections using a virtual network, one of which is a VLAN. The Proxmox server supports Open VSwitch as a virtual switch. Open virtual switch, an alternative virtual switch quite popular among cloud developers, can be a solution for managing traffic between Virtual Machins and external communications. The method used in this study is the Network Development Life Cycle (NDLC). After applying the automation engine using Ansible, it overcame the problem, namely speeding up configuration and reducing human error or errors in configuring virtual networks. The automation system can speed up the virtual network management process compared to the manual method based on 5 (five) experiments, namely when the manufacturing operation has an average time of 08 minutes and 42 seconds faster. Whereas when the addition operation has an average faster time of 08 minutes 17 seconds. On the other hand, when the deletion operation has an average time of 42 seconds faster.

To cite this article: K. Marzuki, M. I. Kholid, I. P. Hariyadi, I. P. Hariyadi, and L. Z. A. Mardedi, "Automation of Open VSwitch-Based Virtual Network Configuration Using Ansible on Proxmox Virtual Environment," *Int. J. Electron. Commun. Syst.,* vol. 3, no. 1, pp. 11-20, 2023.

INTRODUCTION

Nowadays, virtualization technology is trending and is a necessity. Today many companies are already using virtualization technology on servers. Because this virtualization technology aims to avoid wasting expensive processing power or, in other words increasing efficiency and optimizing the use of more than one core processor, another saving is electricity costs because it only uses one or a few servers [1], [2]. Various kinds of software used to manage virtualization are known as hypervisors. The most widely used hypervisor is Proxmox VE. The Proxmox VE is an open-source virtual operating system widely used by virtualization users. Proxmox has many features, including a virtual machine [3]. The virtual machine functions to run the OS just like we use a real machine. Each VM resides on a single host, so they share physical resources, including network connections.

Virtual networking is a network in which several virtual networks are built on the physical network. On this network, we can create and manage multiple virtual networks at the software level without disturbing each other [4]–[6] and allows multiple heterogeneous virtual networks to coexist on a physical basis [7]. VSwitch connects network interface controllers (NICs) and VMs to

• **Corresponding author:**
Khairan Marzuki, Universitas Bumigora Mataram, INDONESIA. ✉ khairan.marzuki@universitasbumigora.ac.id

ensure network connection and virtual machine topology discovery [8]. Virtual networking connects each virtual machine that is built to a virtual switch port. The Proxmox server supports Open VSwitch as a virtual switch. Open virtual switch (OVS), an alternative virtual switch quite popular among cloud developers, can be a solution for managing traffic between VMs with communication from outside. Virtualization can be implemented in various types, such as Network Virtualization, namely VLAN. VLAN (Virtual Local Area Network) is a technology that can configure a logical network independent of the physical network structure[2], [9]. On manageable switches, VLAN configurations can be performed. VLANs have a limited number of configurable 1 switches, namely 4096 VLANs. According to researchers, this number is very large, but it is different if the VLAN is applied to ISPs (Internet Service Providers) with numerous customers or providers. Cloud AWS or Google Cloud, which has customers worldwide, needs a solution to overcome the problem of how to segment the network with the number of 4096, namely by implementing VXLAN (Virtual eXtensible Local Area Network). VXLAN has a 24-bit Segment ID called VNI (Virtual Network identifier) so that it can allow 16 million VXLAN segments that can be created in one administrative domain [10]–[12].

Manage open VSwitch-based virtual networking, which includes creating VLANs and VXLANs. It requires accuracy and a range of human errors occurs, and what about the customer needs of providers who already have customers all over the world, namely in leasing cloud servers, when cloud server rentals at providers increase, so that the configuration of virtual network devices following the many cloud server leases. Managing the configuration of virtual network devices manually and repeatedly becomes inefficient, takes a long time, and ranges from human error [13]–[15]. For this reason, a solution is needed so that the manufacturing process becomes faster and minimizes manual configuration errors, manufacturing time, and human error. The implementation of configuration management using the Ansible tool can streamline the manual configuration process. Automation replaces human power with machine power that automatically performs and manages work so that it no longer requires human supervision (in industry and so on) [16]. Ansible is a simple Information Technology (IT) automation engine that can automate cloud provisioning, configuration management, application deployment, intra-service orchestration, and other IT needs [17], [18].

Ansible automation is built in the form of a playbook file containing tasks related to Virtual networking, namely VLAN and VXLAN. Trials related to the playbook that are made include functionality trials. With this research, the authors would like to provide a solution for configuring Open VSwitch-based virtual networks on Proxmox ve to be more structured and efficient. They also provide insight regarding open VSwitch-based virtual network automation on Proxmox.

METHOD

The research method used in this study is the Network Development Life Cycle (NDLC). Of the six stages in NDLC, the authors only use three stages, namely Analysis, Design, and Simulation Prototyping, in Figure 1.

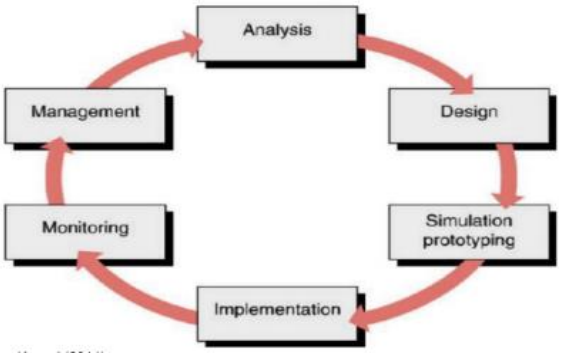

Figure 1. NDLC Methodology

Analysis

At the Analysis stage, the researcher collected data by studying the literature, namely several scientific articles that discussed open VSwitch and the application of virtual

local area networks on virtual machine networks. In addition, the authors also collect data and information from various sources such as the internet, books, papers, e-books, and scientific articles. Based on the results of data collection, research related to Open VSwitch as a Virtual Network [19], [20] and Ansible tool as an automation tool[21]–[23] have been done before. In the open VSwitch research, previous research focused on the application and performance analysis of the Open VSwitch [24]–[26]. However, Open VSwitch has not been implemented on the Proxmox server. In previous research, it was implemented manually and had not been implemented using the Ansible tool as an automation tool. And in previous Ansible tool research focusing on implementing configuration management on cloud servers[27], it has not been implemented on the Proxmox Ve server. This research will focus on implementing Open VSwitch on the Proxmox server and automating configuration management using the Ansible tool.

Design stages

This stage makes a design that includes a trial network design, interface design on PVE1 and PVE2 servers, IP safety design, automation system design, and hardware and software requirements.

Trial Network Design

The test network design used in this study is shown in Figure 2.

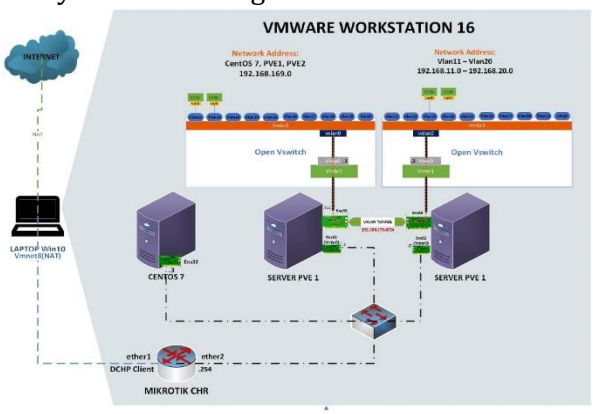

Figure 2. Test Network Design

The design of the test network is simulated using virtualization on one computer with the Windows 10 operating system in which the VMware Workstation 16 hypervisor is installed. In the VMware Workstation, 4 (four) Virtual Machines (VMs) are created, namely Centos 7, Server Proxmox Virtual Environment 1 (PVE1) and Server PVE2, and Mikrotik CHR. Centos 7 is an Ansible server that is used as an automation engine. In comparison, the PVE1 and PVE2 Servers are the targets of the automation engine, namely virtual network configuration management based on automated Open VSwitch (OVS). Finally, Mikrotik CHR is an Internet gateway for the three VMs connected to the Internet network.

Virtual Network Interface Design on PVE1 and PVE2 Servers

The virtual network interface design will be implemented on the PVE1 server, as shown in Figure 3.

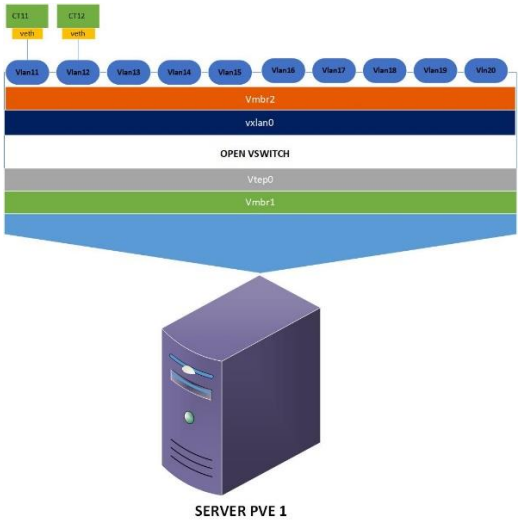

Figure 3. PVE1 Virtual Network Server Interface Design

In the design of the virtual network interface for the pve1 server, there is vmbr1 as an interface that is directly connected to the physical interface of the pve1 server, namely ens33, interface vtep0 as the endpoint interface of the overlay network or virtual network path, interface vxlan0 as a tunneling interface or tunnel for inter-VLAN communication lines different servers, the vmbr2 interface is the connecting interface between VLAN11 to

VLAN20. The VLAN11 and 20 interfaces connect containers, namely CT11 and CT12, on the pve1 server. While the virtual network interface design will be implemented on the PVE2 server, as shown in Figure 4.

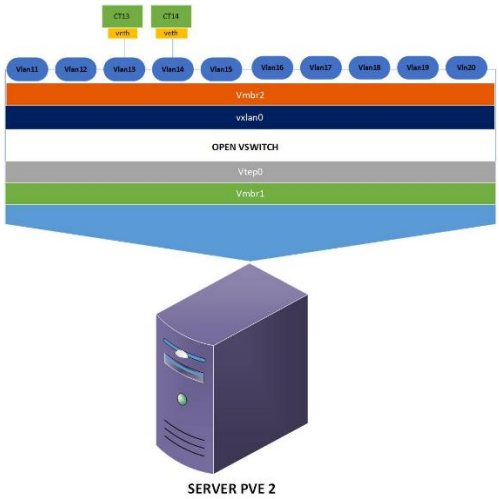

Figure 4. PVE2 Virtual Network Server Interface Design.

In the design of the virtual network interface for the pve2 server, there is vmbr1 as an interface that is directly connected to the physical interface of the pve2 server, namely ens33, interface vtep0 as the endpoint interface of the overlay network or virtual network path, interface vxlan0 as a tunneling interface or tunnel for inter-VLAN communication lines different servers, the vmbr2 interface is the connecting interface between VLAN11 to VLAN20. The VLAN11 and 20 interfaces connect containers, namely CT13 and CT14, on the pve2 server.

Virtual Network Automation System Design

The design of an Open VSwitch (OVS) based virtual network automation system will be implemented on each PVE server using Ansible, as shown in Figure 5.

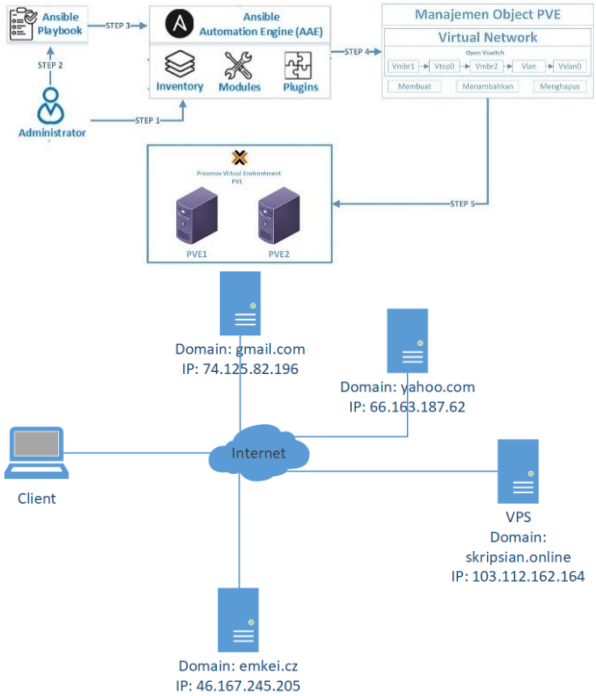

Figure 5. Automation System Design

There are 5 (five) stages in the design of the virtual network automation system, namely the first stage is to define the inventory by the Administrator on CentOS 7 regarding the target machines to be automated, namely the PVE1 and PVE2 servers. In the second stage, the Administrator creates a playbook file in YAML format, which contains tasks that will be executed according to the test scenario: creating, adding, and deleting OVS-based virtual networks on each PVE server. The third stage is that the playbook that has been created is executed using the Ansible Automation Engine (AAE)[28]. The execution of the third phase will impact stages four and five, namely OVS-based virtual network object management on each PVE server.

IP Address Design

In the design of IP addresses for virtual networking automation test networks on servers pve1 and pve2, there are 4 (four) class C network addresses that are used, namely 192.168.169.0/24, 192.168.11.0/24, 192.168.12.0/24, 192.168.13.0 /24. Detailed information on the allocation of IP addresses

for each virtual machine and client Windows 10 can be seen in Table 1.

Table 1. Design of IP Addresses

No	NamaVirtual Machine	Interface	IP Address	SubnetMask
1	*Centos 7*	Ens33	192.168.169.3	255.255.255.0
2	*Server Pve1*	Ens32	192.168.169.1	255.255.255.0
3	*Server Pve2*	Ens32	192.168.169.2	255.255.255.0
4	*Mikrotik CHR*	Ether2	192.168.169.254	255.255.255.0
5	*Container 111*	veth	192.168.11.111	255.255.255.0
6	*Container 112*	veth	192.168.12.112	255.255.255.0
7	*Container 113*	veth	192.168.13.113	255.255.255.0
8	*Container 114*	veth	192.168.14.114	255.255.255.0
9	*Client Win 10*	Vmnet1	192.168.169.253	255.255.255.0

Hardware and Software Requirements

The hardware and software requirements in this research are Hardware Requirements. 1 (one) laptop with the following specifications, Intel Core i3-6006U Processor 2.00GHz CPU, 250GB SSD, 1TB Hard Drive, 12GB Memory, Windows 10 Operating System On the Windows operating system. The laptop is installed with VMware Workstation 16, and 4 (four) Virtual Machines (VM) were created, namely the PVE1 server, PVE2 server, Ansible server, and Mikrotik CHR. Each VM has the following specifications, one pve1 server and pve2 server: 1 core processor, 40 GB hard drive, Memory: 3 GB, 1 Ansible server, one core processor, 20 GB hard drive, 2 GB memory, 1 Mikrotik CHR, one core processor, 64 MB hard drive, 256 MB memory.

Software Requirements, Proxmox Virtual Environment, or Proxmox VE, is an operating system used on PVE1 and PVE2 servers, Centos 7 operating system used as an automation engine for PVE1 servers and PVE2 servers, Linux Container Template CentOS 7 as OS on computer servers and used for an operating system installation on CT on PVE1 servers and PVE2 servers.

VMware Workstation 16 is software for running virtual machine centos7, CHR proxy, PVE1 servers, and PVE2 servers. Mikrotik CHR is an operating system used as an internet gateway so virtual machines can connect to the internet network, Puty is software used to remote servers, and Google Chrome Browser is used to access the PVE server GUI web dashboard.

Simulation Prototyping

A trial simulation will be carried out at this stage, including installation and configuration on each virtual machine and testing scenarios on the system being built.

Installation and Configuration Stage

At this stage, installation and configuration will be carried out for each virtual machine based on the trial design. PVE1 and PVE2 server installation and configuration include IP addressing and installation of packages such as open VSwitch, Python-pip, Proxmoxer, and ifupdown2. Centos 7 server installation and configuration include IP addressing and installation of packages such as Epel-release, Ansible, Jinja2, and creation of the Ansible playbook. Configuration on the client includes IP address configuration so the client can access the Graphical User Interface (GUI) web from the PVE server.

Trial Scenario

At this stage, it contains the installation and configuration test results using several trial scenarios in carrying out test scenarios such as creating, adding, and deleting virtual networks manually and automatically. It also calculates the time needed to manually create and automate virtual network creation. The details of the trial scenario are shown in Table 2.

Table 2. Trial Scenario

Trials	Process	Deskription
Five trials on creating, adding, and deleting on server pve1 and server pve2	Manual (through web GUI *PVE1* and *PVE2*)	Calculate the minimum, and maximum, and average time of creating, adding, and deleting virtual networks.
Five trials in creating, adding, and deleting on server pve1 and server pve2	Automation (Created using the *Ansible*)	Calculate the minimum, and maximum, and average time of creating, adding, and deleting virtual networks.

RESULTS AND DISCUSSION

The system design made at the design stage is outlined as an Ansible playbook. There are 3 (three) Ansible playbook structures generated to automate OVS-based virtual network configuration management, namely the playbook for the virtual network creation (initialization) process, the playbook for updating (adding) virtual networks, and the playbook for deleting virtual networks. Ansible playbook structure for creating virtual networks, as shown in Figure 6.

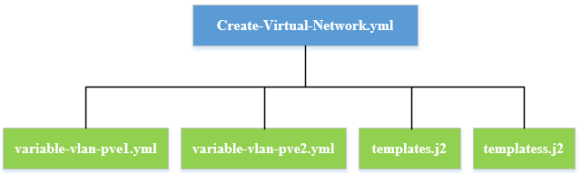

Figure 6. The Structure of Ansible Playbook in Creating Virtual Network

The Create-Virtual-Network.yml file is an Ansible playbook that contains tasks related to creating an OVSBridge type bridge on each PVE server with the names vmbr1 and vmbr2, and vtep0 of type OVSIntPort including setting the IP address on the interface. In addition, it also loads the task of creating 10 (ten) VLANs with IDs 11 to 20, including the IP address on the interface with data sourced from the variable-VLAN-pve1.yml and variable-VLAN-pve2.yml files and creating a VXLAN interface type OVSTunnel on each PVE server to form a tunnel so that hosts on the same VLAN but on different PVE servers can still communicate with each other. Making the vxlan0 interface on the PVE1 server using the Jinja2 template stored in the templates.j2 file. Whereas on the PVE2 server, it uses the templatess.j2 file. Finally, the Ansible playbook also contains a task to restart the networking service to activate virtual network configuration changes to the /etc/network/interfaces file. Snippet content from the Ansible playbook file Create-Virtual-Network.yml with tasks executed on the PVE1 server, as shown in Figure 7.

```
---
- name: Creat Virtual Network Server PVE1
  hosts: pve1
  gather_facts: false
  vars_files: variabel-vlan-pve1.yml
  tasks:
  - name: vmbr1
    command: pvesh create /nodes/pve1/network -iface "vmbr1" -type "OVSBridge" -
ovs_ports "ens33" -mtu "1554" -autostart

  - name: vtep0
    command: pvesh create /nodes/pve1/network -iface "vtep0" -type "OVSIntPort" -address
"192.168.170.1" -netmask "255.255.255.0" -ovs_bridge "vmbr1" -mtu "1554" -autostart

  - name: set MTU ens33
    command: pvesh set /nodes/pve1/network/ens33 -type "OVSPort" -mtu "1554"

  - name: vmbr2
    command: pvesh create /nodes/pve1/network -iface "vmbr2" -type "OVSBridge"  -
autostart

  - name: mengatur rentang vlans
    set_fact:
     vlans: "{{ vlans | default([]) + [item] }}"
    with_sequence: start=11 end=20

  - name: looping pembuatan vlans
    command: pvesh create /nodes/pve1/network -iface "{{ item.0.ifaces }}" -type
"OVSIntPort" -address "{{ item.0.ip }}" -netmask "{{ item.0.mask }}" -ovs_bridge "vmbr2"
-ovs_tag "{{ item.0.tag }}" -autostart
    with_together:
     - "{{ variabel_vlan_pve1 }}"
     - "{{ vlans }}"

  - name: replace file interface
    command: mv /etc/network/interfaces.new  /etc/network/interfaces

  - name: Reload Configuration Network Interface PVE1
    command: systemctl restart networking

  - name: vxlan tunnel Server PVE1
    template: src=templates.j2 dest=/etc/network/interfacesvxlan0

  - name: copy file interface
    command: cp /etc/network/interfaces  /etc/network/interfaces.new

  - name: append vxlan
    shell: /usr/bin/cat /etc/network/interfacesvxlan0 >> /etc/network/interfaces.new

  - name: vxlan0
    command: sed -i 's|vlan19 vlan20|vlan19 vlan20 vxlan0|g' /etc/network/interfaces.new

  - name: replace file interface
    command: mv /etc/network/interfaces.new  /etc/network/interfaces

  - name: Reload Configuration Network Interface PVE1
    command: systemctl restart networking
```

Figure 7. Ansible Playbook File Content Snippet

Creating a Virtual Network for PVE1 Server.

Meanwhile, the Ansible playbook structure for adding (updating) virtual networks is shown in Figure 8.

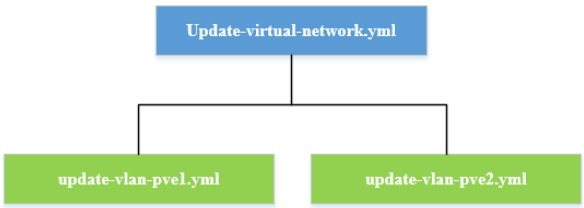

Figure 8. Ansible Playbook Structure Update

The main playbook file from the process of adding or updating the virtual network is called Update-virtual-network.yml, which contains tasks including the addition of 10 (ten)

new VLANs with IDs 21 to 30, including the IP address on the interface with data sourced from the update file -VLAN-pve1.yml to deploy on PVE1 servers and update-VLAN-pve2.yml to deploy on PVE2 servers. Besides that, it also loads a task to restart the networking service so that the virtual network configuration changes made to the /etc/network/interfaces file can be activated. In contrast, the Ansible playbook structure for deleting virtual networks is shown in Figure 9.

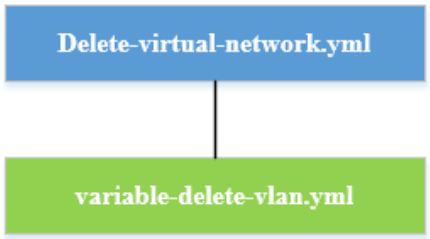

Figure 9. The structure of Ansible Playbook in Creating Virtual Network.

The main playbook file of the virtual network deletion process is called Delete-virtual-network.yml, which contains tasks covering all VLANs on each PVE server[3] with data sourced from the variable-delete-VLAN.yml file. Besides that, it also loads the task to delete the vmbr1, vmbr2, and vxlan0 bridge interfaces. It restarts the networking service so that the virtual configuration changes to the /etc/network/interfaces file can be activated.

Manual and Automated Virtual Network Configuration System Testing.

Testing of the virtual network configuration system was carried out either manually through the web GUI of each PVE server or by automation for 5 (five) attempts. Testing automation is done by executing the command "ansible-playbook Create-Virtual-Network.yml" on the CentOS 7 server, which functions as the Ansible control machine. The snippet of the results of verifying the creation of the virtual network automatically via the Web GUI from the PVE1 server as shown in Figure 10.

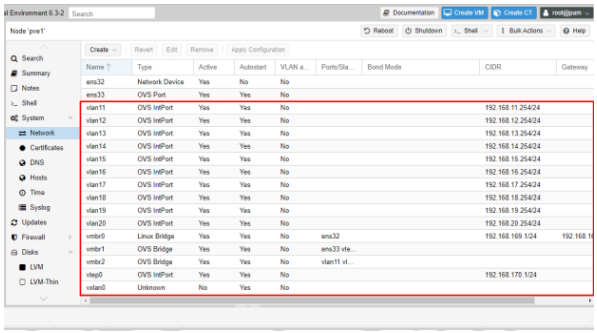

Figure 10. Automated Verification of Virtual Network Creation Results on Server PVE1.

It can be seen that virtual network interfaces, including vmbr1, vmbr2, vtep0, vxlan0, and ten VLANs, each with ID 11 and 20, have been successfully created, including the IP address settings on each of these interfaces. While the operation to add a virtual network is carried out by executing the command "ansible-playbook Update-virtual-network.yml." On the other hand, the virtual network deletion operation is carried out by executing the command "ansible-playbook Delete-virtual-network.yml" in the terminal from the CentOS 7 server.

Analysis of Virtual Network Configuration Automation System.

The following analysis is obtained based on the test results of the OVS-based virtual network configuration automation system that has been carried out on 2 (two) PVE servers. The created Ansible Playbook can be used to automate the OVS-based virtual network creation process on each PVE server and add ten virtual networks new VLANs, and deletion of the virtual network. Connection verification test between containers on the same VLAN but different PVE servers was successfully carried out using the ping utility to indicate VXLAN is functioning properly. Comparison of time for creating OVS-based virtual networks in each PVE server conducted 5 (five) trials manually and automatically Table 3.

Tabel 3. Perbandingan Waktu Pembuatan Jaringan Virtual Manual dan Otomatis pada Server PVE1 & PVE2

Trial Create	Time	
	Manuals	Automatic
First	12 Minute 18 Second	02 Minute 36 Second
Second	11 Minute 53 Second	02 Minute 25 Second
Third	11 Minute 11 Second	02 Minute 23 Second
Fourth	10 Minute 06 Second	02 Minute 20 Second
Fifth	10 Minute 10 Second	02 Minute 21 Second
Average	11 Minute 08 Second	02 Minute 25 Second
Minimun (Fastesttime)	10 Minute 06 Second	02 Minute 20 Second
Maximum (Longest time)	12 Minute 18 Second	02 Minute 36 Second

Table 3 shows that the average time for manually creating a virtual network is 11 minutes and 8 seconds, while for automation, it is 02 minutes and 25 seconds. Creating a virtual network automatically is 08 minutes 42 seconds faster than manually. On the other hand, the fastest time for creating a virtual network manually was 10 minutes 06 seconds, and the longest was 12 minutes 18 seconds. Meanwhile, automatically, the fastest production time is 02 minutes 20 seconds, and the longest is 02 minutes 36 seconds. The time comparison for the process of adding an OVS-based virtual network on each PVE server is 10 (ten) VLANs with IDs 21 to 30, which were carried out in 5 (five) trials both manually and automatically using the Ansible playbook, as shown in Table 4.

Table 4. Comparison of Time to Add VLANs on Server PVE1 & Server PVE2

Trial Create	Time	
	Manuals	Automatic
First	11 minutes 00 seconds	02 minutes 01 seconds
Second	10 minutes 23 seconds	01 minute 51 seconds
Third	08 minutes 44 seconds	01 minute 41 seconds
Fourth	10 minutes 23 seconds	01 minute 43 seconds
Fifth	09 minutes 51 seconds	01 minute 40 seconds
Average	10 minutes 04 seconds	01 minute 47 seconds
Minimum (Fastest time)	08 minutes 44 seconds	01 minute 40 seconds
Maximum (Longes time)	11 minutes 00 seconds	02 minutes 01 second

Table 4 shows that the average time to manually add a virtual network is 10 minutes 4 seconds, while automatically, it is 01 minutes 47 seconds. Automatically adding a virtual network is 08 minutes 17 seconds faster than manually. On the other hand, the fastest time to add a virtual network manually was 08 minutes 44 seconds, and the longest was 11 minutes. Meanwhile, the fastest addition time automatically is 01 minutes 40 seconds, and the longest is 02 minutes 01 seconds. Comparison of the time for the OVS-based virtual network deletion process on each PVE server which was carried out 5 (five) attempts manually and automatically using the Ansible playbook, as shown in Table 5.

Table 5. Comparison of Time to Delete Virtual Network on Server PVE1 & Server PVE2.

Try to Delete	Time	
	Manuals	Automatic
First	03 Minute 48 Second	02 Minute 37 Second
Second	03 Minute 03 Second	02 Minute 41 Second
Third	03 Minute 39 Second	02 Minute 36 Second
Fourth	03 Minute 10 Second	02 Minute 38 Second
Fifth	03 Minute 05 Second	02 Minute 43 Second
Average	03 Minute 21 Second	02 Minute 39 Second
Minimum (Fastest time)	03 Minute 03 Second	02 Minute 36 Second
Maximum (Longes time)	03 Minute 36 Second	02 Minute 43 Second

CONCLUSION

Based on the results of the trials that have been carried out, it can be concluded that the automation system created using Ansible was successfully used to manage OVS-based virtual network configurations on 2 (two) PVE servers,

including the operations of creating and adding and deleting both bridges, VLANs, and VXLANs. The automation system can speed up the virtual network management process compared to the manual method based on 5 (five) experiments, namely when the manufacturing operation has an average time of 08 minutes and 42 seconds faster. Whereas when the addition operation has an average faster time of 08 minutes 17 seconds. On the other hand, when the deletion operation has an average time of 42 seconds faster.

SUGGESTION

The suggestions for developing this research further are to apply Ansible Roles to enable the development of reusable OVS-based virtual network configuration management automation system components, in addition to implementing OVS-based virtual network configuration management automation on cluster-based PVE systems.

REFERENCES

[1] Y. R. Adi, O. D. Nurhayati, and E. D. Widianto, "Perancangan Sistem Cluster Server untuk Jaminan Ketersediaan Layanan Tinggi pada Lingkungan Virtual," *J. Nas. Tek. Elektro dan Teknol. Inf.*, vol. 5, no. 2, 2016, doi: 10.22146/jnteti.v5i2.228.

[2] K. Marzuki, N. Hanif, and I. P. Hariyadi, "Application of Domain Keys Identified Mail, Sender Policy Framework, Anti-Spam, and Anti-Virus: The Analysis on Mail Servers," *Int. J. Electron. Commun. Syst.*, vol. 2, no. 2, pp. 65–73, 2022, doi: 10.24042/ijecs.v2i2.13543.

[3] M. Simon and L. Huraj, "VirtualBox and Proxmox VE in Network Management: A User-Centered Comparison for University Environments," pp. 486–495, 2023, doi: 10.1007/978-3-031-35317-8_44.

[4] M. Geo, U. Putra, K. Utomo, F. T. Elektro, and U. Telkom, "Perancangan Dan Analisis Performansi Open VSwitch Untuk Jaringan Virtual Universitas Telkom," vol. 2, no. 2, pp. 2705–2712, 2015.

[5] C. N. Bangun, "Jaringan Komputer Cindya Novira Bangun 212406025 Kom A'21 Program Studi D-3 Teknik Informatika Fakultas Vokasi Universitas Sumatera Utara Medan 2022," no. January, 2023.

[6] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual Network Embedding: A Survey," *IEEE Commun. Surv. Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013, doi: 10.1109/SURV.2013.013013.00155.

[7] Q. Lu, K. Nguyen, and C. Huang, "Distributed parallel algorithms for online virtual network embedding applications," *Int. J. Commun. Syst.*, vol. 36, no. 1, pp. 1–24, 2023, doi: 10.1002/dac.4325.

[8] Y. Yang *et al.*, "C2QoS: Network QoS guarantee in VSwitch through CPU-cycle management," *J. Syst. Archit.*, vol. 116, no. April, p. 102148, 2021, doi: 10.1016/j.sysarc.2021.102148.

[9] W. N. Hidayat *et al.*, "Mikrotik Training to Improve Computer Network Administration Competence for MTCNA Certification Preparation for Teachers and Students at SMKN 10 Malang," *J. Abdimas Berdaya*, vol. 6, pp. 38–44, 2023.

[10] P. Rito *et al.*, "Aveiro Tech City Living Lab: A Communication, Sensing and Computing Platform for City Environments," in *ArXiv*, 2022, pp. 1–21.

[11] R. L. Bull and J. N. Matthews, "Critical Analysis of Layer 2 Network Security in Virtualised Environments," *Int. J. Commun. Networks Distrib. Syst.*, vol. 17, no. 3, pp. 315–333, 2016, doi: 10.1504/IJCNDS.2016.080113.

[12] A. N. Mian, A. Mamoon, R. Khan, and A. Anjum, "Effects of Virtualization on Network and Processor Performance Using Open VSwitch and Xen Server," *Proc. - 2014 IEEE/ACM 7th Int. Conf. Util. Cloud Comput. UCC 2014*, pp. 762–767, Jan. 2014, doi: 10.1109/UCC.2014.124.

[13] R. D. H. Ontoseno, M. N. Haqqi, and M. Hatta, "Limitasi Pengguna Akses Internet Berdasarkan Kuota Waktu dan Data Menggunakan PC Router Os Mikrotik," *Tek. Eng. Sains J.*, vol. 1, no. 2, p. 125, Dec. 2017, doi: 10.51804/tesj.v1i2.134.125-130.

[14] R. Zhang, M. Xie, and L. Yang, "Isoflat: Flat

Provider Network Multiplexing and Firewalling in OpenStack Cloud," *IEEE Int. Conf. Commun.*, vol. 2019-May, May 2019, doi: 10.1109/ICC.2019.8761652.

[15] S. W. Nourildean, Y. A. Mohammed, and H. A. Attallah, "Virtual Local Area Network Performance Improvement Using Ad Hoc Routing Protocols in a Wireless Network," *Comput. 2023, Vol. 12, Page 28*, vol. 12, no. 2, p. 28, Jan. 2023, doi: 10.3390/COMPUTERS12020028.

[16] R. A. Prayoga and C. Mukmin, "Analisis Peningkatan Jarak Jangkauan Signal Pada Jaringan Nirkabel SMK Muhammadiyah 2 Palembang," *Bina Darma Conf. Comput. Sci.*, vol. 3, no. 2, pp. 329–338, 2021.

[17] Muhammad M and Hasan I, "Analisa Dan Pengembangan Jaringan Wireless Berbasis Mikrotik Router Os V.5.20 Di Sekolah Dasar Negeri 24 Palu," *J. Elektron. Sist. Inf. dan Komput.*, vol. 2, no. 1, pp. 10–19, 2016.

[18] R. Christyo, "Modernisasi Laboratorium Fiber Optik untuk Meningkatkan Kompetensi Siswa Jurusan Teknik Komputer Jaringan," *Dewantara Semin. Nas. Pendidik.*, 2023.

[19] M. V. Bernal, I. Cerrato, F. Risso, and D. Verbeiren, "Transparent Optimization of Inter-Virtual Network Function Communication in Open VSwitch," *Proc. - 2016 5th IEEE Int. Conf. Cloud Networking, CloudNet 2016*, pp. 76–82, Dec. 2016, doi: 10.1109/CLOUDNET.2016.26.

[20] S. Shanmugalingam, A. Ksentini, and P. Bertin, "DPDK Open VSwitch Performance Validation with Mirroring Feature," *2016 23rd Int. Conf. Telecommun. ICT 2016*, Jun. 2016, doi: 10.1109/ICT.2016.7500387.

[21] R. Acheampong, T. C. Balan, D. M. Popovici, and A. Rekeraho, "Security Scenarios Automation and Deployment in Virtual Environment Using Ansible," *IEEE Int. Conf. Commun.*, 2022, doi: 10.1109/COMM54429.2022.9817150.

[22] M. Gupta, M. N. Chowdary, S. Bussa, and C. K. Chowdary, "Deploying Hadoop Architecture Using Ansible and Terraform," *2021 5th Int. Conf. Inf. Syst. Comput. Networks, ISCON 2021*, 2021, doi: 10.1109/ISCON52037.2021.9702299.

[23] J. O. Benson, J. J. Prevost, and P. Rad, "Survey of Automated Software Deployment for Computational and Engineering Research," *10th Annu. Int. Syst. Conf. SysCon 2016 - Proc.*, Jun. 2016, doi: 10.1109/SYSCON.2016.7490666.

[24] A. W. Manggala, Hendrawan, and A. Tanwidjaja, "Performance Analysis of White Box Switch on Software Defined Networking Using Open VSwitch," *IEEE*, Apr. 2016, doi: 10.1109/ICWT.2015.7449257.

[25] R. Yang, X. Chang, J. Mišić, and V. B. Mišić, "Performance Modeling of Linux Network System with Open VSwitch," *Peer-to-Peer Netw. Appl.*, vol. 13, no. 1, pp. 151–162, Jan. 2020, doi: 10.1007/S12083-019-00723-5/METRICS.

[26] P. Emmerich, D. Raumer, S. Gallenmüller, F. Wohlfart, and G. Carle, "Throughput and Latency of Virtual Switching with Open VSwitch: A Quantitative Analysis," *J. Netw. Syst. Manag.*, vol. 26, no. 2, pp. 314–338, Apr. 2018, doi: 10.1007/S10922-017-9417-0/METRICS.

[27] P. Masek, M. Stusek, J. Krejci, K. Zeman, J. Pokorny, and M. Kudlacek, "Unleashing Full Potential of Ansible Framework: University Labs Administration," *Conf. Open Innov. Assoc. Fruct*, vol. 2018-May, pp. 144–150, Sep. 2018, doi: 10.23919/FRUCT.2018.8468270.

[28] S. S. Valente Renato, Carlos Senna, Pedro Reto, "Federated Learning Framework to Decentralize Mobility Forecasting in Smart Cities," *IEEE*, 2022, doi: https://doi.org/10.1109/NOMS56928.2023.10154456.