Perbandingan Waktu Eksekusi Otomasi Manajemen Konfigurasi Sistem Operasi GNU/Linux antara Ansible dengan NixOS

M. Rizqi R [1], Agus Prihanto [2]

[1,3] Prodi Teknik Informatika, Fakultas Teknik, Universitas Negeri Surabaya
[1]rizqirazkafi56@gmail.com
[2]agusprihanto@unesa.ac.id

***Abstrak*— Pentingnya melakukan manajemen konfigurasi dengan tujuan untuk menghindari penulisan manual konfigurasi sistem operasi secara manual setiap kali menyiapkan sistem operasi. Manajemen konfigurasi juga digunakan untuk mencapai konsistensi dalam setiap kali penerapan sehingga hasil akhir yang diinginkan akan sama setiap kali dilakukan. Terdapat beberapa *tool* untuk melakukan manajemen konfigurasi sistem operasi, terutama untuk sistem operasi berbasis GNU/Linux. Ansible dan NixOS menjadi dua dari banyak pilihan untuk melakukan manajemen sistem operasi. Keduanya memiliki tujuan memudahkan proses manajemen konfigurasi dan memastikan hasil akhir yang diinginkan akan sama setiap kali eksekusi. Sebagai tool manajemen konfigurasi, Ansible memiliki waktu eksekusi lebih cepat dari NixOS dengan nixos-rebuild. Namun dari segi deklaratif, NixOS terbukti lebih deklaratif karena semua yang ada dalam konfigurasi di manifestasi dalam sistem. Berbeda dengan Ansible yang hanya mengerjakan apa yang ada dalam Ansible Playbook dan tidak menjadi manifestasi dari sistem. Pada sisi lain, Ansible terbukti lebih cepat dan efektif apabila mengkonfigurasi sistem lebih dari satu karena Ansible memiliki kemampuan paralelisasi sedangkan NixOS dengan nixos-rebuild belum mampu melakukan paralelisasi.**

***Kata Kunci*— Ansible, NixOS, *declarative, imperative*, manajemen konfigurasi.**

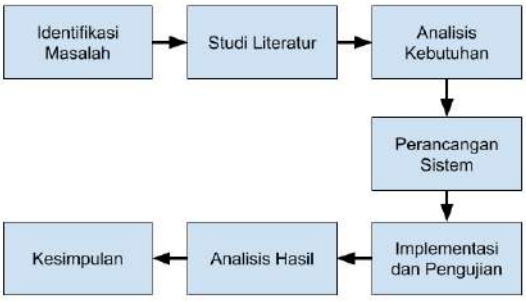
I. PENDAHULUAN

Pentingnya melakukan manajemen konfigurasi dengan tujuan untuk menghindari penulisan manual konfigurasi sistem operasi secara manual setiap kali menyiapkan sistem operasi. Manajemen konfigurasi juga digunakan untuk mencapai konsistensi dalam setiap kali penerapan sehingga hasil akhir yang diinginkan akan sama setiap kali dilakukan. Terdapat beberapa *tool* untuk melakukan manajemen konfigurasi sistem operasi, terutama untuk sistem operasi berbasis GNU/Linux. Ansible dan NixOS menjadi dua dari banyak pilihan untuk melakukan manajemen sistem operasi. Keduanya memiliki tujuan memudahkan proses manajemen konfigurasi dan memastikan hasil akhir yang diinginkan akan sama setiap kali eksekusi. Ansible adalah perangkat lunak otomatisasi TI baris perintah yang ditulis dalam bahasa Python. Aplikasi ini dapat mengonfigurasi sistem, menerapkan perangkat lunak, dan mengatur alur kerja tingkat lanjut untuk mendukung penerapan aplikasi, pembaruan sistem, dan banyak lagi (Ansible, 2016). Ansible memungkinkan kita mendeklarasikan konfigurasi sistem operasi kita dalam sebuah Ansible Playbook. Ansible Playbook akan dijalankan pada sebuah sistem yang telah memiliki sistem operasi. Konfigurasi Ansible Playbook ditulis menggunakan format yml yang merupakan format khusus untuk konfigurasi baik sistem maupun aplikasi. Ansible akan menjalankan setiap perintah pada sistem operasi yang telah di install secara otomatis satu per satu. Ansible memungkinkan kita melakukan setup banyak sistem sekaligus dengan konfigurasi yang telah ada. Diharapkan dari Ansible adalah sistem-sistem yang terdaftar memiliki hasil akhir yang sama. NixOS adalah sistem operasi berbasis GNU/Linux 1 yang dibangun dengan Nix *build system* (NixOS, 2023) . NixOS menggunakan file dalam format ".nix" yang disebut sebagai NixOS *module* untuk mendeklarasikan sebuah sistem. Dalam file tersebut terdapat seluruh konfigurasi sistem mulai dari *bootloader, packages, users, system services*. Apa yang tertulis dalam *module* tersebut adalah manifestasi dari sistem yang dideklarasikan menggunakan bahasa nix yang merupakan bahasa pemrograman fungsional. Ini menghasilkan konfigurasi sistem operasi yang *reproducible* sehingga dapat digunakan berkali-kali pada waktu yang berbeda dan menghasilkan manifestasi yang tetap. Banyak kelebihan dan beberapa kekurangan yang dimiliki oleh metode deklaratif dari NixOS dan metode campuran (imperatif dan deklaratif) dari Ansible. Berdasarkan landasan tersebut, maka penulis ingin meneliti dan membandingkan dari segi performa waktu eksekusi yang dibutuhkan oleh masing-masing *tool* manajemen konfigurasi dengan hasil akhir konfigurasi yang serupa.

II. METODOLOGI PENELITIAN

A. Metodologi Penelitian

Metode yang diterapkan pada penelitian ini adalah metode experimental design. Penerapan metode bertujuan untuk menganalisa perbandingan manajemen konfigurasi sistem operasi GNU/Linux antara Ansible dengan NixOS.



Gbr. 1 Metodologi Penelitian

B. Studi Literatur

Peneliti mencari literatur yang relevan untuk menjadi referensi pendukung dalam penelitian. Literatur yang digunakan dalam penelitian ini berhubungan dengan penerapan manajemen konfigurasi yang didapat dari berbagai macam sumber seperti buku, artikel, jurnal nasional maupun internasional

C. Analisis Kebutuhan

Analisis kebutuhan merupakan analisis yang dibutuhkan untuk menentukan detail kebutuhan pada penelitian analisis perbandingan manajemen konfigurasi sistem operasi GNU/Linux antara Ansible dengan NixOS. Penelitian ini mengotomasi konfigurasi sistem operasi GNU/Linux dan membuat sistem menjadi terdeklarasi. Sehingga dibutuhkan perangkat-perangkat yang dapat mendukung agar penelitian ini berjalan sesuai tujuan. Berikut merupakan kebutuhan yang dibagi menjadi beberapa bagian, yaitu:

1) *Kebutuhan perangkat keras (hardware)*:

Perangkat keras yang digunakan dalam penelitian ini yaitu Laptop untuk mengakses VM melalui SSH adalah sebagai berikut:

Processor : Intel Core i5 11400H
RAM : 24 GB
SSD : 1 TB
Sistem Operasi : NixOS

Untuk spesifikasi server yang akan digunakan sebagai tempat *virtual machine* adalah sebagai berikut:

Processor : Intel Xeon E3-1220 v5
RAM : 32 GB
Storage : SSD 512GB + HDD 2TB
Sistem Operasi : Proxmox VE 7.4-10

2) *Kebutuhan perangkat lunak (software)*:

Perangkat lunak yang digunakan untuk penelitian ini baik untuk sistem operasi, *tool* manajemen konfigurasi, dan perangkat lunak pengukuran kecepatan waktu adalah sebagai berikut:

- Ansible
- NixOS
- Ubuntu Server 24.04 LTS
- Nix Flakes
- Home Manager
- Time

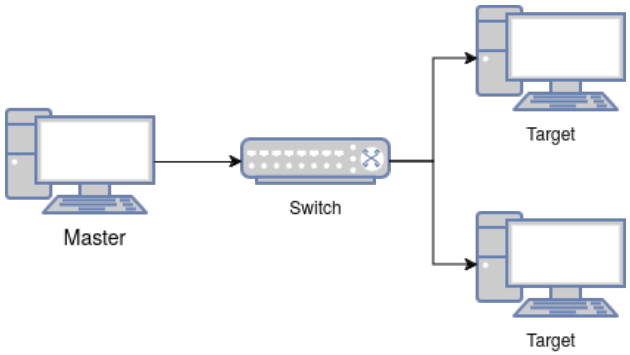
D. Perancangan Sistem

Perancangan dilakukan dengan menerapkan manajemen konfigurasi pada mesin virtual agar mendapatkan hasil waktu yang dibutuhkan untuk menerapkan konfigurasi. Secara keseluruhan menggunakan enam virtual machine, yang terdiri dari tiga instance Ubuntu Server 24.04 LTS dan tiga instance NixOS. Satu dari tiga virtual machine bersifat sebagai master dan dua sebagai target. Alasan perbedaan target dari Ansible dan NixOS adalah karena nixos-rebuild tidak bisa dijalankan pada sistem operasi selain NixOS. Ansible juga tidak dapat digunakan di NixOS karena sifat NixOS sendiri yang immutable membuat metode imperatif untuk konfigurasi sistem operasi tidak memungkinkan.Alasan berikutnya adalah dikarenakan kenyataan dilapangan dimana Ubuntu Server akan dikonfigurasi menggunakan Ansible dan menggunakan package manager apt, sedangkan untuk NixOS pasti akan dikonfigurasi menggunakan file konfigurasi ”.nix”. Topologi Pengujian

Baik master maupun target akan diuji menggunakan *virtual machine* yang dibuat dalam Proxmox VE. Berikut merupakan spesifikasi *virtual machine* yang dibutuhkan untuk penelitian ini:

TABEL I SPESIFIKASI INSTANCE

***Instance* Ansible**	
Sistem Operasi	Ubuntu Server 24.04 LTS
Processor	2
Memory	4 GB
Storage	50 GB
Jumlah *instance*	3
Instance NixOS	
Sistem Operasi	NixOS 23.11
Processor	2
Memory	4 GB
Storage	50GB
Jumlah *Instance*	3



Gbr. 2 Topologi Penelitian

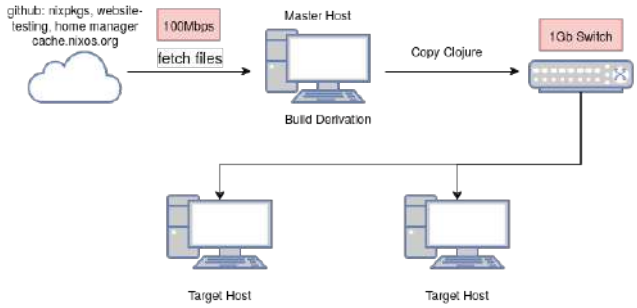
Perancangan topologi untuk penelitian ini adalah dengan sistem master dan target. Sistem ini akan membuat satu perangkat sebagai pusat kendali dari beberapa target yang menjadi endpoint dari manajemen konfigurasi. Dengan toplogi dari gambar 3 ini, diharapkan penerapan manajemen konfigurasi terpusat dan dapat diterapkan di dua mesin target dan memiliki hasil akhir yang sama dan konsisten antar mesin.

E. Implementasi dan Pengujian

Pada tahap ini, rancangan konfigurasi yang sudah dibuat akan diimplementasikan sesuai alur kerja konfigurasi yang

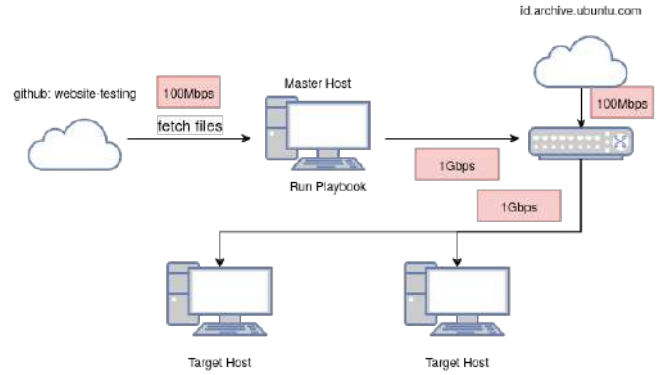
telah dibuat pada lingkungan mesin virtual agar lebih efisien dan lebih terisolasi. Instalasi dan konfigurasi dilakukan sesuai referensi dokumentasi dari masing-masing tool manajemen konfigurasi. Kustomisasi dari perangkat lunak yang diperlukan tergantung pada kebutuhan penelitian. Parameter pengujian yang akan dilakukan adalah menguji kecepatan waktu eksekusi dan *reproducibility* dari masing-masing tool manajemen konfigurasi.

Terkait konten dari konfigurasi dibuat semirip mungkin antara Ansible dengan NixOS agar menghasilkan hasil yang serupa pada target yang berbeda. Konfigurasi meliputi paket-paket yang di *install* pada target, konfigurasi web server, konfigurasi SSH, serta konfigurasi tingkat pengguna dalam sistem operasi.



Gbr. 3 Alur Penerapan Konfigurasi NixOS

Pada gambar 3 merupakan alur penerapan konfigurasi NixOS. *Master* akan melakukan *clone* dari *repository* nixpkgs, home-manager dan website-testing untuk mengambil konfigurasi sistem serta file *website* yang akan di *deploy*. Kemudian *master* akan melakukan *build* untuk konfigurasi keseluruhan sistem sehingga menghasilkan *nix derivation* yang membentuk NixOS. Hasil dari *derivation* tersebut kemudian akan di *copy* dan diterapkan ke dua mesin target.



Gbr. 4 Alur Penerapan Konfigurasi Ansible

Pada gambar 4 merupakan alur penerapan konfigurasi menggunakan Ansible. *Master* akan melakukan *clone* dari *repository* github website-testing. Kemudian *master* akan mulai menerapkan konfigurasi untuk target dan *menginstall* semua paket yang di definisikan dan melakukan konfigurasi yang diinginkan. Masing-masing target dengan sistem operasi Ubuntu Server akan mengunduh paket dari *repository* id.archive.ubuntu.com. Setelah konfigurasi sistem selesai diterapkan kemudian Ansible akan menyalin file *website* yang ada di *master* ke target.

Setiap kali perintah untuk melakukan konfigurasi di eksekusi, waktu eksekusi akan dicatat menggunakan perintah "time". Eksekusi akan dilakukan secara berulang untuk mendapatkan rata-rata waktu yang dibutuhkan untuk menerapkan konfigurasi. Patut di ingat bahwa *bandwidth* maksimal untuk menuju internet adalah 100Mbps dan *bandwidth local connection* adalah 1Gbps.

F. Analisis Hasil

Pada tahap ini, hasil dari implementasi dan pengujian akan dianalisis untuk mengetahui kecepatan penerapan konfigurasi dan konsistensi *tool* manajemen konfigurasi. Untuk mengetahui waktu yang dibutuhkan oleh *tool* manajemen konfigurasi untuk menerapkan konfigurasi, akan digunakan perintah "*time*". Untuk mengetahui konsistensi penerapan, akan dilihat dari hasil akhir konfigurasi setelah diterapkan secara berulang.

III. HASIL DAN PEMBAHASAN

A. Persiapan Lingkungan NixOS

Untuk persiapan lingkungan NixOS baik *master* maupun target adalah sebagai berikut:

1) Unduh ISO NixOS

Unduh ISO NixOS GNOME di https://nixos.org, kemudian *install* dalam VM Proxmox dengan BIOS(OVMF) UEFI q35 dan CPU tipe *host* agar kita dapat memanfaatkan semua *instruction set* dari CPU. Lanjutkan dengan spesifikasi CPU, RAM, dan HDD seperti yang telah dicantumkan pada bagian perancangan sistem. Untuk NixOS pada saat penulisan yaitu versi ISO 23.11 belum mendukung *secure boot* secara bawaan. Maka dari itu, perlu untuk menonaktifkan *secure boot* pada VM Proxmox agar ISO bisa dipakai. Saat *booting,spam* tombol 'Esc' untuk masuk ke *firmware setting* dan matikan *Secure Boot* pada bagian *Device Manager->Secure Boot Configuration ->Disable secure boot next->Attempt Secure Boot* kemudian klik spasi dan tekan 'y'. Tekan 'F10' untuk menyimpan konfigurasi. Tekan 'Esc' dua kali untuk keluar dan pilih *'Reset'*. Setelah itu, kita akan disambut dengan GNOME *Desktop Environment* seperti berikut

Gbr. 5 NixOS Calamares Installer

2) Lanjut Instalasi

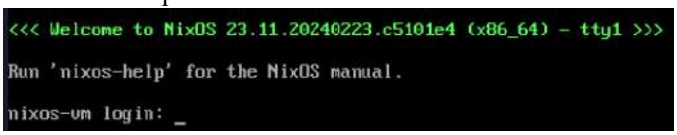
Setelah sukses *booting* dan masuk ke dalam *Live Environment*. Kita tinggal meneruskan proses instalasi seperti distribusi GNU/Linux pada umumnya. Untuk beberapa pilihan khusus yaitu kita harus centang opsi "*Allow Unfree Software"* untuk memastikan *proprietary driver* terpasang pada sistem. Kita juga hanya memilih opsi "*No Desktop"* karena kita hanya membutuhkan akses SSH.

3) Instalasi Stuck 46%

Apabila pada proses instalasi stuck di 46%, maka ini merupakan hal yang umum. Karena pada tahap ini, NixOS sedang melakukan build derivation untuk semua paket yang diperlukan.

4) Hasil Akhir Instalasi

Setelah instalasi sukses, kita akan disambut dengan console TTY NixOS seperti berikut:



Gbr. 6 NixOS TTY

5) Setup NixOS

Setelah itu, kita aktifkan SSH dan mematikan firewall dengan menambahkan baris dibawah ini kedalam file /etc/nixos/configuration.nix:

```
services.openssh.enable = true;
services.openssh.settings.PermitRootLogin ="yes";
networking.firewall.enable = false;
```

Kemudian, kita akan menambahkan beberapa paket untuk mempermudah dalam melakukan penelitian. Kita akan menginstall neovim sebagai *text editor* dan tmux sebagai *terminal multiplexer* untuk menjaga sesi SSH apabila koneksi SSH ke *server* terputus.

```
environment.systemPackages = with pkgs; [ neovim tmux ];
```

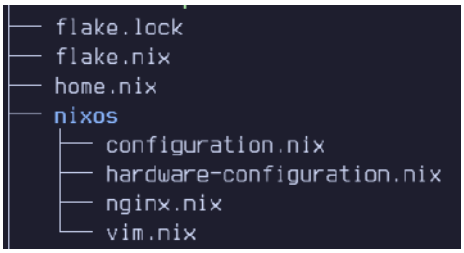
Untuk target, tambahkan opsi berikut agar nixos-rebuild dapat menginstall systemd-boot dengan versi lebih rendah dari yang telah terinstall:

```
environment.sessionVariables =
{NIXOS_INSTALL_BOOTLOADER = "1"; };
```

Setelah itu bisa diterapkan dengan perintah "sudo nixos-rebuild switch". Apabila tidak di spesifikasikan path, maka secara default akan mengarah ke /etc/nixos/configuration.nix.

Struktur konfigurasi NixOS

Untuk struktur konfigurasi target yang digunakan adalah sebagai berikut:



Gbr. 7 Struktur Konfigurasi NixOS

Gbr. 7 Struktur Konfigurasi NixOS

Dalam konfigurasi ini, flake.nix dan flake.lock diletakkan di parent directory yang memang menjadi basis untuk dieksekusinya perintah "nixos-rebuild switch -- flake .#nixos-vm –target-host username@target-ip", dimana opsi --flake sendiri terdiri dari flake-uri[#hostname] dan "–target-host" mengarah kepada target yang harus di konfigurasi.

```
 inputs = {
   nixpkgs.url = "github:nixos/nixpkgs/nixos-23.11";
   home-manager = {
    url = "github:nix-community/home-manager/release-
23.11";
    inputs.nixpkgs.follows = "nixpkgs";
   };
   testing-website = {
    url = "github:rizqirazkafi/testing-website";
    flake = false;
   };
 };
```

Flake.nix berisi tiga *inputs* yang dimana terdiri dari nixpkgs, home-manager, dan testing-website yang semua diambil dari github sebagai sumbernya. Hasil dari inputs digunakan dalam output untuk sistem yang didefinisikan dalam nixosConfiguration dengan nixos-vm sebagai hostname dari sistem seperti berikut:

```
{outputs = { self, nixpkgs, home-manager, ... }@inputs:
let system = "x86_64-linux"; pkgs = import nixpkgs {
inherit system; config = { allowUnfree = true; }; };
in{ nixosConfigurations = {nixos-vm =
nixpkgs.lib.nixosSystem { specialArgs = { inherit inputs
system pkgs; }; modules =
[ ./nixos/configuration.nix ]; }; }; }; }
```

Dengan flake, versi paket yang dihasilkan dalam output tidak akan berubah baik dari *channel* maupun *hash*. Ini dikarenakan, informasi tersebut disimpan dalam flake.lock. Dengan ini, sistem yang dihasilkan akan konsisten dan dapat disebut sebagai sistem yang *reproducible.* Contoh konen flake.lock adalah sebagai berikut:

```
{
"nodes": {"nixpkgs": {"locked": {
"lastModified": 1708702655,
"narHash": "sha256-qxT5jSxxx",
"owner": "nixos",
"repo": "nixpkgs",
"rev": "c5101e457206dd437330d283d6626944e28794b3",
"type": "github"},
"original": {
"owner": "nixos",
"ref": "nixos-23.11",
"repo": "nixpkgs",
"type": "github"},},}}
```

6) Konfigurasi inti configuration.nix

File configuration.nix berisikan konfigurasi sistem seperti system packages ,services, user packages, ssh option,dan lain-lain. File ini merupakan file inti dari konfigurasi sistem secara

menyeluruh dimana file inilah yang di evaluasi oleh nixos-rebuild. Semua module yang dibutuhkan sistem harus di referensikan dalam file ini agar konfigurasi dapat diterapkan ke sistem. Ini merupakan sebagian konten file configuration.nix yang digunakan pada penelitian ini:

```
{inputs, pkgs, ... }: {
nixpkgs.config.allowUnfree = true;
environment.systemPackages = with pkgs; [
lazygit tmux vim neovim wget git home-manager htop
ranger ripgrep ansible];
security.pam.enableSSHAgentAuth = true;
security.pam.enableSSHAgentAuth = true;
services.openssh={enable = true; ports = [ 9005 ];
settings.PasswordAuthentication = false;};}
```

7) Home Manager

Untuk konfigurasi Home Manager ada beberapa yang dimasukkan yaitu kredensial git dan BASH *completion*

```
home.username = "rizqirazkafi";
home.homeDirectory = "/home/rizqirazkafi";
programs.git = {
enable = true;
userName = "Rizqirazkafi";
userEmail = "rizqir****@gmail.com";};
programs.bash = {
enable = true;
shellAliases = { ll = "ls -la"; };
enableCompletion = true;
initExtra = "echo "Hello, what good
shall I do today?"";};
```

B. Persiapan Lingkungan Ansible

Untuk persiapan lingkungan Ansible baik *master* maupun target adalah sebagai berikut:

1) Unduh ISO

Unduh ISO Ubuntu 24.04 live server di *repository* terdekat seperti milik Kartolo maupun Ubuntu ID Archive. Buat VM di Proxmox dengan konfigurasi yang sama dengan lingkungan NixOS. Karena Ubuntu mendukung *secure boot* sepenuhnya, kita tidak perlu mematikan *secure boot* pada *firmware setting* seperti NixOS. Ketika sudah *booting*, maka akan muncul tampilan sebagai berikut:

Gbr. 8 Ubuntu 24.04 live server installer

2) Lanjut Instalasi

Setelah itu, lanjutkan instalasi seperti biasa dengan memilih basis instalasi Ubuntu Server. Pada penelitian ini tidak digunakan proxy dan untuk *mirror address* digunakan dari https://id.archive.ubuntu.com/ubuntu. Untuk disk digunakan seluruhnya tanpa menggunakan LVM dan tidak menggunakan Ubuntu Pro. SSH tidak akan di *install* menggunakan *installer.*

Setelah itu, kita bisa melanjutkan proses instalasi hingga selesai. Apabila sudah selesai, maka hasilnya akan seperti ini:

Gbr. 9 Instalasi Ubuntu Server berhasil

3) Konfigurasi Ansible

Untuk Ansible menggunakan konfigurasi yang lebih sederhana dengan menyalin file konfigurasi dari vim, tmux, ssh dan nginx ke *directory* konfigurasi sesuai *Filesystem Hierarchial Standard* (FHS). Untuk paket yang dibutuhkan di definisikan secara *global package* dimana paket di *install* untuk semua user.

```
- hosts: all
  become: yes
  become_user: root
  tasks:
  - name: install packages
    ansible.builtin.apt:
      pkg:
        - hello
        - neovim
        - tmux
        - vim
        - wget
        - ....
- name: copy nginx default config
  ansible.builtin.copy:
      src: default
      dest: /etc/nginx/sites-enabled/default
- name: copy php8.3-fpm config
  ansible.builtin.copy:
      src: php-demo.conf
      dest: /etc/php/8.3/fpm/pool.d
- name: enable nginx
  ansible.builtin.service:
      name: nginx
      state: restarted
```

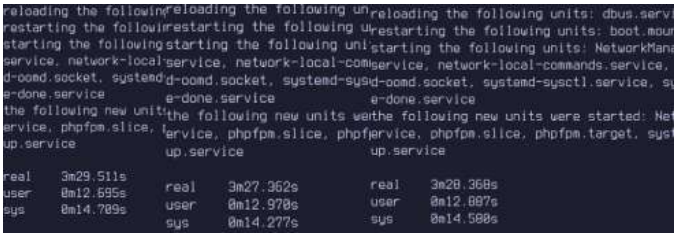
C. Pengujian Waktu Eksekusi

Pada pengujian ini dilakukan empat kategori pengujian yaitu: pasca *install*, pengulangan pasca *install*, *install* paket Go dan

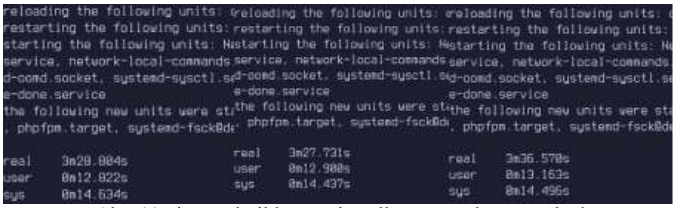
uninstall paket Go. Pengujian akan diukur waktu dari saat menjalankan perintah untuk menerapkan konfigurasi ke target hingga selesai menggunakan perintah "time". Dikarenakan nixos-rebuild pada saat penulisan belum memiliki kemampuan untuk menarget lebih dari satu host, maka untuk Ansible juga ukur waktu eksekusi per satu host dan paralel. Ketiga *host* baik *master* maupun *target* diakses melalui SSH dan hanya di mode CLI.

1) Pasca Install NixOS Tanpa Cache

Pada percobaan ini, master akan mengunduh semua binary dan dependency kemudian akan melakukan build berdasarkan derivation yang telah di definisikan oleh nixpkgs. Hasil dari derivation berupa closure yang disalin ke target. Setelah nixos-rebuild menyalin closure ke target, maka waktu eksekusi telah didapat.Setelah waktu eksekusi didapat, baik VM untuk target dan master akan dikembalikan ke snapshot sebelum dijalankannya nixos-rebuild. Percobaan dilakukan tiga kali untuk masing-masing target.



Gbr. 10 nixos-rebuild pasca install tanpa cache target pertama

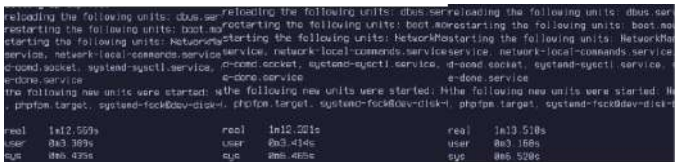


Gbr. 11 nixos-rebuild pasca install tanpa cache target kedua

2) Pasca Install NixOS dengan Cache

Pada percobaan ini, nixos-rebuild tidak lagi perlu mengunduh *binary* karena sudah di *build* di *master*. *Master* hanya perlu menyalin *closure* ke target. Percobaan dilakukan sebanyak tiga kali, dimana sebelum percobaan, VM dan target akan di *snapshot* sehingga dapat kembali ke *state* sebelum diterapkannya konfigurasi.

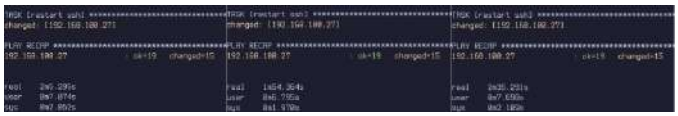
Gbr. 12 nixos-rebuild pasca install dengan cache target pertama



Gbr. 13 nixos-rebuild pasca install dengan cache target kedua

3) Pasca Install Ansible Single Target

Pada percobaan ini, dijalankan menggunakan perintah ansible-playbook dengan satu target pada satu waktu. Total target yang di tuju adalah dua dengan setiap kali dijalankan, VM target akan dikembalikan ke snapshot yang berisi state sebelum ansible-playbook diterapkan ke target. Percobaan ini dilakukan tiga kali di masing-masing target.

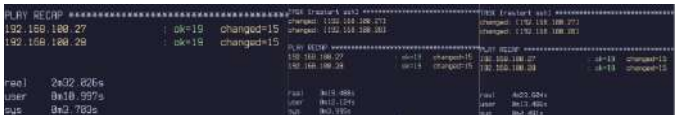


Gbr. 14 Ansible pasca install target pertama

Gbr. 15 Ansible pasca install target kedua

4) Pasca Install Ansible Multi Target

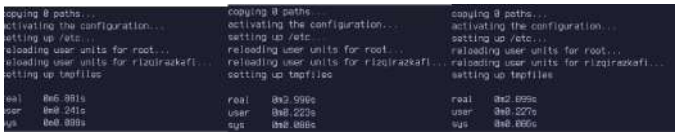
Pada percobaan ini, dijalankan menggunakan perintah ansible-playbook dengan dua target secara paralel. Setiap kali sebelum ansible-playbook diterapkan ke target, target akan dikembalikan ke snapshot sebelum ansible-playbook diterapkan.



Gbr. 16 Ansible pasca install multi target

5) Pengulangan Pasca Install NixOS

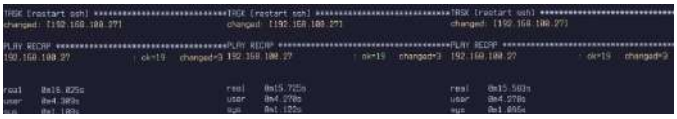
Untuk NixOS, dijalankan perintah nixos-rebuild dengan dua target. Masing-masing target dijalankan sebanyak tiga kali tanpa perlu mengembalikan VM ke kondisi sebelumnya dengan snapshot. Berikut hasil pengujiannya:



Gbr. 17 Pengulangan nixos-rebuild pasca install

6) Pengulangan Pasca Install Ansible Single Target

Untuk percobaan Ansible single target, perintah ansible-playbook dijalankan ke satu target dalam satu waktu. Ini digunakan untuk mendapatkan hasil perbandingan yang setara dengan nixos-rebuild yang hanya mampu mengkonfigurasi satu target dalam satu waktu.

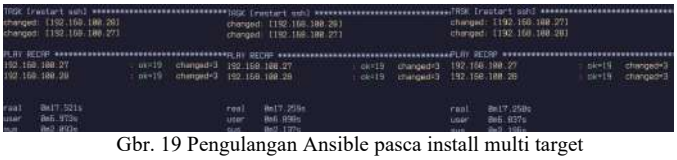


Gbr. 17 Pengulangan Ansible pasca install target pertama

Gbr. 18 Pengulangan Ansible pasca install target kedua

7) Pengulangan Pasca Install Ansible Multi Target

Untuk percobaan Ansible multi target, perintah ansible-playbook dijalankan ke dua target secara paralel. Berikut merupakan hasil dari percobaan Ansible multi target:



Gbr. 19 Pengulangan Ansible pasca install multi target

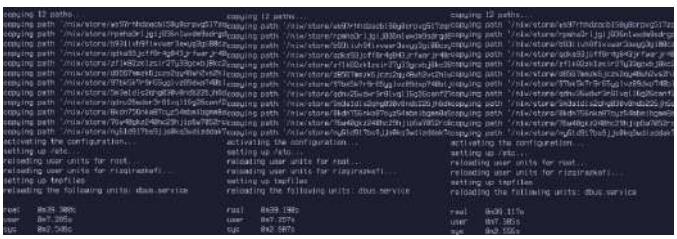
8) Instalasi Paket Go NixOS Tanpa Cache

Untuk NixOS, penambahan konfigurasi dilakukan pada file configuration.nix, pada bagian environment.systemPackages sebagai berikut:

```
environment.systemPackages = with pkgs; [ go ];
```

Pengujian dilakukan dengan pertama mengembalikan kondisi sistem baik master maupun target ke konfigurasi setelah pasca install, kemudian menambahkan ”go” kedalam konfigurasi. Ini dimaksudkan agar NixOS tidak menyimpan cache hasil build untuk paket Go. Konfigurasi kemudian diterapkan dengan nixos-rebuild oleh master ke target. Setelah master melakukan build, master kemudian menyalin hasilnya ke target. Percobaan ini dilakukan berulang sebanyak tiga kali per target.

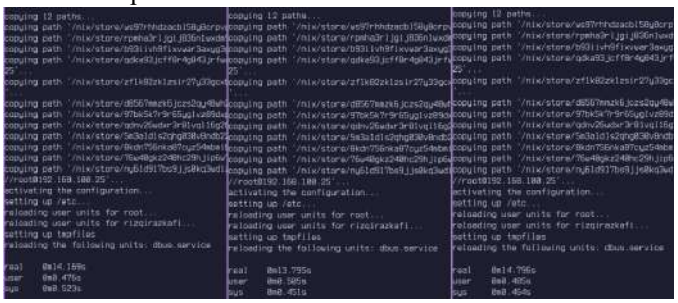
Gbr. 20 Instalasi paket Go NixOS tanpa cache target pertama



Gbr. 21 Instalasi paket Go NixOS tanpa cache target kedua

9) Instalasi Paket Go NixOS dengan Cache

Pada percobaan ini, NixOS telah melakukan build untuk paket GO dan nixos-rebuild hanya perlu menyalin hasilnya ke target. Dari percobaan tersebut, didapatkan hasil sebagai berikut dengan pertimbangan bandwidth dari master ke target adalah 1 Gbps.



Gbr. 22 Instalasi paket Go NixOS dengan cache target pertama

Gbr. 23 Instalasi paket Go NixOS dengan cache target kedua

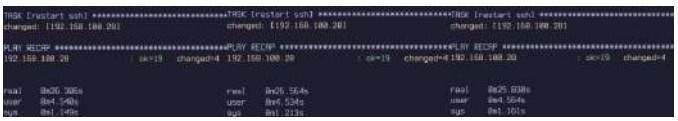
10) Instalasi Paket Go Ansible Single Target

Pada percobaan Ansible single target, penambahan paket ”GO” di definisikan dalam file yml yang digunakan seperti berikut:

```
- name: install go
  ansible.builtin.apt:
    pkg:
      - golang-go
    state: present
```

Pada konfigurasi tersebut, ”state: absent” digunakan untuk memberitahu Ansible untuk memastikan apakah paket ”golang-go” benar-benar ada pada sistem. Percobaan ini dilakukan sebanyak tiga kali untuk masing-masing target dan setiap sebelum percobaan kondisi VM akan dikembalikan ke *snapshot* pasca install.

Gbr. 24 Ansible install Go target pertama

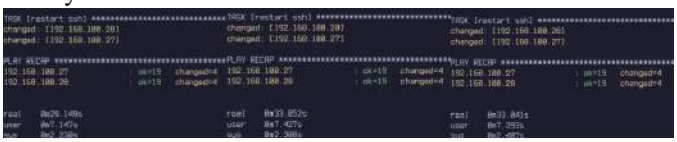


Gbr. 25 Ansible install Go target kedua

11) Instalasi Paket Go Ansible Multi Target

Pada percobaan Ansible multi target, sama seperti percobaan Ansible single target pada bagian konfigurasi dan snapshot.

Namun yang membedakan adalah, perintah ansible-playbook dijalankan pada dua target secara paralel. Berikut untuk hasilnya:

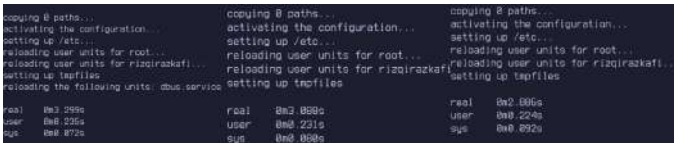


Gbr. 26 Ansible install Go multi target

12) *Uninstall Paket Go NixOS*

Pada pengujian ini, deskripsi paket Go akan dihapus atau dikomen pada file konfigurasi configuration.nix seperti berikut: environment.systemPackages = with pkgs; [# go];

Perintah nixos-rebuild kemudian dijalankan pada masing-masing target sebanyak tiga kali dan hasilnya sebagai berikut:



Gbr. 27 NixOS uninstall Go target pertama

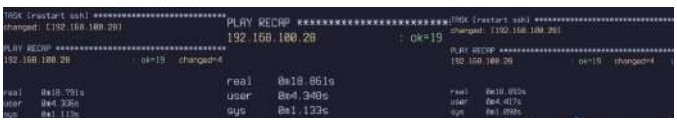
Gbr. 28 NixOS uninstall Go target kedua

13) *Uninstall Paket Go Ansible Single Target*

Untuk Ansible, kita hanya perlu mengganti ”state” untuk paket Go dari ”present” menjadi ”absent”. Dengan ini kita memberitahu Ansible untuk memastikan paket ”golang-go” tidak ada pada target. Perintah ansible-playbook kemudian dijalankan pada satu target dalam satu waktu sebanyak tiga kali dengan hasil dan konfigurasinya adalah sebagai berikut:

```
- name: install go
  ansible.builtin.apt:
    pkg:
      - golang-go
    state: absent
```

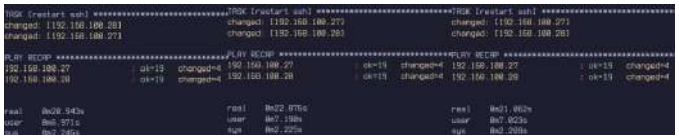
Gbr. 29 Ansible uninstall Go target pertama



Gbr. 30 Ansible uninstall Go target kedua

14) *Uninstall Paket Go Ansible Multi Target*

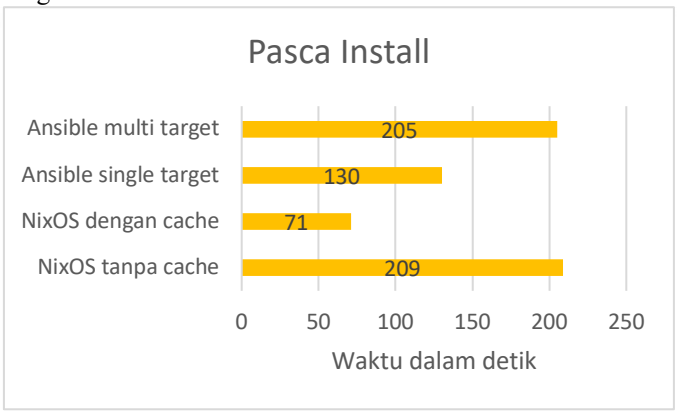
Untuk percobaan Ansible multi target sama seperti Ansible single target. Pembedanya terletak hanya di jumlah target yang dituju dimana untuk Ansible multi target adalah sebanyak dua target dalam percobaan ini. Berikut adalah hasil dari percobaan ini:



Gbr. 31Ansible uninstall Go multi target

D. *Grafik Perbandingan Waktu dalam Detik*

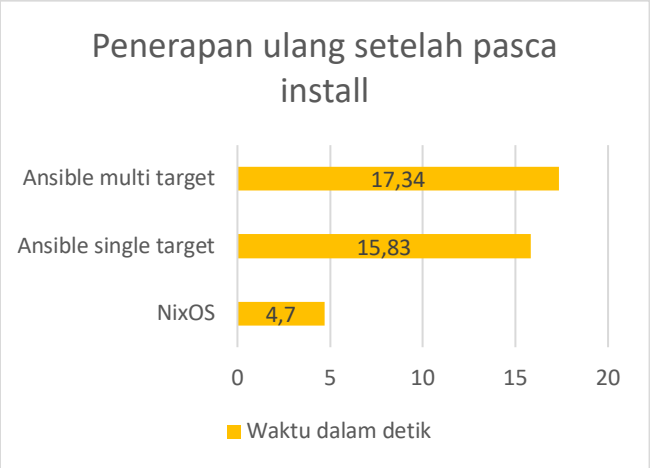
Dari data-data real time diatas, maka didapatkan rata-rata data grafik waktu yang didapat dalam satuan detik adalah sebagai berikut:



Gbr. 32 Grafik perbandingan waktu pasca install

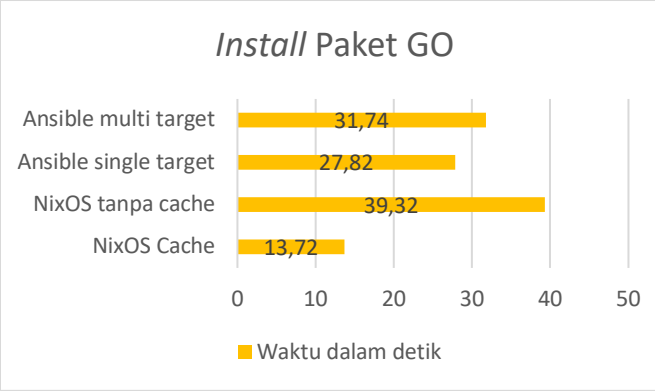
Dapat kita lihat bahwa NixOS tanpa *cache* butuh waktu lebih lama dikarenakan sistem *package manager* yang akan mengunduh *dependency* tiap paket secara terpisah apabila ada perbedaan versi. Faktor lain adalah tidak adanya sistem *mirror* seperti yang dimiliki oleh *repository* Ubuntu dimana paket bisa di simpan di *server* lain yang dekat, setidaknya tidak secara langsung. Namun apabila NixOS sudah menyimpan *cache*, maka NixOS hanya perlu menyalin data tersebut ke target dimana kecepatannya bergantung pada koneksi antara *host* dan target.

Untuk Ansible terutama untuk multi target memang rata-rata waktu lebih lama. Namun melihat hasil yang didapat dari tiga kali percobaan, waktunya sangat fluktuatif dan tidak konsisten. Namun apabila kita tambahkan hasil dari NixOS tanpa *cache* dengan NixOS dengan *cache*, maka total dari rata-ratanya adalah 280 detik untuk sekali build dan dua kali copy. Waktu eksekusi untuk NixOS tentu akan bertambah seiring bertambahnya target yang perlu di konfigurasi pada satu waktu. Berbeda dengan Ansible yang dapat bekerja secara paralel.



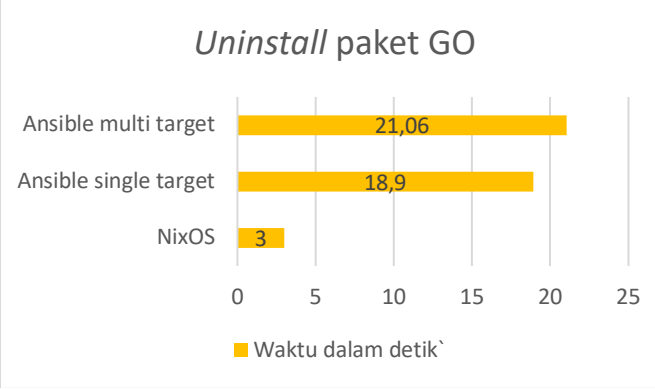
Gbr. 33 Grafik perbandingan waktu penerapan ulang setelah pasca install

Disini dapat dilihat Ansible membutuhkan waktu lebih lama untuk melakukan pengecekan dikarenakan Ansible melakukan pengecekan tiap task pada setiap target sedangkan NixOS hanya mengecek perubahan pada konfigurasi dan membandingkan closure yang ada pada target dengan *host*.



Gbr. 34 Grafik perbandingan waktu *install* paket GO

Pada grafik diatas, dapat dilihat lagi-lagi NixOS dengan cache lebih unggul untuk kasus penambahan paket GO untuk satu target.



Gbr. 35 Grafik perbandingan waktu *uninstall* paket GO

Pada grafik diatas dapat terlihat bahwa untuk menghapus paket, NixOS lebih cepat daripada Ansible. Ini disebabkan karena NixOS tidak langsung menghapus paket dari sistem, melainkan hanya mengapus symlink. Binary tetap disimpan apabila di kemudian hari paket dibutuhkan untuk diinstall kembali. NixOS hanya menghapus paket apabila garbage collector di eksekusi.

E. Hasil Konfigurasi

1) NixOS

Dapat dilihat bahwa NixOS berhasil mengkonfigurasi target dengan paket dan konfigurasi service seperti Nginx dan SSH.

Gbr. 36 Hasil konfigurasi NixOS

2) Ansible

Hasil dari Ansible juga menunjukkan bahwa Ansible mampu mengkonfigurasi Ubuntu Server sesuai dengan konfigurasi dari keinginan penulis seperti dibawah ini.

Gbr. 37 Hasil konfigurasi Ansible

F. Deklaratif dan Imperatif

Dalam percobaan diatas, terdapat beberapa hal yang ditemui oleh penguji tentang tool manajemen konfigurasi. Ansible Playbook menggunakan metodeimperatif dalam praktiknya, dimana Ansible Playbook berisi mengenai apa-apa saja yang harus dilakukan oleh Ansible. Ini menyebabkan ketidak cocokan antara Ansible Playbook dengan kondisi akhir sistem. Apabila kita mendeskripsikan *task* untuk menginstall dan menjalankan Nginx lalu kita terapkan,maka Nginx akan terinstall. Namun apabila *task* tersebut dihapus dari Ansible Playbook kemudian kita jalankan, maka Nginx akan tetap berjalan.

Pada sisi lain, konfigurasi NixOS merupakan representasi dari kondisi akhir sistem. Apabila kita pada awalnya mendeskripsikan *service* Nginx, maka Nginx akan berjalan

pada sistem. Apabila kita menghapus pendeskripsian Nginx dari file konfigurasi, maka Nginx tidak akan berjalan. Metode inilah yang disebut dengan Deklaratif oleh NixOS. Sebagai berikut contohnya:

```
imports = [
./hardware-configuration.nix
./vim.nix
inputs.home-manager.nixosModules.home-manager
./nginx.nix
];
```

Maka Nginx akan di konfigurasi dan berjalan di sistem target seperti berikut:

Gbr. 38 Hasil Nginx NixOS

Dan apabila kita ubah sebagai berikut:

```
imports = [
./hardware-configuration.nix
./vim.nix
inputs.home-manager.nixosModules.home-manager
# ./nginx.nix
];
```

Maka hasilnya akan seperti berikut dimana bahkan service Nginx dihapus dari sistem.

```
[rizqirazkafi@nixos-vm:~]$ systemctl status nginx
Unit nginx.service could not be found.
```

Gbr. 39 Nginx NixOS

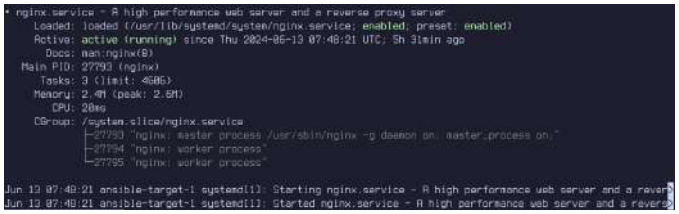
Sedangkan untuk Ansible:

```
tasks:
- name: install packages
  ansible.builtin.apt:
    pkg:
      - nginx
tasks:
- name: enable nginx
  ansible.builtin.service:
    name: nginx
    state: restarted
```

Maka hasilnya akan sebagai berikut:

Gbr. 40 Ansible Nginx aktif

Dan apabila baris diatas kita beri komentar didepannya, maka hasilnya sebagai berikut:



Gbr. 41Ansible Nginx pasca komentar

Hanya jika kita mengganti state dari ”enable nginx” ke ”stopped” dan mendefinisikan ”state” dari pkg Nginx ke ”absent” barulah Nginx dihentikan dan dihapus dari sistem.

IV. KESIMPULAN

A. Waktu Eksekusi Tool Manajemen Konfigurasi

Dalam uji coba pasca install, urutan waktu eksekusi dari yang tercepat hingga terlambat adalah NixOS dengan *cache*, Ansible *multi target*, Ansible *single target*, NixOS tanpa *cache*. Perintah nixos-rebuild tidak dapat digunakan untuk melakukan manajemen konfigurasi secara paralel sedangkan Ansible Playbook dapat dijalankan secara paralel. Ini berdampak pada penerapan dunia nyata dimana semakin banyak target maka waktu yang dibutuhkan untuk penerapan menggunakan nixos-rebuild akan memakan waktu lebih lama walaupun *cache* / *closure* telah di *build*.

B. Deklaratif

Ansible mampu mengkonfigurasi sistem operasi berbasis GNU/Linux (pada kasus ini Ubuntu Server), namun hanya mampu pada tingkat imperatif dan bukan deklaratif. NixOS dengan nixos-rebuild mampu mengkonfigurasi sistem operasi berbasis GNU/Linux (pada kasus ini NixOS) secara deklaratif.

V. SARAN

Disarankan membuat penelitian tentang perbandingan performa waktu eksekusi, maka bisa menggunakan Colmena atau NixOps agar bisa menerapakan konfigurasi NixOS secara paralel. Disarankan bagi peneliti untuk mengubah jenis penyimpanan target dari HDD ke SSD untuk melihat apakah kecepatan penyimpanan target berpengaruh pada waktu eksekusi.

REFERENSI

[1] Alfiandi, T., Diansyah, T. M., & Liza, R. (2020). Analisis Perbandingan Manajemen Konfigurasi Menggunakan Ansible dan Shell Script Pada Cloud Server Deployment AWS. *JiTEKH*, *8*(2), 78–84. https://doi.org/10.35447/jitekh.v8i2.308

[2] Ansible, Red Hat. (2016). *Ansible is Simple IT Automation*. Ansible.com. https://www.ansible.com/

[3] *Flakes - NixOS Wiki*. (n.d.). Nixos.wiki. Retrieved January 2024, from https://nixos.wiki/wiki/Flakes

[4] Hariyadi, I. P., & Marzuki, K. (2020). Implementation of Configuration Management Virtual Private Server Using Ansible. *MATRIK : Jurnal Manajemen, Teknik Informatika Dan Rekayasa Komputer*, *19*(2), 347–357. https://doi.org/10.30812/matrik.v19i2.724

[5] *Home Manager Manual*. (n.d.). Nix-Community.github.io. Retrieved January 2024, from https://nix-community.github.io/home-manager/

[6] *How Nix Works | Nix & NixOS*. (n.d.). Nixos.org. Retrieved December 20, 2023, from https://nixos.org/guides/how-nix-works/

[7] Kumpulainen, K. (2019). NixOS: Järjestelmäkonfiguraation Hallintaan Erikoistunut Linux-jakelu. *Trepo.tuni.fi*. https://urn.fi/URN:NBN:fi:tty-201905311795

[8] Pratama, M. A. A., & Hariyadi, I. P. (2021). Otomasi Manajemen dan Pengawasan Linux Container (LCX) Pada Proxmox VE Menggunakan Ansible. *Jurnal Bumigora Information Technology (BITe)*, *3*(1), 82–95. https://doi.org/10.30812/bite.v3i1.807

[9] Thufail Qolba, A. (2023). *Configuration Management dengan Ansible dan Telegram untuk Automasi Laboratorium Komputer di JTIK* (p. 66) [Thesis]. https://repository.pnj.ac.id/id/eprint/12406/1/Halaman%20Identitas%20Skripsi.pdf