Analisis Perbandingan *Ansible* dengan *Terraform* untuk Memanajemen *Cloud Server*

Azqia Nabila[1], Indrawati[2*], Muhammad Davi[3]

[1,2,3] *Jurusan Tekniknologi Informasi dan Komputer Politeknik Negeri Lhokseumawe*
Jln. B.Aceh Medan Km.280 Buketrata 24301 INDONESIA
[1]nabilaazqia4@gmail.com
[2*]indrawati@pnl.ac.id
[3]muhammad.davi@pnl.ac.id

Abstrak*—*Cloud Computing* menjadi komponen vital dalam operasional IT, namun pengelolaan manual yang memakan waktu dan rentan kesalahan mendorong adopsi *Infrastructure as Code* (IaC) untuk otomatisasi. *Ansible* dan *Terraform*, dua alat populer dalam IaC, menawarkan pendekatan otomatisasi yang berbeda dan saling melengkapi. Penelitian ini bertujuan untuk menganalisis dan membandingkan *Ansible* dan *Terraform* dalam manajemen *cloud server* di *Azure*, guna mengidentifikasi kelebihan dan kekurangan masing-masing alat. Metode penelitian mencakup evaluasi kualitas layanan (*Quality of Service*) dan efisiensi penggunaan sumber daya sistem, meliputi penggunaan CPU, memori, dan waktu eksekusi. Hasil penelitian menunjukkan bahwa *Ansible* membutuhkan waktu eksekusi lebih lama dibandingkan *Terraform*, dengan *Ansible* memerlukan 1 menit 34,158 detik, sementara *Terraform* hanya 1 menit 25,974 detik. Penggunaan CPU pada Terraform lebih efisien, mencapai puncak 41% di awal, sedangkan *Ansible* tidak melebihi 100%. Dalam hal QoS, *Terraform* memiliki *throughput* lebih tinggi, yaitu 15.000 Kbps dibandingkan *Ansible* yang hanya 76 Kbps. Keduanya memiliki *packet loss* sebesar 0%. Selain itu, *Terraform* juga lebih unggul dalam total dan rata-rata *delay*, serta *jitter*. Kesimpulannya, *Terraform* lebih efisien dalam eksekusi dan QoS, sementara *Ansible* tidak menunjukkan keunggulan khusus lainnya.
***Kata kunci*—Infrastruktur IT, *Infrastructure As Code (IaC), Ansible, Terraform, Manajemen Cloud Server* .**

Abstract*—*Cloud Computing has become an important element in IT infrastructure operations in various sectors, including education in the Department of Information and Computer Technology (ICT) of Politeknik Negeri Lhokseumawe. Time-consuming and error-prone manual IT management encourages the adoption of Infrastructure as Code (IaC) methods for automation. Ansible and Terraform are two popular tools in IaC used for this purpose. This research aims to analyze and compare Ansible and Terraform in cloud server management in Azure, to identify the advantages and disadvantages of each tool. The research method includes evaluating the quality of service (QoS) and efficiency of system resource usage, including CPU usage, memory, and execution time. The results show that Ansible takes longer to execute than Terraform, with Ansible requiring 1 minute 34.158 seconds, while Terraform only 1 minute 25.974 seconds. CPU usage on Terraform was more efficient, reaching a peak of 41% at the beginning, while Ansible did not exceed 100%. In terms of QoS, Terraform has a higher throughput of 15,000 Kbps compared to Ansible's 76 Kbps. Both have a packet loss of 0%. In addition, Terraform is also superior in total and average delay, and jitter. In conclusion, Terraform was more efficient in execution and QoS, while Ansible showed no other particular advantages.
Keywords*—*IT infrastructure, Infrastructure as Code (IaC), Ansible, Terraform, cloud server management.

I. Pendahuluan

Cloud Computing telah menjadi elemen krusial dalam mendukung operasional infrastruktur TI di berbagai sektor, termasuk pemerintahan, pendidikan, kesehatan, dan industri swasta. Teknologi ini memungkinkan organisasi memanfaatkan sumber daya komputasi seperti penyimpanan, jaringan, dan daya pemrosesan dengan cara yang efisien dan fleksibel melalui internet, tanpa perlu memiliki atau mengelola infrastruktur fisik secara langsung [1]. Dengan layanan on-demand, *Cloud Computing* tidak hanya mengurangi beban operasional dan biaya, tetapi juga meningkatkan kemampuan organisasi untuk beradaptasi dengan dinamika bisnis yang berubah dengan cepat. Namun, pengelolaan infrastruktur TI secara manual masih menghadirkan tantangan besar, karena melibatkan banyak langkah yang memerlukan waktu, tenaga, serta berisiko tinggi terhadap kesalahan manusia. Infrastruktur yang tidak dikelola dengan baik dapat mengakibatkan inefisiensi, ketidakstabilan, dan bahkan kegagalan sistem, yang berdampak pada ketidakmampuan bisnis untuk merespons perubahan lingkungan yang dinamis [2].

Cloud server, sebagai salah satu penerapan *cloud computing*, memberikan pengguna kemampuan untuk mengatur sumber daya *hardware* sesuai dengan kebutuhan layanan yang digunakan, mirip dengan konsep virtualisasi server. Pengguna dapat mengakses dan menggunakan sumber daya komputasi seperti penyimpanan, pemrosesan, dan jaringan melalui internet, tanpa harus membeli dan mengelola perangkat keras secara fisik. Penyedia layanan *cloud* menawarkan berbagai model, termasuk *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS), dan *Software as a Service* (SaaS). Keuntungan utama dari *cloud server* adalah fleksibilitas dan skalabilitas, yang memungkinkan pengguna untuk menambah atau mengurangi sumber daya komputasi sesuai kebutuhan bisnis seperti menambah CPU, memori, atau penyimpanan saat beban kerja meningkat, dan menguranginya kembali saat beban kerja menurun. Hal ini membantu dalam optimalisasi biaya operasional karena pengguna hanya membayar untuk sumber daya yang digunakan [3].

Microsoft Azure adalah platform *cloud* yang menawarkan layanan komprehensif, termasuk IaaS, dengan manfaat seperti skalabilitas dinamis, keandalan tinggi, dan keamanan data yang optimal. Azure memungkinkan penyesuaian sumber daya tanpa investasi awal besar, menyediakan kontrol akses ketat, enkripsi data, serta kepatuhan standar industri. Dengan model pembayaran *pay-as-you-go*, pengguna membayar hanya untuk yang digunakan. Azure juga menawarkan fleksibilitas layanan komputasi, penyimpanan, AI, dan IoT sesuai kebutuhan bisnis [4].

Manajemen *Cloud Server* menjadi praktik penting dalam mengelola dan mengawasi sumber daya komputasi di lingkungan *cloud*, baik publik, privat, maupun *hybrid*. Proses ini mencakup provisioning sumber daya cloud, manajemen konfigurasi, dan keamanan *cloud server*, yang mendukung aplikasi dan layanan berbasis *cloud* secara efisien, efektif, dan aman. *Provisioning* sumber daya *cloud* adalah proses penyediaan *server virtual*, penyimpanan, jaringan, dan layanan lainnya di lingkungan *cloud*, yang dapat diotomatisasi menggunakan alat seperti *Terraform, Ansible,* atau AWS CloudFormation. Manajemen konfigurasi memastikan konsistensi perangkat keras dan perangkat lunak di seluruh mesin virtual, mengurangi risiko kesalahan konfigurasi manual, serta meningkatkan keamanan dan performa sistem. Keamanan *cloud server* mencakup pengelolaan akses, enkripsi data, dan penerapan kebijakan keamanan untuk melindungi data dan aplikasi di *cloud* [5].

Ubuntu adalah sistem operasi *open source* yang dibangun berdasarkan Linux, bertujuan untuk menyediakan perangkat lunak secara gratis, dapat digunakan dalam bahasa lokal, serta memberikan kebebasan bagi pengguna untuk menyesuaikan dan mengubah program sesuai kebutuhan mereka. Ubuntu mendorong penggunaan, perbaikan, dan penyebaran perangkat lunak berdasarkan prinsip pengembangan perangkat lunak bebas. Sebagai pengembangan dari Debian, Ubuntu dikenal dengan sistem manajemen paket yang kuat, yang mempermudah instalasi dan penghapusan program, menjadikannya salah satu distro Linux yang banyak dipilih untuk digunakan di *cloud server* karena stabilitas dan kemudahan manajemennya [6].

Virtual Machine (VM) adalah perangkat lunak yang beroperasi dalam sistem operasi komputer, berfungsi seperti komputer fisik berbasis perangkat lunak. VM adalah wadah perangkat lunak yang terisolasi secara ketat, yang dapat menjalankan sistem operasi dan aplikasi seolah-olah itu adalah komputer fisik. Meskipun VM tidak memerlukan perangkat keras fisik, VM tetap memiliki komponen seperti CPU, RAM, hard disk, dan kartu antarmuka jaringan (NIC), layaknya komputer fisik. VM memungkinkan bisnis untuk menjalankan beberapa sistem operasi pada perangkat keras yang sama secara simultan, seperti Windows dan Linux, tanpa memerlukan dua unit fisik yang terpisah. Penggunaan umum VM mencakup pengujian aplikasi dalam lingkungan yang aman dan menjalankan perangkat lunak yang memerlukan sistem operasi berbeda. Tujuan utama virtualisasi adalah untuk mengurangi biaya yang terkait dengan perangkat keras fisik, seperti pemeliharaan, pendinginan, energi, dan ruang fisik. Dengan menempatkan beberapa sistem operasi ke dalam struktur virtual, bisnis dapat mengoperasikan banyak instans dari perangkat keras yang sama, sehingga mengurangi kebutuhan akan perangkat keras fisik tambahan dan biaya *overhead* yang berlebihan [7].

CPU, atau prosesor, adalah sirkuit elektronik yang berfungsi sebagai otak dan pusat pengendalian komputer. CPU bertanggung jawab melaksanakan instruksi dari program komputer dengan melakukan operasi aritmatika dasar, logika, pengontrolan, dan input/output (I/O). CPU melakukan operasi matematika dasar seperti penjumlahan, pengurangan, perkalian, dan pembagian, serta operasi logika seperti AND, OR, NOT, dan XOR. Selain itu, CPU juga mengatur aliran data antara berbagai komponen komputer, seperti memori, perangkat input/output, dan unit penyimpanan, serta menangani interupsi dari perangkat keras. Memantau penggunaan CPU adalah langkah penting dalam optimasi kinerja, deteksi masalah, perencanaan kapasitas, dan manajemen energi. Data penggunaan CPU membantu dalam mengidentifikasi aplikasi yang mengonsumsi banyak sumber daya, mendeteksi masalah sistem, serta merencanakan kapasitas untuk menangani beban kerja saat ini dan di masa depan [8].

Bpytop adalah alat pemantauan sumber daya berbasis Python yang sepenuhnya gratis dan *open-source*. Dirancang untuk memantau sumber daya sistem secara *real-time,* Bpytop mampu menampilkan informasi tentang penggunaan disk, jaringan, proses, dan CPU secara detail. Alat ini memiliki fungsi yang serupa dengan program pemantauan lainnya seperti top, htop, dan bashtop, tetapi menawarkan beberapa keunggulan tambahan. Sebagai alat lintas *platform*, Bpytop dapat diinstal pada berbagai sistem operasi, termasuk Linux, macOS, dan FreeBSD, menjadikannya pilihan fleksibel bagi pengguna di berbagai lingkungan. Alat ini juga dilengkapi dengan antarmuka pengguna yang intuitif dan mudah dioperasikan, mendukung penggunaan baik dengan *keyboard* maupun *mouse*. Bpytop menggunakan beberapa satuan ukuran untuk menampilkan informasi tentang penggunaan sumber daya, seperti Gibibyte (GiB), Mebibyte (MiB), Kibibyte (KiB), Byte (B), serta bits per second (bps). Selain itu, Bpytop menawarkan fitur seperti filter informasi, tampilan statistik yang terperinci, dan opsi konfigurasi yang dapat disesuaikan, memberikan pengalaman pemantauan sistem yang optimal bagi pengguna [9].

Kemampuan suatu jaringan untuk menyediakan layanan yang baik, seperti bandwidth, pengelolaan jitter, dan delay, dikenal sebagai *Quality of Service* (QoS). Parameter QoS meliputi latency, jitter, packet loss, throughput, dan *Mean Opinion Score* (MOS). Kualitas jaringan sangat bergantung pada infrastruktur yang digunakan, di mana faktor-faktor seperti redaman, distorsi, dan suara dapat memengaruhi nilai QoS. Beberapa masalah seperti *bandwidth*, *latency*, dan *jitter* dapat berdampak besar pada kinerja jaringan komputer. Sebagai contoh, *video streaming* dapat mengalami masalah jika *bandwidth* tidak cukup, *latency* tidak stabil, atau *jitter* berlebihan. Fitur QoS dapat menangani masalah ini dengan mengontrol pengiriman paket data, mengurangi *latency*, dan

menyesuaikan jitter sesuai kebutuhan aplikasi jaringan, sehingga memastikan kualitas layanan tetap optimal [10].

Parameter QoS adalah metrik penting yang digunakan untuk mengukur kualitas jaringan dalam mendukung kinerja aplikasi dan layanan. Beberapa parameter utama dari QoS yang sering digunakan adalah *throughput, delay, jitter,* dan *packet loss*. *Throughput* mengukur kecepatan transfer data efektif dalam jaringan dan mencerminkan jumlah paket data yang berhasil diterima di tujuan dalam periode waktu tertentu. Semakin tinggi nilai *throughput*, semakin baik performa jaringan [11]. *Delay* merupakan waktu yang diperlukan oleh paket data untuk mencapai tujuannya dari titik asal. Keterlambatan yang tinggi pada *delay* dapat mengganggu aplikasi yang memerlukan respons cepat, seperti komunikasi VoIP atau *streaming* video [2]. *Jitter* menggambarkan variasi dalam waktu keterlambatan antar paket data yang tiba di tujuan. Perubahan dalam beban trafik dan kondisi jaringan yang tidak stabil sering menjadi penyebab utama tingginya jitter, yang dapat berdampak buruk pada aplikasi yang memerlukan pengiriman data yang konsisten dan tepat waktu. *Packet loss,* di sisi lain, terjadi ketika paket data gagal mencapai tujuan akibat berbagai faktor, seperti kongesti jaringan, kesalahan pada media fisik, atau *overload* pada *buffer*. Kehilangan paket yang tinggi menunjukkan adanya masalah signifikan dalam jaringan yang dapat mengganggu kualitas layanan yang diberikan [2].

TShark adalah alat baris perintah yang terkenal dan sangat efektif sebagai penganalisis jaringan. Dikembangkan oleh tim di balik Wireshark, cara kerjanya sangat mirip dengan Tcpdump, namun TShark memiliki beberapa *decoder* dan filter yang lebih kuat. TShark mampu menangkap informasi paket data dari berbagai lapisan jaringan dan menampilkannya dalam berbagai format. TShark digunakan untuk menganalisis lalu lintas jaringan secara *real-time* dan juga dapat membaca file .pcap untuk menganalisis data, mengeksplorasi detail koneksi, serta membantu profesional keamanan dalam mengidentifikasi masalah jaringan mereka. Sebagai alat berbasis baris perintah, TShark dapat melakukan hampir semua yang bisa dilakukan Wireshark [12].

Untuk mengatasi tantangan pengelolaan infrastruktur TI, konsep *Infrastructure as Code* (IaC) muncul sebagai solusi inovatif yang memungkinkan otomatisasi konfigurasi, manajemen, dan pengembangan melalui kode yang dapat dijalankan secara programatik. IaC memungkinkan manajemen infrastruktur yang lebih cepat, konsisten, dan dapat diulang, mengurangi potensi kesalahan manusia dan memastikan standar konfigurasi yang seragam di seluruh sistem [13]. Di antara alat-alat yang mendukung pendekatan IaC, *Ansible* dan *Terraform* adalah dua pilihan yang populer dan sering digunakan dalam otomatisasi pengelolaan infrastruktur *cloud*. Keduanya menawarkan pendekatan yang berbeda tetapi saling melengkapi dalam hal otomatisasi infrastruktur. *Ansible* terkenal karena kemudahan penggunaannya dan kemampuannya untuk mengelola konfigurasi dan *deployment* secara deklaratif melalui *playbooks* berbasis YAML. *Ansible* memungkinkan pengguna mendefinisikan tugas-tugas otomatisasi dengan bahasa yang mudah dibaca dan ditulis, serta dapat digunakan secara fleksibel pada berbagai platform dan aplikasi tanpa perlu agen tambahan [14]. *Ansible* sangat berguna dalam mengelola konfigurasi perangkat lunak, *patching*, dan automasi tugas rutin yang bersifat konfigurasional. Di sisi lain, Terraform adalah alat manajemen infrastruktur yang lebih terstruktur dan efisien, terutama dalam skenario yang kompleks, dengan fitur *state management* yang kuat dan dukungan untuk berbagai penyedia layanan *cloud* [15].

Penelitian ini bertujuan untuk membandingkan efisiensi penggunaan sumber daya sistem serta QoS antara implementasi otomatisasi manajemen *cloud server* menggunakan *Ansible* dan *Terraform*. Studi ini mengevaluasi aspek keandalan, skalabilitas, dan kinerja sistem dengan mempertimbangkan parameter QoS seperti *throughput*, *delay*, *jitter*, dan *packet loss*, serta kecepatan *upload* dan *download*. Dengan pemahaman yang lebih mendalam tentang perbedaan kedua alat ini, diharapkan organisasi dapat mengoptimalkan pengelolaan infrastruktur TI mereka dan meningkatkan kualitas layanan *cloud* secara keseluruhan.

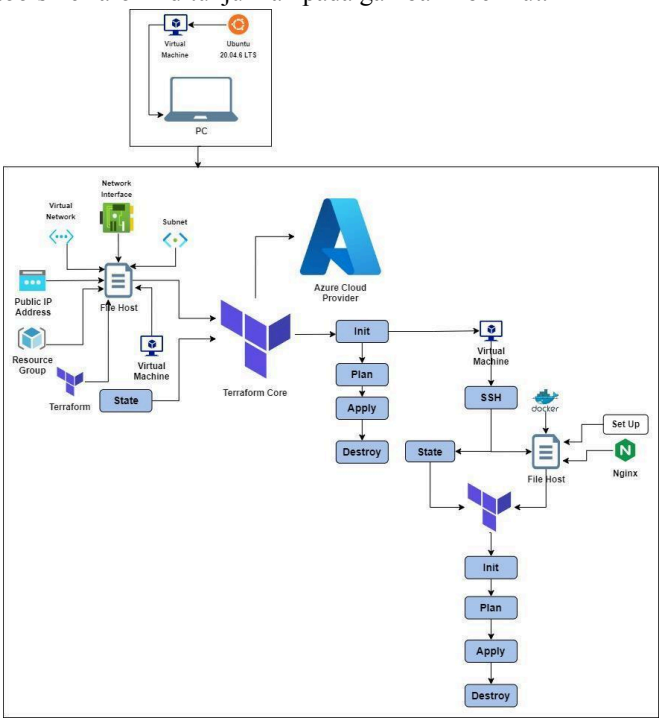
II. Metodologi Penelitian

A. *Rancangan Sistem*

Adapun rancangan sistem yang akan digunakan untuk melakukan penelitian ini ada dua rancangan sistem, yaitu :

1) *Rancangan sistem Terraform*

Adapun rancangan sistem yang akan digunakan untuk tools Terraform ditunjukkan pada gambar 1 berikut.



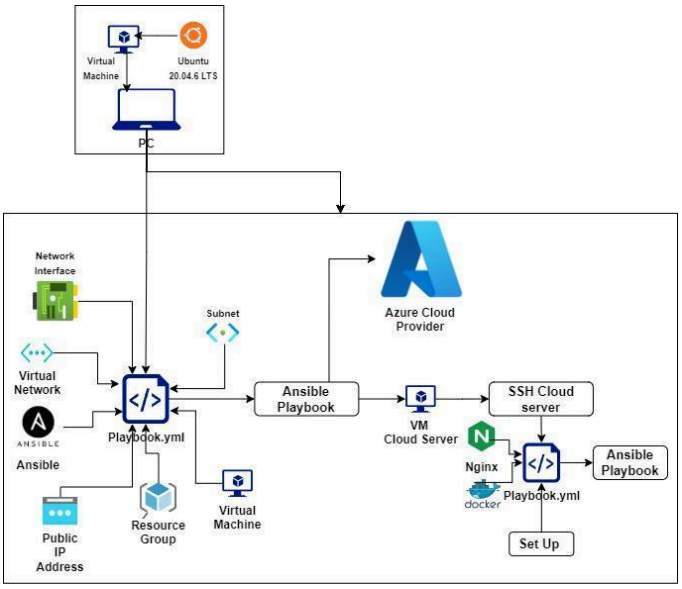
Gambar 1. Rancangan Sistem *Terraform*

Sistem ini mengimplementasikan otomatisasi manajemen *cloud server* menggunakan *Terraform* untuk mengelola infrastruktur di Microsoft Azure dari PC lokal dengan Ubuntu

20.04.6 LTS. *Terraform*, yang diinstal di PC lokal, menggunakan konfigurasi berformat *HashiCorp Configuration Language* (HCL) untuk mengotomatisasi pembuatan dan pengelolaan komponen infrastruktur seperti *virtual network, network interface, subnet, resource group, public* IP, dan *virtual machine*. Proses otomatisasi dimulai dengan inisialisasi (*init*), perencanaan (*plan*), dan penerapan (*apply*) konfigurasi untuk membangun infrastruktur di Azure. Setelah VM dibuat, *Terraform* menggunakan SSH untuk *remote access* dan otomatisasi konfigurasi lebih lanjut, termasuk penginstalan *Terraform* di dalam VM serta layanan seperti Nginx dan Docker. *File state* dikelola untuk melacak status infrastruktur, memastikan perubahan dapat diatur dengan aman. *Terraform* juga mendukung penghancuran (*destroy*) infrastruktur saat tidak lagi diperlukan, menghindari penggunaan sumber daya berlebihan. Siklus kerja *Terraform* (*init, plan, apply, state, destroy*) memastikan pengelolaan infrastruktur *cloud* yang otomatis, konsisten, dan mudah diulang, meningkatkan efisiensi dan mengurangi kesalahan manual.

2) Rancangan Sistem Ansible

Adapun rancangan sistem yang akan digunakan untuk *tools Ansible* ditunjukkan pada gambar 2 berikut ini.

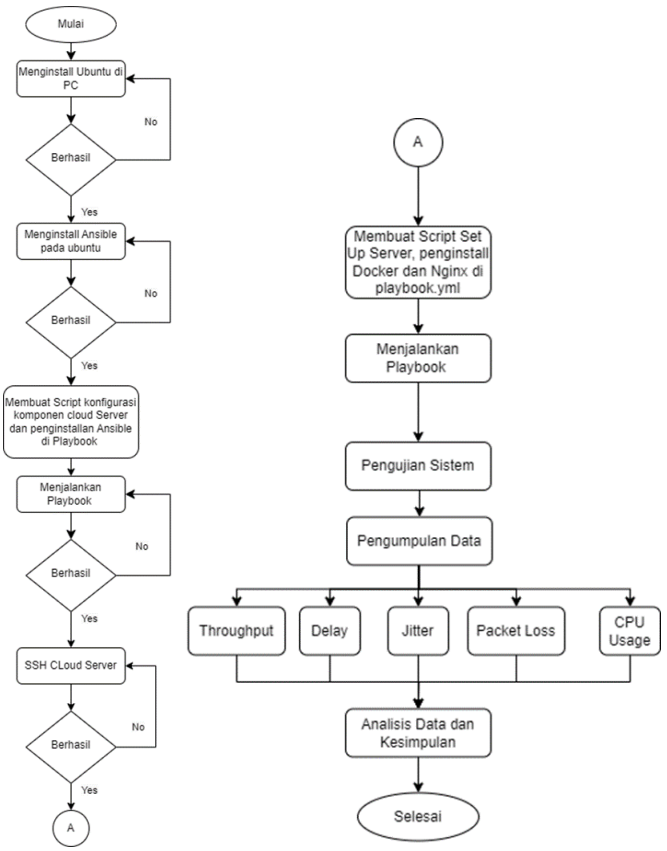


Gambar 2. Rancangan Sistem *Ansible*

Sistem ini mengimplementasikan otomatisasi manajemen *cloud server* menggunakan *Ansible* untuk mengelola infrastruktur di Microsoft Azure dari PC lokal dengan Ubuntu 20.04.6 LTS. *Ansible*, yang diinstal di PC lokal, menggunakan Playbook berformat YAML untuk mengotomatisasi pembuatan dan konfigurasi komponen infrastruktur seperti *virtual network, network interface, subnet, resource group, public* IP, dan *virtual machine* di Azure. Proses dimulai dengan menjalankan *Playbook Ansible* yang terhubung ke *server cloud* melalui SSH untuk konfigurasi lebih lanjut. *Ansible* juga mengotomatiskan penginstalan layanan seperti Nginx dan Docker di VM, memastikan pengaturan *server cloud* yang konsisten dan dapat diulang. Sistem ini meningkatkan efisiensi, mengurangi kesalahan manual, dan memastikan pengelolaan infrastruktur *cloud* yang terstruktur dengan *Ansible* dan Azure.

B. Alur Penelitian

Metode yang akan digunakan dalam penelitian ini adalah *Infrastructure as Code* (IaC). IaC merupakan suatu pendekatan dalam manajemen infrastruktur IT yang menggunakan otomasi melalui kode untuk mengelola dan menyediakan infrastruktur menggunakan representasi kode yang dapat dieksekusi. Metode ini bekerja seperti alur atau *flowchart* yang diuraikan pada gambar 3 berikut ini.



Gambar 3. Flow Chart *Ansible*

Langkah awal penelitian ini adalah menginstal Ubuntu dan *Ansible* di komputer pribadi. Setelah itu, dibuat skrip *Playbook* untuk otomatisasi pembangunan infrastruktur *cloud server*, mencakup konfigurasi mesin virtual, jaringan, dan penyimpanan. Selanjutnya, dilakukan *remote* ke *cloud server* untuk *set up server,* menginstal Docker dan Nginx melalui Playbook.yml, dan menjalankan *Playbook* tersebut. Pengujian dilakukan untuk mengukur kinerja sistem seperti *throughput*, *delay*, *jitter*, *packet loss*, dan penggunaan CPU. Data hasil pengujian dianalisis untuk menilai kinerja sistem, dan kesimpulan diambil berdasarkan analisis tersebut.

Adapun *flowchart* dari *terraform* sama saja dengan *flowchart ansible*, yang membedakannya adalah skrip *terraform* menggunakan *file host* sedangkan *ansible* menggunakan *playbook*.

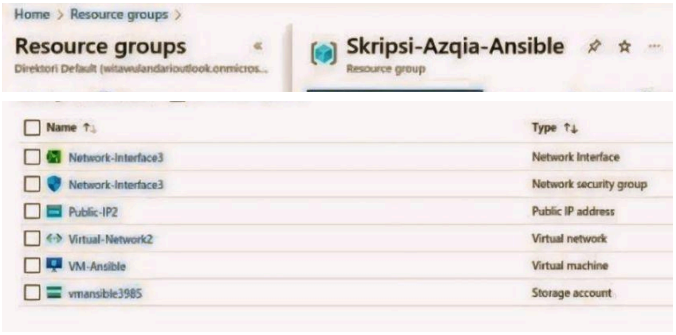
III. Hasil dan Pembahasan

A. *Hasil Infrastruktur*

Hasil dari infrastruktur yang dibuat dengan menggunakan *Ansible* dan *terraform* dapat dilihat pada gambar 4 dan gambar 5.

1) *Hasil Infrastruktur Ansible*

Hasil dari infrastruktur yang dibuat menggunakan Ansible dapat dilihat pada gambar 4 berikut.

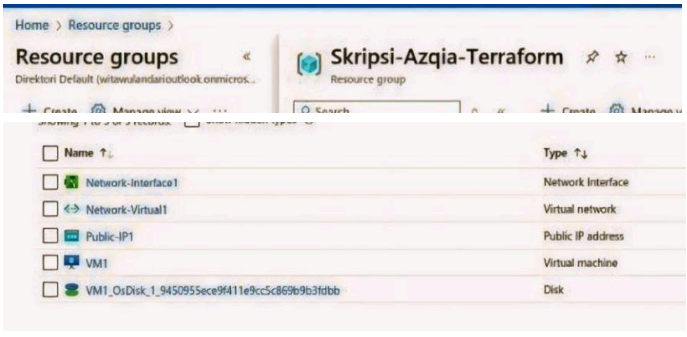


Gambar 4. Tampilan *Consol* Microsoft Azure untuk *Ansible*

Gambar 4 menunjukkan antarmuka portal Microsoft Azure dengan *resource group* bernama "Skripsi-Azqia-Ansible" di wilayah *Southeast Asia*. *Resource group* ini mencakup berbagai sumber daya seperti *network interface, Network Security Group, public IP, virtual network*, VM "VM_Ansible", dan *storage account* "vmAnsible896". *Resource group* ini memudahkan pengelolaan dan pengaturan infrastruktur Azure secara efisien, mendukung proyek skripsi yang menggunakan *Ansible* sebagai alat otomatisasi IT.

2) *Hasil Infrastruktur Terraform*

Hasil dari infrastruktur yang dibuat menggunakan terraform dapat dilihat pada gambar 5. Tampilan Consol Microsoft Azure untuk *Terraform*,

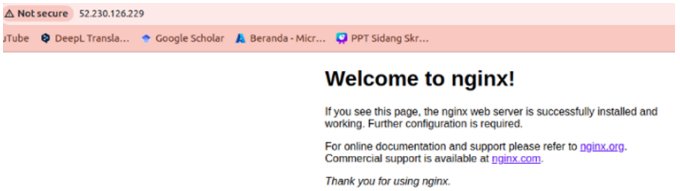


Gambar 5. Tampilan *Consol* Microsoft Azure untuk *Terraform*

Gambar 5 menunjukkan antarmuka portal Microsoft Azure dengan resource group "Skripsi-Azqia-Terraform" di wilayah Southeast Asia. Resource group ini berisi berbagai sumber daya seperti network interface, virtual network, public IP, VM "VM1", dan disk sistem operasi VM. Resource group ini mendukung pengelolaan infrastruktur yang dibuat dan diatur menggunakan Terraform, alat IaC untuk otomatisasi penyediaan dan pengelolaan sumber daya cloud, memudahkan manajemen infrastruktur dalam proyek atau skripsi berfokus pada Terraform.

B. *Hasil Manajemen Cloud Server*

Hasil dari manajemen cloud server dengan menjalankan container nginx menggunakan Ansible dan terraform dapat dilihat pada gambar 6 berikut.



Gambar 6. Tampilan Landing Page Web server dari Ansible dan Terraform

Landing page dari web server Nginx yang dibuat menggunakan Ansible dan Terraform berfungsi sebagai konfirmasi bahwa server telah berhasil diinstal dan berjalan dengan baik. Halaman ini menampilkan pesan "Welcome to nginx!" beserta informasi bahwa instalasi Nginx telah berhasil. Ansible dan Terraform digunakan untuk mengotomatisasi proses deployment ini, memastikan bahwa Nginx berjalan di dalam container dengan konfigurasi dasar yang siap untuk digunakan atau dimodifikasi lebih lanjut sesuai kebutuhan.

C. *Pengujian Efisiensi dari Penggunaan Ansible dengan Terraform*

Pengujian ini bertujuan untuk menganalisis penggunaan sumber daya sistem, terutama CPU, saat menjalankan tugas-tugas otomatisasi menggunakan Ansible dan Terraform. Data yang dikumpulkan akan memberikan gambaran tentang efisiensi dan performa sistem dalam mengelola beban kerja yang diberikan oleh kedua alat ini. Selain itu, pengujian ini juga mencatat waktu eksekusi masing-masing alat untuk memberikan wawasan lebih mendalam tentang seberapa cepat dan efisien Ansible dan Terraform dalam menyelesaikan tugas yang diberikan.

1) *Analisis Pengujian Waktu Eksekusi*

Dalam pengujian ini, kedua alat, Ansible dan Terraform, digunakan untuk melakukan tugas otomatisasi pada platform Azure. Hasil pengujian menunjukkan perbedaan waktu eksekusi antara keduanya, yang ditunjukkan dalam tabel I berikut:

TABEL I
WAKTU EKSEKUSI

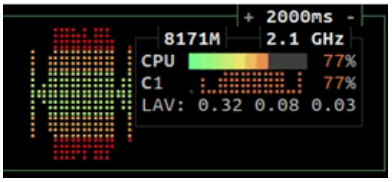
Waktu Eksekusi	
Ansible	**Terraform**
1 menit 34,158 detik	1 menit 25,974 detik

Tabel I menunjukkan perbedaan efisiensi waktu eksekusi antara *Ansible* dan *Terraform* dalam menyelesaikan tugas yang sama, seperti menginstal Docker, menjalankan kontainer Nginx, dan mengunggah file ke GitHub. *Ansible* membutuhkan 1 menit 34,158 detik, sementara *Terraform* hanya memerlukan 1 menit 25,974 detik meskipun menggunakan koneksi SSH untuk *server* jarak jauh.

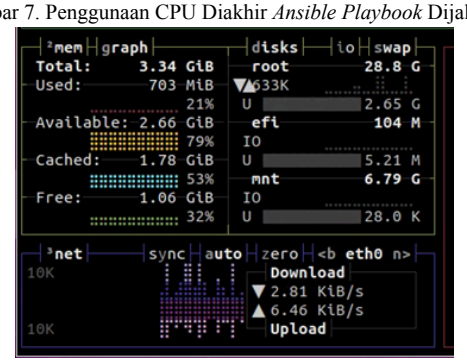
Hasil ini menunjukkan bahwa meskipun *Ansible* lebih cocok untuk tugas-tugas lokal dengan komunikasi jarak jauh yang minim, *Terraform* lebih efisien dalam mengelola banyak *server* atau infrastruktur di berbagai lokasi. Kemampuan *Terraform* untuk mengotomatisasi tugas secara *remote* menjadikannya lebih unggul dalam skenario *multi-server*.

2) Analisis Penggunaan CPU Ansible

Pengujian ini dilakukan dengan mengamati penggunaan CPU saat menjalankan tugas-tugas yang diberikan *Ansible*. Adapun Penggunaan CPU *Ansible* dapat dilihat pada gambar 7 dan gambar 8 berikut ini.



Gambar 7. Penggunaan CPU Diakhir *Ansible Playbook* Dijalankan

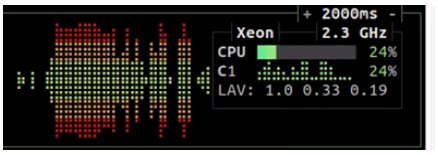


Gambar 8. Penggunaan Memori Diakhir *Ansible Playbook* Dijalankan

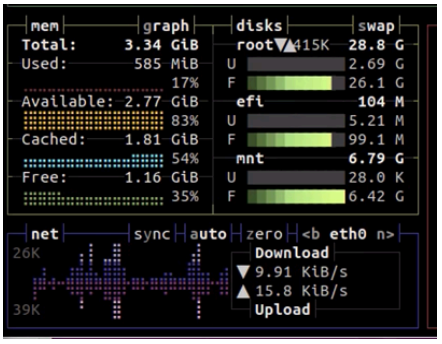
Gambar 7 dan gambar 8 menunjukkan penggunaan CPU pada 77%, dengan *ansible-Playbook* menggunakan 3.7%, python2 3.0%, dan python3 1.5%, menandakan beban CPU yang wajar. Memori digunakan sebesar 703 MiB dari total 3.34 GiB, dengan 1.06 GiB bebas. Penggunaan disk minimal, partisi *root* hanya memakai 233K dari 28.8G. Aktivitas jaringan ringan, dengan unduhan 2.81 KiB/s dan unggahan 6.46 KiB/s. Secara keseluruhan, sistem menangani tugas dengan efektif, menunjukkan penggunaan CPU, memori, dan disk yang efisien serta aktivitas jaringan yang ringan, mencerminkan kinerja sistem yang baik.

3) Analisis Penggunaan CPU Terraform

Pengujian ini dilakukan dengan mengamati penggunaan CPU saat menjalankan tugas-tugas yang diberikan *Terraform*. Adapun Penggunaan CPU Terraform dapat dilihat pada gambar 9 dan gambar 10 berikut ini.



Gambar 9. Penggunaan CPU Diakhir *Terraform* Dijalankan



Gambar 10. Penggunaan Memori Diakhir *Terraform* Dijalankan

Gambar 9 dan gambar 10 menunjukkan metrik sistem saat menjalankan skrip untuk mengatur Docker, Nginx, dan operasi git. Penggunaan CPU sebesar 24% dengan dockerd sebagai proses tertinggi (2.0%), menandakan beban CPU rendah. Memori digunakan 585 MiB dari total 3.34 GiB, dengan 1.16 GiB bebas. Penggunaan disk minimal, dengan partisi root hanya memakai 115K dari 28.8G. Aktivitas jaringan sedang, dengan unduhan 9.91 KiB/s dan unggahan 15.8 KiB/s. Proses terkait Docker, Nginx, dan git berjalan lancar tanpa tekanan sumber daya signifikan, menunjukkan kinerja sistem yang efisien dan efektif.

D. Pengujian Quality of Service Ansible dan Terraform

1) Analisis Throughput

Pada Pengujian ini, throughput dari *Ansible* dan *Terraform* diukur dan dibandingkan untuk memberikan Gambaran mengenai kinerja masing-masing *tools*. Berikut adalah hasil pengujian *throughput* untuk *Ansible* dan *Terraform* yang dapat dilihat pada tabel II berikut.

TABEL II
HASIL *TROUGHPUT*

Troughput	
Ansible	**Terraform**
76 Kbps	15000 Kbps

Tabel II menunjukkan perbedaan signifikan antara *Ansible* dan *Terraform* dalam kecepatan transfer data dan waktu eksekusi. *Ansible* memiliki *throughput* 76 Kbps dan waktu eksekusi 1 menit 34,158 detik (kategori "Buruk", indeks 1). *Terraform* unggul dengan *throughput* 15000 Kbps

dan waktu eksekusi 1 menit 25,974 detik (kategori "Sangat Bagus", indeks 4). Meskipun waktu eksekusi *Terraform* sedikit lebih cepat, *throughput* yang jauh lebih tinggi membuatnya lebih efisien untuk transfer data. *Terraform* lebih unggul dalam kecepatan transfer, sementara *Ansible* cocok untuk eksekusi lebih cepat dengan throughput lebih rendah.

2) Analisis Packet loss

Pada Pengujian ini, *packet loss* dari *Ansible* dan *Terraform* diukur dan dibandingkan untuk memberikan Gambaran mengenai kinerja masing-masing *tools*. Berikut adalah hasil pengujian *packet loss* untuk *Ansible* dan Terraform yang dapat dilihat pada tabel III berikut.

Tabel III
Hasil Packet Loss

Packet Loss	
Ansible	**Terraform**
0%	0%

Tabel III menunjukkan bahwa *Ansible* dan *Terraform* memiliki kinerja yang sangat baik dalam menjaga integritas data, dengan *packet loss* 0% pada keduanya, menandakan tidak ada paket yang hilang selama transmisi. Waktu eksekusi *Ansible* adalah 1 menit 34,158 detik, sementara *Terraform* lebih singkat dengan 1 menit 25,974 detik. Kedua alat ini masuk dalam kategori "Sangat Bagus" dalam hal keandalan data. Meskipun *Terraform* sedikit lebih cepat, keduanya sama-sama andal untuk keandalan data, menjadikannya pilihan tepat dalam konteks ini.

3) Analisis Delay

Pada Pengujian ini, *Delay* dari *Ansible* dan *Terraform* diukur dan dibandingkan untuk memberikan Gambaran mengenai kinerja masing-masing *tools*. Berikut adalah hasil pengujian *Delay* untuk *Ansible* dan *Terraform* yang dapat dilihat pada tabel IV berikut.

Tabel IV
Hasil *Delay*

	Delay	
	Ansible	**Terraform**
Total Delay	72.817 ms	96.765 ms
Rata-rata Delay	37 ms	4,3 ms

Tabel IV menunjukkan perbedaan *Delay* antara *Ansible* dan *Terraform*. *Ansible* memiliki Total *Delay* 72,817 ms dan rata-rata 37 ms, sedangkan *Terraform* memiliki Total *Delay* 96,765 ms dengan rata-rata 4,3 ms. Keduanya masuk kategori "Sangat Bagus" (*indeks* 4) dengan nilai Delay di bawah 150 ms. Meskipun Total *Delay Terraform* lebih tinggi, rata-rata *Delay* yang lebih rendah menunjukkan konsistensi dan kecepatan yang lebih baik dalam memproses data. Dengan waktu eksekusi lebih cepat (1 menit 25,974 detik), *Terraform* lebih unggul dalam efisiensi waktu dibandingkan *Ansible* (1 menit 34,158 detik). Keduanya sangat baik untuk waktu respon cepat, dengan *Terraform* sedikit lebih unggul dalam kecepatan rata-rata.

4) Analisis Jitter

Pada Pengujian ini, *Jitter* dari *Ansible* dan *Terraform* diukur dan dibandingkan untuk memberikan Gambaran mengenai kinerja masing-masing *tools*. Berikut adalah hasil pengujian *jitter* untuk *Ansible* dan *Terraform* yang dapat dilihat pada tabel V berikut.

Tabel V
Hasil *Jitter*

	Jitter	
	Ansible	**Terraform**
Total Jitter	129 ms	330 ms
Rata-rata Jitter	0,047 ms	0,014 ms

Tabel V menunjukkan *Ansible* memiliki rata-rata jitter 0,047 ms, sementara *Terraform* lebih rendah pada 0,014 ms. Keduanya masuk kategori "Sangat Bagus" (indeks 4) dengan rata-rata *jitter* di bawah 75 ms. Rata-rata *jitter* yang lebih rendah pada *Terraform* menandakan keunggulannya dalam menjaga kestabilan jaringan dan konsistensi pengiriman data. Meskipun *Ansible* memiliki *jitter* lebih tinggi, performanya tetap andal. *Terraform* unggul dalam kestabilan, namun *Ansible* juga merupakan pilihan yang sangat baik.

IV. Kesimpulan

Hasil analisis efisiensi dan *Quality of Service* (QoS) antara *Ansible* dan Terraform menunjukkan perbedaan kinerja yang signifikan dalam otomatisasi manajemen *cloud server*. *Terraform* lebih efisien dengan waktu eksekusi 1 menit 25,974 detik, sedikit lebih cepat dibandingkan *Ansible* yang memerlukan 1 menit 34,158 detik. Dalam penggunaan sumber daya, *Ansible* menunjukkan pola penggunaan CPU yang konsisten tinggi, mencapai 100% di awal dan stabil di 77% di akhir, sementara aktivitas jaringannya cenderung rendah dan stabil. Sebaliknya, *Terraform* mengalami lonjakan penggunaan CPU hingga 100% di pertengahan pengujian, namun tetap efisien dengan aktivitas jaringan yang meningkat dinamis, menunjukkan kemampuan untuk menangani tugas-tugas yang lebih berat dengan efisiensi yang baik dalam penggunaan memori dan disk. Dari sisi QoS, *Terraform* unggul dengan *throughput* sebesar 15000 Kbps, jauh lebih tinggi dibandingkan *Ansible* yang hanya 76 Kbps. Keduanya memiliki *packet loss* 0%, menandakan keandalan yang sangat baik dalam menjaga integritas data selama proses otomatisasi. Meskipun *Ansible* menunjukkan waktu eksekusi yang sedikit lebih lama, *Terraform* memiliki keunggulan dalam rata-rata *Delay* yang lebih rendah (4,3 ms) dan *jitter* yang lebih stabil (0,014 ms), yang mengindikasikan kestabilan jaringan dan konsistensi dalam pengiriman data yang lebih baik.

Secara keseluruhan, *Terraform* menonjol dalam efisiensi penggunaan sumber daya, kecepatan transfer data, serta kestabilan dan konsistensi jaringan, menjadikannya pilihan yang lebih baik untuk skenario yang membutuhkan transfer data cepat dan manajemen jaringan yang stabil. *Ansible*,

meskipun memiliki penggunaan CPU yang lebih stabil, menunjukkan aktivitas jaringan yang lebih rendah dan performa yang kurang efisien dalam throughput dibandingkan *Terraform*. Kedua alat ini tetap andal dalam menjaga integritas data, namun keunggulan *Terraform* dalam waktu eksekusi yang lebih cepat dan kestabilan jaringan memberikan keuntungan tambahan dalam efisiensi operasional. Pemilihan antara *Ansible* dan *Terraform* harus didasarkan pada kebutuhan spesifik, apakah lebih mengutamakan efisiensi dan kecepatan transfer data dengan *Terraform*, atau kestabilan penggunaan CPU dan jaringan yang lebih konsisten dengan *Ansible*.

Referensi

[1] Harimurti, Y., & Udariansyah, D. (2023). Iplementasi Service EC2 & S3 Amazon Web Service Pada Niche Blog Menggunakan Metode SDLC. KLIK: Kajian Ilmiah Informatika dan Komputer, 4(2), 675–685. https://www.djournals.com/klik/article/view/1192.

[2] Hidayat, Y., & Arifwidodo, B. (2021). Implementasi Web Server Menggunakan Infrastructure as Code Terraform Berbasis Cloud Computing. FORMAT: Jurnal Ilmiah Teknik Informatika, 10(2), 192–201. https://doi.org/10.22441/format.2021.v10.i2.010.

[3] Asterina, D., Munandi, R., & Mayasari, R. (2015). Implementasi Dan Analisis Metode Failover Pada Sistem Redundant Dedicated Server Dan Cloud Server Untuk Layanan Voip. e-Proceeding of Engineering, 2(2), 3137. https://openlibraryPublications.telkomuniversity.ac.id/index.php/engineering/article/view/471.

[4] Alamsyah, N., & Febrianto, N. I. (2021). Analysis of The Utilization and Implementation of Cloud Computing Infrastructure Services on The Azure Microsoft Platform. Jurnal Mantik, 5(1), 127–136. http://iocscience.org/ejournal/index.php/mantik/article/view/1287.

[5] Afdhal. (2013). Studi Perbandingan Layanan Cloud Computing. Jurnal Rekayasa Elektrika, 10(4), 193-201. https://jurnal.usk.ac.id/JRE/article/view/1110.

[6] Lubis, M. D. S., Hasannudin, D., Efendi, J., Wiljono, L., & Sufiani, M. (2020). Membangun Router Pada Jaringan Komputer Menggunakan Ubuntu OS. Jurnal Teknik Informatika Kaputama (JTIK), 4(2), 111–125. https://garuda.kemdikbud.go.id/documents/detail/1743061.

[7] Budijono, S., & Saleh, R. (2014). Penggunaan Virtual machine dalam Pembelajaran Jaringan. ComTech: Computer, Mathematics and Engineering Applications, 5(1), 495. https://doi.org/10.21512/comtech.v5i1.2643.

[8] Fernanto, A. T., Suryani, V., & Ariyanto, E. (2021). Implementasi Sistem Monitoring CPU Usage dan Temperature pada Processor berbasis Internet of Things menggunakan Windows Management Instrumentation. e-Proceeding of Engineering, 8(2), 3111. https://openlibraryPublications.telkomuniversity.ac.id/index.php/engineering/article/view/14706/0.

[9] Ramirez, M. (2023). How to Install Bpytop Resource Monitoring Tool on AlmaLinux. https://www.liquidweb.com/blog/install-bpytop-resource-monitoring-tool-almalinux/.

[10] Wardhana, A. N. W., Yamin, M., & Aksara, L. F. (2017). Analisis Quality of Service (QoS) Jaringan Internet Berbasis Wireless LAN pada Layanan Indihome. Semantik, 3(2), 49–58. https://www.academia.edu/download/76273237/2431.pdf.

[11] Sutarti, Siswanto, & Subandi, A. (2018). Implementasi Dan Analisis QoS (Quality of Service) Pada VoIP (Voice Over Internet Protocol) Berbasis Linux. Jurnal PROSISKO, 5(2), 92–101. https://e-jurnal.lppmunsera.org/index.php/PROSISKO/article/view/786

[12] Raj. (2020). Beginners Guide to TShark (Part 1). https://www.hackingarticles.in/beginners-guide-to-tshark-part-1/.

[13] Putra, W. R. A., Nurwa, A. R. A., Priambodo, D. F., & Hasbi, M. (2022). Infrastructure as Code for Security Automation and Network Infrastructure Monitoring. Matrik: Jurnal Manajemen, Teknik Informatika, dan Rekayasa Komputer, 22(1), 203–217. https://doi.org/10.30812/matrik.v22i1.2471.

[14] Syah, I., Muhammad, A. H., & Gunawan, E. (2020). Simulasi Network Automation Menggunakan Ansible Di GNS3 (Studi Kasus Smile Project). J-TIFA: Jurnal Teknologi Informatika, 3(2), 1–8. https://jurnal.ummu.ac.id/index.php/J-TIFA/article/view/1065.

[15] Gustian, D., Fitrisia, Y., Novayani, W., Purwantoro E.S.G.S, S., & Novayani, W. (2023). Implementasi Automation Deployment pada Google Cloud Compute VM menggunakan Terraform. INOVTEK Polbeng - Seri Informatika, 8(2), 50–62. http://103.174.114.133/index.php/ISI/article/view/3095.