

PANDUAN PRAKTIKUM

PEMROGRAMAN SISTEM JARINGAN



I PUTU HARIYADI

PANDUAN PRAKTIKUM

PEMROGRAMAN SISTEM JARINGAN

OLEH

I PUTU HARIYADI

admin@iputuhariyadi.net

WWW.IPUTUHARIYADI.NET

[Learn](#) ● [Code](#) ● [Config](#) ● [Share](#)

PRAKATA

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas berkat dan rahmatnya sehingga **electronic book (ebook)** tentang “**Panduan Praktikum Pemrograman Sistem Jaringan**” ini dapat terselesaikan. *Ebook* ini dibuat sebagai panduan praktikum terkait pemrograman sistem jaringan. Pembahasan difokuskan pada praktikum penerapan **Network Development Operations (NetDevOps)** khususnya menggunakan **Continuous Integration/Continuous Deployment (CI/CD) pipelines** untuk mengkonfigurasi dan manajemen infrastruktur jaringan berbasis **MikroTik**.

Penulis menyadari bahwa *ebook* ini masih jauh dari sempurna. Untuk itu kritik dan saran demi pengembangan *ebook* ini sangat diharapkan. Kritik dan saran tersebut dapat dikirimkan melalui email dengan alamat: admin@iputuhariyadi.net. Terimakasih.

Mataram, 15 Juni 2023

DAFTAR ISI

HALAMAN JUDUL	(i)
PRAKATA	(ii)
DAFTAR ISI	(iii)
PENDAHULUAN	(1)
BAB 1 INSTALASI DAN KONFIGURASI PNETLAB PADA	
VMWARE WORKSTATION PRO	(5)
A. Rancangan Jaringan Ujicoba	(5)
B. Instalasi dan Konfigurasi PNETLab	(6)
C. Konfigurasi Client Windows 10	(19)
D. Mengakses Web Graphical User Interface (GUI) dari PNETLab	(23)
E. Menginstalasi Windows Client Side Pack	(25)
F. Mematikan VM PNETLab	(26)
BAB 2 MENAMBAHKAN IMAGE MIKROTIK CLOUD HOSTED ROUTER (CHR)	
PADA PNETLAB	(27)
BAB 3 MANAJEMEN LAB NETDEVOPS PADA PNETLAB	(34)
A. Membuat Lab Baru di PNETLab	(34)
B. Mendesain Topology Pada Lab NetDevOps	(35)
BAB 4 KONFIGURASI DASAR LAB NETDEVOPS	(53)
A. Mengakses Console Dari Node Pada Lab	(55)
B. Konfigurasi Dasar Router R_Development	(58)
C. Konfigurasi Dasar Router R_Test	(61)
D. Konfigurasi Dasar Switch SW_Test	(63)

E. Konfigurasi Dasar Router R_Production	(66)
F. Konfigurasi Dasar Switch SW_Production	(68)
BAB 5 KONFIGURASI NAT, VLAN DAN DHCP SERVER PADA LAB NETDEVOPS	(71)
A. Konfigurasi DHCP Client, NAT, VLAN Dan DHCP Server Secara Manual Pada Router R_Test	(71)
B. Konfigurasi Bridge Port Dan Bridge VLAN Serta VLAN Filtering Secara Manual Pada Switch SW_Test	(77)
C. Konfigurasi DHCP Client Dan Verifikasi Koneksi Antar VLAN Serta Internet Pada VPCS Network TEST	(81)
D. Memverifikasi Pengalamatan IP Yang Telah Disewakan Oleh DHCP Server Pada Router R_TEST	(86)
E. Menghapus Konfigurasi Bridge Port Dan Bridge VLAN Serta VLAN Filtering Secara Manual Pada Switch SW_Test	(86)
F. Menghapus Konfigurasi DHCP Client, NAT, VLAN Dan DHCP Server Secara Manual Pada Router R_Test	(88)
BAB 6 ANSIBLE UNTUK OTOMATISASI LAB NETDEVOPS DI PNETLAB	(91)
A. Pengenalan Ansible	(91)
B. Mengimport VM GITLAB NETDEVOPS Di VMware Workstation	(96)
C. Mengatur Ansible Inventory	(99)
D. Ansible Configuration	(106)
E. Ansible AD-HOC Commands	(107)
F. Playbook	(108)
G. Ansible Playbook Untuk Mengotomatisasi Penerapan Konfigurasi NAT, VLAN, Dan DHCP Server Pada Lab NETDEVOPS	(113)
H. Memverifikasi Hasil Proses Otomatisasi Penerapan Konfigurasi NAT, VLAN Dan DHCP Server Pada Lab NETDEVOPS Di PNETLAB	(137)
I. Ansible Playbook Untuk Mengotomatisasi Penghapusan Konfigurasi NAT, VLAN, Dan DHCP Server Pada Lab NETDEVOPS	(139)

J. Memverifikasi Hasil Proses Otomatisasi Penghapusan Konfigurasi NAT, VLAN Dan DHCP Server Pada Lab NETDEVOPS Di PNETLAB	(153)
BAB 7 MEMBANGUN GITLAB CI/CD PIPELINES	(157)
A. Manajemen Gitlab Container	(157)
B. Pembuatan Gitlab Repository Untuk NETDEVOPS	(160)
C. Clone Gitlab Remote Repository	(162)
D. Push Local Repository Ke Gitlab Remote Repository	(164)
E. Membuat Gitlab CI/CD Pipeline Untuk Mengotomatisasi Penerapan Konfigurasi NAT, VLAN Dan DHCP Server Pada Lab NETDEVOPS Di PNETLAB	(167)
F. Memverifikasi Hasil Proses Otomatisasi Penerapan Konfigurasi NAT, VLAN Dan DHCP Server Pada Lab NETDEVOPS Di PNETLAB	(174)
G. Mengubah Gitlab CI/CD Pipeline Untuk Mengotomatisasi Penghapusan Konfigurasi NAT, VLAN Dan DHCP Server Pada Lab NETDEVOPS Di PNETLAB	(176)
H. Memverifikasi Hasil Proses Otomatisasi Penghapusan Konfigurasi NAT, VLAN Dan DHCP Server Pada Lab NETDEVOPS Di PNETLAB	(178)
I. Menghentikan Penggunaan Gitlab Container Dan Lab NETDEVOPS Pada PNETLAB	(181)
DAFTAR REFERENSI	(187)
TENTANG PENULIS	(188)

PENDAHULUAN

Network Development Operations (NetDevOps) adalah membawa budaya, metode teknis, strategi dan praktik terbaik dari **Development Operations (DevOps)** ke jaringan komputer. *NetDevOps* membangun dan mengelola jaringan komputer yang memungkinkan layanan jaringan dikonsumsi dengan pendekatan *DevOps*. **DevOps** merupakan terminologi yang diterapkan pada rekayasa perangkat lunak sebagai pendekatan baru yang menggabungkan bagian **Development (Dev)** dengan bagian **Operations (Ops)** ke dalam sebuah tim dan pola pikir tunggal yang bersatu atau saling berkolaborasi (Hank Preston, 2017).

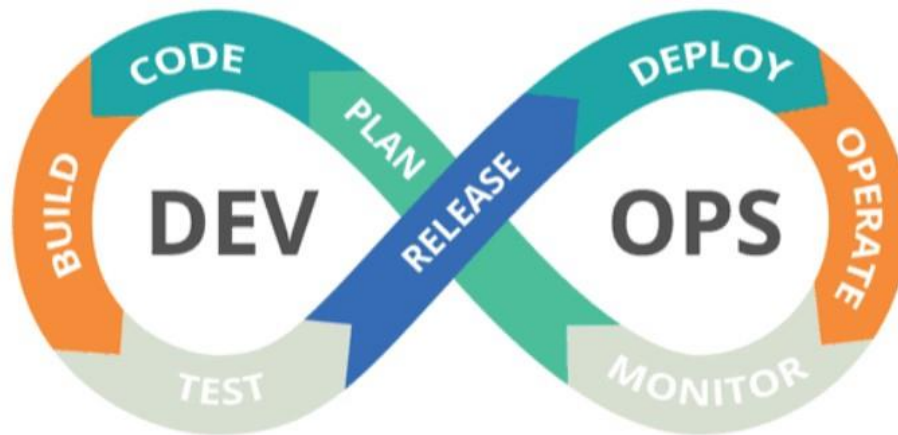
Kolaborasi dapat terjadi melalui pemanfaatan **Version Control System (VCS)**. VCS digunakan untuk mencatat perubahan pada *file* kode sumber atau dokumen lain yang menjadi bagian dari proyek pengembangan. [GitLab](#) merupakan salah satu perangkat lunak yang dapat digunakan untuk mengelola kode sumber atau **Source Code Management (SCM)** berbasis **Git repository** dengan fitur **version control** dan kolaborasi serta **Continuous Integration/Continuous Deployment (CI/CD)**. **GitLab Runner** merupakan aplikasi yang bekerja dengan **GitLab CI/CD** untuk menjalankan **jobs** pada **pipeline**.

NetDevOps memberikan berbagai manfaat seperti memecah **silos** antara pengembangan perangkat lunak dan operasional Teknologi Informasi (TI) sehingga mengakselerasi pengiriman produk. Selain itu juga mengurangi intervensi manual dengan menerapkan **Infrastructure as Code (IaC)** dan otomatisasi jaringan (**Network Automation**). Otomatisasi dapat membantu dalam mengatasi kebutuhan terkait kecepatan penerapan (*deployment*), stabilitas infrastruktur melalui pemanfaatan **Configuration Management tools** seperti **Ansible** dan proses yang konsisten (Eric Chou, 2022).

Apa itu DevOps Pipeline?

DevOps Pipeline merupakan serangkaian praktik, alat, dan proses otomatis yang digunakan oleh tim pengembangan (**Dev**) dan operasi (**Ops**) untuk membangun, menguji dan menerapkan kode perangkat lunak dengan lebih cepat. Selain itu *pipeline* dapat menjaga proses pengembangan perangkat lunak tetap terorganisir (Bojana Dobran, 2020). **DevOps** memiliki

siklus hidup (**lifecycle**) yang terdiri dari 8 (delapan) tahapan yang berulang (*iterative*) yaitu **plan**, **code**, **build**, **test**, **release**, **deploy**, **operate** dan **monitor**, seperti terlihat pada gambar berikut:



Sumber gambar: Bojana Dobran (2020)

Adapun penjelasan singkat dari setiap tahapan pada *DevOps lifecycle* tersebut adalah sebagai berikut:

1. Tahap **Plan**, tim melakukan identifikasi kebutuhan bisnis dan mengumpulkan umpan balik dari pengguna serta mendesain perencanaan proyek untuk menghasilkan produk yang diharapkan (Hiren Dhaduk, 2022).
2. Tahap **Code**, tim pengembang mulai menulis kode program untuk proyek dan menggunakan beberapa *tool* dan *plugin* seperti **Git** untuk merampingkan proses pengembangan. Selain itu juga untuk menghindari kelemahan keamanan dan praktik pengkodean yang buruk.
3. Tahap **Build**, tim pengembang melakukan *commit* kode program ke *shared repository* setelah menyelesaikan pekerjaannya dan secara otomatis mengkompilasi kode (*build*) menjadi paket yang dapat diterapkan atau yang dapat dieksekusi (Bojana Dobran, 2020).
4. Tahap **Test**, melakukan pengujian otomatis untuk memastikan proyek berfungsi seperti yang diharapkan dan menghasilkan perangkat lunak yang berkualitas. Pengujian dapat meliputi *user acceptance test*, *security test*, *integration testing*, *performance testing* dan lainnya.
5. Tahap **Release**, tim *Ops* mengkonfirmasi bahwa proyek siap dirilis dan diterapkan pada lingkungan produksi setelah berhasil melewati keseluruhan pengujian.

6. Tahap **Deploy**, perangkat lunak diterapkan ke lingkungan produksi sehingga dapat diakses atau digunakan oleh pengguna akhir.
7. Tahap **Operate**, tim operasi mengkonfigurasi dan mengelola proyek di lingkungan produksi serta akan bergantung pada otomatisasi untuk membantu memelihara proyek.
8. Tahap **Monitor**, melakukan pemantauan keseluruhan lingkungan sehingga dapat membantu tim dalam menemukan hambatan yang berdampak pada produktivitas tim *DevOps*. Pemantauan dilakukan berdasarkan pada data yang dikumpulkan dari perilaku konsumen, unjuk kerja aplikasi dan lainnya untuk memastikan semuanya berjalan lancar (Hiren Dhaduk, 2022).

Komponen DevOps Pipeline

Terdapat berbagai strategi dan praktik *DevOps* yang dapat diterapkan untuk memastikan perpindahan antar tahapan pada *pipeline* dapat berjalan dengan mulus dan efektif serta berulang secara berkelanjutan yaitu diantaranya (Bojana Dobran, 2020):

1. **Continuous Integration (CI)** merupakan metode dalam mengintegrasikan pembaruan kode dari tim pengembang ke dalam *repository* terpusat secara rutin atau sesering mungkin. Hal ini bertujuan untuk meningkatkan kualitas perangkat lunak dengan melakukan pengujian secara otomatis sehingga dapat menemukan dan mengatasi cacat (*bug*) pada perangkat lunak dengan lebih cepat. Selain itu juga pembaruan perangkat lunak dapat diverifikasi atau divalidasi dan dirilis dengan lebih cepat.
2. **Continuous Delivery (CD)** memungkinkan tim pengembangan untuk menerapkan perangkat lunak, fitur dan pembaruan kode program secara manual. Selain itu juga memastikan kode program selalu siap untuk diterapkan ke lingkungan produksi dengan melakukan pengujian kode program tersebut pada lingkungan yang menyerupai produksi (**testing environment**) guna menilai perilakunya sehingga dapat diterapkan kapan saja.
3. **Continuous Deployment (CD)** melibatkan kerangka kerja pengujian (**testing framework**) untuk memastikan kode baru bebas dari kesalahan dan siap untuk segera diterapkan ke lingkungan produksi. Pembaruan kode dirilis secara otomatis ke pengguna akhir tanpa adanya intervensi secara manual.

Kebutuhan Perangkat Keras & Lunak Ujicoba Materi

Adapun kebutuhan perangkat keras (*hardware*) dan lunak (*software*) yang diperlukan untuk dapat mengujicoba materi yang terdapat pada *ebook NetDevOps* ini adalah sebagai berikut:

A. Kebutuhan *Hardware*

Satu unit komputer dengan rekomendasi spesifikasi minimal adalah sebagai berikut:

1. CPU: 64 bit.
2. RAM: 8 GB.
3. Hard drive.
4. 1 (satu) *Network Interface Card*.

B. Kebutuhan *Software*

1. *Hypervisor Type 2 (hosted)* yaitu *VMWare Workstation Pro* yang digunakan untuk manajemen **Virtual Machine (VM)**.
2. *Bitvise SSH Client* yang dapat diunduh pada alamat <https://www.bitvise.com/>
3. *Browser Chrome* yang dapat diunduh pada alamat <https://www.google.com/chrome/>.
4. **Open Virtual Appliance (OVA) PNETLab** yang dapat diunduh pada situs *PNETLab* di alamat <https://pnetlab.com/pages/download>. **PNETLab** merupakan platform yang dapat digunakan untuk membuat, berbagi dan mempraktekan laboratorium jaringan yang mendukung beragam vendor.
5. *Mikrotik Cloud Hosted Router (CHR)* yang dapat diunduh pada situs *Mikrotik* di alamat <https://mikrotik.com/download>
6. **GitLab NetDevOps** berupa *Virtual Machine (VM)* berbasis sistem operasi **Ubuntu 22.04** yang didalamnya telah terinstalasi dan terkonfigurasi **Ansible, GitLab Community Edition (CE)** dan **GitLab Runner** yang berjalan pada **Docker Container** serta memuat **file-file playbook** yang dibahas pada *ebook* ini.

Selain itu juga diperlukan koneksi *Internet* untuk mengunduh perangkat lunak tersebut dan ujicoba materi.

Adapter VMnet1 di *Windows* 10 dan 192.168.0.14 untuk *interface pnet2* di *Guest VM PNETLab*.

Secara keseluruhan **VM PNETLab** memiliki 3 (tiga) *interface* yaitu **pnet0** dengan jenis koneksi **Network Address Translation (NAT)** untuk menjembatani kebutuhan koneksi *Internet* bagi *node* di dalam lab *NetDevOps* dan **pnet1** dengan jenis koneksi **Bridge** untuk menjembatani kebutuhan koneksi *node* di dalam lab *NetDevOps* ke jaringan riil. Selain itu *interface pnet2* dengan jenis koneksi **Host-only** untuk menjembatani koneksi antar **Guest VM PNETLab** ke **Guest VM GitLab NetDevOps** dan ke **Windows Host Operating System**.

B. Instalasi dan Konfigurasi PNETLab

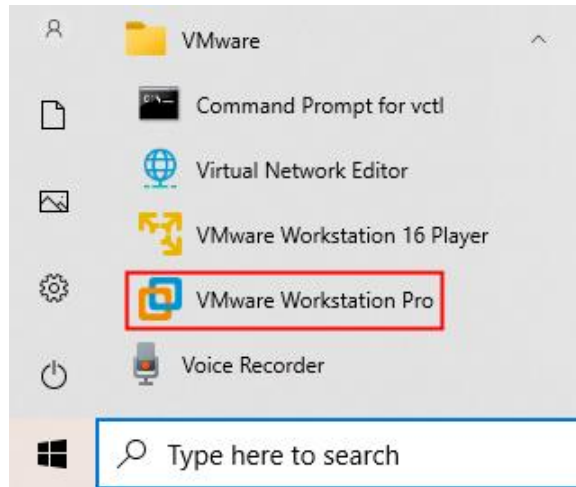
Adapun langkah-langkah untuk menginstalasi dan mengkonfigurasi PNETLab di *VMware Workstation Pro* adalah sebagai berikut:

1. Mengunduh *file .ova* dari PNETLab dengan mengakses halaman **Download** dari situs PNETLab menggunakan *browser* pada alamat <https://pnetlab.com/pages/download>. *Scroll* ke bawah pada halaman *Download* tersebut maka akan terlihat informasi terkait tautan unduh, seperti yang ditunjukkan pada gambar berikut:

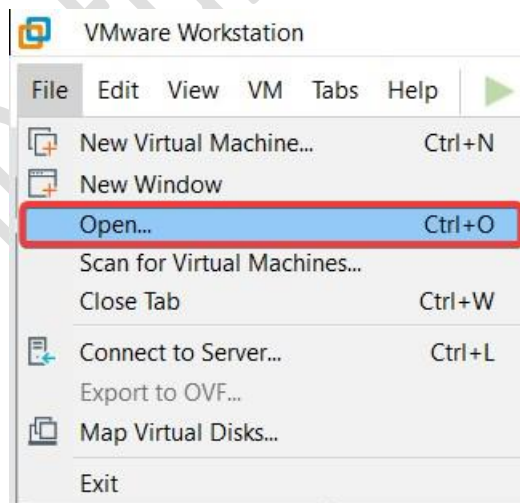
Link Download	MD5 Checksum	Size
Link Download PNET 4.2.10 Google Drive		2G
Link Download PNET 4.2.10 Google Drive (Backup 1)		2G
Link Download PNET 4.2.10 Mega (Backup 2)		2G

Terlihat terdapat 3 (tiga) tautan unduh PNETLab dimana dua diantaranya melalui [Google Drive](#) dan [Google Drive \(Backup 1\)](#) serta yang ketiga melalui [Mega](#). Ketika buku ini dibuat versi PNETLab terbaru adalah **4.2.10** dengan ukuran file **2G**. Silakan memilih salah satu dari tautan tersebut untuk mengunduh PNETLab. Tunggu hingga proses unduh selesai dilakukan.

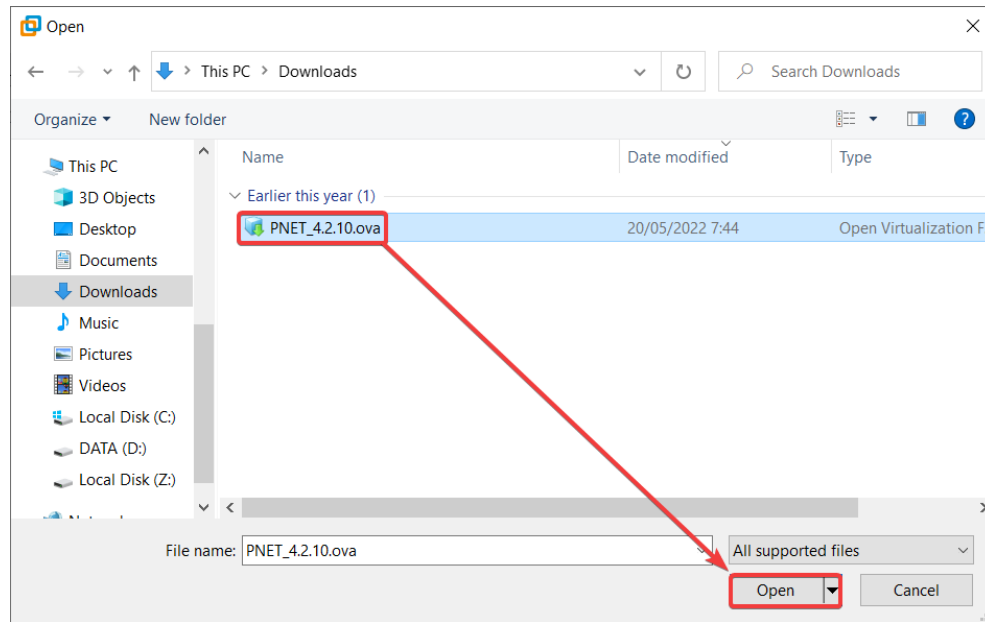
2. Menjalankan aplikasi *VMWare Workstation* dengan mengakses *shortcut VMware Workstation Pro* pada *Desktop* atau melalui *icon Start* pada *taskbar Windows* dan memilih menu **VMware > VMware Workstation Pro**, seperti yang ditunjukkan pada gambar berikut:



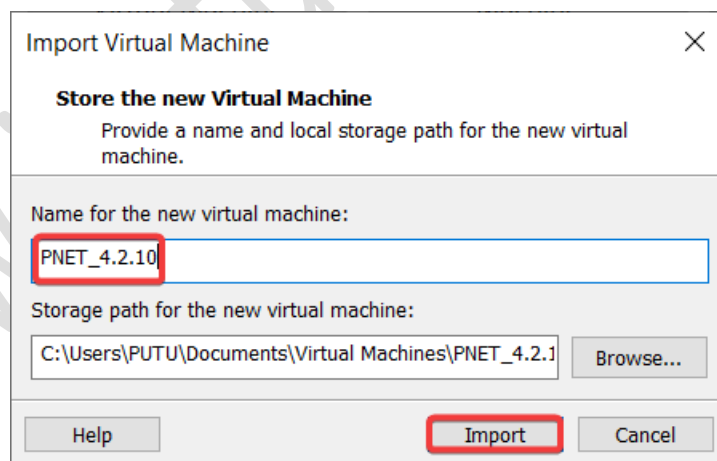
3. Pada kotak dialog **VMware Workstation** yang tampil lakukan *import file .ova* dari *PNETLab* yang telah diunduh sebelumnya dengan memilih menu **File > Open**, seperti yang ditunjukkan pada gambar berikut:



Pada kotak dialog **Open** yang tampil, arahkan ke lokasi direktori yang menyimpan file **PNET_4.2.10.ova** yang telah diunduh sebelumnya, sebagai contoh tersimpan di direktori **Downloads**. Pilih file **PNET_4.2.10.ova** dan klik tombol **Open**, seperti ditunjukkan pada gambar berikut:



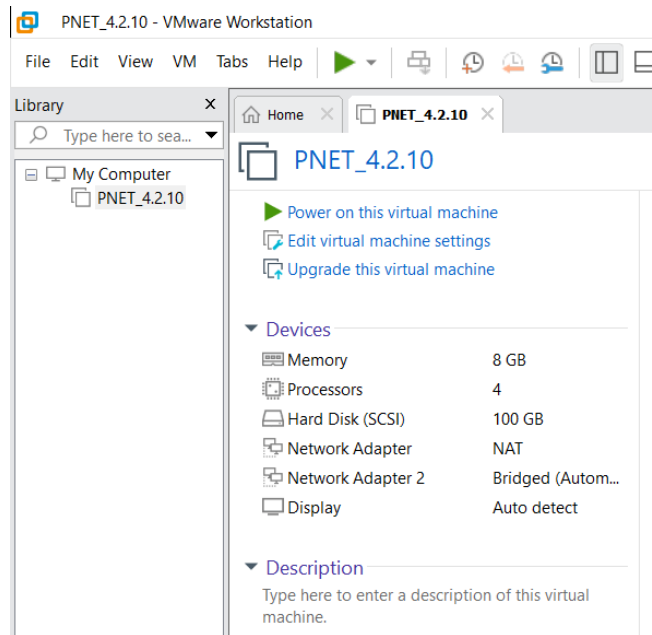
Selanjutnya tampil kotak dialog **Import Virtual Machine** untuk mengatur nama pengenal dari *virtual machine* dan lokasi penyimpanan untuk *virtual machine*. Pada inputan **Name for the new virtual machine** masukkan nama pengenal bagi *virtual machine* baru yang akan diimport, sebagai contoh masukkan **PNET_4.2.10**, seperti ditunjukkan pada gambar berikut:



Secara *default* lokasi penyimpanan (*path*) **virtual machine** adalah di *home* direktori dari *user* yang digunakan login ke sistem *Windows*. Umumnya berlokasi di drive **C:\User\NamaLoginUser\Documents\Virtual Machines\PNET_4.2.10**. Sebagai contoh untuk *user* dengan nama *login* **PUTU** maka lokasinya adalah di **C:\Users\PUTU\Documents\Virtual Machines\PNET_4.2.10**.

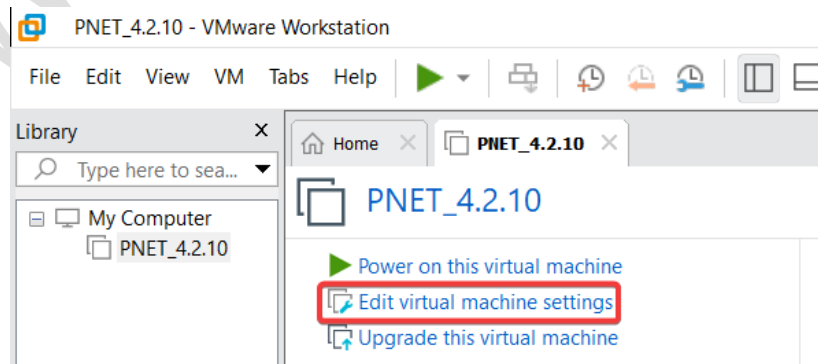
Apabila diperlukan dapat dilakukan penyesuaian pada lokasi penyimpanan dari *virtual machine* baru yang akan di import tersebut.

Tekan tombol **Import** untuk menindaklanjuti proses *import* dan tunggu hingga proses tersebut selesai dilakukan. Hasil dari proses *import*, seperti ditunjukkan pada gambar berikut:

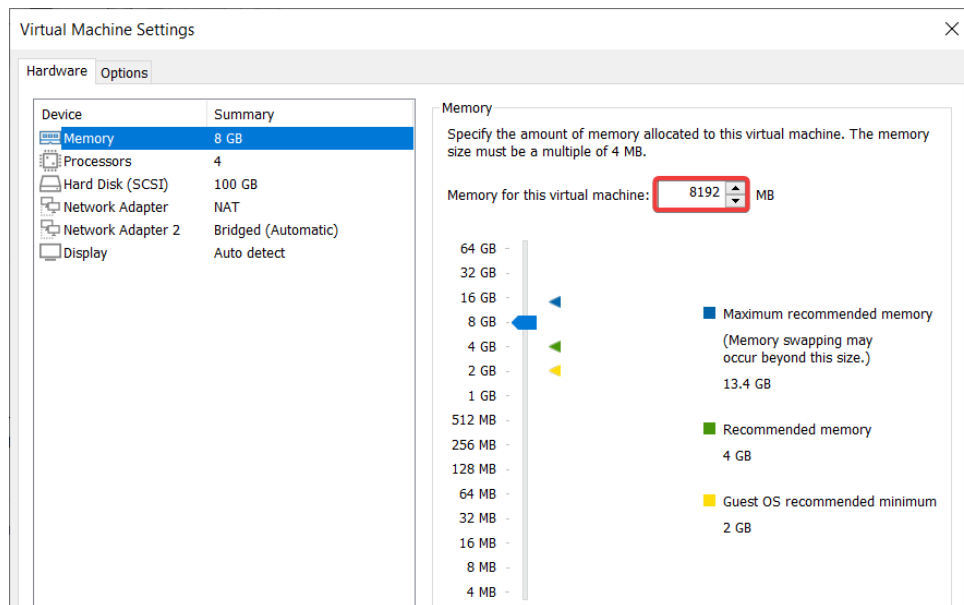


Terlihat pada daftar dari *VMware Workstation* telah terdapat *Virtual Machine (VM)* dengan nama pengenal **PNET_4.2.10**.

4. Mengubah pengaturan dari **VM PNET_4.2.10** meliputi kapasitas memori yang digunakan, jumlah prosesor, mengaktifkan *nested virtualization* dan menambahkan *network adapter* baru. Klik **Edit virtual machine settings**, seperti yang ditunjukkan pada gambar berikut:

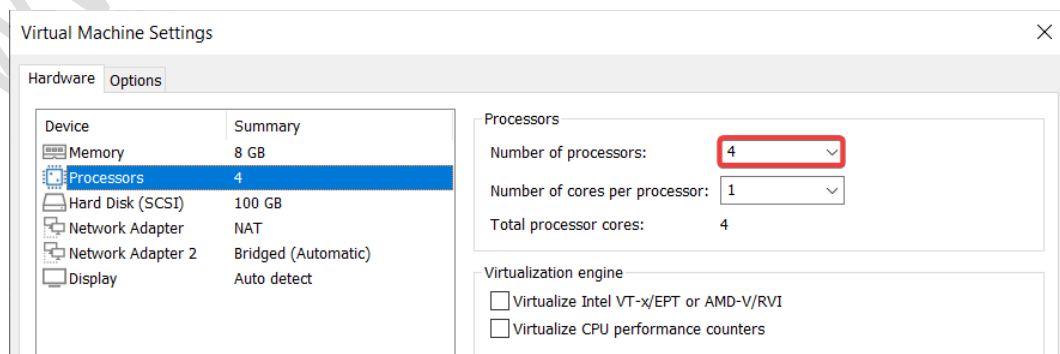


Tampil kotak dialog **Virtual Machine Settings**. Pada panel detail dari **Memory** terdapat inputan **Memory for this virtual machine** untuk menyesuaikan ukuran memori yang digunakan oleh *virtual machine* dengan pengaturan awal bernilai **8192 MB**, seperti terlihat pada gambar berikut:

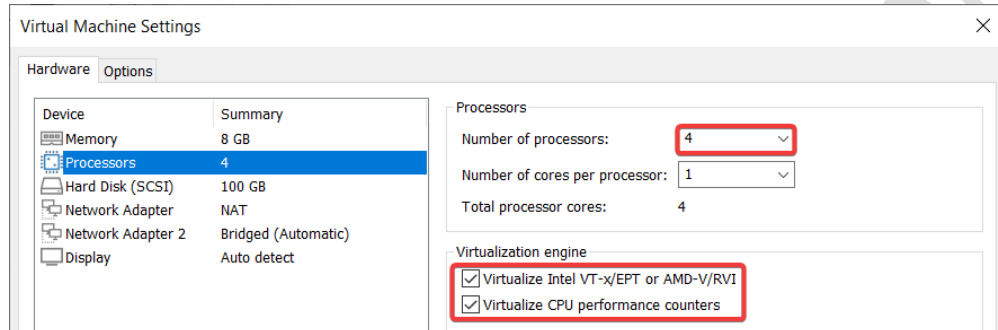


Kebetulan komputer penulis memiliki kapasitas **16 GB** sehingga nilai **Memory for this virtual machine** yang akan digunakan adalah **8192 MB** atau **8 GB**. Apabila komputer yang digunakan memiliki kapasitas memori **8 GB** maka nilai dari tersebut dapat diubah menjadi **4096 MB** atau **4 GB**.

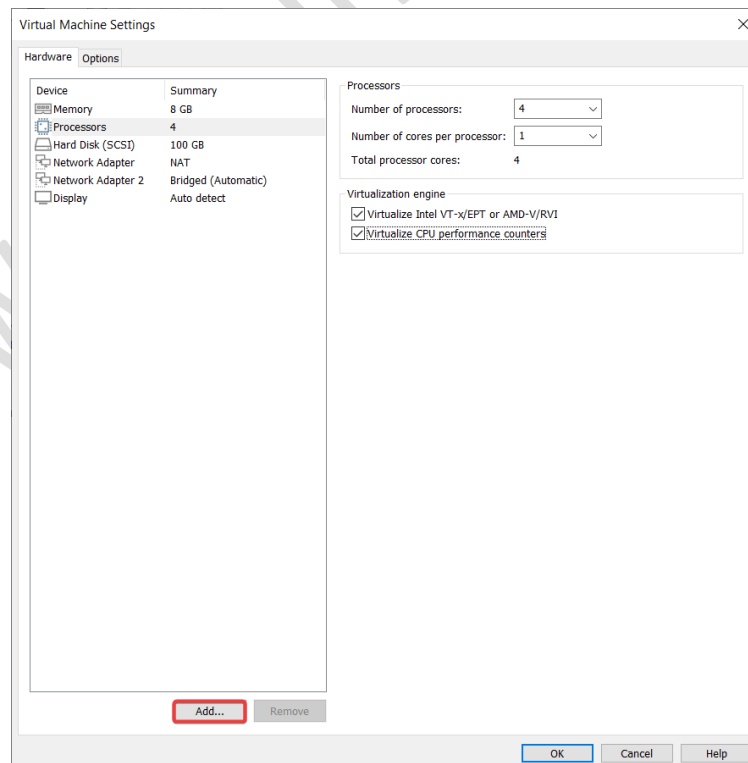
Selanjutnya pilih **Processors** pada panel sebelah kiri maka pada panel detail sebelah kanan terdapat dropdown **Number of processors** untuk menyesuaikan jumlah prosesor yang digunakan oleh *virtual machine* dimana secara default bernilai **4** dan pengaturan **Virtualization engine**, seperti terlihat pada gambar berikut:



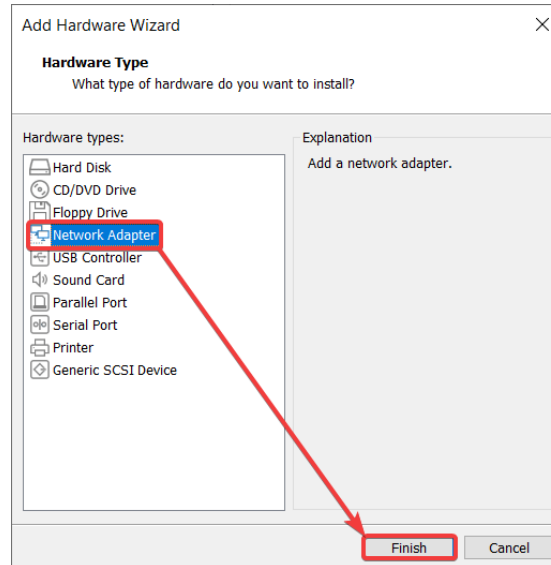
Kebetulan komputer penulis memiliki **4 core processors** sehingga nilai **Number of processors** yang akan digunakan adalah **4**. Silakan menyesuaikan dengan jumlah *core prosesor* yang dimiliki oleh komputer yang digunakan. Kemudian lakukan pengaktifan **Virtualization engine** agar mendukung *nested virtualization* dengan mencentang (**v**) pada parameter **Virtualize Intel VT-x/EPT or AMD-V/RVI** dan **Virtualize CPU performance counters**, seperti terlihat pada gambar berikut:



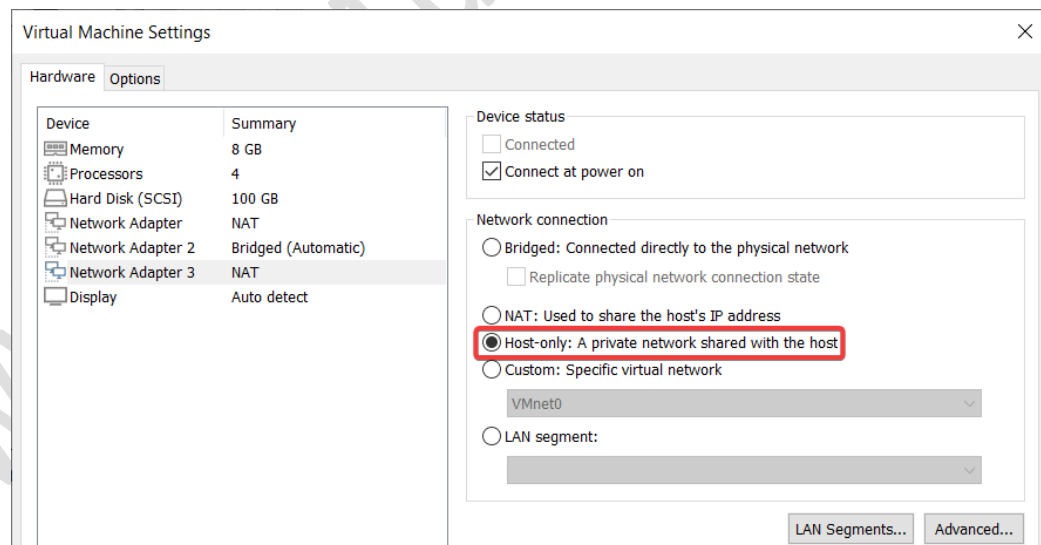
Selanjutnya pada pojok kiri bawah dari **Virtual Machine Settings**, tekan tombol **Add** untuk menambahkan **hardware** baru yaitu penambahan **Network Adapter** pada **VM PNET_4.2.10**, seperti yang ditunjukkan pada gambar berikut:



Tampil panel detail dari **Add Hardware Wizard**. Pada bagian **Hardware types**, pilih **Network Adapter** dan tekan tombol **Finish**, seperti yang ditunjukkan pada gambar berikut:

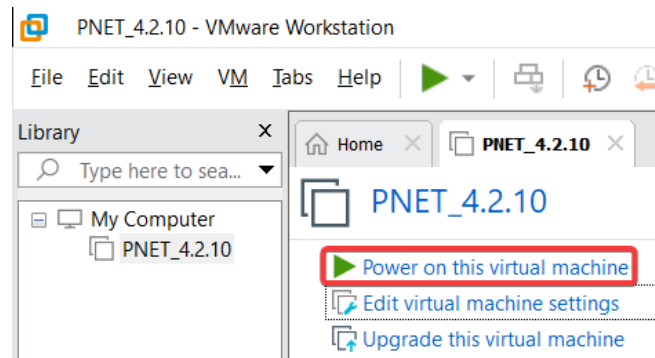


Selanjutnya pada panel detail sebelah kanan dari **Network Adapter 3** lakukan perubahan **Network connection** menjadi **Host-only: A private network shared with the host**, seperti yang ditunjukkan pada gambar berikut:

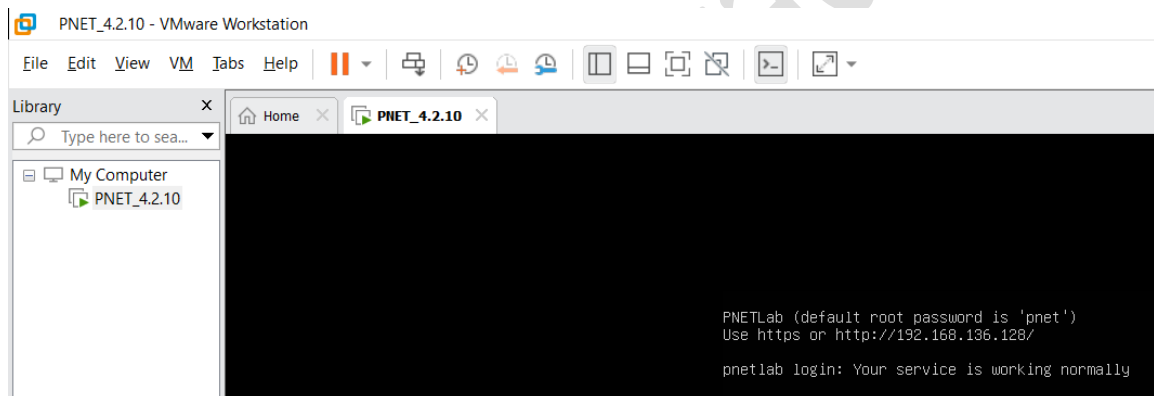


Klik tombol **OK** untuk menyimpan perubahan dan menutup kotak dialog **Virtual Machine Settings**.

5. Menjalankan **VM PNET_4.2.10** dengan memilih **Power on this virtual machine**, seperti yang ditunjukkan pada gambar berikut:



Tunggu hingga proses *booting* VM tersebut selesai dilakukan dan memperlihatkan *prompt Login* dari PNETLab, seperti yang ditunjukkan pada gambar berikut:



Untuk masuk ke dalam *console* dari VM dapat dilakukan dengan menekan tombol **CTRL+G** atau klik kiri pada **mouse** di dalam VM. Sedangkan untuk keluar dari *console* VM dapat dilakukan dengan menekan tombol **CTRL+ALT** atau geser kursor *mouse* keluar dari area *console* VM.

Tekan tombol **Enter** untuk memunculkan *prompt login* dari PNETLab sehingga tidak terganggu dengan pesan **Your service is working normally**.

Lakukan ujicoba *login* ke PNETLab menggunakan *username* "**root**" dengan *password default* adalah "**pnet**", seperti yang ditunjukkan pada gambar berikut:

```

PNETLab (default root password is 'pnet')
Use https or http://192.168.136.128/

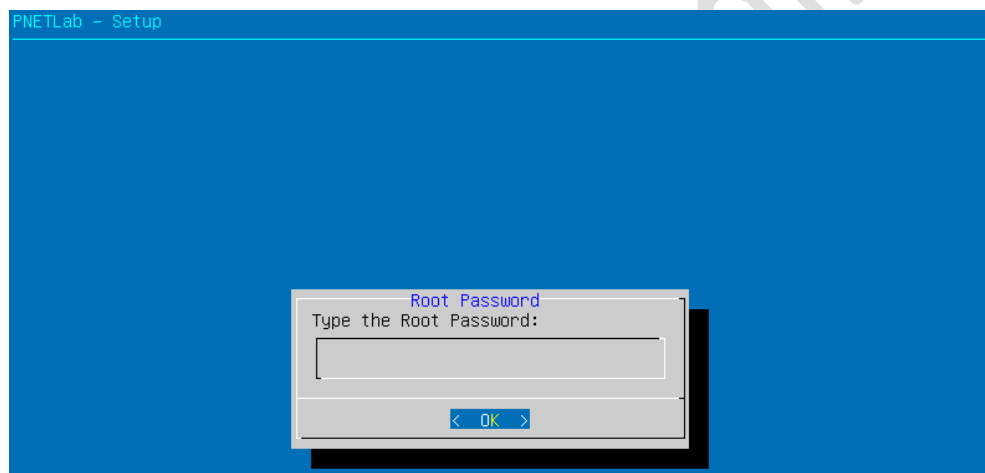
pnetlab login: Your service is working normally

PNETLab (default root password is 'pnet')
Use https or http://192.168.136.128/

pnetlab login: root
Password: _

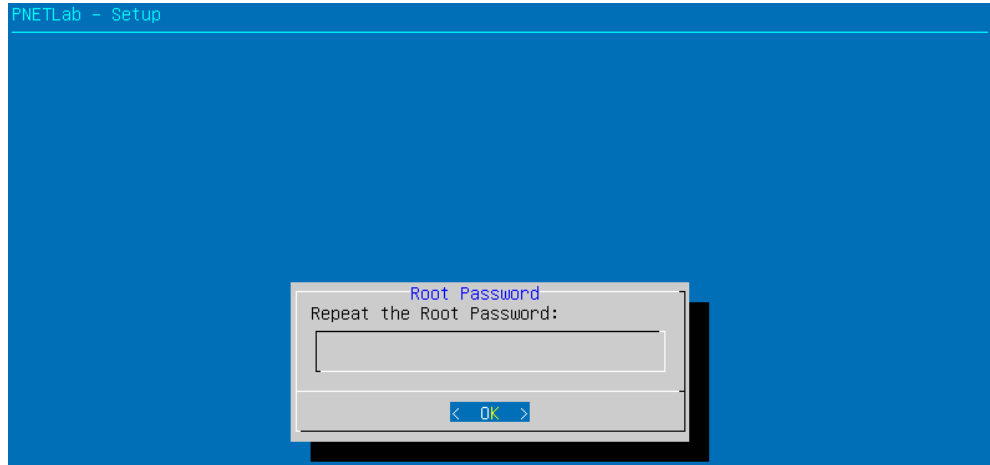
```

Selanjutnya akan tampil **PNETLab – Setup** dengan kotak dialog **Root Password** yang meminta pengguna untuk memasukkan *password* baru untuk user “root” pada parameter **Type the Root Password**, seperti yang ditunjukkan pada gambar berikut:



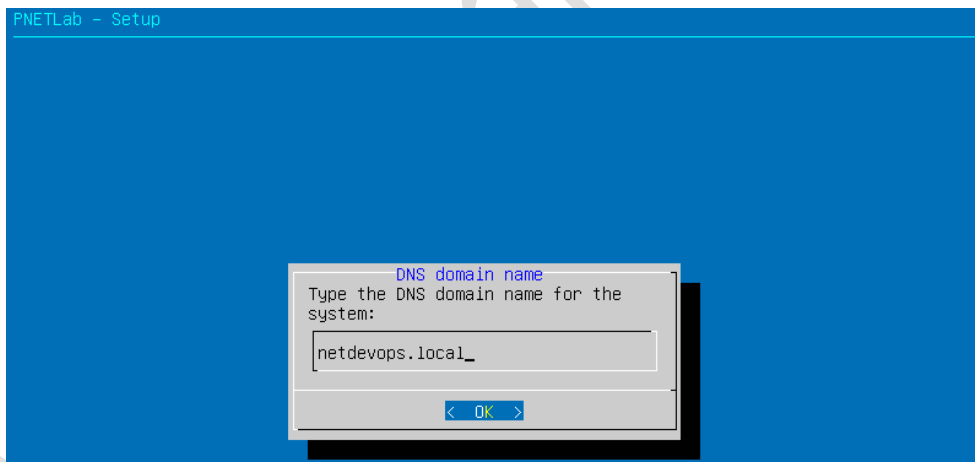
Masukkan *password* yang ingin digunakan, sebagai contoh “netdevops”. *Password* yang diketik tidak akan terlihat. Tekan tombol **Enter** untuk melanjutkan proses *setup*.

Tampil kotak dialog **Root Password** kembali dengan parameter **Repeat the Root Password** yang meminta pengguna untuk memasukkan kembali *password* baru dari user “root”, seperti ditunjukkan pada gambar berikut:



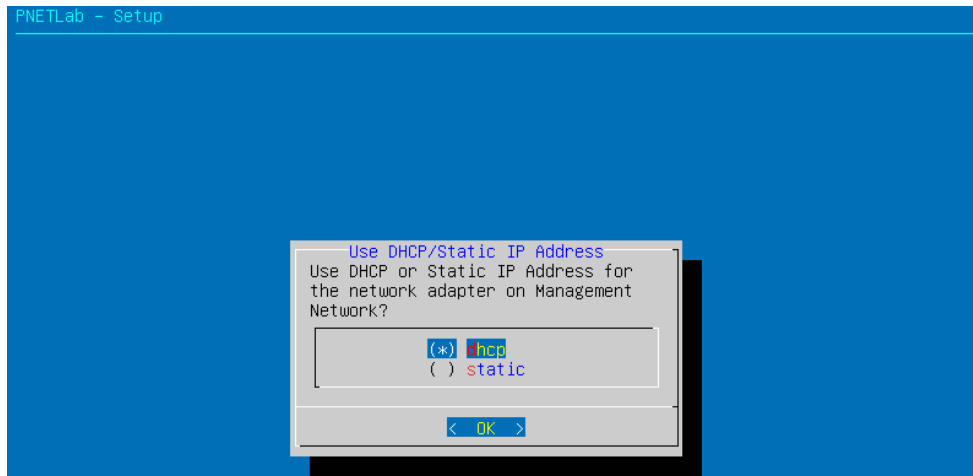
Masukkan "**netdevops**". *Password* yang diketik tidak akan terlihat. Tekan tombol **Enter** untuk melanjutkan proses *setup*.

Tampil kotak dialog **DNS domain name** untuk menentukan nama domain yang digunakan oleh sistem, seperti ditunjukkan pada gambar berikut:



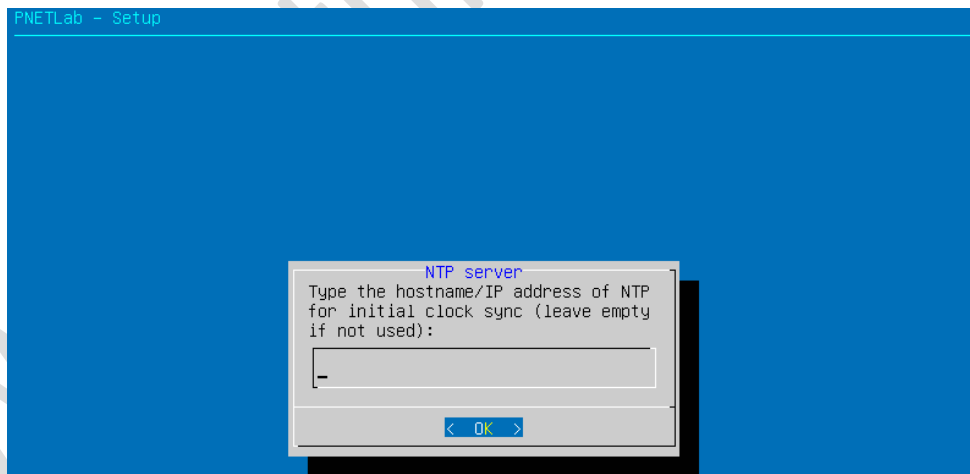
Sebagai contoh nama domain yang digunakan adalah **netdevops.local**. Tekan tombol **Enter** untuk melanjutkan proses *setup*.

Tampil kotak dialog **Use DHCP/Static IP Address** untuk menentukan metode alokasi pengalamatan IP bagi *network adapter* pada *Management Network*, seperti ditunjukkan pada gambar berikut:



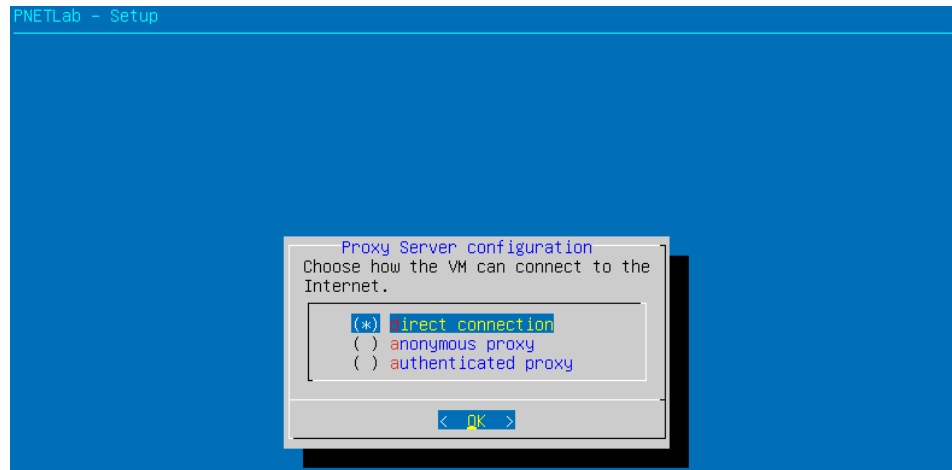
Terdapat 2 (dua) pilihan yaitu secara **dhcp** (mengalokasikan secara dinamis dengan alamat IP yang diperoleh dari server DHCP) atau **static** (mengalokasikan secara manual). Sebagai contoh dipilih **dhcp** yang juga telah terpilih *default*. Tekan tombol **Enter** untuk melanjutkan proses *setup*.

Tampil kotak dialog **NTP server** untuk memasukkan *hostname* atau alamat IP dari *Network Time Protocol (NTP)* yang dapat digunakan untuk melakukan sinkronisasi waktu, seperti ditunjukkan pada gambar berikut:



Sebagai contoh dibiarkan kosong untuk tidak menggunakan NTP. Tekan tombol **Enter** untuk melanjutkan proses *setup*.

Tampil kotak dialog **Proxy Server configuration** untuk menentukan bagaimana VM dapat terkoneksi ke *Internet*, seperti yang ditunjukkan pada gambar berikut:



Terdapat 3 (tiga) pilihan yaitu *direct connection*, *anonymous proxy* dan *authenticated proxy*. Sebagai contoh dipilih **direct connection** karena VM dapat terkoneksi langsung ke *Internet* tanpa melalui *proxy*. Tekan tombol **Enter** maka selanjutnya VM PNETLab akan melakukan *reboot*. Tunggu hingga proses *reboot* selesai dilakukan.

Apabila telah muncul *prompt Login* dari PNETLab maka lakukan ujicoba *login* kembali menggunakan *username* “**root**” dengan *password* baru yaitu “**netdevops**”, seperti ditunjukkan pada gambar berikut:

```
PNETLab (default root password is 'pnet')
Use https or http://192.168.136.128/

pnetlab login: root
Password:
Welcome to Ubuntu 18.04.5 LTS

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:        https://ubuntu.com/advantage

System information as of Fri Sep 23 06:14:53 UTC 2022

System load: 0.11           Users logged in:      0
Usage of /: 8.4% of 96.94GB IP address for pnet0: 192.168.136.128
Memory usage: 8%           IP address for pnet_nat: 10.0.137.1
Swap usage: 0%             IP address for docker0: 10.177.0.1
Processes: 236

* Canonical Livepatch is available for installation.
- Reduce system reboots and improve kernel security. Activate at:
  https://ubuntu.com/livepatch

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

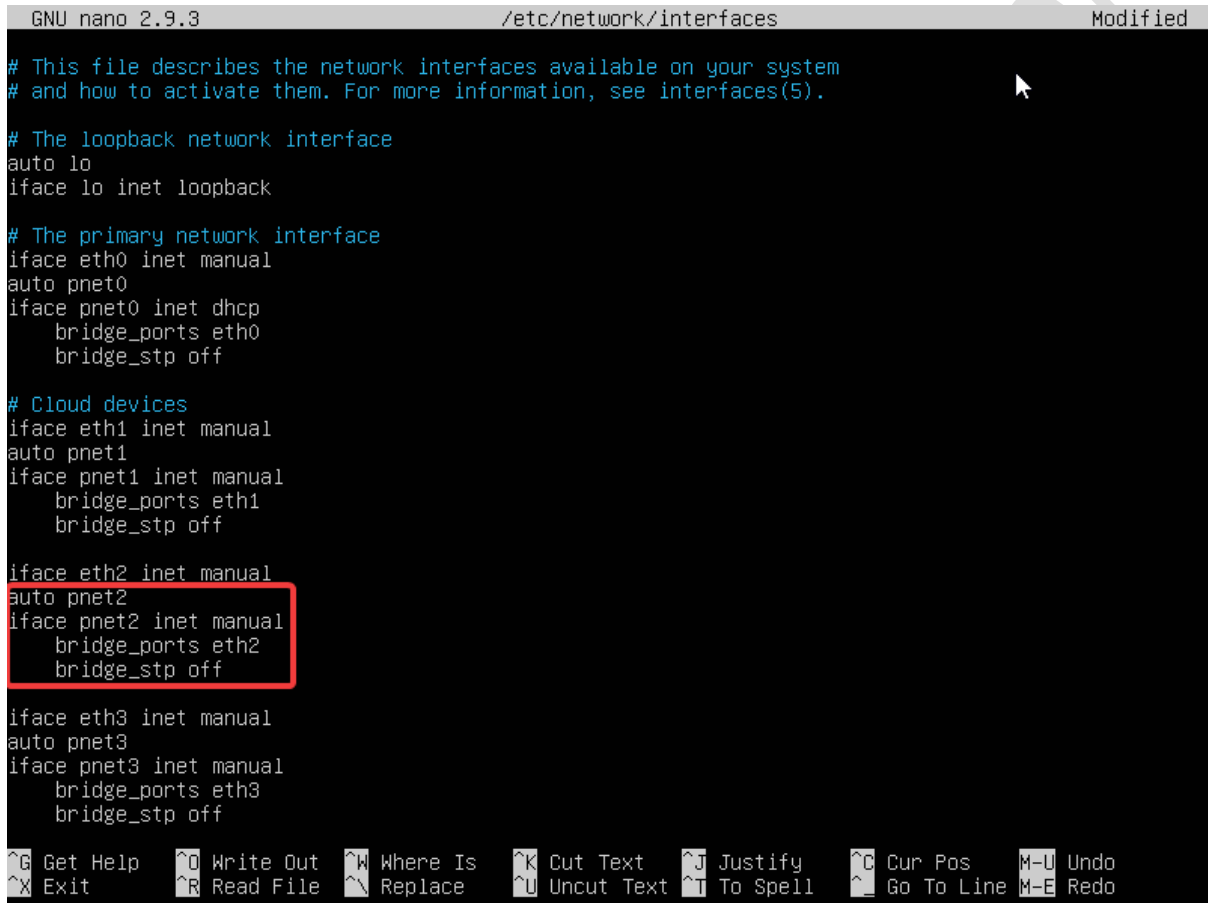
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@pnetlab:~# _
```

6. Mengatur pengalamatan IP untuk *interface pnet2* dengan nilai **192.168.0.14/29** pada *file /etc/network/interfaces* menggunakan *editor nano* dengan mengeksekusi perintah **nano /etc/network/interfaces**, seperti yang ditunjukkan pada gambar berikut:

```
root@pnetlab:~# nano /etc/network/interfaces
```

Temukan baris yang memuat pengaturan terkait *interface pnet2*, seperti ditunjukkan pada gambar berikut:



```
GNU nano 2.9.3 /etc/network/interfaces Modified
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
iface eth0 inet manual
auto pnet0
iface pnet0 inet dhcp
    bridge_ports eth0
    bridge_stp off

# Cloud devices
iface eth1 inet manual
auto pnet1
iface pnet1 inet manual
    bridge_ports eth1
    bridge_stp off

iface eth2 inet manual
auto pnet2
iface pnet2 inet manual
    bridge_ports eth2
    bridge_stp off

iface eth3 inet manual
auto pnet3
iface pnet3 inet manual
    bridge_ports eth3
    bridge_stp off

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo
^X Exit      ^R Read File  ^_ Replace   ^U Uncut Text ^T To Spell   ^_ Go To Line M-E Redo
```

Lakukan penyesuaian baris **iface pnet2 inet manual** dengan mengubah **manual** menjadi **static** untuk mengalokasikan pengalamatan IP secara statik bagi *interface pnet2*. Setelah itu lakukan penambahan 2 (dua) baris baru setelah baris yang diawali dengan **iface** tersebut untuk mengalokasikan pengalamatan IP dan *subnetmask* pada *interface* yaitu masing-masing menggunakan **address 192.168.0.14** dan **netmask 255.255.255.248**. Hasil akhir penyesuaiannya akan terlihat seperti gambar berikut:

```

iface eth2 inet manual
auto pnet2
iface pnet2 inet static
    address 192.168.0.14
    netmask 255.255.255.248
    bridge_ports eth2
    bridge_stp off

```

Simpan perubahan pada *file* tersebut dengan menekan tombol **CTRL+O** dan tekan tombol **Enter**. Untuk keluar dari *editor nano* tekan tombol **CTRL+X**.

- Melakukan *restart service networking* agar perubahan pengalamatan IP pada *interface pnet2* diterapkan dengan mengeksekusi perintah **systemctl restart networking**, seperti ditunjukkan pada gambar berikut:

```
root@pnetlab:~# systemctl restart networking
```

- Memverifikasi pengaturan pengalamatan IP pada *interface pnet2* dengan mengeksekusi perintah **ip address show dev pnet2**, seperti ditunjukkan pada gambar berikut:

```

root@pnetlab:~# ip address show dev pnet2
40: pnet2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 08:00:27:6d:b0:ed brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.14/29 brd 192.168.0.15 scope global pnet2
        valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe6d:b0ed/64 scope link
        valid_lft forever preferred_lft forever

```

Terlihat *interface pnet2* telah menggunakan alamat IP **192.168.0.14/29**.

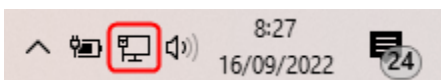
- Keluar dari akses terminal **PNETLab** dengan mengeksekusi perintah **logout**, seperti yang ditunjukkan pada gambar berikut:

```
root@pnetlab:~# logout_
```

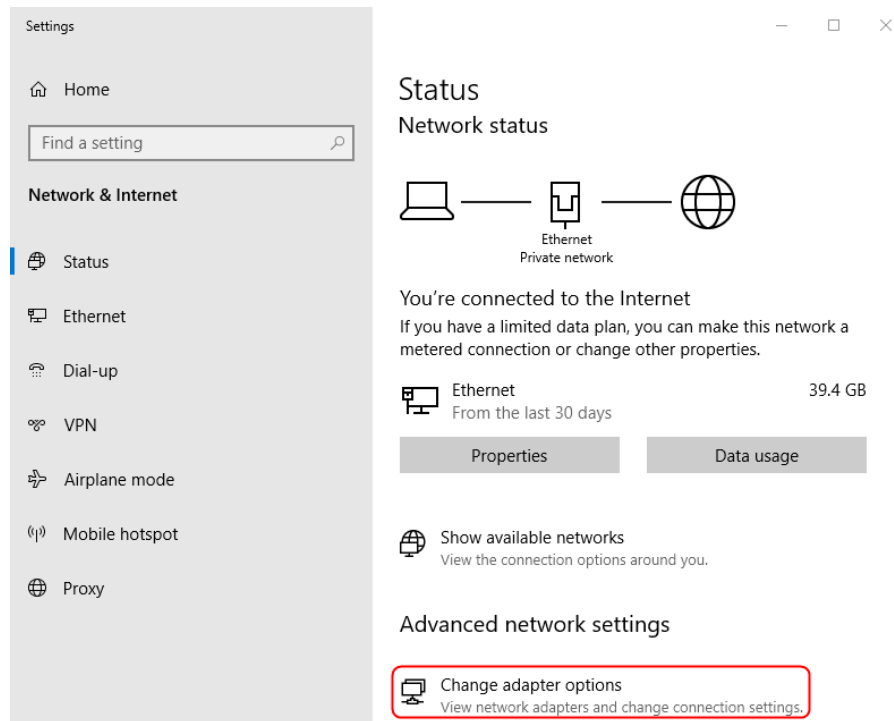
C. Konfigurasi Client Windows 10

Adapun langkah-langkah konfigurasi yang dilakukan pada *Client Windows 10* adalah sebagai berikut:

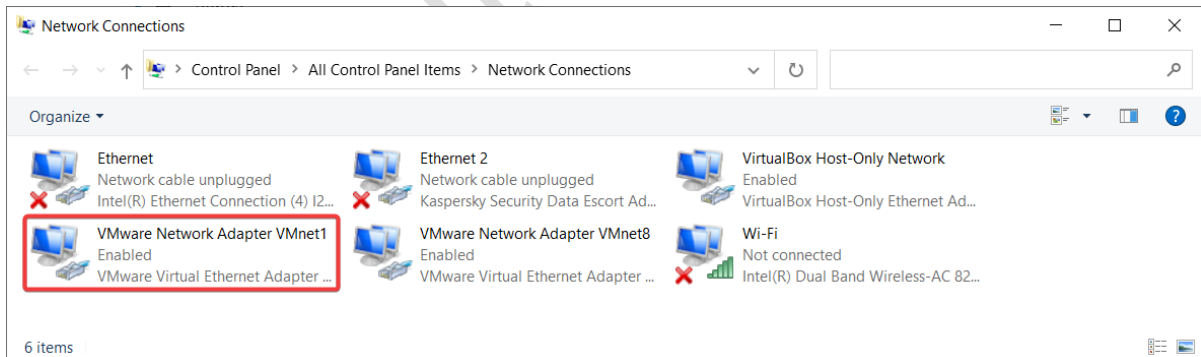
- Mengatur pengalamatan IP dan parameter TCP/IP lainnya melalui **taskbar bagian pojok kanan bawah** dengan cara **klik kanan** pada icon **Network** dan pilih **Open Network & Internet Settings**, seperti terlihat pada gambar berikut:



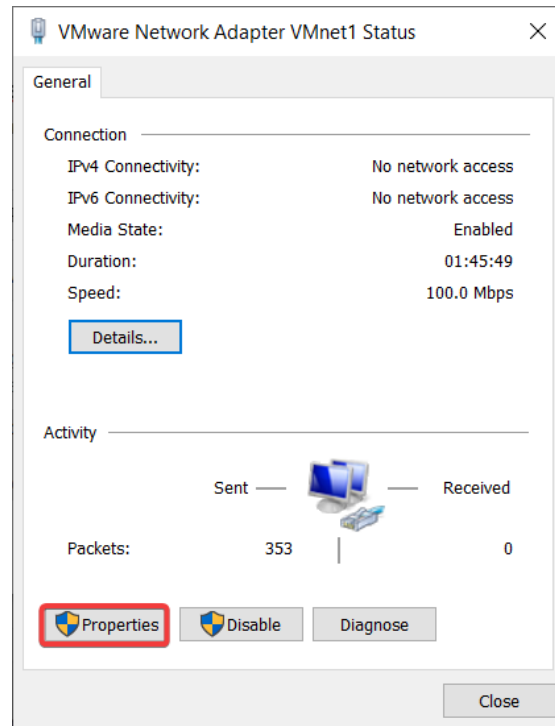
2. Tampil kotak dialog **Settings**. Pilih **Change adapter options** pada bagian bawah dari **Advanced Settings**, seperti terlihat pada gambar berikut:



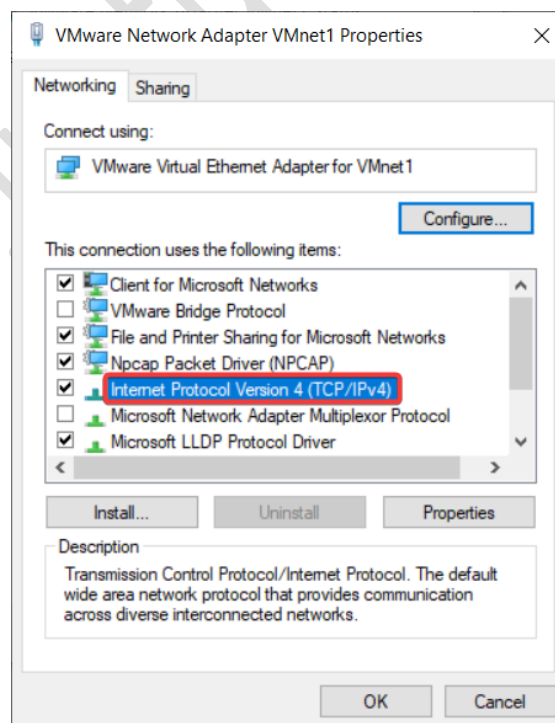
3. Tampil kotak dialog **Network Connections**. Klik dua kali pada **VMware Network Adapter VMnet1**, seperti terlihat pada gambar berikut:



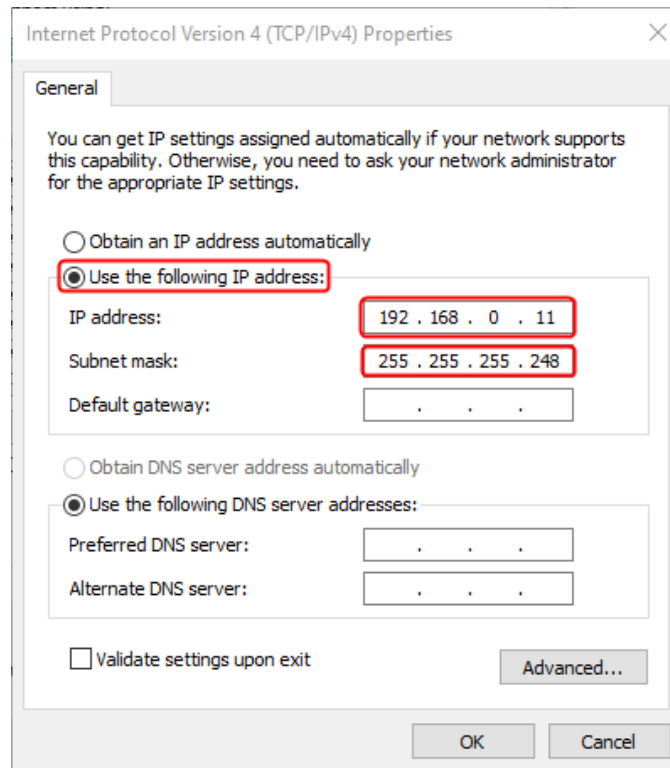
4. Tampil kotak dialog **VMware Network Adapter VMnet1 Status**. Klik tombol **Properties**, seperti terlihat pada gambar berikut:



5. Tampil kotak dialog **VMware Network Adapter VMnet1 Properties**. Pada bagian “**This connection uses the following items:**”, klik dua kali pada pilihan **Internet Protocol Version 4 (TCP/IPv4)**, seperti terlihat pada gambar berikut:

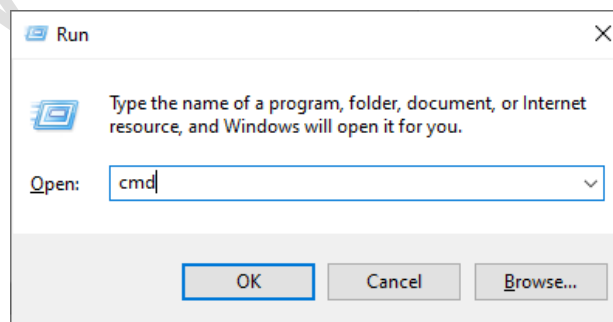


6. Tampil kotak dialog **Internet Protocol Version 4 (TCP/IPv4) Properties**. Pilih *Use the following IP Address*, seperti terlihat pada gambar berikut:



Pada isian **IP address**:, masukkan **192.168.0.11**. Sedangkan pada isian **Subnet mask**:, masukkan **255.255.255.248**. Klik tombol **OK** > **OK** > **Close**. Tutup kotak dialog **Network and Sharing Center**.

7. Buka **Command Prompt Windows** dengan menekan tombol **Windows+R**. Pada inputan form yang tampil, ketik "**cmd**" dan tekan tombol **Enter**.



8. Pada **Command Prompt** masukkan perintah untuk memverifikasi koneksi dari *client Windows 10* ke *VM PNETLab* menggunakan perintah "**ping 192.168.0.14**", seperti terlihat pada gambar berikut:

```
C:\WINDOWS\system32\cmd.exe

C:\Users\ASUS>ping 192.168.0.14

Pinging 192.168.0.14 with 32 bytes of data:
Reply from 192.168.0.14: bytes=32 time<1ms TTL=128
Reply from 192.168.0.14: bytes=32 time<1ms TTL=128
Reply from 192.168.0.14: bytes=32 time<1ms TTL=128
Reply from 192.168.0.14: bytes=32 time<1ms TTL=128

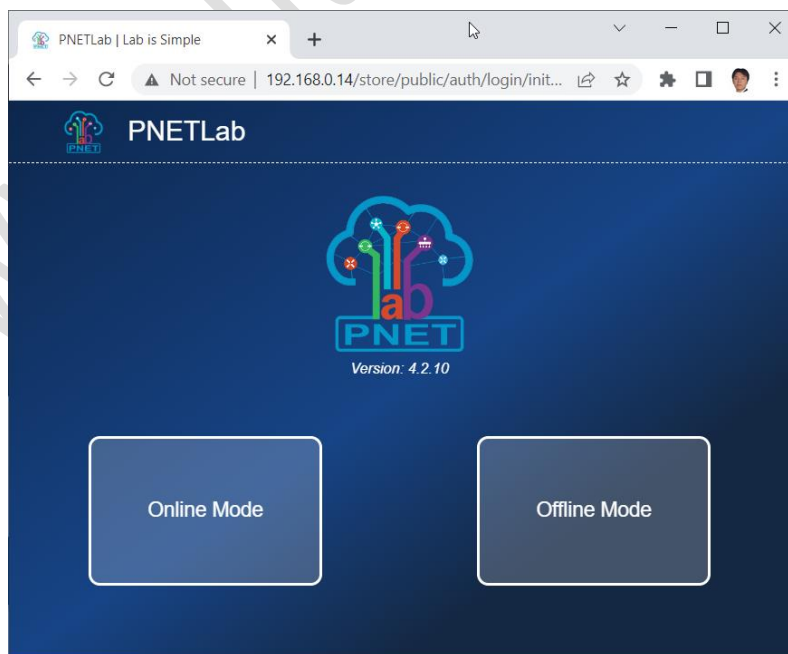
Ping statistics for 192.168.0.14:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Terlihat koneksi ke VM *PNETLab* telah berhasil dilakukan.

D. Mengakses Web Graphical User Interface (GUI) dari PNETLab

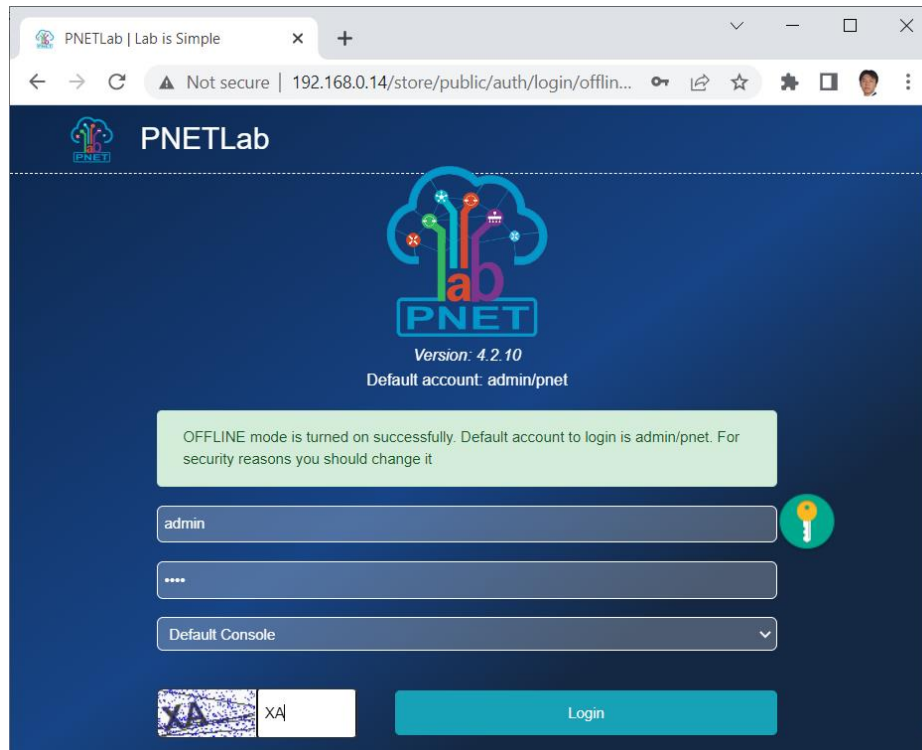
Adapun langkah-langkah untuk mengakses *Web GUI* dari **PNETLab** adalah sebagai berikut:

1. Buka *browser* di *Windows* sebagai contoh **Google Chrome** dan pada *address bar* masukkan alamat akses dari PNETLab yaitu <http://192.168.14> dan tekan tombol **Enter**. Selanjutnya akan tampil antarmuka manajemen berbasis web dari PNETLab, seperti ditunjukkan pada gambar berikut:



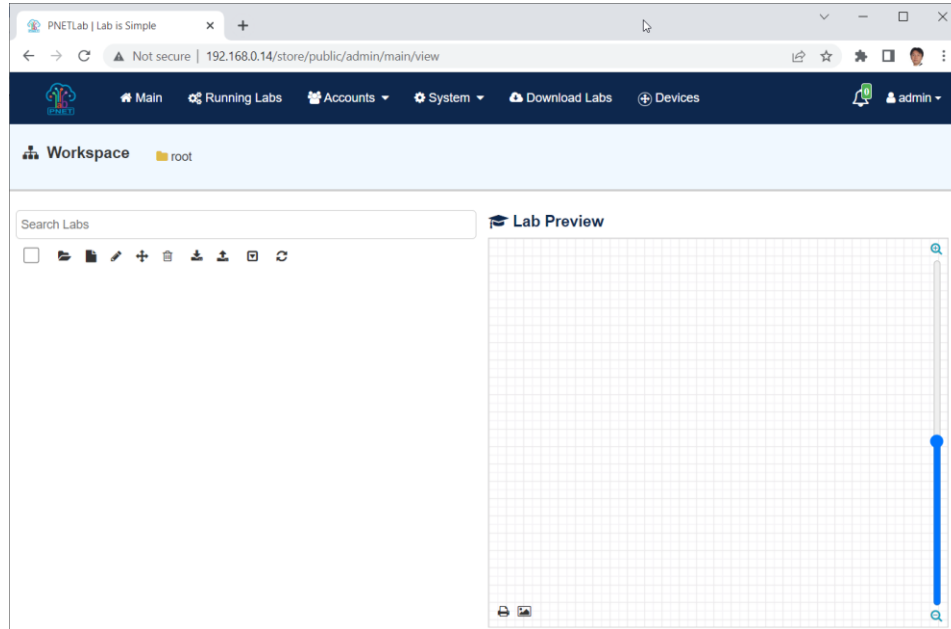
Terdapat 2 (dua) pilihan mode sistem di PNETLab yaitu **Offline Mode** dan **Online Mode**. Klik pada **Offline Mode** agar penggunaan PNETLab tidak memerlukan akses *Internet*.

2. Tampil halaman **login** dari PNETLab yang proses otentikasi sebelum dapat memanajemen PNETLab, seperti yang ditunjukkan pada gambar berikut:

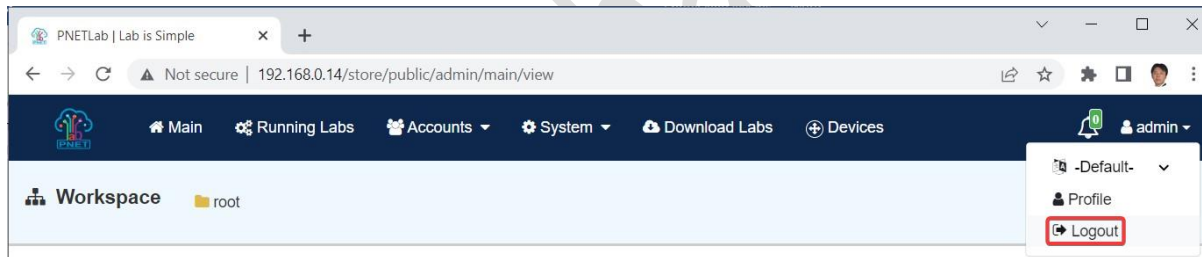


Secara *default* PNETLab menyediakan satu akun otentikasi *login* yaitu menggunakan **username “admin”** dengan **password “pnet”**. Silakan melengkapi isian pada *form login* meliputi **username** menggunakan “admin”, **password** menggunakan “pnet” dan **Captcha** sesuai dengan nilai yang tampil, sebagai contoh bernilai “XA” Klik tombol **Login**.

Apabila proses otentikasi *login* berhasil dilakukan maka pengguna akan dibawa ke halaman utama (**main**) dari manajemen PNETLab, seperti ditunjukkan pada gambar berikut:



3. Untuk keluar dari penggunaan **Web GUI PNETLab** maka klik pada *dropdown admin* di pojok kanan atas dan pilih **Logout**, seperti yang ditunjukkan pada gambar berikut:



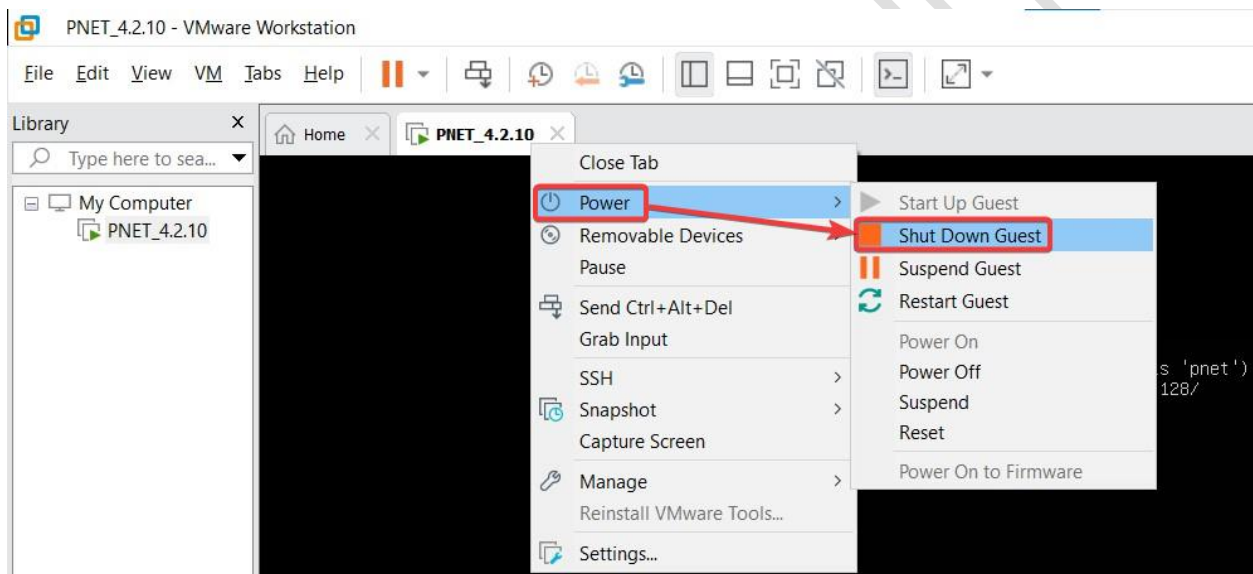
E. Menginstalasi Windows Client Side Pack

Windows client side pack memuat paket-paket aplikasi yang diperlukan untuk dapat menjalankan **telnet**, **vnc**, **wireshark** dan aplikasi **rdp** ketika bekerja dengan lab pada PNETLab. *Windows client side pack* didalamnya memuat instalasi *Wireshark 3.0.6.0*, *UltraVNC 1.2.3.1*, *Putty 0.73* yang digunakan sebagai *default telnet client*, *Plink 0.73* untuk *wireshark*, *Registry files* untuk Windows 8 dan 10 sehingga mendukung *tabbed SecureCRT*. *Windows client side pack* atau *Windows integration pack* dapat diunduh melalui [Mega](#) dan [Google Drive](#) dengan nama file "EVE-NG-Win-Client-Pack-2.0.exe" yang berukuran 61.2 MB. Tunggu hingga proses

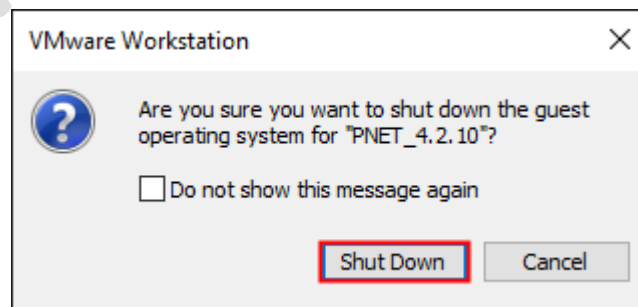
unduh selesai dilakukan. Apabila telah berhasil diunduh maka lakukan instalasi *file executable* tersebut.

F. Mematikan VM PNETLab

Apabila telah selesai digunakan maka **VM PNETLab** dapat dimatikan dengan cara klik kanan pada nama pengenal dari **VM** yaitu **PNET_4.2.10** di aplikasi **VMWare Workstation**. Pada menu yang tampil, pilih **Power > Shut Down Guest**, seperti terlihat pada gambar berikut:



Tampil kotak dialog konfirmasi **Are you sure you want to shut down the guest operating system for "PNET_4.2.10"?**. Klik tombol **Shutdown**, seperti terlihat pada gambar berikut:



Tutup kotak dialog aplikasi **VMWare Workstation**.

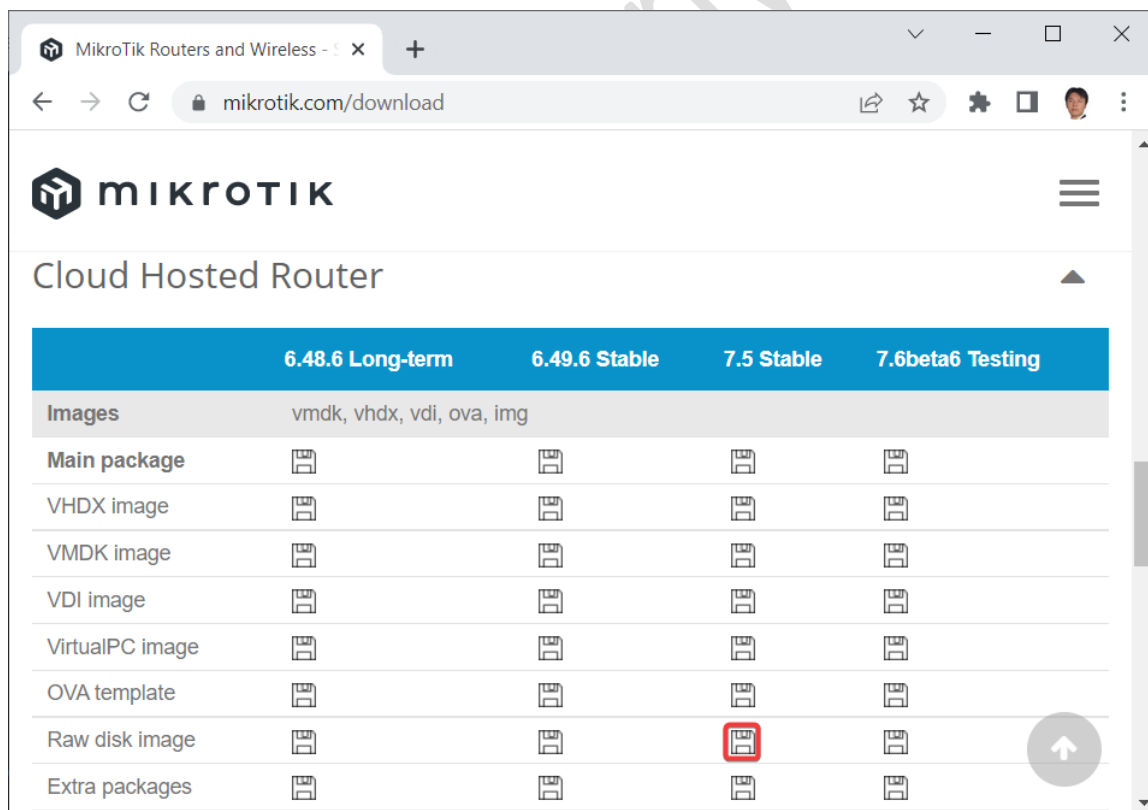
BAB 2

MENAMBAHKAN IMAGE MIKROTIK CLOUD HOSTED ROUTER

(CHR) PADA PNETLAB

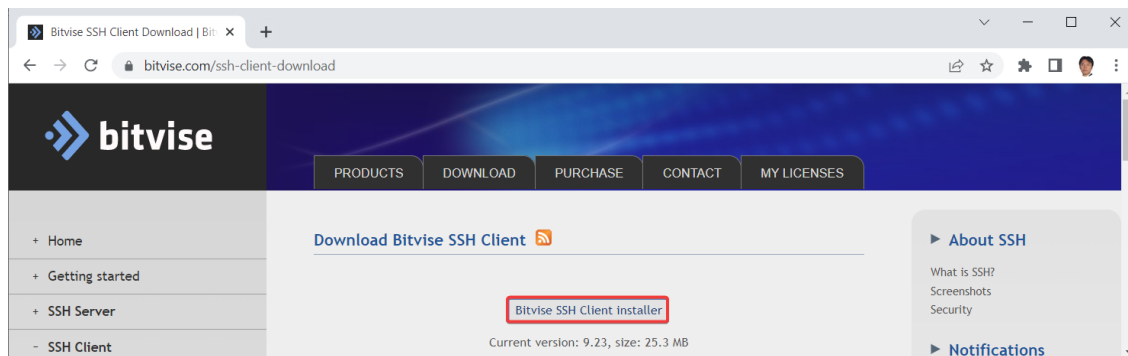
Adapun langkah-langkah untuk menambahkan **image MikroTik CHR** pada PNETLab adalah sebagai berikut:

1. Mengunduh **image MikroTik CHR** melalui situs [MikroTik](https://mikrotik.com/download) di alamat <https://mikrotik.com/download>. Tampil halaman **Download** dan lakukan *scroll* ke bawah yaitu ke bagian **Cloud Hosted Router**. Klik tautan unduh bersimbol disket pada bagian **Raw disk image** untuk versi **7.5 Stable**, seperti ditunjukkan pada gambar berikut:



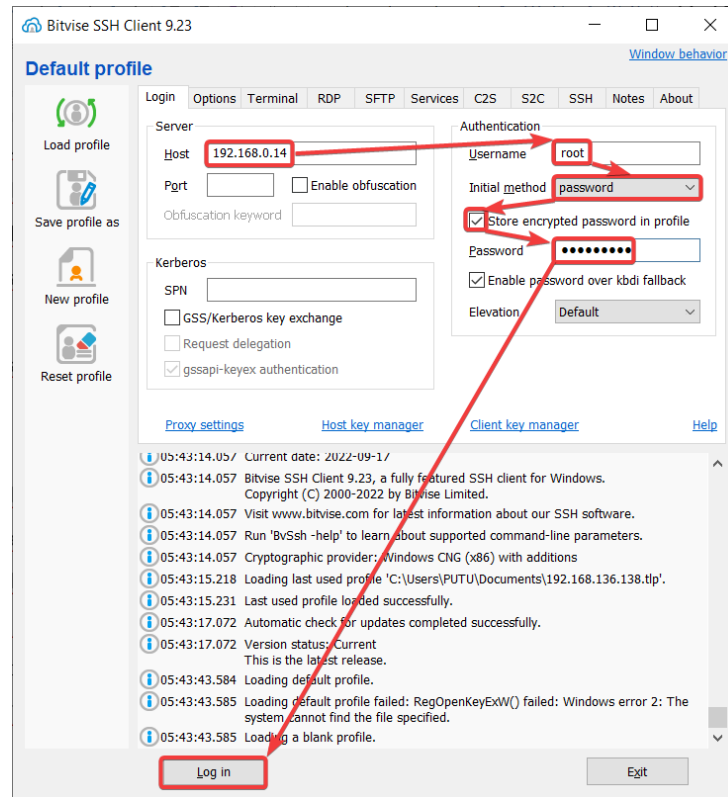
Sebagai alternatif, **raw disk image CHR** untuk versi **7.5 Stable** tersebut juga dapat diunduh dengan mengakses alamat <https://download.mikrotik.com/routeros/7.5/chr-7.5.img.zip>.

2. Mengunggah *file image chr-7.5.img.zip* yang telah diunduh sebelumnya ke PNETLab pada direktori `/opt/unetlab/addons/qemu` menggunakan aplikasi **Bitvise SSH Client**. Apabila aplikasi tersebut belum terinstalasi di komputer yang digunakan maka dapat diunduh melalui alamat <https://www.bitvise.com/ssh-client-download> dan klik tombol **Bitvise SSH client installer**, seperti terlihat pada gambar berikut:



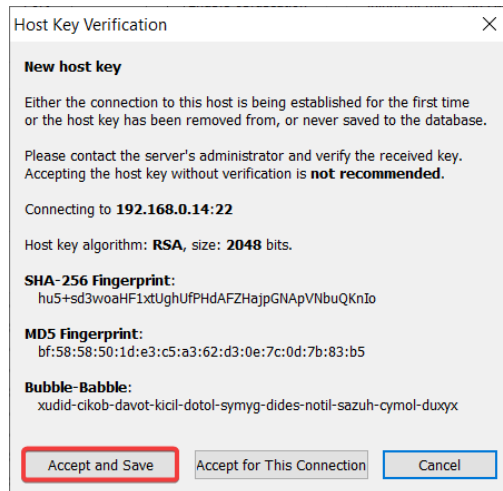
Sebagai alternatif, aplikasi tersebut juga dapat diunduh dengan mengakses alamat <https://dl.bitvise.com/BvSshClient-Inst.exe>. Tunggu hingga proses unduh selesai dilakukan. Apabila *file installer* aplikasi tersebut telah selesai diunduh maka lakukan instalasi secara mandiri.

Jalankan aplikasi tersebut dengan mengakses menu **Start Windows > Bitvise SSH Client > Bitvise SSH Client**. Tampil kotak dialog **Bitvise SSH Client 9.23** dan lengkapi pengaturan beberapa parameter, seperti yang ditunjukkan pada gambar berikut:

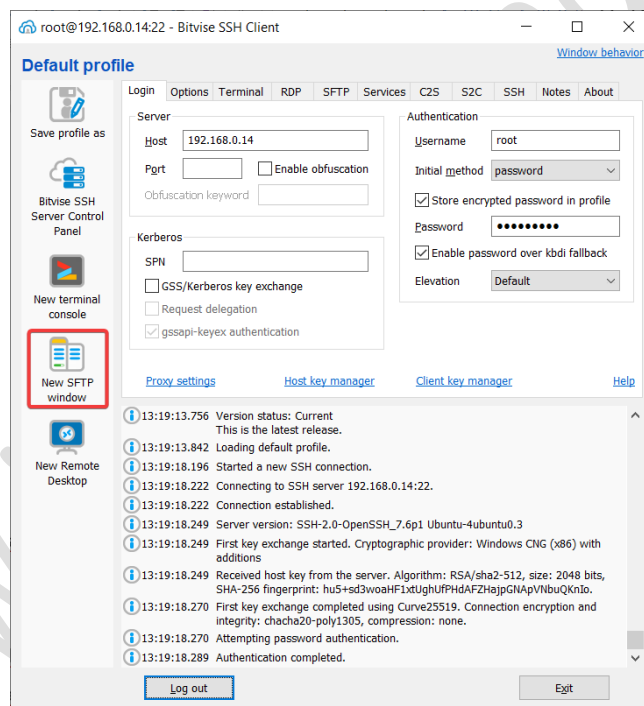


- Host**, masukkan alamat IP dari PNETLab yaitu **192.168.0.14**.
- Username**, masukkan **"root"**.
- Initial method**, pilih **password**.
- Centang (v) **Store encrypted password in profile** untuk menyimpan sandi yang terenkripsi di dalam profil.
- Password**, masukkan **"netdevops"**.

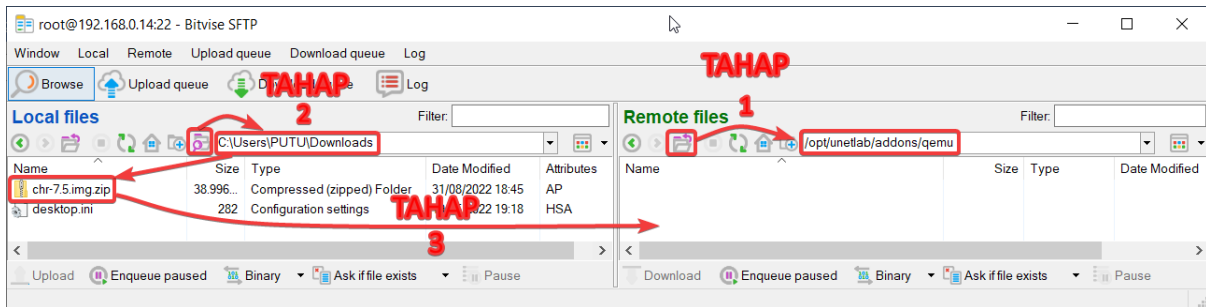
Klik tombol **Log in**. Tampil kotak dialog **Host Key Verification** dan klik tombol **Accept and Save**, seperti yang ditunjukkan pada gambar berikut:



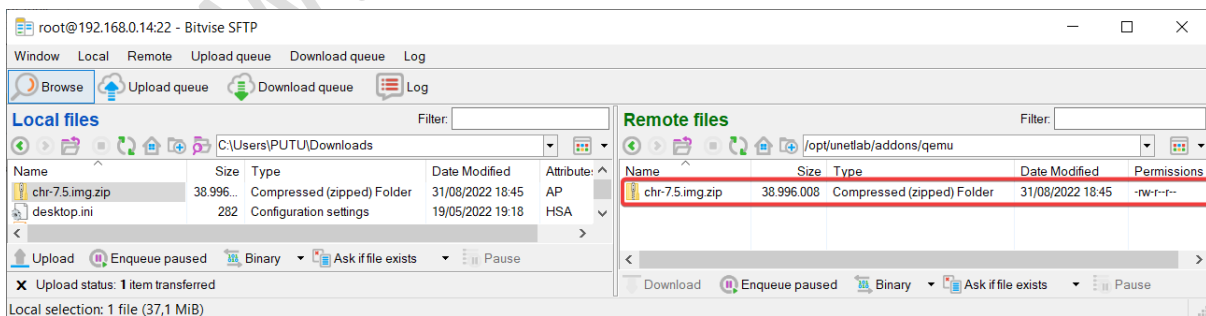
Klik **New SFTP window** pada panel menu sebelah kiri, seperti terlihat pada gambar berikut:



Tampil kotak dialog **Bitvise SFTP** dengan antarmuka terbagi menjadi 2 (dua) bagian yaitu panel sebelah kiri menampilkan informasi **Local files** di **Windows** dan panel sebelah kanan menampilkan informasi **Remote Files** di **PNETLab**. Terdapat 3 (tiga) tahap yang perlu dilakukan yaitu:

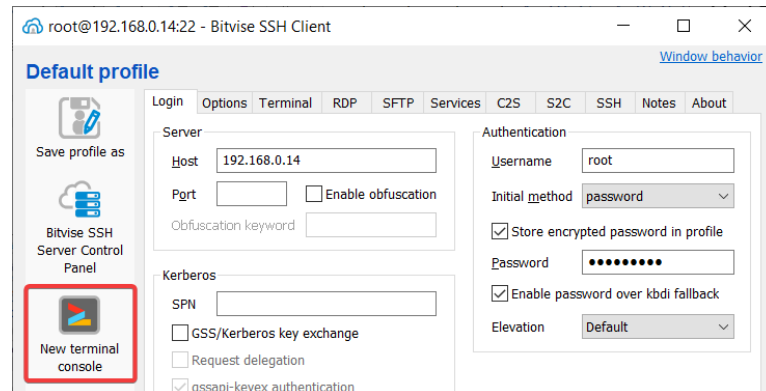


- **Tahap 1:** Mengubah direktori pada panel **Remote files** ke direktori `/opt/unetlab/addons/qemu` sebagai lokasi tujuan unggah *file image*. Pada *toolbar* dari panel **Remote files** sebelah kanan, pilih **Up** dengan simbol  yang digunakan untuk mengubah direktori yang awalnya `/root` agar berpindah ke direktori `/opt/unetlab/addons/qemu`.
- **Tahap 2:** Mengubah direktori pada panel **Local files** ke direktori yang menyimpan *file chr-7.5.img.zip* yang telah diunduh sebelumnya, sebagai contoh di `C:\Users\PUTU\Downloads`. Apabila lokasi penyimpanannya berbeda maka silakan menyesuaikan. Pada *toolbar* dari panel **Local files** sebelah kanan, pilih **Browse for folder** dengan simbol  yang digunakan untuk mengubah direktori ke `C:\Users\PUTU\Downloads` sehingga akan memperlihatkan isi didalamnya yaitu *file chr-7.5.img.zip*.
- **Tahap 3:** Lakukan **drag and drop** *file chr-7.5.img.zip* dari **Local files** ke **Remotes files** sehingga hasil akhirnya, seperti ditunjukkan pada gambar berikut:



Terlihat *file chr-7.5.img.zip* telah berhasil diunggah ke PNETLab. Tutup kotak dialog aplikasi Bitvise SFTP.

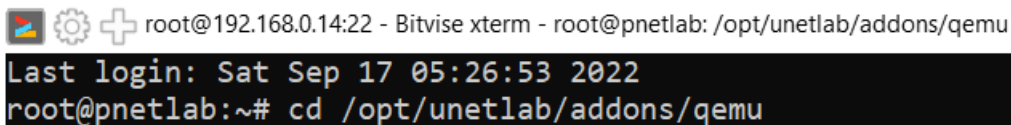
3. Pada kotak dialog **Bitvise SSH Client**, klik **New terminal console** pada panel menu sebelah kiri, seperti terlihat pada gambar berikut:



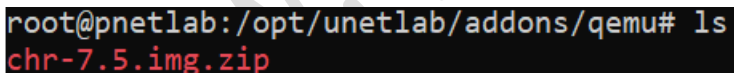
Tampil kotak dialog **Bitvise xterm**, seperti yang ditunjukkan pada gambar berikut:



4. Berpindah ke direktori **/opt/unetlab/addons/qemu** dengan mengeksekusi perintah **cd /opt/unetlab/addons/qemu** melalui *terminal*.

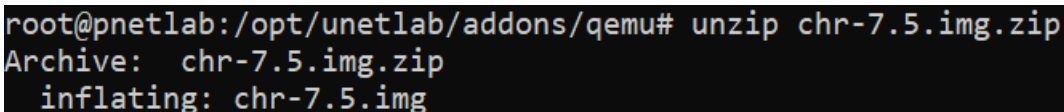


5. Menampilkan informasi isi dari direktori dimana saat ini berada dengan mengeksekusi perintah **ls**.

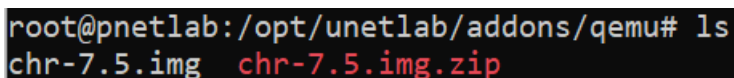


Terlihat terdapat *file* **chr-7.5.img.zip**.

6. Mengekstrak *file* **chr-7.5.img.zip** dengan mengeksekusi perintah **unzip chr-7.5.img.zip**.



7. Memverifikasi hasil ekstrak dengan mengeksekusi perintah **ls**.



Terlihat *file* dengan nama **chr-7.5.img** sebagai hasil proses ekstrak.

8. Mengubah nama *file* **chr-7.5.img** menjadi **hda.qcow2** dengan mengeksekusi perintah **mv chr-7.5.img hda.qcow2**.

```
root@pnetlab:/opt/unetlab/addons/qemu# mv chr-7.5.img hda.qcow2
```

9. Memverifikasi hasil perubahan nama *file* dengan mengeksekusi perintah **ls**.

```
root@pnetlab:/opt/unetlab/addons/qemu# ls
chr-7.5.img.zip  hda.qcow2
```

10. Membuat direktori untuk menampung *image* CHR bernama "**mikrotik-chr-7.5**" dengan mengeksekusi perintah **mkdir mikrotik-chr-7.5**.

```
root@pnetlab:/opt/unetlab/addons/qemu# mkdir mikrotik-chr-7.5
```

11. Memverifikasi hasil pembuatan direktori dengan mengeksekusi perintah **ls**.

```
root@pnetlab:/opt/unetlab/addons/qemu# ls
chr-7.5.img.zip  hda.qcow2  mikrotik-chr-7.5
```

12. Memindahkan file **hda.qcow2** ke dalam direktori **mikrotik-chr-7.5**. dengan mengeksekusi perintah **mv hda.qcow2 mikrotik-chr-7.5**.

```
root@pnetlab:/opt/unetlab/addons/qemu# mv hda.qcow2 mikrotik-chr-7.5/
```

13. Memverifikasi hasil pemindahan *file* ke direktori dengan mengeksekusi perintah **ls mikrotik-chr-7.5**.

```
root@pnetlab:/opt/unetlab/addons/qemu# ls mikrotik-chr-7.5/
hda.qcow2
```

14. Memperbaiki *ijin akses* atau *permission* dengan mengeksekusi perintah **/opt/unetlab/wrappers/unl_wrapper -a fixpermissions** pada *terminal* PNETLab.

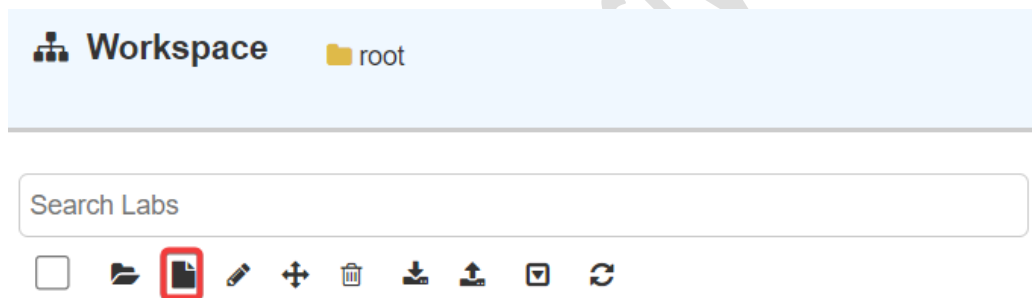
```
root@pnetlab:/opt/unetlab/addons/qemu# /opt/unetlab/wrappers/unl_wrapper -a fixpermissions
```

BAB 3

MANAJEMEN LAB NETDEVOPS PADA PNETLAB

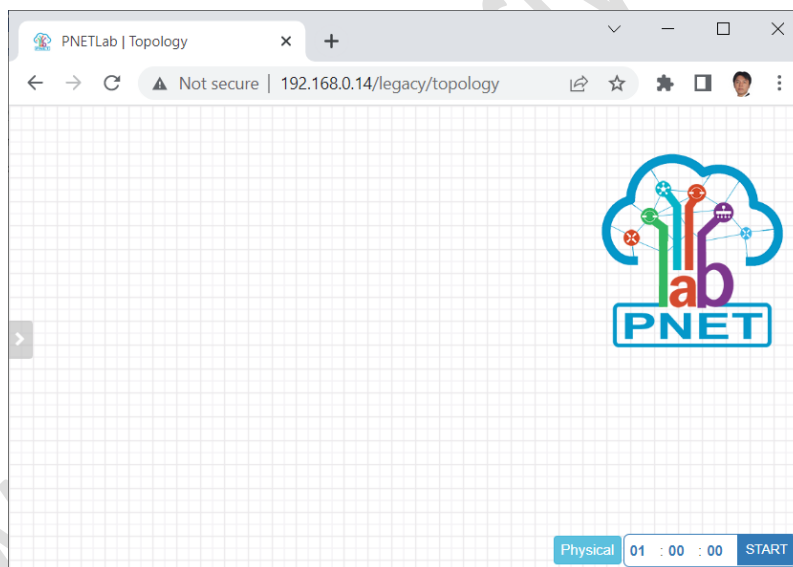
A. Membuat Lab Baru Di PNETLAB

Manajemen lab pada PNETLab dapat dilakukan dengan terlebih dahulu mengakses Web GUI melalui *browser* pada alamat <http://192.168.0.14>. Pada halaman *Login* yang tampil, gunakan *username* “**admin**” dengan *password* “**pnet**” untuk melewati proses otentikasi. Apabila proses otentikasi berhasil dilakukan maka akan tampil halaman **Main**. Untuk membuat lab di PNETLab dapat dilakukan melalui halaman **Main** dan menekan tombol **Add New Lab** pada **management buttons**, seperti yang ditunjukkan pada gambar berikut:



Tampil kotak dialog **Add new Lab**. Minimal parameter yang wajib diatur adalah **Name** yang merupakan nama pengenal dari lab yang dibuat, sebagai contoh **NetDevOps**. Parameter lain yang juga dapat diatur namun tidak bersifat wajib yaitu **Author** yang merupakan nama pembuat lab, sebagai contoh dimasukkan nama diri sendiri, seperti yang ditunjukkan pada gambar berikut:

Tekan tombol **Add** untuk memproses pembuatan lab baru. Selanjutnya akan tampil halaman **Topology**, seperti yang ditunjukkan pada gambar berikut:

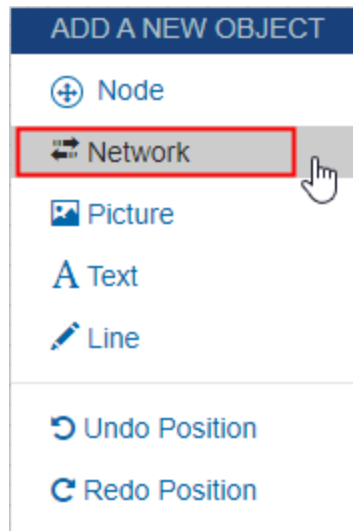


B. Mendesain Topology Pada Lab NetDevOps

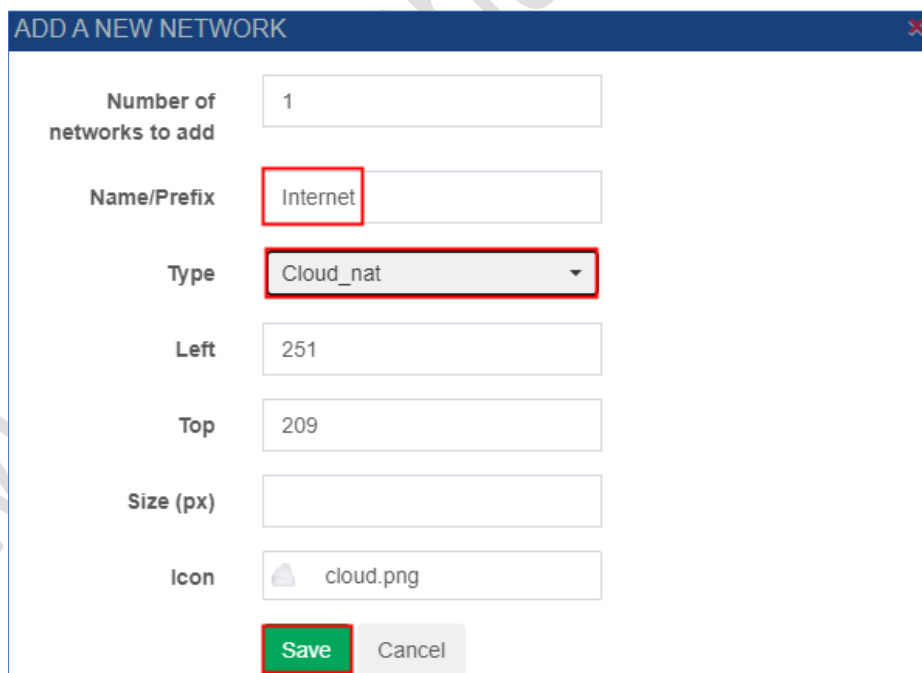
Adapun langkah-langkah dalam mendesain topologi pada lab NetDevOps adalah sebagai berikut:

1. Menambahkan **Network** dengan jenis **Cloud_nat** agar *node-node* pada lab *NetDevOps* dapat terkoneksi ke *Internet* yaitu dengan cara klik kanan pada topologi maka akan tampil kotak

dialog **ADD A NEW OBJECT** dan klik pada pilihan **Network**, seperti terlihat pada gambar berikut:



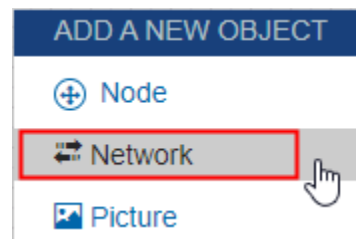
Tampil kotak dialog **ADD A NEW NETWORK**. Lakukan penyesuaian pada nilai dari parameter **Name/Prefix** menjadi **Internet** dan pilihan **Type** menjadi **Cloud_nat**, seperti terlihat pada gambar berikut:



Klik tombol **Save** untuk menyimpan perubahan. Jika diperlukan, lakukan penyesuaian atau penataan ulang lokasi penempatan *object Network Internet* tersebut pada topologi sehingga hasilnya akan terlihat seperti pada gambar berikut:

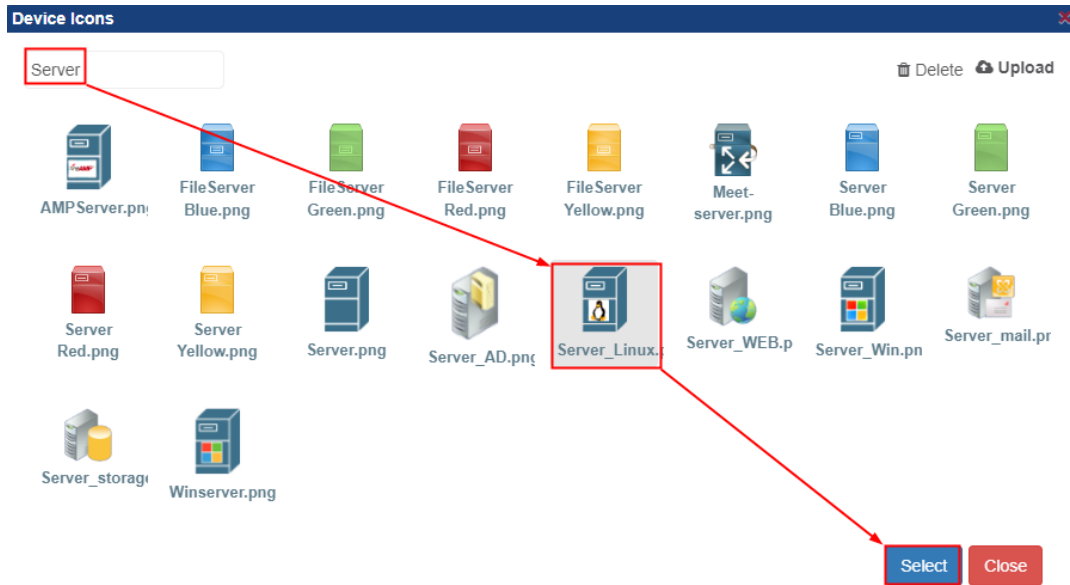


2. Menambahkan **Network** dengan jenis **Cloud2** agar **VM GitLab NetDevOps** yang berjalan pada **VMware Workstation Pro** dapat berkomunikasi dengan *node-node* pada lab *NetDevOps* yaitu dengan cara klik kanan pada topologi maka akan tampil kotak dialog **ADD A NEW OBJECT** dan klik pada pilihan **Network**, seperti terlihat pada gambar berikut:

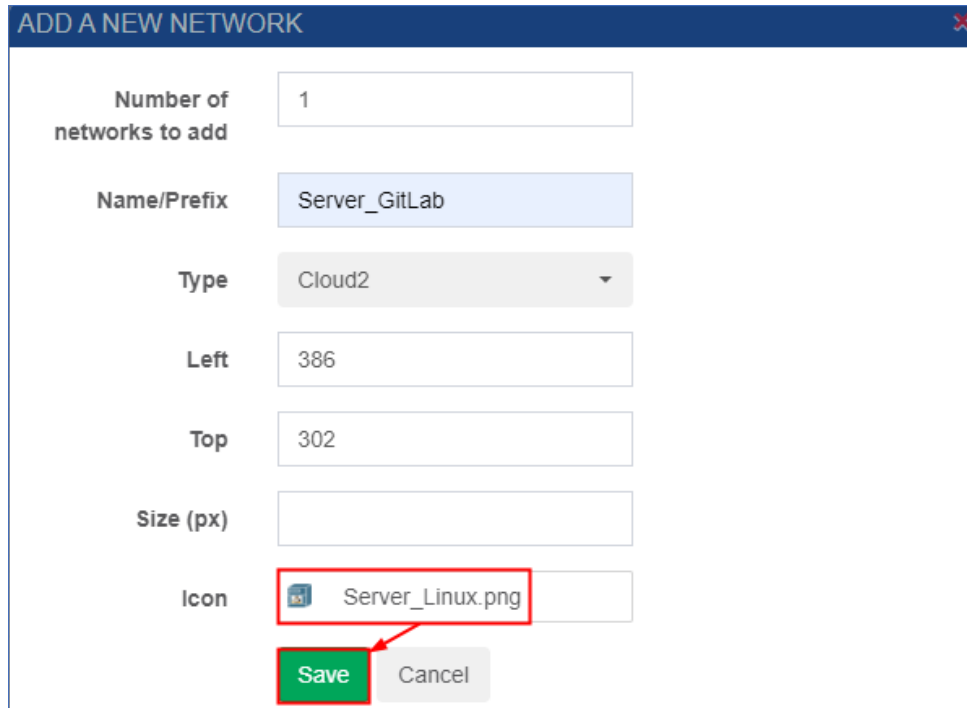


Tampil kotak dialog **ADD A NEW NETWORK**. Lakukan penyesuaian pada nilai dari parameter **Name/Prefix** menjadi **Server_GitLab** dan pilihan **Type** menjadi **Cloud2**, seperti terlihat pada gambar berikut:

Klik **cloud.png** pada parameter **Icon** untuk mengubah *icon* dari *object Network* yang ditambahkan maka akan tampil kotak dialog **Device Icons**. Pada inputan pencarian di bagian pojok kiri atas, masukkan kata kunci pencarian yaitu **Server** maka akan terlihat pilihan icon untuk **Server**. Pilih pada **Server_Linux.png** serta klik tombol **Select**, seperti terlihat pada gambar berikut:



Kembali tampil kotak dialog **ADD A NEW NETWORK**. Terlihat nilai dari parameter **Icon** telah berubah menjadi **Server_Linux.png** dan klik pada tombol **Save** untuk menyimpan perubahan, seperti yang ditunjukkan pada gambar berikut:



ADD A NEW NETWORK

Number of networks to add: 1

Name/Prefix: Server_GitLab

Type: Cloud2

Left: 386

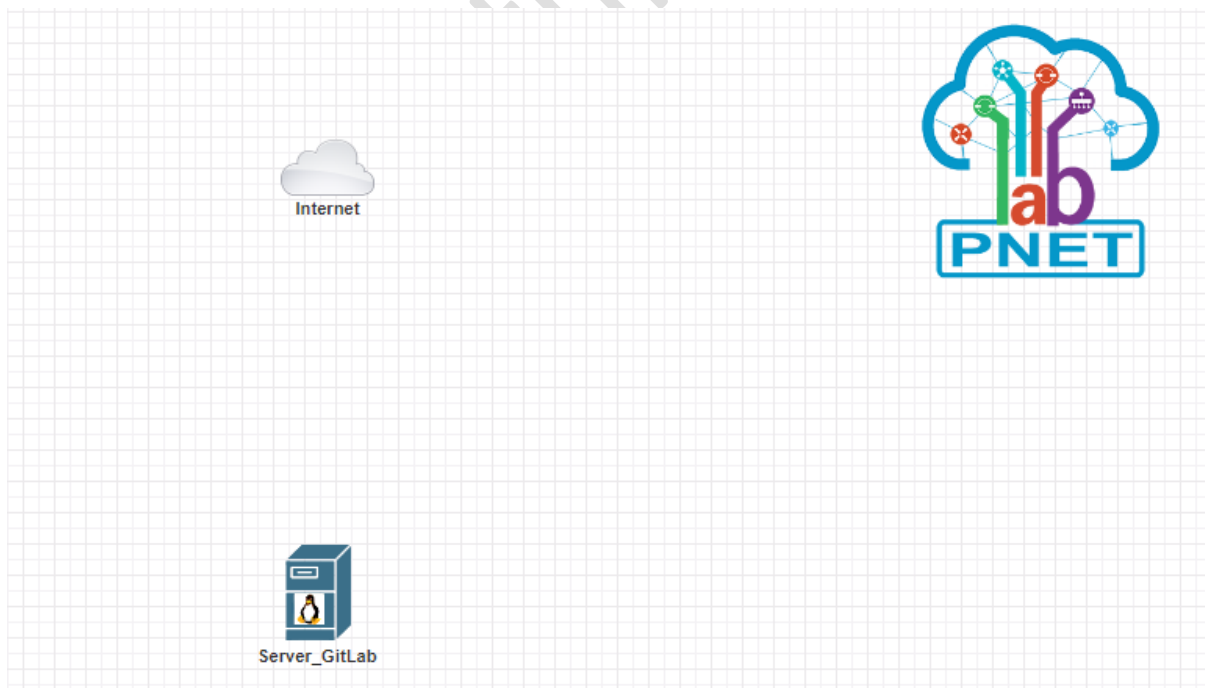
Top: 302

Size (px):

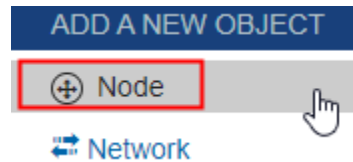
Icon: Server_Linux.png

Save Cancel

Jika diperlukan, lakukan penyesuaian atau penataan ulang lokasi penempatan *object Network Server_GitLab* tersebut pada topologi sehingga hasilnya akan terlihat seperti pada gambar berikut:

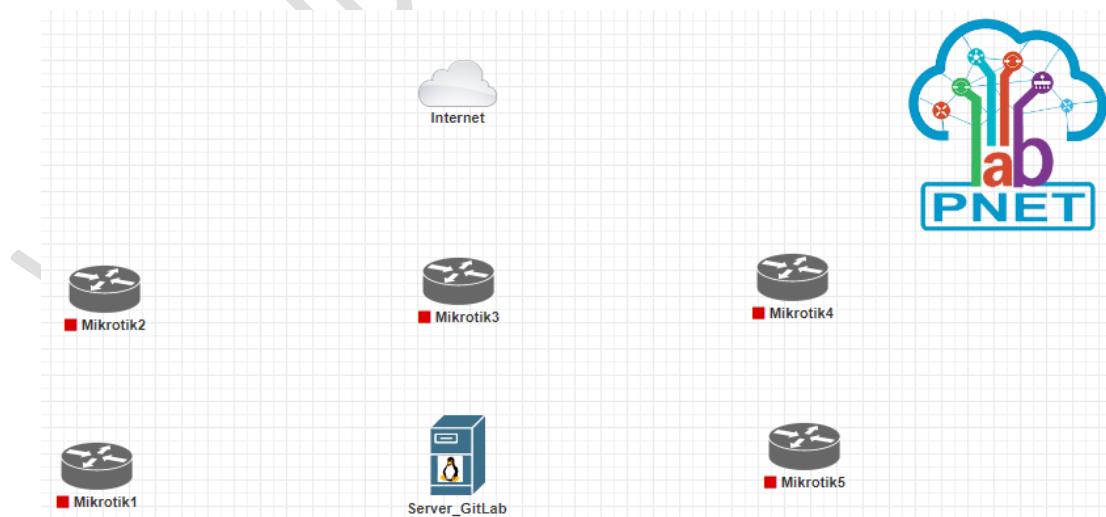


3. Menambahkan *node Mikrotik CHR* versi **7.5** sejumlah 5 (lima) unit yaitu dengan cara klik kanan pada topologi maka akan tampil kotak dialog **ADD A NEW OBJECT** dan klik pada pilihan **Node**, seperti terlihat pada gambar berikut:



Tampil kotak dialog **ADD A NEW NODE**. Lakukan penyesuaian pada nilai dari parameter **Template** menjadi **MikroTik RouterOS** dan pilihan **Number of nodes to add** menjadi **5** serta pilihan **Image** menjadi **mikrotik-7.5**, seperti terlihat pada gambar berikut:

Scroll ke bawah dan klik tombol **Save** untuk menyimpan perubahan. Lakukan penyesuaian atau penataan ulang lokasi penempatan lima *object node router MikroTik RouterOS* tersebut pada topologi sehingga hasilnya akan terlihat seperti pada gambar berikut:



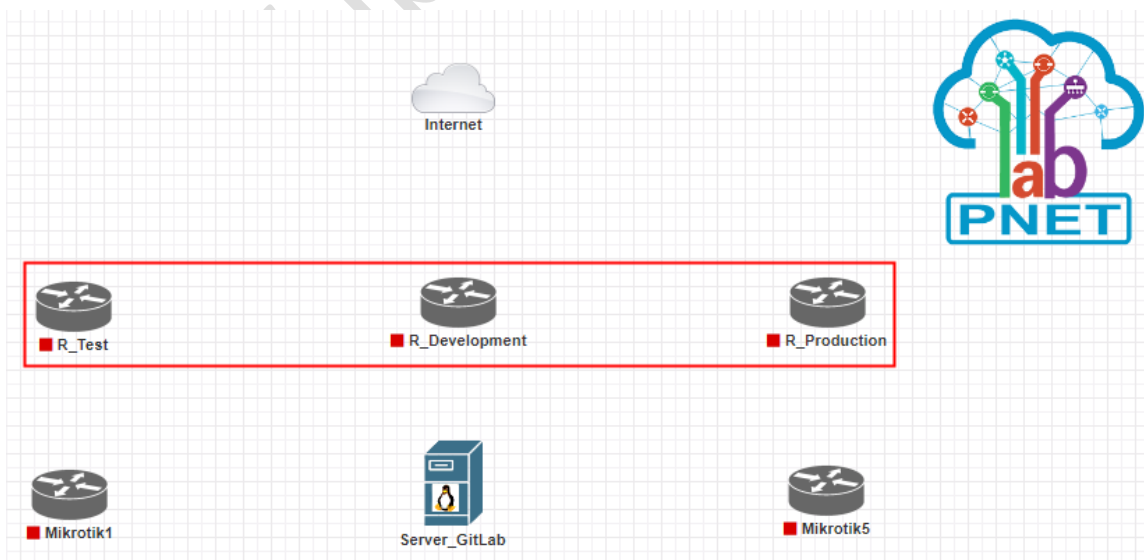
4. Mengubah penamaan dari *node Mikrotik2* menjadi **R_Test** yaitu dengan cara menempatkan kursor *mouse* diatas *node MikroTik2* dan klik tombol **Edit**, seperti terlihat pada gambar berikut:



Tampil kotak dialog **EDIT NODE** dan lakukan penyesuaian nilai dari parameter **Name** menjadi **R_Test**, seperti terlihat pada gambar berikut:

Scroll ke bawah dan klik tombol **Save** untuk menyimpan perubahan.

Dengan cara yang sama lakukan perubahan penamaan dari *node Mikrotik3* menjadi **R_Development**, *node Mikrotik4* menjadi **R_Production** sehingga hasilnya terlihat seperti pada gambar berikut:



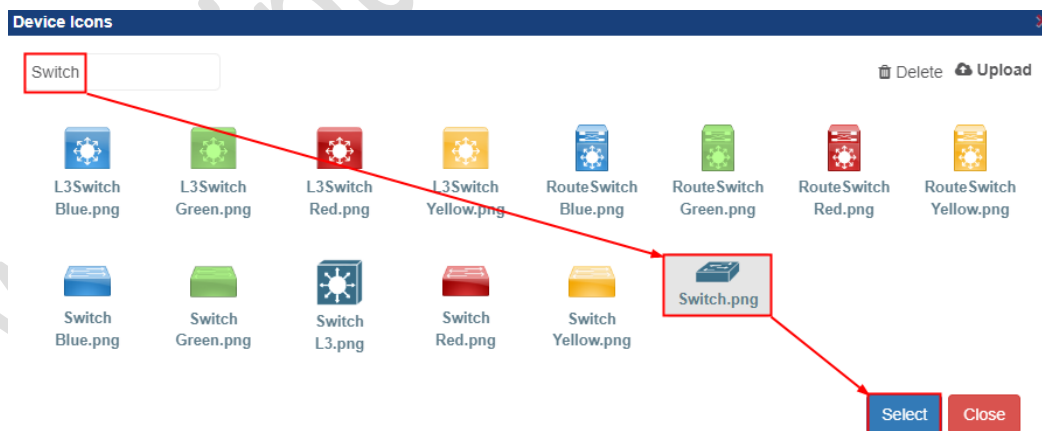
5. Mengubah penamaan dari *node* **Mikrotik1** menjadi **SW_Test** dan **Icon** menjadi **Switch.png** serta jumlah *interface Ethernet* menjadi **8** (delapan) yaitu dengan cara menempatkan cursor *mouse* diatas *node MikroTik2* dan klik tombol **Edit**. Pada kotak dialog **Edit Node** yang tampil lakukan penyesuaian pada nilai dari parameter **Name** menjadi **SW_Test**, seperti terlihat pada gambar berikut:

The screenshot shows the 'EDIT NODE' dialog box with the following fields:

- Template:** MikroTik RouterOS
- ID:** 1
- Image:** mikrotik-7.5
- Name:** SW_Test (highlighted with a red box)
- Description:** MikroTik RouterOS
- Icon:** Router.png (highlighted with a red box)

A red arrow points from the 'Name' field to the 'Icon' field.

Klik **Router.png** pada parameter **Icon** untuk mengubah *icon* dari *node* maka akan tampil kotak dialog **Device Icons**. Pada inputan pencarian di bagian pojok kiri atas, masukkan kata kunci pencarian yaitu **Switch** maka akan terlihat pilihan icon untuk **Switch**. Pilih pada **Switch.png** serta klik tombol **Select**, seperti terlihat pada gambar berikut:



Kembali tampil kotak dialog **ADD A NEW NETWORK**. Terlihat nilai dari parameter **Icon** telah berubah menjadi **Switch.png**, seperti yang ditunjukkan pada gambar berikut:

EDIT NODE

Template ☐ Show all unsupported

MikroTik RouterOS

ID: 1 Image: mikrotik-7.5

Name: SW_Test

Description: MikroTik RouterOS

Icon:  Switch.png

Scroll ke bawah dan lakukan penyesuaian pada nilai dari parameter **Ethernet** menjadi **8** (delapan) serta klik tombol **Save** untuk menyimpan perubahan, seperti terlihat pada gambar berikut:

Ethernet: 8

Save and Restart to take effect

Qemu Arch: x86_64 Qemu NIC: e1000 Qemu Version: 2.4.0(Default)

Qemu Options: -machine type=pc.1.0,accel=kvm -serial mon:stdio -nographic -nodefconfig -nodefaults -display none -vga std -rtc base=utc

Config Script: config_mikrotik.py Script Timeout: 1200

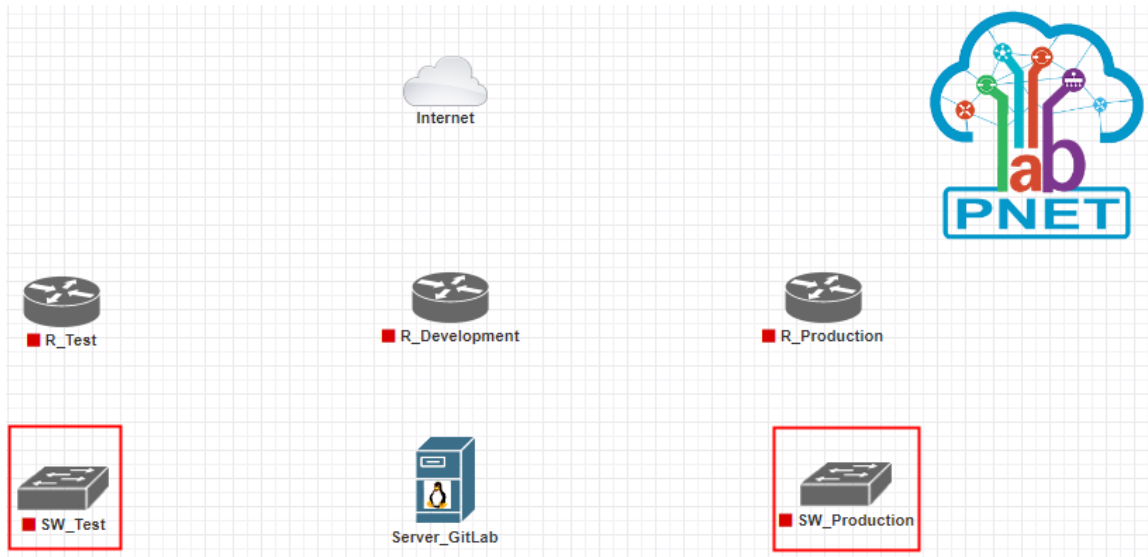
Config: None

Delay: 0

Size (px): 0 Left (px): 243 Top (px): 369

Save Cancel

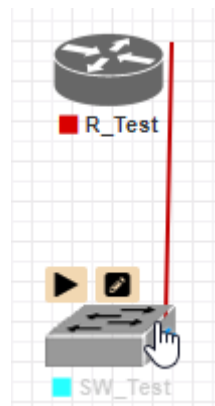
Dengan cara yang sama lakukan perubahan penamaan dari *node* **Mikrotik5** menjadi **SW_Production** dan **Icon** menjadi **Switch.png** serta jumlah *interface* **Ethernet** menjadi **8** (delapan), sehingga terlihat seperti gambar berikut:



6. Menghubungkan antar *node* yang telah terdapat pada topologi baik antar *router* dan *switch* serta *network Internet & Server_Gitlab*. Koneksi antar *node* yang terdapat di topologi PNETLab dapat dilakukan dengan cara menempatkan kursor *mouse* di atas *node* yang akan dihubungkan, sebagai contoh pada *node R_Test* maka akan tampil simbol **connector**, seperti terlihat pada gambar berikut:

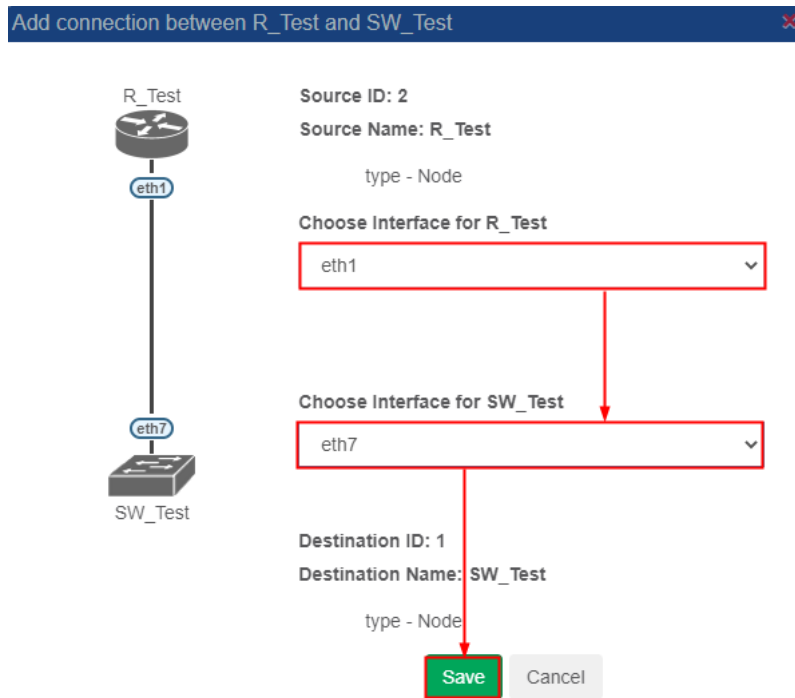


Kemudian lakukan **drag and drop** ke *node* tujuan, sebagai contoh ke *node SW_Test* sehingga terlihat seperti gambar berikut:

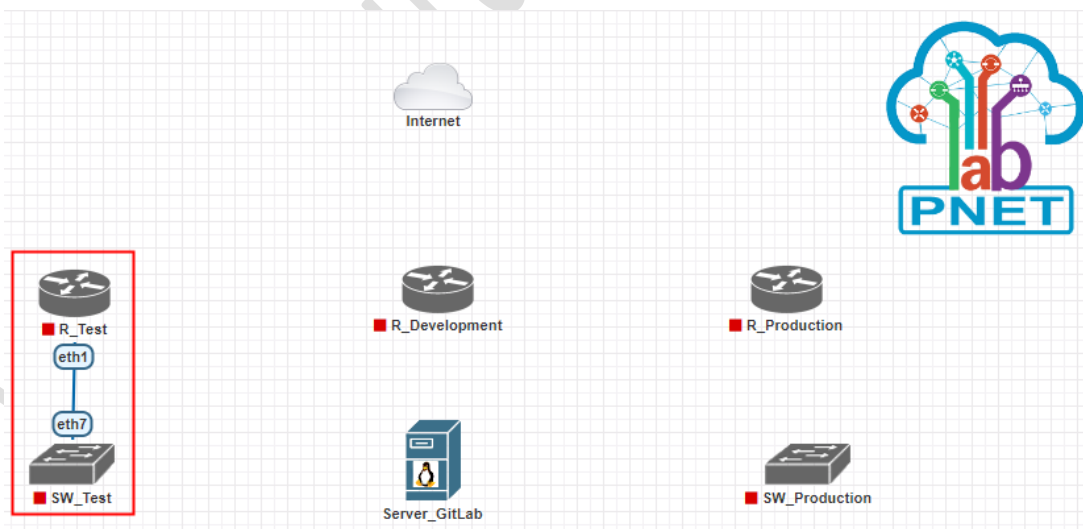


Selanjutnya akan tampil kotak dialog **Add connection between R_Test and SW_Test**. Lakukan penyesuaian pada pilihan parameter **Choose Interface for R_Test** sebagai *interface* pada *node*

sumber yaitu **eth1** dan **Choose Interface for SW_Test** sebagai *interface* pada *node* sumber yaitu **eth7**, seperti terlihat pada gambar berikut:

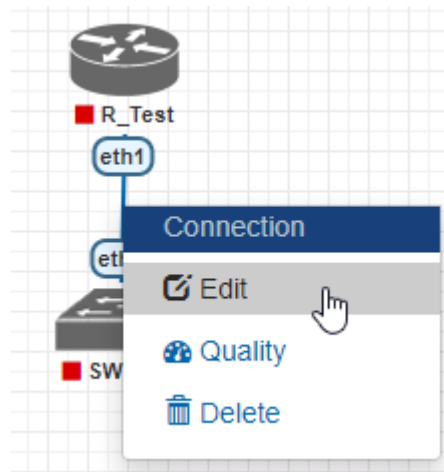


Klik tombol **Save** untuk menyimpan perubahan sehingga hasil dari pengkoneksian tersebut akan terlihat seperti gambar berikut:



Untuk mengubah koneksi antar *node* atau menghapus link koneksi yang sudah dibentuk maka dapat dilakukan dengan menempatkan kursor *mouse* pada link yang menghubungkan antar

node tersebut kemudian klik kanan pada *mouse* sehingga akan tampil kotak dialog *Connection*, seperti terlihat pada gambar berikut:

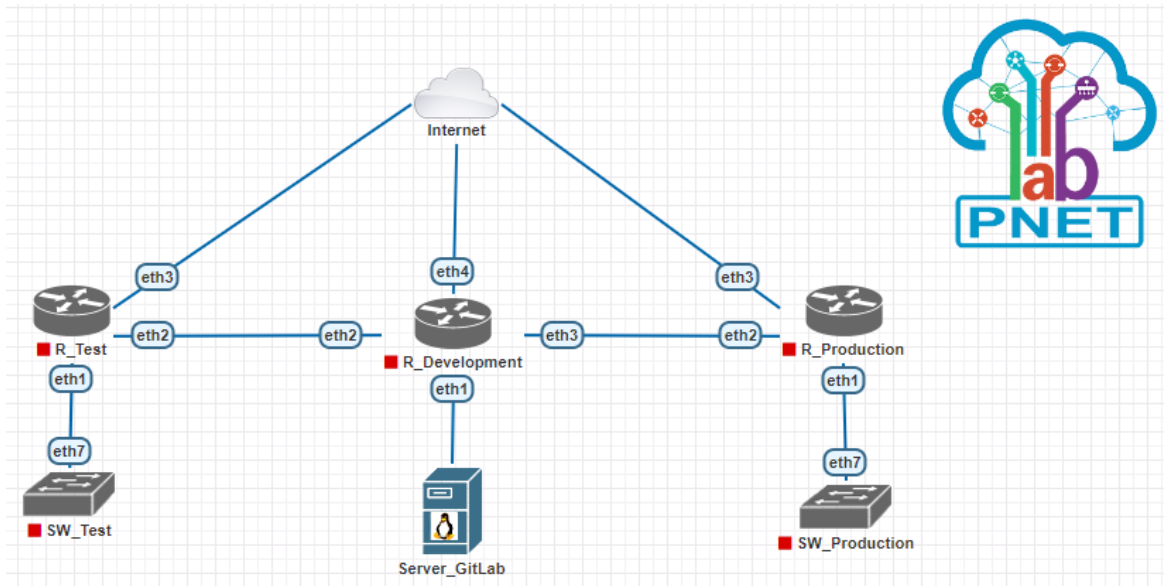


Pilih **Edit** untuk mengubah koneksi. Sebaliknya **Delete** untuk menghapus koneksi.

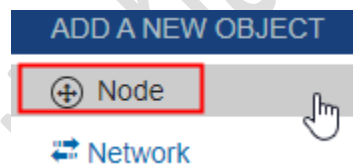
Dengan cara yang sama lakukan pengkoneksian antar *node* dengan mengikuti ketentuan seperti pada tabel berikut:

No.	Node Sumber	Interface pada Node Sumber	Node Tujuan	Interface pada Node Tujuan
1.	R_Test	Ether2	R_Development	Ether2
		Ether3	Internet	
2.	R_Development	Ether1	Server_GitLab	
		Ether3	R_Production	Ether2
		Ether4	Internet	
3.	R_Production	Ether1	SW_Production	Ether7
		Ether3	Internet	

Hasil akhir pembuatan koneksi antar *node* sesuai dengan ketentuan pada tabel tersebut, seperti terlihat pada gambar berikut:



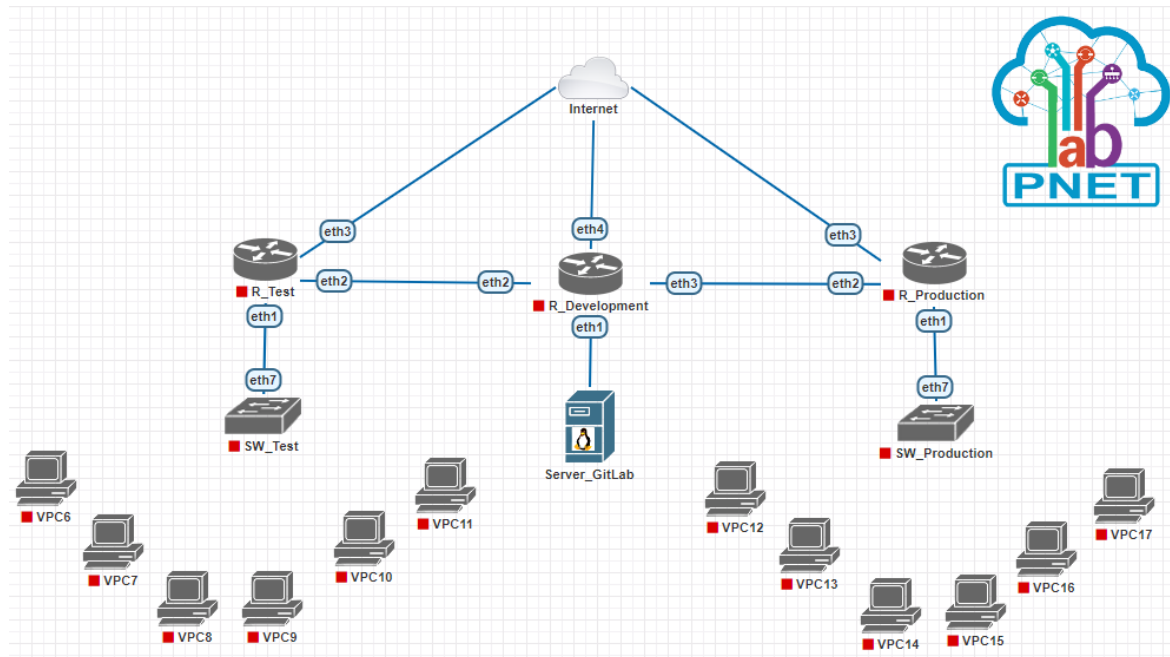
7. Menambahkan *node Virtual PC (VPCS)* sejumlah **12 (duabelas)** unit yaitu dengan cara klik kanan pada topologi maka akan tampil kotak dialog **ADD A NEW OBJECT** dan klik pada pilihan **Node**, seperti terlihat pada gambar berikut:



Tampil kotak dialog **ADD A NEW NODE**. Lakukan penyesuaian pada nilai dari parameter **Template** menjadi **VPCS** dan pilihan **Number of nodes to add** menjadi **12**, seperti terlihat pada gambar berikut:

Scroll ke bawah dan klik tombol **Save** untuk menyimpan perubahan.

Lakukan penyesuaian atau penataan ulang lokasi penempatan keduabelas *object node* VPCS tersebut pada topologi sehingga hasilnya akan terlihat seperti pada gambar berikut:



8. Mengubah penamaan dari *node* **VPC6** menjadi **MKT1_Test** yaitu dengan cara menempatkan kursor *mouse* diatas *node* VPC6 dan klik tombol **Edit**, seperti terlihat pada gambar berikut:



Tampil kotak dialog **EDIT NODE** dan lakukan penyesuaian nilai dari parameter **Name** menjadi **MKT1_Test**, seperti terlihat pada gambar berikut:

EDIT NODE

Template

Virtual PC (VPCS)

ID

6

Name

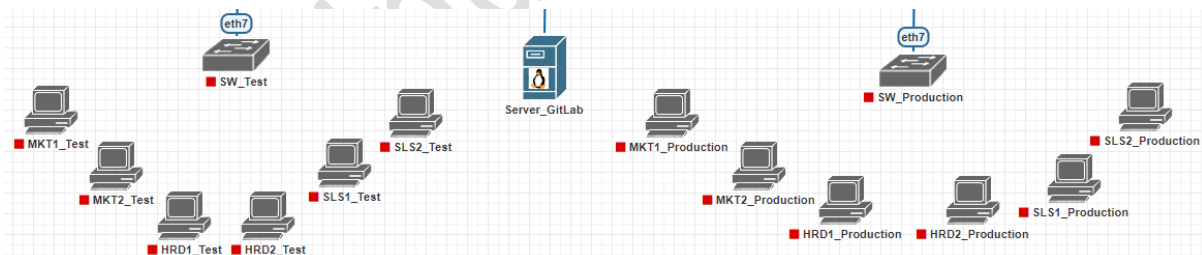
MKT1_Test

Scroll ke bawah dan klik tombol **Save** untuk menyimpan perubahan.

Dengan cara yang sama lakukan perubahan penamaan dari *node* **VPC7 sampai dengan VPC17** sesuai dengan ketentuan pada tabel berikut:

No.	VPC	Nama Node Baru
1.	VPC7	MKT2_Test
2.	VPC8	HRD1_Test
3.	VPC9	HRD2_Test
4.	VPC10	SLS1_Test
5.	VPC11	SLS2_Test
6.	VPC12	MKT1_Production
7.	VPC13	MKT2_Production
8.	VPC14	HRD1_Production
9.	VPC15	HRD2_Production
10.	VPC16	SLS1_Production
11.	VPC17	SLS2_Production

Cuplikan hasil akhir perubahan nama setiap *node* untuk keseluruhan VPC akan terlihat seperti pada gambar berikut:

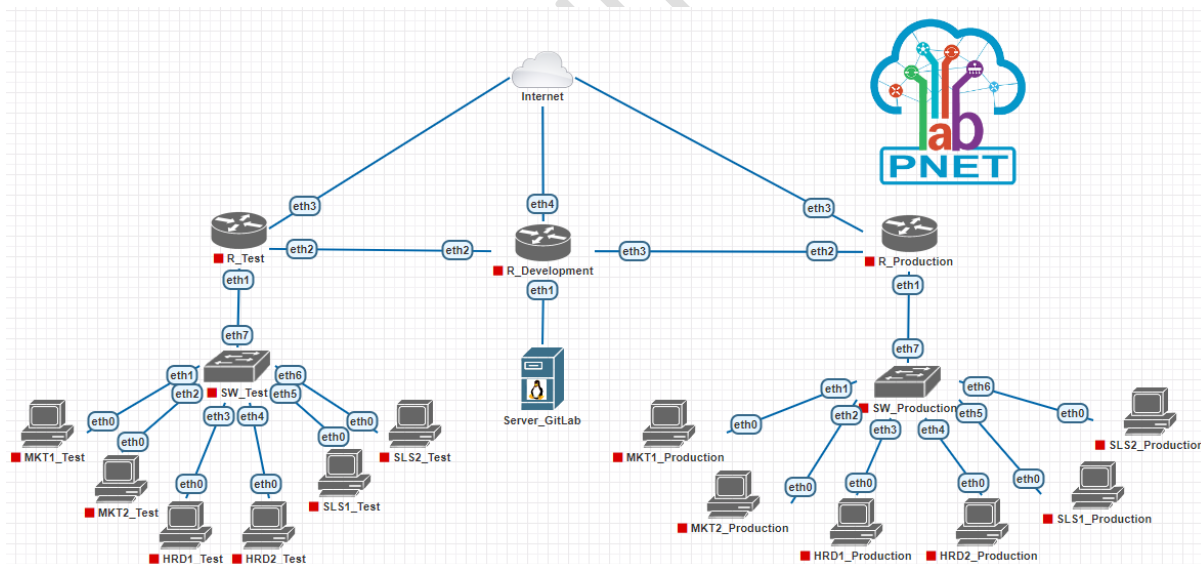


4. Menghubungkan setiap *node* **VPCS** ke **switch** baik **SW_Test** maupun **SW_Production** dengan ketentuan seperti tabel berikut:

No.	Node Sumber	Interface pada Node Sumber	Node Tujuan	Interface pada Node Tujuan
1.	MKT1_Test	eth0	SW_Test	Eth1

2.	MKT2_Test			Eth2
3.	HRD1_Test			Eth3
4.	HRD2_Test			Eth4
5.	SLS1_Test			Eth5
6.	SLS2_Test			Eth6
7.	MKT1_Production		SW_Production	Eth1
8.	MKT2_Production			Eth2
9.	HRD1_Production			Eth3
10.	HRD2_Production			Eth4
11.	SLS1_Production			Eth5
12.	SLS2_Production			Eth6

Lakukan koneksi setiap **VPCS** ke **switch** sesuai dengan ketentuan pada tabel tersebut dengan mengikuti tata cara koneksi yang telah dijelaskan sebelumnya sehingga cuplikan hasil akhirnya akan terlihat seperti pada gambar berikut:



- Menambahkan keterangan terkait **Test Network** dan **Production Network** termasuk VLAN dan pengalamatan IP dari setiap VLAN tersebut pada topologi dengan menggunakan *object Text*. Klik kanan pada topologi maka akan tampil kotak dialog **ADD A NEW OBJECT** dan klik pada pilihan **Text**, seperti terlihat pada gambar berikut:

ADD A NEW OBJECT

 Node

 Network

 Picture

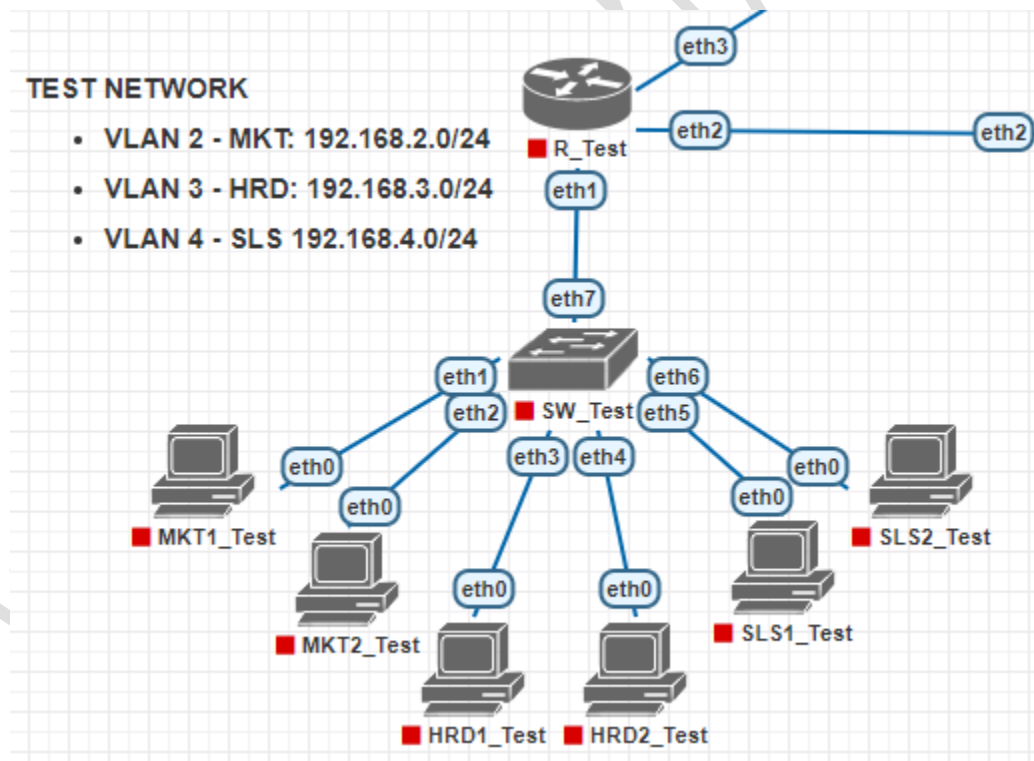
 A Text

 Line

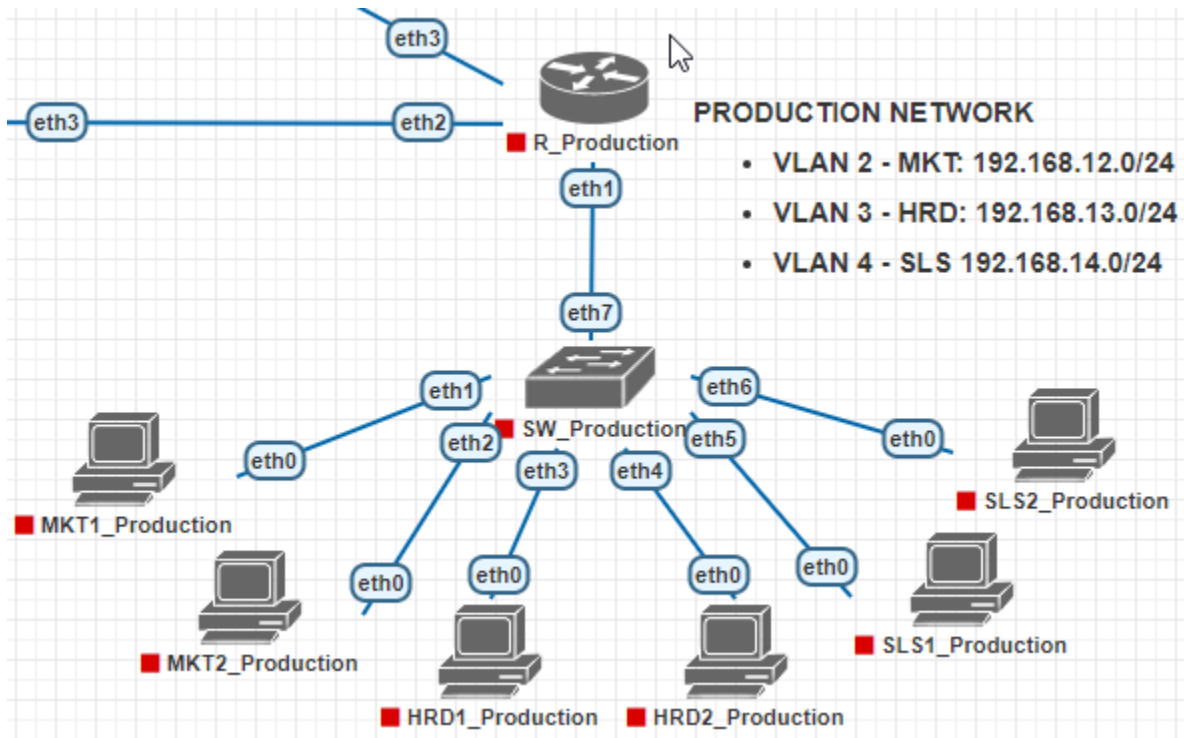
 Undo Position

 Redo Position

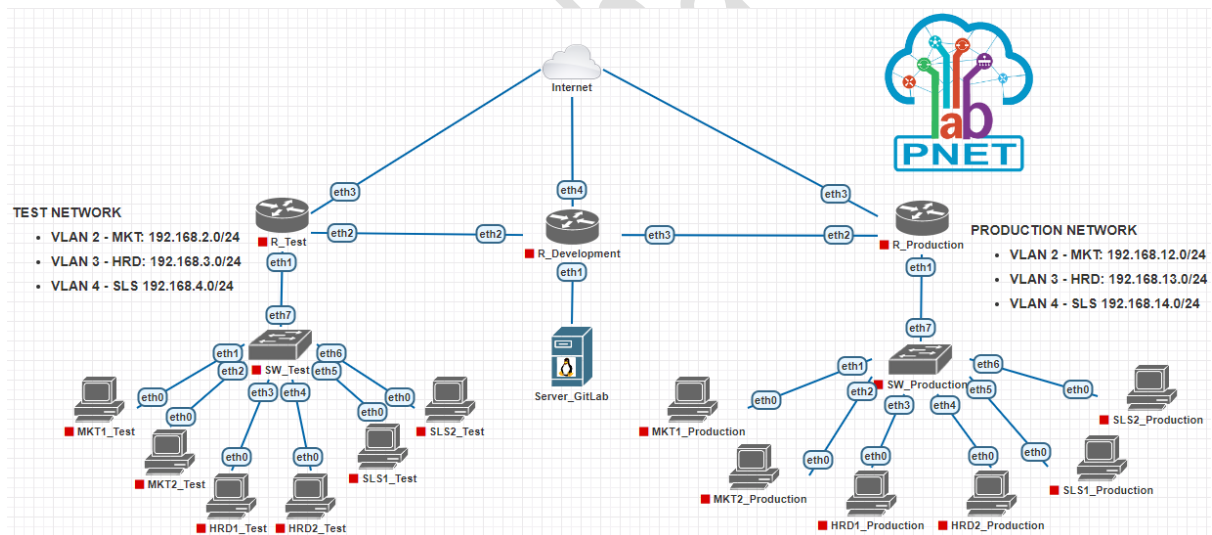
Cantumkan konten pada **textarea** yang tampil dan lakukan penyesuaian atau penataan ulang lokasi penempatan dari textare tersebut sehingga hasil akhirnya untuk **TEST NETWORK**, seperti terlihat pada gambar berikut:



Sedangkan untuk **PRODUCTION NETWORK**, seperti terlihat pada gambar berikut:



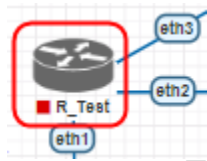
Hasil akhir dari pembuatan topologi pada lab **NetDevOps**, seperti terlihat pada gambar berikut:



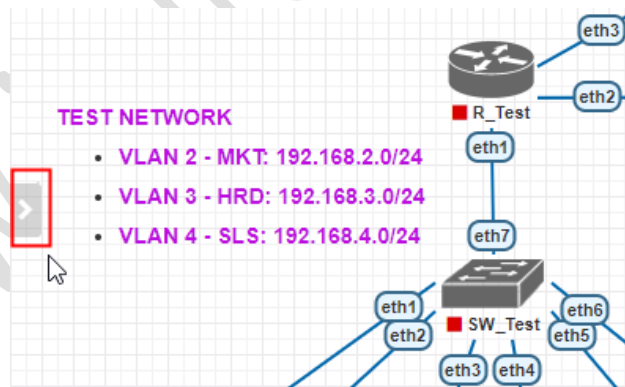
BAB 4

KONFIGURASI DASAR LAB NETDEVOPS

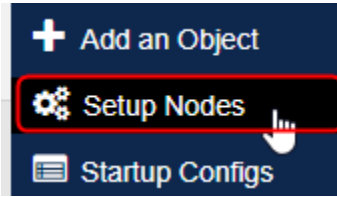
Sebelum dapat melakukan konfigurasi dasar pada *node* baik *router*, *switch* dan VPCS di lab *NetDevOps* maka *node* di dalam lab harus di jalankan secara individual (per *node*) atau per grup (kelompok) atau keseluruhan. *Node-node* yang belum berjalan pada lab ditandai dengan warna abu-abu dan simbol kotak berwarna merah dibawah *node*, seperti yang ditunjukkan pada gambar berikut:



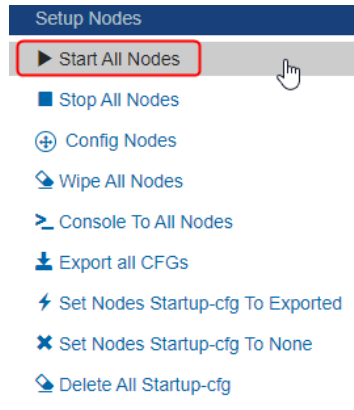
Sebagai contoh untuk menjalankan keseluruhan *node* secara bersamaan maka gerakan penunjuk (kursor) *mouse* ke sebelah kiri menuju *side bar* yang diperkecil sehingga memperluas *side bar* tersebut, seperti yang ditunjukkan pada gambar berikut:



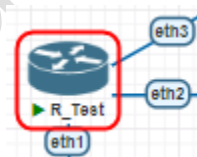
Pada menu *side bar* yang tampil klik pada **Setup Nodes**, seperti yang ditunjukkan pada gambar berikut:



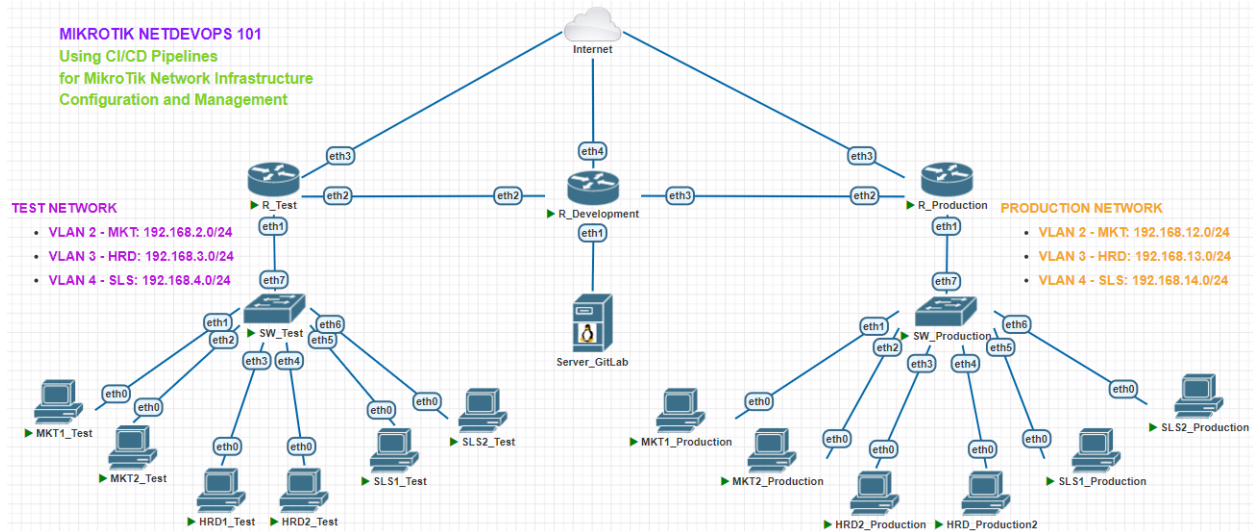
Selanjutnya akan tampil menu **Setup Nodes** dan pilih **Start All Nodes**, seperti yang ditunjukkan pada gambar berikut:



Node yang telah berjalan dan dalam kondisi bekerja/berfungsi ditandai dengan warna biru dan dibawah node tersebut terdapat simbol segitiga berwarna hijau, seperti yang ditunjukkan pada gambar berikut:

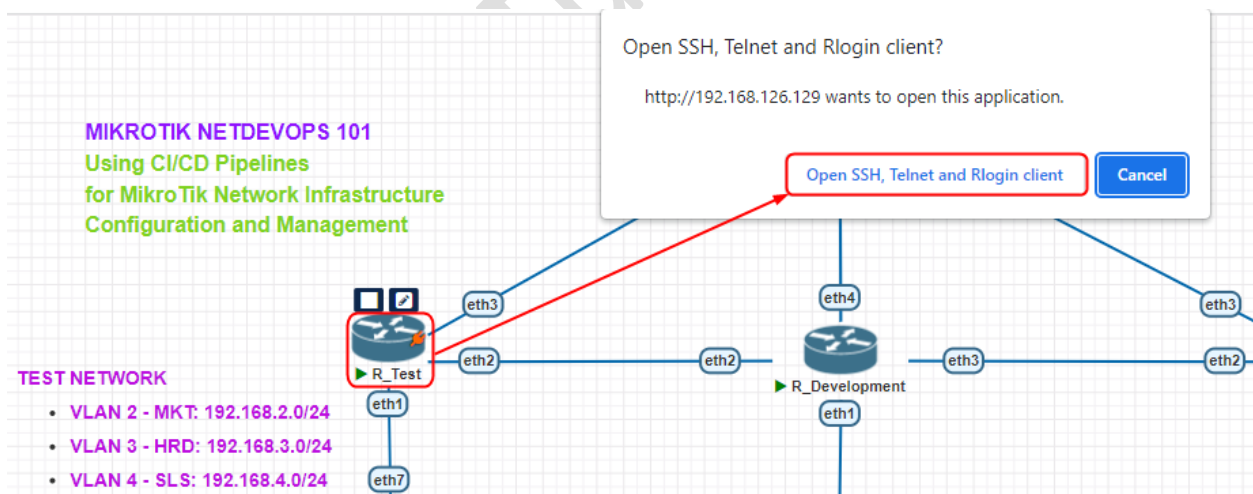


Keseluruhan *node* pada lab yang telah berjalan, seperti yang ditunjukkan pada gambar berikut:



A. Mengakses Console Dari Node Pada Lab

Untuk memulai konfigurasi *node* yang terdapat pada lab maka diperlukan akses ke *console* dari *node* tersebut. Sebagai contoh klik pada *node* dengan nama **R_Test** maka akan tampil kotak dialog **Open SSH, Telnet and Rlogin client?**, seperti yang ditunjukkan pada gambar berikut:



Klik pada tombol **Open SSH, Telnet and Rlogin client** maka akan tampil kotak dialog *telnet* untuk **R_Test**. Pada kotak dialog tersebut tekan tombol **Enter** sehingga tampil *form login* dari **MikroTik**, seperti yang ditunjukkan pada gambar berikut:



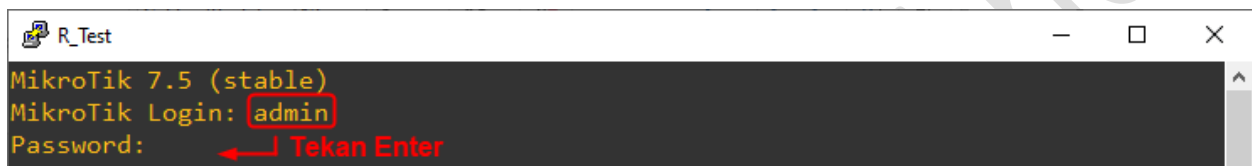
```

MikroTik 7.5 (stable)
MikroTik Login:

```

Secara *default* akun otentikasi *login* ke Mikrotik menggunakan **username “admin”** dengan **password** kosong (**blank**).

Pada *input MikroTik Login*, masukkan “**admin**” dan tekan tombol **Enter**. Sedangkan pada *input Password*, tekan tombol **Enter**, seperti terlihat pada gambar berikut:



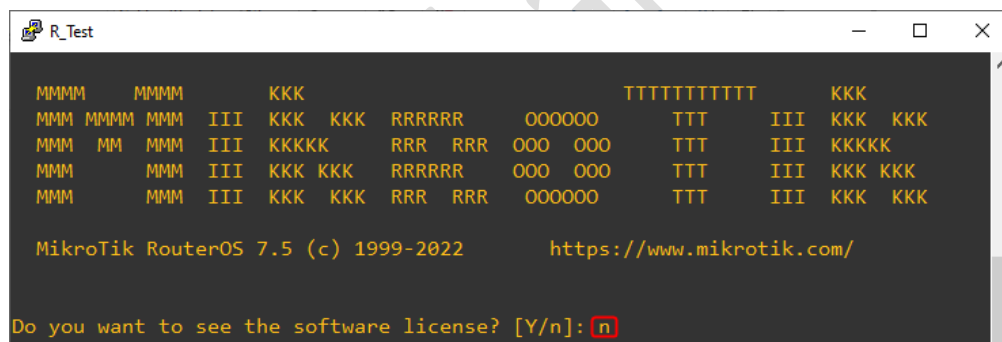
```

MikroTik 7.5 (stable)
MikroTik Login: admin
Password:

```

← Tekan Enter

Tampil pesan konfirmasi **Do you want to see the software license? [Y/n]**. Ketik **n** untuk tidak melihat lisensi perangkat lunak, seperti yang ditunjukkan pada gambar berikut:



```

MMMM  MMMM  KKK               TTTTTTTTTT  KKK
MMM MMMM MMM III KKK KKK RRRRRR  000000    TTT   III KKK KKK
MMM MM  MMM III KKKKK  RRR RRR  000 000    TTT   III KKKKK
MMM  MMM  III KKK KKK  RRRRRR  000 000    TTT   III KKK KKK
MMM  MMM  III KKK KKK  RRR RRR  000000    TTT   III KKK KKK

MikroTik RouterOS 7.5 (c) 1999-2022      https://www.mikrotik.com/

Do you want to see the software license? [Y/n]: n

```

Selanjutnya tampil pesan **Change your password** untuk mengubah sandi dari user “**admin**”. Sebagai contoh *password* yang ingin digunakan adalah “**admin**” sehingga pada **input new password>** yang tampil, masukkan “**admin**” dan tekan tombol **Enter**. Tampil **input repeat new password>**, untuk memasukkan kembali sandi baru yaitu “**admin**” dan tekan tombol **Enter**, seperti terlihat pada gambar berikut:

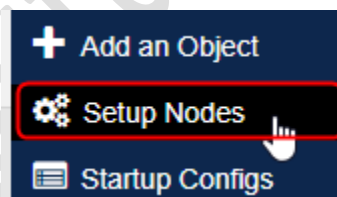
```
Change your password
new password> *****
repeat new password> *****

Password changed
[admin@MikroTik] >
```

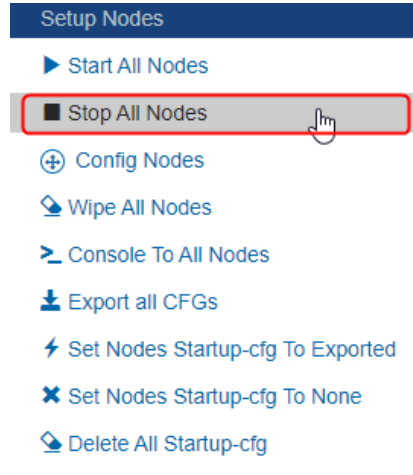
Konfigurasi pada *node-node* lainnya di lab *NetDevOps* dapat dilakukan dengan cara yang sama. Apabila konfigurasi telah selesai dilakukan maka kotak dialog *telnet* untuk *node* tersebut dapat ditutup.

PERHATIAN:

Untuk mengakhiri pemakaian lab maka keseluruhan *node* dapat dihentikan baik secara individual (per *node*) atau grup (per kelompok) atau keseluruhan dengan cara yang mirip dengan ketika menjalankan *node*. Gerakkan penunjuk (kursor) *mouse* ke sebelah kiri menuju *side bar* yang diperkecil sehingga memperluas *side bar* tersebut. Pada menu *side bar* yang tampil klik pada **Setup Nodes**, seperti yang ditunjukkan pada gambar berikut:



Selanjutnya akan tampil menu **Setup Nodes** dan pilih **Stop All Nodes**, seperti yang ditunjukkan pada gambar berikut:



B. Konfigurasi Dasar Router R_Development

Adapun langkah-langkah konfigurasi dasar pada *router R_Development* adalah sebagai berikut:

1. Mengakses *console* dari *node* dengan cara memilih (klik) **R_Development** pada halaman *topology*. Langkah selanjutnya serupa dengan yang telah dijelaskan pada bagian A sebelumnya.
2. Mengatur nama pengenalan dari perangkat atau identitas dari sistem dengan mengeksekusi perintah `system identity set name=R_Development`.

```
[admin@MikroTik] > system identity set name=R_Development
[admin@R_Development] >
```

3. Menampilkan informasi *interface* dengan pengaturan sebagai DHCP Client dengan mengeksekusi perintah `ip dhcp-client print`.

```
[admin@R_Development] > ip dhcp-client print
Columns: INTERFACE, USE-PEER-DNS, ADD-DEFAULT-ROUTE, STATUS
# INTERFACE  USE-PEER-DNS  ADD-DEFAULT-ROUTE  STATUS
0 ether1      yes            yes                 searching...
```

Terlihat *interface ether1* diatur sebagai *DHCP client*.

4. Menghapus pengaturan *DHCP Client* pada *interface ether1* dengan mengeksekusi perintah `ip dhcp-client remove 0`.

```
[admin@R_Development] > ip dhcp-client remove 0
```

5. Mengatur agar *interface ether4* yang terhubung ke *Internet* sebagai **DHCP Client** dengan mengeksekusi perintah `ip dhcp-client add interface=ether4 disabled=no`.

```
[admin@R_Development] > ip dhcp-client add interface=ether4 disabled=no
```

6. Memverifikasi hasil pengaturan *DHCP Client* pada *interface ether4* dengan mengeksekusi perintah `ip dhcp-client print`.

```
[admin@R_Development] > ip dhcp-client print
Columns: INTERFACE, USE-PEER-DNS, ADD-DEFAULT-ROUTE, STATUS, ADDRESS
# INTERFACE  USE-PEER-DNS  ADD-DEFAULT-ROUTE  STATUS  ADDRESS
0 ether4      yes            yes            bound   10.0.137.134/24
```

Terlihat *interface ether4* telah memperoleh pengalamatan IP secara dinamis dari *DHCP Server* dengan alamat **10.0.137.134/24**.

7. Menampilkan informasi terkait **Domain Name System (DNS)** dengan mengeksekusi perintah `ip dns print`.

```
[admin@R_Development] > ip dns print
servers:
dynamic-servers: 10.0.137.1
use-doh-server:
verify-doh-cert: no
allow-remote-requests: no
max-udp-packet-size: 4096
query-server-timeout: 2s
query-total-timeout: 10s
max-concurrent-queries: 100
max-concurrent-tcp-sessions: 20
cache-size: 2048KiB
cache-max-ttl: 1w
cache-used: 27KiB
```

Terlihat alamat IP Server DNS yang diperoleh dari *DHCP Server* adalah **10.0.137.1**.

8. Menampilkan informasi tabel *routing* dengan mengeksekusi perintah `ip route print`.

```
[admin@R_Development] > ip route print
Flags: D - DYNAMIC; A - ACTIVE; c, d, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
DST-ADDRESS  GATEWAY  DISTANCE
DAd 0.0.0.0/0  10.0.137.1  1
DAc 10.0.137.0/24 ether4      0
```

9. Memverifikasi koneksi Internet dengan mengeksekusi perintah `ping google.com`.

```
[admin@R_Development] > ping google.com
```

SEQ	HOST	SIZE	TTL	TIME	STATUS
0	216.239.38.120	56	127	92ms811us	
1	216.239.38.120	56	127	65ms621us	
2	216.239.38.120	56	127	67ms694us	

```
sent=3 received=3 packet-loss=0% min-rtt=65ms621us avg-rtt=75ms375us
max-rtt=92ms811us
```

Terlihat verifikasi koneksi ke **google.com** berhasil dilakukan. Tekan **CTRL+C** untuk menghentikan *ping*.

- Mengatur pengalamatan IP pada *interface ether1* dengan mengeksekusi perintah **ip address add address=192.168.0.9/29 interface=ether1**.

```
[admin@R_Development] > ip address add address=192.168.0.9/29 interface=ether1
```

- Mengatur pengalamatan IP pada *interface ether2* dengan mengeksekusi perintah **ip address add address=192.168.0.2/30 interface=ether2**.

```
[admin@R_Development] > ip address add address=192.168.0.2/30 interface=ether2
```

- Mengatur pengalamatan IP pada *interface ether3* dengan mengeksekusi perintah **ip address add address=192.168.0.5/30 interface=ether3**.

```
[admin@R_Development] > ip address add address=192.168.0.5/30 interface=ether3
```

- Memverifikasi hasil pengaturan pengalamatan IP pada *interface* dengan mengeksekusi perintah **ip address print**.

```
[admin@R_Development] > ip address print
```

Flags: D - DYNAMIC

Columns: ADDRESS, NETWORK, INTERFACE

#	ADDRESS	NETWORK	INTERFACE
0	D 10.0.137.134/24	10.0.137.0	ether4
1	192.168.0.9/29	192.168.0.8	ether1
2	192.168.0.2/30	192.168.0.0	ether2
3	192.168.0.5/30	192.168.0.4	ether3

- Mengatur **Network Address Translation (NAT)** agar **Server GitLab** yang terdapat di LAN dari *router R_Development* dapat mengakses ke Internet dengan mengeksekusi perintah **ip firewall nat add chain=srcnat out-interface=ether4 action=masquerade**.

```
[admin@R_Development] > ip firewall nat add chain=srcnat out-interface=ether4 action=masquerade
```

15. Memverifikasi hasil penambahan NAT dengan mengeksekusi perintah `ip firewall nat print`.

```
[admin@R_Development] > ip firewall nat print
Flags: X - disabled, I - invalid; D - dynamic
0 chain=srcnat action=masquerade out-interface=ether4
```

16. Menambahkan **static route** agar dapat menjangkau alamat *network* dari LAN R_Test yaitu **192.168.1.0/24** melalui *router* R_Test dengan mengeksekusi perintah `ip route add dst-address=192.168.1.0/24 gateway=192.168.0.1`.

```
[admin@R_Development] > ip route add dst-address=192.168.1.0/24 gateway=192.168.0.1
```

17. Menambahkan **static route** agar dapat menjangkau alamat *network* dari LAN R_Production yaitu **192.168.11.0/24** melalui *router* R_Production dengan mengeksekusi perintah `ip route add dst-address=192.168.11.0/24 gateway=192.168.0.6`.

```
[admin@R_Development] > ip route add dst-address=192.168.11.0/24 gateway=192.168.0.6
```

18. Menampilkan informasi tabel *routing* dengan mengeksekusi perintah `ip route print`.

```
[admin@R_Development] > ip route print
Flags: D - DYNAMIC; A - ACTIVE; c, s, d, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
# DST-ADDRESS GATEWAY DISTANCE
DAc 0.0.0.0/0 10.0.137.1 1
DAc 10.0.137.0/24 ether4 0
DAc 192.168.0.0/30 ether2 0
DAc 192.168.0.4/30 ether3 0
DAc 192.168.0.8/29 ether1 0
0 As 192.168.1.0/24 192.168.0.1 1
1 As 192.168.11.0/24 192.168.0.6 1
```

C. Konfigurasi Dasar Router R_Test

Adapun langkah-langkah konfigurasi dasar pada *router* R_Test adalah sebagai berikut:

1. Mengakses *console* dari *node* dengan cara memilih (klik) R_Test pada halaman *topology*. Langkah selanjutnya serupa dengan yang telah dijelaskan pada bagian A sebelumnya.
2. Mengatur nama pengenalan dari perangkat atau identitas dari sistem dengan mengeksekusi perintah `system identity set name=R_Test`.

```
[admin@MikroTik] > system identity set name=R_Test
```

3. Menampilkan informasi *interface* dengan pengaturan sebagai DHCP Client dengan mengeksekusi perintah `ip dhcp-client print`.

```
[admin@R_Test] > ip dhcp-client print
Columns: INTERFACE, USE-PEER-DNS, ADD-DEFAULT-ROUTE, STATUS
# INTERFACE USE-PEER-DNS ADD-DEFAULT-ROUTE STATUS
0 ether1 yes yes searching...
```

Terlihat *interface ether1* diatur sebagai *DHCP client*.

4. Menghapus pengaturan *DHCP Client* pada *interface ether1* dengan mengeksekusi perintah `ip dhcp-client remove 0`.

```
[admin@R_Test] > ip dhcp-client remove 0
```

5. Memverifikasi penghapusan *interface ether1* sebagai *DHCP Client* dengan mengeksekusi perintah `ip dhcp-client print`.

```
[admin@R_Test] > ip dhcp-client print
```

6. Mengatur pengalamatan IP pada *interface ether1* dengan mengeksekusi perintah `ip address add address=192.168.1.1/24 interface=ether1`.

```
[admin@R_Test] > ip address add address=192.168.1.1/24 interface=ether1
```

7. Mengatur pengalamatan IP pada *interface ether2* dengan mengeksekusi perintah `ip address add address=192.168.0.1/30 interface=ether2`.

```
[admin@R_Test] > ip address add address=192.168.0.1/30 interface=ether2
```

8. Memverifikasi hasil pengaturan pengalamatan IP pada *interface* dengan mengeksekusi perintah `ip address print`.

```
[admin@R_Test] > ip address print
Columns: ADDRESS, NETWORK, INTERFACE
# ADDRESS NETWORK INTERFACE
0 192.168.1.1/24 192.168.1.0 ether1
1 192.168.0.1/30 192.168.0.0 ether2
```

9. Menambahkan **static route** agar dapat menjangkau alamat *network* dari **LAN R_Development** yaitu **192.168.0.8/29** melalui *router R_Development* dengan mengeksekusi perintah `ip route add dst-address=192.168.0.8/29 gateway=192.168.0.2`.

```
[admin@R_Test] > ip route add dst-address=192.168.0.8/29 gateway=192.168.0.2
```

10. Menampilkan informasi tabel *routing* dengan mengeksekusi perintah `ip route print`.

```
[admin@R_Test] > ip route print
Flags: D - DYNAMIC; A - ACTIVE; c, s, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
#      DST-ADDRESS      GATEWAY      DISTANCE
0      DAc 192.168.0.0/30 ether2          0
0      As 192.168.0.8/29 192.168.0.2    1
0      DAc 192.168.1.0/24 ether1          0
```

11. Memverifikasi koneksi ke *router R_Development* dengan mengeksekusi perintah `ping 192.168.0.2`.

```
[admin@R_Test] > ping 192.168.0.2
SEQ HOST                SIZE TTL TIME          STATUS
0 192.168.0.2           56 64 1ms231us
1 192.168.0.2           56 64 1ms935us
sent=2 received=2 packet-loss=0% min-rtt=1ms231us avg-rtt=1ms583us
max-rtt=1ms935us
```

Terlihat verifikasi koneksi ke *router R_Development* berhasil dilakukan. Tekan **CTRL+C** untuk menghentikan *ping*.

D. Konfigurasi Dasar Switch SW_Test

Adapun langkah-langkah konfigurasi dasar pada *switch SW_Test* adalah sebagai berikut:

1. Mengakses *console* dari *node* dengan cara memilih (klik) **SW_Test** pada halaman *topology*. Langkah selanjutnya serupa dengan yang telah dijelaskan pada bagian A sebelumnya.
2. Mengatur nama pengenalan dari perangkat atau identitas dari sistem dengan mengeksekusi perintah `system identity set name=SW_Test`.

```
[admin@MikroTik] > system identity set name=SW_Test
```

3. Menampilkan informasi *interface* dengan pengaturan sebagai DHCP Client dengan mengeksekusi perintah `ip dhcp-client print`.

```
[admin@SW_Test] > ip dhcp-client print
Columns: INTERFACE, USE-PEER-DNS, ADD-DEFAULT-ROUTE, STATUS
# INTERFACE USE-PEER-DNS ADD-DEFAULT-ROUTE STATUS
0 ether1     yes             yes             searching...
```

Terlihat *interface ether1* diatur sebagai *DHCP client*.

- Menghapus pengaturan *DHCP Client* pada *interface ether1* dengan mengeksekusi perintah `ip dhcp-client remove 0`.

```
[admin@SW_Test] > ip dhcp-client remove 0
```

- Memverifikasi penghapusan *interface ether1* sebagai *DHCP Client* dengan mengeksekusi perintah `ip dhcp-client print`.

```
[admin@SW_Test] > ip dhcp-client print
```

- Membuat *interface bridge* bernama **BR_Test** dengan mengeksekusi perintah `interface bridge add name=BR_Test`.

```
[admin@SW_Test] > interface bridge add name=BR_Test
```

- Memverifikasi hasil pembuatan *interface bridge* dengan mengeksekusi perintah `interface bridge print`.

```
[admin@SW_Test] > interface bridge print
Flags: X - disabled, R - running
0 R name="BR_Test" mtu=auto actual-mtu=1500 l2mtu=65535 arp=enabled
  arp-timeout=auto mac-address=22:36:EA:BD:4C:6B protocol-mode=rstp
  fast-forward=yes igmp-snooping=no auto-mac=yes ageing-time=5m
  priority=0x8000 max-message-age=20s forward-delay=15s
  transmit-hold-count=6 vlan-filtering=no dhcp-snooping=no
```

- Menambahkan *interface ether7* sebagai **port** dari **bridge BR_Test** dengan mengeksekusi perintah `interface bridge port add interface=ether7 bridge=BR_Test`.

```
[admin@SW_Test] > interface bridge port add interface=ether7 bridge=BR_Test
```

- Memverifikasi hasil penambahan *interface bridge port* dengan mengeksekusi perintah `interface bridge port print`.

```
[admin@SW_Test] > interface bridge port print
Columns: INTERFACE, BRIDGE, HW, PVID, PRIORITY, PATH-COST, INTERNAL-PATH-COST, H
ORIZON
# INTERFACE BRIDGE HW PVID PRIORITY PATH-COST IN HORIZON
0 ether7 BR_Test yes 1 0x80 10 10 none
```

- Mengatur pengalamatan IP pada *interface ether1* dengan mengeksekusi perintah `ip address add address=192.168.1.2/24 interface=ether1`.

```
[admin@SW_Test] > ip address add address=192.168.1.2/24 interface=BR_Test
```

11. Memverifikasi hasil pengaturan pengalamatan IP pada *interface* dengan mengeksekusi perintah **ip address print**.

```
[admin@SW_Test] > ip address print
Columns: ADDRESS, NETWORK, INTERFACE
# ADDRESS      NETWORK      INTERFACE
0 192.168.1.2/24 192.168.1.0  BR_Test
```

12. Menambahkan **default route** agar dapat menjangkau alamat *network* yang tidak terhubung langsung melalui *router R_Test* dengan mengeksekusi perintah **ip route add gateway=192.168.1.1**.

```
[admin@SW_Test] > ip route add gateway=192.168.1.1
```

13. Menampilkan informasi tabel *routing* dengan mengeksekusi perintah **ip route print**.

```
[admin@SW_Test] > ip route print
Flags: D - DYNAMIC; A - ACTIVE; c, s, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
#      DST-ADDRESS      GATEWAY      DISTANCE
0  As 0.0.0.0/0          192.168.1.1      1
    DAc 192.168.1.0/24  BR_Test          0
```

14. Memverifikasi koneksi ke *router R_Test* dengan mengeksekusi perintah **ping 192.168.1.1**.

```
[admin@SW_Test] > ping 192.168.1.1
SEQ HOST      SIZE TTL TIME      STATUS
0 192.168.1.1  56  64 5ms539us
1 192.168.1.1  56  64 2ms25us
2 192.168.1.1  56  64 4ms797us
sent=3 received=3 packet-loss=0% min-rtt=2ms25us avg-rtt=4ms120us
max-rtt=5ms539us
```

Terlihat verifikasi koneksi ke **router R_Test** berhasil dilakukan. Tekan **CTRL+C** untuk menghentikan *ping*.

15. Memverifikasi koneksi ke *router R_Development* dengan mengeksekusi perintah **ping 192.168.0.2**.

```
[admin@SW_Test] > ping 192.168.0.2
SEQ HOST      SIZE TTL TIME      STATUS
0 192.168.0.2  56  63 4ms260us
1 192.168.0.2  56  63 3ms761us
2 192.168.0.2  56  63 2ms411us
sent=3 received=3 packet-loss=0% min-rtt=2ms411us avg-rtt=3ms477us
max-rtt=4ms260us
```

Terlihat verifikasi koneksi ke *router R_Development* berhasil dilakukan. Tekan **CTRL+C** untuk menghentikan *ping*.

E. Konfigurasi Dasar Router R_Production

Adapun langkah-langkah konfigurasi dasar pada *router R_Production* adalah sebagai berikut:

1. Mengakses *console* dari *node* dengan cara memilih (klik) **R_Production** pada halaman *topology*. Langkah selanjutnya serupa dengan yang telah dijelaskan pada bagian A sebelumnya.
2. Mengatur nama pengenalan dari perangkat atau identitas dari sistem dengan mengeksekusi perintah **system identity set name=R_Production**.

```
[admin@MikroTik] > system identity set name=R_Production
```

3. Menampilkan informasi *interface* dengan pengaturan sebagai DHCP Client dengan mengeksekusi perintah **ip dhcp-client print**.

```
[admin@R_Production] > ip dhcp-client print
Columns: INTERFACE, USE-PEER-DNS, ADD-DEFAULT-ROUTE, STATUS
# INTERFACE  USE-PEER-DNS  ADD-DEFAULT-ROUTE  STATUS
0 ether1      yes            yes                 searching...
```

Terlihat *interface ether1* diatur sebagai *DHCP client*.

4. Menghapus pengaturan *DHCP Client* pada *interface ether1* dengan mengeksekusi perintah **ip dhcp-client remove 0**.

```
[admin@R_Production] > ip dhcp-client remove 0
```

5. Memverifikasi penghapusan *interface ether1* sebagai *DHCP Client* dengan mengeksekusi perintah **ip dhcp-client print**.

```
[admin@R_Production] > ip dhcp-client print
```

6. Mengatur pengalamatan IP pada *interface ether1* dengan mengeksekusi perintah **ip address add address=192.168.11.1/24 interface=ether1**.

```
[admin@R_Production] > ip address add address=192.168.11.1/24 interface=ether1
```

7. Mengatur pengalamatan IP pada *interface* **ether2** dengan mengeksekusi perintah **ip address add address=192.168.0.6/30 interface=ether2**.

```
[admin@R_Production] > ip address add address=192.168.0.6/30 interface=ether2
```

8. Memverifikasi hasil pengaturan pengalamatan IP pada *interface* dengan mengeksekusi perintah **ip address print**.

```
[admin@R_Production] > ip address print
Columns: ADDRESS, NETWORK, INTERFACE
# ADDRESS          NETWORK           INTERFACE
0 192.168.11.1/24    192.168.11.0      ether1
1 192.168.0.6/30     192.168.0.4       ether2
```

9. Menambahkan **static route** agar dapat menjangkau alamat *network* dari LAN **R_Development** yaitu **192.168.0.8/29** melalui *router* **R_Development** dengan mengeksekusi perintah **ip route add dst-address=192.168.0.8/29 gateway=192.168.0.5**.

```
[admin@R_Production] > ip route add dst-address=192.168.0.8/29 gateway=192.168.0.5
```

10. Menampilkan informasi tabel *routing* dengan mengeksekusi perintah **ip route print**.

```
[admin@R_Production] > ip route print
Flags: D - DYNAMIC; A - ACTIVE; c, s, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
#   DST-ADDRESS      GATEWAY          DISTANCE
0   DAc 192.168.0.4/30 ether2            0
0   As  192.168.0.8/29 192.168.0.5      1
0   DAc 192.168.11.0/24 ether1            0
```

11. Memverifikasi koneksi ke *router* **R_Development** dengan mengeksekusi perintah **ping 192.168.0.5**.

```
[admin@R_Production] > ping 192.168.0.5
SEQ HOST                                SIZE TTL TIME          STATUS
0 192.168.0.5                          56  64 914us
1 192.168.0.5                          56  64 3ms582us
sent=2 received=2 packet-loss=0% min-rtt=914us avg-rtt=2ms248us
max-rtt=3ms582us
```

Terlihat verifikasi koneksi ke *router* **R_Development** berhasil dilakukan. Tekan **CTRL+C** untuk menghentikan *ping*.

F. Konfigurasi Dasar Switch SW_Production

Adapun langkah-langkah konfigurasi dasar pada *switch* **SW_Production** adalah sebagai berikut:

1. Mengakses *console* dari *node* dengan cara memilih (klik) **SW_Production** pada halaman *topology*. Langkah selanjutnya serupa dengan yang telah dijelaskan pada bagian A sebelumnya.
2. Mengatur nama pengenalan dari perangkat atau identitas dari sistem dengan mengeksekusi perintah **system identity set name=SW_Production**.

```
[admin@MikroTik] > system identity set name=SW_Production
```

3. Menampilkan informasi *interface* dengan pengaturan sebagai DHCP Client dengan mengeksekusi perintah **ip dhcp-client print**.

```
[admin@SW_Production] > ip dhcp-client print
Columns: INTERFACE, USE-PEER-DNS, ADD-DEFAULT-ROUTE, STATUS
# INTERFACE  USE-PEER-DNS  ADD-DEFAULT-ROUTE  STATUS
0 ether1      yes            yes                 searching...
```

Terlihat *interface* **ether1** diatur sebagai *DHCP client*.

4. Menghapus pengaturan *DHCP Client* pada *interface* **ether1** dengan mengeksekusi perintah **ip dhcp-client remove 0**.

```
[admin@SW_Production] > ip dhcp-client remove 0
```

5. Memverifikasi penghapusan *interface* **ether1** sebagai *DHCP Client* dengan mengeksekusi perintah **ip dhcp-client print**.

```
[admin@SW_Production] > ip dhcp-client print
```

6. Membuat *interface* **bridge** bernama **BR_Production** dengan mengeksekusi perintah **interface bridge add name=BR_Production**.

```
[admin@SW_Production] > interface bridge add name=BR_Production
```

7. Memverifikasi hasil pembuatan *interface* *bridge* dengan mengeksekusi perintah **interface bridge print**.

```
[admin@SW_Production] > interface bridge print
Flags: X - disabled, R - running
0 R name="BR_Production" mtu=auto actual-mtu=1500 l2mtu=65535 arp=enabled
  arp-timeout=auto mac-address=66:C5:70:B1:6F:3F protocol-mode=rstp
  fast-forward=yes igmp-snooping=no auto-mac=yes ageing-time=5m
  priority=0x8000 max-message-age=20s forward-delay=15s
  transmit-hold-count=6 vlan-filtering=no dhcp-snooping=no
```

8. Menambahkan *interface ether7* sebagai **port** dari **bridge BR_Production** dengan mengeksekusi perintah `interface bridge port add interface=ether7 bridge=BR_Production`.

```
[admin@SW_Production] > interface bridge port add interface=ether7 bridge=BR_Production
```

9. Memverifikasi hasil penambahan *interface bridge port* dengan mengeksekusi perintah `interface bridge port print`.

```
[admin@SW_Production] > interface bridge port print
Columns: INTERFACE, BRIDGE, HW, PVID, PRIORITY, PATH-COST, INTERNAL-PATH-COST, HORIZON
# INTERFACE BRIDGE HW PVID PRIORITY PATH-COST IN HORIZON
0 ether7 BR_Production yes 1 0x80 10 10 none
```

10. Mengatur pengalamatan IP pada *interface ether1* dengan mengeksekusi perintah `ip address add address=192.168.11.2/24 interface=ether1`.

```
[admin@SW_Production] > ip address add address=192.168.11.2/24 interface=BR_Production
```

11. Memverifikasi hasil pengaturan pengalamatan IP pada *interface* dengan mengeksekusi perintah `ip address print`.

```
[admin@SW_Production] > ip address print
Columns: ADDRESS, NETWORK, INTERFACE
# ADDRESS NETWORK INTERFACE
0 192.168.11.2/24 192.168.11.0 BR_Production
```

12. Menambahkan **default route** agar dapat menjangkau alamat *network* yang tidak terhubung langsung melalui *router R_Production* dengan mengeksekusi perintah `ip route add gateway=192.168.11.1`.

```
[admin@SW_Production] > ip route add gateway=192.168.11.1
```

13. Menampilkan informasi tabel *routing* dengan mengeksekusi perintah `ip route print`.

```
[admin@SW_Production] > ip route print
Flags: D - DYNAMIC; A - ACTIVE; c, s, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
#      DST-ADDRESS      GATEWAY      DISTANCE
0  As 0.0.0.0/0          192.168.11.1      1
    DAc 192.168.11.0/24  BR_Production      0
```

14. Memverifikasi koneksi ke *router R_Production* dengan mengeksekusi perintah **ping** 192.168.11.1.

```
[admin@SW_Production] > ping 192.168.11.1
SEQ HOST                                SIZE TTL TIME                STATUS
0 192.168.11.1                          56  64 1ms928us
1 192.168.11.1                          56  64 3ms160us
sent=2 received=2 packet-loss=0% min-rtt=1ms928us avg-rtt=2ms544us
max-rtt=3ms160us
```

Terlihat verifikasi koneksi ke *router R_Production* berhasil dilakukan. Tekan **CTRL+C** untuk menghentikan *ping*.

15. Memverifikasi koneksi ke *router R_Development* dengan mengeksekusi perintah **ping** 192.168.0.5.

```
[admin@SW_Production] > ping 192.168.0.5
SEQ HOST                                SIZE TTL TIME                STATUS
0 192.168.0.5                          56  63 5ms947us
1 192.168.0.5                          56  63 7ms827us
sent=2 received=2 packet-loss=0% min-rtt=5ms947us avg-rtt=6ms887us
max-rtt=7ms827us
```

Terlihat verifikasi koneksi ke *router R_Development* berhasil dilakukan. Tekan **CTRL+C** untuk menghentikan *ping*.

BAB 5

KONFIGURASI NAT, VLAN DAN DHCP SERVER PADA LAB NETDEVOPS

Sebelum dapat mengotomatisasi *node-node* pada lab *NetDevOps* menggunakan **Ansible** maka diperlukan pengetahuan terkait bagaimana melakukan konfigurasi secara manual meliputi **DHCP Client** dan **Network Address Translation (NAT)** agar dapat berbagi koneksi *Internet* di *router* serta **Virtual Local Area Network (VLAN)** baik di *router* maupun *switch*. Selain itu mengkonfigurasi **DHCP Server** di *router* sehingga dapat mendistribusikan pengalamatan IP secara dinamis ke *client* pada setiap VLAN. Ujicoba konfigurasi secara manual ini dilakukan pada lingkungan **Test** atau **Test Network** dari lab *NetDevOps* yang melibatkan *node R_Test* dan *SW_Test* serta 6 (enam) **VPCS** yaitu **MKT_Test1**, **MKT_Test2**, **HRD_Test1**, **HRD_Test2**, **SLS_Test1**, dan **SLS_Test2**.

A. Konfigurasi DHCP Client, NAT, VLAN Dan DHCP Server Secara Manual Pada Router R_Test

Adapun langkah-langkah konfigurasi **DHCP Client**, **NAT** dan **DHCP Server** secara manual pada *router R_Test* adalah sebagai berikut:

1. Mengakses *console* dari *node R_Test*.
2. Mengatur **DHCP Client** pada *interface ether3* yang terhubung ke *Internet* dengan komentar bernilai **Test** melalui eksekusi perintah `ip dhcp-client add interface=ether3 disabled=no comment=Test`.

```
[admin@R_Test] > ip dhcp-client add interface=ether3 disabled=no comment=Test
```

3. Memverifikasi pengaturan **DHCP Client** dengan mengeksekusi perintah `ip dhcp-client print`.

```
[admin@R_Test] > ip dhcp-client print
Columns: INTERFACE, USE-PEER-DNS, ADD-DEFAULT-ROUTE, STATUS, ADDRESS
# INTERFACE  USE-PEER-DNS  ADD-DEFAULT-ROUTE  STATUS  ADDRESS
;;; Test
0 ether3      yes            yes           bound   10.0.137.73/24
```

4. Mengatur NAT untuk berbagi pakai koneksi *Internet* dengan komentar bernilai **Test** melalui eksekusi perintah `/ip firewall nat add chain=srcnat out-interface=ether3 action=masquerade comment=Test`.

```
[admin@R_Test] > ip firewall nat add chain=srcnat out-interface=ether3 action=masquerade comment=Test
```

5. Memverifikasi hasil pengaturan NAT dengan mengeksekusi perintah `ip firewall nat print`.

```
[admin@R_Test] > ip firewall nat print
Flags: X - disabled, I - invalid; D - dynamic
0      ;;; Test
      chain=srcnat action=masquerade out-interface=ether3
```

6. Membuat *interface VLAN* pada *interface ether1* dengan nama **vlan2_mkt** dan ID **2** serta komentar bernilai **Test** melalui eksekusi perintah `/interface vlan add interface=ether1 name=vlan2_mkt vlan-id=2 comment=Test`.

```
[admin@R_Test] > interface vlan add interface=ether1 name=vlan2_mkt vlan-id=2 comment=Test
```

7. Membuat *interface VLAN* pada *interface ether1* dengan nama **vlan3_hrd** dan ID **3** serta komentar bernilai **Test** melalui eksekusi perintah `/interface vlan add interface=ether1 name=vlan3_hrd vlan-id=3 comment=Test`.

```
[admin@R_Test] > interface vlan add interface=ether1 name=vlan3_hrd vlan-id=3 comment=Test
```

8. Membuat *interface VLAN* pada *interface ether1* dengan nama **vlan4_sls** dan ID **4** serta komentar bernilai **Test** melalui eksekusi perintah `/interface vlan add interface=ether1 name=vlan4_sls vlan-id=4 comment=Test`.

```
[admin@R_Test] > interface vlan add interface=ether1 name=vlan4_sls vlan-id=4 comment=Test
```

9. Memverifikasi hasil pembuatan *interface VLAN* dengan mengeksekusi perintah `interface vlan print`.

```
[admin@R_Test] > interface vlan print
Flags: R - RUNNING
Columns: NAME, MTU, ARP, VLAN-ID, INTERFACE
#  NAME      MTU  ARP      VLAN-ID  INTERFACE
;;; Test
0 R vlan2_mkt 1500 enabled    2  ether1
;;; Test
1 R vlan3_hrd 1500 enabled    3  ether1
;;; Test
2 R vlan4_sls 1500 enabled    4  ether1
```

10. Mengatur pengalamatan IP pada *interface* **vlan2** bernilai **192.168.2.1/24** dengan mengeksekusi perintah **ip address add address=192.168.2.1/24 interface=vlan2 comment=Test.**

```
[admin@R_Test] > ip address add address=192.168.2.1/24 interface=vlan2 comment=Test
```

11. Mengatur pengalamatan IP pada *interface* **vlan3** bernilai **192.168.3.1/24** dengan mengeksekusi perintah **ip address add address=192.168.3.1/24 interface=vlan3 comment=Test.**

```
[admin@R_Test] > ip address add address=192.168.3.1/24 interface=vlan3 comment=Test
```

12. Mengatur pengalamatan IP pada *interface* **vlan4** bernilai **192.168.4.1/24** dengan mengeksekusi perintah **ip address add address=192.168.4.1/24 interface=vlan4 comment=Test.**

```
[admin@R_Test] > ip address add address=192.168.4.1/24 interface=vlan4 comment=Test
```

13. Memverifikasi hasil pengaturan pengalamatan IP pada *interface* dengan mengeksekusi perintah **ip address print.**

```
[admin@R_Test] > ip address print
Flags: D - DYNAMIC
Columns: ADDRESS, NETWORK, INTERFACE
#  ADDRESS      NETWORK      INTERFACE
0  192.168.1.1/24 192.168.1.0  ether1
1  192.168.0.1/30 192.168.0.0  ether2
2 D 10.0.137.73/24 10.0.137.0   ether3
;;; Test
3  192.168.2.1/24 192.168.2.0  vlan2_mkt
;;; Test
4  192.168.3.1/24 192.168.3.0  vlan3_hrd
;;; Test
5  192.168.4.1/24 192.168.4.0  vlan4_sls
```

14. Menambahkan **IP Pool** untuk menentukan rentang alamat IP yang akan didistribusikan secara dinamis ke *client* **VLAN MKT** yaitu dari **192.168.2.2** sampai dengan **192.168.2.254** dengan nama **pool_vlan2_mkt** dan komentar bernilai **Test** melalui eksekusi perintah **ip pool add name=pool_vlan2_mkt ranges=192.168.2.2-192.168.2.254 comment=Test**.

```
[admin@R_Test] > ip pool add name=pool_vlan2_mkt ranges=192.168.2.2-192.168.2.254 comment=Test
```

15. Menambahkan **IP Pool** untuk menentukan rentang alamat IP yang akan didistribusikan secara dinamis ke *client* **VLAN HRD** yaitu dari **192.168.3.2** sampai dengan **192.168.3.254** dengan nama **pool_vlan3_hrd** dan komentar bernilai **Test** melalui eksekusi perintah **ip pool add name=pool_vlan3_hrd ranges=192.168.3.2-192.168.3.254 comment=Test**.

```
[admin@R_Test] > ip pool add name=pool_vlan3_hrd ranges=192.168.3.2-192.168.3.254 comment=Test
```

16. Menambahkan **IP Pool** untuk menentukan rentang alamat IP yang akan didistribusikan secara dinamis ke *client* **VLAN SLS** yaitu dari **192.168.4.2** sampai dengan **192.168.4.254** dengan nama **pool_vlan4_sls** dan komentar bernilai **Test** melalui eksekusi perintah **ip pool add name=pool_vlan3_sls ranges=192.168.4.2-192.168.4.254 comment=Test**.

```
[admin@R_Test] > ip pool add name=pool_vlan4_sls ranges=192.168.4.2-192.168.4.254 comment=Test
```

17. Memverifikasi hasil penambahan **IP Pool** dengan mengeksekusi perintah **ip pool print**.

```
[admin@R_Test] > ip pool print
Columns: NAME, RANGES
#  NAME          RANGES
;;; Test
0  pool_vlan2_mkt 192.168.2.2-192.168.2.254
;;; Test
1  pool_vlan3_hrd 192.168.3.2-192.168.3.254
;;; Test
2  pool_vlan4_sls 192.168.4.2-192.168.4.254
```

18. Menambahkan **IP DHCP-Server Network** untuk menentukan alamat **network** yang IP-nya didistribusikan secara dinamis ke *client* **VLAN MKT** yaitu **192.168.2.0/24**. Termasuk

parameter **Transmission Control Protocol/Internet Protocol (TCP/IP)** meliputi alamat **IP Server DNS 8.8.8.8 dan 8.8.4.4**, **gateway 192.168.2.1** serta komentar bernilai **Test** dengan mengeksekusi perintah `ip dhcp-server network add address=192.168.2.0/24 dns-server=8.8.8.8,8.8.4.4 gateway=192.168.2.1 comment=Test.`

```
[admin@R_Test] > ip dhcp-server network add address=192.168.2.0/24 dns-server=8.8.8.8,8.8.4.4 gateway=192.168.2.1 comment=Test
```

19. Menambahkan **IP DHCP-Server Network** untuk menentukan alamat **network** yang IP-nya didistribusikan secara dinamis ke *client* **VLAN HRD** yaitu **192.168.3.0/24**. Termasuk parameter **Transmission Control Protocol/Internet Protocol (TCP/IP)** meliputi alamat **IP Server DNS 8.8.8.8 dan 8.8.4.4**, **gateway 192.168.3.1** serta komentar bernilai **Test** dengan mengeksekusi perintah `ip dhcp-server network add address=192.168.3.0/24 dns-server=8.8.8.8,8.8.4.4 gateway=192.168.3.1 comment=Test.`

```
[admin@R_Test] > ip dhcp-server network add address=192.168.3.0/24 dns-server=8.8.8.8,8.8.4.4 gateway=192.168.3.1 comment=Test
```

20. Menambahkan **IP DHCP-Server Network** untuk menentukan alamat **network** yang IP-nya didistribusikan secara dinamis ke *client* **VLAN SLS** yaitu **192.168.4.0/24**. Termasuk parameter **Transmission Control Protocol/Internet Protocol (TCP/IP)** meliputi alamat **IP Server DNS 8.8.8.8 dan 8.8.4.4**, **gateway 192.168.4.1** serta komentar bernilai **Test** dengan mengeksekusi perintah `ip dhcp-server network add address=192.168.4.0/24 dns-server=8.8.8.8,8.8.4.4 gateway=192.168.4.1 comment=Test.`

```
[admin@R_Test] > ip dhcp-server network add address=192.168.4.0/24 dns-server=8.8.8.8,8.8.4.4 gateway=192.168.4.1 comment=Test
```

21. Memverifikasi hasil penambahan **IP DHCP-Server Network** dengan mengeksekusi perintah `ip dhcp-server network print.`

```
[admin@R_Test] > ip dhcp-server network print
Columns: ADDRESS, GATEWAY, DNS-SERVER
# ADDRESS          GATEWAY          DNS-SERVER
;;; Test
0 192.168.2.0/24    192.168.2.1      8.8.8.8
                        8.8.4.4

;;; Test
1 192.168.3.0/24    192.168.3.1      8.8.8.8
                        8.8.4.4

;;; Test
2 192.168.4.0/24    192.168.4.1      8.8.8.8
                        8.8.4.4
```

22. Menambahkan **IP DHCP-Server** untuk **VLAN MKT** dengan menentukan nama dari DHCP Server yaitu **dhcp_vlan2_mkt** dan **IP Pool** yang diterapkan pada *interface* **vlan2_mkt** yaitu **pool_vlan2_mkt** serta menentukan masa sewa (**lease**) selama 3 jam (**3h** dimana **h=hour**). Selain itu juga komentar bernilai **Test** dengan mengeksekusi perintah **ip dhcp-server add address-pool=pool_vlan2_mkt interface=vlan2_mkt lease-time=3h name=dhcp_vlan2_mkt comment=Test**.

```
[admin@R_Test] > ip dhcp-server add address-pool=pool_vlan2_mkt interface=vlan2_mkt lease-time=3h name=dhcp_vlan2_mkt comment=Test
```

23. Menambahkan **IP DHCP-Server** untuk **VLAN HRD** dengan menentukan nama dari DHCP Server yaitu **dhcp_vlan3_hrd** dan **IP Pool** yang diterapkan pada *interface* **vlan3_hrd** yaitu **pool_vlan3_hrd** serta menentukan masa sewa (**lease**) selama 3 jam (**3h** dimana **h=hour**). Selain itu juga komentar bernilai **Test** dengan mengeksekusi perintah **ip dhcp-server add address-pool=pool_vlan3_hrd interface=vlan3_hrd lease-time=3h name=dhcp_vlan3_hrd comment=Test**.

```
[admin@R_Test] > ip dhcp-server add address-pool=pool_vlan3_hrd interface=vlan3_hrd lease-time=3h name=dhcp_vlan3_hrd comment=Test
```

24. Menambahkan **IP DHCP-Server** untuk **VLAN SLS** dengan menentukan nama dari DHCP Server yaitu **dhcp_vlan4_sls** dan **IP Pool** yang diterapkan pada *interface* **vlan4_sls** yaitu **pool_vlan4_sls** serta menentukan masa sewa (**lease**) selama 3 jam (**3h** dimana **h=hour**). Selain itu juga komentar bernilai **Test** dengan mengeksekusi perintah **ip dhcp-server add address-pool=pool_vlan4_sls interface=vlan4_sls lease-time=3h name=dhcp_vlan4_sls comment=Test**.

```
[admin@R_Test] > ip dhcp-server add address-pool=pool_vlan4_sls interface=vlan4_sls lease-time=3h name=dhcp_vlan4_sls comment=Test
```

25. Memverifikasi hasil penambahan IP DHCP-Server dengan mengeksekusi perintah `ip dhcp-server print`.

```
[admin@R_Test] > ip dhcp-server print
Columns: NAME, INTERFACE, ADDRESS-POOL, LEASE-TIME
# NAME          INTERFACE ADDRESS-POOL  LEASE-TIME
;;; Test
0 dhcp_vlan2_mkt vlan2_mkt    pool_vlan2_mkt 3h
;;; Test
1 dhcp_vlan3_hrd vlan3_hrd    pool_vlan3_hrd 3h
;;; Test
2 dhcp_vlan4_sls vlan4_sls    pool_vlan4_sls 3h
```

B. Konfigurasi Bridge Port Dan Bridge VLAN Serta VLAN Filtering Secara Manual Pada Switch SW_Test

Adapun langkah-langkah konfigurasi **bridge port** dan **bridge VLAN** serta **VLAN Filtering** secara manual pada switch **SW_Test** adalah sebagai berikut:

1. Mengakses *console* dari *node SW_Test*.
2. Menambahkan *interface ether1* sebagai **bridge port** dari **bridge BR_Test** dan mengatur **Port VLAN ID (PVID)** dari *interface* tersebut dengan nilai **2**. Hal ini dilakukan agar *interface ether1* menjadi **access port** atau anggota dari **VLAN MKT** dengan mengeksekusi perintah `interface bridge port add interface=ether1 bridge=BR_Test pvid=2 comment=Test`.

```
[admin@SW_Test] > interface bridge port add bridge=BR_Test interface=ether1 pvid=2 comment=Test
```

3. Menambahkan *interface ether2* sebagai **bridge port** dari **bridge BR_Test** dan mengatur **Port VLAN ID (PVID)** dari *interface* tersebut dengan nilai **2**. Hal ini dilakukan agar *interface ether2* menjadi **access port** atau anggota dari **VLAN MKT** dengan mengeksekusi perintah `interface bridge port add interface=ether2 bridge=BR_Test pvid=2 comment=Test`.

```
[admin@SW_Test] > interface bridge port add bridge=BR_Test interface=ether2 pvid=2 comment=Test
```

- Menambahkan *interface ether3* sebagai **bridge port** dari **bridge BR_Test** dan mengatur **Port VLAN ID (PVID)** dari *interface* tersebut dengan nilai **3**. Hal ini dilakukan agar *interface ether3* menjadi **access port** atau anggota dari **VLAN HRD** dengan mengeksekusi perintah `interface bridge port add interface=ether3 bridge=BR_Test pvid=3 comment=Test`.

```
[admin@SW_Test] > interface bridge port add bridge=BR_Test interface=ether3 pvid=3 comment=Test
```

- Menambahkan *interface ether4* sebagai **bridge port** dari **bridge BR_Test** dan mengatur **Port VLAN ID (PVID)** dari *interface* tersebut dengan nilai **3**. Hal ini dilakukan agar *interface ether4* menjadi **access port** atau anggota dari **VLAN HRD** dengan mengeksekusi perintah `interface bridge port add interface=ether4 bridge=BR_Test pvid=3 comment=Test`.

```
[admin@SW_Test] > interface bridge port add bridge=BR_Test interface=ether4 pvid=3 comment=Test
```

- Menambahkan *interface ether5* sebagai **bridge port** dari **bridge BR_Test** dan mengatur **Port VLAN ID (PVID)** dari *interface* tersebut dengan nilai **4**. Hal ini dilakukan agar *interface ether5* menjadi **access port** atau anggota dari **VLAN SLS** dengan mengeksekusi perintah `interface bridge port add interface=ether5 bridge=BR_Test pvid=4 comment=Test`.

```
[admin@SW_Test] > interface bridge port add bridge=BR_Test interface=ether5 pvid=4 comment=Test
```

- Menambahkan *interface ether6* sebagai **bridge port** dari **bridge BR_Test** dan mengatur **Port VLAN ID (PVID)** dari *interface* tersebut dengan nilai **4**. Hal ini dilakukan agar *interface ether6* menjadi **access port** atau anggota dari **VLAN SLS** dengan mengeksekusi perintah `interface bridge port add interface=ether6 bridge=BR_Test pvid=4 comment=Test`.

```
[admin@SW_Test] > interface bridge port add bridge=BR_Test interface=ether6 pvid=4 comment=Test
```

- Memverifikasi hasil penambahan *interface bridge port* dengan mengeksekusi perintah `interface bridge port print`.

```
[admin@SW_Test] > interface bridge port print
Columns: INTERFACE, BRIDGE, HW, PVID, PRIORITY, PATH-COST, INTERNAL-PATH-COST, H
ORIZON
# INTERFACE BRIDGE HW PVID PRIORITY PATH-COST IN HORIZON
0 ether7 BR_Test yes 1 0x80 10 10 none
;;; Test
1 ether1 BR_Test yes 2 0x80 10 10 none
;;; Test
2 ether2 BR_Test yes 2 0x80 10 10 none
;;; Test
3 ether3 BR_Test yes 3 0x80 10 10 none
;;; Test
4 ether4 BR_Test yes 3 0x80 10 10 none
;;; Test
5 ether5 BR_Test yes 4 0x80 10 10 none
;;; Test
6 ether6 BR_Test yes 4 0x80 10 10 none
```

9. Menambahkan **bridge VLAN** dan menentukan port yang ditandai (**tagged**) serta tidak ditandai (**untagged**) dengan mengeksekusi perintah. **Tagged port** mengirim *frame* keluar dengan penandaan **VLAN ID** yang dipelajari. Sebaliknya **untagged port** menghapus penandaan VLAN sebelum mengirim *frame* keluar jika **VLAN ID** yang dipelajari cocok dengan **PVID** dari *port*. Terdapat 3 (tiga) entri **bridge VLAN** yang akan ditambahkan yaitu:

- a. Pada **bridge BR_Test** dengan **tagged port ether7**, **untagged port ether1** dan **ether2**, **vlan-ids 2** dan **komentar** bernilai **Test** melalui eksekusi perintah `interface bridge vlan add bridge=BR_Test tagged=ether7 untagged=ether1,ether2 vlan-ids=2 comment=Test`.

```
[admin@SW_Test] > interface bridge vlan add bridge=BR_Test tagged=ether7 untagge
d=ether1,ether2 vlan-ids=2 comment=Test
```

- b. Pada **bridge BR_Test** dengan **tagged port ether7**, **untagged port ether3** dan **ether4**, **vlan-ids 3** dan **komentar** bernilai **Test** melalui eksekusi perintah `interface bridge vlan add bridge=BR_Test tagged=ether7 untagged=ether3,ether4 vlan-ids=3 comment=Test`.

```
[admin@SW_Test] > interface bridge vlan add bridge=BR_Test tagged=ether7 untagge
d=ether3,ether4 vlan-ids=3 comment=Test
```

- c. Pada **bridge BR_Test** dengan **tagged port ether7**, **untagged port ether5** dan **ether6**, **vlan-ids 4** dan **komentar** bernilai **Test** melalui eksekusi perintah **interface bridge vlan add bridge=BR_Test tagged=ether7 untagged=ether5,ether6 vlan-ids=4 comment=Test**.

```
[admin@SW_Test] > interface bridge vlan add bridge=BR_Test tagged=ether7 untagged=ether5,ether6 vlan-ids=4 comment=Test
```

10. Memverifikasi hasil penambahan **bridge VLAN** dengan mengeksekusi perintah **interface bridge vlan print**.

```
[admin@SW_Test] > interface bridge vlan print
Columns: BRIDGE, VLAN-IDS
# BRIDGE    VLAN-IDS
;;; Test
0 BR_Test    2
;;; Test
1 BR_Test    3
;;; Test
2 BR_Test    4
```

11. Mengaktifkan **VLAN Filtering** pada *interface bridge BR_Test* dengan mengeksekusi perintah **interface bridge set vlan-filtering=yes BR_Test**.

```
[admin@SW_Test] > interface bridge set vlan-filtering=yes BR_Test
```

12. Memverifikasi hasil pengaktifan **VLAN Filtering** pada *interface bridge BR_Test* dengan mengeksekusi perintah **interface bridge print**.

```
[admin@SW_Test] > interface bridge print
Flags: X - disabled, R - running
0 R name="BR_Test" mtu=auto actual-mtu=1500 l2mtu=65535 arp=enabled
  arp-timeout=auto mac-address=50:B2:3D:00:02:06 protocol-mode=rstp
  fast-forward=yes igmp-snooping=no auto-mac=yes ageing-time=5m
  priority=0x8000 max-message-age=20s forward-delay=15s
  transmit-hold-count=6 vlan-filtering=yes ether-type=0x8100 pvid=1
  frame-types=admit-all ingress-filtering=yes dhcp-snooping=no
```

C. Konfigurasi DHCP Client Dan Verifikasi Koneksi Antar VLAN Serta Internet Pada VPCS Network TEST

Adapun langkah-langkah konfigurasi **DHCP Client** dan verifikasi koneksi antar VLAN serta *Internet* pada **VPCS NETWORK TEST** adalah sebagai berikut:

1. Mengakses *console* dari *node VPCS*, sebagai contoh **MKT1_Test**.
2. Pada kotak dialog **MKT1_Test** yang tampil, lakukan permintaan **DHCP** dengan mengeksekusi perintah **ip dhcp -r** pada command prompt dari **VPCS>**, seperti yang ditunjukkan pada gambar berikut:

 MKT1_Test

```
VPCS> ip dhcp -r
DORA IP 192.168.2.254/24 GW 192.168.2.1
```

Terlihat **MKT1_Test** berhasil memperoleh pengalamatan **IP** dari **DHCP Server** dengan nilai **192.168.2.254/24** dan **gateway 192.168.2.1**.

3. Menampilkan detail informasi pengalamatan IP dan parameter TCP/IP lainnya yang diperoleh oleh **VPCS MKT1_Test** dari **DHCP Server** dengan mengeksekusi perintah **show ip**.

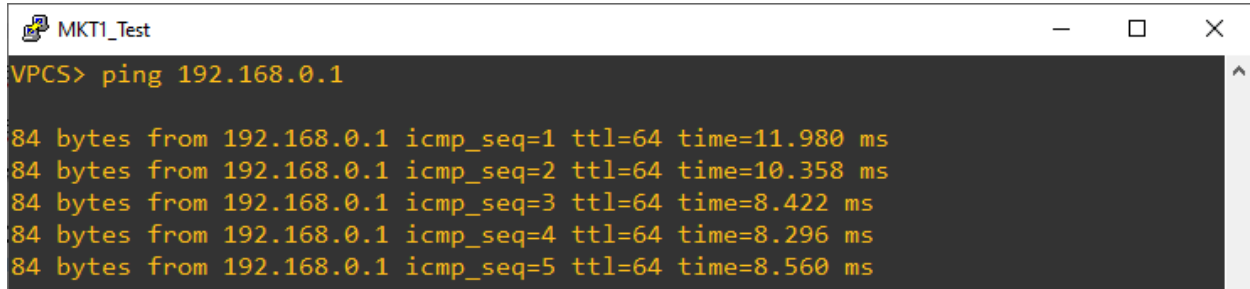
 MKT1_Test

```
VPCS> show ip

NAME       : VPCS[1]
IP/MASK    : 192.168.2.254/24
GATEWAY    : 192.168.2.1
DNS        : 8.8.8.8  8.8.4.4
DHCP SERVER : 192.168.2.1
DHCP LEASE  : 10794, 10800/5400/9450
MAC        : 00:50:79:66:68:06
LPORT      : 20000
RHOST:PORT  : 127.0.0.1:30000
MTU        : 1500
```

Terlihat informasi terkait alamat IP dari **Server DNS**, **DHCP Server** yang mengalokasikan pengalamatan IP dan masa sewa (**lease**) dari DHCP serta alamat **MAC** dari **VPCS**.

4. Memverifikasi koneksi ke beda *network* menggunakan utilitas **ping**, sebagai contoh ke alamat IP dari *interface router R_Test* yang terhubung ke *router R_Development* yaitu dengan alamat IP **192.168.0.1**. Eksekusi perintah **ping 192.168.0.1**.



```

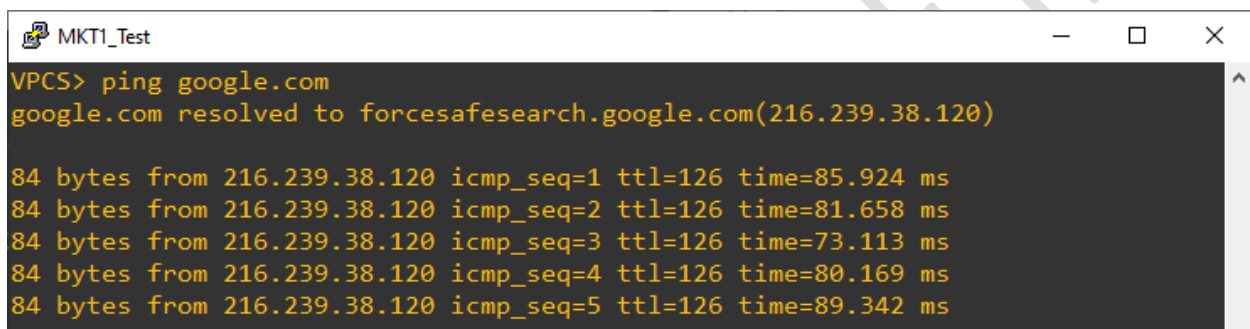
MKT1_Test
VPCS> ping 192.168.0.1

84 bytes from 192.168.0.1 icmp_seq=1 ttl=64 time=11.980 ms
84 bytes from 192.168.0.1 icmp_seq=2 ttl=64 time=10.358 ms
84 bytes from 192.168.0.1 icmp_seq=3 ttl=64 time=8.422 ms
84 bytes from 192.168.0.1 icmp_seq=4 ttl=64 time=8.296 ms
84 bytes from 192.168.0.1 icmp_seq=5 ttl=64 time=8.560 ms

```

Terlihat koneksi berhasil dilakukan.

- Memverifikasi koneksi ke *Internet* sebagai contoh menggunakan utilitas **ping** ke **google.com** dengan mengeksekusi perintah **ping google.com**.



```

MKT1_Test
VPCS> ping google.com
google.com resolved to forcesafesearch.google.com(216.239.38.120)

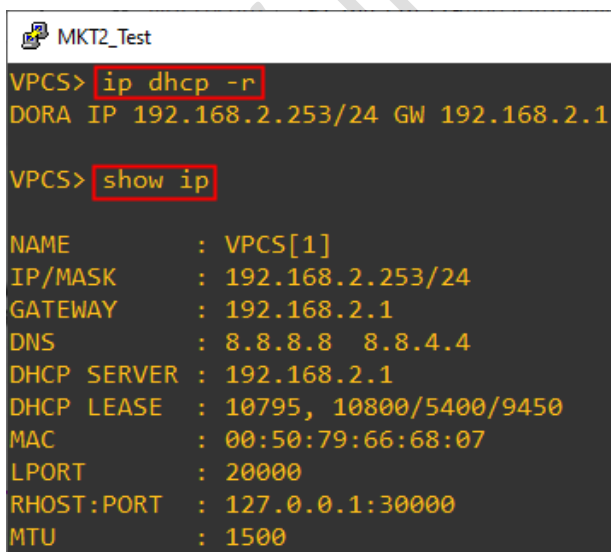
84 bytes from 216.239.38.120 icmp_seq=1 ttl=126 time=85.924 ms
84 bytes from 216.239.38.120 icmp_seq=2 ttl=126 time=81.658 ms
84 bytes from 216.239.38.120 icmp_seq=3 ttl=126 time=73.113 ms
84 bytes from 216.239.38.120 icmp_seq=4 ttl=126 time=80.169 ms
84 bytes from 216.239.38.120 icmp_seq=5 ttl=126 time=89.342 ms

```

Terlihat koneksi *Internet* berhasil dilakukan.

Dengan cara yang sama lakukan **DHCP request** pada **VPCS MKT2_Test**, **HRD1_Test**, **HRD2_Test**, **SLS1_Test** dan **SLS2_Test**.

- Hasil **DHCP request** menggunakan perintah **ip dhcp -r** dan verifikasi pengalamatan IP menggunakan perintah **show ip** pada **VPCS MKT2_Test**.



```

MKT2_Test
VPCS> ip dhcp -r
DORA IP 192.168.2.253/24 GW 192.168.2.1

VPCS> show ip

NAME       : VPCS[1]
IP/MASK    : 192.168.2.253/24
GATEWAY    : 192.168.2.1
DNS        : 8.8.8.8 8.8.4.4
DHCP SERVER : 192.168.2.1
DHCP LEASE  : 10795, 10800/5400/9450
MAC        : 00:50:79:66:68:07
LPORT      : 20000
RHOST:PORT  : 127.0.0.1:30000
MTU        : 1500

```

7. Hasil **DHCP request** menggunakan perintah `ip dhcp -r` dan verifikasi pengalamatan IP menggunakan perintah `show ip` pada **VPCS HRD1_Test**.

```
HRD1_Test
VPCS> ip dhcp -r
DORA IP 192.168.3.254/24 GW 192.168.3.1

VPCS> show ip

NAME       : VPCS[1]
IP/MASK    : 192.168.3.254/24
GATEWAY    : 192.168.3.1
DNS        : 8.8.8.8 8.8.4.4
DHCP SERVER : 192.168.3.1
DHCP LEASE  : 10797, 10800/5400/9450
MAC        : 00:50:79:66:68:08
LPORT      : 20000
RHOST:PORT  : 127.0.0.1:30000
MTU        : 1500
```

8. Hasil **DHCP request** menggunakan perintah `ip dhcp -r` dan verifikasi pengalamatan IP menggunakan perintah `show ip` pada **VPCS HRD2_Test**.

```
HRD2_Test
VPCS> ip dhcp -r
DORA IP 192.168.3.253/24 GW 192.168.3.1

VPCS> show ip

NAME       : VPCS[1]
IP/MASK    : 192.168.3.253/24
GATEWAY    : 192.168.3.1
DNS        : 8.8.8.8 8.8.4.4
DHCP SERVER : 192.168.3.1
DHCP LEASE  : 10798, 10800/5400/9450
MAC        : 00:50:79:66:68:09
LPORT      : 20000
RHOST:PORT  : 127.0.0.1:30000
MTU        : 1500
```

9. Hasil **DHCP request** menggunakan perintah `ip dhcp -r` dan verifikasi pengalamatan IP menggunakan perintah `show ip` pada **VPCS SLS1_Test**.

```

SLS1_Test
VPCS> ip dhcp -r
DORA IP 192.168.4.254/24 GW 192.168.4.1

VPCS> show ip

NAME       : VPCS[1]
IP/MASK    : 192.168.4.254/24
GATEWAY    : 192.168.4.1
DNS        : 8.8.8.8 8.8.4.4
DHCP SERVER : 192.168.4.1
DHCP LEASE  : 10797, 10800/5400/9450
MAC        : 00:50:79:66:68:0a
LPORT      : 20000
RHOST:PORT  : 127.0.0.1:30000
MTU        : 1500

```

10. Hasil **DHCP request** menggunakan perintah **ip dhcp -r** dan verifikasi pengalamatan IP menggunakan perintah **show ip** pada **VPCS SLS2_Test**.

```

SLS2_Test
VPCS> ip dhcp -r
DORA IP 192.168.4.253/24 GW 192.168.4.1

VPCS> show ip

NAME       : VPCS[1]
IP/MASK    : 192.168.4.253/24
GATEWAY    : 192.168.4.1
DNS        : 8.8.8.8 8.8.4.4
DHCP SERVER : 192.168.4.1
DHCP LEASE  : 10798, 10800/5400/9450
MAC        : 00:50:79:66:68:0b
LPORT      : 20000
RHOST:PORT  : 127.0.0.1:30000
MTU        : 1500

```

11. Memverifikasi koneksi dari **VPCS SLS2_Test** ke **MKT1_Test** dengan mengeksekusi perintah **ping 192.168.2.254**.

```

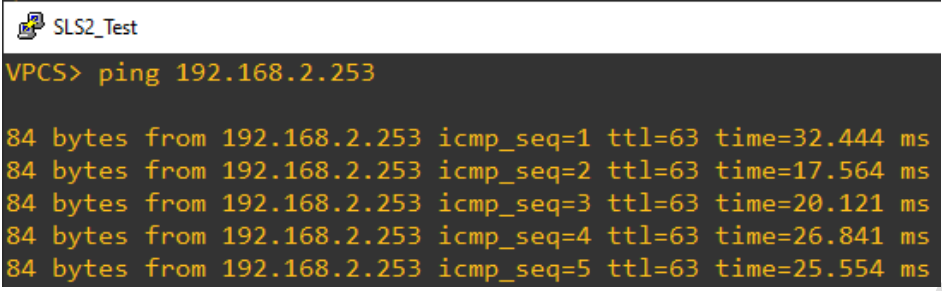
SLS2_Test
VPCS> ping 192.168.2.254

84 bytes from 192.168.2.254 icmp_seq=1 ttl=63 time=29.296 ms
84 bytes from 192.168.2.254 icmp_seq=2 ttl=63 time=20.835 ms
84 bytes from 192.168.2.254 icmp_seq=3 ttl=63 time=18.966 ms
84 bytes from 192.168.2.254 icmp_seq=4 ttl=63 time=28.365 ms
84 bytes from 192.168.2.254 icmp_seq=5 ttl=63 time=38.517 ms

```

Terlihat koneksi berhasil dilakukan.

12. Memverifikasi koneksi dari **VPCS SLS2_Test** ke **MKT2_Test** dengan mengeksekusi perintah **ping 192.168.2.253**.

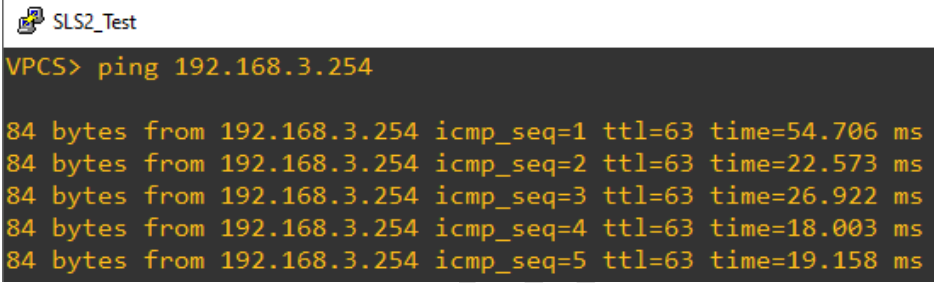


```
VPCS> ping 192.168.2.253

84 bytes from 192.168.2.253 icmp_seq=1 ttl=63 time=32.444 ms
84 bytes from 192.168.2.253 icmp_seq=2 ttl=63 time=17.564 ms
84 bytes from 192.168.2.253 icmp_seq=3 ttl=63 time=20.121 ms
84 bytes from 192.168.2.253 icmp_seq=4 ttl=63 time=26.841 ms
84 bytes from 192.168.2.253 icmp_seq=5 ttl=63 time=25.554 ms
```

Terlihat koneksi berhasil dilakukan.

13. Memverifikasi koneksi dari **VPCS SLS2_Test** ke **HRD1_Test** dengan mengeksekusi perintah **ping 192.168.3.254**.

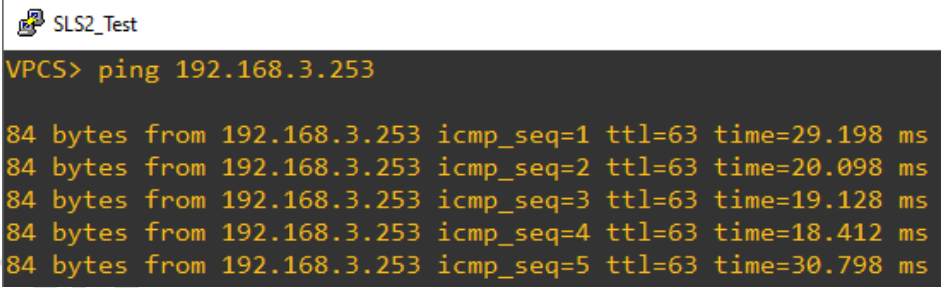


```
VPCS> ping 192.168.3.254

84 bytes from 192.168.3.254 icmp_seq=1 ttl=63 time=54.706 ms
84 bytes from 192.168.3.254 icmp_seq=2 ttl=63 time=22.573 ms
84 bytes from 192.168.3.254 icmp_seq=3 ttl=63 time=26.922 ms
84 bytes from 192.168.3.254 icmp_seq=4 ttl=63 time=18.003 ms
84 bytes from 192.168.3.254 icmp_seq=5 ttl=63 time=19.158 ms
```

Terlihat koneksi berhasil dilakukan.

14. Memverifikasi koneksi dari **VPCS SLS2_Test** ke **HRD2_Test** dengan mengeksekusi perintah **ping 192.168.3.253**.



```
VPCS> ping 192.168.3.253

84 bytes from 192.168.3.253 icmp_seq=1 ttl=63 time=29.198 ms
84 bytes from 192.168.3.253 icmp_seq=2 ttl=63 time=20.098 ms
84 bytes from 192.168.3.253 icmp_seq=3 ttl=63 time=19.128 ms
84 bytes from 192.168.3.253 icmp_seq=4 ttl=63 time=18.412 ms
84 bytes from 192.168.3.253 icmp_seq=5 ttl=63 time=30.798 ms
```

Terlihat koneksi berhasil dilakukan.

15. Memverifikasi koneksi dari **VPCS SLS2_Test** ke **SLS1_Test** dengan mengeksekusi perintah **ping 192.168.4.254**.

```

SLS2_Test
VPCS> ping 192.168.4.254

84 bytes from 192.168.4.254 icmp_seq=1 ttl=64 time=7.772 ms
84 bytes from 192.168.4.254 icmp_seq=2 ttl=64 time=7.207 ms
84 bytes from 192.168.4.254 icmp_seq=3 ttl=64 time=6.735 ms
84 bytes from 192.168.4.254 icmp_seq=4 ttl=64 time=6.228 ms
84 bytes from 192.168.4.254 icmp_seq=5 ttl=64 time=7.407 ms

```

Terlihat koneksi berhasil dilakukan.

D. Memverifikasi Pengalamatan IP Yang Telah Disewakan Oleh DHCP Server Pada Router R_TEST

Adapun langkah-langkah memverifikasi pengalamatan IP yang telah disewakan oleh DHCP Server ke DHCP Client pada *router R_Test* adalah sebagai berikut:

1. Mengakses *console* dari *node R_Test*.
2. Mengeksekusi perintah `ip dhcp-server lease print`.

```

[admin@R_Test] > ip dhcp-server lease print
Flags: D, B - BLOCKED
Columns: ADDRESS, MAC-ADDRESS, HOST-NAME, SERVER, STATUS, LAST-SEEN
#  ADDRESS      MAC-ADDRESS      HOST-  SERVER      STATUS  LAST-SEEN
0  D 192.168.2.254  00:50:79:66:68:06 VPCS1  dhcp_vlan2_mkt bound   41m30s
1  D 192.168.2.253  00:50:79:66:68:07 VPCS1  dhcp_vlan2_mkt bound   14m23s
2  D 192.168.3.254  00:50:79:66:68:08 VPCS1  dhcp_vlan3_hrd bound   12m30s
3  D 192.168.3.253  00:50:79:66:68:09 VPCS1  dhcp_vlan3_hrd bound   11m31s
4  D 192.168.4.254  00:50:79:66:68:0A VPCS1  dhcp_vlan4_sls bound   10m25s
5  D 192.168.4.253  00:50:79:66:68:0B VPCS1  dhcp_vlan4_sls bound    9m20s

```

Terlihat daftar alamat IP yang telah disewakan ke setiap **client VPCS** yang terdapat pada setiap VLAN.

E. Menghapus Konfigurasi Bridge Port Dan Bridge VLAN Serta VLAN Filtering Secara Manual Pada Switch SW_Test

Adapun langkah-langkah menghapus konfigurasi **bridge port** dan **bridge VLAN** serta **VLAN Filtering** secara manual pada *switch SW_Test* adalah sebagai berikut:

1. Mengakses *console* dari *node SW_Test*.

2. Menonaktifkan VLAN Filtering pada interface bridge BR_Test dengan mengeksekusi perintah `interface bridge set vlan-filtering=no BR_Test`.

```
[admin@SW_Test] > interface bridge set vlan-filtering=no BR_Test
```

3. Memverifikasi hasil penonaktifan VLAN Filtering dengan mengeksekusi perintah `interface bridge print`.

```
[admin@SW_Test] > interface bridge print
Flags: X - disabled, R - running
0 R name="BR_Test" mtu=auto actual-mtu=1500 l2mtu=65535 arp=enabled
  arp-timeout=auto mac-address=50:B2:3D:00:02:06 protocol-mode=rstp
  fast-forward=yes igmp-snooping=no auto-mac=yes ageing-time=5m
  priority=0x8000 max-message-age=20s forward-delay=15s
  transmit-hold-count=6 vlan-filtering=no dhcp-snooping=no
```

4. Menghapus seluruh *interface* Bridge VLAN dengan komentar bernilai Test melalui eksekusi perintah `interface bridge vlan remove [find where comment=Test]`.

```
[admin@SW_Test] > interface bridge vlan remove [find where comment=Test]
```

5. Memverifikasi hasil penghapusan *interface* Bridge VLAN dengan mengeksekusi perintah `interface bridge vlan print`.

```
[admin@SW_Test] > interface bridge vlan print
```

6. Menghapus seluruh *interface* Bridge port dengan komentar bernilai Test melalui eksekusi perintah `interface bridge port remove [find where comment=Test]`.

```
[admin@SW_Test] > interface bridge port remove [find where comment=Test]
```

7. Memverifikasi hasil penghapusan *interface* Bridge port dengan mengeksekusi perintah `interface bridge port print`.

```
[admin@SW_Test] > interface bridge port print
Columns: INTERFACE, BRIDGE, HW, PVID, PRIORITY, PATH-COST, INTERNAL-PATH-COST, H
ORIZON
# INTERFACE  BRIDGE  HW  PVID  PRIORITY  PATH-COST  IN  HORIZON
0 ether7     BR_Test  yes   1   0x80           10  10  none
```

Sudah tidak terlihat *interface bridge port* dengan komentar bernilai **Test** sehingga penghapusan *bridge port* telah berhasil dilakukan.

F. Menghapus Konfigurasi DHCP Client, NAT, VLAN Dan DHCP Server Secara Manual Pada Router R_Test

Adapun langkah-langkah menghapus konfigurasi **DHCP Client**, **NAT**, **VLAN** dan **DHCP Server** secara manual pada *router R_Test* adalah sebagai berikut:

1. Mengakses *console* dari *node R_Test*.
2. Menghapus **DHCP lease** dengan mengeksekusi perintah `ip dhcp lease remove [find dynamic]`.

```
[admin@R_Test] > ip dhcp-server lease remove [find dynamic]
```

3. Memverifikasi penghapusan **DHCP lease** dengan mengeksekusi perintah `ip dhcp-server lease print`.

```
[admin@R_Test] > ip dhcp-server lease print
```

4. Menghapus pengaturan **IP DHCP-Server Network** dengan komentar bernilai **Test** melalui eksekusi perintah `ip dhcp-server network remove [find where comment=Test]`.

```
[admin@R_Test] > ip dhcp-server network remove [find where comment=Test]
```

5. Memverifikasi penghapusan **IP DHCP-Server Network** dengan mengeksekusi perintah `ip dhcp-server network print`.

```
[admin@R_Test] > ip dhcp-server network print
```

6. Menghapus pengaturan **IP DHCP-Server** dengan komentar bernilai **Test** melalui eksekusi perintah `ip dhcp-server remove [find where comment=Test]`.

```
[admin@R_Test] > ip dhcp-server remove [find where comment=Test]
```

7. Memverifikasi penghapusan **IP DHCP-Server** dengan mengeksekusi perintah **ip dhcp-server print**.

```
[admin@R_Test] > ip dhcp-server print
```

8. Menghapus pengaturan **IP Pool** dengan komentar bernilai **Test** melalui eksekusi perintah **ip pool remove [find where comment=Test]**.

```
[admin@R_Test] > ip pool remove [find where comment=Test]
```

9. Memverifikasi penghapusan **IP Pool** dengan mengeksekusi perintah **ip pool print**.

```
[admin@R_Test] > ip pool print
```

10. Menghapus pengaturan pengalamatan **IP** pada *interface* **VLAN** dengan komentar bernilai **Test** melalui eksekusi perintah **ip address remove [find where comment=Test]**.

```
[admin@R_Test] > ip address remove [find where comment=Test]
```

11. Memverifikasi penghapusan pengalamatan **IP** pada *interface* **VLAN** dengan mengeksekusi perintah **ip address print**.

```
[admin@R_Test] > ip address print
Flags: D - DYNAMIC
Columns: ADDRESS, NETWORK, INTERFACE
#  ADDRESS      NETWORK      INTERFACE
0   192.168.1.1/24  192.168.1.0  ether1
1   192.168.0.1/30  192.168.0.0  ether2
2 D 10.0.137.73/24  10.0.137.0   ether3
```

Sudah tidak terlihat pengalamatan **IP** dengan komentar **Test** sehingga seluruh pengalamatan **IP** pada *interface* **VLAN** telah berhasil dihapus.

12. Menghapus seluruh *interface* **VLAN** dengan komentar bernilai **Test** melalui eksekusi perintah **interface vlan remove [find where comment=Test]**.

```
[admin@R_Test] > interface vlan remove [find where comment=Test]
```

13. Memverifikasi penghapusan *interface* **VLAN** dengan mengeksekusi perintah **interface vlan print**.

```
[admin@R_Test] > interface vlan print
```

14. Menghapus pengaturan **NAT** dengan komentar bernilai **Test** melalui eksekusi perintah `ip firewall nat remove [find where comment=Test]`.

```
[admin@R_Test] > ip firewall nat remove [find where comment=Test]
```

15. Memverifikasi penghapusan **NAT** dengan mengeksekusi perintah `ip firewall nat print`.

```
[admin@R_Test] > ip firewall nat print
Flags: X - disabled, I - invalid; D - dynamic
[admin@R_Test] >
```

16. Menghapus pengaturan **DHCP Client** dengan komentar bernilai **Test** melalui eksekusi perintah `ip dhcp-client remove [find where comment=Test]`.

```
[admin@R_Test] > ip dhcp-client remove [find where comment=Test]
```

17. Memverifikasi penghapusan **DHCP Client** dengan mengeksekusi perintah `ip dhcp-client print`.

```
[admin@R_Test] > ip dhcp-client print
```

BAB 6

ANSIBLE UNTUK OTOMATISASI LAB NETDEVOPS DI PNETLAB

A. Pengenalan Ansible

Menurut situs [Ansible](#), *Ansible* merupakan mesin otomatisasi Teknologi Informasi (TI) sederhana yang dapat mengotomatisasi *cloud provisioning*, manajemen konfigurasi, penerapan aplikasi, *intra-service orchestration* dan kebutuhan TI lainnya. Berdasarkan situs [Edureka](#), keuntungan menggunakan *Ansible* antara lain:

1. **Simple**, menggunakan sintak penulisan yang ditulis menggunakan YAML yang disebut dengan *playbook*.
2. **Agentless**, tidak diperlukan *agent* atau *software* khusus untuk diinstalasi pada *host* yang diotomatisasi.
3. **Powerful & Flexible**, memiliki banyak modul untuk manajemen infrastruktur, jaringan, sistem operasi dan layanan.
4. **Efficient**, tidak ada *software* yang diperlukan untuk instalasi pada *server* sehingga lebih banyak sumber daya yang dapat digunakan oleh aplikasi.

Ansible dapat melakukan beberapa hal, meliputi:

1. **Provisioning**

Ansible memastikan paket-paket yang dibutuhkan akan diunduh dan diinstalasi sehingga aplikasi dapat diterapkan.

2. **Configuration Management**

Menetapkan dan mempertahankan konsistensi dari kinerja produk dengan mencatat dan memperbaharui informasi lengkap yang menjelaskan perangkat keras dan lunak perusahaan. Seperti versi dan pembaharuan yang telah diterapkan pada paket *software* yang terinstal dan lokasi serta alamat jaringan dari perangkat keras.

3. Application Deployment

Ansible dapat memajemen secara efektif keseluruhan *life cycle* dari aplikasi, mulai dari *development* sampai produksi.

4. Security and Compliance

Kebijakan keamanan dapat didefinisikan pada *Ansible* sehingga proses pemindaian dan pemulihan kebijakan ke lokasi dapat diintegrasikan ke dalam proses secara otomatis.

5. Orchestration

Ansible menyediakan *orchestration* dalam arti menyelaraskan permintaan bisnis dengan aplikasi, data, dan infrastruktur. Ini mendefinisikan kebijakan dan tingkat layanan melalui alur kerja otomatis, penyediaan, dan manajemen perubahan sehingga menciptakan aplikasi selaras dengan infrastruktur yang dapat ditingkatkan dari atas ke bawah berdasarkan kebutuhan setiap aplikasi.

Terminologi pada Ansible

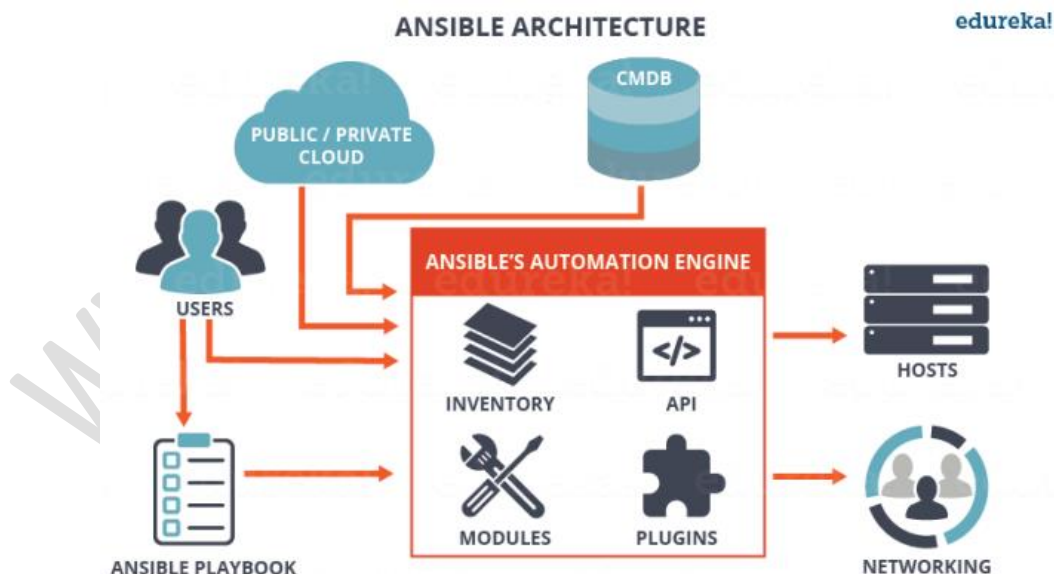
Terdapat beberapa terminologi pada *Ansible* yaitu antara lain:

- a. **Controller Machine:** mesin dimana *Ansible* diinstalasi. Mesin ini membantu untuk menjalankan *provisioning* pada *server* yang dikelola.
- b. **Inventory:** *file* inisialisasi yang memuat informasi tentang *server* yang dikelola.
- c. **Playbook:** *file* yang memuat definisi terkait tugas (*tasks*) yang harus diotomatisasi oleh *Ansible* dan ditulis dalam format YAML. *Playbook* dapat memuat satu atau lebih *play*. *Play* mendefinisikan tugas atau aktivitas yang dieksekusi pada *host* yang dikelola pada *file inventory*.
- d. **Task:** tindakan yang akan diterapkan atau dieksekusi pada *host* yang dikelola, sebagai contoh instalasi *package* tertentu.

- e. **Module:** kode atau *binary* yang disalin dan dieksekusi oleh Ansible pada setiap node yang dikelola untuk menyelesaikan tindakan yang didefinisikan pada *task*. *Ansible* memiliki banyak modul *built-in* (bawaan), namun dapat juga dibuat modul khusus.
- f. **Role:** mekanisme yang digunakan dalam memecah *playbook* ke dalam beberapa *file* untuk memfasilitasi berbagi pakai dan menggunakan kembali bagian dari *provisioning*.
- g. **Play:** eksekusi dari *playbook* mulai dari awal sampai akhir.
- h. **Facts:** data yang dikumpulkan terkait *node* target atau *host* yang dikonfigurasi berupa *variable* global yang memuat informasi tentang sistem, seperti alamat IP atau sistem operasi.
- i. **Handlers:** Digunakan untuk menjalankan task *hanya* ketika perubahan tertentu dibuat pada *host* yang dikelola. Sebagai contoh *restart service* ketika dilakukan perubahan pada *file* konfigurasi.

Arsitektur Ansible

Ansible memiliki arsitektur seperti terlihat pada gambar berikut:



Sumber gambar: Reshma Ahmed (2021)

Ansible Automation Engine (AAE) terdiri dari *Inventory*, *Application Programming Interface (API)*, *Modules* dan *Plugins*. Pengguna yang menulis *ansible playbook* berinteraksi secara langsung dengan AAE ketika *playbook* tersebut dieksekusi. *Playbook* ditulis menggunakan format *YAML* yang didalamnya memuat *task* yang akan dieksekusi oleh *Ansible*. *Inventory* merupakan daftar dari *host* atau *node* yang memiliki alamat IP sebagai mesin yang dimanajemen oleh *ansible* dan lokasi eksekusi dari *playbook*. API pada *ansible* digunakan sebagai metode *transport* untuk komunikasi dengan layanan *Cloud* baik publik maupun privat.

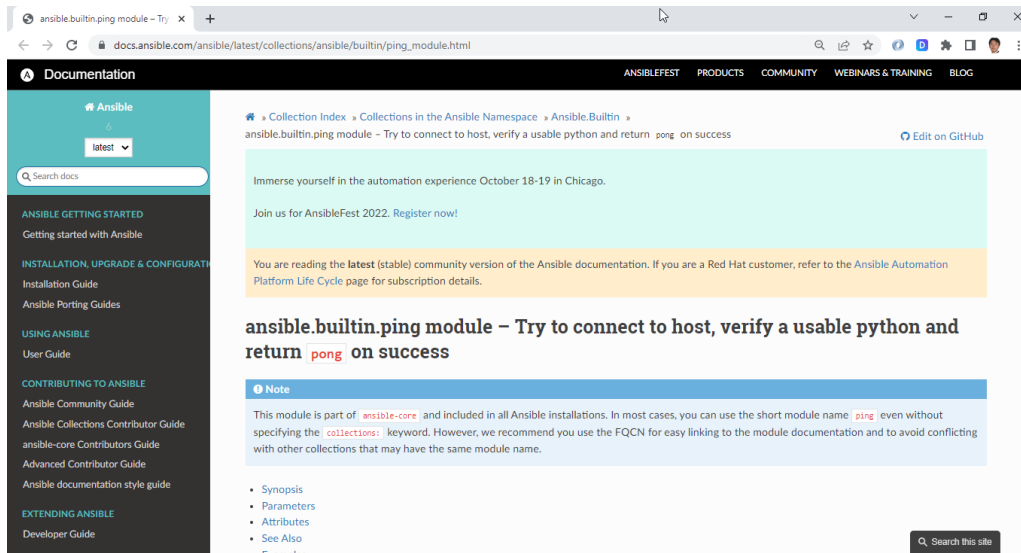
Module dieksekusi secara langsung pada mesin yang dimanajemen atau sistem remote oleh *ansible* melalui *playbook*. *Module* dapat mengatur *service*, *package*, sumber daya, *file* atau mengeksekusi perintah sistem. Sebaliknya *plugin* dieksekusi pada *control node* atau mesin dimana *ansible* terinstalasi yaitu di dalam proses `/usr/bin/ansible`. *Plugin* menawarkan perluasan dari fitur inti *ansible* seperti transformasi data, *logging output*, koneksi ke *inventory* dan lain-lain. Selain itu *ansible* juga dapat digunakan untuk mengotomatisasi jaringan (Reshma Ahmed, 2021).

Ansible Modules

Ansible memiliki berbagai *modules*, beberapa diantaranya yang digunakan pada studi kasus otomatisasi lab NetDevOps adalah:

1. ping

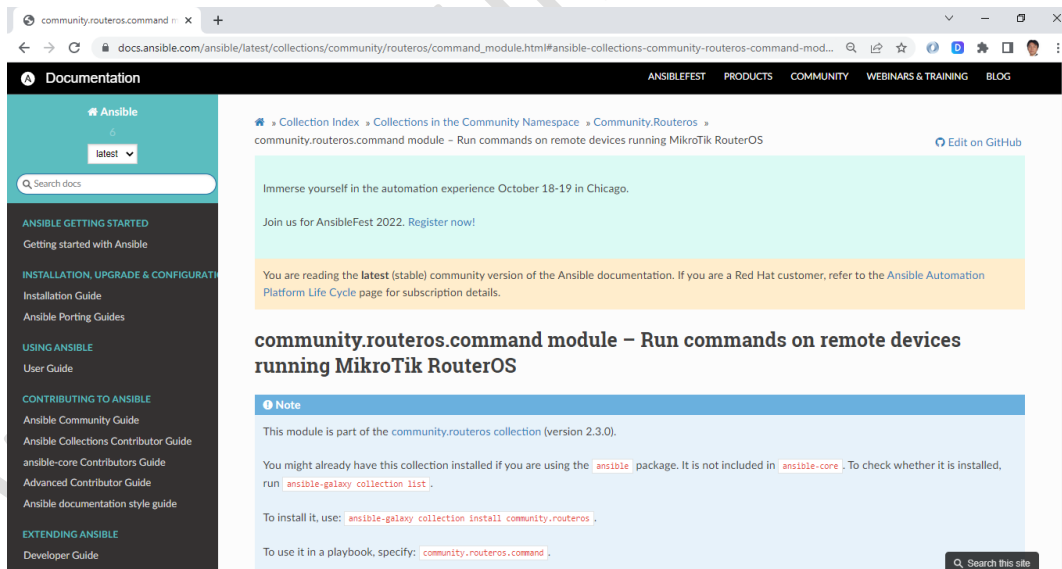
Digunakan untuk mencoba terkoneksi ke *host*, memverifikasi python yang dapat digunakan dan mengembalikan *pong* jika sukses.



Dokumentasi terkait *module* ini dapat diakses pada alamat https://docs.ansible.com/ansible/latest/collections/ansible/builtin/ping_module.html.

2. **routeros_command**

Digunakan untuk mengeksekusi perintah pada perangkat jarak jauh (*remote*) yang menjalankan **MikroTik RouterOS**.

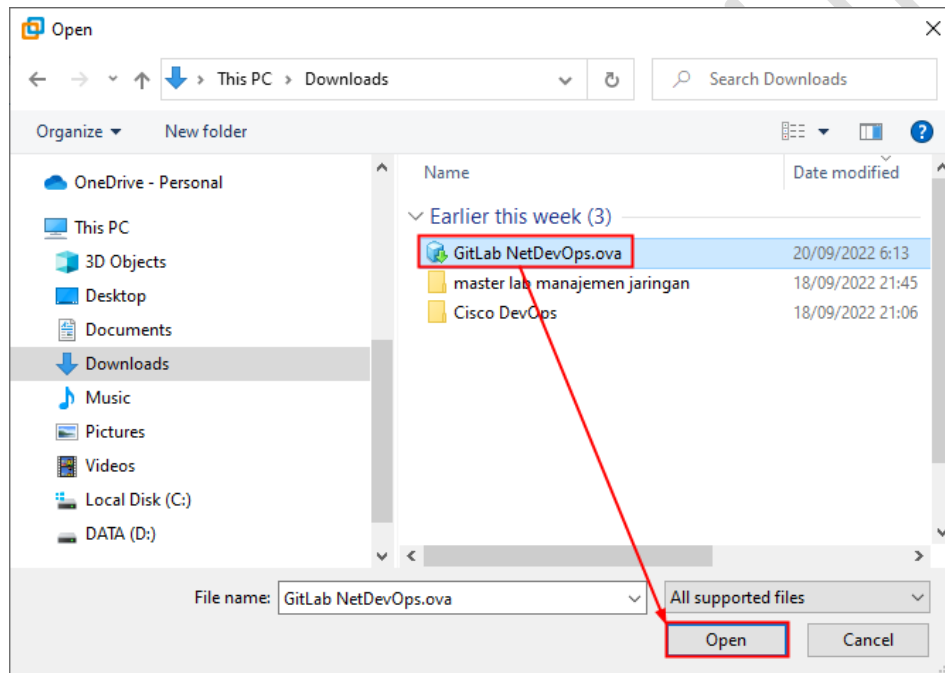


Dokumentasi terkait *module* ini dapat diakses pada alamat https://docs.ansible.com/ansible/latest/collections/community/routeros/command_module.html#ansible-collections-community-routeros-command-module.

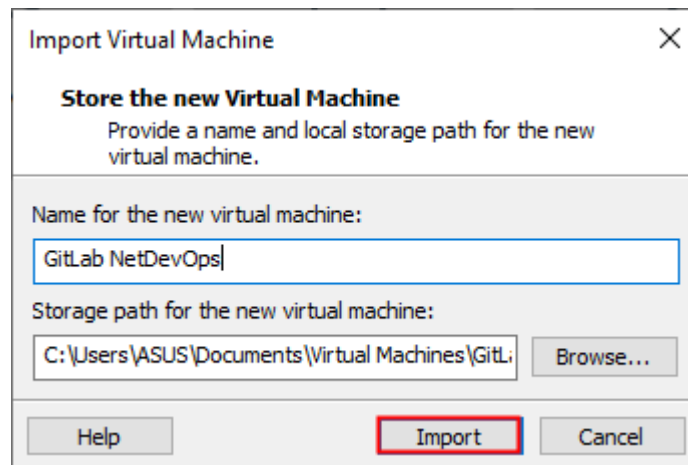
B. Mengimport VM GITLAB NETDEVOPS Di VMware Workstation

Adapun langkah-langkah untuk melakukan *import GitLab Container* adalah sebagai berikut:

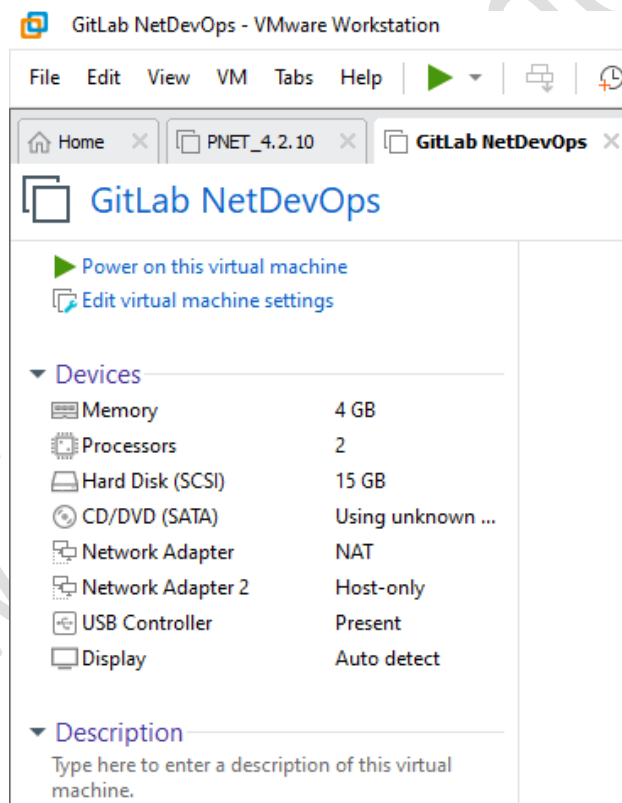
1. Mengakses aplikasi **VMWare Workstation**. Pada kotak dialog *VMWare Workstation* yang tampil, pilih menu **File > Open**. Tampil kotak dialog **Open** dan arahkan ke lokasi penyimpanan *file GitLab NetDevOps.ova*. Pilih *file* tersebut dan klik tombol **Open**, seperti yang ditunjukkan pada gambar berikut:



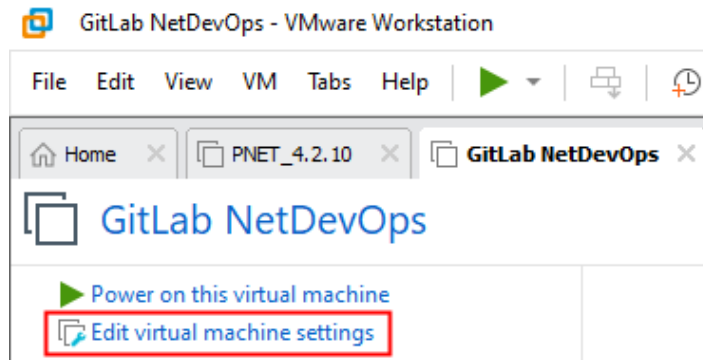
Selanjutnya tampil kotak dialog **Import Virtual Machine** untuk mengatur nama pengenalan dari *virtual machine* dan lokasi penyimpanan untuk *virtual machine*. Secara *default* nama pengenalan dari *virtual machine* adalah **GitLab NetDevOps**. Sedangkan lokasi penyimpanan (*path*) **virtual machine default** adalah di *home* direktori dari *user* yang digunakan login ke sistem Windows. Umumnya berlokasi di drive **C:\User\NamaLoginUser\Documents\Virtual Machines\GitLab NetDevOps**. Sebagai contoh untuk *user* dengan nama login **ASUS** maka lokasinya adalah di **C:\Users\ASUS\Documents\Virtual Machines\GitLab NetDevOps**, seperti yang ditunjukkan pada gambar berikut:



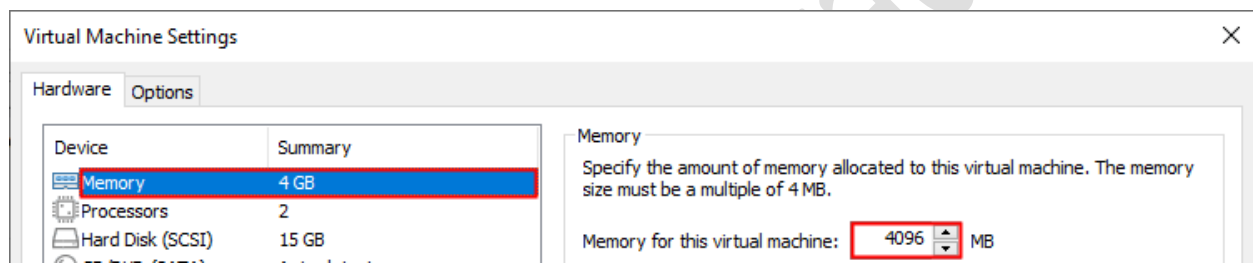
Tekan tombol **Import** dan tunggu hingga selesai dilakukan. Apabila proses *import* tersebut telah berhasil dilakukan maka akan terlihat seperti gambar berikut:



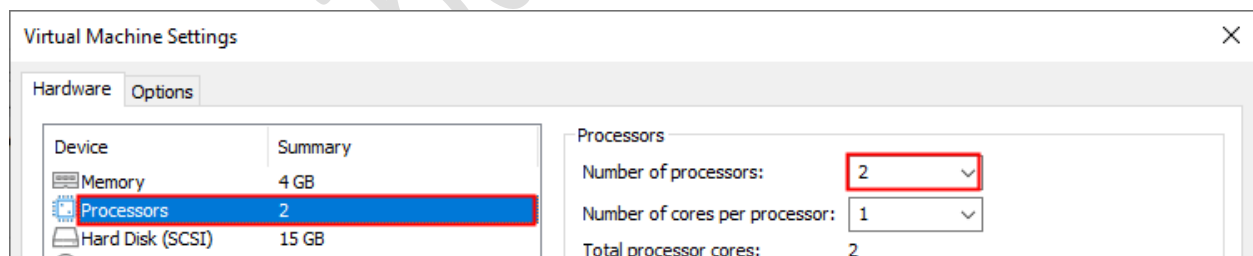
Apabila diperlukan maka dapat dilakukan penyesuaian terkait ukuran memori dan prosesor dengan memilih **Edit virtual machine settings**, seperti terlihat pada gambar berikut:



Pada kotak dialog yang tampil **Virtual Machine Settings**. Pilih **Memory** pada panel sebelah kiri maka pada panel sebelah kanan terdapat inputan **Memory for this virtual machine** untuk menyesuaikan ukuran memori yang digunakan oleh *virtual machine*, seperti terlihat pada gambar berikut:

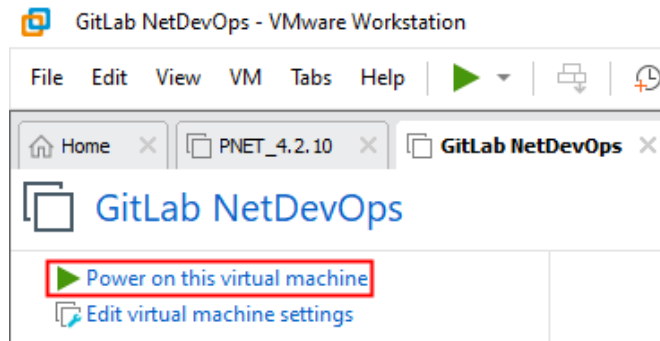


Selanjutnya pilih **Processors** pada panel sebelah kiri maka pada panel sebelah kanan terdapat dropdown **Number of processors** untuk menyesuaikan jumlah prosesor yang digunakan oleh *virtual machine*, seperti terlihat pada gambar berikut:

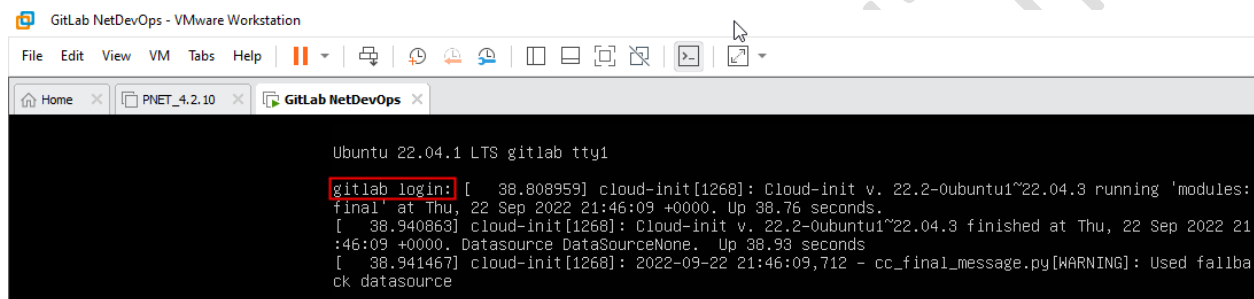


Klik tombol **OK** untuk menyimpan perubahan.

2. Menjalankan **VM GitLab NetDevOps** dengan memilih **Power on this virtual machine**, seperti terlihat pada gambar berikut:



Tunggu hingga proses *booting* selesai dilakukan dimana ditandai dengan munculnya *prompt login* seperti terlihat pada gambar berikut:



C. Mengatur Ansible Inventory

Inventory merupakan file inisialisasi yang digunakan oleh *Ansible* untuk mendaftarkan dan mengelompokkan mesin atau *host* yang akan dikelola (*managed machine*). Lokasi dari file *inventory* secara *default* adalah `"/etc/ansible/hosts"` dan memiliki format seperti file INI atau *initialization file format*. Konten di dalam *file inventory* dapat memuat *hostname* atau alamat IP. Contoh sederhana dari konten *file hosts* adalah sebagai berikut:

```
# cat /etc/ansible/hosts
192.168.1.2
R_Test.netdevops.local
```

Terlihat terdapat 2 (dua) *host* yaitu `192.168.1.2` dan `R_Test.netdevops.local`.

Selain itu dapat pula diterapkan pengelompokan (*group*) dimana penulisannya diawali pada bagian atas dengan nama *group* yang diapit dengan tanda kurung siku buka tutup, `[groupname]` dan pada baris berikutnya diikuti oleh *host* yang tergabung pada kelompok tersebut. Menurut dokumentasi [ansible](https://docs.ansible.com/ansible/latest/user_guide/playbooks_inventory.html), *group* digunakan untuk mengklasifikasi sistem dan

menentukan sistem yang dikontrol berdasarkan waktu dan tujuan tertentu. Berikut adalah contoh dari *file hosts* yang telah menerapkan pengelompokan (*group*):

```
# cat /etc/ansible/hosts

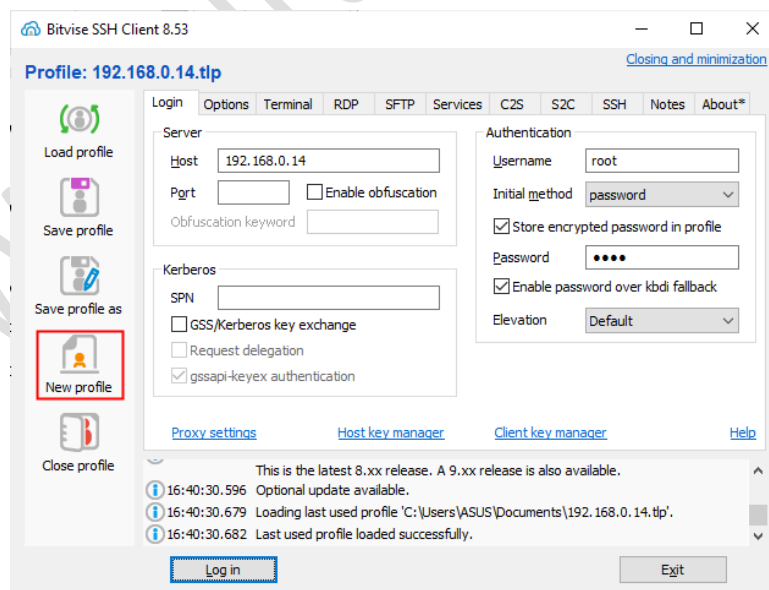
[webservers]
www01.netdevops.local
www02.netdevops.local

[routers]
192.168.0.1
192.168.1.2
```

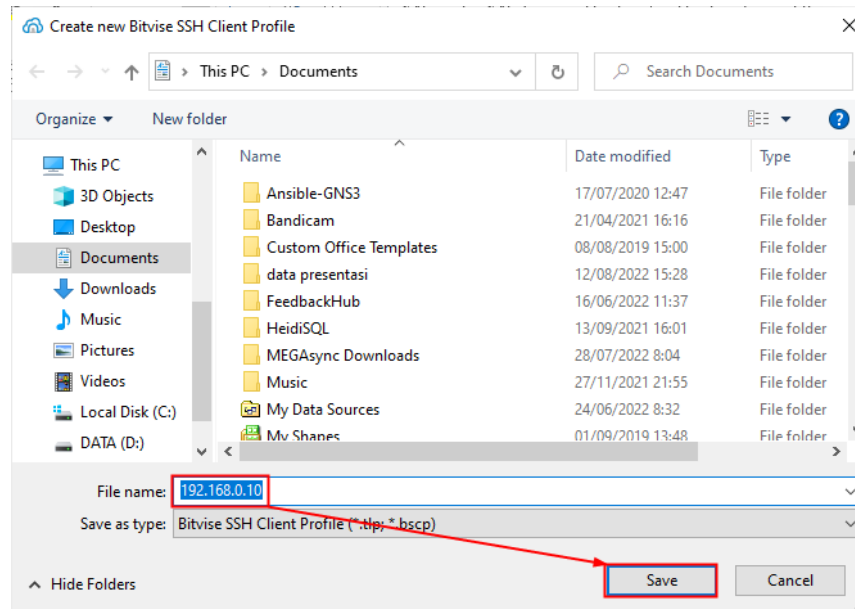
Terlihat terdapat 2 (dua) *group* yaitu **webservers** dan **routers**. *Group webservers* memiliki 2 (dua) *host* yang ditulis menggunakan format *Fully Qualified Domain Name (FQDN)* yaitu *www01.netdevops.local* dan *www02.netdevops.local*. Begitu pula *group routers* juga memiliki 2 (dua) *host* yang dituliskan menggunakan alamat IP yaitu *192.168.0.1* dan *192.168.1.2*.

Adapun langkah-langkah untuk mengatur *inventory* pada VM *GitLab NetDevOps* yang bertindak sebagai *Ansible Control Machine* adalah sebagai berikut:

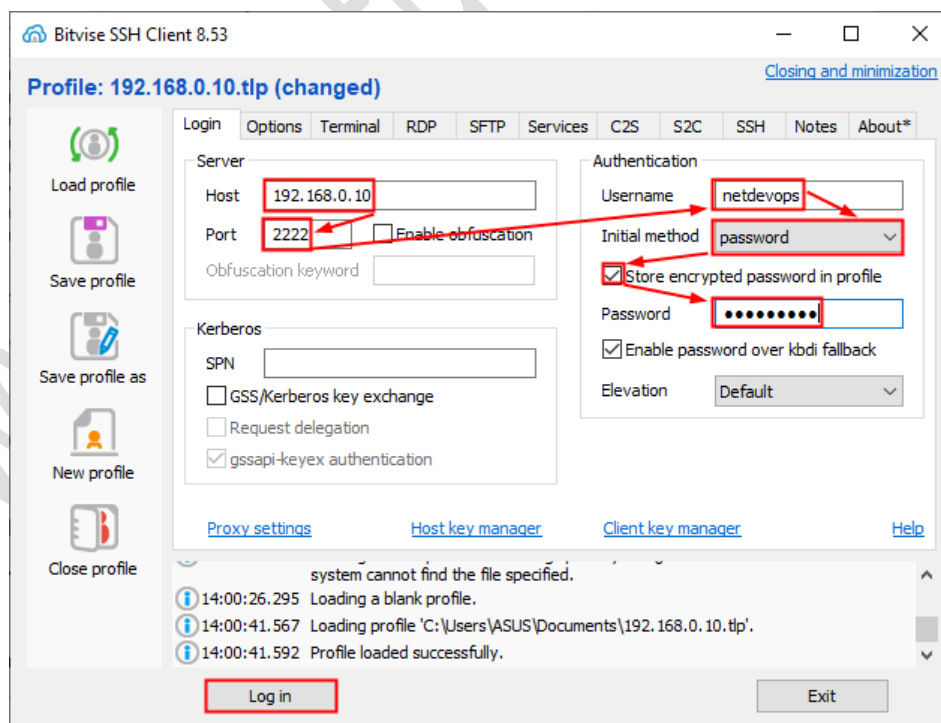
- Mengakses aplikasi **Bitvise SSH Client**. Pada kotak dialog **Bitvise SSH Client** yang tampil, klik **New profile**, seperti yang ditunjukkan pada gambar berikut:



Tampil kotak dialog **Create new Bitvise SSH Client Profile** dan pada inputan **File name**, masukkan **192.168.0.10** sebagai nama profil dari *Bitvise*, seperti yang ditunjukkan pada gambar berikut:



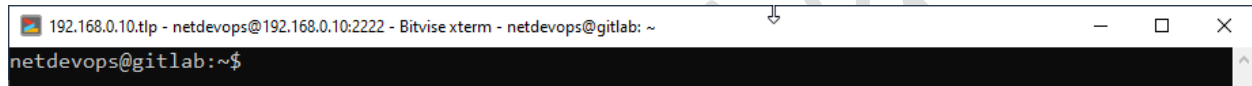
Selanjutnya tampil kotak dialog **Bitvise SSH Client**. Lengkapi pengaturan beberapa parameter, seperti yang ditunjukkan pada gambar berikut:



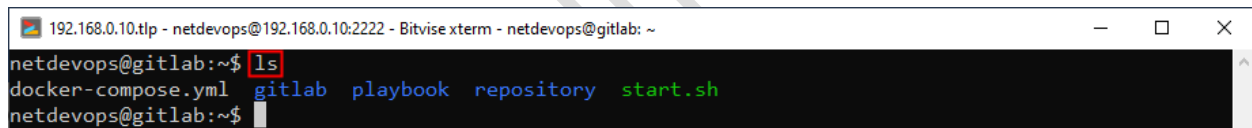
- **Host**, masukkan alamat IP dari PNETLab yaitu **192.168.0.10**.
- **Port**, masukkan **"2222"**.
- **Username**, masukkan **"netdevops"**.
- **Initial method**, pilih **password**.
- Centang (v) **Store encrypted password in profile** untuk menyimpan sandi yang terenkripsi di dalam profil.
- **Password**, masukkan **"netdevops"**.

Klik tombol **Log in**. Tampil kotak dialog **Host Key Verification** dan klik tombol **Accept and Save**.

- b. Pada panel sebelah kiri pilih pada **New terminal console** maka akan tampil kotak dialog **Bitvise xterm**, seperti terlihat pada gambar berikut:

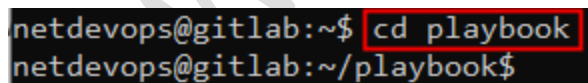


- c. Menampilkan isi dari direktori dimana saat ini berada dengan mengeksekusi perintah **ls**, seperti terlihat pada gambar berikut:



Terlihat terdapat beberapa direktori dimana salah satunya bernama **playbook** yang digunakan untuk menampung *file-file* YAML dari *Ansible*.

- d. Berpindah ke direktori **playbook** dengan mengeksekusi perintah **cd playbook**, seperti terlihat pada gambar berikut:



- e. Membuat *file* dengan nama **"hosts"** di dalam direktori dimana saat ini berada yaitu **playbook** menggunakan *editor nano*. *File* tersebut bertindak sebagai *inventory* yang didalamnya memuat pengelompokan mesin atau *host* yang akan dikelola (*managed machine*) yaitu *node-node* yang terdapat di lab *NetDevOps* pada PNETLab. *Node* yang dimaksud meliputi *router* baik **R_Test** maupun **R_Production** dan *switch* baik **SW_Test** maupun **SW_Production**). Eksekusi

perintah **nano hosts** untuk memproses pembuatan *file hosts*, seperti terlihat pada gambar berikut:

```
netdevops@gitlab:~/playbook$ nano hosts
```

Dengan konten sebagai berikut:

```
1  [routers:children]
2  rtr_Test
3  msw_Test
4  rtr_Production
5  msw_Production
6
7  [rtr_Test]
8  R_Test ansible_host=192.168.0.1
9
10 [msw_Test]
11 SW_Test ansible_host=192.168.1.2
12
13 [rtr_Production]
14 R_Production ansible_host=192.168.0.6
15
16 [msw_Production]
17 SW_Production ansible_host=192.168.11.2
18
19 [routers:vars]
20 ansible_user=admin
21 ansible_password=admin
22 ansible_connection=network_cli
23 ansible_network_os=routeros
```

Penjelasan:

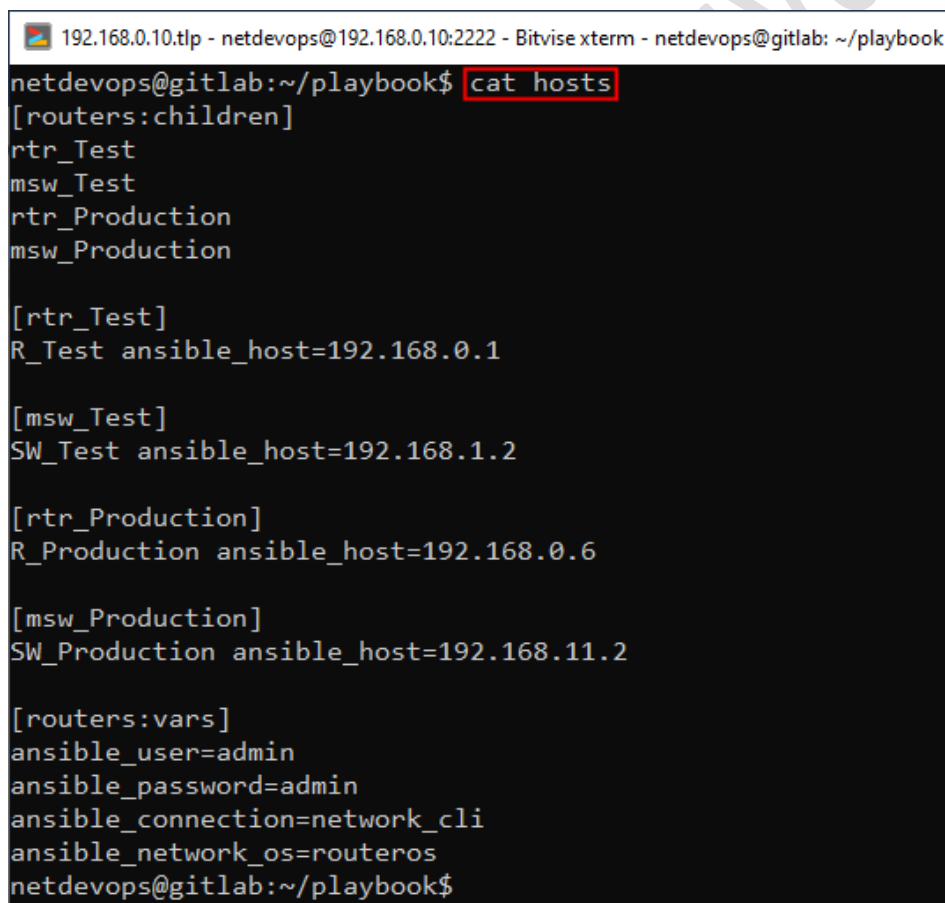
- Baris 1 sampai dengan 5 digunakan untuk membuat grup dari grup yaitu **rtr_Test**, **msw_Test** dan **rtr_Production** serta **msw_Production** menggunakan **:children** dengan nama **routers**.
- Baris 7 merupakan nama pengelompokan atau *group* yaitu **rtr_Test**.
- Baris 8 merupakan *inventory_hostname* atau nama alias bagi *host* dengan alamat IP **192.168.0.1** yang menjadi nilai dari *variable* **ansible_host**. **ansible_host** merupakan *variable* yang dapat digunakan oleh *ansible* untuk melakukan koneksi dengan *remote host* alias **R_Test**.
- Baris 10 merupakan nama pengelompokan atau *group* yaitu **msw_Test**.
- Baris 11 merupakan *inventory_hostname* atau nama alias bagi *host* dengan alamat IP **192.168.1.2** yang menjadi nilai dari *variable* **ansible_host**. **ansible_host** merupakan *variable* yang dapat digunakan oleh *ansible* untuk melakukan koneksi dengan *remote host* alias **SW_Test**.
- Baris 13 merupakan nama pengelompokan atau *group* yaitu **R_Production**.
- Baris 14 merupakan *inventory_hostname* atau nama alias bagi *host* dengan alamat IP **192.168.0.6** yang menjadi nilai dari *variable* **ansible_host**. **ansible_host** merupakan *variable* yang dapat digunakan oleh *ansible* untuk melakukan koneksi dengan *remote host* alias **R_Production**.
- Baris 16 merupakan nama pengelompokan atau *group* yaitu **msw_Production**.
- Baris 17 merupakan *inventory_hostname* atau nama alias bagi *host* dengan alamat IP **192.168.11.2** yang menjadi nilai dari *variable* **ansible_host**. **ansible_host** merupakan *variable* yang dapat digunakan oleh *ansible* untuk melakukan koneksi dengan *remote host* alias **SW_Production**.
- Baris 19 sampai dengan 23 digunakan untuk mengatur *group variables* untuk grup **routers**.
- Baris 20: *Variable* **ansible_user** menentukan *user* yang digunakan untuk terkoneksi ke seluruh *node* di PNETLab yaitu **admin**.

- Baris 21: *Variable* **ansible_password** menentukan password yang digunakan untuk mengotentikasi host yaitu **admin**.
- Baris 22: *Variable* **ansible_connection** digunakan untuk mengatur agar *Ansible* memperlakukan *managed machine* sebagai perangkat jaringan dengan lingkungan eksekusi yang terbatas yaitu **network_cli**.
- Baris 23: *Variable* **ansible_network_os** digunakan untuk menginformasikan platform jaringan dari *managed machine* yaitu **routeros**.

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan tombol **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

- f. Memverifikasi hasil pembuatan *file inventory* dengan nama **hosts** tersebut melalui mengeksekusi perintah **cat hosts**, seperti terlihat pada gambar berikut:



```

192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~/playbook
netdevops@gitlab:~/playbook$ cat hosts
[routers:children]
rtr_Test
msw_Test
rtr_Production
msw_Production

[rtr_Test]
R_Test ansible_host=192.168.0.1

[msw_Test]
SW_Test ansible_host=192.168.1.2

[rtr_Production]
R_Production ansible_host=192.168.0.6

[msw_Production]
SW_Production ansible_host=192.168.11.2

[routers:vars]
ansible_user=admin
ansible_password=admin
ansible_connection=network_cli
ansible_network_os=routeros
netdevops@gitlab:~/playbook$

```

Terlihat *inventory* terkait *node-node* pada lab *NetDevOps* telah berhasil dibuat.

D. Ansible Configuration

ansible.cfg merupakan *file* yang memuat penyesuaian konfigurasi untuk mengontrol *Ansible*. Adapun langkah-langkah untuk membuat *file ansible.cfg* di dalam direktori *playbook* adalah sebagai berikut:

- a. Mengakses **Bitvise xterm** dari **VM GitLab NetDevOps**.
- b. Mengeksekusi perintah **nano ansible.cfg** untuk membuat *file ansible.cfg* menggunakan *editor nano*.

```
netdevops@gitlab:~/playbook$ nano ansible.cfg
```

Dengan konten sebagai berikut:

```
1 [defaults]
2 inventory=./hosts
3 host_key_checking = False
4 retry_files_enabled = False
5 deprecation_warnings = False
```

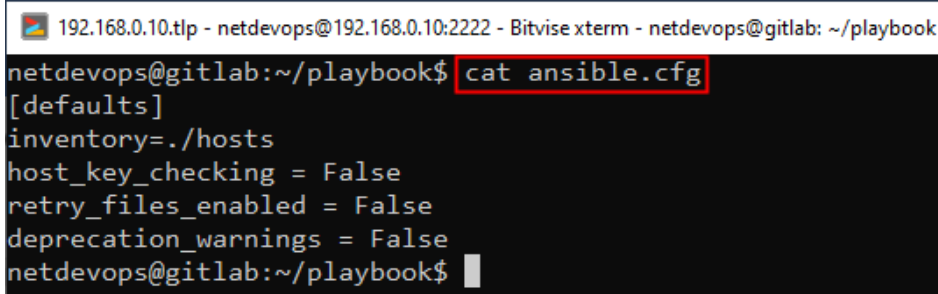
Penjelasan:

- Baris 1 digunakan untuk menentukan *section* dari file konfigurasi *Ansible* yaitu **[defaults]**.
- Baris 2 digunakan untuk menentukan *file inventory* yang digunakan yaitu *file hosts* yang terdapat di direktori dimana saat ini berada.
- Baris 3 digunakan untuk menonaktifkan pengecekan **host key** terutama pada lingkungan pengembangan lokal yang tidak memiliki pengaturan SSH *key*.
- Baris 4 digunakan untuk menonaktifkan pembuatan *file .retry* ketika terjadi kegagalan saat menjalankan *playbook*.
- Baris 5 digunakan untuk menonaktifkan peringatan penghentian penggunaan fitur lama yang dijadwalkan untuk dihapus dalam rilis *Ansible* di masa mendatang.

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan tombol **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

- c. Memverifikasi hasil pembuatan *file inventory* dengan nama **ansible.cfg** tersebut melalui mengeksekusi perintah **cat ansible.cfg**, seperti terlihat pada gambar berikut:



```
192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~/playbook
netdevops@gitlab:~/playbook$ cat ansible.cfg
[defaults]
inventory=./hosts
host_key_checking = False
retry_files_enabled = False
deprecation_warnings = False
netdevops@gitlab:~/playbook$
```

E. Ansible AD-HOC Commands

Menurut dokumentasi [Ansible](#), **Ad-Hoc command** merupakan salah satu cara dalam menjalankan *ansible* sehingga dapat digunakan untuk melakukan sesuatu dengan cepat dan pengguna tidak berkeinginan untuk menyimpannya ke dalam *playbook*. *Ansible Ad-Hoc command* memiliki sintak penulisan:

```
$ ansible <inventory> -m
```

Penjelasan parameter:

- **<inventory>**, menentukan lokasi dari *file inventory host* yang akan digunakan
- **-m**, menentukan nama *module ansible* yang akan digunakan.

Salah satu contoh penggunaan perintah *ansible ad-hoc command* adalah untuk memverifikasi koneksi ke seluruh *managed machines* dengan nama *group routers* menggunakan *module ping*. Eksekusi perintah **ansible routers -m ping** melalui **Bitvise xterm** dari **VM GitLab NetDevOps** sehingga hasilnya akan terlihat seperti pada gambar berikut:

```

192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~/playbook
netdevops@gitlab:~/playbook$ ansible routers -m ping
SW_Production | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
R_Production | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
SW_Test | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
R_Test | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}

```

Output menunjukkan bahwa verifikasi koneksi ke *node* *SW_Production*, *R_Production*, *SW_Test* dan *R_Test* yang tergabung dalam *group* **routers** berhasil dilakukan dimana ditandai dengan pesan **pong**. Selain itu terlihat pula informasi *interpreter python* ditemukan pada setiap *node* tersebut yaitu berada di **/usr/bin/python3**.

F. Playbook

Merupakan *file* dimana kode *ansible* ditulis dalam format YAML. Setiap *playbook* dapat terdiri dari satu atau lebih *play* didalamnya. *Play* berfungsi untuk memetakan sekumpulan instruksi yang didefinisikan terhadap *host* tertentu. Setiap *play* memuat daftar atau *list* dari *task*

yang dieksekusi secara berurutan yaitu satu task dalam satu waktu pada semua *host* yang sesuai sebelum menuju ke *task* berikutnya. *Playbook* dieksekusi secara *top-down* sehingga ketika terjadi kegagalan eksekusi *task* pada *host* tertentu maka *playbook* akan berhenti. Untuk itu diperlukan eksekusi ulang *playbook* setelah selesai diperbaiki.

Menurut situs [Tutorialspoint](https://www.tutorialspoint.com/ansible/ansible-variables.htm), YAML memiliki beberapa *tags* yang umum digunakan pada *playbook* yaitu antara lain:

- a) `name` digunakan untuk menentukan nama dari *ansible playbook*.
- b) `hosts` digunakan untuk menentukan daftar *host* atau *group host* sebagai lokasi eksekusi *task*.
- c) `vars` digunakan untuk mendefinisikan *variable* *playbook* dimana penggunaannya serupa dengan yang terdapat di bahasa pemrograman.
- d) `tasks` merupakan daftar tindakan atau aksi yang perlu dieksekusi. Didalam *tasks* memuat nama dari *task* dan kode yang berhubungan dengan modul yang harus dieksekusi.

Adapun contoh sederhana dari pembuatan dan verifikasi serta eksekusi *ansible playbook* adalah sebagai berikut:

- a. Mengakses **Bitwise xterm** dari **VM GitLab NetDevOps**.
- b. Membuat file *playbook* dengan nama "`basic.yml`" menggunakan *editor nano* dengan mengeksekusi perintah `nano basic.yml`.

```
netdevops@gitlab:~/playbook$ nano basic.yml
```

Dengan konten seperti berikut:

```
1 ---
2 - name: Dasar YAML
3   hosts: routers
4   vars:
5     pesan: "Hello World"
6   tasks:
```

```
7 - name: Menampilkan Variable
8   debug: var=pesan
```

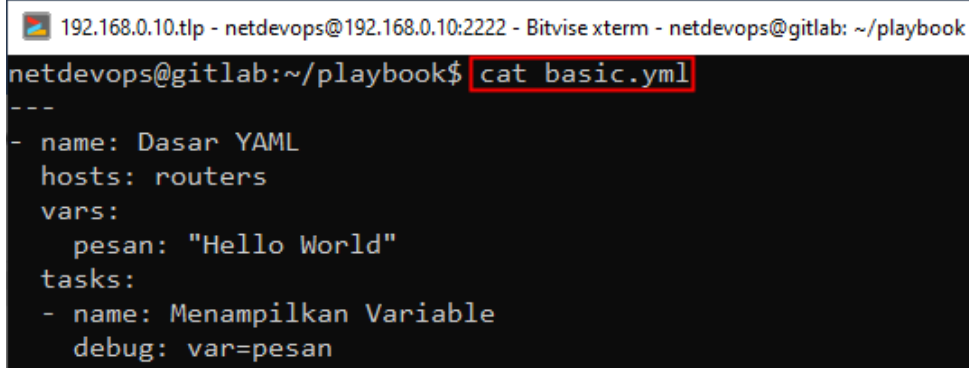
Penjelasan:

- Baris 1: `---` (3 *hyphen*) merupakan awal dari dokumen YAML.
- Baris 2: `name` digunakan untuk menentukan nama dari *ansible playbook* yaitu **“Dasar YAML”**.
- Baris 3: `hosts` digunakan untuk menentukan daftar *host* atau *group host* sebagai lokasi eksekusi *task* yaitu *group routers*.
- Baris 4: `vars` digunakan untuk mendefinisikan *variable* pada *playbook* dimana penggunaannya serupa dengan yang terdapat di bahasa pemrograman.
- Baris 5: Deklarasi *variable* dengan nama **“pesan”** yang memiliki nilai **“Hello World”**.
- Baris 6: `tasks` merupakan daftar tindakan atau aksi yang perlu dieksekusi.
- Baris 7: `name` digunakan untuk menentukan nama dari *task* yaitu **“Menampilkan Variable”**. Di dalam setiap *task* memuat kode yang berhubungan dengan *modul* yang harus dieksekusi.
- Baris 8: nama modul yang dieksekusi yaitu **debug** yang berguna untuk menampilkan pesan ketika eksekusi. Modul ini memiliki parameter **var** yang digunakan untuk menentukan nama *variable* yang di *debug* yaitu **pesan**.

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan tombol **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

- c. Memverifikasi hasil pembuatan *file basic.yml* dengan mengeksekusi perintah **cat basic.yml**.



```

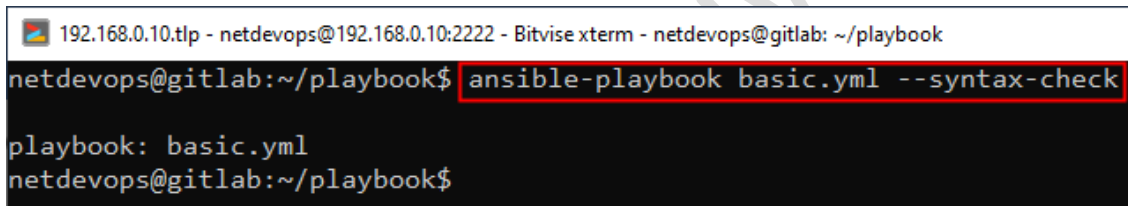
192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~/playbook
netdevops@gitlab:~/playbook$ cat basic.yml
---
- name: Dasar YAML
  hosts: routers
  vars:
    pesan: "Hello World"
  tasks:
    - name: Menampilkan Variable
      debug: var=pesan

```

- d. Mengecek sintak dari *playbook* sebelum dieksekusi sehingga memastikan tidak terdapat permasalahan terkait sintak dengan mengeksekusi perintah `ansible-playbook` dengan flag `--syntax-check`.

```
$ ansible-playbook basic.yml --syntax-check
```

Hasil eksekusi dari perintah tersebut, seperti terlihat pada gambar berikut:



```

192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~/playbook
netdevops@gitlab:~/playbook$ ansible-playbook basic.yml --syntax-check

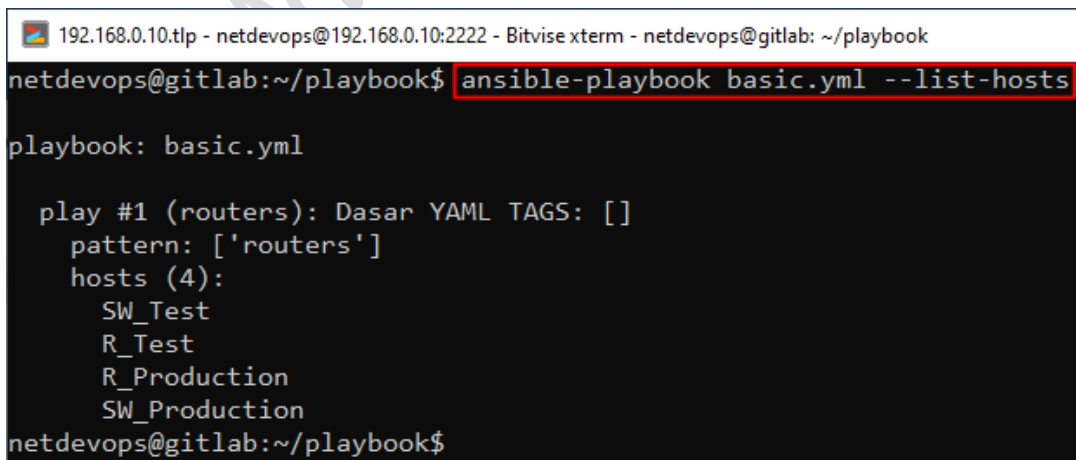
playbook: basic.yml
netdevops@gitlab:~/playbook$

```

- e. Menampilkan informasi *host* yang terdampak oleh *playbook* sebelum dieksekusi dapat menggunakan perintah:

```
$ ansible-playbook basic.yml --list-hosts
```

Hasil eksekusi dari perintah tersebut, seperti terlihat pada gambar berikut:



```

192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~/playbook
netdevops@gitlab:~/playbook$ ansible-playbook basic.yml --list-hosts

playbook: basic.yml

play #1 (routers): Dasar YAML TAGS: []
  pattern: ['routers']
  hosts (4):
    SW_Test
    R_Test
    R_Production
    SW_Production
netdevops@gitlab:~/playbook$

```

Terlihat bahwa *node SW_Production, R_Production, SW_Test* dan *R_Test* pada *group routers* yang terdampak oleh eksekusi *playbook*.

f. Mengeksekusi *playbook*.

```
$ ansible-playbook basic.yml
```

Hasil eksekusi dari *playbook* tersebut, seperti terlihat pada gambar berikut:

```
192.168.0.10:tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~/playbook
netdevops@gitlab:~/playbook$ ansible-playbook basic.yml

PLAY [Dasar YAML] *1*****

TASK [Gathering Facts] *2*****
ok: [R_Production]
ok: [SW_Test]
ok: [SW_Production]
ok: [R_Test]

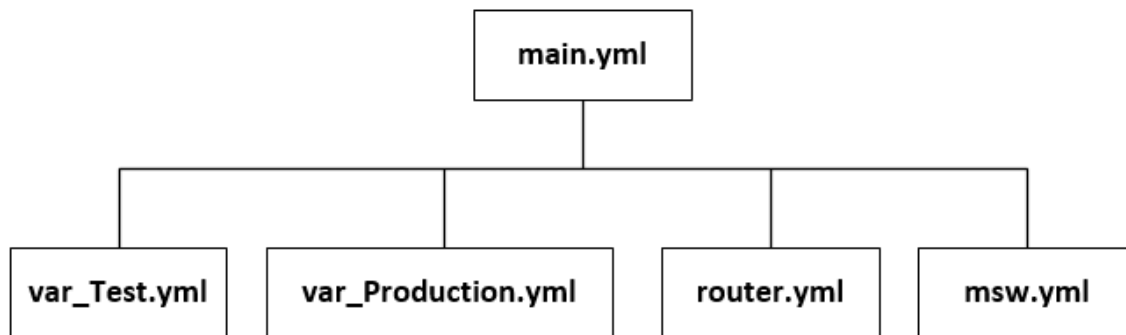
TASK [Menampilkan Variable] *3*****
ok: [R_Test] => {
  "pesan": "Hello World" *4
}
ok: [R_Production] => {
  "pesan": "Hello World" *4
}
ok: [SW_Test] => {
  "pesan": "Hello World" *4
}
ok: [SW_Production] => {
  "pesan": "Hello World" *4
}

PLAY RECAP *****
R_Production      : ok=2 *5 changed=0  unreachable=0  failed=0  skipped=0  rescued=
0 ignored=0
R_Test            : ok=2 *5 changed=0  unreachable=0  failed=0  skipped=0  rescued=
0 ignored=0
SW_Production     : ok=2 *5 changed=0  unreachable=0  failed=0  skipped=0  rescued=
0 ignored=0
SW_Test           : ok=2 *5 changed=0  unreachable=0  failed=0  skipped=0  rescued=
0 ignored=0
```

Penjelasan: (1) nama *play* yang dieksekusi yaitu “**Dasar YAML**”, (2) eksekusi *task gathering facts* yang memuat informasi terkait *remote host* dan terlihat pesan status **ok** yang menyatakan *task* dapat dieksekusi dengan baik, (3) eksekusi *task “Menampilkan Variable”* dan terlihat pesan status **ok** yang menyatakan *task* dapat dieksekusi dengan baik serta menampilkan hasil *debug variable pesan* (4), (5) metric **ok** menginformasikan rangkuman jumlah *task* yang dapat dieksekusi dengan baik atau tidak mengubah apapun pada **routers**.

G. Ansible Playbook Untuk Mengotomatisasi Penerapan Konfigurasi NAT, VLAN, Dan DHCP Server Pada Lab NETDEVOPS

Terdapat 5 (lima) *file* YAML yang terlibat untuk mengotomatisasi penerapan konfigurasi NAT, VLAN dan DHCP Server pada lab **NetDevOps** yaitu **main.yml**, **var_Test.yml**, **var_Production.yml**, **router.yml** dan **msw.yml**. Adapun struktur hirarki dari kelima *file* tersebut, seperti terlihat pada gambar berikut:



Adapun langkah-langkah pembuatan kelima *file* tersebut dan eksekusi *file Ansible playbook* untuk menerapkan konfigurasi pada lab *NetDevOps* adalah sebagai berikut:

- a. Mengakses **Bitvise xterm** dari **VM GitLab NetDevops**.
- b. Membuat *file* YAML dengan nama **var_Test.yml** menggunakan *editor nano* yang memuat *variable* terkait penerapan konfigurasi NAT, VLAN, DHCP Server pada *node R_Test* dan **SW_Test** di lingkungan **Test network** dari lab **NetDevOps** di PNETLab. Perintah yang dieksekusi adalah **nano var_Test.yml**, seperti terlihat pada gambar berikut:

```
netdevops@gitlab:~/playbook$ nano var_Test.yml
```

Dengan konten sebagai berikut:

```

1  internet_interface: ether3
2  vlan_interface: ether1
3
4  vlans:
5    - {id: 2, name: "vlan2_mkt", address: "192.168.2.1/24"}
  
```

```
6      - {id: 3, name: "vlan3_hrd", address: "192.168.3.1/24"}
7      - {id: 4, name: "vlan4_sls", address: "192.168.4.1/24"}
8
9      lease_time: "3h"
10     dns_server: "8.8.8.8,8.8.4.4"
11
12     dhcp_pools:
13         - {name: "vlan2_mkt", ranges: "192.168.2.2-192.168.2.254",
14           network_address: "192.168.2.0/24", gateway:
15             "192.168.2.1"}
16         - {name: "vlan3_hrd", ranges: "192.168.3.2-192.168.3.254",
17           network_address: "192.168.3.0/24", gateway:
18             "192.168.3.1"}
19         - {name: "vlan4_sls", ranges: "192.168.4.2-192.168.4.254",
20           network_address: "192.168.4.0/24", gateway:
21             "192.168.4.1"}
22
23     bridge_ports:
24         - {interface: "ether1", pvid: 2}
25         - {interface: "ether2", pvid: 2}
26         - {interface: "ether3", pvid: 3}
27         - {interface: "ether4", pvid: 3}
28         - {interface: "ether5", pvid: 4}
29         - {interface: "ether6", pvid: 4}
30
31     vlan_tagging:
32         - {tagged: "ether7", untagged: "ether1,ether2", vlan_ids: 2}
33         - {tagged: "ether7", untagged: "ether3,ether4", vlan_ids: 3}
34         - {tagged: "ether7", untagged: "ether5,ether6", vlan_ids: 4}
```

Penjelasan:

- Baris 1 merupakan deklarasi *variable* **internet_interface** dengan nilai **ether3**.
- Baris 2 merupakan deklarasi *variable* **vlan_interface** dengan nilai **ether1**.
- Baris 4 sampai dengan 7 merupakan deklarasi *variable* bernama “**vlangs**” dengan struktur data **list of dictionaries** yang didalamnya memuat 3 (tiga) data untuk pembuatan VLAN, seperti terlihat pada tabel berikut:

Id	name	address
2	vlan2_mkt	192.168.2.1/24
3	vlan3_hrd	192.168.3.1/24
4	vlan4_sls	192.168.4.1/24

- Baris 9 merupakan deklarasi *variable* **lease_time** dengan nilai **3h**.
- Baris 10 merupakan deklarasi *variable* **dns_server** dengan nilai **8.8.8.8,8.8.4.4**.
- Baris 12 sampai dengan 15 merupakan deklarasi *variable* bernama “**dhcp_pools**” dengan struktur data **list of dictionaries** yang didalamnya memuat 3 (tiga) data untuk pembuatan **DHCP** di *node R_Test*, seperti terlihat pada tabel berikut:

Name	ranges	network_address	gateway
vlan2_mkt	192.168.2.2-192.168.2.254	192.168.2.0/24	192.168.2.1
vlan3_hrd	192.168.3.2-192.168.3.254	192.168.3.0/24	192.168.3.1
vlan4_sls	192.168.4.2-192.168.4.254	192.168.4.0/24	192.168.4.1

- Baris 17 sampai dengan 23 merupakan deklarasi *variable* bernama “**bridge_ports**” dengan struktur data **list of dictionaries** yang didalamnya memuat 6 (enam) data untuk pengaturan **Bridge Port** di *node SW_Test*, seperti terlihat pada tabel berikut:

interface	pvid
ether1	2
ether2	2
ether3	3
ether4	3

ether5	4
ether6	4

- Baris 25 sampai dengan 28 merupakan deklarasi *variable* bernama “**vlan_tagging**” dengan struktur data **list of dictionaries** yang didalamnya memuat 3 (tiga) data untuk mengatur **VLAN tagging** pada **interface bridge VLAN** dari *node SW_Test* , seperti terlihat pada tabel berikut:

tagged	untagged	vlan_ids
ether7	ether1,ether2	2
ether7	ether3,ether4	3
ether7	ether5,ether6	4

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan tombol **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

- c. Membuat *file* YAML dengan nama **var_Production.yml** menggunakan *editor nano* yang memuat *variable* terkait penerapan konfigurasi NAT, VLAN, DHCP Server pada *node R_Production* dan *SW_Production* di lingkungan **Production network** dari lab **NetDevOps** di PNETLab. Perintah yang dieksekusi adalah **nano var_Production.yml**, seperti terlihat pada gambar berikut:

```
netdevops@gitlab:~/playbook$ nano var_Production.yml
```

Dengan konten sebagai berikut:

```
1  internet_interface: ether3
2  vlan_interface: ether1
3
4  vlans:
5    - {id: 2, name: "vlan2_mkt", address: "192.168.12.1/24"}
6    - {id: 3, name: "vlan3_hrd", address: "192.168.13.1/24"}
7    - {id: 4, name: "vlan4_sls", address: "192.168.14.1/24"}
8
```

```

9   lease_time: "3h"
10  dns_server: "8.8.8.8,8.8.4.4"
11
12  dhcp_pools:
13    - {name: "vlan2_mkt", ranges: "192.168.12.2-192.168.12.254",
        network_address: "192.168.12.0/24", gateway:
          "192.168.12.1"}
14    - {name: "vlan3_hrd", ranges: "192.168.13.2-192.168.13.254",
        network_address: "192.168.13.0/24", gateway:
          "192.168.13.1"}
15    - {name: "vlan4_sls", ranges: "192.168.14.2-192.168.14.254",
        network_address: "192.168.14.0/24", gateway:
          "192.168.14.1"}
16
17  bridge_ports:
18    - {interface: "ether1", pvid: 2}
19    - {interface: "ether2", pvid: 2}
20    - {interface: "ether3", pvid: 3}
21    - {interface: "ether4", pvid: 3}
22    - {interface: "ether5", pvid: 4}
23    - {interface: "ether6", pvid: 4}
24
25  vlan_tagging:
26    - {tagged: "ether7", untagged: "ether1,ether2", vlan_ids: 2}
27    - {tagged: "ether7", untagged: "ether3,ether4", vlan_ids: 3}
28    - {tagged: "ether7", untagged: "ether5,ether6", vlan_ids: 4}

```

Penjelasan:

- Baris 1 merupakan deklarasi *variable* **internet_interface** dengan nilai **ether3**.
- Baris 2 merupakan deklarasi *variable* **vlan_interface** dengan nilai **ether1**.

- Baris 4 sampai dengan 7 merupakan deklarasi *variable* bernama “**vlan**” dengan struktur data **list of dictionaries** yang didalamnya memuat 3 (tiga) data untuk pembuatan VLAN, seperti terlihat pada tabel berikut:

Id	name	address
2	vlan2_mkt	192.168.12.1/24
3	vlan3_hrd	192.168.13.1/24
4	vlan4_sls	192.168.14.1/24

- Baris 9 merupakan deklarasi *variable* **lease_time** dengan nilai **3h**.
- Baris 10 merupakan deklarasi *variable* **dns_server** dengan nilai **8.8.8.8,8.8.4.4**.
- Baris 12 sampai dengan 15 merupakan deklarasi *variable* bernama “**dhcp_pools**” dengan struktur data **list of dictionaries** yang didalamnya memuat 3 (tiga) data untuk pembuatan **DHCP** di *node R_Production*, seperti terlihat pada tabel berikut:

Name	ranges	network_address	gateway
vlan2_mkt	192.168.12.2-192.168.12.254	192.168.12.0/24	192.168.12.1
vlan3_hrd	192.168.13.2-192.168.13.254	192.168.13.0/24	192.168.13.1
vlan4_sls	192.168.14.2-192.168.14.254	192.168.14.0/24	192.168.14.1

- Baris 17 sampai dengan 23 merupakan deklarasi *variable* bernama “**bridge_ports**” dengan struktur data **list of dictionaries** yang didalamnya memuat 6 (enam) data untuk pengaturan **Bridge Port** di *node SW_Production*, seperti terlihat pada tabel berikut:

interface	pvid
ether1	2
ether2	2
ether3	3
ether4	3
ether5	4
ether6	4

- Baris 25 sampai dengan 28 merupakan deklarasi *variable* bernama “**vlan_tagging**” dengan struktur data **list of dictionaries** yang didalamnya memuat 3 (tiga) data untuk mengatur

VLAN tagging pada **interface bridge VLAN** dari *node SW_Production* , seperti terlihat pada tabel berikut:

tagged	untagged	vlan_ids
ether7	ether1,ether2	2
ether7	ether3,ether4	3
ether7	ether5,ether6	4

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan tombol **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

- d. Membuat *file* **YAML** dengan nama **main.yml** menggunakan *editor nano* yang merupakan *playbook* utama untuk manajemen penerapan otomatisasi konfigurasi infrastruktur baik pada *node-node* di lingkungan **Test** maupun **Production network**. Di dalam *file playbook* tersebut memuat *task-task* untuk menyisipkan *file variable* dan *file task* yang disimpan secara terpisah terkait konfigurasi **NAT, VLAN** dan **DHCP Server**. Perintah yang dieksekusi adalah **nano main.yml**, seperti terlihat pada gambar berikut:

```
netdevops@gitlab:~/playbook$ nano main.yml
```

Dengan konten sebagai berikut:

```
1  ---
2  - name: Playbook NetDevOps 101 (Create)
3    hosts: localhost
4    connection: network_cli
5    gather_facts: no
6    tasks:
7      - name: include variable
8        include_vars: var_{{ lokasi }}.yaml
9
10     - name: Variable untuk Router di Setiap Lokasi
11       set_fact:
```

```

12         router: "rtr_{{ lokasi }}"
13
14     - name: Task untuk Router
15       include_tasks: router.yml
16       with_items: "{{ groups[router] }}"
17
18     - name: Variable untuk Multilayer Switch di Setiap Lokasi
19       set_fact:
20         msw: "msw_{{ lokasi }}"
21
22     - name: Task untuk Multilayer Switch
23       include_tasks: msw.yml
24       with_items: "{{ groups[msw] }}"

```

Penjelasan:

- Baris 1: `---` (3 *hyphen*) merupakan awal dari dokumen YAML.
- Baris 2: **name** digunakan untuk menentukan nama dari ansible playbook yaitu "**Playbook NetDevOps 101 (Create)**".
- Baris 3: **hosts** digunakan untuk menentukan daftar *host* atau *group host* sebagai lokasi eksekusi task yaitu **localhost**.
- Baris 4: **connection** digunakan untuk mengatur agar *Ansible* memperlakukan *managed machine* sebagai perangkat jaringan dengan lingkungan eksekusi yang terbatas yaitu **network_cli**.
- Baris 5: **gather_facts** digunakan untuk menonaktifkan pengumpulan informasi terkait *remote host* yaitu **no**.
- Baris 6: **tasks** merupakan daftar tindakan atau aksi yang perlu dieksekusi.
- Baris 7: **name** digunakan untuk menentukan nama dari *task* yaitu "**include variable**". Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.

- Baris 8: **include_vars** digunakan untuk memuat *variable* dari *file* yaitu “**var_{{ lokasi }}**.yml”. Penentuan penerapan apakah di lingkungan **Test (var_Test.yml)** atau **Production (var_Production.yml)** berdasarkan nilai yang diambil dari **extra variable** dengan nama **lokasi**. *Variable* di dalam file tersebut diperlukan untuk pembuatan konfigurasi *infrastruktur* di *node* **router** dan **switch** baik untuk lingkungan **Test** maupun **Production network**.
- Baris 10: **name** digunakan untuk menentukan nama dari *task* yaitu “**Variable untuk Router di Setiap Lokasi**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 11: **set_fact** digunakan untuk membuat *variable* baru.
- Baris 22: digunakan untuk mendeklarasikan nama *variable* baru yaitu bernama **router** dengan nilai **rtr_{{ lokasi }}** dimana **{{ lokasi }}** merupakan **extra variable** dengan nama **lokasi**.
- Baris 14: **name** digunakan untuk menentukan nama dari *task* yaitu “**Task untuk Router**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 15: **include_tasks** digunakan untuk menyisipkan *file task* dengan nama “**router.yml**” yang didalamnya memuat instruksi untuk mengatur **DHCP Client** pada *interface*, pembuatan konfigurasi **NAT**, *interface* **VLAN** dan pengalamatan IP pada setiap *interface* VLAN serta **DHCP Server**.
- Baris 16: digunakan untuk melakukan perulangan eksekusi *task* “**Task untuk Router**” sejumlah nilai yang terdapat pada *variable* “**groups[router]**”. **router** adalah *variable* yang telah dideklarasikan sebelumnya.
- Baris 18: **name** digunakan untuk menentukan nama dari *task* yaitu “**Variable untuk Multilayer Switch di Setiap Lokasi**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 19: **set_fact** digunakan untuk membuat *variable* baru.

- Baris 20: digunakan untuk mendeklarasikan nama *variable* baru yaitu bernama **msw** dengan nilai **msw_{{ lokasi }}** dimana **{{ lokasi }}** merupakan **extra variable** dengan nama **lokasi**.
- Baris 22: **name** digunakan untuk menentukan nama dari *task* yaitu “**Task untuk Multilayer Switch**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 23: **include_tasks** digunakan untuk menyisipkan *file task* dengan nama “**msw.yml**” yang didalamnya memuat instruksi untuk mengatur **interface bridge port**, **interface bridge VLAN** dan mengaktifkan **VLAN filtering** pada *interface bridge*.
- Baris 24: digunakan untuk melakukan perulangan eksekusi *task* “**Task untuk Multilayer Switch**” sejumlah nilai yang terdapat pada *variable* “**groups[msw]**”. **msw** merupakan nama *variable* yang telah dideklarasikan sebelumnya.

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan tombol **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

- e. Membuat *file* **YAML** dengan nama **router.yml** menggunakan *editor nano* yang memuat 7 (tujuh) *task* meliputi mengatur **DHCP Client** pada *interface*, membuat **IP Firewall NAT**, membuat *interface* **VLAN**, mengatur pengalamatan IP pada setiap **VLAN**, **IP Pool**, **IP DHCP-Server Network** dan **IP DHCP-Server**. Perintah yang dieksekusi adalah **nano router.yml**, seperti terlihat pada gambar berikut:

```
netdevops@gitlab:~/playbook$ nano router.yml
```

Dengan konten sebagai berikut:

- ```
1 - name: Mengatur DHCP Client pada interface {{
 internet_interface }}
2 routeros_command:
3 commands: /ip dhcp-client add interface={{
 internet_interface }} disabled=no comment={{ lokasi }}
4 delegate_to: "{{ item }}"
5
```

```
6 - name: Membuat IP Firewall NAT
7 routers_command:
8 commands: /ip firewall nat add chain=srcnat out-interface=
9 {{ internet_interface }} action=masquerade comment=
10 {{ lokasi }}
11 delegate_to: "{{ item }}"
12
13 - name: Membuat interface VLAN
14 routers_command:
15 commands:
16 - /interface vlan add interface={{ vlan_interface }}
17 name={{ router_item.name }} vlan-id={{ router_item.id
18 }} comment={{ lokasi }}
19 with_items: "{{ vlans }}"
20 loop_control:
21 loop_var: router_item
22 delegate_to: "{{ item }}"
23
24 - name: Mengatur pengalamatan IP pada setiap interface VLAN
25 routers_command:
26 commands:
27 - /ip address add address={{ router_item.address }}
28 interface={{ router_item.name }} comment={{ lokasi }}
29 with_items: "{{ vlans }}"
30 loop_control:
31 loop_var: router_item
32 delegate_to: "{{ item }}"
33
34 - name: Mengatur IP Pool
```

```

30 routers_command:
31 commands:
32 - /ip pool add name=pool_{{ router_item.name }} ranges={{
 router_item.ranges }} comment={{ lokasi }}
33 with_items: "{{ dhcp_pools }}"
34 loop_control:
35 loop_var: router_item
36 delegate_to: "{{ item }}"
37
38 - name: Mengatur IP DHCP-Server Network
39 routers_command:
40 commands:
41 - /ip dhcp-server network add address=
 {{ router_item.network_address }} dns-server=
 {{ dns_server }} gateway={{ router_item.gateway }}
 comment={{ lokasi }}
42 with_items: "{{ dhcp_pools }}"
43 loop_control:
44 loop_var: router_item
45 delegate_to: "{{ item }}"
46
47 - name: Mengatur IP DHCP-Server
48 routers_command:
49 commands:
50 - /ip dhcp-server add address-pool=pool_{{
 router_item.name }} interface={{ router_item.name }}
 lease-time={{ lease_time }}
 name=dhcp_{{ router_item.name }} comment={{ lokasi }}
51 with_items: "{{ dhcp_pools }}"

```

```

52 loop_control:
53 loop_var: router_item
54 delegate_to: "{{ item }}"

```

Penjelasan:

- Baris 1: **name** digunakan untuk menentukan nama dari *task* yaitu “**Mengatur DHCP Client pada interface {{ internet\_interface }}**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi. **{{ internet\_interface }}** digunakan untuk mengambil nilai dari *variable* dengan nama **internet\_interface**.
- Baris 2: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 3: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi. Perintahnya adalah **/ip dhcp-client add interface={{ internet\_interface }} disabled=no comment={{ lokasi }}**.  
**{{ internet\_interface }}** digunakan untuk mengambil nilai dari *variable* dengan nama **internet\_interface**. Sedangkan **{{ lokasi }}** digunakan untuk mengambil nilai dari extra *variable* dengan nama **lokasi**.
- Baris 4: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable* **item**.
- Baris 6: **name** digunakan untuk menentukan nama dari *task* yaitu “**Membuat IP Firewall NAT**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 7: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 8: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi. Perintahnya adalah **/ip firewall nat add chain=srcnat out-interface={{ internet\_interface }} action=masquerade comment={{ lokasi }}**.

`{{ internet_interface }}` digunakan untuk mengambil nilai dari *variable* dengan nama **internet\_interface**. Sedangkan `{{ lokasi }}` digunakan untuk mengambil nilai dari extra *variable* dengan nama **lokasi**.

- Baris 9: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable* **item**.
- Baris 11: **name** digunakan untuk menentukan nama dari *task* yaitu “**Membuat interface VLAN**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 12: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 13: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi.
- Baris 14: merupakan nilai dari parameter **commands** yaitu bertipe *list* yang memuat perintah `/interface vlan add interface={{ vlan_interface }} name={{ router_item.name }} vlan-id={{ router_item.id }} comment={{ lokasi }}`.  
`{{ vlan_interface }}` digunakan untuk mengambil nilai dari *variable* **vlan\_interface**. `{{ router_item.name }}` digunakan untuk mengambil nilai dari *variable* **router\_item** dengan **key name**. `{{ router_item.id }}` digunakan untuk mengambil nilai dari *variable* **router\_item** dengan **key id**. Sedangkan `{{ lokasi }}` digunakan untuk mengambil nilai dari extra *variable* dengan nama **lokasi**.
- Baris 15: **with\_items** digunakan untuk melakukan perulangan eksekusi *task* “**Membuat interface VLAN**” sejumlah nilai yang terdapat dalam *variable* “**vlangs**”.
- Baris 16 dan 17: **loop\_control** dengan parameter **loop\_var** berfungsi untuk menentukan nama *variable* yang digunakan ketika perulangan yaitu **router\_item**.
- Baris 18: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable* **item**.

- Baris 20: **name** digunakan untuk menentukan nama dari *task* yaitu “**Mengatur pengalamatan IP pada setiap interface VLAN**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 21: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 22: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi.
- Baris 23: merupakan nilai dari parameter **commands** yaitu bertipe *list* yang memuat perintah `/ip address add address={{ router_item.address }} interface={{ router_item.name }} comment={{ lokasi }}`.  
`{{ router_item.address }}` digunakan untuk mengambil nilai dari *variable* **router\_item** dengan **key** **address**. `{{ router_item.name }}` digunakan untuk mengambil nilai dari *variable* **router\_item** dengan **key** **name**. Sedangkan `{{ lokasi }}` digunakan untuk mengambil nilai extra *variable* dengan nama **lokasi**.
- Baris 24: **with\_items** digunakan untuk melakukan perulangan eksekusi *task* “**Mengatur pengalamatan IP pada setiap interface VLAN**” sejumlah nilai yang terdapat dalam *variable* “**vlangs**”.
- Baris 25 dan 26: **loop\_control** dengan parameter **loop\_var** berfungsi untuk menentukan nama *variable* yang digunakan ketika perulangan yaitu **router\_item**.
- Baris 27: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable* **item**.
- Baris 29: **name** digunakan untuk menentukan nama dari *task* yaitu “**Mengatur IP Pool**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 30: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 31: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi.

- Baris 32: merupakan nilai dari parameter **commands** yaitu bertipe *list* yang memuat perintah `/ip pool add name=pool_{{ router_item.name }} ranges={{ router_item.ranges }} comment={{ lokasi }}`.  
`{{ router_item.name }}` digunakan untuk mengambil nilai dari *variable* **router\_item** dengan **key** `name`. `{{ router_item.ranges }}` digunakan untuk mengambil nilai dari *variable* **router\_item** dengan **key** `ranges`. Sedangkan `{{ lokasi }}` digunakan untuk mengambil nilai extra *variable* dengan nama **lokasi**.
- Baris 33: **with\_items** digunakan untuk melakukan perulangan eksekusi *task* “**Mengatur IP Pools**” sejumlah nilai yang terdapat dalam *variable* “**dhcp\_pools**”.
- Baris 34 dan 35: **loop\_control** dengan parameter **loop\_var** berfungsi untuk menentukan nama *variable* yang digunakan ketika perulangan yaitu **router\_item**.
- Baris 36: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable* **item**.
- Baris 38: **name** digunakan untuk menentukan nama dari *task* yaitu “**Mengatur IP DHCP-Server Network**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 39: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 40: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi.
- Baris 41: merupakan nilai dari parameter **commands** yaitu bertipe *list* yang memuat perintah `/ip dhcp-server network add address={{ router_item.network_address }} dns-server={{ dns_server }} gateway={{ router_item.gateway }} comment={{ lokasi }}`.  
`{{ router_item.network_address }}` digunakan untuk mengambil nilai dari *variable* **router\_item** dengan **key** `network_address`. `{{ dns_server }}` digunakan untuk mengambil nilai dari *variable* **dns\_server**. `{{ router_item.gateway }}` digunakan untuk mengambil nilai dari *variable* **router\_item** dengan **key** `gateway`.

Sedangkan `{{ lokasi }}` digunakan untuk mengambil nilai extra *variable* dengan nama **lokasi**.

- Baris 42: **with\_items** digunakan untuk melakukan perulangan eksekusi *task* “**Mengatur IP DHCP-Server Network**” sejumlah nilai yang terdapat dalam *variable* “**dhcp\_pools**”.
- Baris 43 dan 44: **loop\_control** dengan parameter **loop\_var** berfungsi untuk menentukan nama *variable* yang digunakan ketika perulangan yaitu **router\_item**.
- Baris 45: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable* **item**.
- Baris 47: **name** digunakan untuk menentukan nama dari *task* yaitu “**Mengatur IP DHCP-Server**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 48: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 49: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi.
- Baris 50: merupakan nilai dari parameter **commands** yaitu bertipe *list* yang memuat perintah `/ip dhcp-server add address-pool=pool_{{ router_item.name }} interface={{ router_item.name }} lease-time={{ lease_time }} name=dhcp_{{ router_item.name }} comment={{ lokasi }}`.  
`{{ router_item.name }}` digunakan untuk mengambil nilai dari *variable* **router\_item** dengan **key name**. `{{ lease_time }}` digunakan untuk mengambil nilai dari *variable* **lease\_time**. Sedangkan `{{ lokasi }}` digunakan untuk mengambil nilai extra *variable* dengan nama **lokasi**.
- Baris 51: **with\_items** digunakan untuk melakukan perulangan eksekusi *task* “**Mengatur IP DHCP-Server**” sejumlah nilai yang terdapat dalam *variable* “**dhcp\_pools**”.
- Baris 52 dan 53: **loop\_control** dengan parameter **loop\_var** berfungsi untuk menentukan nama *variable* yang digunakan ketika perulangan yaitu **router\_item**.

- Baris 54: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable item*.

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan tombol **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

- f. Membuat *file* YAML dengan nama **msw.yml** menggunakan *editor nano* yang memuat 3 (tiga) *task* meliputi mengatur *interface bridge port*, *interface bridge VLAN* dan mengaktifkan *VLAN filtering* pada *interface bridge*. Perintah yang dieksekusi adalah **nano msw.yml**, seperti terlihat pada gambar:

```
netdevops@gitlab:~/playbook$ nano msw.yml
```

Dengan konten sebagai berikut:

```
1 - name: Mengatur interface bridge port
2 routeros_command:
3 commands:
4 - /interface bridge port add bridge=BR_{{ lokasi }}
5 interface={{ msw_item.interface }}
6 pvid={{ msw_item.pvid }} comment={{ lokasi }}
7 with_items: "{{ bridge_ports }}"
8 loop_control:
9 loop_var: msw_item
10 delegate_to: "{{ item }}"
11
12 - name: Mengatur interface bridge VLAN
13 routeros_command:
14 commands:
15 - /interface bridge vlan add bridge=BR_{{ lokasi }}
16 tagged={{ msw_item.tagged }}
17 untagged={{ msw_item.untagged }}
18 vlan-ids={{ msw_item.vlan_ids }} comment={{ lokasi }}
19 with_items: "{{ vlan_tagging }}"
```

```

15 loop_control:
16 loop_var: msw_item
17 delegate_to: "{{ item }}"
18
19 - name: Mengaktifkan vlan filtering pada interface bridge
20 routeros_command:
21 commands: /interface bridge set vlan-filtering=yes
22 BR_{{ lokasi }}
23 delegate_to: "{{ item }}"

```

Penjelasan:

- Baris 1: **name** digunakan untuk menentukan nama dari *task* yaitu “**Mengatur interface bridge port**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 2: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 3: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi.
- Baris 4: merupakan nilai dari parameter **commands** yaitu bertipe *list* yang memuat perintah `/interface bridge port add bridge=BR_{{ lokasi }} interface={{ msw_item.interface }} pvid={{ msw_item.pvid }} comment={{ lokasi }}`.  
 {{ lokasi }} digunakan untuk mengambil nilai dari extra *variable* dengan nama **lokasi**. {{ msw\_item.interface }} digunakan untuk mengambil nilai dari *variable* **msw\_item** dengan **key interface**. Sedangkan {{ msw\_item.pvid }} digunakan untuk mengambil nilai dari *variable* **msw\_item** dengan **key pvid**.
- Baris 5: **with\_items** digunakan untuk melakukan perulangan eksekusi *task* “**Mengatur interface bridge port**” sejumlah nilai yang terdapat dalam *variable* “**bridge\_ports**”.

- Baris 6 dan 7: **loop\_control** dengan parameter **loop\_var** berfungsi untuk menentukan nama *variable* yang digunakan ketika perulangan yaitu **msw\_item**.
- Baris 8: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable item*.
- Baris 10: **name** digunakan untuk menentukan nama dari *task* yaitu “**Mengatur interface bridge VLAN**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 11: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 12: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi.
- Baris 13: merupakan nilai dari parameter **commands** yaitu bertipe *list* yang memuat perintah `/interface bridge vlan add bridge=BR_{{ lokasi }} tagged={{ msw_item.tagged }} untagged={{ msw_item.untagged }} vlan-ids={{ msw_item.vlan_ids }} comment={{ lokasi }}`.  
 {{ lokasi }} digunakan untuk mengambil nilai dari extra *variable* dengan nama **lokasi**. {{ msw\_item.tagged }} digunakan untuk mengambil nilai dari *variable msw\_item* dengan **key tagged**. {{ msw\_item.untagged }} digunakan untuk mengambil nilai dari *variable msw\_item* dengan **key untagged**. Sedangkan {{ msw\_item.vlan\_ids }} digunakan untuk mengambil nilai dari *variable msw\_item* dengan **key vlan\_ids**.
- Baris 14: **with\_items** digunakan untuk melakukan perulangan eksekusi *task* “**Mengatur interface bridge VLAN**” sejumlah nilai yang terdapat dalam *variable* “**vlan\_tagging**”.
- Baris 15 dan 16: **loop\_control** dengan parameter **loop\_var** berfungsi untuk menentukan nama *variable* yang digunakan ketika perulangan yaitu **msw\_item**.
- Baris 17: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable item*.

- Baris 19: **name** digunakan untuk menentukan nama dari *task* yaitu “**Mengaktifkan vlan filtering pada interface bridge**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 20: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 21: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi. Perintahnya adalah **/interface bridge set vlan-filtering=yes BR\_{{ lokasi }}** dimana **{{ lokasi }}** digunakan untuk mengambil nilai extra *variable* dengan nama **lokasi**.
- Baris 22: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable item*.

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan tombol **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

- g. Menguji coba eksekusi *Ansible playbook* **main.yml** untuk memproses penerapan konfigurasi di lingkungan **Test network** dengan perintah **ansible-playbook -i hosts main.yml -e lokasi=Test**.

```
netdevops@gitlab:~/playbook$ ansible-playbook -i hosts main.yml -e lokasi=Test
```

Hasil eksekusi dari perintah tersebut, seperti terlihat pada gambar berikut:

```

192.168.0.10:tlp - netdevops@192.168.0.10:2222 - Bitwise xterm - netdevops@gitlab: ~/playbook
netdevops@gitlab:~/playbook$ ansible-playbook -i hosts main.yml -e lokasi=Test

PLAY [Playbook NetDevOps 101 (Create)] *****

TASK [include variable] *****
ok: [localhost]

TASK [Variable untuk Router di Setiap Lokasi] *****
ok: [localhost]

TASK [Task untuk Router] *****
included: /home/netdevops/playbook/router.yml for localhost => (item=R_Test)

TASK [Mengatur DHCP Client pada interface ether3] *****
ok: [localhost -> 192.168.0.1]

TASK [Membuat IP Firewall NAT] *****
ok: [localhost -> 192.168.0.1]

TASK [Membuat interface VLAN] *****
ok: [localhost -> 192.168.0.1] => (item={'id': 2, 'name': 'vlan2_mkt', 'address': '192.168.2.1/24'})
ok: [localhost -> 192.168.0.1] => (item={'id': 3, 'name': 'vlan3_hrd', 'address': '192.168.3.1/24'})
ok: [localhost -> 192.168.0.1] => (item={'id': 4, 'name': 'vlan4_sls', 'address': '192.168.4.1/24'})

TASK [Mengatur pengalamatan IP pada setiap interface VLAN] *****
ok: [localhost -> 192.168.0.1] => (item={'id': 2, 'name': 'vlan2_mkt', 'address': '192.168.2.1/24'})
ok: [localhost -> 192.168.0.1] => (item={'id': 3, 'name': 'vlan3_hrd', 'address': '192.168.3.1/24'})
ok: [localhost -> 192.168.0.1] => (item={'id': 4, 'name': 'vlan4_sls', 'address': '192.168.4.1/24'})

TASK [Mengatur IP Pool] *****
ok: [localhost -> 192.168.0.1] => (item={'name': 'vlan2_mkt', 'ranges': '192.168.2.2-192.168.2.254',
'network_address': '192.168.2.0/24', 'gateway': '192.168.2.1'})
ok: [localhost -> 192.168.0.1] => (item={'name': 'vlan3_hrd', 'ranges': '192.168.3.2-192.168.3.254',
'network_address': '192.168.3.0/24', 'gateway': '192.168.3.1'})
ok: [localhost -> 192.168.0.1] => (item={'name': 'vlan4_sls', 'ranges': '192.168.4.2-192.168.4.254',
'network_address': '192.168.4.0/24', 'gateway': '192.168.4.1'})

TASK [Mengatur IP DHCP-Server Network] *****
ok: [localhost -> 192.168.0.1] => (item={'name': 'vlan2_mkt', 'ranges': '192.168.2.2-192.168.2.254',
'network_address': '192.168.2.0/24', 'gateway': '192.168.2.1'})
ok: [localhost -> 192.168.0.1] => (item={'name': 'vlan3_hrd', 'ranges': '192.168.3.2-192.168.3.254',
'network_address': '192.168.3.0/24', 'gateway': '192.168.3.1'})
ok: [localhost -> 192.168.0.1] => (item={'name': 'vlan4_sls', 'ranges': '192.168.4.2-192.168.4.254',
'network_address': '192.168.4.0/24', 'gateway': '192.168.4.1'})

TASK [Mengatur IP DHCP-Server] *****
ok: [localhost -> 192.168.0.1] => (item={'name': 'vlan2_mkt', 'ranges': '192.168.2.2-192.168.2.254',
'network_address': '192.168.2.0/24', 'gateway': '192.168.2.1'})
ok: [localhost -> 192.168.0.1] => (item={'name': 'vlan3_hrd', 'ranges': '192.168.3.2-192.168.3.254',
'network_address': '192.168.3.0/24', 'gateway': '192.168.3.1'})
ok: [localhost -> 192.168.0.1] => (item={'name': 'vlan4_sls', 'ranges': '192.168.4.2-192.168.4.254',
'network_address': '192.168.4.0/24', 'gateway': '192.168.4.1'})

TASK [Variable untuk Multilayer Switch di Setiap Lokasi] *****
ok: [localhost]

TASK [Task untuk Multilayer Switch] *****
included: /home/netdevops/playbook/msw.yml for localhost => (item=SW_Test)

```

```

TASK [Mengatur interface bridge port] *****
ok: [localhost -> 192.168.1.2] => (item={'interface': 'ether1', 'pvid': 2})
ok: [localhost -> 192.168.1.2] => (item={'interface': 'ether2', 'pvid': 2})
ok: [localhost -> 192.168.1.2] => (item={'interface': 'ether3', 'pvid': 3})
ok: [localhost -> 192.168.1.2] => (item={'interface': 'ether4', 'pvid': 3})
ok: [localhost -> 192.168.1.2] => (item={'interface': 'ether5', 'pvid': 4})
ok: [localhost -> 192.168.1.2] => (item={'interface': 'ether6', 'pvid': 4})

TASK [Mengatur interface bridge VLAN] *****
ok: [localhost -> 192.168.1.2] => (item={'tagged': 'ether7', 'untagged': 'ether1,ether2', 'vlan_ids': 2})
ok: [localhost -> 192.168.1.2] => (item={'tagged': 'ether7', 'untagged': 'ether3,ether4', 'vlan_ids': 3})
ok: [localhost -> 192.168.1.2] => (item={'tagged': 'ether7', 'untagged': 'ether5,ether6', 'vlan_ids': 4})

TASK [Mengaktifkan vlan filtering pada interface bridge] *****
ok: [localhost -> 192.168.1.2]

PLAY RECAP *****
localhost : ok=15 changed=0 unreachable=0 failed=0 skipped=0 rescued=0
 0 ignored=0

```

- h. Mengujicoba eksekusi *Ansible playbook* **main.yml** untuk memproses penerapan konfigurasi di lingkungan **Production network** dengan perintah **ansible-playbook -i hosts main.yml -e lokasi=Production**.

```
netdevops@gitlab:~/playbook$ ansible-playbook -i hosts main.yml -e lokasi=Production
```

Hasil eksekusi dari perintah tersebut, seperti terlihat pada gambar berikut:

```

192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~/playbook
netdevops@gitlab:~/playbook$ ansible-playbook -i hosts main.yml -e lokasi=Production

PLAY [Playbook NetDevOps 101 (Create)] *****

TASK [include variable] *****
ok: [localhost]

TASK [Variable untuk Router di Setiap Lokasi] *****
ok: [localhost]

TASK [Task untuk Router] *****
included: /home/netdevops/playbook/router.yml for localhost => (item=R_Production)

TASK [Mengatur DHCP Client pada interface ether3] *****
ok: [localhost -> 192.168.0.6]

TASK [Membuat IP Firewall NAT] *****
ok: [localhost -> 192.168.0.6]

TASK [Membuat interface VLAN] *****
ok: [localhost -> 192.168.0.6] => (item={'id': 2, 'name': 'vlan2_mkt', 'address': '192.168.12.1/24'})
ok: [localhost -> 192.168.0.6] => (item={'id': 3, 'name': 'vlan3_hrd', 'address': '192.168.13.1/24'})
ok: [localhost -> 192.168.0.6] => (item={'id': 4, 'name': 'vlan4_sls', 'address': '192.168.14.1/24'})

```

```

TASK [Mengatur pengalamatan IP pada setiap interface VLAN] *****
ok: [localhost -> 192.168.0.6] => (item={'id': 2, 'name': 'vlan2_mkt', 'address': '192.168.12.1/24'})
ok: [localhost -> 192.168.0.6] => (item={'id': 3, 'name': 'vlan3_hrd', 'address': '192.168.13.1/24'})
ok: [localhost -> 192.168.0.6] => (item={'id': 4, 'name': 'vlan4_sls', 'address': '192.168.14.1/24'})

TASK [Mengatur IP Pool] *****
ok: [localhost -> 192.168.0.6] => (item={'name': 'vlan2_mkt', 'ranges': '192.168.12.2-192.168.12.254', 'network_address': '192.168.12.0/24', 'gateway': '192.168.12.1'})
ok: [localhost -> 192.168.0.6] => (item={'name': 'vlan3_hrd', 'ranges': '192.168.13.2-192.168.13.254', 'network_address': '192.168.13.0/24', 'gateway': '192.168.13.1'})
ok: [localhost -> 192.168.0.6] => (item={'name': 'vlan4_sls', 'ranges': '192.168.14.2-192.168.14.254', 'network_address': '192.168.14.0/24', 'gateway': '192.168.14.1'})

TASK [Mengatur IP DHCP-Server Network] *****
ok: [localhost -> 192.168.0.6] => (item={'name': 'vlan2_mkt', 'ranges': '192.168.12.2-192.168.12.254', 'network_address': '192.168.12.0/24', 'gateway': '192.168.12.1'})
ok: [localhost -> 192.168.0.6] => (item={'name': 'vlan3_hrd', 'ranges': '192.168.13.2-192.168.13.254', 'network_address': '192.168.13.0/24', 'gateway': '192.168.13.1'})
ok: [localhost -> 192.168.0.6] => (item={'name': 'vlan4_sls', 'ranges': '192.168.14.2-192.168.14.254', 'network_address': '192.168.14.0/24', 'gateway': '192.168.14.1'})

TASK [Mengatur IP DHCP-Server] *****
ok: [localhost -> 192.168.0.6] => (item={'name': 'vlan2_mkt', 'ranges': '192.168.12.2-192.168.12.254', 'network_address': '192.168.12.0/24', 'gateway': '192.168.12.1'})
ok: [localhost -> 192.168.0.6] => (item={'name': 'vlan3_hrd', 'ranges': '192.168.13.2-192.168.13.254', 'network_address': '192.168.13.0/24', 'gateway': '192.168.13.1'})
ok: [localhost -> 192.168.0.6] => (item={'name': 'vlan4_sls', 'ranges': '192.168.14.2-192.168.14.254', 'network_address': '192.168.14.0/24', 'gateway': '192.168.14.1'})

TASK [Variable untuk Multilayer Switch di Setiap Lokasi] *****
ok: [localhost]

TASK [Task untuk Multilayer Switch] *****
included: /home/netdevops/playbook/msw.yml for localhost => (item=SW_Production)

TASK [Mengatur interface bridge port] *****
ok: [localhost -> 192.168.11.2] => (item={'interface': 'ether1', 'pvid': 2})
ok: [localhost -> 192.168.11.2] => (item={'interface': 'ether2', 'pvid': 2})
ok: [localhost -> 192.168.11.2] => (item={'interface': 'ether3', 'pvid': 3})
ok: [localhost -> 192.168.11.2] => (item={'interface': 'ether4', 'pvid': 3})
ok: [localhost -> 192.168.11.2] => (item={'interface': 'ether5', 'pvid': 4})
ok: [localhost -> 192.168.11.2] => (item={'interface': 'ether6', 'pvid': 4})

TASK [Mengatur interface bridge VLAN] *****
ok: [localhost -> 192.168.11.2] => (item={'tagged': 'ether7', 'untagged': 'ether1,ether2', 'vlan_ids': 2})
ok: [localhost -> 192.168.11.2] => (item={'tagged': 'ether7', 'untagged': 'ether3,ether4', 'vlan_ids': 3})
ok: [localhost -> 192.168.11.2] => (item={'tagged': 'ether7', 'untagged': 'ether5,ether6', 'vlan_ids': 4})

TASK [Mengaktifkan vlan filtering pada interface bridge] *****
ok: [localhost -> 192.168.11.2]

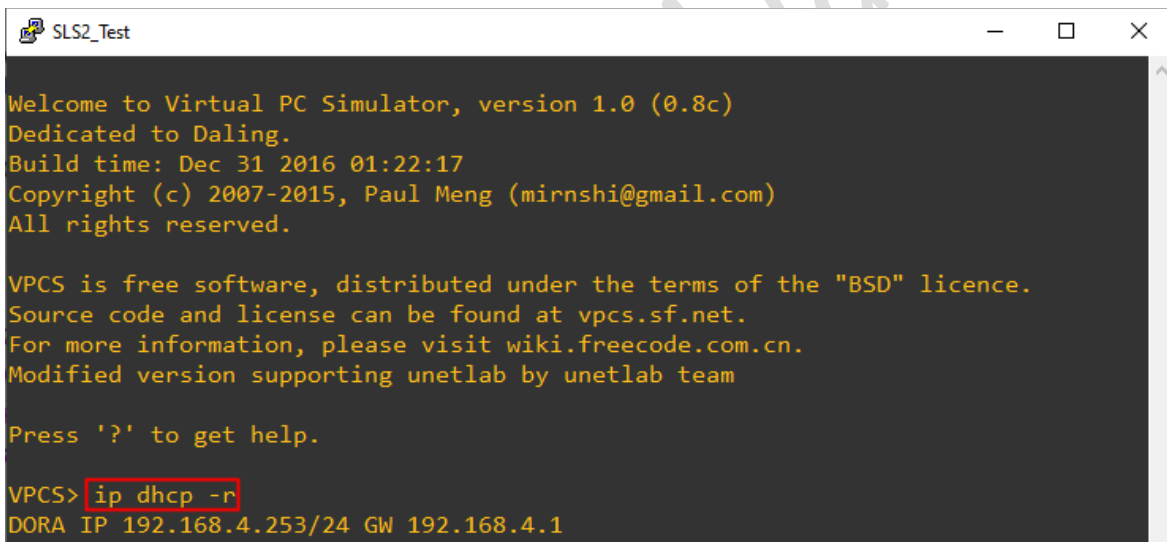
PLAY RECAP *****
localhost : ok=15 changed=0 unreachable=0 failed=0 skipped=0 rescued=0
0 ignored=0

```

## H. Memverifikasi Hasil Proses Otomatisasi Penerapan Konfigurasi NAT, VLAN Dan DHCP Server Pada Lab NETDEVOPS Di PNETLAB

Adapun langkah-langkah untuk memverifikasi hasil dari proses otomatisasi penerapan konfigurasi NAT, VLAN dan DHCP Server pada lab **NetDevOps** di PNETLab adalah sebagai berikut:

1. Berpindah ke tab **PNETLab** pada *browser* yang digunakan.
2. Melakukan **DHCP Request** ulang pada setiap *node VPCS* yang terdapat di lingkungan **test network** dari PNETLAB yaitu **MKT1\_Test**, **MKT2\_Test**, **HRD1\_Test**, **HRD2\_Test** dan **SLS1\_Test** serta **SLS2\_Test** dengan mengeksekusi perintah `ip dhcp -r` pada *console* dari *node* VPCS tersebut. Pastikan setiap **VPCS** telah berhasil memperoleh pengalamatan IP secara dinamis dari **DHCP Server**. Terlampir cuplikan hasil eksekusi dari perintah tersebut pada salah satu VPCS yaitu **SLS2\_Test**.



```
Welcome to Virtual PC Simulator, version 1.0 (0.8c)
Dedicated to Daling.
Build time: Dec 31 2016 01:22:17
Copyright (c) 2007-2015, Paul Meng (mirnshi@gmail.com)
All rights reserved.

VPCS is free software, distributed under the terms of the "BSD" licence.
Source code and license can be found at vpcs.sf.net.
For more information, please visit wiki.freecode.com.cn.
Modified version supporting unetlab by unetlab team

Press '?' to get help.

VPCS> ip dhcp -r
DORA IP 192.168.4.253/24 GW 192.168.4.1
```

Terlihat **SLS2\_Test** memperoleh alamat IP **192.168.4.253/24**.

3. Melakukan verifikasi pengalamatan IP yang telah disewakan oleh *router R\_Test* ke *client VPCS* dari setiap VLAN dengan mengeksekusi perintah `ip dhcp-server lease print` pada *console* dari *node R\_Test*.

```

R_Test
[admin@R_Test] > ip dhcp-server lease print
Flags: D, B - BLOCKED
Columns: ADDRESS, MAC-ADDRESS, HOST-NAME, SERVER, STATUS, LAST-SEEN
ADDRESS MAC-ADDRESS HOST-NAME SERVER STATUS LAST-SEEN
0 D 192.168.2.254 00:50:79:66:68:06 VPCS1 dhcp_vlan2_mkt bound 6m16s
1 D 192.168.2.253 00:50:79:66:68:07 VPCS1 dhcp_vlan2_mkt bound 6m6s
2 D 192.168.3.254 00:50:79:66:68:08 VPCS1 dhcp_vlan3_hrd bound 5m55s
3 D 192.168.3.253 00:50:79:66:68:09 VPCS1 dhcp_vlan3_hrd bound 5m27s
4 D 192.168.4.254 00:50:79:66:68:0A VPCS1 dhcp_vlan4_sls bound 5m17s
5 D 192.168.4.253 00:50:79:66:68:0B VPCS1 dhcp_vlan4_sls bound 5m5s

```

4. Melakukan **DHCP Request** ulang pada setiap *node VPCS* yang terdapat di lingkungan **production network** dari PNETLAB yaitu **MKT1\_Production**, **MKT2\_Production**, **HRD1\_Production**, **HRD2\_Production** dan **SLS1\_Production** serta **SLS2\_Production** dengan mengeksekusi perintah `ip dhcp -r` pada *console* dari *node VPCS* tersebut. Pastikan setiap **VPCS** telah berhasil memperoleh pengalamatan IP secara dinamis dari **DHCP Server**. Terlampir cuplikan hasil eksekusi dari perintah tersebut pada salah satu VPCS yaitu **SLS2\_Test**.

```

192.168.126.129 - PuTTY
Welcome to Virtual PC Simulator, version 1.0 (0.8c)
Dedicated to Daling.
Build time: Dec 31 2016 01:22:17
Copyright (c) 2007-2015, Paul Meng (mirnshi@gmail.com)
All rights reserved.

VPCS is free software, distributed under the terms of the "BSD" licence.
Source code and license can be found at vpcs.sf.net.
For more information, please visit wiki.freecode.com.cn.
Modified version supporting unetlab by unetlab team

Press '?' to get help.

VPCS> ip dhcp -r
DORA IP 192.168.14.253/24 GW 192.168.14.1

```

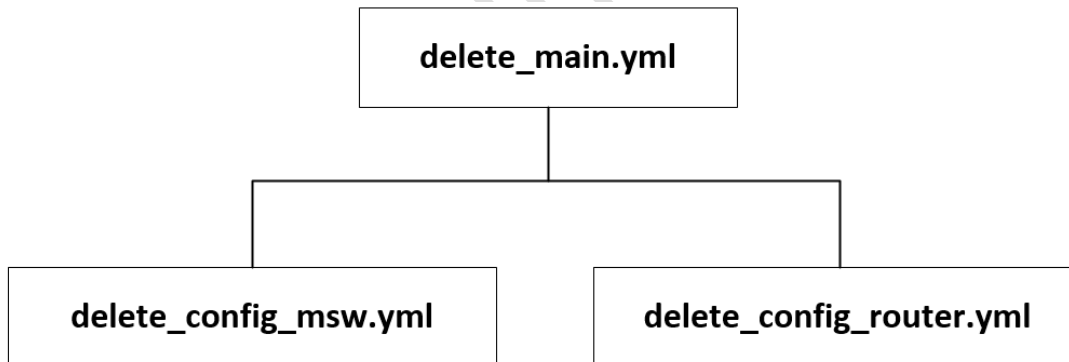
Terlihat **SLS2\_Production** memperoleh alamat IP **192.168.14.253/24**.

5. Melakukan verifikasi pengalamatan IP yang telah disewakan oleh *router R\_Production* ke **client VPCS** dari setiap VLAN dengan mengeksekusi perintah `ip dhcp-server lease print` pada *console* dari *node R\_Production*.

```
[admin@R_Production] > ip dhcp-server lease print
Flags: D, B - BLOCKED
Columns: ADDRESS, MAC-ADDRESS, HOST-NAME, SERVER, STATUS, LAST-SEEN
ADDRESS MAC-ADDRESS HOST-NAME SERVER STATUS LAST-SEEN
0 D 192.168.12.254 00:50:79:66:68:0C VPCS1 dhcp_vlan2_mkt bound 1m52s
1 D 192.168.12.253 00:50:79:66:68:0D VPCS1 dhcp_vlan2_mkt bound 1m37s
2 D 192.168.13.254 00:50:79:66:68:0E VPCS1 dhcp_vlan3_hrd bound 1m22s
3 D 192.168.13.253 00:50:79:66:68:0F VPCS1 dhcp_vlan3_hrd bound 1m16s
4 D 192.168.14.254 00:50:79:66:68:10 VPCS1 dhcp_vlan4_sls bound 1m11s
5 D 192.168.14.253 00:50:79:66:68:11 VPCS1 dhcp_vlan4_sls bound 1m5s
```

## I. Ansible Playbook Untuk Mengotomatisasi Penghapusan Konfigurasi NAT, VLAN, Dan DHCP Server Pada Lab NETDEVOPS

Terdapat 3 (tiga) *file* YAML yang terlibat untuk mengotomatisasi penghapusan konfigurasi NAT, VLAN dan DHCP Server pada lab **NetDevOps** yaitu **delete\_main.yml**, **delete\_config\_msw.yml** dan **delete\_config\_router.yml**. Adapun struktur hirarki dari ketiga *file* tersebut, seperti terlihat pada gambar berikut:



Adapun langkah-langkah pembuatan ketiga *file* tersebut dan eksekusi *file Ansible playbook* untuk menghapus konfigurasi pada lab *NetDevOps* adalah sebagai berikut:

- Mengakses **Bitwise xterm** dari **VM GitLab NetDevops**.
- Membuat *file* YAML dengan nama **delete\_main.yml** menggunakan *editor nano* yang merupakan *file playbook* utama untuk mengotomatisasi penghapusan konfigurasi infrastruktur baik pada *node-node* di lingkungan **Test** maupun **Production network**. Di dalam *file playbook* tersebut memuat *task-task* untuk menyisipkan *file variable* dan *file task* yang

disimpan secara terpisah terkait penghapusan konfigurasi **NAT**, **VLAN** dan **DHCP Server**. Perintah yang dieksekusi adalah **nano delete\_main.yml**, seperti terlihat pada gambar berikut:

```
netdevops@gitlab:~/playbook$ nano delete_main.yml
```

Dengan konten sebagai berikut:

```
1 ---
2 - name: Playbook NetDevOps 101 (Delete)
3 hosts: localhost
4 connection: network_cli
5 gather_facts: no
6 tasks:
7 - name: include variable
8 include_vars: var_{{ lokasi }}.yaml
9
10 - name: Variable untuk Router di Setiap Lokasi
11 set_fact:
12 router: "rtr_{{ lokasi }}"
13
14 - name: Task untuk Router
15 include_tasks: delete_config_router.yml
16 with_items: "{{ groups[router] }}"
17
18 - name: Variable untuk Multilayer Switch di Setiap Lokasi
19 set_fact:
20 msw: "msw_{{ lokasi }}"
21
22 - name: Task untuk Multilayer Switch
23 include_tasks: delete_config_msw.yml
```

24      `with_items: "{{ groups[msw] }}"`

Penjelasan:

- Baris 1: `---` (3 *hyphen*) merupakan awal dari dokumen YAML.
- Baris 2: **name** digunakan untuk menentukan nama dari ansible playbook yaitu "**Playbook NetDevOps 101 (Delete)**".
- Baris 3: **hosts** digunakan untuk menentukan daftar *host* atau *group host* sebagai lokasi eksekusi task yaitu **localhost**.
- Baris 4: *connection* digunakan untuk mengatur agar *Ansible* memperlakukan *managed machine* sebagai perangkat jaringan dengan lingkungan eksekusi yang terbatas yaitu **network\_cli**.
- Baris 5: **gather\_facts** digunakan untuk menonaktifkan pengumpulan informasi terkait *remote host* yaitu **no**.
- Baris 6: **tasks** merupakan daftar tindakan atau aksi yang perlu dieksekusi.
- Baris 7: **name** digunakan untuk menentukan nama dari *task* yaitu "**include variable**". Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 8: **include\_vars** digunakan untuk memuat *variable* dari *file* yaitu "**var\_{{ lokasi }}**.yml". Penentuan penerapan apakah di lingkungan **Test (var\_Test.yml)** atau **Production (var\_Production.yml)** berdasarkan nilai yang diambil dari **extra variable** dengan nama **lokasi**. *Variable* di dalam file tersebut diperlukan untuk pembuatan konfigurasi *infrastruktur* di *node* **router** dan **switch** baik untuk lingkungan **Test** maupun **Production network**.
- Baris 10: **name** digunakan untuk menentukan nama dari *task* yaitu "**Variable untuk Router di Setiap Lokasi**". Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 11: **set\_fact** digunakan untuk membuat *variable* baru.
- Baris 22: digunakan untuk mendeklarasikan nama *variable* baru yaitu bernama **router** dengan nilai **rtr\_{{ lokasi }}** dimana **{{ lokasi }}** merupakan **extra variable** dengan nama **lokasi**.

- Baris 14: **name** digunakan untuk menentukan nama dari **task** yaitu “**Task untuk Router**”. Di dalam setiap **task** memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 15: **include\_tasks** digunakan untuk menyisipkan *file task* dengan nama “**delete\_config\_router.yml**” yang didalamnya memuat instruksi untuk menghapus pengaturan **DHCP Client** pada *interface*, konfigurasi **NAT**, *interface* **VLAN** dan pengalamatan IP pada setiap *interface* VLAN serta **DHCP Server**.
- Baris 16: digunakan untuk melakukan perulangan eksekusi *task* “**Task untuk Router**” sejumlah nilai yang terdapat pada *variable* “**groups[router]**”. **router** adalah *variable* yang telah dideklarasikan sebelumnya.
- Baris 18: **name** digunakan untuk menentukan nama dari *task* yaitu “**Variable untuk Multilayer Switch di Setiap Lokasi**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 19: **set\_fact** digunakan untuk membuat *variable* baru.
- Baris 20: digunakan untuk mendeklarasikan nama *variable* baru yaitu bernama **msw** dengan nilai **msw\_{{ lokasi }}** dimana **{{ lokasi }}** merupakan **extra variable** dengan nama **lokasi**.
- Baris 22: **name** digunakan untuk menentukan nama dari *task* yaitu “**Task untuk Multilayer Switch**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 23: **include\_tasks** digunakan untuk menyisipkan *file task* dengan nama “**delete\_config\_msw.yml**” yang didalamnya memuat instruksi untuk menonaktifkan **VLAN filtering** pada *interface bridge*, menghapus **interface bridge port** dan **interface bridge VLAN**.
- Baris 24: digunakan untuk melakukan perulangan eksekusi *task* “**Task untuk Multilayer Switch**” sejumlah nilai yang terdapat pada *variable* “**groups[msw]**”. **msw** merupakan nama *variable* yang telah dideklarasikan sebelumnya.

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan tombol **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

- c. Membuat *file* YAML dengan nama **delete\_config\_router.yml** yang merupakan *file task* yang didalamnya memuat *task* meliputi menghapus pengaturan IP *Firewall NAT*, *DHCP Client* pada *interface ether3*, IP *DHCP Lease*, IP *DHCP-Server Network*, IP *DHCP-Server*, IP *Pool* dan pengalamatan IP pada setiap *interface* VLAN serta *interface* VLAN. Perintah yang dieksekusi adalah **nano delete\_config\_router.yml**, seperti terlihat pada gambar berikut:

```
netdevops@gitlab:~/playbook$ nano delete_config_router.yml
```

Dengan konten sebagai berikut:

```
1 - name: Menghapus pengaturan IP Firewall NAT
2 routeros_command:
3 commands: /ip firewall nat remove [find where
4 comment={{ lokasi }}]
5 delegate_to: "{{ item }}"
6
7 - name: Menghapus pengaturan DHCP Client pada interface ether3
8 routeros_command:
9 commands: /ip dhcp-client remove [find where
10 comment={{ lokasi }}]
11 delegate_to: "{{ item }}"
12
13 - name: Menghapus IP DHCP Lease
14 routeros_command:
15 commands: /ip dhcp lease remove [find dynamic]
16 delegate_to: "{{ item }}"
17
18 - name: Menghapus pengaturan IP DHCP-Server Network
19 routeros_command:
20 commands:
```

```
19 - /ip dhcp-server network remove [find where
 comment={{ lokasi }}]
20 delegate_to: "{{ item }}"
21
22 - name: Menghapus pengaturan IP DHCP-Server
23 routeros_command:
24 commands:
25 - /ip dhcp-server remove [find where
 comment={{ lokasi }}]
26 delegate_to: "{{ item }}"
27
28 - name: Menghapus pengaturan IP Pool
29 routeros_command:
30 commands:
31 - /ip pool remove [find where comment={{ lokasi }}]
32 delegate_to: "{{ item }}"
33
34 - name: Menghapus pengaturan pengalamatan IP pada setiap
 interface VLAN
35 routeros_command:
36 commands:
37 - /ip address remove [find where comment={{ lokasi }}]
38 delegate_to: "{{ item }}"
39
40 - name: Menghapus interface VLAN
41 routeros_command:
42 commands:
43 - /interface vlan remove [find where
 comment={{ lokasi }}]
```

```
44 delegate_to: "{{ item }}"
```

Penjelasan:

- Baris 1: **name** digunakan untuk menentukan nama dari *task* yaitu “**Menghapus pengaturan IP Firewall NAT**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 2: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 3: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi. Perintahnya adalah **/ip firewall nat remove [find where comment={{ lokasi ]}}**.  
**{{ lokasi }}** digunakan untuk mengambil nilai dari extra *variable* dengan nama **lokasi**.
- Baris 4: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable item*.
- Baris 6: **name** digunakan untuk menentukan nama dari *task* yaitu “**Menghapus pengaturan DHCP Client pada interface ether3**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 7: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 8: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi. Perintahnya adalah **/ip dhcp-client remove [find where comment={{ lokasi ]}}**.  
**{{ lokasi }}** digunakan untuk mengambil nilai dari extra *variable* dengan nama **lokasi**.
- Baris 9: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable item*.
- Baris 11: **name** digunakan untuk menentukan nama dari *task* yaitu “**IP DHCP Lease**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.

- Baris 12: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 13: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi. Perintahnya adalah `/ip dhcp lease remove [find dynamic]`.
- Baris 14: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable item*.
- Baris 16: **name** digunakan untuk menentukan nama dari *task* yaitu “**Menghapus pengaturan IP DHCP-Server Network**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 17: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 18: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi.
- Baris 19: merupakan nilai dari parameter **commands** yaitu bertipe *list* yang memuat perintah `/ip dhcp-server network remove [find where comment={{ lokasi }}]`.  
`{{ lokasi }}` digunakan untuk mengambil nilai dari extra *variable* dengan nama **lokasi**.
- Baris 20: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable item*.
- Baris 22: **name** digunakan untuk menentukan nama dari *task* yaitu “**Menghapus pengaturan IP DHCP Server**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 23: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 24: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi.

- Baris 25: merupakan nilai dari parameter **commands** yaitu bertipe *list* yang memuat perintah `/ip dhcp-server remove [find where comment={{ lokasi }}]`.  
`{{ lokasi }}` digunakan untuk mengambil nilai dari extra *variable* dengan nama **lokasi**.
- Baris 26: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable item*.
- Baris 28: **name** digunakan untuk menentukan nama dari *task* yaitu “**Menghapus pengaturan IP Pool**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 29: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 30: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi.
- Baris 31: merupakan nilai dari parameter **commands** yaitu bertipe *list* yang memuat perintah `/ip pool remove [find where comment={{ lokasi }}]`.  
`{{ lokasi }}` digunakan untuk mengambil nilai dari extra *variable* dengan nama **lokasi**.
- Baris 32: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable item*.
- Baris 34: **name** digunakan untuk menentukan nama dari *task* yaitu “**Menghapus pengaturan pengalamatan IP pada setiap interface VLAN**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 35: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 36: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi.

- Baris 37: merupakan nilai dari parameter **commands** yaitu bertipe *list* yang memuat perintah `/ip address remove [find where comment={{ lokasi }}]`.  
`{{ lokasi }}` digunakan untuk mengambil nilai dari extra *variable* dengan nama **lokasi**.
- Baris 38: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable* **item**.
- Baris 40: **name** digunakan untuk menentukan nama dari *task* yaitu “**Menghapus interface VLAN**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 41: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 42: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi.
- Baris 43: merupakan nilai dari parameter **commands** yaitu bertipe *list* yang memuat perintah `/interface vlan remove [find where comment={{ lokasi }}]`.  
`{{ lokasi }}` digunakan untuk mengambil nilai dari extra *variable* dengan nama **lokasi**.
- Baris 44: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable* **item**.

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan tombol **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

- d. Membuat *file* YAML dengan nama **delete\_config\_msw.yml** yang merupakan *file task* untuk menonaktifkan *vlan filtering* pada *interface bridge* dan *interface bridge* VLAN serta *interface bridge port*. Perintah yang dieksekusi adalah `nano delete_config_msw.yml`, seperti terlihat pada gambar berikut:

```
netdevops@gitlab:~/playbook$ nano delete_config_msw.yml
```

Dengan konten sebagai berikut:

```

1 - name: Menonaktifkan vlan filtering pada interface bridge
2 routeros_command:
3 commands: /interface bridge set vlan-filtering=no
4 BR_{{ lokasi }}
5 delegate_to: "{{ item }}"
6
7 - name: Menghapus interface bridge VLAN
8 routeros_command:
9 commands:
10 - /interface bridge vlan remove [find where
11 comment={{ lokasi }}]
12 delegate_to: "{{ item }}"
13
14 - name: Menghapus interface bridge port
15 routeros_command:
16 commands:
17 - /interface bridge port remove [find where
18 comment={{ lokasi }}]
19 delegate_to: "{{ item }}"

```

Penjelasan:

- Baris 1: **name** digunakan untuk menentukan nama dari *task* yaitu “**Menonaktifkan vlan filtering pada interface bridge**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 2: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 3: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS**

sehingga dapat dieksekusi. Perintahnya adalah `/interface bridge set vlan-filtering=no BR_{{ lokasi }}`.

`{{ lokasi }}` digunakan untuk mengambil nilai dari extra *variable* dengan nama **lokasi**.

- Baris 4: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable item*.
- Baris 6: **name** digunakan untuk menentukan nama dari *task* yaitu “**Menghapus interface bridge VLAN**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 7: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 8: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi.
- Baris 9: merupakan nilai dari parameter **commands** yaitu bertipe *list* yang memuat perintah `/interface bridge vlan remove [find where comment={{ lokasi ]}}`.  
`{{ lokasi }}` digunakan untuk mengambil nilai dari extra *variable* dengan nama **lokasi**.
- Baris 10: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable item*.
- Baris 12: **name** digunakan untuk menentukan nama dari *task* yaitu “**Menghapus interface bridge port**”. Di dalam setiap *task* memuat kode yang berhubungan dengan modul yang harus dieksekusi.
- Baris 13: nama modul yang dieksekusi yaitu **routeros\_command**.
- Baris 14: **commands** merupakan parameter dari modul **routeros\_command** yang digunakan untuk menampung perintah yang akan dikirimkan ke perangkat **RouterOS** sehingga dapat dieksekusi.

- Baris 15: merupakan nilai dari parameter **commands** yaitu bertipe *list* yang memuat perintah `/interface bridge port remove [find where comment={{ lokasi }}]`.  
`{{ lokasi }}` digunakan untuk mengambil nilai dari extra *variable* dengan nama **lokasi**.
- Baris 16: **delegate\_to** digunakan untuk mengontrol lokasi eksekusi dari *task* yaitu sesuai nilai dari *variable item*.

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan tombol **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

- e. Mengujicoba eksekusi *Ansible playbook delete\_main.yml* untuk memproses penghapusan konfigurasi di lingkungan **Test network** dengan perintah `ansible-playbook -i hosts delete_main.yml -e lokasi=Test`.

```
netdevops@gitlab:~/playbook$ ansible-playbook -i hosts delete_main.yml -e lokasi=Test
```

Hasil eksekusi perintah tersebut seperti terlihat pada gambar berikut:

```
192.168.0.10:tlp - netdevops@192.168.0.10:2222 - Bitwise xterm - netdevops@gitlab: ~/playbook
netdevops@gitlab:~/playbook$ ansible-playbook -i hosts delete_main.yml -e lokasi=Test

PLAY [Playbook Workshop Profit Lombok - NetDevOps 101 (Delete)] *****

TASK [include variable] *****
ok: [localhost]

TASK [Variable untuk Router di Setiap Lokasi] *****
ok: [localhost]

TASK [Task untuk Router] *****
included: /home/netdevops/playbook/delete_config_router.yml for localhost => (item=R_Test)

TASK [Menghapus IP DHCP Lease] *****
ok: [localhost -> 192.168.0.1]

TASK [Menghapus pengaturan IP DHCP-Server Network] *****
ok: [localhost -> 192.168.0.1]

TASK [Menghapus pengaturan IP DHCP-Server] *****
ok: [localhost -> 192.168.0.1]

TASK [Menghapus pengaturan IP Pool] *****
ok: [localhost -> 192.168.0.1]

TASK [Menghapus pengaturan pengalamatan IP pada setiap interface VLAN] *****
ok: [localhost -> 192.168.0.1]

TASK [Menghapus interface VLAN] *****
ok: [localhost -> 192.168.0.1]
```

```

TASK [Menghapus pengaturan IP Firewall NAT] *****
ok: [localhost -> 192.168.0.1]

TASK [Menghapus pengaturan DHCP Client pada interface ether3] *****
ok: [localhost -> 192.168.0.1]

TASK [Variable untuk Multilayer Switch di Setiap Lokasi] *****
ok: [localhost]

TASK [Task untuk Multilayer Switch] *****
included: /home/netdevops/playbook/delete_config_msw.yml for localhost => (item=SW_Test)

TASK [Menonaktifkan vlan filtering pada interface bridge] *****
ok: [localhost -> 192.168.1.2]

TASK [Menghapus interface bridge VLAN] *****
ok: [localhost -> 192.168.1.2]

TASK [Menghapus interface bridge port] *****
ok: [localhost -> 192.168.1.2]

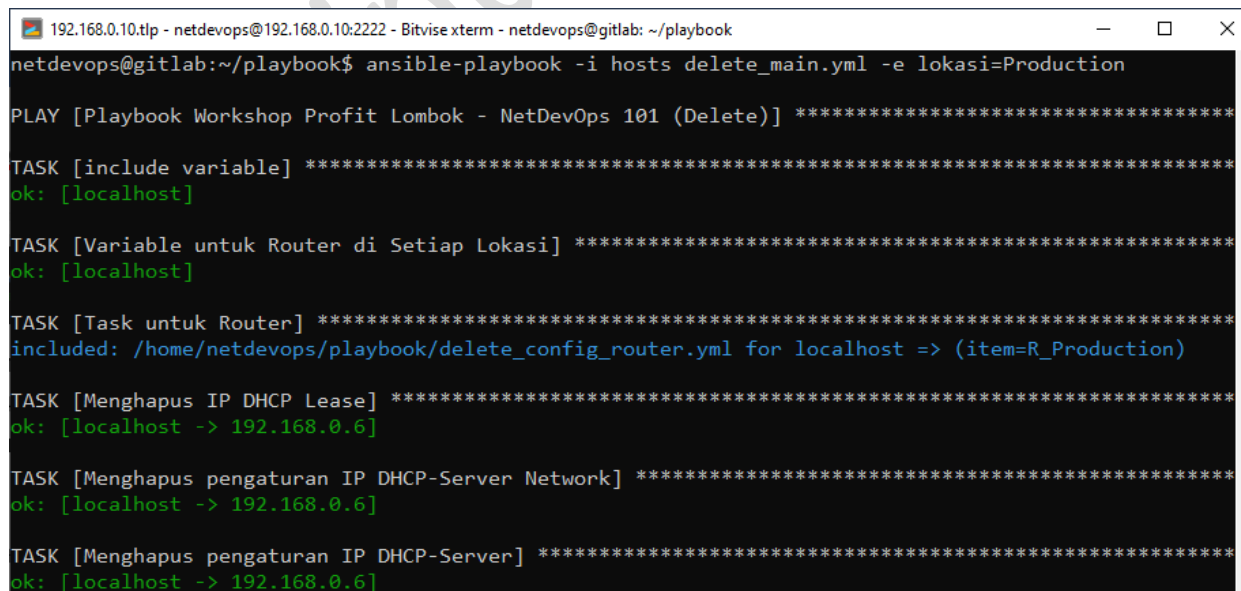
PLAY RECAP *****
localhost : ok=16 changed=0 unreachable=0 failed=0 skipped=0 rescued=
0 ignored=0

```

- f. Mengujicoba eksekusi *Ansible playbook* **delete\_main.yml** untuk memproses penghapusan konfigurasi di lingkungan **Production network** dengan perintah **ansible-playbook -i hosts delete\_main.yml -e lokasi=Production**.

```
netdevops@gitlab:~/playbook$ ansible-playbook -i hosts delete_main.yml -e lokasi=Production
```

Hasil eksekusi perintah tersebut seperti terlihat pada gambar berikut:



```

192.168.0.10:tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~/playbook
netdevops@gitlab:~/playbook$ ansible-playbook -i hosts delete_main.yml -e lokasi=Production

PLAY [Playbook Workshop Profit Lombok - NetDevOps 101 (Delete)] *****

TASK [include variable] *****
ok: [localhost]

TASK [Variable untuk Router di Setiap Lokasi] *****
ok: [localhost]

TASK [Task untuk Router] *****
included: /home/netdevops/playbook/delete_config_router.yml for localhost => (item=R_Production)

TASK [Menghapus IP DHCP Lease] *****
ok: [localhost -> 192.168.0.6]

TASK [Menghapus pengaturan IP DHCP-Server Network] *****
ok: [localhost -> 192.168.0.6]

TASK [Menghapus pengaturan IP DHCP-Server] *****
ok: [localhost -> 192.168.0.6]

```

```

TASK [Menghapus pengaturan IP Pool] *****
ok: [localhost -> 192.168.0.6]

TASK [Menghapus pengaturan pengalamatan IP pada setiap interface VLAN] *****
ok: [localhost -> 192.168.0.6]

TASK [Menghapus interface VLAN] *****
ok: [localhost -> 192.168.0.6]

TASK [Menghapus pengaturan IP Firewall NAT] *****
ok: [localhost -> 192.168.0.6]

TASK [Menghapus pengaturan DHCP Client pada interface ether3] *****
ok: [localhost -> 192.168.0.6]

TASK [Variable untuk Multilayer Switch di Setiap Lokasi] *****
ok: [localhost]

TASK [Task untuk Multilayer Switch] *****
included: /home/netdevops/playbook/delete_config_msw.yml for localhost => (item=SW_Production)

TASK [Menonaktifkan vlan filtering pada interface bridge] *****
ok: [localhost -> 192.168.11.2]

TASK [Menghapus interface bridge VLAN] *****
ok: [localhost -> 192.168.11.2]

TASK [Menghapus interface bridge port] *****
ok: [localhost -> 192.168.11.2]

PLAY RECAP *****
localhost : ok=16 changed=0 unreachable=0 failed=0 skipped=0 rescued=0
 ignored=0

```

#### J. Memverifikasi Hasil Proses Otomatisasi Penghapusan Konfigurasi NAT, VLAN Dan DHCP Server Pada Lab NETDEVOPS Di PNETLAB

Adapun langkah-langkah untuk memverifikasi hasil dari proses otomatisasi penghapusan konfigurasi NAT, VLAN dan DHCP Server pada lab **NetDevOps** di PNETLab adalah sebagai berikut:

1. Berpindah ke tab **PNETLab** pada *browser* yang digunakan.
2. Memverifikasi penonaktifan *VLAN Filtering* pada *interface bridge*, penghapusan *interface bridge VLAN* dan penghapusan *interface bridge port* pada lingkungan **test network** dengan mengeksekusi 3 (tiga) perintah meliputi **interface bridge print**, **interface bridge vlan print** dan **interface bridge port print** pada *console* dari *node SW\_Test*.

```

SW_Test
[admin@SW_Test] > interface bridge print
Flags: X - disabled, R - running
0 R name="BR_Test" mtu=auto actual-mtu=1500 l2mtu=65535 arp=enabled
 arp-timeout=auto mac-address=50:6E:D8:00:02:06 protocol-mode=rstp
 fast-forward=yes igmp-snooping=no auto-mac=yes ageing-time=5m
 priority=0x8000 max-message-age=20s forward-delay=15s
 transmit-hold-count=6 vlan-filtering=no dhcp-snooping=no
[admin@SW_Test] > interface bridge vlan print

[admin@SW_Test] > interface bridge port print
Columns: INTERFACE, BRIDGE, HW, PVID, PRIORITY, PATH-COST, INTERNAL-PATH-COST, H
ORIZON
INTERFACE BRIDGE HW PVID PRIORITY PATH-COST IN HORIZON
0 ether7 BR_Test yes 1 0x80 10 10 none
[admin@SW_Test] >

```

Terlihat ketiga operasi menonaktifkan atau penghapusan berhasil dilakukan.

- Memverifikasi penghapusan IP DHCP Lease, IP DHCP-Server Network, IP DHCP-Server, IP Pool, pengalamatan IP pada setiap interface VLAN, interface VLAN, IP Firewall NAT dan pengaturan DHCP Client pada *interface ether3* pada lingkungan **test network** dengan mengeksekusi 8 (delapan) perintah meliputi `ip dhcp-server lease print`, `ip dhcp-server network print`, `ip dhcp-server print`, `ip pool print`, `ip address print`, `interface vlan print`, `ip firewall nat print` dan `ip dhcp-client print` pada *console* dari *node R\_Test*.

```

R_Test
[admin@R_Test] > ip dhcp-server lease print

[admin@R_Test] > ip dhcp-server network print

[admin@R_Test] > ip dhcp-server print

[admin@R_Test] > ip pool print

[admin@R_Test] > ip address print
Columns: ADDRESS, NETWORK, INTERFACE
ADDRESS NETWORK INTERFACE
0 192.168.1.1/24 192.168.1.0 ether1
1 192.168.0.1/30 192.168.0.0 ether2
[admin@R_Test] > interface vlan print

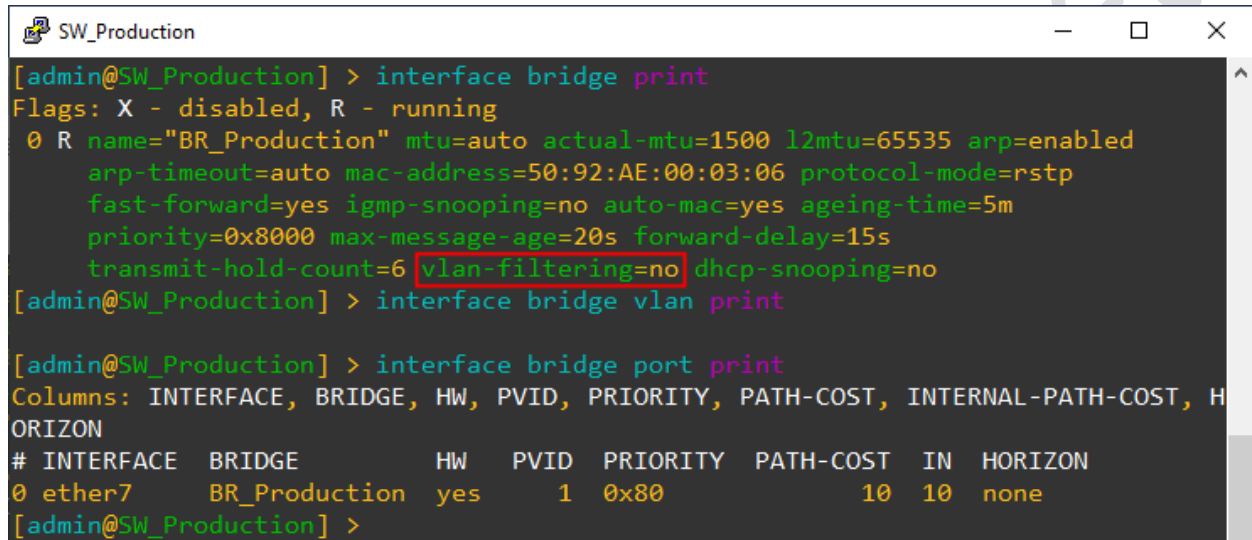
[admin@R_Test] > ip firewall nat print
Flags: X - disabled, I - invalid; D - dynamic
[admin@R_Test] > ip dhcp-client print

[admin@R_Test] >

```

Terlihat kedelapan operasi penghapusan berhasil dilakukan.

4. Memverifikasi penonaktifan *VLAN Filtering* pada *interface bridge*, penghapusan *interface bridge VLAN* dan penghapusan *interface bridge port* pada lingkungan **production network** dengan mengeksekusi 3 (tiga) perintah meliputi **interface bridge print**, **interface bridge vlan print** dan **interface bridge port print** pada *console* dari *node SW\_Production*.

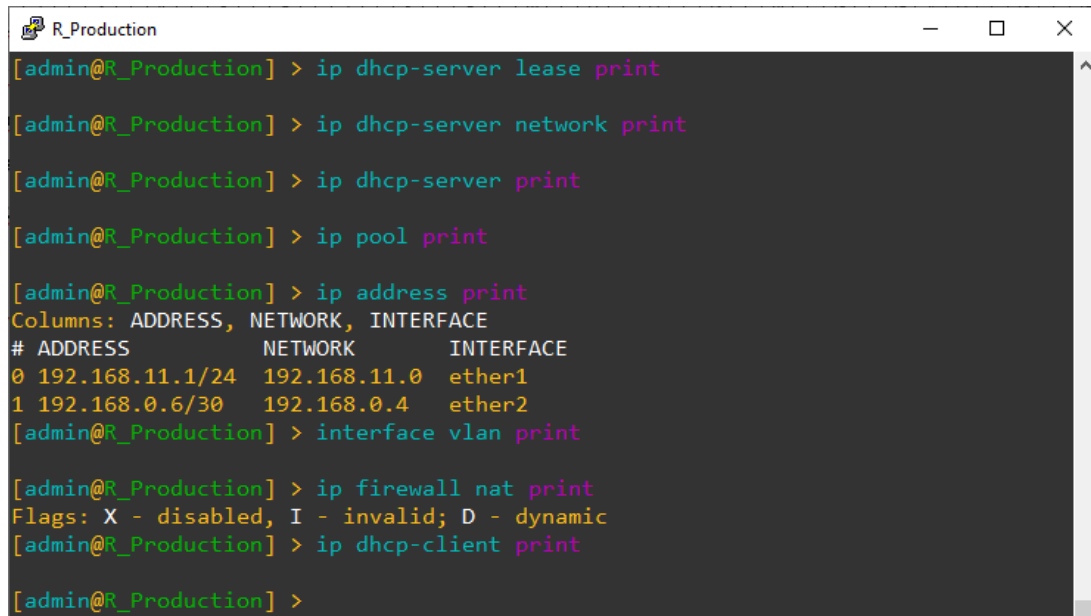


```
[admin@SW_Production] > interface bridge print
Flags: X - disabled, R - running
0 R name="BR_Production" mtu=auto actual-mtu=1500 l2mtu=65535 arp=enabled
 arp-timeout=auto mac-address=50:92:AE:00:03:06 protocol-mode=rstp
 fast-forward=yes igmp-snooping=no auto-mac=yes ageing-time=5m
 priority=0x8000 max-message-age=20s forward-delay=15s
 transmit-hold-count=6 vlan-filtering=no dhcp-snooping=no
[admin@SW_Production] > interface bridge vlan print

[admin@SW_Production] > interface bridge port print
Columns: INTERFACE, BRIDGE, HW, PVID, PRIORITY, PATH-COST, INTERNAL-PATH-COST, H
ORIZON
INTERFACE BRIDGE HW PVID PRIORITY PATH-COST IN HORIZON
0 ether7 BR_Production yes 1 0x80 10 10 none
[admin@SW_Production] >
```

Terlihat ketiga operasi penonaktifan atau penghapusan berhasil dilakukan.

5. Memverifikasi penghapusan **IP DHCP Lease**, **IP DHCP-Server Network**, **IP DHCP-Server**, **IP Pool**, pengalamatan IP pada setiap **interface VLAN**, **interface VLAN**, **IP Firewall NAT** dan pengaturan **DHCP Client** pada *interface ether3* pada lingkungan **production network** dengan mengeksekusi 8 (delapan) perintah meliputi **ip dhcp-server lease print**, **ip dhcp-server network print**, **ip dhcp-server print**, **ip pool print**, **ip address print**, **interface vlan print**, **ip firewall nat print** dan **ip dhcp-client print** pada *console* dari *node R\_Production*.



```
[admin@R_Production] > ip dhcp-server lease print

[admin@R_Production] > ip dhcp-server network print

[admin@R_Production] > ip dhcp-server print

[admin@R_Production] > ip pool print

[admin@R_Production] > ip address print
Columns: ADDRESS, NETWORK, INTERFACE
ADDRESS NETWORK INTERFACE
0 192.168.11.1/24 192.168.11.0 ether1
1 192.168.0.6/30 192.168.0.4 ether2
[admin@R_Production] > interface vlan print

[admin@R_Production] > ip firewall nat print
Flags: X - disabled, I - invalid; D - dynamic
[admin@R_Production] > ip dhcp-client print

[admin@R_Production] >
```

Terlihat kedelapan operasi penghapusan berhasil dilakukan.

## BAB 7

## MEMBANGUN GITLAB CI/CD PIPELINES

## A. Manajemen Gitlab Container

Adapun langkah-langkah untuk manajemen *GitLab Container* adalah sebagai berikut:

1. Mengakses **Bitvise xterm** dari **VM GitLab NetDevOps**.
2. Berpindah ke home direktori dari *user netdevops* yang digunakan *login* pada **VM GitLab NetDevOps** yaitu **/home/netdevops** dengan mengeksekusi perintah **cd**, seperti terlihat pada gambar berikut:

```
netdevops@gitlab:~/playbook$ cd
```

3. Menampilkan informasi direktori dimana saat ini berada dengan mengeksekusi perintah **pwd**, seperti terlihat pada gambar berikut:

```
netdevops@gitlab:~$ pwd
/home/netdevops
```

4. Menampilkan informasi isi dari direktori dimana saat ini berada dengan mengeksekusi perintah **ls**.

```
192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~
netdevops@gitlab:~$ ls
docker-compose.yml gitlab playbook repository start.sh
```

Terlihat terdapat 3 (tiga) direktori yaitu **gitlab** (digunakan untuk menyimpan data dan konfigurasi **GitLab Container**), **playbook** (digunakan untuk menampung file-file **Ansible playbook**) dan **repository** (digunakan sebagai tempat pembuatan *repository* lokal).

Selain itu juga terlihat terdapat 2 (dua) *file* yaitu **docker-compose.yml** dan **start.sh**. *File docker-compose.yml* merupakan *file* konfigurasi dari **Docker Compose** yang digunakan untuk mendefinisikan dan menjalankan *multi-container Docker applications* yaitu **GitLab** dan **GitLab Runner**. Sedangkan *file start.sh* merupakan *shell script* yang didalamnya memuat instruksi menampilkan informasi *environment variable* dengan nama **GITLAB\_HOME** yang

menyimpan **PATH** dari direktori **gitlab** dan perintah **docker compose** untuk menjalankan **GitLab Container**, seperti yang ditunjukkan pada gambar berikut:

```
192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~
netdevops@gitlab:~$ cat start.sh
#!/bin/bash
echo $GITLAB_HOME
docker compose up -d
netdevops@gitlab:~$
```

5. Mengeksekusi **shell script** **start.sh** dengan perintah **./start.sh** yang didalamnya memuat perintah untuk menjalankan **GitLab Container** berdasarkan ketentuan yang terdapat pada file **docker-compose.yml**.

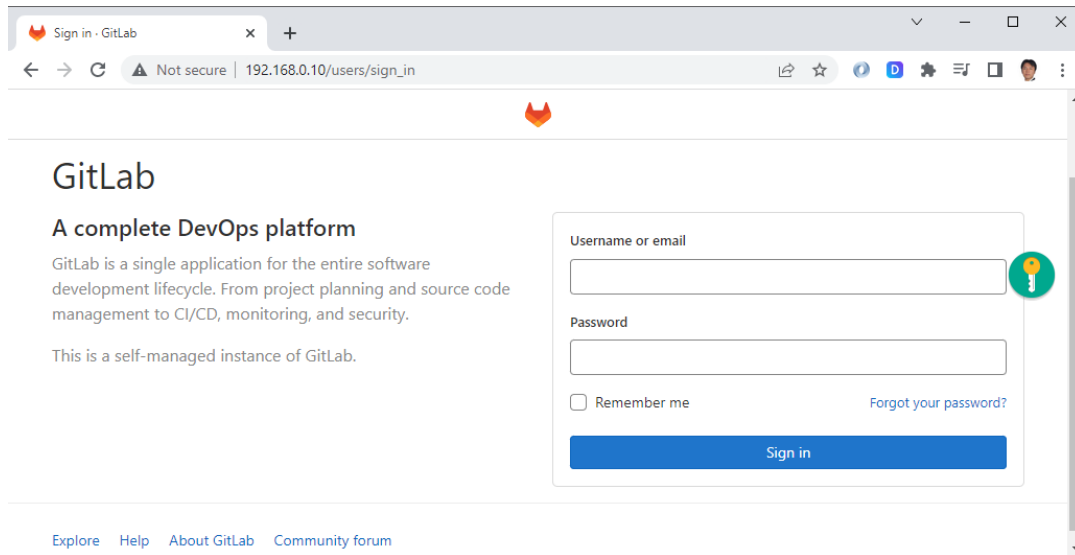
```
192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~
netdevops@gitlab:~$./start.sh
/home/netdevops/gitlab
[+] Running 3/3
 # Network gitlab-network Created 0.1s
 # Container gitlab-ce Started 1.3s
 # Container gitlab-runner Started 2.5s
netdevops@gitlab:~$
```

6. Menampilkan informasi daftar *container* yang berjalan dengan mengeksekusi perintah **docker ps**.

```
netdevops@gitlab:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES
efc4f42db49b gitlab/gitlab-runner:alpine "/usr/bin/dumb-init ..." 13 seconds ago Up 8 seconds
gitlab-runner
f68d8af576da gitlab/gitlab-ce:latest "/assets/wrapper" 13 seconds ago Up 10 seconds
(health: starting) 0.0.0.0:22->22/tcp, :::22->22/tcp, 0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443->443/tcp, :::443->443/tcp gitlab-ce
netdevops@gitlab:~$
```

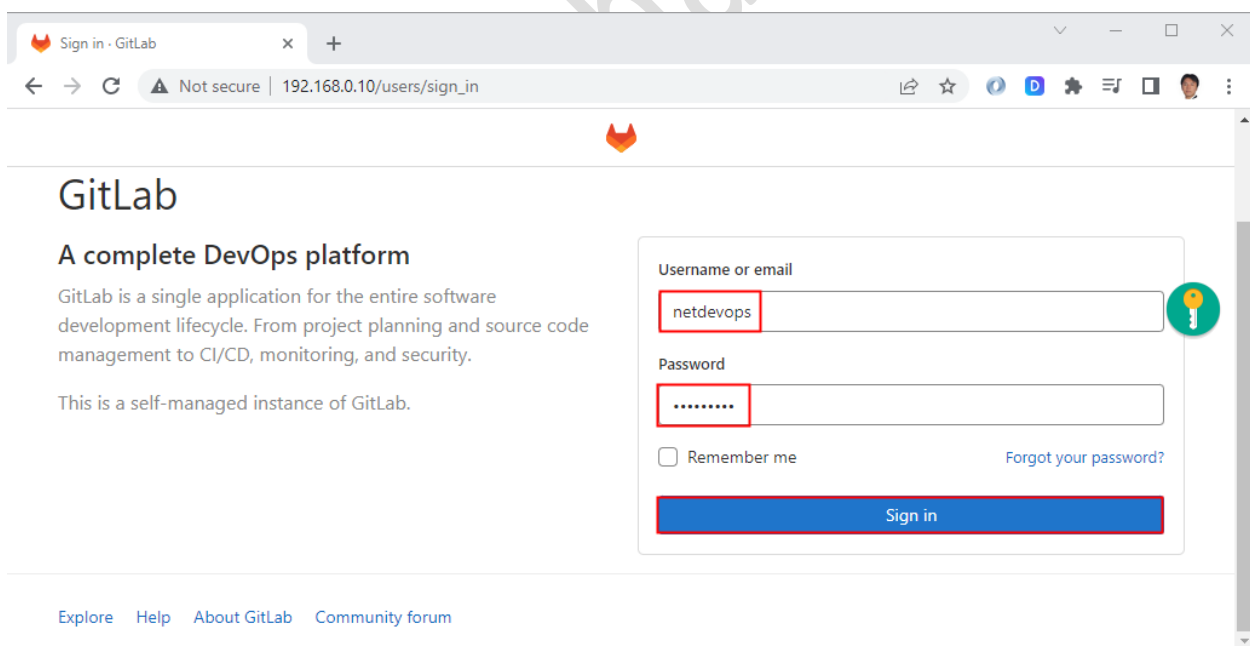
Terlihat container **gitlab-runner** dan **gitlab-ce** telah berjalan.

7. Mengakses **GitLab Web Graphical User Interface (GUI)** atau portal melalui *browser* sebagai contoh menggunakan *Google Chrome* pada alamat <http://192.168.0.10> sehingga akan tampil halaman *homepage GitLab*, seperti yang ditunjukkan pada gambar berikut:

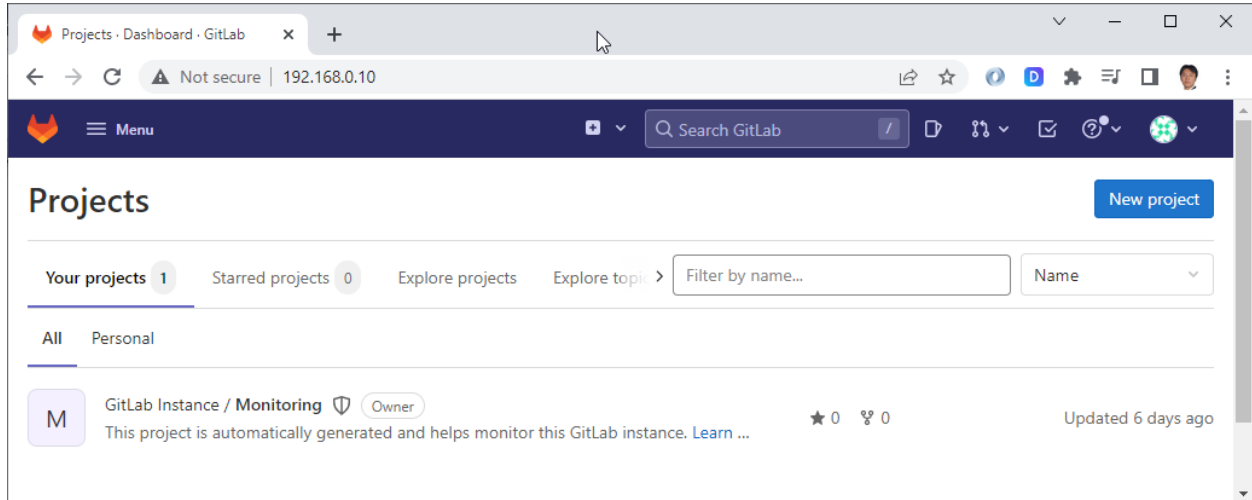


Apabila belum terlihat maka mohon menunggu kira-kira 5 menit dan mencoba mengakses kembali dengan melakukan *refresh* pada halaman tersebut.

**Login** menggunakan akun otentikasi berupa **username** dan **password** “**netdevops**” serta tekan tombol **Sign in**, seperti yang ditunjukkan pada gambar berikut:



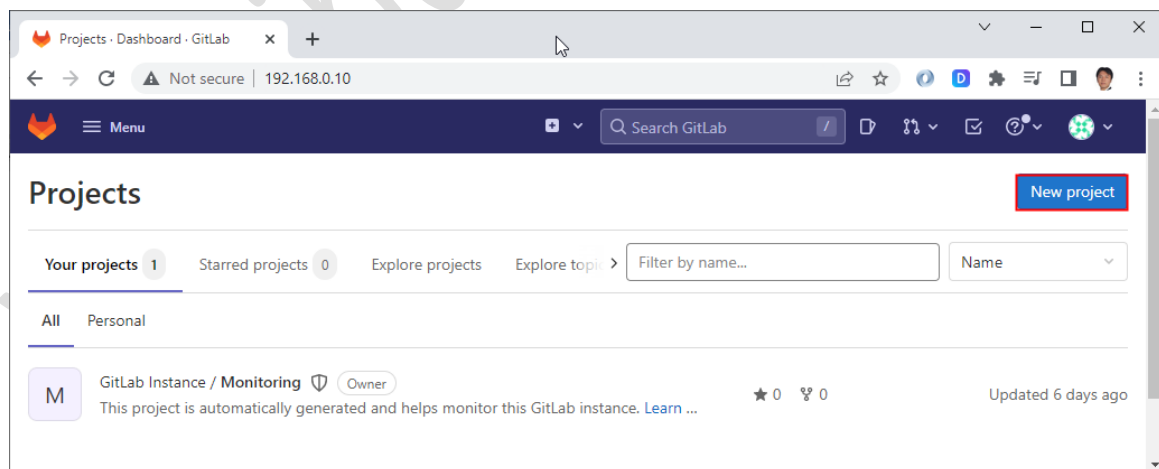
Apabila proses otentikasi *login* berhasil dilakukan maka akan tampil halaman **Dashboard**, seperti yang ditunjukkan pada gambar berikut:



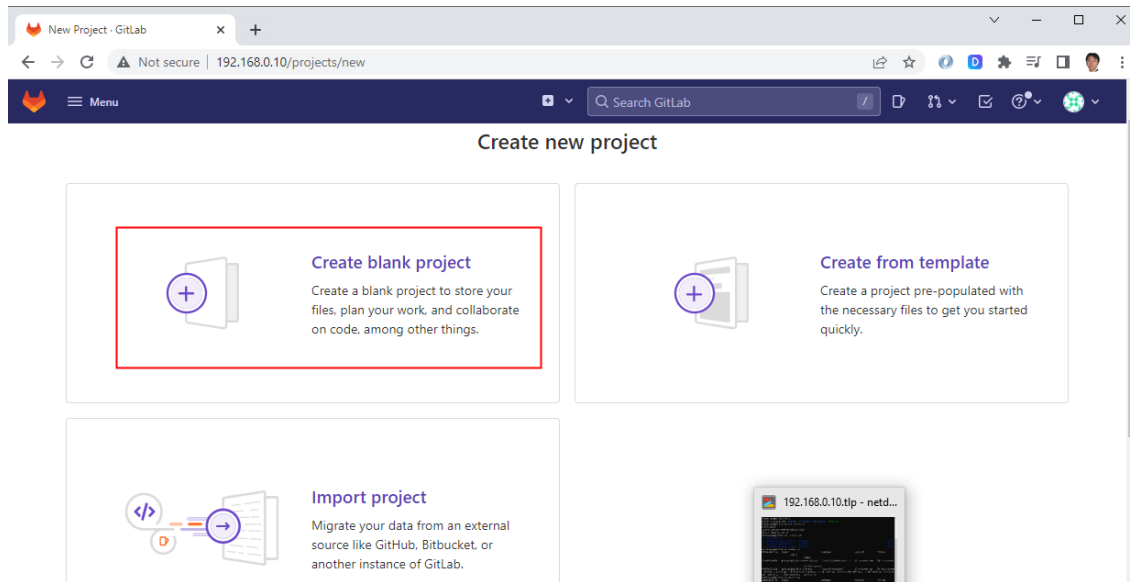
## B. Pembuatan Gitlab Repository Untuk NETDEVOPS

Adapun langkah-langkah untuk membuat **GitLab Repository** untuk lab **NetDevOps** adalah sebagai berikut:

1. Setiap repository di *GitLab* menjadi milik sebuah **Project**. *Project* mencakup berbagai fitur meliputi **repository**, *issue tracker*, *merge requests*, **Continuous Integration/Continuous Delivery (CI/CD)** dan lain-lain. Klik tombol **New Project** untuk membuat proyek baru, seperti yang ditunjukkan pada gambar berikut:

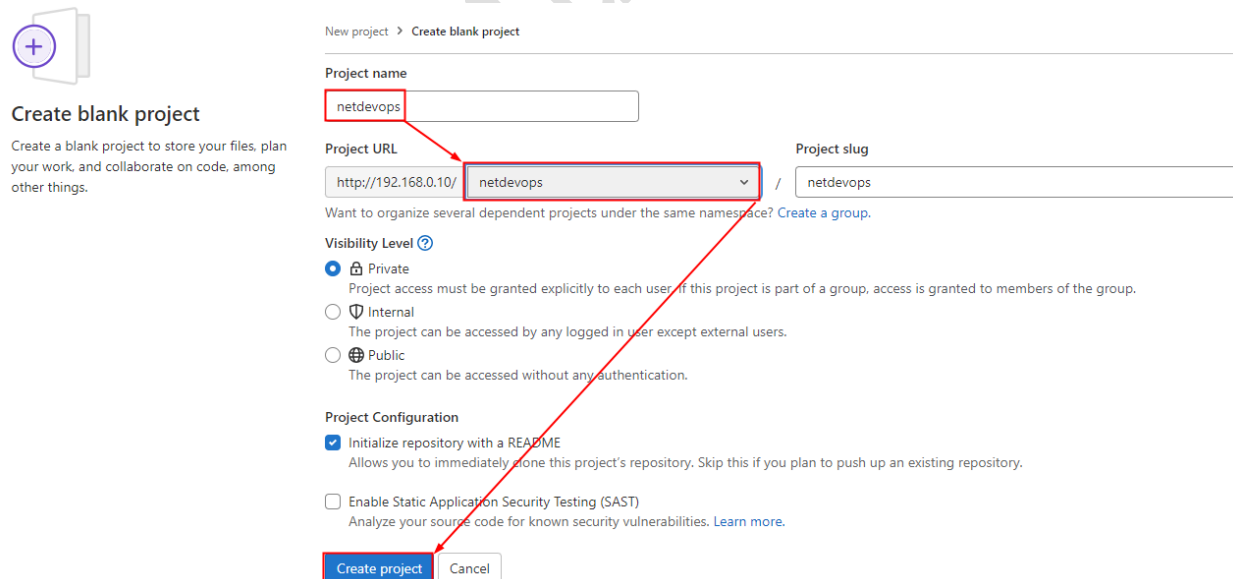


2. Tampil halaman **Create new project**, seperti yang ditunjukkan pada gambar berikut:

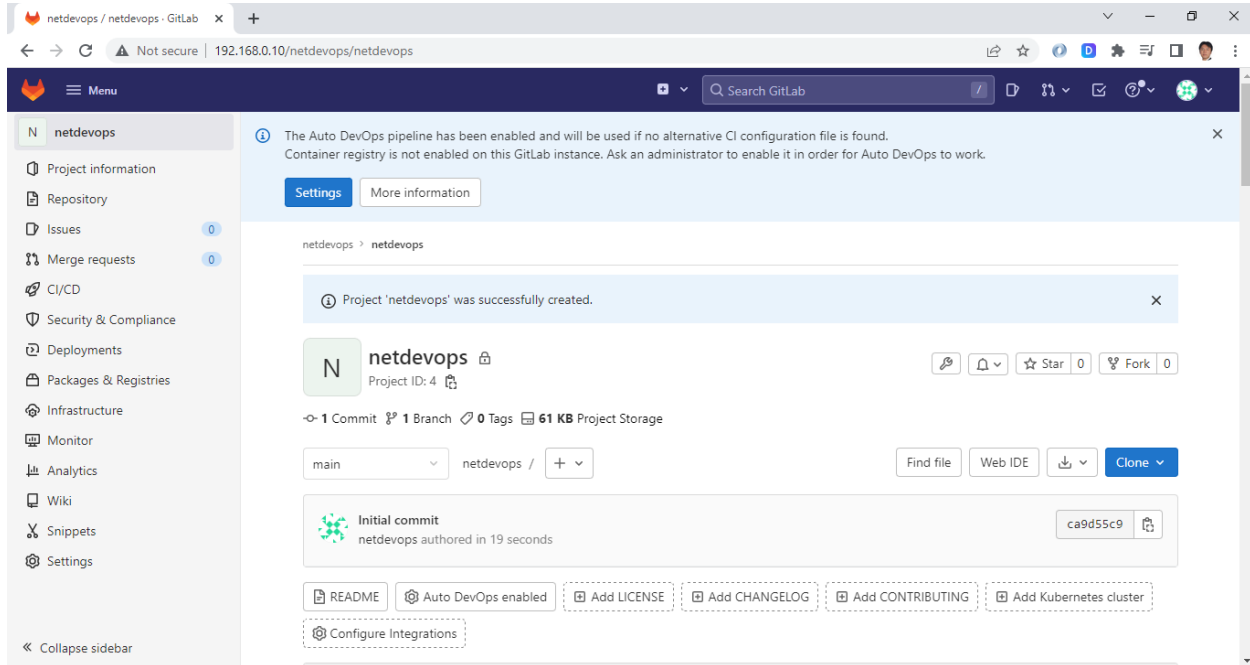


Klik pada pilihan **Create blank project**.

Tampil halaman **Create blank project**. Lengkapi pengaturan parameter **Project Name** berupa nama proyek, sebagai contoh **"netdevops"** dan pada menu dropdown dari parameter **Project URL** memilih **"netdevops"** serta menekan tombol **Create project**, seperti yang ditunjukkan pada gambar berikut:



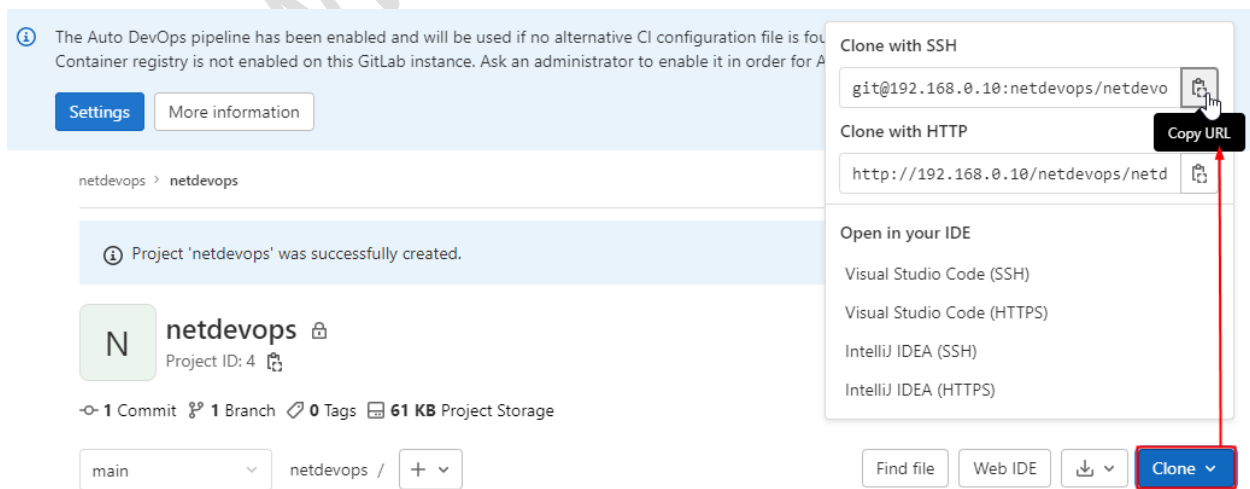
Hasil dari pembuatan **Project**, seperti ditunjukkan pada gambar berikut:



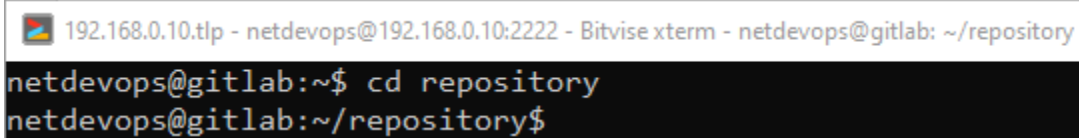
## C. Clone Gitlab Remote Repository

Adapun langkah-langkah melakukan **clone GitLab Remote Repository** ke repository lokal adalah sebagai berikut:

1. Pada halaman **Project netdevops**, klik pada tombol **Clone**. Pada menu yang tampil, klik **icon Copy URL** dari parameter **Clone with SSH** untuk menyalin *clone* URL melalui SSH dari *repository netdevops*, seperti yang ditunjukkan pada gambar berikut:

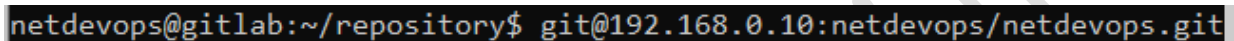


2. Akses **Bitvise xterm** dari **VM GitLab NetDevOps** dan lakukan perpindahan direktori ke **repository** dengan mengeksekusi perintah **cd repository**, seperti ditunjukkan pada gambar berikut:



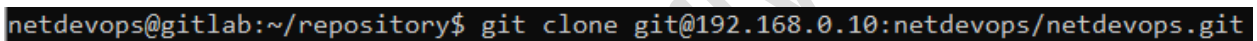
```
192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~/repository
netdevops@gitlab:~$ cd repository
netdevops@gitlab:~/repository$
```

3. **Paste Clone URL SSH** ke **Bitvise xterm** dari **VM GitLab NetDevOps** dengan menekan klik kanan pada *mouse*, seperti yang ditunjukkan pada gambar berikut:



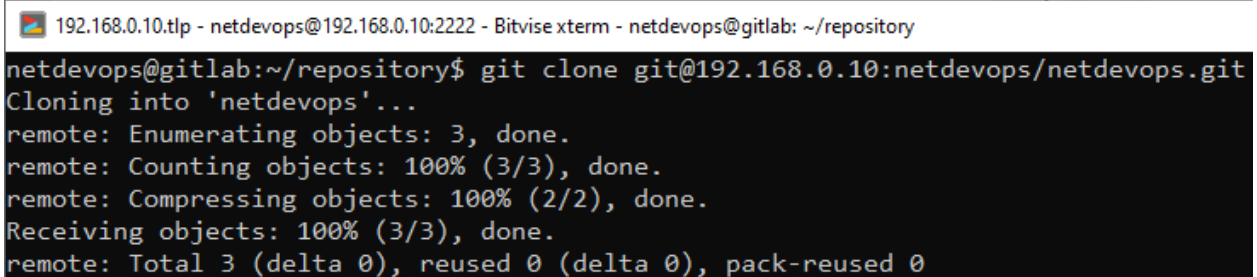
```
netdevops@gitlab:~/repository$ git@192.168.0.10:netdevops/netdevops.git
```

Tambahkan perintah "**git clone** " di awal dari *Clone URL SSH* sehingga perintahnya menjadi "**git clone git@192.168.0.10:netdevops/netdevops.git**", terlihat seperti gambar berikut:



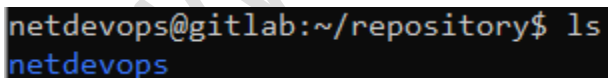
```
netdevops@gitlab:~/repository$ git clone git@192.168.0.10:netdevops/netdevops.git
```

Tekan tombol **Enter** untuk memproses operasi *clone repository* sehingga hasilnya akan terlihat seperti yang ditunjukkan pada gambar berikut:



```
192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~/repository
netdevops@gitlab:~/repository$ git clone git@192.168.0.10:netdevops/netdevops.git
Cloning into 'netdevops'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
Receiving objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
```

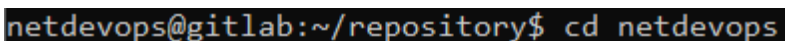
4. Memverifikasi hasil dari *clone remote repository* dengan mengeksekusi perintah **ls**.



```
netdevops@gitlab:~/repository$ ls
netdevops
```

Terlihat direktori dengan nama **netdevops** sebagai hasil dari operasi *clone repository*.

5. Berpindah ke direktori **netdevops** dengan mengeksekusi perintah **cd netdevops**, seperti yang ditunjukkan pada gambar berikut:



```
netdevops@gitlab:~/repository$ cd netdevops
```

6. Menampilkan isi dari direktori dimana saat ini berada dengan mengeksekusi perintah **ls**.

```
netdevops@gitlab:~/repository/netdevops$ ls
README.md
```

Terlihat satu *file* dengan nama **README.md**.

#### D. Push Local Repository Ke Gitlab Remote Repository

Adapun langkah-langkah untuk melakukan *push local repository* ke *GitLab Remote repository* adalah sebagai berikut:

1. Menyalinkan *file-file playbook* yang terdapat di direktori *playbook* ke dalam direktori **netdevops** saat ini dengan mengeksekusi perintah "**cp ~/playbook/\* .**", seperti terlihat pada gambar berikut:

```
netdevops@gitlab:~/repository/netdevops$ cp ~/playbook/* .
```

2. Memverifikasi hasil penyalinan *file playbook* dengan mengeksekusi perintah **ls**, seperti terlihat pada gambar berikut:

```
netdevops@gitlab:~/repository/netdevops$ ls
ansible.cfg delete_main.yml msw.yml var_Production.yml
delete_config_msw.yml hosts README.md var_Test.yml
delete_config_router.yml main.yml router.yml
```

Terlihat *file-file playbook* telah berhasil disalinkan.

3. Menampilkan informasi perubahan pada *file-file* di direktori kerja dengan mengeksekusi perintah **git status**. Terlihat **untracked files** yaitu *file-file playbook* yang sebelumnya disalinkan dari direktori *playbook*, seperti terlihat pada gambar berikut:

```

192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~/repository/netdevops
netdevops@gitlab:~/repository/netdevops$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

Untracked files:
 (use "git add <file>..." to include in what will be committed)
 ansible.cfg
 delete_config_msw.yml
 delete_config_router.yml
 delete_main.yml
 hosts
 main.yml
 msw.yml
 router.yml
 var_Production.yml
 var_Test.yml

no changes added to commit (use "git add" and/or "git commit -a")
netdevops@gitlab:~/repository/netdevops$

```

4. Menambahkan seluruh *file* di direktori kerja saat ini ke **staging area** dengan mengeksekusi perintah `"git add ."` dan memverifikasi hasil penambahannya dengan mengeksekusi perintah `git status`, seperti terlihat pada gambar berikut:

```

192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~/repository/netdevops
netdevops@gitlab:~/repository/netdevops$ git add .
netdevops@gitlab:~/repository/netdevops$ git status
On branch main
Your branch is up to date with 'origin/main'.

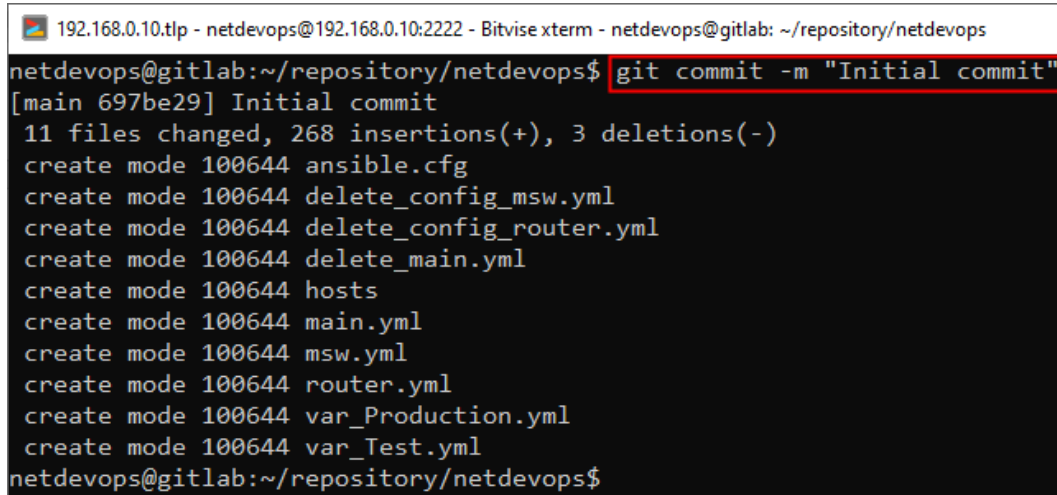
Changes to be committed:
 (use "git restore --staged <file>..." to unstage)
 modified: README.md
 new file: ansible.cfg
 new file: delete_config_msw.yml
 new file: delete_config_router.yml
 new file: delete_main.yml
 new file: hosts
 new file: main.yml
 new file: msw.yml
 new file: router.yml
 new file: var_Production.yml
 new file: var_Test.yml

netdevops@gitlab:~/repository/netdevops$

```

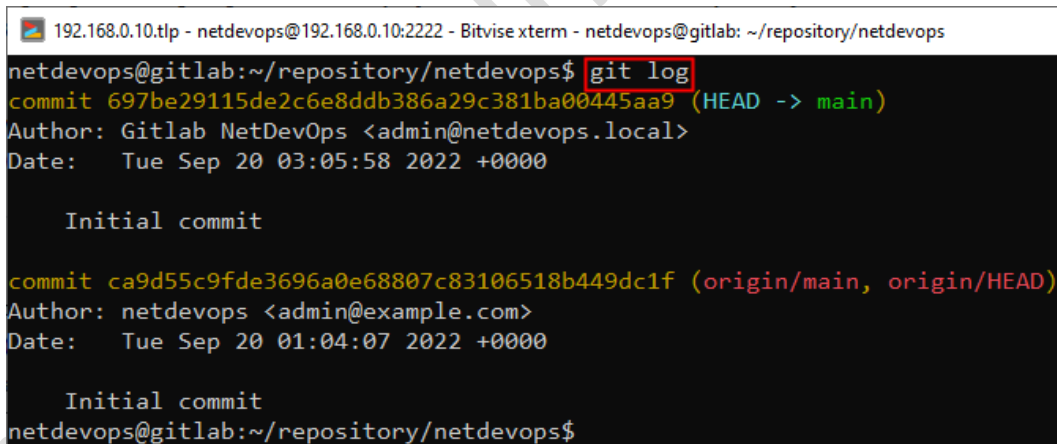
Terlihat baik file yang dimodifikasi maupun file yang baru ditambahkan telah berada pada **staging area**.

5. Melakukan **commit** konten pada **staging area** dengan pesan "**Initial commit**" dengan mengeksekusi perintah `git commit -m "Initial commit"`, seperti terlihat pada gambar berikut:



```
192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~/repository/netdevops
netdevops@gitlab:~/repository/netdevops$ git commit -m "Initial commit"
[main 697be29] Initial commit
11 files changed, 268 insertions(+), 3 deletions(-)
create mode 100644 ansible.cfg
create mode 100644 delete_config_msw.yml
create mode 100644 delete_config_router.yml
create mode 100644 delete_main.yml
create mode 100644 hosts
create mode 100644 main.yml
create mode 100644 msw.yml
create mode 100644 router.yml
create mode 100644 var_Production.yml
create mode 100644 var_Test.yml
netdevops@gitlab:~/repository/netdevops$
```

6. Menampilkan informasi seluruh *commit* pada *history repository* dengan mengeksekusi perintah `git log`, seperti terlihat pada gambar berikut:



```
192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~/repository/netdevops
netdevops@gitlab:~/repository/netdevops$ git log
commit 697be29115de2c6e8ddb386a29c381ba00445aa9 (HEAD -> main)
Author: Gitlab NetDevOps <admin@netdevops.local>
Date: Tue Sep 20 03:05:58 2022 +0000

 Initial commit

commit ca9d55c9fde3696a0e68807c83106518b449dc1f (origin/main, origin/HEAD)
Author: netdevops <admin@example.com>
Date: Tue Sep 20 01:04:07 2022 +0000

 Initial commit
netdevops@gitlab:~/repository/netdevops$
```

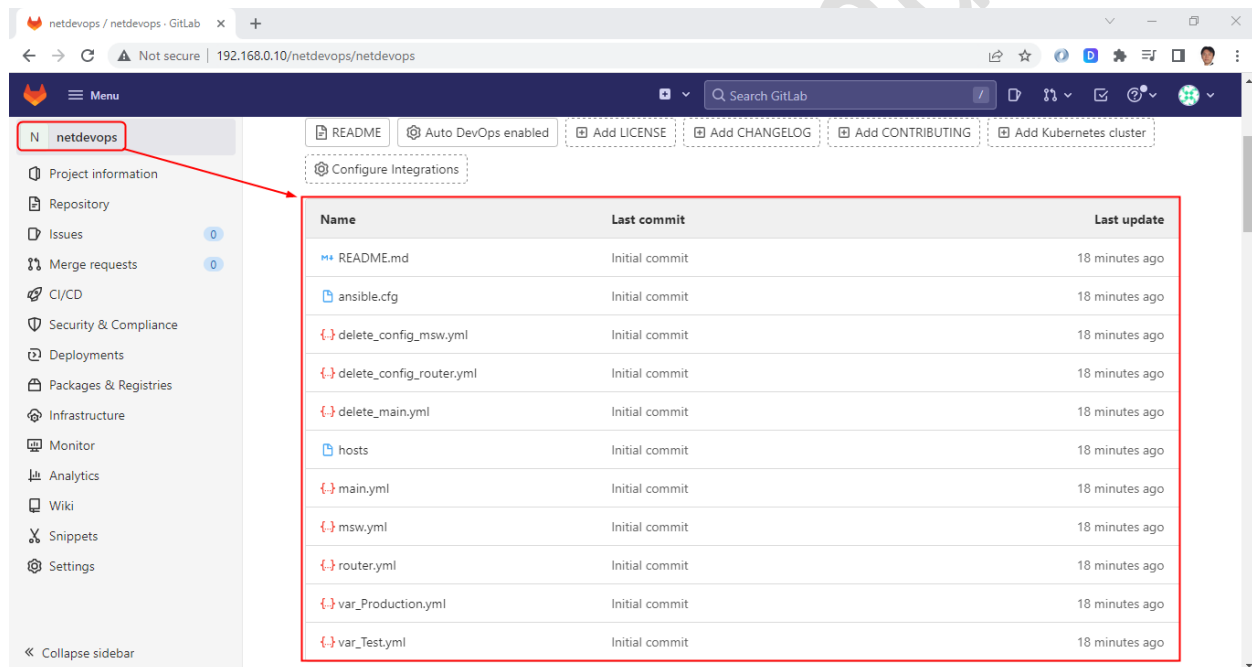
7. Melakukan **push** konten pada *repository* lokal ke *remote GitLab repository* dengan mengeksekusi perintah `git push origin main`, seperti terlihat pada gambar berikut:

```

192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~/repository/netdevops
netdevops@gitlab:~/repository/netdevops$ git push origin main
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 2 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (13/13), 2.93 KiB | 187.00 KiB/s, done.
Total 13 (delta 3), reused 0 (delta 0), pack-reused 0
To 192.168.0.10:netdevops/netdevops.git
 ca9d55c..697be29 main -> main
netdevops@gitlab:~/repository/netdevops$

```

8. Memverifikasi hasil **push** dengan melakukan **refresh** pada *browser* yang digunakan mengakses ke **GitLab Web GUI** atau portal sebelumnya, seperti terlihat pada gambar berikut:

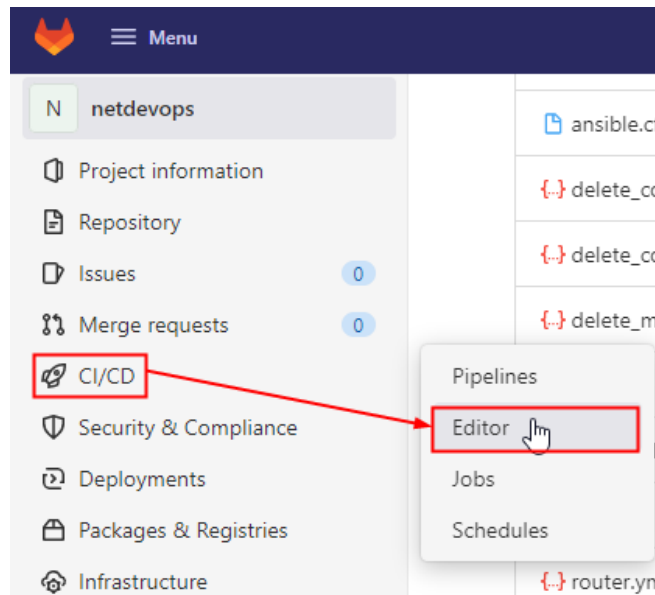


Terlihat *file-file* dari lokal *repository* telah berhasil di **push** ke *GitLab remote repository*.

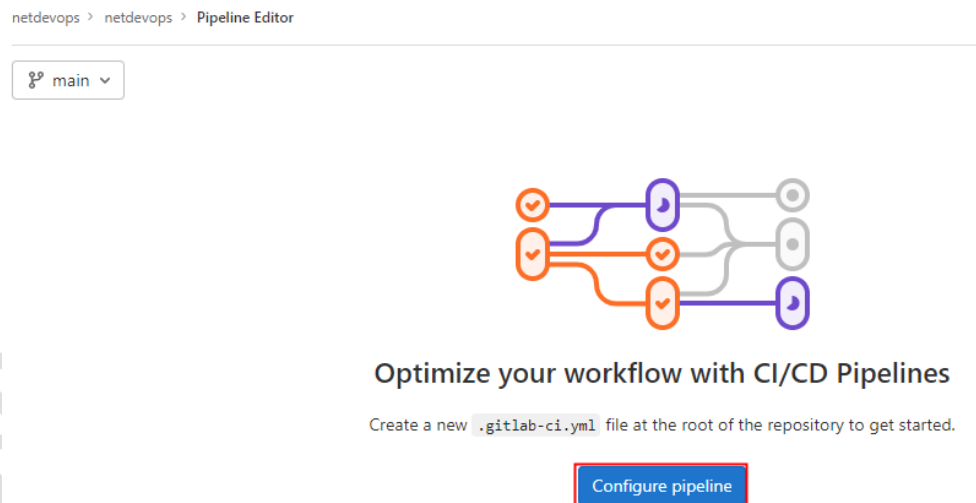
## E. Membuat Gitlab CI/CD Pipeline Untuk Mengotomatisasi Penerapan Konfigurasi NAT, VLAN Dan DHCP Server Pada Lab NETDEVOPS Di PNETLAB

Adapun langkah-langkah untuk membuat **GitLab CI/CD Pipeline** untuk mengotomatisasi penerapan konfigurasi **NAT, VLAN** dan **DHCP Server** pada lab **NetDevOps** adalah sebagai berikut:

1. Membuat **CI/CD pipeline** pada *project NetDevOps* dengan memilih **CI/CD** pada menu sebelah kiri dan kemudian memilih **Editor**, seperti terlihat pada gambar berikut:

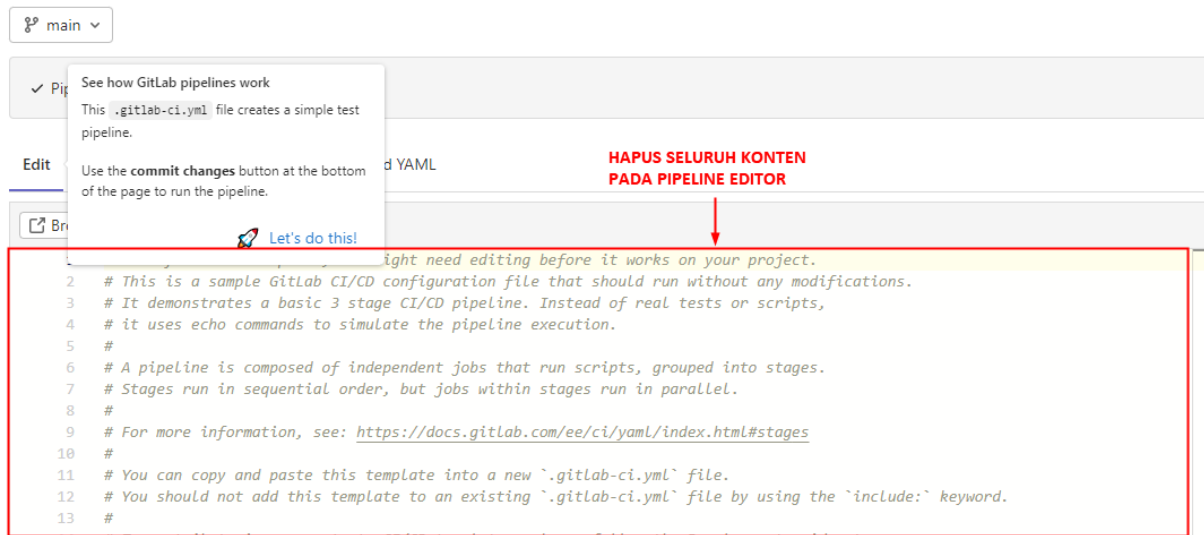


2. Tampil halaman **Pipeline Editor**. Klik tombol **Configure pipeline** untuk mengkonfigurasi pipeline yaitu dengan membuat *file .gitlab-ci.yml* pada *root* dari *repository*, seperti terlihat pada gambar berikut:



Tampil halaman *Pipeline Editor* dengan bagian **Edit** yang telah memuat konten **file template** berupa contoh konfigurasi **CI/CD pipeline**. Klik sembarang baris di dalam *textarea* dan tekan tombol **CTRL+A** serta tekan tombol **Delete** untuk menghapus seluruh konten yang terdapat di bagian **Edit** tersebut, seperti terlihat pada gambar berikut:

netdevops &gt; netdevops &gt; Pipeline Editor



Setelah berhasil dikosongkan maka masukkan konfigurasi *pipeline* berikut ke dalam *textarea* dari bagian **Edit** sehingga menjadi konten dari *file .gitlab-ci.yml*:

```

1 ---
2 before_script:
3 - export ANSIBLE_HOST_KEY_CHECKING=False
4
5 stages:
6 - test
7 - deploy
8
9 variables:
10 GIT_STRATEGY: clone
11
12 test_deployment:
13 stage: test
14 image: "chusiang/ansible:latest"
15 tags:
16 - test

```

```
17 script:
18 - ansible-playbook -i hosts main.yml -e lokasi=Test
19
20 deploy:
21 stage: deploy
22 image: "chusiang/ansible:latest"
23 only:
24 refs:
25 - main
26 tags:
27 - deploy
28 script:
29 - ansible-playbook -i hosts main.yml -e
 lokasi=Production
```

Penjelasan:

- Baris 1 yaitu `---` (3 *hyphen*) merupakan awal dari dokumen YAML.
- Baris 2 sampai dengan 3 memuat deklarasi **keyword before\_script** yang digunakan untuk mengganti sekumpulan perintah yang dieksekusi sebelum pekerjaan yaitu dalam hal ini **export variable ANSIBLE\_HOST\_KEY\_CHECK** bernilai **FALSE**.
- Baris 5 sampai dengan 7 merupakan deklarasi **stage** dari *job* yaitu **test** dan **deploy**.
- Baris 9 sampai dengan 10 merupakan deklarasi **variable** CI/CD untuk seluruh *job* di *pipeline*. **Variable GIT\_STRATEGY** dengan nilai **clone** digunakan untuk melakukan *clone repository* dari awal untuk setiap *job* sehingga memastikan salinannya selalu murni.
- Baris 12 sampai dengan 18 merupakan deklarasi **job** dengan nama **test\_deployment** yang dieksekusi pada **stage deploy**. **Docker image** yang digunakan untuk menjalankan *job* tersebut adalah **Ansible** sesuai dengan nilai dari **keyword image**. **Keyword tags** digunakan untuk memilih **GitLab runner** yaitu **test**. Sedangkan **keyword script** digunakan oleh **GitLab runner** untuk mengeksekusi **Ansible playbook** dengan nama *file main.yml* pada *node-node* di PNETLab yang terdefinisi pada *file inventory* dengan

nama **hosts**. Eksekusi **task** pada *Ansible playbook* tersebut hanya diterapkan pada lingkungan **Test network**.

- Baris 20 sampai dengan 29 merupakan deklarasi **job** dengan nama **deploy** yang dieksekusi pada **stage deploy**. **Docker image** yang digunakan untuk menjalankan job tersebut adalah **Ansible** sesuai dengan nilai dari **keyword image**. **Keyword only:refs** digunakan untuk mengontrol kapan menambahkan job ke pipeline berdasarkan pada nama **branch** yaitu **main**. **Keyword tags** digunakan untuk memilih **GitLab runner** yaitu **deploy**. Sedangkan **keyword script** digunakan oleh **GitLab runner** untuk mengeksekusi **Ansible playbook** dengan nama *file main.yml* pada *node-node* di PNETLab yang terdefinisi pada *file inventory* dengan nama **hosts**. Eksekusi **task** pada *Ansible playbook* tersebut hanya diterapkan pada lingkungan **Production network**.

Apabila telah selesai maka tekan tombol **Commit changes**, seperti terlihat pada gambar berikut:

```

12 test_deployment:
13 stage: test
14 image: "chusiang/ansible:latest"
15 tags:
16 - test
17 script:
18 - ansible-playbook -i hosts main.yml -e lokasi=Test
19
20 deploy:
21 stage: deploy
22 image: "chusiang/ansible:latest"
23 only:
24 refs:
25 - main
26 tags:
27 - deploy
28 script:
29 - ansible-playbook -i hosts main.yml -e lokasi=Production
30

```

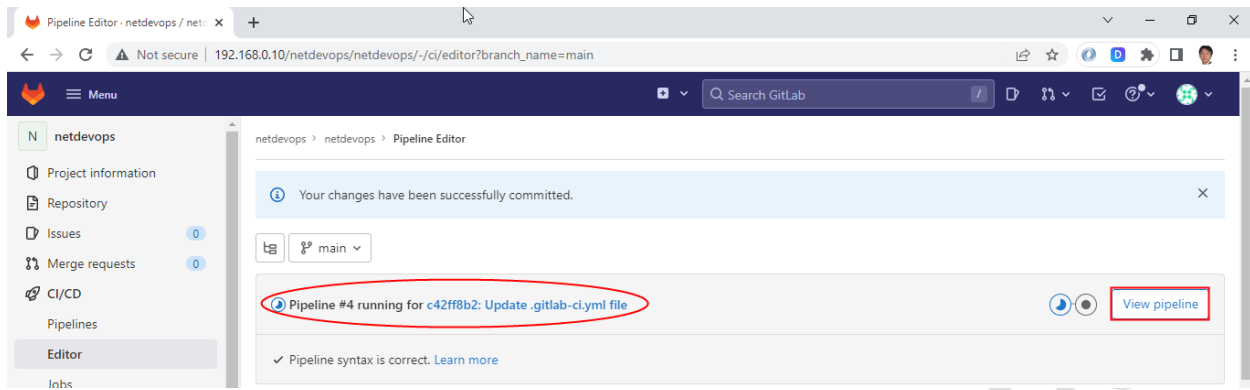
Commit message

Branch

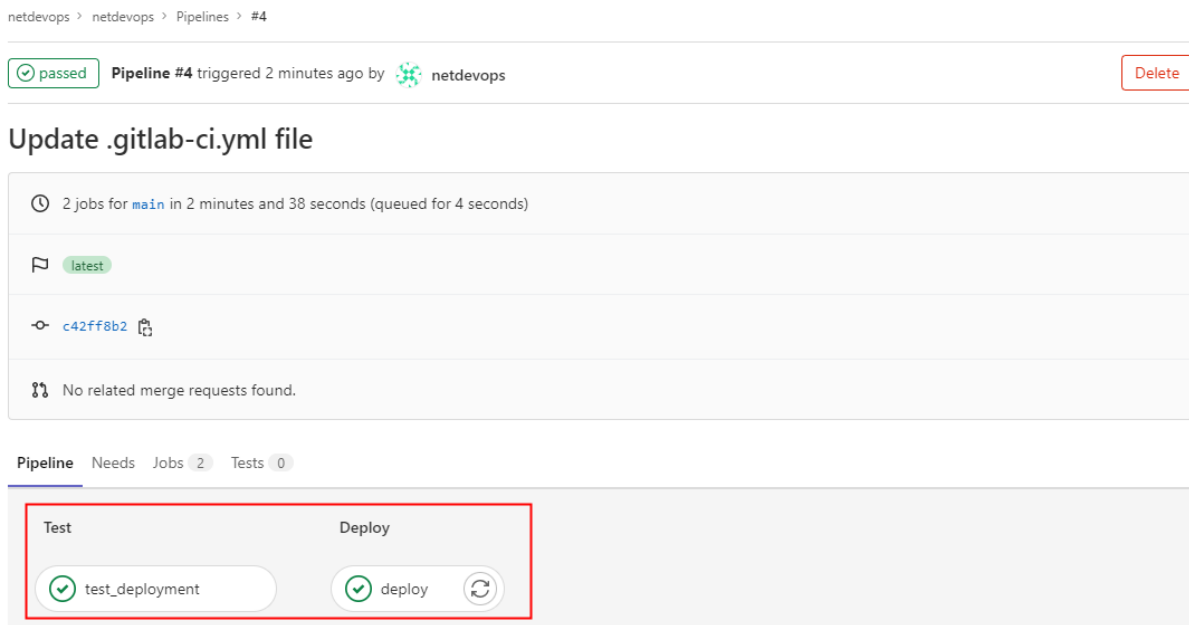
**Commit changes**

Tampil pesan bahwa *Pipeline* telah berjalan. Klik tombol **View pipeline** untuk menampilkan informasi detail terkait **stage** dan **jobs** yang berjalan pada *pipeline*, seperti terlihat pada gambar berikut:

seperti terlihat pada gambar berikut:

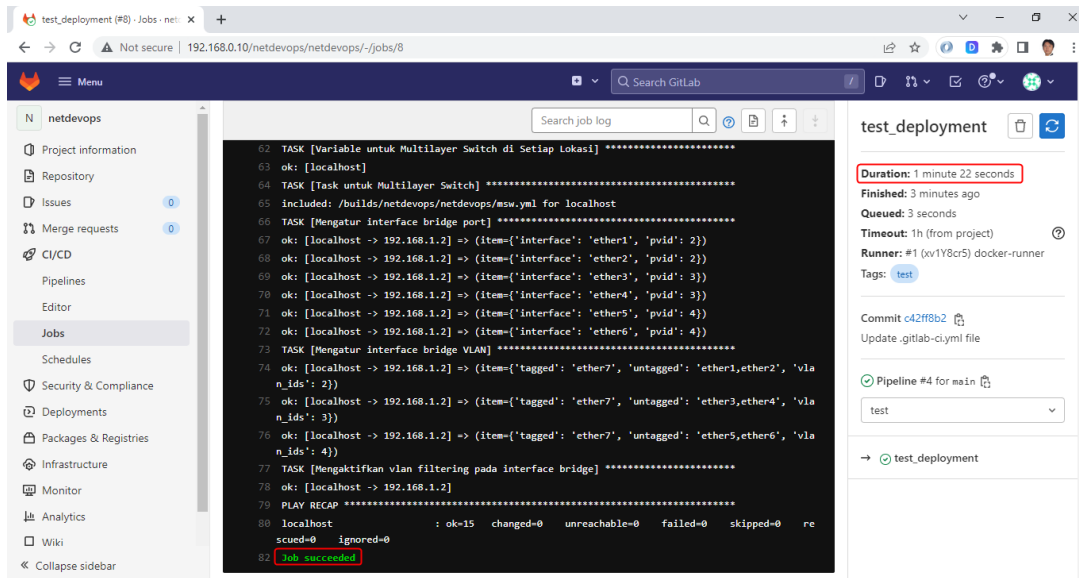


Tampil halaman yang memperlihatkan detail dari **stage** dan **jobs** yang berjalan pada *pipeline*, seperti yang ditunjukkan pada gambar berikut:

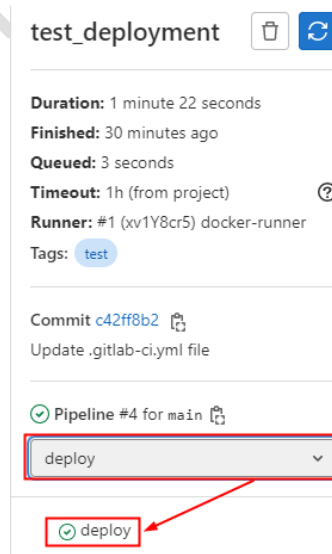


Terlihat terdapat 2 (dua) **stage** yaitu **Test** dan **Deploy**. Pada *stage* **Test** terdapat satu **job** yang berjalan yaitu **test\_deployment** dan telah sukses dieksekusi dimana ditandai dengan centang berwarna hijau. Sedangkan pada *stage* **Deploy** juga terdapat satu **job** yang berjalan yaitu **deploy** dan juga telah sukses dieksekusi dimana ditandai dengan centang berwarna hijau.

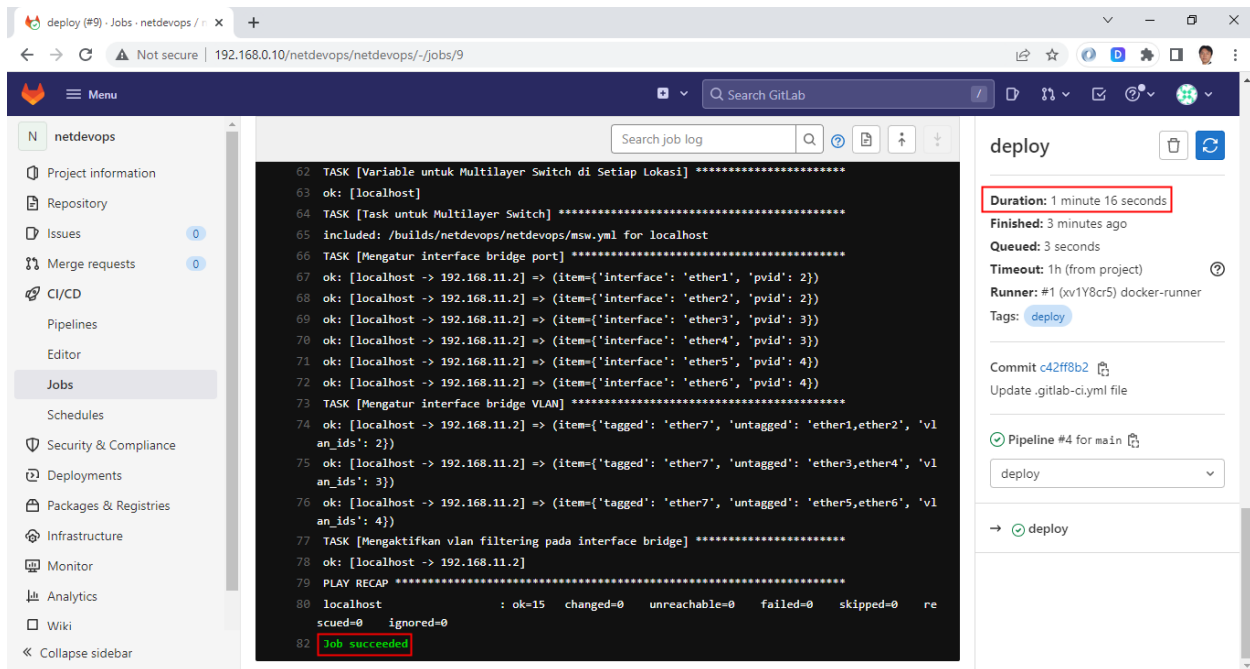
Selanjutnya klik pada *job* **test\_deployment** maka akan menampilkan informasi detail terkait eksekusi *job* tersebut, seperti terlihat pada gambar berikut:



Pada bagian tengah dari halaman ini yaitu dengan latar belakang berwarna hitam menunjukkan hasil eksekusi setiap *task* dari *Ansible playbook* di lingkungan **test network** dari lab **NetDevOps** pada PNETLab. Selain itu juga pada bagian sebelah kiri bawah terdapat pesan **Job succeeded** yang menginformasikan bahwa *job* telah berhasil dieksekusi. Sedangkan pada bagian pojok kanan atas memperlihatkan informasi **Duration** yaitu durasi waktu eksekusi *job* tersebut selama **1 menit 22 detik**. Sebaliknya untuk melihat informasi detail dari *job deploy* maka klik pada *dropdown* di panel sebelah kanan bawah dan pilih **deploy**. Kemudian pilih tautan **deploy** yang muncul di bagian bawah *dropdown* tersebut, seperti terlihat pada gambar berikut:



Tampil informasi detail terkait eksekusi **job deploy**, seperti terlihat pada gambar berikut:



Pada bagian tengah dari halaman ini yaitu dengan latar belakang berwarna hitam menunjukkan hasil eksekusi setiap *task* dari *Ansible playbook* di lingkungan **production network** dari lab **NetDevOps** pada PNETLab. Selain itu juga pada bagian sebelah kiri bawah terdapat pesan **Job succeeded** yang menginformasikan bahwa *job* telah berhasil dieksekusi. Sedangkan pada bagian pojok kanan atas memperlihatkan informasi **Duration** yaitu durasi waktu eksekusi *job* tersebut selama **1 menit 16 detik**.

## F. Memverifikasi Hasil Proses Otomatisasi Penerapan Konfigurasi NAT, VLAN Dan DHCP Server Pada Lab NETDEVOPS Di PNETLAB

Adapun langkah-langkah untuk memverifikasi hasil dari proses otomatisasi penerapan konfigurasi NAT, VLAN dan DHCP Server pada lab **NetDevOps** di PNETLab adalah sebagai berikut:

1. Berpindah ke tab **PNETLab** pada *browser* yang digunakan.
2. Melakukan **DHCP Request** ulang pada setiap *node VPCS* yang terdapat di lingkungan **test network** dari PNETLAB yaitu **MKT1\_Test**, **MKT2\_Test**, **HRD1\_Test**, **HRD2\_Test** dan **SLS1\_Test** serta **SLS2\_Test** dengan mengeksekusi perintah `ip dhcp -r` pada *console* dari *node VPCS*

tersebut. Pastikan setiap **VPCS** telah berhasil memperoleh pengalamatan IP secara dinamis dari **DHCP Server**. Terlampir cuplikan hasil eksekusi dari perintah tersebut pada salah satu VPCS yaitu **SLS2\_Test**.



```

Welcome to Virtual PC Simulator, version 1.0 (0.8c)
Dedicated to Daling.
Build time: Dec 31 2016 01:22:17
Copyright (c) 2007-2015, Paul Meng (mirnshi@gmail.com)
All rights reserved.

VPCS is free software, distributed under the terms of the "BSD" licence.
Source code and license can be found at vpcs.sf.net.
For more information, please visit wiki.freecode.com.cn.
Modified version supporting unetlab by unetlab team

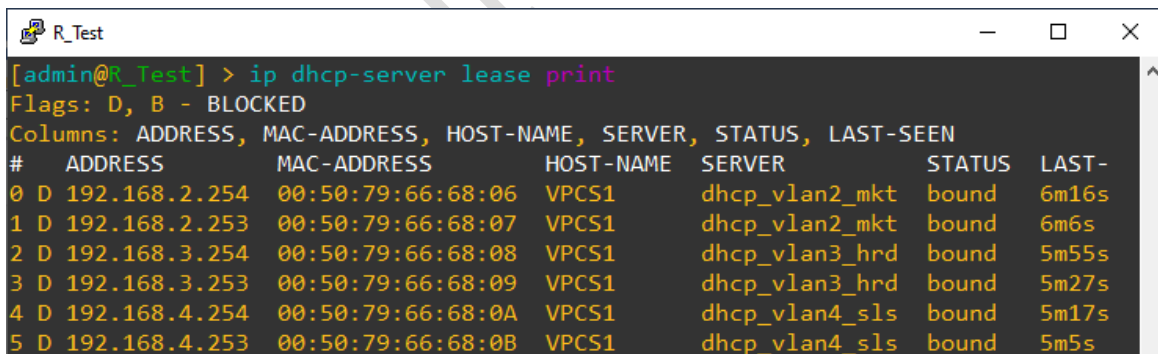
Press '?' to get help.

VPCS> ip dhcp -r
DORA IP 192.168.4.253/24 GW 192.168.4.1

```

Terlihat **SLS2\_Test** memperoleh alamat IP **192.168.4.253/24**.

3. Melakukan verifikasi pengalamatan IP yang telah disewakan oleh *router R\_Test* ke **client VPCS** dari setiap VLAN dengan mengeksekusi perintah **ip dhcp-server lease print** pada *console* dari *node R\_Test*.



```

[admin@R_Test] > ip dhcp-server lease print
Flags: D, B - BLOCKED
Columns: ADDRESS, MAC-ADDRESS, HOST-NAME, SERVER, STATUS, LAST-SEEN
ADDRESS MAC-ADDRESS HOST-NAME SERVER STATUS LAST-SEEN
0 D 192.168.2.254 00:50:79:66:68:06 VPCS1 dhcp_vlan2_mkt bound 6m16s
1 D 192.168.2.253 00:50:79:66:68:07 VPCS1 dhcp_vlan2_mkt bound 6m6s
2 D 192.168.3.254 00:50:79:66:68:08 VPCS1 dhcp_vlan3_hrd bound 5m55s
3 D 192.168.3.253 00:50:79:66:68:09 VPCS1 dhcp_vlan3_hrd bound 5m27s
4 D 192.168.4.254 00:50:79:66:68:0A VPCS1 dhcp_vlan4_sls bound 5m17s
5 D 192.168.4.253 00:50:79:66:68:0B VPCS1 dhcp_vlan4_sls bound 5m5s

```

4. Melakukan **DHCP Request** ulang pada setiap *node VPCS* yang terdapat di lingkungan **production network** dari PNETLAB yaitu **MKT1\_Production**, **MKT2\_Production**, **HRD1\_Production**, **HRD2\_Production** dan **SLS1\_Production** serta **SLS2\_Production** dengan mengeksekusi perintah **ip dhcp -r** pada *console* dari *node VPCS* tersebut. Pastikan setiap **VPCS** telah berhasil memperoleh pengalamatan IP secara dinamis dari **DHCP Server**. Terlampir cuplikan hasil eksekusi dari perintah tersebut pada salah satu VPCS yaitu **SLS2\_Test**.

```

Welcome to Virtual PC Simulator, version 1.0 (0.8c)
Dedicated to Daling.
Build time: Dec 31 2016 01:22:17
Copyright (c) 2007-2015, Paul Meng (mirnshi@gmail.com)
All rights reserved.

VPCS is free software, distributed under the terms of the "BSD" licence.
Source code and license can be found at vpcs.sf.net.
For more information, please visit wiki.freecode.com.cn.
Modified version supporting unetlab by unetlab team

Press '?' to get help.

VPCS> ip dhcp -r
DORA IP 192.168.14.253/24 GW 192.168.14.1

```

Terlihat **SLS2\_Production** memperoleh alamat IP **192.168.14.253/24**.

5. Melakukan verifikasi pengalamatan IP yang telah disewakan oleh *router R\_Production* ke *client VPCS* dari setiap VLAN dengan mengeksekusi perintah **ip dhcp-server lease print** pada *console* dari *node R\_Production*.

```

[admin@R_Production] > ip dhcp-server lease print
Flags: D, B - BLOCKED
Columns: ADDRESS, MAC-ADDRESS, HOST-NAME, SERVER, STATUS, LAST-SEEN
ADDRESS MAC-ADDRESS HOST-NAME SERVER STATUS LAST-SEEN
0 D 192.168.12.254 00:50:79:66:68:0C VPCS1 dhcp_vlan2_mkt bound 1m52s
1 D 192.168.12.253 00:50:79:66:68:0D VPCS1 dhcp_vlan2_mkt bound 1m37s
2 D 192.168.13.254 00:50:79:66:68:0E VPCS1 dhcp_vlan3_hrd bound 1m22s
3 D 192.168.13.253 00:50:79:66:68:0F VPCS1 dhcp_vlan3_hrd bound 1m16s
4 D 192.168.14.254 00:50:79:66:68:10 VPCS1 dhcp_vlan4_sls bound 1m11s
5 D 192.168.14.253 00:50:79:66:68:11 VPCS1 dhcp_vlan4_sls bound 1m5s

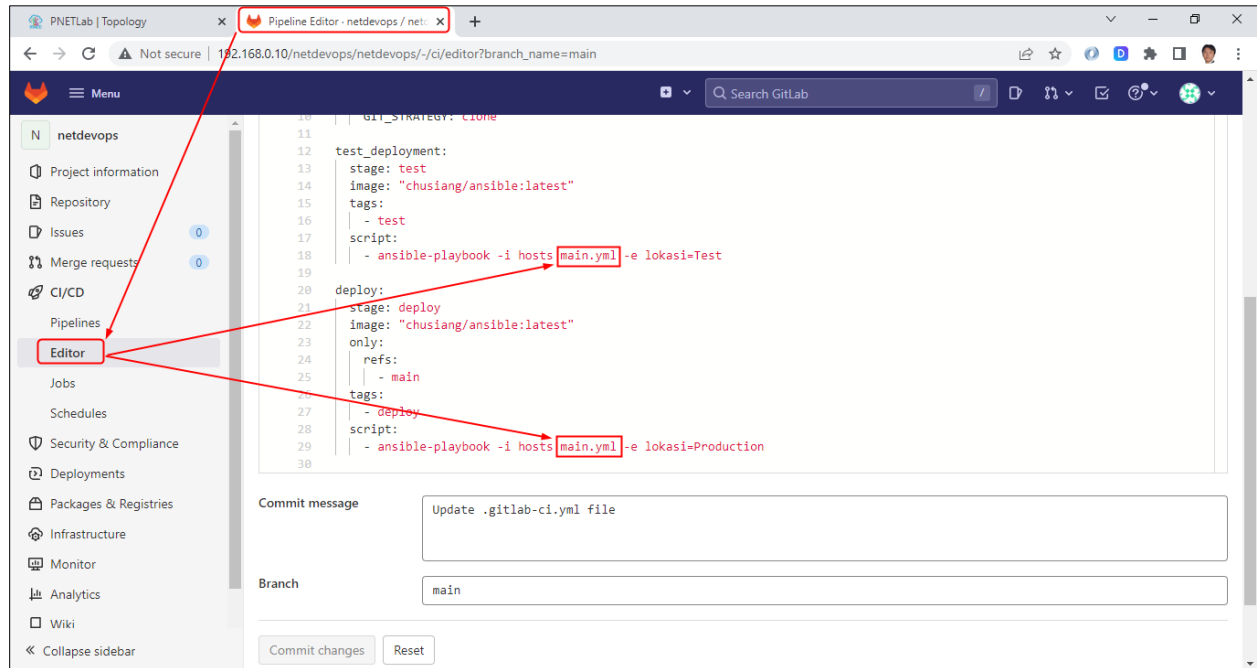
```

## G. Mengubah Gitlab CI/CD Pipeline Untuk Mengotomatisasi Penghapusan Konfigurasi NAT, VLAN Dan DHCP Server Pada Lab NETDEVOPS Di PNETLAB

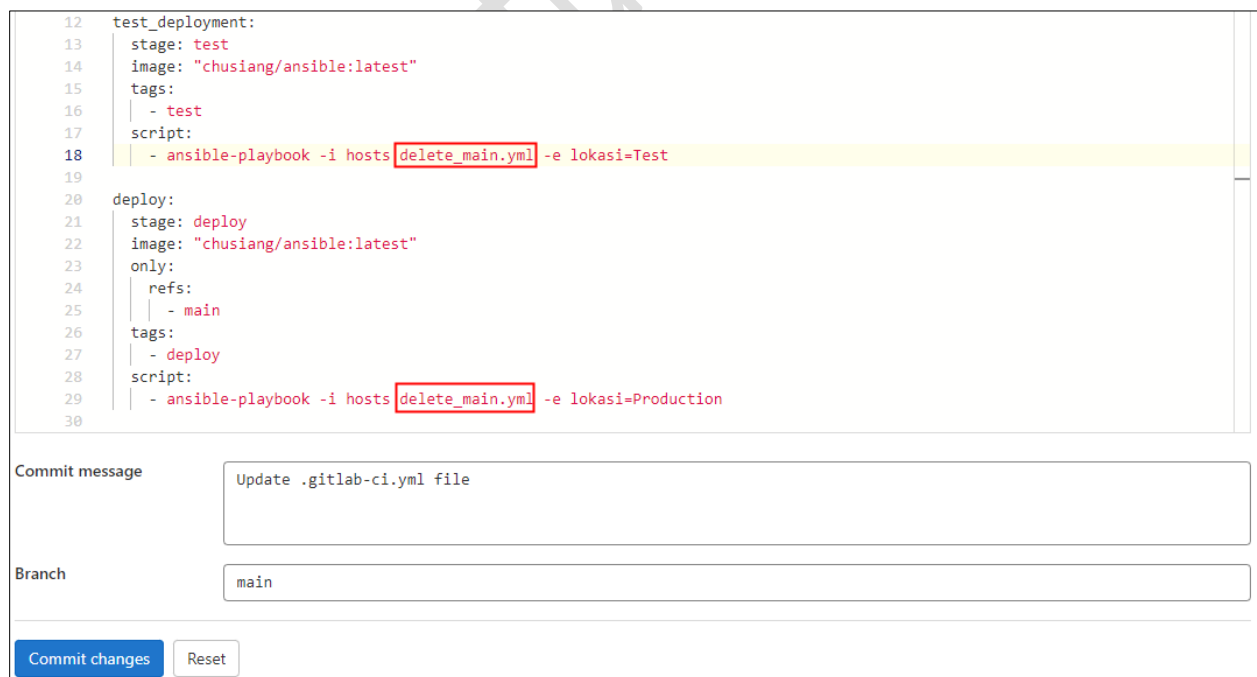
Untuk mengotomatisasi proses penghapusan konfigurasi NAT, VLAN dan DHCP Server melalui **GitLab CI/CD Pipeline** maka diperlukan perubahan konfigurasi pada file **.gitlab-ci.yml**. Adapun langkah-langkah untuk melakukan perubahan tersebut adalah sebagai berikut:

1. Berpindah ke tab **GitLab** pada *browser* yang digunakan.

2. Pada panel menu sebelah kiri pilih submenu **Editor** dari **CI/CD** dan lakukan perubahan pada baris **18** dan **29** yaitu mengubah nama *file playbook* yang dieksekusi dari **main.yml** menjadi **delete\_main.yml**, seperti terlihat pada gambar berikut:

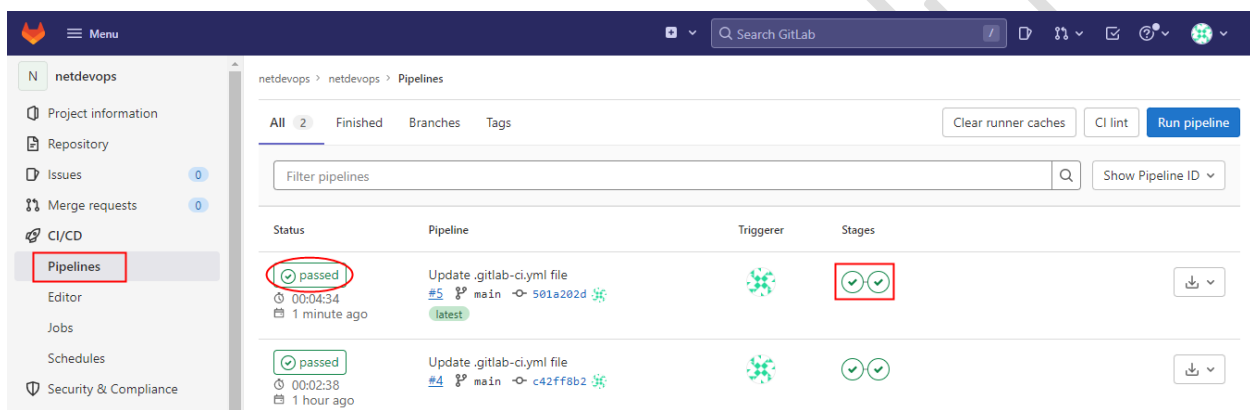


Hasil akhir dari perubahan tersebut akan terlihat seperti pada gambar berikut:



Tekan tombol **Commit changes** untuk menyimpan perubahan dan secara otomatis *GitLab CI/CD pipeline* akan terpicu untuk mengeksekusi **jobs** pada setiap **stage**. Selanjutnya akan tampil pesan bahwa *Pipeline* telah berjalan.

3. Pada panel menu sebelah kiri pilih submenu **Pipelines** dari **CI/CD** maka pada panel detail sebelah kanan akan tampil informasi terkait **status** dan **stage** dari *pipeline* yang sedang dieksekusi. Tunggu beberapa saat hingga centang penanda setiap **stages** yaitu **test** dan **deploy** berwarna hijau yang menandakan bahwa eksekusi **job** di dalam **stage** tersebut sukses dilakukan, seperti terlihat pada gambar berikut:



## H. Memverifikasi Hasil Proses Otomatisasi Penghapusan Konfigurasi NAT, VLAN Dan DHCP Server Pada Lab NETDEVOPS Di PNETLAB

Adapun langkah-langkah untuk memverifikasi hasil dari proses otomatisasi penghapusan konfigurasi NAT, VLAN dan DHCP Server pada lab **NetDevOps** di PNETLab adalah sebagai berikut:

1. Berpindah ke tab **PNETLab** pada *browser* yang digunakan.
2. Memverifikasi penonaktifan *VLAN Filtering* pada *interface bridge*, penghapusan *interface bridge VLAN* dan penghapusan *interface bridge port* pada lingkungan **test network** dengan mengeksekusi 3 (tiga) perintah meliputi **interface bridge print**, **interface bridge vlan print** dan **interface bridge port print** pada *console* dari *node SW\_Test*.

```

SW_Test
[admin@SW_Test] > interface bridge print
Flags: X - disabled, R - running
0 R name="BR_Test" mtu=auto actual-mtu=1500 l2mtu=65535 arp=enabled
 arp-timeout=auto mac-address=50:6E:D8:00:02:06 protocol-mode=rstp
 fast-forward=yes igmp-snooping=no auto-mac=yes ageing-time=5m
 priority=0x8000 max-message-age=20s forward-delay=15s
 transmit-hold-count=6 vlan-filtering=no dhcp-snooping=no
[admin@SW_Test] > interface bridge vlan print

[admin@SW_Test] > interface bridge port print
Columns: INTERFACE, BRIDGE, HW, PVID, PRIORITY, PATH-COST, INTERNAL-PATH-COST, H
ORIZON
INTERFACE BRIDGE HW PVID PRIORITY PATH-COST IN HORIZON
0 ether7 BR_Test yes 1 0x80 10 10 none
[admin@SW_Test] >

```

Terlihat ketiga operasi penonaktifan atau penghapusan berhasil dilakukan.

- Memverifikasi penghapusan IP DHCP Lease, IP DHCP-Server Network, IP DHCP-Server, IP Pool, pengalamatan IP pada setiap interface VLAN, interface VLAN, IP Firewall NAT dan pengaturan DHCP Client pada interface ether3 pada lingkungan test network dengan mengeksekusi 8 (delapan) perintah meliputi `ip dhcp-server lease print`, `ip dhcp-server network print`, `ip dhcp-server print`, `ip pool print`, `ip address print`, `interface vlan print`, `ip firewall nat print` dan `ip dhcp-client print` pada console dari node R\_Test.

```

R_Test
[admin@R_Test] > ip dhcp-server lease print

[admin@R_Test] > ip dhcp-server network print

[admin@R_Test] > ip dhcp-server print

[admin@R_Test] > ip pool print

[admin@R_Test] > ip address print
Columns: ADDRESS, NETWORK, INTERFACE
ADDRESS NETWORK INTERFACE
0 192.168.1.1/24 192.168.1.0 ether1
1 192.168.0.1/30 192.168.0.0 ether2
[admin@R_Test] > interface vlan print

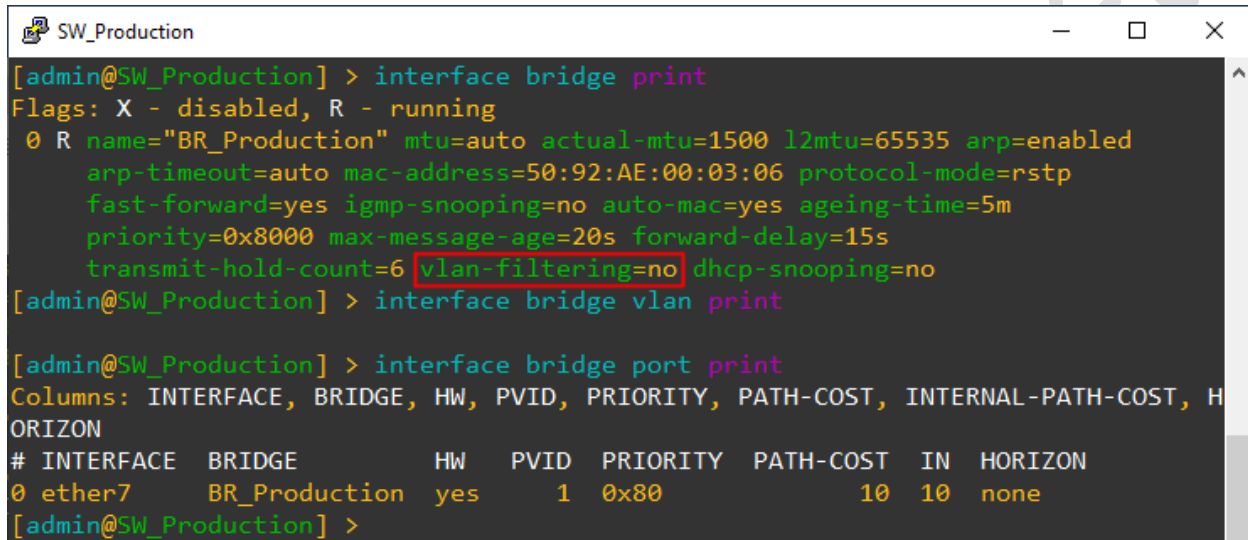
[admin@R_Test] > ip firewall nat print
Flags: X - disabled, I - invalid; D - dynamic
[admin@R_Test] > ip dhcp-client print

[admin@R_Test] >

```

Terlihat kedelapan operasi penghapusan berhasil dilakukan.

- Memverifikasi penonaktifan *VLAN Filtering* pada *interface bridge*, penghapusan *interface bridge VLAN* dan penghapusan *interface bridge port* pada lingkungan **production network** dengan mengeksekusi 3 (tiga) perintah meliputi **interface bridge print**, **interface bridge vlan print** dan **interface bridge port print** pada *console* dari *node SW\_Production*.

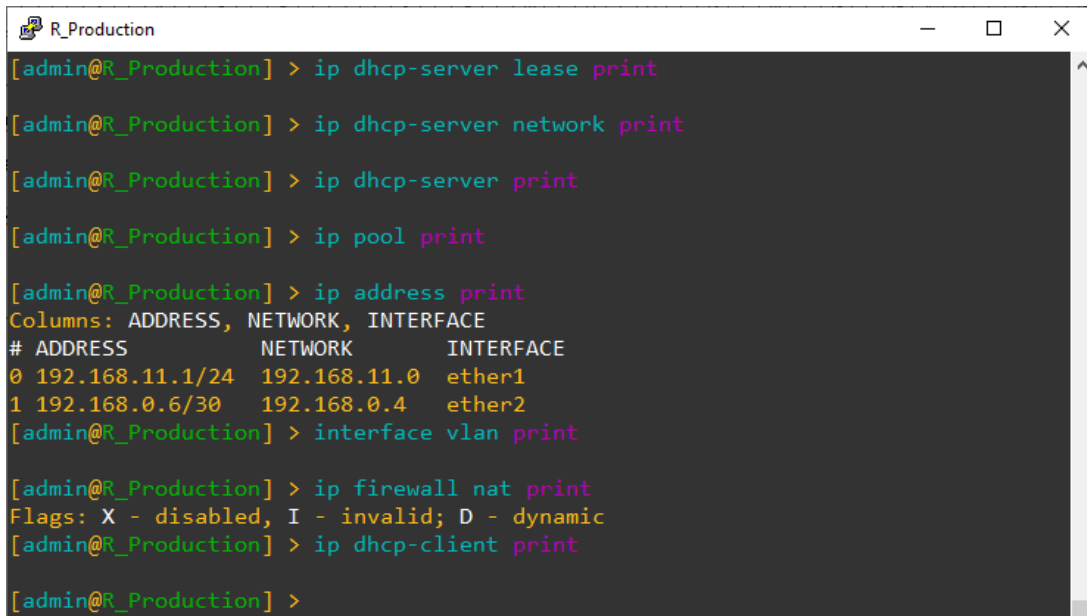


```
[admin@SW_Production] > interface bridge print
Flags: X - disabled, R - running
0 R name="BR_Production" mtu=auto actual-mtu=1500 l2mtu=65535 arp=enabled
 arp-timeout=auto mac-address=50:92:AE:00:03:06 protocol-mode=rstp
 fast-forward=yes igmp-snooping=no auto-mac=yes ageing-time=5m
 priority=0x8000 max-message-age=20s forward-delay=15s
 transmit-hold-count=6 vlan-filtering=no dhcp-snooping=no
[admin@SW_Production] > interface bridge vlan print

[admin@SW_Production] > interface bridge port print
Columns: INTERFACE, BRIDGE, HW, PVID, PRIORITY, PATH-COST, INTERNAL-PATH-COST, H
ORIZON
INTERFACE BRIDGE HW PVID PRIORITY PATH-COST IN HORIZON
0 ether7 BR_Production yes 1 0x80 10 10 none
[admin@SW_Production] >
```

Terlihat ketiga operasi penonaktifan atau penghapusan berhasil dilakukan.

- Memverifikasi penghapusan **IP DHCP Lease**, **IP DHCP-Server Network**, **IP DHCP-Server**, **IP Pool**, pengalamatan IP pada setiap **interface VLAN**, **interface VLAN**, **IP Firewall NAT** dan pengaturan **DHCP Client** pada *interface ether3* pada lingkungan **production network** dengan mengeksekusi 8 (delapan) perintah meliputi **ip dhcp-server lease print**, **ip dhcp-server network print**, **ip dhcp-server print**, **ip pool print**, **ip address print**, **interface vlan print**, **ip firewall nat print** dan **ip dhcp-client print** pada *console* dari *node R\_Production*.



```

[admin@R_Production] > ip dhcp-server lease print

[admin@R_Production] > ip dhcp-server network print

[admin@R_Production] > ip dhcp-server print

[admin@R_Production] > ip pool print

[admin@R_Production] > ip address print
Columns: ADDRESS, NETWORK, INTERFACE
ADDRESS NETWORK INTERFACE
0 192.168.11.1/24 192.168.11.0 ether1
1 192.168.0.6/30 192.168.0.4 ether2
[admin@R_Production] > interface vlan print

[admin@R_Production] > ip firewall nat print
Flags: X - disabled, I - invalid; D - dynamic
[admin@R_Production] > ip dhcp-client print

[admin@R_Production] >

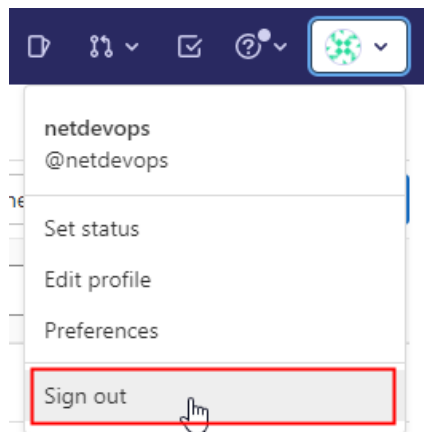
```

Terlihat kedelapan operasi penghapusan berhasil dilakukan.

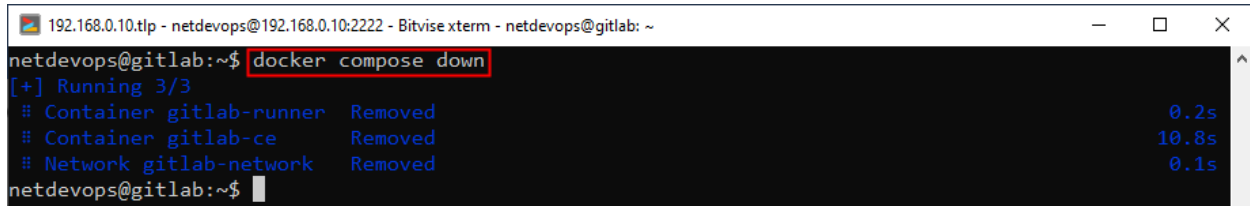
#### I. Menghentikan Penggunaan Gitlab Container Dan Lab NETDEVOPS Pada PNETLAB

Adapun langkah-langkah untuk menghentikan penggunaan *GitLab Container* dan lab **NetDevOps** pada PNETLab adalah sebagai berikut:

1. Berpindah ke tab **GitLab** pada *browser* yang digunakan.
2. Lakukan **Sign out** dari **Gitlab** untuk keluar dari penggunaan **GitLab Web GUI/Portal** dengan cara mengakses menu *dropdown* di bagian pojok kanan atas halaman dan memilih **Sign out**, seperti terlihat pada gambar berikut:



3. Berpindah ke **Bitvise xterm** dari **VM GitLab NetDevOps** dan lakukan eksekusi perintah **docker compose down** untuk menghentikan **GitLab Runner** dan **GitLab CE Docker Container** serta menghapus **gitlab-network**, seperti terlihat pada gambar berikut:

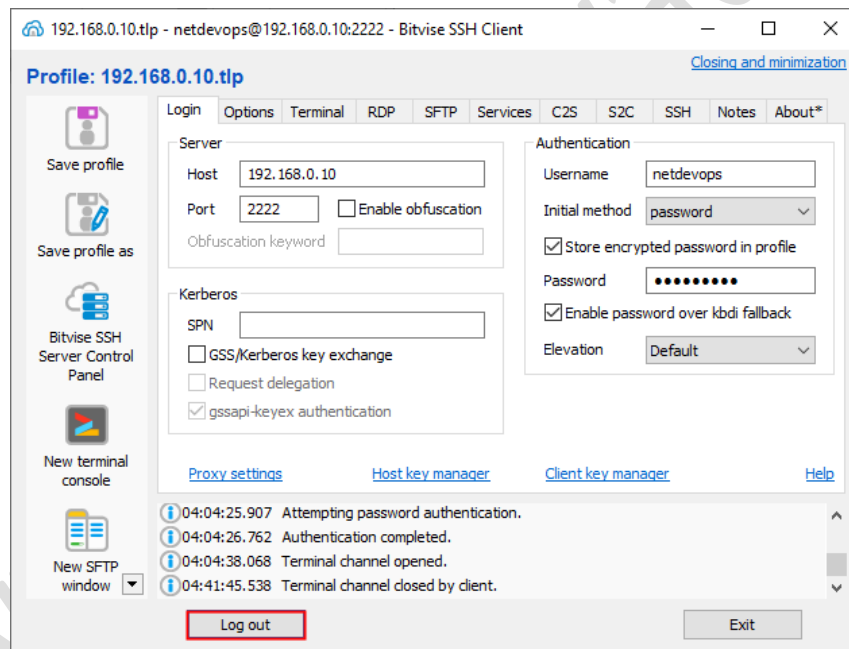


```

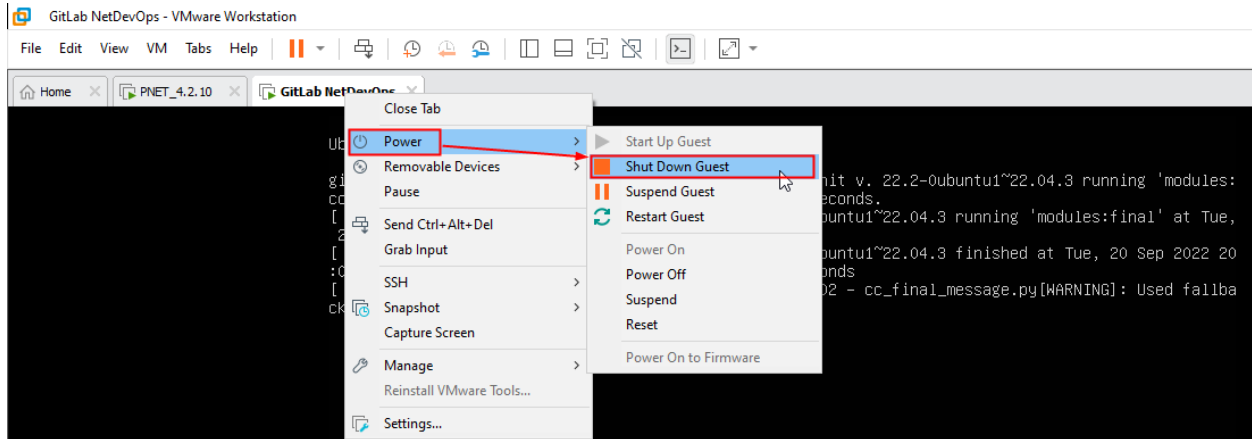
192.168.0.10.tlp - netdevops@192.168.0.10:2222 - Bitvise xterm - netdevops@gitlab: ~
netdevops@gitlab:~$ docker compose down
[+] Running 3/3
 # Container gitlab-runner Removed 0.2s
 # Container gitlab-ce Removed 10.8s
 # Network gitlab-network Removed 0.1s
netdevops@gitlab:~$

```

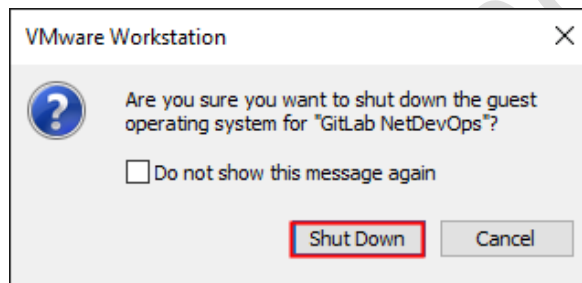
4. Tutup kotak dialog **Bitvise xterm** dari **VM GitLab NetDevOps**.
5. Berpindah ke kotak dialog **Bitvise SSH Client** dan klik tombol **Logout**.
6. Tutup kotak dialog tersebut, seperti terlihat pada gambar berikut:



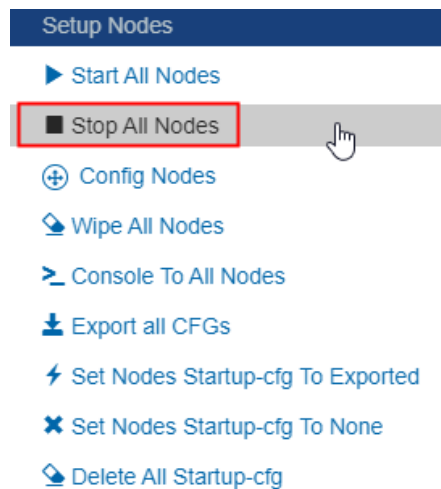
7. Berpindah ke kotak dialog aplikasi **VMWare Workstation**. Lakukan klik kanan pada nama pengenal dari **VM** yaitu **GitLab NetDevOps** dan menu yang tampil, pilih **Power > Shut Down Guest** untuk mematikan VM tersebut, seperti terlihat pada gambar berikut:



Tampil kotak dialog konfirmasi **Are you sure you want to shut down the guest operating system for "GitLab NetDevOps"?**. Klik tombol **Shutdown**, seperti terlihat pada gambar berikut:

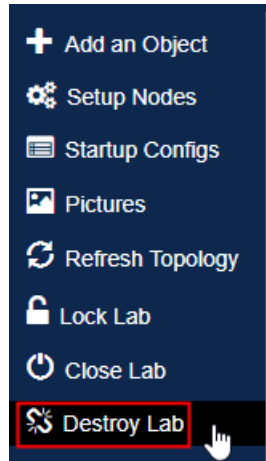


8. Menghentikan keseluruhan *node* yang berjalan pada lab **NetDevOps** di PNETLab dengan cara mengakses kembali *browser*. Pada halaman *topology* yang tampil, gerakkan penunjuk (kursor) *mouse* ke sebelah kiri menuju *side bar* yang diperkecil sehingga memperluas *side bar* tersebut dan pilih **Setup Nodes**. Tampil menu **Setup Nodes** dan pilih pada **Stop All Nodes**, seperti terlihat pada gambar berikut:

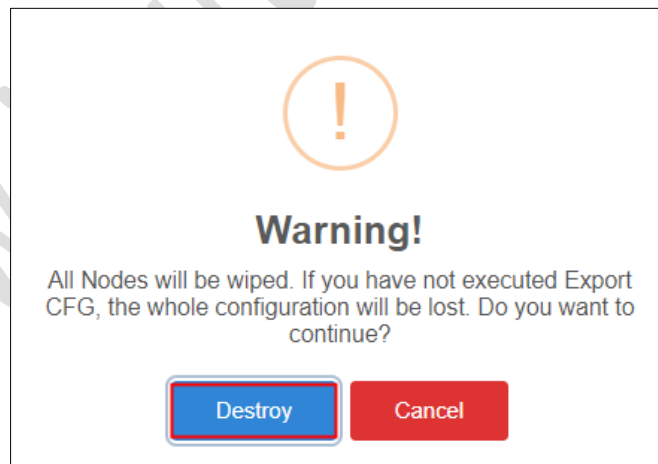


Tunggu hingga keseluruhan *node* pada lab tersebut berhasil dihentikan dimana ditandai dengan simbol dari setiap *node* berubah dari warna biru menjadi abu-abu. Selain itu pada bagian **Notifications** sebelah kanan dari halaman *topology* juga memunculkan informasi setiap *node* yang dihentikan.

Selanjutnya gerakkan kembali kursor menuju *side bar* dan pilih **Destroy Lab**, seperti terlihat pada gambar berikut:

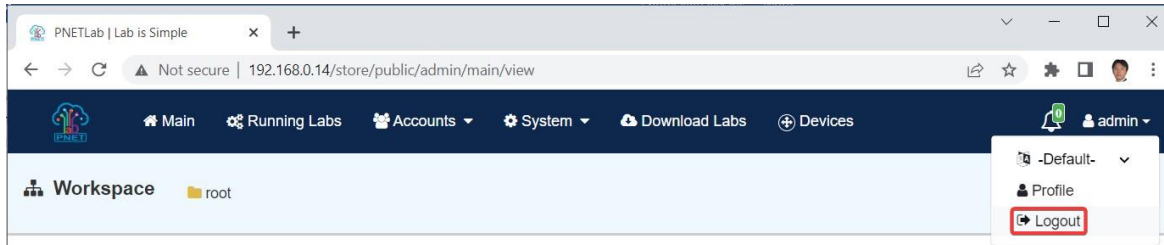


Tampil kotak dialog **Warning!** yang memberikan peringatan bahwa seluruh *node* akan di **wipe** sehingga seluruh konfigurasi akan dihapus, seperti terlihat pada gambar berikut:

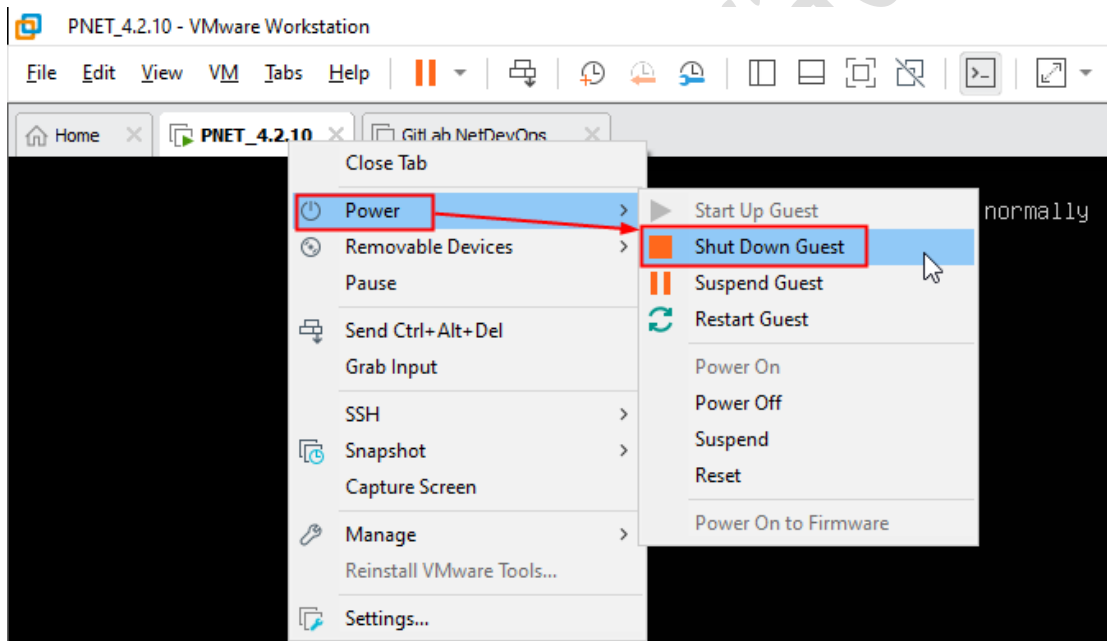


Apabila diperlukan agar konfigurasi dari setiap *node* tersimpan maka lakukan proses **Export all CFGs** terlebih dahulu sebelum melakukan **Destroy Lab**. Untuk melanjutkan proses maka klik tombol **Destroy**.

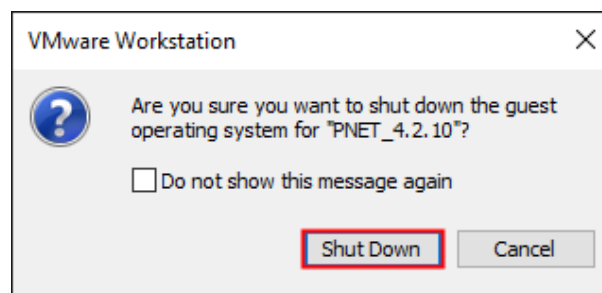
Selanjutnya tampil halaman utama (**main**) dari PNETLab dan pada *dropdown* **admin** di pojok kanan atas, pilih **Logout** untuk keluar dari penggunaan PNETLab, seperti terlihat pada gambar berikut:



- Matikan **VM PNETlab** dengan cara klik kanan pada nama pengenal dari **VM** yaitu **PNETLab 4.2.10** di aplikasi **VMWare Workstation**. Pada menu yang tampil, pilih **Power > Shut Down Guest**, seperti terlihat pada gambar berikut:.



Tampil kotak dialog konfirmasi **Are you sure you ant to shut down the guest operating system for "PNET\_4.2.10"?**. Klik tombol **Shutdown**, seperti terlihat pada gambar berikut:



10. Tutup kotak dialog aplikasi **VMWare Workstation**.

[www.iputuhariyadi.net](http://www.iputuhariyadi.net)

**DAFTAR REFERENSI**

1. Hank Preston, 2017, Part 1: Embrace NetDevOps, Say Goodbye to a “Culture of Fear”, Cisco, <https://blogs.cisco.com/developer/embrace-netdevops-part-1>
2. Eric Chou, 2022, NetOps (DevOps for Network Engineers): Automating Networks, LinkedIn Learning, <https://www.linkedin.com/learning/netops-devops-for-network-engineers-automating-networks>
3. Bojana Dobran, 2020, What is DevOps Pipeline & How to Build One, PhoenixNAP, <https://phoenixnap.com/blog/devops-pipeline>
4. Athena Osanich, 2020, Understanding the DevOps Pipeline & How to Build One, HubSpot, <https://blog.hubspot.com/website/devops-pipeline>
5. Hiren Dhaduk, 2022, DevOps Lifecycle: 7 Phases Explained in Detail with Examples, SIMFORM, <https://www.simform.com/blog/devops-lifecycle/>
6. PNETLab, 2022, <https://pnetlab.com/>
7. GitLab, 2022, <https://about.gitlab.com/>
8. Docker, 2022, <https://www.docker.com/>
9. Ansible Documentation, 2022, <https://docs.ansible.com/>
10. Reshma Ahmed, 2021, *What Is Ansible? – Configuration Management And Automation With Ansible*, Edureka, <https://www.edureka.co/blog/what-is-ansible/>
11. Mikrotik, 2022, Mikrotik Documentation, [https://wiki.mikrotik.com/wiki/Main\\_Page](https://wiki.mikrotik.com/wiki/Main_Page)
12. Tutorialspoint, 2022, Ansible-Playbooks, [https://www.tutorialspoint.com/ansible/ansible\\_playbooks.htm](https://www.tutorialspoint.com/ansible/ansible_playbooks.htm)

## TENTANG PENULIS



### **I Putu Hariyadi**

adalah dosen di program studi Ilmu Komputer, [Universitas Bumigora](#), Mataram, Nusa Tenggara Barat (NTB). Penulis sangat antusias untuk mendalami dunia Teknologi Informasi & Komunikasi (TIK). Memiliki ketertarikan pada bidang Jaringan Komputer, *Network Programmability*, *Cloud Computing*, *Pemrograman Web* dan Keamanan Sistem Informasi serta Sistem Temu Kembali Informasi (*Information Retrieval*). Profil detail

terkait penulis dapat diakses di [Linkedin](#).

Sebagian besar pengalaman penulis ketika mengeksplorasi bidang tersebut dituangkan pada situs pribadi yang beralamat di <https://www.iputuhariyadi.net>. Untuk korespondensi dapat menghubungi penulis melalui email di alamat: [admin@iputuhariyadi.net](mailto:admin@iputuhariyadi.net) atau [putu.hariyadi@universitasbumigora.ac.id](mailto:putu.hariyadi@universitasbumigora.ac.id). Selain itu juga dapat melalui media sosial [Facebook](#), [Instagram](#), atau [Twitter](#).