Implementasi Web Server Menggunakan *Infrastructure As Code Terraform* Berbasis *Cloud Computing*

Yusuf Hidayat[1], Bongga Arifwidodo [2]
Program Studi S1 Teknik Telekomunikasi, Institut Teknologi Telkom Purwokerto[1,2]
Jl. D.I Panjaitan No. 128, Purwokerto 53147, Indonesia [1,2]
17101203@ittelkom-pwt.ac.id [1], Bongga@ittelkom-pwt.ac.id[2],

Abstract - Damage that usually occurs suddenly and cannot be detected immediately by car owners because of the lack of knowledge about car engines makes car users confused and panicked and unable to handle their car for a while before the damaged car is taken to the nearest repair shop. Therefore an application is needed, namely an expert system which is a knowledge-based computer program from an expert that can help reduce the risk of car damage by knowing where the damage is known and then looking for the cause. So that through the application, this expert system can simplify, assist and find out the causes of the damage that occurs in the Toyota car cooling system.
Keywords - Expert System, Backward Chaining, Toyota Car Cooling System.

Abstrak - Kemajuan teknologi otomatisasi menjadikan suatu kemudahan untuk pembangunan infrastruktur web server, tanpa harus melakukan perintah pembuatan secara berulang ulang, hanya dengan serangkaian kode yang telah didefinisikan menggunakan Infrastructure as code Terraform maka infrastruktur web server berhasil dibangun. Namun, pertanyaan mulai muncul terhadap kualitas nya pada instance web server yang dibuat secara otomatis akankah sesuai standar, hal ini menjadi pertanyaan para penyedia layanan web server. Pada parameter yang dilihat pada penelitian ini yaitu perbandingan terhadap waktu yang dibutuhkan dalam pembuatan instance web server dari kedua metode tersebut dan perbandingan kualitas web server dengan melakukan pengujian beban request terhadap web server. Hasilnya disimpulkan bahwa Infrastructure As Code Terraform layak untuk dijadikan tools automasi sebagai penyediaan infrastruktur web server otomatis pada pembangunan web server beserta Virtual Machine dengan selisih nilai rata-rata perbandingan dari kedua metode tersebut didapatkan untuk waktu yang dibutuhkan dalam pembuatan Instance web server yang menggunakan Infrastructure As Code Terraform lebih unggul dengan selisih waktu 12 Menit 19 Detik, perbandingan seluruh rata – rata nilai pengujian Quality Of Service untuk instance web server manual unggul tipis dengan selisih nilai throughput 0.750 Mbit/s, packetloss 0.00%, nilai delay 0,0035 ms, nilai jitter 0,004 ms. Parameter CPU usage dengan selisih nilai rata – rata diseluruh pengujian yaitu 5,899%.
Kata Kunci : *web server, virtual machine, Infrastructure as code Terraform, Quality Of Service*

I. PENDAHULUAN

Ilmu pengetahuan dalam bidang teknologi internet kini telah berkembang, salah satu perkembangan yang terjadi adalah teknologi cloud computing. Pada cloud computing terdapat model pengembangan public cloud dan private cloud. Public cloud memberikan kemudahan pada pengguna, untuk mengeluarkan biaya sesuai dengan sumber daya yang digunakan serta terbebas dari biaya perancangan infrastruktur, namun terdapat permasalahan seperti keamanan data menjadi salah satu poin yang diperhitungkan dalam menggunakan public cloud. Penggunaan private cloud merupakan salah satu solusi dalam menjaga keamanan data. Sehingga, private cloud memungkinkan para pengguna dapat melacak permasalahan dalam menjalankan sistem [1].

Pasar IT saat ini semakin didominasi oleh "kebutuhan akan kecepatan". Kebutuhan ini terlihat dalam penggunaan yang mempersingkat siklus pengembangan perangkat lunak dan juga kegiatan pengembangan perangkat lunak. Tren penggunaan taktik rekayasa perangkat lunak ini yang mengurangi ruang, waktu, dan upaya antara pengembangan dan operasi perangkat lunak serta jarak teknis dan organisasi antara kedua jenis tim perangkat lunak ini dikenal sebagai DevOps. Sebagai bagian dari menu DevOps, banyak praktik memerlukan penggunaan kembali alat standar dari pengembangan perangkat lunak (misalnya, penerapan kode, manajemen coderevision, dll.) untuk mengelola apa yang dikenal sebagai Infrastructure as code. Dengan mempromosikan pengelolaan pengetahuan dan pengalaman sejumlah besar subsistem sebagai sumber yang umum tersedia untuk administrator sistem [2].

Microsoft Azure merupakan layanan public cloud yang mempunyai data center Microsoft yang tersebar luas di seluruh dunia. Pada pengembangan cloud computing, pengembang dapat memilih data center yang terdekat sehingga tingkat konektifitas menjadi lebih tinggi [3]. Openstack salah satu layanan private cloud yang mengendalikan proses komputasi dan sumber daya jaringan dalam sebuah data center melalui dashboard untuk melakukan kontrol terhadap administrasi dan memberikan hak akses pada pengguna melalui antarmuka web [4].

Penelitian Ramandeep Singh, Dr. Ravindra Kumar Purwar pada tahun 2019 yang berjudul “Cloud Automation with Configuration Management using CHEF Tool”, mengusulkan implementasi Web server dengan menggunakan layanan Automation pada CHEF Tool Penyebaran perangkat lunak otomatis dan efisien sangat penting untuk penyedia hosting cloud modern saat ini. penting bagi server perusahaan untuk menginstal, mengonfigurasi, dan berjalan secepat

mungkin dan sekonsisten mungkin untuk membantu mengurangi biaya. Instalasi manual, updation dan konfigurasi paket membutuhkan banyak tenaga kerja dan waktu. Penelitian ini telah membandingkan instalasi manual dengan alat otomatisasi untuk menyediakan penyebaran cloud yang akurat. Alat otomatisasi mengurangi waktu yang diperlukan untuk menyelesaikan tugas yang disebutkan dari beberapa jam hingga beberapa menit [5].

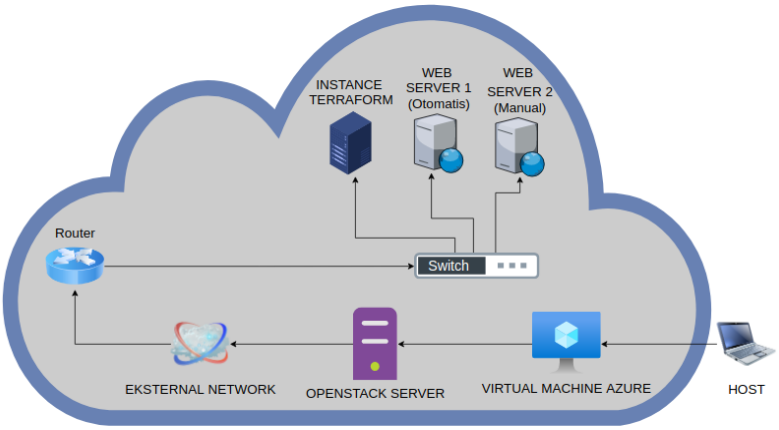
Penelitian Leonardo Rebouc as de Carvalho dan Aleteia Patricia Favacho de Araujo pada tahun 2020 yang berjudul "Performance Comparison of Terraform and Cloudify as Multicloud Orchestrators" mengusulkan perbandingan model cloud computing dengan menggunakan layanan Automation Cloudify dan Terraform. Cloudify tidak hanya menyelesaikan tugas yang mengkonsumsi lebih banyak sumber daya lingkungan host, tetapi juga membutuhkan waktu lebih lama untuk memberikan hasilnya daripada Terraform. Selain itu, Cloudify tidak mendukung beberapa layanan cloud yang sudah banyak digunakan di pasar. Selama eksperimen pekerjaan ini, Terraform menyajikan tingkat kedewasaan yang mengejutkan, terutama yang berkaitan dengan penanganan kesalahan. Ditambah dengan ini, serangkaian fitur dan plugin yang didokumentasikan dengan benar meningkatkan tumpukan teknologi alat yang menjanjikan ini. memberi Terraform keterampilan yang diperlukan untuk bertindak efisien, yang dibuktikan dengan hasil eksperimen pada penelitian [6].

Teknologi otomatisasi menjadikan suatu kemudahan untuk pembangunan infrastruktur cloud computing, tanpa harus melakukan pembuatan secara berulang ulang ketika ingin membuat lebih dari 1 Instance Web server, hanya dengan serangkaian kode yang telah didefinisikan menggunakan Infrastructure as code Terraform. Namun beberapa pertanyaan muncul, mulai dari kualitas nya pada instance Web server yang dibuat secara otomatis akankah bagus, hal ini menjadi sebuah pertanyaan para penyedia layanan Web server.

II. METODE PENELITIAN

2.1 Topologi Jaringan

Topologi jaringan yang digunakan pada penelitian, ditunjukkan pada Gambar 1 yang terdiri dari 1 *Host* untuk mengakses *Server*, 1 *Virtual Machine Azure* untuk membangun *Openstack* dengan metode Devstack, kemudian didalam *openstack* akan dibuat 3 buah *instance* dengan membangun 1 buah *instance* pada openstack terlebih dahulu untuk menginstal *Infrastructure as code Terraform* yang akan digunakan untuk membangun infrastruktur 1 *instance Web server* secara otomatis dengan kode perintah dari *Infrastructure as code Terraform* didalam *Openstack* dan 1 *instance Web server* yang dibuat secara manual dengan menggunakan *dashboard* pada openstack untuk dilakukannya perbandingan kualitas dari *instance Web server* yang dibuat secara otomatis dengan menggunakan *Infrastructure as code Terraform* dan secara manual dengan menggunakan *dashboard* pada openstack, terdapat juga 1 *external network* yang terhubung dengan O*penstack* berfungsi untuk menghubungkan seluruh *virtual machine* yang ada pada *cloud environment* ke jaringan luar.

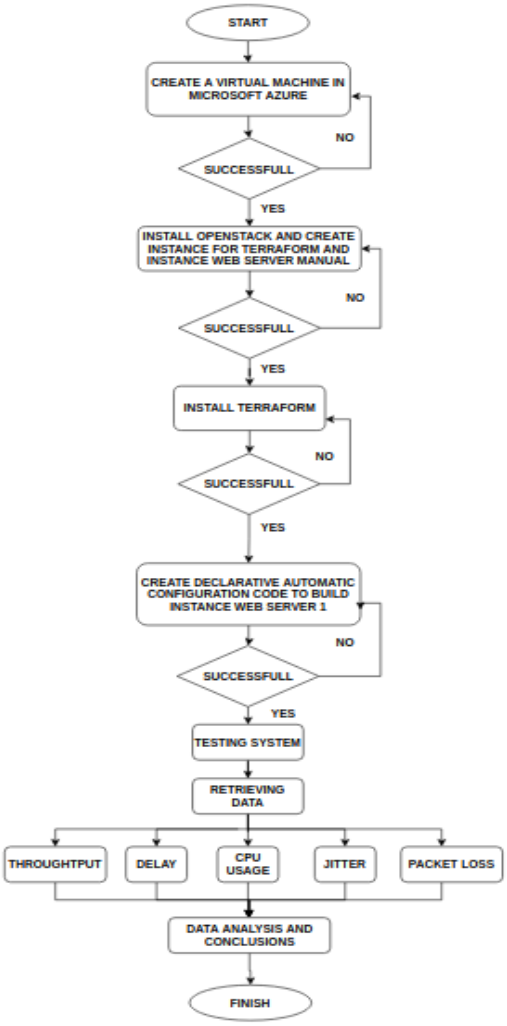


Gambar 1. Topologi Jaringan

2.2 Alur Penelitian

Dalam Penelitian yang dilakukan mempunyai beberapa tahapan diagram alur yang ditunjukkan pada Gambar 2. dimulai dari tahapan membuat *Virtual Machine* di *Microsoft Azure*, kemudian tahapan instalasi infrastruktur menggunakan Openstack dan membuat 1 buah *instance* khusus untuk berjalan nya Terraform, dilanjutkan dengan tahapan instalasi *Software Infrastructure As Code* dan membuat 1 *Web server* secara otomatis dengan serangkaian kode yang sudah didefinisikan,kemudian 1 *instance Web server* yang dibuat secara manual dengan menggunakan *dashboard* pada openstack untuk dilakukannya perbandingan kualitas dari *instance Web server* yang dibuat secara otomatis dengan menggunakan *Infrastructure as code Terraform* dan secara manual dengan menggunakan *dashboard* pada openstack,

lalu masuk pada tahapan sistem pengujian *Web server* yang telah dibangun dan terakhir yaitu menganalisa jaringan dengan pengambilan data berupa waktu yang dibutuhkan dalam pembangunan infrastruktur layanan *web server* parameter *Quality Of Service* (QoS) yaitu *Throughput, packet loss, jitter, delay* dengan standar TIPHON dan *CPU Usage.*



Gambar 2. Diagram Alur Penelitian

2.3 *Quality Of Service* (QoS)

Quality of Service (QoS) adalah kemampuan suatu jaringan untuk menyediakan layanan yang baik. *Quality of Service* mengacu pada kemampuan jaringan untuk menyediakan layanan yang lebih baik pada trafik jaringan tertentu melalui teknologi yang berbeda-beda. Tujuan dari *Quality of Service* adalah untuk memenuhi kebutuhan-kebutuhan layanan yang berbeda, dengan menggunakan infrastruktur yang sama [7].

a. *Throughput*
Troughput merupakan jumlah total kedatangan paket yang sukses yang diamati pada destination selama interval waktu tertentu dibagi oleh durasi *interval* waktu tersebut [7].

Tabel 1. Kategori Nilai *Throughput*

Kategori *Throughput*	***Throughput (Mbit/s)***	***Indeks***
Sangat Bagus	< 2,1 Mbps	4
Bagus	1200 Kbps – 2,1 Mbps	3
Sedang	700 – 1200 Kbps	2
Buruk	338 – 700 Kbps	1

Rumus perhitungan *Throughput* :

$$Throughput = \frac{Jumlah\ Data\ yang\ Dikirim\ (kb)}{Waktu\ Pengiriman\ Data\ (s)} \quad (1)$$

b. *Packet loss*

Packet loss merupakan suatu parameter yang menggambarkan suatu kondisi yang menunjukkan jumlah total paket yang hilang [7].

Tabel 2. Kategori Nilai *Packet loss* [8].

Kategori *Packet loss*	***Packet loss (%)***	***Indeks***
Sangat Bagus	0%	4
Bagus	3%	3
Sedang	15%	2
Buruk	25%	1

$$Packet\ Loss = \frac{(Paket\ Data\ Dikirim - Paket\ Data\ Diterima)}{Paket\ Data\ yang\ Dikirim} x100\% \quad (2)$$

c. *Delay*

Delay adalah waktu yang dibutuhkan data untuk menempuh jarak dari asal ke tujuan. *Delay* dapat dipengaruhi oleh jarak, media fisik, kongesti atau juga waktu proses yang lama [7].

Tabel 3. Kategori Nilai *Delay* [8].

Kategori *Delay*	***Delay (ms)***	***Indeks***
Sangat Bagus	< 150 ms	4
Bagus	150 s/d 300 ms	3
Sedang	300 s/d 450 ms	2
Buruk	> 450 ms	1

$$Delay\ (s) = \frac{Total\ Delay}{Total\ Paket\ yang\ Diterima} \quad (3)$$

d. *Jitter*

Jitter diakibatkan oleh variasi-variasi dalam panjang antrian, dalam waktu pengolahan data, dan juga dalam waktu penghimpunan ulang paket-paket di akhir perjalanan jitter, berhubungan dekat dengan *latency*, yang menunjukkan banyaknya variasi *delay* pada transmisi data di jaringan [7].

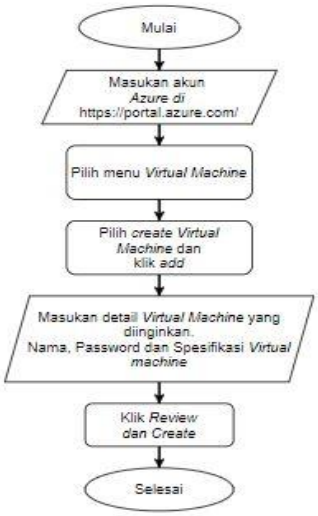
Tabel 4. Kategori Nilai *Jitter* [8].

Kategori *Jitter*	***Jitter (ms)***	***Indeks***
Sangat Bagus	0 ms	4
Bagus	0 s/d 75 ms	3
Sedang	76 s/d 125 ms	2
Buruk	125 s/d 225 ms	1

$$Jitter = \frac{Total\ Variasi\ Delay}{Total\ Paket\ Yang\ Diterima} \quad (4)$$

2.4 Membuat *Virtual Machine* Azure

Virtual Machine di *Microsoft Azure* merupakan salah satu layanan yang diberikan oleh *Microsoft Azure* yang bertujuan untuk memberikan sumber daya yang dibutuhkan dalam komputasi sesuai dengan permintaan yang diinginkan oleh pengguna layanan. Pada penelitian yang dibangun, untuk perangkat komputasi yang dibutuhkan yaitu *Standard A4 v2* dengan spesifikasi 4vcpus dan 8GiB Memory.



Gambar 3. Diagram alur pembuatan *Virtual Machine Azure*

2.5 Instalasi dan konfigurasi *Openstack*

Instalasi infrastruktur *Openstack* berada diatas V*irtual Machine Azure*, metode yang dijalankan yaitu menggunakan *script* dari *Devstack*, metode ini dilakukan agar lebih mudah dan cepat dalam instalasi. Pada penelitian, D*evstack* berjalan pada *node Openstack server* dengan sistem operasi *Ubuntu Server* 18.04. Masuk sebagai user stack dan install git untuk *Clone* Devstack *deployment code* dilakukan nya *clone* dari opendev kemudian masuk ke direktori devstack, openstack akan di instalasi di direktori devstack namun perlu untuk memilih versi stable dari openstack yaitu ussuri agar tidak masuk ke stable victoria yang masih dalam kondisi baru dan belum stabil, dilakukan nya git branch dan membuat konfigurasi utama untuk openstack di *file local.conf* , kemudian dilakukan nya *clone* dari opendev dengan perintah ./stack.sh.

```
$ sudo su - stack
$ sudo apt -y install git
$ git clone https://opendev.org/openstack/devstack
$ cd devstack
$ git checkout stable/ussuri
$ git branch
$ sudo nano local.conf
$ ./stack.sh
```

Gambar 4. Instalasi Openstack

konfigurasi Openstack dilakukan sebagai persiapan infrastruktur yang akan dijadikan sebagai cloud *environment* untuk menerapkan sistem, Konfigurasi utama pada *file local.conf* bertujuan untuk menentukan *username* dan *password* serta *host ip* untuk mengakses *dashboard* Openstack. Konfigurasi dibuat pada *file local.conf* dan direktori */home/devstack dari node Openstack server*.

```
[[local|localrc]]

# Password untuk KeyStone, Database, RabbitMQ dan Service
ADMIN_PASSWORD=Yusufhidayat29
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD

# Host IP - IP VM yang telah dikonfigurasi sebelumnya (cek : ifconfig)
HOST_IP=10.0.0.4
```

Gambar 5. File isi pada *local.conf*

1.6 Instalasi *Infrastructure As Code Terraform*

Infrastructure As Code Terraform merupakan *Software* automatisasi yang digunakan untuk membangun infrastruktur *Web server* sesuai dengan keinginan pengguna tanpa harus mengkonfigurasi infrastruktur secara manual. Penyedia menentukan sumber daya yang tersedia dan medapatkan permintaan dari pengguna, *Infrastructure As Code Terraform* dapat digunakan oleh berbagai *platform cloud computing.*

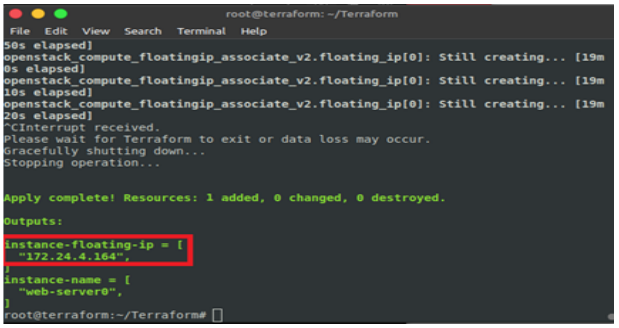
```
ubuntu@terraform:~$ sudo su
root@terraform:/home/ubuntu# cd ~
root@terraform:~# apt-get update
root@terraform:~# apt-get install unzip
root@terraform:~# wget
https://releases.hashicorp.com/terraform/0.
12.24/terraform_0.12.24_linux_amd64.zip
root@terraform:~# unzip
terraform_0.12.24_linux_amd64.zip
root@terraform:~# ls
root@terraform:~#
terraform terraform_0.12.24_linux_amd64
root@terraform:~# mv terraform
/usr/local/bin/
root@terraform:~# terraform version
Terraform v0.12.24
```

Gambar 6. Perintah Instalasi Terraform

Infrastruktur yang dibangun dalam penelitian ini berada pada *cloud* Openstack, kemudian membangun 1 *Instance Manual* dengan menggunakan *Dashboard* pada Openstack dan dari *instance* yang telah dibuat secara *manual*, akan dilakukan nya instalasi *software* didalamnya yaitu *Terraform* untuk membangun infrastruktur 1 *instance Web server* secara otomatis yang dapat dikelola dalam bentuk file menggunakan bahasa konfigurasi deklaratif *Infrastructure As Code* yang sudah didefinisikan sesuai dengan penelitian yang dilakukan.

1.7 Membuat *Simple code Web server* otomatis

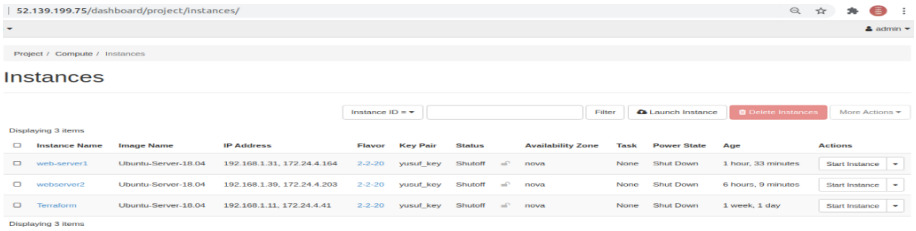
Terraform pada penelitian ini digunakan untuk membangun Infrastruktur dengan menggunakan *Infrastructure As Code* yang sudah didefinisikan sesuai dengan penelitian. Pada pembuatan *Instance Web server* yang dibutuhkan yaitu *file provider.tf* pada direktori */root/Terraform/* dibuat bertujuan untuk memperkenalkan *provider cloud* yang digunakan yaitu openstack yang diperkenalkan dengan Terraform agar dapat terhubung dan instalasi *Web server* beserta *instance* dengan cara otomatis menggunakan *Infrastructure As Code* berjalan dengan baik dan pada *file instance.tf* pada direktori */root/Terraform/* dibuat bertujuan untuk membangun 1 *Web server* beserta *instance* dengan cara otomatis menggunakan *Infrastructure As Code* yang sudah didefinisikan sebelumnya



Gambar 7. Hasil perintah code *Terraform*

1.8 Membuat *Instance Manual*

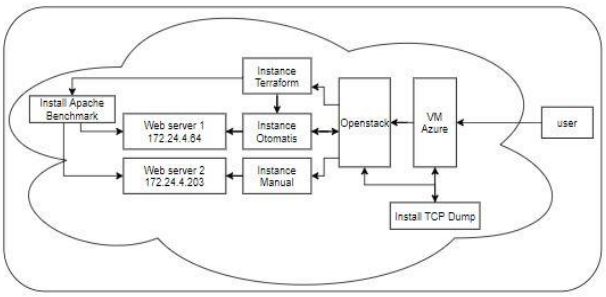
Instance adalah mesin virtual yang berjalan di dalam *cloud*, pada penelitian ini untuk *cloud* yang digunakan adalah Openstack yang dibuat diatas *Virtual Machine* Azure dan pada pembuatan *Instance* secara *manual* dengan menggunakan *dashboard* pada openstack dibutuhkan banyak tahapan secara manual, seperti memberikan nama *instance*, memilih *Boot Source, Flavor, Network, Security Group, Key Pair Instance* dan instalasi Apache2 yang dijadikan sebagai *Web server*.



Gambar 8. Instance *pada Openstack*

1.9 Skenario Pengujian

Pengujian pada penelitian ini bertujuan untuk mengetahui perbandingan terhadap waktu yang dibutuhkan dalam pembuatan *instance* dan *Web server* dari kedua metode tersebut dan analisis performa dari kualitas *Quality Of Service* (QoS) layanan sistem dari *Web server* yang dibuat secara otomatis dengan menggunakan *Infrastructure As Code Terraform* dan *Web server* yang dibuat secara manual menggunakan *dashboard* pada openstack dan konfigurasi nya menggunakan perintah *Command Line Interface* (CLI). Peneliti akan melakukan pengujian sebanyak 2 skenario, yaitu 1000, dan 5000. Pada masing – masing skenario diberikan nilai concurrency sebanyak 50 koneksi dan dilakukan pengujian sebanyak 15 kali sehingga didapatkan hasil rata – rata pada setiap parameter QoS.



Gambar 9. Diagram Alur Pengujian

Pada uji coba beban yang dilakukan, akan menggunakan *software Apache benchmark* yang berada diatas *instance* Terraform dan untuk *capture* hasil jaringan dari *Web server* diambil menggunakan Tcpdump yang dapat berjalan di *Command Line Interface* (CLI) dan untuk hasil nya akan dikirimkan dari server ke komputer *local*, kemudian wireshark akan digunakan untuk menampilkan hasil dari *capture* paket tersebut. Agar data yang dibuat mendapatkan hasil yang sesuai dan nyata maka akan dilakukan sebanyak 15 kali pengujian pada tiap skenario pengujian dengan paramater *Quality Of Service* (QoS) *Throughput, packet loss, jitter, delay* standar TIPHON dan juga *CPU Usage.* Pengambilan kesimpulan dilakukan dengan melakukan perbandingan kedua *Web server* dan memperhatikan tujuan dari penelitan agar diperoleh hasil yang sesuai.

III. HASIL DAN PEMBAHASAN

3.1 Perbandingan Waktu Pembuatan

Perbandingan dilakukan menggunakan *Virtual Machine Azure* dengan spesifikasi *Standard A4 v2* yaitu 4vcpus dan 8GiB Memory. Pada *Instance* w*eb server* yang dibuat secara otomatis untuk pembuatan 1 *Instance* yang didalam nya terdapat w*eb server* mendapatkan waktu yaitu 19 Menit 20 Detik dan untuk pembuatan 1 *Instance* yang didalam nya terdapat w*eb server* yang dibuat secara manual mendapatkan waktu 31 Menit 1 Detik. Maka, untuk *Instance* w*eb server* yang dibuat secara otomatis menggunakan *Infrastructur as code Terraform* lebih unggul jauh dan lebih efisien terhadap waktu dalam pembuatan *instance* beserta *web server* dengan selisih waktu sebesar 12 Menit 19 Detik.

Tabel 5. Perbandingan Waktu Pembuatan *Instance Web Server*

TIME		
Instance Web server 1 **(Otomatis)**	***Instance Web server 2*** **(Manual)**	**Hasil Perbandingan**
19 Menit 20 Detik	31 Menit 1 Detik	12 Menit 19 Detik

3.2 Analisis *Throughput*

Pada penelitian yang dilakukan, untuk jumlah *Request* 1000 pengujian yang dilakukan sebanyak 15 kali didapatkan hasil rata – rata nilai *throughput* untuk *Instance Web server 1* (Otomatis) sebesar 16.564 Mbit/s dan hasil rata – rata nilai *throughput* pada *Instance Web server 2* (Manual) didapatkan nilai sebesar 17.546 Mbit/s. Pada jumlah *Request* 5000 pengujiannya dilakukan sebanyak 15 kali didapatkan hasil rata – rata nilai *throughput* untuk *Instance Web server 1* (Otomatis) sebesar 41.458 Mbit/s dan hasil rata – rata nilai *throughput* pada *Instance Web server 2* (Manual) didapatkan nilai sebesar 41.977 Mbit/s. Jika merujuk pada Tabel 1 maka semua metode dari beberapa skenario pengujian memiliki parameter throughput kategori sangat bagus. Kemudian berdasarkan rata – rata nilai throughput yang didapatkan, untuk *Instance Web server 1* (Otomatis) dan *Instance Web server 2* (Manual) mendapatkan rata – rata nilai throughput dengan perbandingan nya sangat sedikit, hal ini disebabkan karena spesifkasi antara kedua web server tersebut sama yaitu menggunakan CPU 2 core dengan 2 GB RAM dan untuk hasil dari penggunaan *Infrastructur as code* terraform sesuai dengan standard dan tidak mempengaruhi dari kualitas layanan web server yang dibuat.

Tabel 6. Nilai Rata – Rata Pengujian *Throughput*

THROUGHPUT		
Jumlah *Request*	***Instance Web server 1*** **(Otomatis)**	***Instance Web server 2*** **(Manual)**
1000	16.564	17.546
5000	41.458	41.977

3.3 Analisis *Packet loss*

Pada penelitian yang dilakukan, *Packet loss* merupakan hasil olah data dari *capture packet* wireshark dan Tcp dump, nilai *packet loss* didapatkan dari hasil pembagian selisih antara paket data yang dikirim dengan paket data yang diterima, selanjutnya diubah dalam bentuk persen. Jika merujuk pada Tabel 2 Standarisasi *Packet loss* pada TIPHON maka semua metode dari beberapa skenario pengujian memiliki parameter *Packet loss* kategori sangat bagus ditunjukkan dengan data yang didapatkan bahwa tidak ada data yang hilang pada pengujian diseluruh skenario. Hal ini bisa terjadi karena nilai throughput yang dihasilkan begitu besar berkisar 16 Mbit/s hingga 45 Mbit/s. Besarnya throughput yang dihasilkan dapat mengurangi intensitas terjadinya paket yang hilang pada proses transmisi data. Sehingga paket data yang dikirim dapat diterima seluruhnya dengan sempurna.

Tabel 7. Nilai Rata – Rata Pengujian *Packet loss*

PACKET LOSS		
Jumlah *Request*	***Instance Web server 1*** **(Otomatis)**	***Instance Web server 2*** **(Manual)**
1000	0.000	0.000
5000	0.000	0.000

3.4 Analisis *Delay*

Pada penelitian yang dilakukan, untuk jumlah *Request* 1000 pengujiannya dilakukan sebanyak 15 kali didapatkan hasil rata – rata *delay* untuk *Instance Web server 1* (Otomatis) sebesar 0.146 ms dan hasil rata – rata *delay* pada *Instance Web server 2* (Manual) didapatkan nilai sebesar 0.143 ms. Pada jumlah *Request* 5000 pengujiannya dilakukan sebanyak 15 kali didapatkan hasil rata – rata *delay* mengalami kenaikan, untuk *Instance Web server 1* (Otomatis) sebesar 0.107 ms dan hasil rata – rata *delay* pada *Instance Web server 2* (Manual) didapatkan nilai sebesar 0.103 ms. Berdasarkan rata – rata nilai *delay* yang didapatkan, untuk *Instance Web server 2* (Manual) mendapatkan rata – rata *delay* dengan perbandingan nya sangat sedikit dengan hasil yang didapatkan dari rata – rata *delay* pada *Instance Web server 1*

(Otomatis). Salah satu alasan yang mendukung adalah dikarenakan pada data nilai *throughput* semakin besar maka nilai delay yang dihasilkan akan semakin kecil. dan didapatkan hasil nilai *throughput* dengan perbandingan nya yang tidak jauh antar kedua *Instance Web server*.

Tabel 8. Nilai Rata – Rata Pengujian *Delay*

DELAY		
Jumlah *Request*	***Instance Web server 1* (Otomatis)**	***Instance Web server 2* (Manual)**
1000	0.146	0.143
5000	0.107	0.103

3.5 Analisis *Jitter*

Pada penelitian yang dilakukan, untuk jumlah *Request* 1000 pengujiannya dilakukan sebanyak 15 kali didapatkan hasil rata – rata *jitter* berdasarkan pada Gambar 4.7 untuk *Instance Web server 1* (Otomatis) sebesar 0.145 ms dan hasil rata – rata *jitter* pada *Instance Web server 2* (Manual) didapatkan nilai sebesar 0.142 ms. Pada jumlah *Request* 5000 pengujiannya dilakukan sebanyak 15 kali didapatkan hasil rata – rata *jitter* mengalami kenaikan, untuk *Instance Web server 1* (Otomatis) sebesar 0.107 ms dan hasil rata – rata *jitter* pada *Instance Web server 2* (Manual) didapatkan nilai sebesar 0.103 ms. Berdasarkan rata – rata nilai *jitter* yang didapatkan, untuk *Instance Web server 2* (Manual) mendapatkan rata – rata *jitter* dengan perbandingan nya sangat sedikit dengan hasil yang didapatkan dari rata – rata *jitter* pada *Instance Web server 1* (Otomatis).hal ini disebabkan *jitter* akan semakin besar ketika nilai *delay* yang dihasilkan semakin besar. Nilai *jitter* dan *delay* yang dihasilkan juga akan mempengaruhi *throughput*.

Tabel 9. Nilai Rata – Rata Pengujian *Jitter*

JITTER		
Jumlah *Request*	***Instance Web server 1* (Otomatis)**	***Instance Web server 2* (Manual)**
1000	0.145	0.142
5000	0.108	0.103

3.6 Analisis CPU *Usage*

Pengukuran CPU *usage* dilakukan untuk mengetahui penggunaan CPU pada masing – masing web server, dengan tujuan untuk mengetahui kualitas yanag dihasilkan pada kedua metode pembuatan *Instance Web server* tersebut. presentasi CPU *usage* dari seluruh pengujian didapatkan untuk jumlah *Request* 1000 pengujiannya dilakukan sebanyak 15 kali didapatkan hasil rata – rata *CPU usage* berdasarkan pada Gambar 4.8 untuk *Instance Web server 1* (Otomatis) sebesar 56.531 % dan hasil rata – rata *CPU usage* pada *Instance Web server 2* (Manual) didapatkan nilai sebesar 52.331 %. Pada jumlah *Request* 5000 pengujiannya dilakukan sebanyak 15 kali didapatkan hasil rata – rata *CPU usage* mengalami kenaikan dikarenakan jumlah akses pengunjung lebih besar, untuk *Instance Web server 1* (Otomatis) sebesar 76.290 % dan hasil rata – rata *CPU usage* pada *Instance Web server 2* (Manual) didapatkan nilai sebesar 68.692 %. Keseluruhan pengujian menghasilkan *Instance Web server 2* (Manual) lebih stabil dengan selisih 5.899 % dikarenakan pada tampilan *website* untuk *Instance Web server 1* (Otomatis) memiliki tampilan lebih baik sehingga *script* yang dibuat lebih banyak dan membuat CPU *usage* pada *Instance Web server 1* (Otomatis) lebih tinggi.

Tabel 10. Nilai Rata – Rata Pengujian *CPU* usage

CPU USAGE		
Jumlah *Request*	***Instance Web server 1* (Otomatis)**	***Instance Web server 2* (Manual)**
1000	56.531	52.331
5000	76.290	68.692

IV. KESIMPULAN

Berdasarkan dari semua yang telah dilakukan selama pengerjaan penelitian, dapat disimpulkan bahwa untuk pengujian terhadap waktu dengan spesifikasi *Standard A4 v2* yaitu 4 vcpu dan 8GiB Memory pada *Virtual Machine* Microsoft Azure yang didalam nya dilakukan instalasi openstack, untuk *Instance web server* yang dibuat menggunakan

Infrastruktur As Code Terraform lebih unggul dengan selisih waktu antar kedua *Instance web server* 12 Menit 19 Detik menggunakan *Infrastruktur As Code Terraform* mampu lebih efisien terhadap waktu dalam pembuatan 1 *instance* beserta instalasi *web server* dan *Infrastruktur As Code Terraform* layak untuk dijadikan *tools* automasi sebagai penyedian infrastruktur *web server* otomatis pada pembangunan *web server* beserta *Virtual Machine* dengan selisih nilai rata-rata perbandingan dari kedua metode tersebut didapatkan nilai *throughput* 0.750 Mbit/s, *packetloss* 0.00%, selisih nilai *delay* 0,0035 ms, selisih nilai parameter *jitter* yaitu 0,004 ms, parameter CPU *usage*, didapatkan dengan selisih rata – rata diseluruh pengujian yaitu 5,899%.

DAFTAR PUSTAKA

[1] H. Triyanto, A. B. P. Negara, and M. A. Irwansyah, "Analisa Perbandingan Performa Openstack dan Apache Cloudstack dalam Model Cloud Computing Berbasis Infrastructure As a Service," *J. Sist. dan Teknol. Inf.*, vol. 8, no. 1, p. 78, 2020, doi: 10.26418/justin.v8i1.31936.

[2] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero, and D. A. Tamburri, "DevOps: Introducing infrastructure-as-code," *Proc. - 2017 IEEE/ACM 39th Int. Conf. Softw. Eng. Companion, ICSE-C 2017*, no. May, pp. 497–498, 2017, doi: 10.1109/ICSE-C.2017.162.

[3] perkom.co.id, "Mengenal Microsoft Azure," *www.perkom.co.id*, 2016. www.perkom.co.id/mengenal-microsoft-azure/%0D (accessed Mar. 30, 2020).

[4] I. M. A. S. Putu Gede Surya Cipta Nugraha, I Komang Ari Mogi, "Implementasi Private Cloud Computing Sebagai Layanan Infrastructure As a Service (Iaas)," *Stud. Progr. Inform. Tek. Komputer, Jur. Ilmu*, vol. 8, no. 2, pp. 7–14, 2015.

[5] R. Singh and R. K. Purwar, "Cloud Automation with Configuration Management using CHEF Tool," vol. 8, no. 04, pp. 140–146, 2019.

[6] L. R. De Carvalho and A. P. F. De Araujo, "Performance Comparison of Terraform and Cloudify as Multicloud Orchestrators," *Proc. - 20th IEEE/ACM Int. Symp. Clust. Cloud Internet Comput. CCGRID 2020*, pp. 380–389, 2020, doi: 10.1109/CCGrid49817.2020.00-55.

[7] U. Gunadarma, "Network Traffic Management, Quality of Services (QoS), Congestion Control dan Frame Relay," pp. 12–24, 2011, [Online]. Available: http://mujahidin.staff.gunadarma.ac.id/Downloads/files/37741/Materi+Jaringan+Komputer+Lanjut+2.pdf.

[8] ETSI, "Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON); General aspects of Quality of Service (QoS)," *Etsi Tr 101 329 V2.1.1*, vol. 1, pp. 1–37, 1999, [Online]. Available: http://www.etsi.org/deliver/etsi_tr/101300_101399/101329/02.01.01_60/tr_101329v020101p.pdf.