

IMPLEMENTASI CONTINUOUS INTEGRATION/CONTINUOUS DEPLOYMENT UNTUK MENGOTOMATISASI MANAJEMEN KONFIGURASI INFRASTRUKTUR JARINGAN KOMPUTER

(IMPLEMENTATION OF CONTINUOUS INTEGRATION/CONTINUOUS DEPLOYMENT TO AUTOMATE COMPUTER NETWORK INFRASTRUCTURE CONFIGURATION MANAGEMENT)

I Putu Hariyadi¹⁾, Raisul Azhar²⁾, Heroe Santoso³⁾, Khairan Marzuki⁴⁾, I Made Yadi Dharma⁵⁾
^{1,2,3,4,5)} Universitas Bumigora Mataram

Jl. Ismail Marzuki 22, Cilinaya, Cakranegara, Kota Mataram, Nusa Tenggara Barat
 e-mail: putu.hariyadi@universitasbumigora.ac.id¹⁾, raisulazhar@universitasbumigora.ac.id²⁾,
hero.santoso@universitasbumigora.ac.id³⁾, khairan.marzuki@universitasbumigora.ac.id⁴⁾,
yadi_dharma@universitasbumigora.ac.id⁵⁾

ABSTRAK

Infrastruktur jaringan komputer menjadi salah satu komponen pendukung kelancaran operasional bagi perusahaan yang memanfaatkan Teknologi Informasi dan Komunikasi (TIK). Perusahaan dituntut untuk terus berinovasi agar dapat terus berkembang sehingga memerlukan berbagai penyesuaian termasuk pada perubahan kebijakan konfigurasi infrastruktur jaringan komputer. Penerapan perubahan konfigurasi pada infrastruktur jaringan komputer jika dilakukan secara manual maka menjadi tidak efisien dan efektif. Selain itu rentan terhadap kesalahan sebagai dampak human error sehingga mengakibatkan downtime. Penerapan Continuous Integration/Continuous Deployment (CI/CD) dapat menjadi solusi untuk mengatasi permasalahan tersebut. Penelitian ini menganalisa penerapan CI/CD pada infrastruktur jaringan yang disimulasikan menggunakan tool PNETLab dan GitLab CI/CD serta Ansible Configuration Management. Berdasarkan uji coba yang dilakukan sebanyak lima kali maka dapat disimpulkan bahwa Ansible Playbook yang telah dibuat dan diintegrasikan dengan GitLab CI/CD pipeline dapat digunakan untuk mengotomatisasi manajemen konfigurasi infrastruktur jaringan berbasis MikroTik yang disimulasikan pada PNETLab. CI/CD dapat mempercepat penerapan perubahan konfigurasi pada perangkat jaringan melalui proses yang diotomatisasi. Pengujian penambahan konfigurasi di test network memerlukan waktu rata-rata 82,6 detik. Sedangkan di production network memerlukan waktu rata-rata 80,2 detik. Sebaliknya pengujian penghapusan konfigurasi di test network memerlukan waktu rata-rata 56,6 detik. Sedangkan di production network rata-rata 54,4 detik. Selain itu penerapan CI/CD dapat meminimalkan kesalahan karena sebelum konfigurasi diterapkan ke production network maka konfigurasi tersebut diuji terlebih dahulu di test network secara otomatis. Penerapan konfigurasi ke production network dilakukan hanya jika pengujian terhadap konfigurasi di test network berhasil dilakukan. Mekanisme CI/CD tersebut dapat meningkatkan performansi dan reabilitas infrastuktur jaringan komputer.

Kata Kunci: Jaringan Komputer, Manajemen Konfigurasi, Continuous Integration, Continuous Deployment, Otomatisasi

ABSTRACT

Computer network infrastructure is one of the supporting components for smooth operations for companies that utilize Information and Communication Technology (ICT). Companies are required to continue to innovate to continue to grow so it requires various adjustments, including changes to the policy configuration of computer network infrastructure. Implementation of configuration changes to computer network infrastructure if done manually will be inefficient and ineffective. In addition, it is prone to errors as a result of human error resulting in downtime. The application of Continuous Integration/Continuous Deployment (CI/CD) can be a solution to overcome these problems. This study analyzes the implementation of CI/CD on network infrastructure which is simulated using the PNETLab and GitLab CI/CD tools as well as Ansible Configuration Management. Based on the tests conducted five times, it can be concluded that the Ansible Playbook that has been created and integrated with the GitLab CI/CD pipeline can be used to automate configuration management of MikroTik-based network infrastructure that is simulated in PNETLab. CI/CD can speed up the implementation of configuration changes on network devices through an automated process. Testing additional configurations on the test network takes an average of 82.6 seconds. Meanwhile, on the production network, it takes an average of 80.2 seconds. On the other hand, testing the configuration deletion on the test network takes an average of 56.6 seconds. Meanwhile, on the production network, the average is 54.4 seconds. In addition, the application of CI/CD can minimize errors because before the configuration is applied to the production network, the configuration is tested first on the test network

automatically. Application of the configuration to the production network is carried out only if testing of the configuration on the test network is successful. The CI/CD mechanism can improve the performance and reliability of computer network infrastructure.

Keywords: *Computer Network, Configuration Management, Continuous Integration, Continuous Deployment, Automation*

I. PENDAHULUAN

Infrastruktur jaringan komputer menjadi salah satu komponen pendukung kelancaran operasional bagi perusahaan yang memanfaatkan Teknologi Informasi dan Komunikasi (TIK). Perusahaan bertumbuh dan dituntut untuk terus berinovasi. Inovasi tersebut memerlukan penyesuaian di berbagai bidang termasuk perubahan pada kebijakan konfigurasi infrastruktur jaringan komputer. Penerapan perubahan konfigurasi pada infrastruktur jaringan komputer dengan jumlah perangkat yang banyak dan kompleks jika dilakukan secara manual maka menjadi tidak efisien. Penambahan Sumber Daya Manusia (SDM) yang bertindak sebagai *Network Engineer (NE)* untuk membantu mempercepat penerapan konfigurasi menjadi solusi yang tidak efektif. Selain itu cara manual rentan terhadap kesalahan sebagai dampak kesalahan konfigurasi oleh pengguna (*human error*) sehingga mengakibatkan terjadinya gangguan operasional (*downtime*).

Otomatisasi jaringan menjadi solusi efisien di berbagai area [1]. Manajemen konfigurasi jaringan yang diterapkan secara otomatis meminimalisir kesalahan pada konfigurasi perangkat jaringan [2]. Metode ini dapat mempermudah tugas NE dalam mengkonfigurasi perangkat jaringan dalam jumlah banyak melalui *control node* [3]. Terdapat berbagai penelitian terkait otomatisasi konfigurasi jaringan meliputi *routing*[1], [3], manajemen *Virtual Local Area Network (VLAN)* berbasis *VLAN Trunking Protocol (VTP)* dan Layanan *Dynamic Host Configuration Protocol (DHCP)*[2], [4], penyediaan konfigurasi *Internet* dan manajemen *MikroTik* [5]. Kode program pada penelitian tersebut dibangun menggunakan bahasa pemrograman *Python* [5] dengan *library Paramiko* atau *Netmiko* [6] dan *tool* manajemen konfigurasi *Ansible* [2]–[4], [7], [8]. Antarmuka untuk mengotomatisasi konfigurasi jaringan pada penelitian tersebut berupa *Command Line Interface (CLI)* dan berbasis *web* menggunakan *Django*

Framework. Sedangkan *tool* simulasi untuk otomatisasi jaringan yang digunakan adalah GNS3 [4], [7] dan *PNETLab* [2].

Penerapan otomatisasi konfigurasi jaringan pada penelitian terdahulu dilakukan secara langsung pada jaringan produksi. Apabila terdapat kesalahan pada kode program otomatisasi konfigurasi maka akan mengakibatkan gangguan pada jaringan produksi tersebut. Untuk itu diperlukan pengujian pada kode program tersebut terlebih dahulu di lingkungan jaringan non produksi atau *test network*. Apabila pengujian berhasil maka kode program tersebut dapat diterapkan ke jaringan produksi. Penelitian ini bertujuan menggunakan *Continuous Integration (CI)* dan *Continuous Deployment (CD)* untuk mengotomatiskan pengujian dan penerapan manajemen konfigurasi pada infrastruktur jaringan komputer *on-premise* baik pada lingkungan *test network* maupun *production network*. CI merupakan praktik untuk mengintegrasikan keseluruhan perubahan kode program ke *repository* lebih awal dan sesering mungkin serta menguji setiap perubahan secara otomatis sehingga dapat mengidentifikasi dan memperbaiki kesalahan ketika pengembangan [9]. Sedangkan CD merupakan fase dimana kode dikirim ke jaringan produksi untuk diterapkan secara otomatis tanpa adanya intervensi atau konfirmasi manual [9].

II. STUDI PUSTAKA

Terdapat beberapa penelitian terkait CI/CD yang dilakukan oleh peneliti sebelumnya. Penelitian [10] menyajikan otomatisasi CI/CD *pipeline* untuk menerapkan aplikasi *web* berbasis *Java* di *Amazon Web Service (AWS)* yang menggunakan *Ansible* pada proses CD. Pemanfaatan *Ansible* memungkinkan untuk mengirimkan perintah secara langsung ke *cluster Kubernetes* dan memastikan skalabilitas yang lebih baik secara keseluruhan. Hasil percobaan pada penelitian ini menunjukkan bahwa solusi yang diusulkan adalah andal dan memiliki waktu henti 0 detik serta mudah diskalakan dan cepat. Setiap perubahan pada kode sumber aplikasi akan secara otomatis terdeteksi dan memicu keseluruhan

rangkaian peristiwa yang terdiri dari enam teknologi berbeda, hanya dalam waktu 37,6 detik. Selain itu setiap kegagalan *job* pada *Jenkins* saat menerapkan versi aplikasi yang baru akan terdeteksi dan sistem bergulir kembali ke versi stabil terakhir.

Penelitian [11] berfokus pada penerapan otomatis menggunakan teknik CI/CD untuk mengembangkan aplikasi *real-time* menggunakan berbagai bahasa pemrograman yang diimplementasikan pada infrastruktur *cloud* menggunakan *AWS code pipeline*. Kode sumber aplikasi disimpan pada *GitHub* dan *Amazon S3* serta diuji secara otomatis menggunakan *AWS code pipeline*. Hasil dari penelitian ini menunjukkan penerapan konsep CI/CD untuk penyebaran otomatis pada aplikasi dapat mempercepat proses mulai dari pengembangan aplikasi yang mendukung berbagai platform bahasa pemrograman hingga ke produksi yaitu dengan waktu rata-rata penerapan 60 detik. *Amazon S3* digunakan sebagai *repository* kode sumber yang memberikan keuntungan penyebaran lebih cepat daripada *repository* eksternal seperti *GitHub*. Selain itu penerapan *Cloudformation* lebih efisien daripada *Elastic Beanstalk* tetapi dari perspektif kemudahan penggunaan *Elastic Beanstalk* lebih unggul.

Penelitian [9] membandingkan *tool* CI/CD yang diintegrasikan dengan platform *cloud* yaitu *GitLab* dan *Jenkins* untuk menyebarkan *dockerized microservices* pada platform *cloud* *AWS*. Performansi dari *tool* CI/CD yang dievaluasi meliputi waktu tanggap, waktu penyebaran, kemudahan penggunaan dan pengaturan lingkungan kerja dari CI. Hasil dari penelitian tersebut adalah *Jenkins* dan *GitLab* menyediakan dukungan yang baik untuk CI dan pemilihan *tool* mana yang digunakan berdasarkan pada kebutuhan proyek. *Jenkins* menjadi pilihan yang tepat ketika ingin menerapkan CI untuk proyek skala besar pada perusahaan. Sebaliknya *GitLab* menjadi pilihan yang baik untuk implementasi CI pada proyek berbasis layanan mikro yang lebih sederhana, canggih dan kecil. *Jenkins* merupakan *tool* yang mudah digunakan dan dipelajari karena mendukung berbagai *plugin* berbeda sehingga memudahkan pengembang untuk mengkonfigurasi proyek untuk mengotomatisasi penyebaran. Namun seiring dengan bertumbuhnya aplikasi dan meningkatnya jumlah *plugin* yang digunakan maka akan membuat *Jenkins* sulit untuk dikelola. Hal tersebut berbanding terbalik dengan *GitLab* karena

keseluruhan konfigurasi *pipeline* dilakukan pada sebuah *file* *YAML* yang mudah dipahami dan dimodifikasi terlepas dari seberapa besar proyek bertumbuh. *GitLab* merupakan sistem kontrol versi berbasis *git* dengan antarmuka *web* dan bersifat *open source* serta menyediakan fitur CI/CD *pipeline*. *Pipeline* dipicu ketika perubahan baru di *commit* ke *GitLab*. *Pipeline* pada *GitLab* berupa *file* *YAML* yang memuat serangkaian *stage*. Setiap *stage* mengandung daftar *job* yang dieksekusi untuk melakukan pengujian atau *deployment* otomatis.

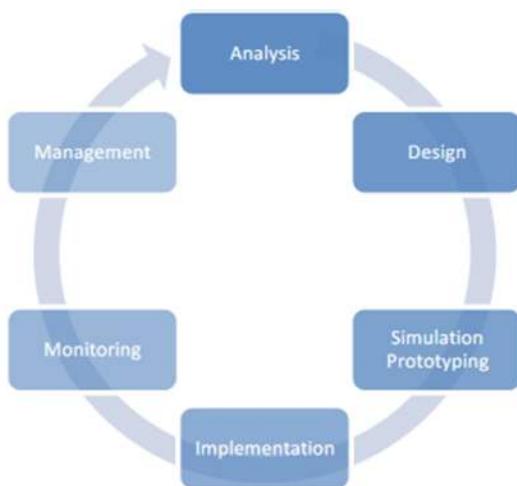
Penelitian [12] berusaha menyelesaikan permasalahan dari tim operasi sebuah perusahaan yang kewalahan dalam menerapkan perubahan sebagai dampak tim pengembangan yang menghasilkan perubahan kecil namun sering sehingga menimbulkan kemacetan. *DevOps* digunakan sebagai solusi untuk mengatasi kemacetan tersebut sehingga perangkat lunak dapat dirilis dengan cepat dan memungkinkan tim pengembangan dan operasi untuk bekerjasama. Perusahaan tersebut telah menerapkan infrastruktur *DevOps* secara *on-premise* namun belum optimal. Penelitian ini menawarkan saran untuk memaksimalkan implementasi dari *DevOps* menggunakan *GitLab* dengan memberikan pemahaman langkah-langkah yang harus dilakukan oleh perusahaan. Hasil dari penelitian tersebut menunjukkan bahwa diperlukan pekerjaan tambahan dalam mengimplementasikan *DevOps* menggunakan infrastruktur lokal yang harus dilakukan secara bertahap. Praktik *DevOps* diadopsi satu per satu meliputi kontrol versi menggunakan *GitLab Version Control* yang memerlukan pengelolaan mandiri sebagai infrastruktur pendukung dan CI/CD menggunakan *GitLab CI/CD* yang memerlukan *GitLab Runner* sebagai infrastruktur pendukung. Selain itu perusahaan juga menerapkan teknologi *container* sehingga memerlukan *Docker Registry lokal* sebagai infrastruktur pendukung. Keseluruhan infrastruktur tersebut terdapat pada data *center* lokal perusahaan termasuk *server*, media penyimpanan dan jaringan yang pengelolaannya dilakukan secara terpisah.

Penelitian terdahulu tersebut hanya berfokus pada mengotomatisasi penerapan CI/CD *pipeline* pada aplikasi *web* dan containerisasi aplikasi di infrastruktur *cloud computing* serta membandingkan *tool* CI/CD. Mendorong ketertarikan peneliti untuk menerapkan CI/CD *pipeline* untuk mengotomatisasi manajemen

konfigurasi infrastruktur jaringan komputer secara *on-premise* yang melibatkan perangkat *router* dan *switch*.

III. METODE PENELITIAN

Metodologi penelitian yang digunakan pada penelitian ini adalah *Network Development Life Cycle (NDLC)*. Terdapat enam tahapan yang terdapat pada NDLC yaitu *Analysis*, *Design*, *Simulation Prototyping*, *Implementation*, *Monitoring* dan *Management*, seperti terlihat pada gambar 1 [13].



Gambar 1. Tahapan NDLC[13]

Dari enam tahapan yang ada pada NDLC, peneliti hanya menggunakan tiga tahapan pertama yaitu *Analysis*, *Design* dan *Simulation Prototyping*.

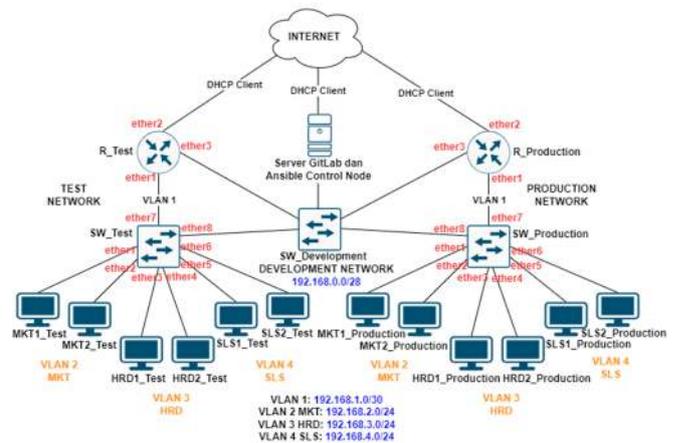
A. Tahap *Analysis*

Tahap ini terdiri dari dua bagian yaitu pengumpulan data dan analisa terhadap data yang telah dikumpulkan. Pengumpulan data meliputi artikel ilmiah yang berkaitan dengan CI/CD dan otomatisasi manajemen konfigurasi infrastruktur jaringan. Berdasarkan hasil analisa terhadap data yang telah dikumpulkan maka dapat diperoleh informasi bahwa penelitian sebelumnya hanya berfokus pada otomatisasi manajemen konfigurasi infrastruktur jaringan namun belum menerapkan CI/CD *pipeline*. Hal tersebut mendorong ketertarikan peneliti untuk meneliti tentang penerapan CI/CD *pipeline* untuk mengotomatisasi manajemen konfigurasi infrastruktur jaringan.

B. Tahap *Design*

Tahap ini terdiri dari 4 (empat) bagian yaitu rancangan jaringan ujicoba, rancangan pengalamatan IP dan rancangan sistem CI/CD

pipeline untuk mengotomatisasi manajemen konfigurasi infrastruktur jaringan komputer serta kebutuhan perangkat keras dan lunak. Rancangan jaringan ujicoba yang digunakan pada penelitian ini, seperti terlihat pada gambar 2.

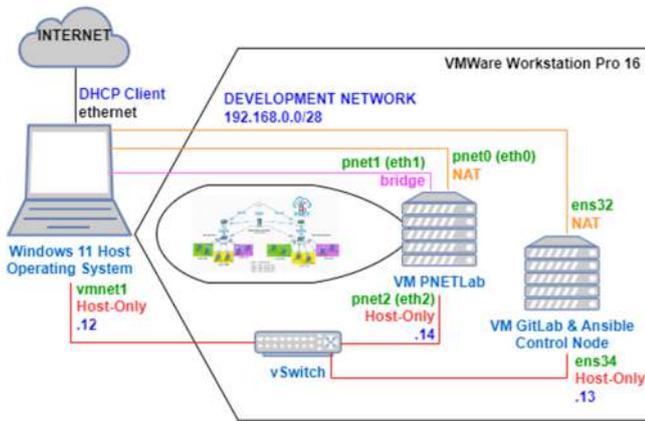


Gambar 2. Rancangan Jaringan Ujicoba

Rancangan tersebut terdiri dari tiga *network* yaitu *Development Network* sebagai jaringan pengembangan, *Test Network* sebagai jaringan pengujian fitur yang dikembangkan dan *Production Network* sebagai jaringan produksi yang digunakan untuk operasional dan menjadi lokasi penerapan fitur yang telah sukses diuji. *Development Network* terdiri dari dua perangkat yaitu satu *switch* untuk menghubungkan ke *Test Network & Production Network* dan satu *server* yang digunakan untuk menyediakan layanan *Repository* dan *Gitlab CI/CD* serta *Ansible Control Node*. Sedangkan *Test Network* terdiri dari dua perangkat yaitu satu *router* dengan nama *R_Test* untuk menjembatani koneksi ke *Internet* dan *routing* antar *Virtual Local Area Network (VLAN)* dan satu *switch* dengan nama *SW_Test* sebagai lokasi pembuatan VLAN baru yaitu dengan ID 2 bernama *Marketing (MKT)*, ID 3 bernama *Human Resource Department (HRD)* dan ID 4 bernama *Sales (SLS)*. Keanggotaan port atau *interface* pada *switch SW_Test* adalah *ether1* dan *ether2* menjadi anggota dari VLAN 2 MKT, *ether3* dan *ether4* menjadi anggota VLAN 3 HRD, *ether5* dan *ether6* menjadi anggota VLAN 4 SLS. *Interface ether7* dari *SW_Test* terhubung ke *router R_Test* dan difungsikan sebagai *trunk* sehingga dapat membawa trafik dari seluruh VLAN. Setiap VLAN terdiri dari 2 *Virtual Personal Computer (VPC)* yang menjadi bagian dari VLAN 2 sampai dengan

5. Selain itu *interface ether8* digunakan untuk koneksi ke *Development Network*. Sedangkan *Production Network* memiliki jenis (*router R_Production* dan *switch SW_Production*), jumlah perangkat dan fungsi serta *port* atau *interface* dengan interkoneksi yang identik dengan ketentuan yang diterapkan pada *Test Network*.

Rancangan jaringan ujicoba tersebut disimulasikan secara virtual menggunakan *Packet Network Emulator Tool Lab (PNETLab)* versi 4.2.10 yang diinstalasi sebagai *Virtual Machine (VM)* pada *VMWare Workstation*, seperti terlihat pada gambar 3.



Gambar 3. Simulasi Rancangan Jaringan Ujicoba

Selain itu pada *VMWare Workstation* juga dibuat VM dengan sistem operasi *Ubuntu* versi 20.04 yang bertindak sebagai *server Gitlab* dan *Ansible Control Node*. VM *PNETLab* memiliki 3 (tiga) interface yaitu *pneth0 (eth0)* dengan jenis *Network Address Translation (NAT)* untuk koneksi Internet, *pneth1 (eth1)* dengan jenis *bridge* untuk koneksi ke jaringan riil dan *pneth2 (eth2)* dengan jenis *host-only* untuk koneksi ke VM *GitLab* dan *Ansible Control Node* serta *Windows 10 Host Operating System*. Sebaliknya pada VM *GitLab* dan *Ansible Control Node* memiliki dua interface yaitu *ens32* dengan jenis *NAT* untuk koneksi Internet dan *ens34* dengan jenis *host-only* untuk koneksi ke VM *PNETLab* dan *Windows 10 Host Operating System*.

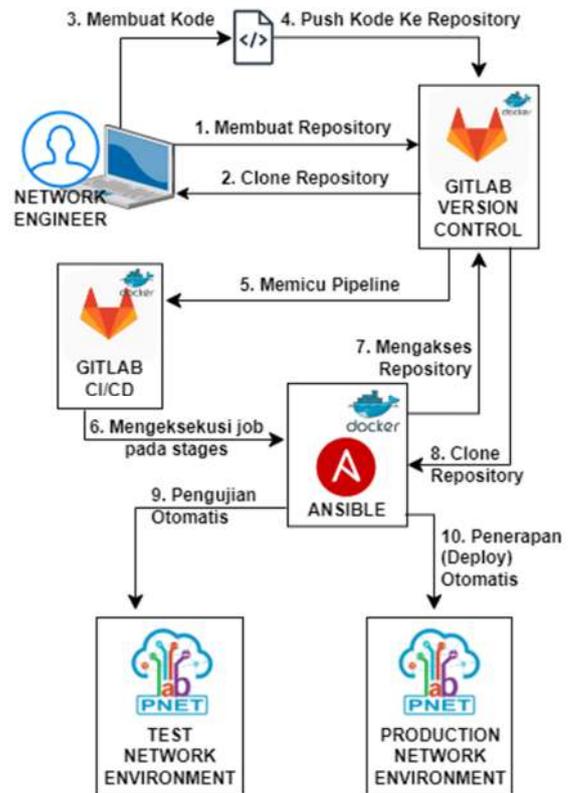
Rancangan pengalamatan IP menggunakan 4 (empat) alamat *network Class C* yaitu 192.168.1.0/24 yang di *subnetting* dua bit menjadi 192.168.1.0/30 untuk VLAN 1, 192.168.2.0/24 untuk VLAN 2 *Marketing (MKT)*, 192.168.3.0/24 untuk VLAN 3 *Human Resource Development (HRD)* dan 192.168.4.0/24 untuk VLAN 4 *Sales (SLS)*. Sedangkan setiap *PC Client* baik di *test network* maupun *production network* dialokasikan pengalamatan IP-nya secara dinamis menggunakan

Dynamic Host Configuration Protocol (DHCP). Tabel 1 memperlihatkan detail alokasi pengalamatan IP yang digunakan pada setiap perangkat di jaringan ujicoba yang disimulasikan pada *VMWare Workstation*.

Tabel 1. Pengalamatan IP Setiap Perangkat

Perangkat	Interface	Alamat IP/Subnetmask
R_Test	ether3	192.168.0.1/28
SW_Test	ether8	192.168.0.2/28
R_Production	ether3	192.168.0.3/28
SW_Production	ether8	192.168.0.4/28
Windows 11	vmnet1	192.168.0.12/28
Server GitLab	ens32	DHCP Client
	ens34	192.168.0.13/28
PNETLab	pneth4	192.168.0.14/28
Client Test	Eth0	DHCP
Network		
Client	Eth0	DHCP
Production		
Network		

Rancangan sistem *CI/CD pipeline* untuk mengotomatisasi manajemen konfigurasi infrastruktur jaringan komputer, seperti terlihat pada gambar 4.



Gambar 4. Rancangan Sistem CI/CD Pipeline

Terdapat 10 (sepuluh) alur kerja yang terdapat pada rancangan sistem *CI/CD pipeline* tersebut yaitu pertama *Network Engineer* dengan membuat

GitLab remote repository bernama "infrastruktur-jaringan" dengan *visibility level private*. Kedua melakukan *clone repository* ke lokal dengan menyalin *clone URL* melalui Secure Shell (SSH) dari *repository "infrastruktur-jaringan"*. Perintah yang dieksekusi untuk melakukan hal tersebut adalah *git clone git@localhost:user/infrastruktur-jaringan.git*. Ketiga membuat kode program untuk mengotomatisasi penerapan manajemen konfigurasi infrastruktur jaringan berbasis MikroTik meliputi *Ansible Configuration, Ansible Inventory, Ansible Variable* dan *Ansible Playbook* serta *GitLab CI/CD Pipeline* dengan nama ".gitlabci.yml". Keempat melakukan *push* kode program yang telah dibuat sebelumnya ke *GitLab remote repository* dengan mengeksekusi perintah *git push origin main*. Kelima *GitLab CI/CD pipeline* akan terpicu secara otomatis sebagai akibat *push* kode program sebelumnya ke *repository* yaitu file ".gitlab-ci.yml". Keenam *GitLab Runner* akan mengeksekusi *job* pada setiap *stages* yang ditetapkan pada file ".gitlab-ci.yml". Secara berturut-turut tahap ketujuh dan kedelapan yaitu mengakses dan melakukan *clone repository* dari *GitLab remote repository* yang memuat kode program *Ansible* untuk otomatisasi manajemen konfigurasi ke *Ansible Docker Container*. Kesembilan melakukan pengujian otomatis ke *test network environment* dimana *Ansible* akan memicu *playbook* dengan target perangkat jaringan di lingkungan *testing*. Apabila keseluruhan proses pengujian tersebut berhasil dilakukan maka akan berlanjut ke tahap kesepuluh yaitu penerapan (*deploy*) otomatis ke *production network environment* sehingga perangkat jaringan produksi menggunakan konfigurasi jaringan yang telah teruji.

Kebutuhan perangkat keras yang diperlukan untuk mendukung penyelesaian penelitian adalah satu unit laptop dengan spesifikasi *Intel Core i7*, memori 16 GB, hardisk 512 GB. Sedangkan kebutuhan perangkat lunak meliputi *VMWare Workstation Pro 16, PNETLab, Ubuntu 20.04, MikroTik Cloud Hosted Router (CHR), Docker Engine, GitLab Community Edition* dan *GitLab Runner Docker Images* serta *Ansible*. Selain itu koneksi *Internet* untuk mengunduh paket aplikasi terkait penelitian ini.

C. Tahap *Simulation Prototyping*

Tahap ini terdiri dari 6 (enam) bagian yaitu pertama melakukan *import* dan konfigurasi

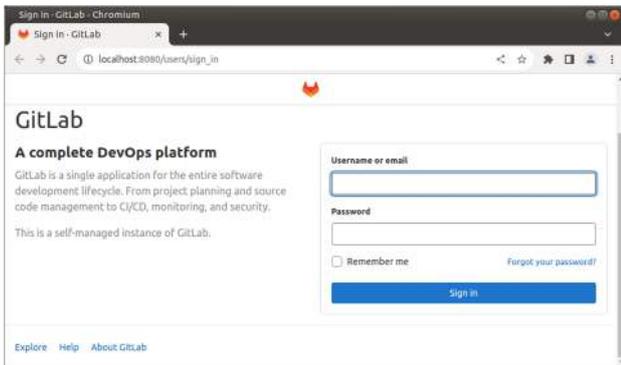
PNETLab Open Virtual Appliance (OVA) pada *VMWare Workstation Pro*. Kedua, menginstalasi dan mengkonfigurasi *server GitLab* menggunakan sistem operasi *Ubuntu 20.04* sebagai VM pada *VMWare Workstation Pro*. Ketiga, membuat lab sesuai rancangan jaringan uji coba pada *PNETLab* dan konfigurasi dasar pada setiap *node* pada lab tersebut. Keempat, membuat *repository* pada *server GitLab*. Kelima, membuat *Ansible configuration, inventory* dan *playbook* yang diintegrasikan dengan *GitLab CI/CD pipeline* pada *server GitLab* sehingga dapat mengotomatisasi penerapan manajemen konfigurasi infrastruktur jaringan baik pada *test network* maupun *production network*. Keenam, melakukan uji coba yaitu verifikasi konfigurasi dan skenario.

Verifikasi konfigurasi dilakukan pada VM *PNETLab* terkait pengalamanan IP pada *interface* dan koneksi *Internet* serta memeriksa konfigurasi dasar yang telah diterapkan pada setiap *node* yang terdapat pada lab di *PNETLab* tersebut. Selain itu juga dilakukan verifikasi konfigurasi pada *server GitLab* terkait pengalamanan IP dan koneksi ke *Internet* dan ke setiap *node* pada lab di *PNETLab* serta status berjalannya *Docker container GitLab Community Edition (CE)* dan *Runner*. Sedangkan uji coba berbasis skenario meliputi *CI/CD pipeline* untuk penambahan konfigurasi infrastruktur jaringan, verifikasi hasil *CI/CD pipeline* untuk penambahan konfigurasi infrastruktur jaringan di setiap PC dan *CI/CD pipeline* untuk penghapusan konfigurasi infrastruktur jaringan baik pada *test network* maupun *production network*. Setiap skenario tersebut dilakukan sebanyak lima kali dan diamati waktu pemrosesan baik ketika operasi penambahan maupun penghapusan konfigurasi infrastruktur jaringan melalui *CI/CD pipeline*. Selain itu juga dilakukan skenario *Ansible Playbook* yang didalamnya dengan sengaja memuat kesalahan untuk mengamati dampak yang ditimbulkan terhadap *CI/CD pipeline*.

IV. HASIL DAN PEMBAHASAN

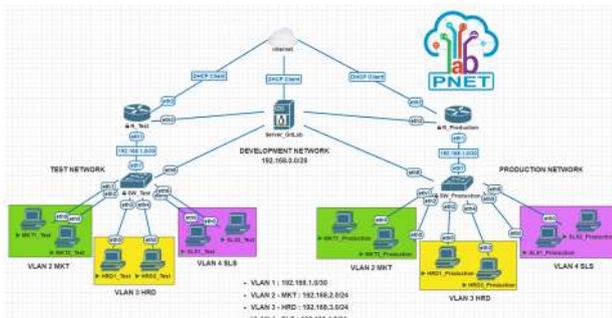
Pada bagian ini membahas tentang hasil dari instalasi dan konfigurasi, *Ansible Playbook*, uji coba serta analisa terhadap hasil uji coba. Instalasi dan konfigurasi yang dilakukan meliputi pembuatan *Virtual Machine (VM)* pada *VMWare Workstation* dengan sistem operasi *Ubuntu 20.04* yang difungsikan sebagai *Ansible Control Node* dan *server GitLab* menggunakan *container*

Docker. Hasil pengaksesan dari *server GitLab* melalui *browser* pada alamat <http://localhost:8080>, seperti terlihat pada gambar 5.



Gambar 5. Hasil Pengaksesan Server *GitLab*

Selain itu juga melakukan *import PNETLab* sebagai VM pada *VMWare Workstation* dan membuat topologi sesuai dengan rancangan jaringan ujicoba melalui antarmuka berbasis *web* dari *PNETLab* serta konfigurasi dasar dari setiap perangkat jaringan. Hasil dari pembuatan topologi dari rancangan jaringan ujicoba di *PNETLab* yang diakses melalui *browser* pada alamat <https://192.168.136.138>, seperti terlihat pada gambar 6.



Gambar 6. Hasil Pembuatan Topologi Jaringan Ujicoba pada *PNETLab*

Terdapat 5 (lima) konfigurasi dasar yang dilakukan pada setiap *router* yaitu *R_Test* dan *R_Production* meliputi mengatur *hostname* sebagai identitas dari sistem, mengubah sandi dari *user* “admin” yang merupakan akun administrator dari *MikroTik* dan mengatur pengalamatan IP pada *interface ether3* yang terhubung ke *Server GitLab*. Hasil konfigurasi dasar pada *router R_Test*, seperti terlihat pada gambar 7.

```
[admin@MikroTik] > system identity set name=R_Test
[admin@R_Test] > password
old-password:
new-password: ****
confirm-new-password: ****

[admin@R_Test] > ip address add address=192.168.0.1/28 interface=ether3
[admin@R_Test] > ip address print
Columns: ADDRESS, NETWORK, INTERFACE
# ADDRESS NETWORK INTERFACE
0 192.168.0.1/28 192.168.0.0 ether3
```

Gambar 7. Hasil Konfigurasi Dasar pada *Router R_Test*

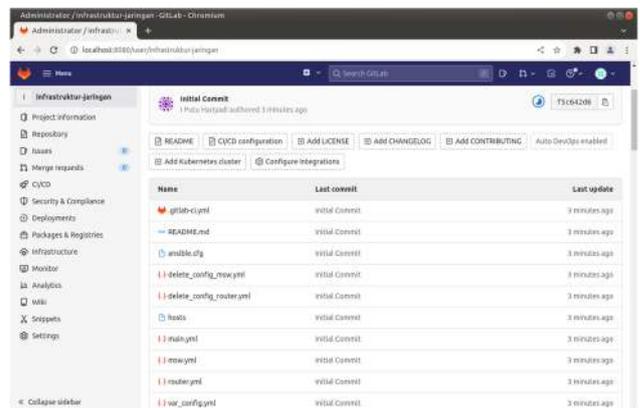
Terlihat pengalamatan IP yang digunakan pada *interface ether3* dari *router R_Test* adalah 192.168.0.1/28. Sedangkan konfigurasi dasar yang dilakukan pada setiap *switch* yaitu *SW_Test* dan *SW_Production* meliputi mengatur *hostname* sebagai identitas dari sistem, mengubah sandi dari *user* “admin” yang merupakan akun administrator dari *MikroTik* dan mengatur pengalamatan IP pada *interface ether8* yang terhubung ke *Server GitLab*. Hasil konfigurasi dasar pada *switch SW_Test*, seperti terlihat pada gambar 8. Terlihat pengalamatan IP yang digunakan pada *interface ether8* dari *switch SW_Test* adalah 192.168.0.2/28.

```
[admin@MikroTik] > system identity set name=SW_Test
[admin@SW_Test] > password
old-password:
new-password: ****
confirm-new-password: ****

[admin@SW_Test] > ip address add address=192.168.0.2/28 interface=ether8
[admin@SW_Test] > ip address print
Columns: ADDRESS, NETWORK, INTERFACE
# ADDRESS NETWORK INTERFACE
0 192.168.0.2/28 192.168.0.0 ether8
```

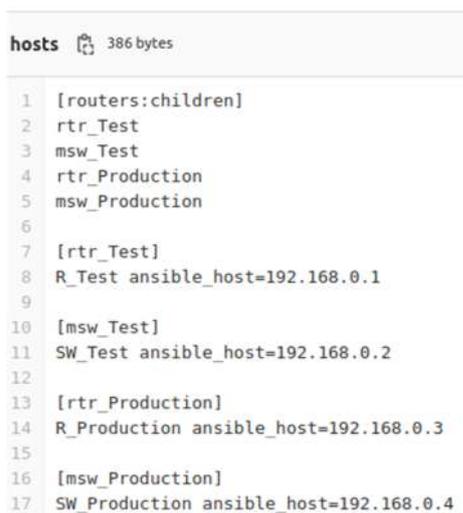
Gambar 8. Hasil Konfigurasi Dasar pada *Switch SW_Test*

Terdapat empat jenis *file* terkait *Ansible* yaitu *Configuration*, *Inventory*, *Variable* dan *Playbook* serta satu *file* terkait *GitLab CI/CD Pipeline* yang dihasilkan untuk mengotomatiskan penerapan manajemen konfigurasi infrastruktur jaringan berbasis *MikroTik*. Keseluruhan *file* tersebut disimpan pada *GitLab Repository* bernama “infrastruktur-jaringan”, seperti terlihat pada gambar 9.



Gambar 9. *GitLab Repository*

File *ansible.cfg* merupakan file yang memuat penyesuaian konfigurasi untuk mengontrol perilaku *Ansible*. Sedangkan file *inventory* bernama “*hosts*” merupakan file inisialisasi yang digunakan oleh *Ansible* untuk mendaftarkan dan mengelompokkan perangkat jaringan yang dikelola. Cuplikan isi dari file “*hosts*” tersebut, seperti terlihat pada gambar 10. 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*. Sedangkan baris 7 merupakan nama pengelompokan atau grup yaitu *rtr_Test*. Sebaliknya 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*. Demikian pula baris 10 merupakan nama grup yaitu *msw_Test*. Baris 11 merupakan *inventory_hostname* bagi *host* dengan alamat IP 192.168.0.2 yang menjadi nilai dari *variable ansible_host*. *ansible_host* merupakan *variable* yang digunakan oleh *Ansible* untuk terkoneksi dengan *remote host* alias *SW_Test*.



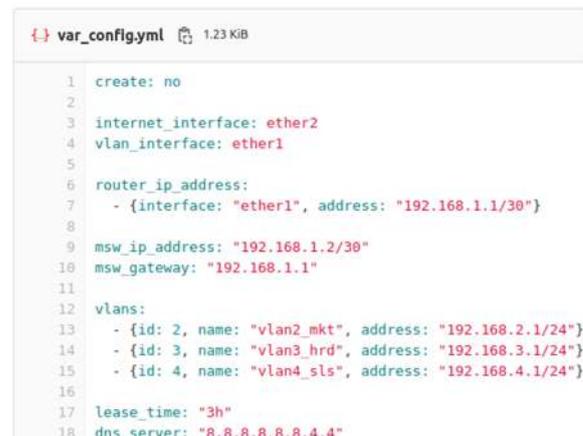
```
hosts 386 bytes
1 [routers:children]
2 rtr_Test
3 msw_Test
4 rtr_Production
5 msw_Production
6
7 [rtr_Test]
8 R_Test ansible_host=192.168.0.1
9
10 [msw_Test]
11 SW_Test ansible_host=192.168.0.2
12
13 [rtr_Production]
14 R_Production ansible_host=192.168.0.3
15
16 [msw_Production]
17 SW_Production ansible_host=192.168.0.4
```

Gambar 10. File *Ansible Inventory hosts*

Baris 13 merupakan nama grup yaitu *R_Production*. Baris 14 merupakan *inventory_hostname* bagi *host* dengan alamat IP 192.168.0.3 yang menjadi nilai dari *variable ansible_host*. *ansible_host* merupakan *variable* yang digunakan oleh *Ansible* untuk terkoneksi dengan *remote host* alias *R_Production*. Sedangkan baris 16 merupakan nama grup yaitu *msw_Production*. Terakhir, baris 17 merupakan

inventory_hostname bagi *host* dengan alamat IP 192.168.0.4 yang menjadi nilai dari *variable ansible_host*. *ansible_host* merupakan *variable* yang digunakan oleh *Ansible* untuk terkoneksi dengan *remote host* alias *SW_Production*.

File *variable* bernama “*var_config.yml*” memuat deklarasi *variable* yang digunakan untuk penerapan konfigurasi di *router* dan *switch* secara dinamis baik pada *test network* maupun *production network*. Cuplikan isi dari file *variable* “*var_config.yml*”, seperti terlihat pada gambar 11.



```
var_config.yml 1.23 KIB
1 create: no
2
3 internet_interface: ether2
4 vlan_interface: ether1
5
6 router_ip_address:
7 - {interface: "ether1", address: "192.168.1.1/30"}
8
9 msw_ip_address: "192.168.1.2/30"
10 msw_gateway: "192.168.1.1"
11
12 vlans:
13 - {id: 2, name: "vlan2_mkt", address: "192.168.2.1/24"}
14 - {id: 3, name: "vlan3_hrd", address: "192.168.3.1/24"}
15 - {id: 4, name: "vlan4_sls", address: "192.168.4.1/24"}
16
17 lease_time: "3h"
18 dns_server: "8.8.8.8,8.8.4.4"
```

Gambar 11. File *Variable Ansible var_config.yml*

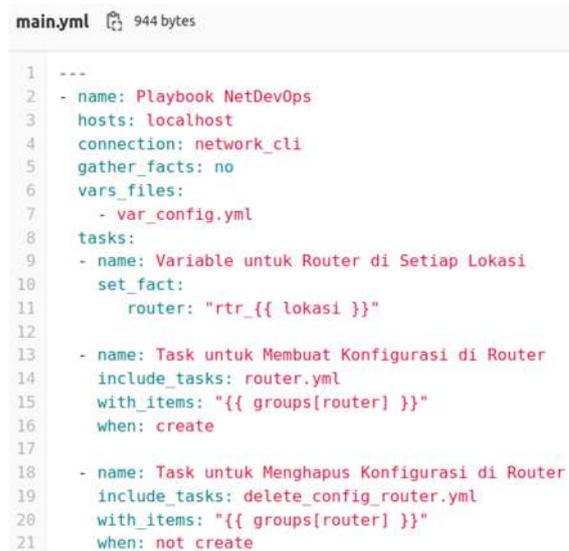
Pada baris 1 memuat deklarasi *variable* bernama “*create*” dengan nilai “*no*” untuk operasi penghapusan konfigurasi atau “*yes*” untuk operasi penambahan konfigurasi pada perangkat yang dikelola. Sedangkan baris 3 memuat deklarasi *variable internet_interface* dengan nilai “*ether2*” yang digunakan untuk menentukan *interface* pada perangkat *router R_Test* dan *R_Production* yang terhubung ke *Internet*. Selanjutnya baris 4 memuat deklarasi *variable vlan_interface* dengan nilai “*ether1*” yang merupakan *interface* pada perangkat *router R_Test* dan *R_Production* yang terhubung ke *switch*. Baris 9 dan 10 digunakan untuk mengatur alamat IP dari *interface bridge* dengan nilai 192.168.1.2/30 dan alamat *default gateway* bagi *node switch* dengan nilai 192.168.1.1. Baris 12 sampai dengan 15 memuat deklarasi *variable* bernama “*vlans*” dengan struktur data *list of dictionaries* yang didalamnya memuat 3 (tiga) data untuk pembuatan VLAN yaitu masing-masing dengan id 2 dengan alamat IP 192.168.2.1/24, id 3 dengan alamat IP 192.168.3.1/24, dan id 4 dengan alamat IP 192.168.4.1/24. Baris 17 deklarasi *variable lease_time* dengan nilai 3h untuk mengatur masa sewa dari alamat IP yang

didistribusikan dari *server* DHCP ke DHCP *Client* selama 3 jam. Terakhir pada baris 18 memuat deklarasi *variable dns_server* dengan nilai "8.8.8.8,8.8.4.4" yang digunakan ketika pembuatan server DHCP.

Pada *file variable* tersebut juga memuat deklarasi *variable* bernama "*dhcp_pools*" dengan struktur data *list of dictionaries* yang didalamnya memuat 3 (tiga) data untuk pembuatan DHCP di *node R_Test* dan *R_Production*. *Variable* bernama "*bridge_ports*" dengan struktur data *list of dictionaries* yang didalamnya memuat 7 (tujuh) data untuk pengaturan *Bridge Port* di *node switch* baik *SW_Test* maupun *SW_Production*. Data pertama dan kedua yaitu *interface ether1* dan *ether2* sebagai *bridge port* dari *bridge VLAN* pada *switch* tersebut yang dapat berupa *BR_Test* atau *BR_Production* dan mengatur *Port VLAN ID* (PVID) dari *interface* tersebut dengan nilai 2. Selanjutnya data ketiga dan keempat yaitu *interface ether3* dan *ether4* sebagai *bridge port* dari *bridge VLAN* pada *switch* tersebut dan mengatur PVID dari *interface* tersebut dengan nilai 3. Sedangkan data kelima dan keenam yaitu *interface ether5* dan *ether6* sebagai *bridge port* dari *bridge VLAN* pada *switch* tersebut dan mengatur PVID dari *interface* tersebut dengan nilai 4. Terakhir *interface ether7* sebagai *bridge port* dari *bridge VLAN* pada *switch* tersebut dan mengatur PVID dari *interface* tersebut dengan nilai 1.

Variable "vlan_tagging" dengan struktur data *list of dictionaries* yang didalamnya memuat 3 (tiga) data untuk mengatur *VLAN tagging* pada *interface bridge VLAN* dari *node switch SW_Test* bernama *BR_Test* dan *SW_Production* bernama *BR_Production*. Pertama, pada *interface bridge* tersebut dengan *tagged port ether7*, *untagged port ether1* dan *ether2*, *vlan-ids* 2 dan komentar bernilai isi dari *variable* bernama "*lokasi*" yaitu dapat berupa *test* atau *production*. Kedua, pada *interface bridge* tersebut dengan *tagged port ether7*, *untagged port ether3* dan *ether4*, *vlan-ids* 3 dan komentar bernilai isi dari *variable* bernama "*lokasi*". Ketiga, pada *interface bridge* tersebut dengan *tagged port ether7*, *untagged port ether5* dan *ether6*, *vlan-ids* 4 dan komentar bernilai isi dari *variable* bernama "*lokasi*".

Cuplikan isi dari *file Ansible playbook* dengan nama "*main.yml*", seperti terlihat pada gambar 12.



```

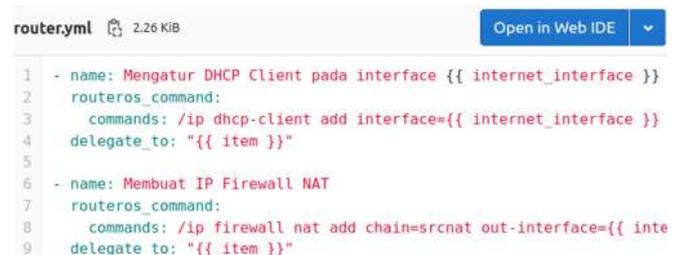
1 ---
2 - name: Playbook NetDevOps
3   hosts: localhost
4   connection: network_cli
5   gather_facts: no
6   vars_files:
7     - var_config.yml
8   tasks:
9     - name: Variable untuk Router di Setiap Lokasi
10      set_fact:
11        router: "rtr_{{ lokasi }}"
12
13    - name: Task untuk Membuat Konfigurasi di Router
14      include_tasks: router.yml
15      with_items: "{{ groups[router] }}"
16      when: create
17
18    - name: Task untuk Menghapus Konfigurasi di Router
19      include_tasks: delete_config_router.yml
20      with_items: "{{ groups[router] }}"
21      when: not create

```

Gambar 12. File Playbook *main.yml*

File tersebut merupakan *playbook* utama yang digunakan untuk manajemen penerapan otomatisasi konfigurasi infrastruktur baik pada perangkat jaringan di lingkungan *test network* maupun *production network*. *File playbook* tersebut memuat instruksi untuk menyisipkan *file variable "var_config.yml"* dan *file task* untuk menambahkan konfigurasi ke *router* bernama "*router.yml*" dan menghapus konfigurasi dari *router* bernama "*delete_config_router.yml*". Selain itu juga menyisipkan *file task* untuk menambahkan konfigurasi ke *switch* bernama "*msw.yml*" dan menghapus konfigurasi dari *switch* bernama "*delete_config_msw.yml*".

Cuplikan isi dari *file "router.yml"*, seperti terlihat pada gambar 13.



```

1 - name: Mengatur DHCP Client pada interface {{ internet_interface }}
2   routeros_command:
3     commands: /ip dhcp-client add interface={{ internet_interface }}
4     delegate_to: "{{ item }}"
5
6 - name: Membuat IP Firewall NAT
7   routeros_command:
8     commands: /ip firewall nat add chain=srcnat out-interface={{ inte
9     delegate_to: "{{ item }}"

```

Gambar 13. File Task *router.yml*

File task tersebut memuat 7 (tujuh) *task* meliputi mengatur DHCP *Client* pada *interface*, membuat IP *Firewall* NAT, membuat *interface VLAN*, mengatur pengalamatan IP pada setiap VLAN, membuat IP *Pool* dan IP DHCP-*Server Network* serta IP DHCP-*Server*.

Konfigurasi *GitLab CI/CD pipeline* dilakukan dengan membuat file *.gitlab-ci.yml* pada *root* dari *repository* dengan isi seperti terlihat pada gambar 14. Baris 1 yaitu `---` (3 *hyphen*) merupakan awal dari dokumen *YAML*.

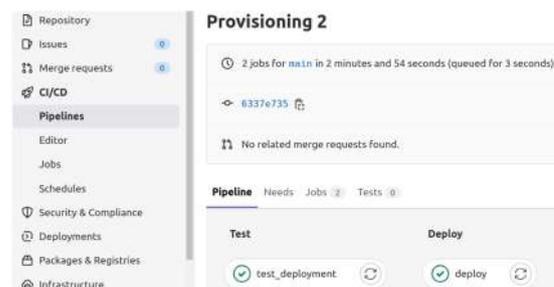
```
.gitlab-ci.yml 463 bytes
1 ---
2 before_script:
3   - export ANSIBLE_HOST_KEY_CHECKING=False
4
5 stages:
6   - test
7   - deploy
8
9 variables:
10  GIT_STRATEGY: clone
11
12 test_deployment:
13   stage: test
14   image: "chusiang/ansible:latest"
15   tags:
16     - test
17   script:
18     - ansible-playbook -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
30
```

Gambar 14. File Konfigurasi *.gitlab-ci.yml*

Baris 2 sampai dengan 3 memuat deklarasi *keyword before_script* yang digunakan untuk mengganti sekumpulan perintah yang dieksekusi sebelum *job* yaitu dalam hal ini *export variable ANSIBLE_HOST_KEY_CHECKING* 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*.

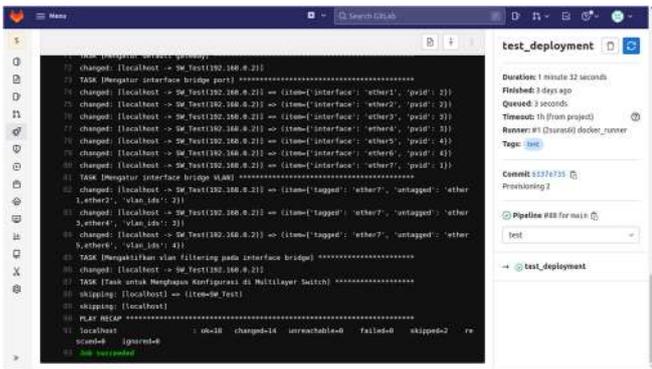
Cuplikan salah satu *GitLab CI/CD Pipeline* yang telah berjalan, seperti terlihat pada gambar 15.



Gambar 15. *GitLab CI/CD Pipelines*

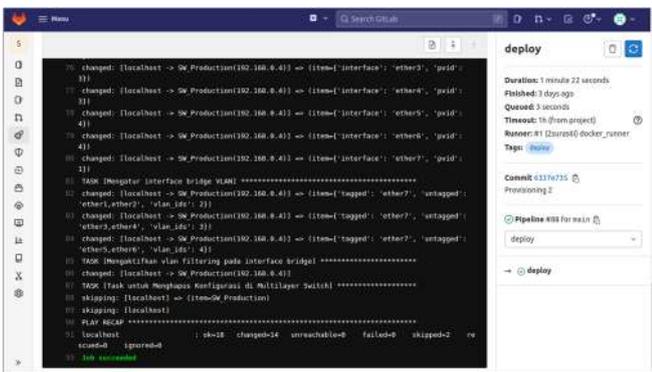
Terlihat terdapat 2 (dua) *stage* yaitu *Test* dan *Deploy*. Pada *stage Test* terdapat satu *job* yang berjalan yaitu *test_deployment* dan telah sukses dieksekusi dimana ditandai dengan centang berwarna hijau. Sedangkan pada *stage Deploy* juga terdapat satu *job* yang berjalan yaitu *deploy* dan juga telah sukses dieksekusi dimana ditandai dengan centang berwarna hijau.

Informasi detail terkait eksekusi dari *job test_deployment*, seperti terlihat pada gambar 16.



Gambar 16. Hasil Eksekusi Job *test_deployment*

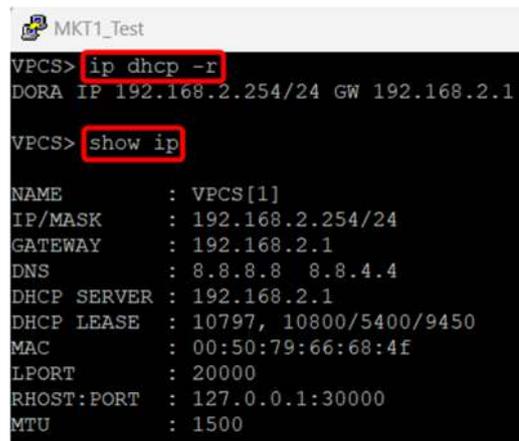
Pada bagian tengah dari halaman yang menampilkan informasi tersebut yaitu dengan latar belakang berwarna hitam menunjukkan hasil eksekusi setiap *task* dari *Ansible playbook* di lingkungan *test network* dari lab pada PNETLab. Selain itu juga pada bagian sebelah kiri bawah terdapat pesan *Job succeeded* 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 32 detik. Sedangkan informasi detail terkait eksekusi dari *job deploy*, seperti terlihat pada gambar 17.



Gambar 17. Hasil Eksekusi Job *deploy*

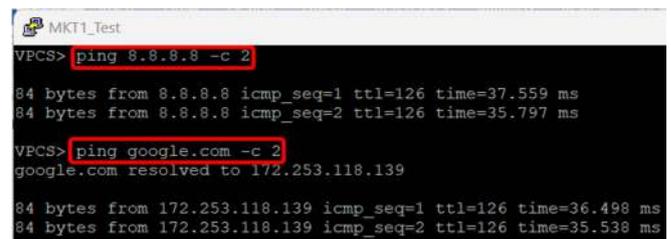
Pada bagian tengah dari halaman yang menampilkan informasi tersebut yaitu dengan latar belakang berwarna hitam menunjukkan hasil eksekusi setiap *task* dari *Ansible playbook* di lingkungan *production network* dari lab pada PNETLab. Selain itu juga pada bagian sebelah kiri bawah terdapat pesan *Job 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.

CI/CD pipeline yang telah berhasil dieksekusi tersebut memerlukan verifikasi terkait hasil konfigurasi pada setiap PC baik pada *test network* maupun *production network*. Verifikasi konfigurasi dilakukan untuk memastikan PC tersebut telah menjadi anggota dari VLAN tertentu dan memperoleh pengalamatan IP secara dinamis dari *server DHCP*. Gambar 18 memperlihatkan salah satu contoh hasil verifikasi yang dilakukan pada PC *MKT1_Test* yang terhubung ke *interface ether1* dari *switch SW_Test*. Terlihat eksekusi perintah *ip dhcp -r* yang digunakan untuk memperoleh pengalamatan IP secara dinamis dari *server DHCP*. Sedangkan perintah *show ip* digunakan untuk menampilkan hasil pengalamatan IP yang diperoleh PC *MKT1_Test* dari server DHCP yaitu 192.168.2.254/24 dengan default gateway 192.168.2.1 serta alamat IP server DNS 8.8.8.8 dan 8.8.4.4. Alamat IP yang diperoleh tersebut merupakan salah satu alamat pada network 192.168.2.0/24 yang dialokasikan untuk VLAN 2 MKT sehingga membuktikan bahwa PC tersebut telah menjadi anggota dari VLAN 2.



Gambar 18. Verifikasi Konfigurasi pada PC *MKT1_Test*

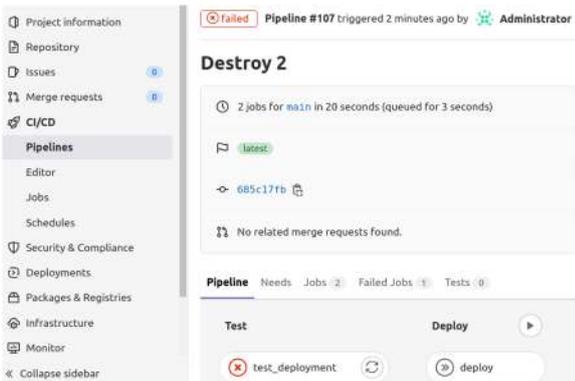
Selain itu juga dilakukan verifikasi koneksi ke *Internet* menggunakan perintah *ping 8.8.8.8 -c 2* dan *ping google.com -c 2*, seperti terlihat pada gambar 19.



Gambar 19. Hasil Verifikasi Koneksi *Internet* dari PC *MKT1_Test*

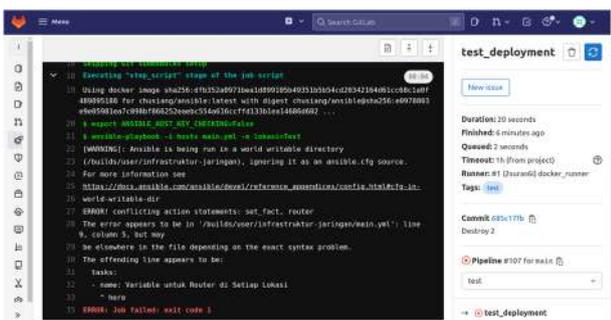
Terlihat verifikasi koneksi *Internet* ke alamat IP 8.8.8.8 dan *google.com* sukses dilakukan.

GitLab CI/CD Pipeline akan gagal dieksekusi apabila terdapat kesalahan pada instruksi di dalam file *Ansible variable, playbook* atau *task* yang dibuat, seperti terlihat pada gambar 20.



Gambar 20. Kegagalan Eksekusi CI/CD Pipeline

Terlihat eksekusi *pipeline* pada *stage Test* terhenti karena terdapat kesalahan pada *job test deployment*. Hal ini ditandai dengan simbol silang berwarna merah sehingga *job* yang terdapat pada *stage* berikutnya yaitu *Deploy* tidak akan dieksekusi (*skipped*). *Job* pada *stage Deploy* akan berjalan hanya apabila *job* pada *stage Test* sukses dieksekusi. Detail penyebab kegagalan eksekusi dari *job* tersebut, seperti terlihat pada gambar 21.



Gambar 21. *Job test deployment* Yang Gagal Dieksekusi

Pada bagian tengah dari halaman informasi eksekusi *job test deployment* tersebut yaitu dengan latar belakang berwarna hitam menunjukkan kesalahan terjadi pada baris 9 kolom 5 terkait pernyataan *set_fact* “*router*”. Melalui *troubleshooting* maka diketahui penyebab kesalahan tersebut adalah kurangnya indentasi pada *variable* “*router*” yang dideklarasikan

menggunakan *set_fact* di bagian *task* bernama “*Variable untuk Router di Setiap Lokasi*” dari file *playbook* “*main.yml*”.

Percobaan CI/CD untuk mengotomatisasi manajemen konfigurasi infrastruktur jaringan meliputi penambahan dan penghapusan konfigurasi yang dilakukan sebanyak 5 (lima) baik pada *test network* maupun *production network*. Tabel 2 memperlihatkan hasil ujicoba CI/CD untuk mengotomatisasi penambahan konfigurasi infrastruktur jaringan.

Tabel 2. CI/CD Otomatisasi Penambahan Konfigurasi Infrastruktur Jaringan

Percobaan	Test Network (detik)	Production Network (detik)
1	81	79
2	92	82
3	81	78
4	79	81
5	80	81

Berdasarkan tabel 2 tersebut maka diperoleh informasi bahwa proses penambahan konfigurasi infrastruktur jaringan secara otomatisasi melalui CI/CD pada *test network* membutuhkan waktu terlama 92 detik ketika percobaan kedua dan tercepat 79 detik ketika percobaan keempat. Sedangkan pada *production network* membutuhkan waktu terlama 82 detik ketika percobaan kedua dan tercepat 79 detik ketika percobaan pertama. Waktu rata-rata proses penambahan konfigurasi infrastruktur jaringan pada *test network* adalah 82,6 detik. Sebaliknya waktu rata-rata proses penambahan pada *production network* adalah 80,2 detik. Selisih waktu penambahan konfigurasi antara *test network* dengan *production network* adalah 2,4 detik.

Hasil ujicoba CI/CD untuk mengotomatisasi penghapusan konfigurasi infrastruktur jaringan, seperti terlihat pada tabel 3.

Tabel 3. CI/CD Otomatisasi Penghapusan Konfigurasi Infrastruktur Jaringan

Percobaan	Test Network (detik)	Production Network (detik)
1	55	55
2	59	57
3	57	53
4	56	54

Berdasarkan tabel 3 tersebut maka diperoleh informasi bahwa proses penghapusan konfigurasi infrastruktur jaringan secara otomatisasi melalui CI/CD pada *test network* membutuhkan waktu terlama 59 detik ketika percobaan kedua dan tercepat 55 detik ketika percobaan pertama. Sedangkan pada *production network* membutuhkan waktu terlama 57 detik ketika percobaan kedua dan tercepat 53 detik ketika percobaan ketiga dan kelima. Waktu rata-rata proses penghapusan konfigurasi infrastruktur jaringan pada *test network* adalah 56,6 detik. Sebaliknya waktu rata-rata proses penghapusan pada *production network* adalah 54,4 detik. Selisih waktu penghapusan konfigurasi antara *test network* dengan *production network* adalah 2,2 detik.

Hasil verifikasi penambahan konfigurasi infrastruktur jaringan pada setiap PC yang terdapat baik di *test network* maupun *production network*, seperti terlihat pada tabel 4.

Tabel 4. Verifikasi Penambahan Konfigurasi Infrastruktur Jaringan Pada Setiap PC

Komponen Verifikasi	Test Network	Production Network
Pengalamatan IP secara dinamis dari server DHCP.	√	√
Komunikasi ke PC lain pada VLAN yang sama.	√	√
Komunikasi ke PC lain pada VLAN berbeda.	√	√
Koneksi Internet.	√	√

Terlihat terdapat empat komponen verifikasi yang dilakukan pada setiap PC meliputi pengalamatan IP secara dinamis dari *server* DHCP, komunikasi ke PC lain pada VLAN yang sama dan berbeda serta koneksi *Internet*. Hasil dari setiap komponen yang diverifikasi ditandai dengan simbol centang (√) yang menandakan bahwa setiap PC telah berhasil atau sukses. Sebaliknya jika gagal akan ditandai dengan simbol silang (×). Berdasarkan tabel 4 diperoleh informasi bahwa setiap PC baik di *test network* maupun *production network* telah berhasil memenuhi setiap komponen verifikasi konfigurasi yang diujikan.

V. KESIMPULAN

Adapun kesimpulan yang dapat diambil berdasarkan hasil ujicoba yang telah dilakukan adalah *Ansible Playbook* yang telah dibuat dan diintegrasikan dengan *GitLab CI/CD pipeline* dapat digunakan untuk mengotomatisasi penerapan manajemen konfigurasi infrastruktur jaringan berbasis *MikroTik* yang disimulasikan pada PNETLab. CI/CD dapat mempercepat penerapan perubahan konfigurasi jaringan melalui proses yang diotomatisasi dan meminimalkan terjadinya kesalahan. Selain itu penerapan konfigurasi ke *production network* dilakukan hanya jika pengujian terhadap konfigurasi di *test network* berhasil dilakukan. Mekanisme CI/CD tersebut dapat meningkatkan performansi dan reabilitas infrastuktur jaringan komputer.

DAFTAR PUSTAKA

- [1] A. Datta, A. T. M. A. Imran, and C. Biswas, "Network Automation: Enhancing Operational Efficiency Across the Network Environment," *ICRRD Quality Index Research Journal*, vol. 4, no. 1, 2023, doi: 10.53272/icrrd.v4i1.1.
- [2] N. M. A. Yalestia Chandrawaty and I. P. Hariyadi, "Implementasi Ansible Playbook Untuk Mengotomatisasi Manajemen Konfigurasi VLAN Berbasis VTP Dan Layanan DHCP," *Jurnal Bumigora Information Technology (BITE)*, vol. 3, no. 2, pp. 107–122, Dec. 2021, doi: 10.30812/bite.v3i2.1577.
- [3] M. F. Islami, P. Musa, and M. Lamsani, "Implementation of Network Automation Using Ansible to Configure Routing Protocol in Cisco and Mikrotik Router with Raspberry PI," *Jurnal Ilmiah Komputasi*, vol. 19, no. 2, pp. 127–134, 2020, doi: 10.32409/jikstik.19.2.2766.
- [4] K. Marzuki *et al.*, "OTOMASISASI MANAJEMEN VLAN INTERVLAN DAN DHCP SERVER MENGGUNAKAN ANSIBLE," *Jurnal Informatika & Rekayasa Elektronika*, vol. 4, no. 2, pp. 171–180, 2021, Accessed: Aug. 08, 2023. [Online]. Available: <http://e-journal.stmiklombok.ac.id/index.php/jire/article/view/461>
- [5] M. Maisyaroh, K. Ishak, S. Faizah, and I. Fadhillah, "Otomatisasi Jaringan Menggunakan Script Python Untuk Penyediaan Konfigurasi Internet Dan Manajemen Mikrotik," *Bina Insani ICT Journal*, vol. 8, no. 1, pp. 53–62, 2021.
- [6] L. G. Mauboy and T. Wellem, "Studi Perbandingan Library Untuk Implementasi Network Automation Menggunakan Paramiko Dan Netmiko Pada Router Mikrotik," *JURIKOM (Jurnal Riset Komputer)*, vol. 9, no. 4, p. 790, Aug. 2022, doi: 10.30865/jurikom.v9i4.4420.
- [7] I. Syah, A. H. Muhammad, and E. Gunawan, "Simulasi Network Automation Menggunakan

- Ansible Di GNS3 (Studi Kasus Smile Project),” *Jurnal Teknik Informatika (J-Tifa)*, vol. 3, no. 2, pp. 1–8, Sep. 2020, doi: 10.52046/j-tifa.v3i2.1065.
- [8] M. Rifki Afandi, P. Hatta, A. Efendi, K. Kunci-Otomatisasi Jaringan, L. Komputer, and P. Jaringan, “Otomatisasi Perangkat Jaringan Komputer Menggunakan Ansible Pada Laboratorium Komputer,” *SMARTICS Journal*, vol. 6, no. 2, pp. 48–53, 2020, doi: 10.21067/smartics.v6i2.4599.
- [9] S. Charanjot, S. G. Nikita, K. Manjot, and K. Bhavleen, “Comparison of Different CI/CD Tools Integrated with Cloud Platform,” in *2019 9th International Conference on Cloud Computing, Data Science & Engineering*, 2019. doi: 10.1109/CONFLUENCE.2019.8776985.
- [10] A. Cepuc, R. Botez, O. Craciun, I. A. Ivanciu, and V. Dobrota, “Implementation of a continuous integration and deployment pipeline for containerized applications in amazon web services using jenkins, ansible and kubernetes,” in *Proceedings - RoEduNet IEEE International Conference*, IEEE Computer Society, Dec. 2020. doi: 10.1109/RoEduNet51892.2020.9324857.
- [11] A. Alanda, H. A. Mooduto, and R. Hadelina, “Continuous Integration and Continuous Deployment (CI/CD) for Web Applications on Cloud Infrastructures,” *JITCE (Journal of Information Technology and Computer Engineering)*, vol. 6, no. 02, pp. 50–55, Sep. 2022, doi: 10.25077/jitce.6.02.50-55.2022.
- [12] M. Alvin and R. Fathoni Aji, “DevOps Implementation with Enterprise On-Premise Infrastructure,” *JURNAL MEDIA INFORMATIKA BUDIDARMA*, vol. 7, no. 1, Jan. 2023, doi: 10.30865/mib.v7i1.5500.
- [13] Y. Ariyanto, B. Harijanto, V. A. H. Firdaus, and S. N. Arief, “Performance analysis of Proxmox VE firewall for network security in cloud computing server implementation,” in *IOP Conference Series: Materials Science and Engineering*, Institute of Physics Publishing, Jan. 2020. doi: 10.1088/1757-899X/732/1/012081.