BAB 3
METODE PENELITIAN

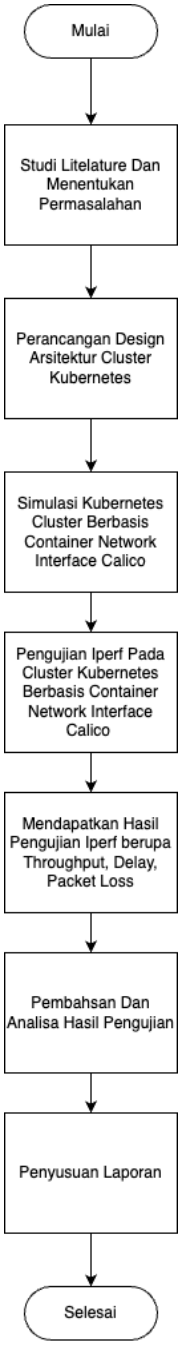
3.1 ALUR PENELITIAN

Dalam penulisan skripsi ini, terdapat beberapa tahapan yang dilakukan, mulai dari studi literatur hingga penyusunan laporan. Pada tahap awal, dilakukan studi literatur untuk memahami konsep dasar *Kubernetes* dan *Container Network Interface Calico*. Tahap ini sangat penting untuk menentukan permasalahan yang akan dijawab dalam skripsi ini. Setelah menentukan permasalahan, dilakukan.

Gambar 3.1 merupakan *flowchart* penelitian. Perancangan *design* arsitektur kluster *Kubernetes* yang akan digunakan dalam implementasi dan pengujian. Tahap selanjutnya adalah implementasi kluster *Kubernetes* berbasis *Container Network Interface Calico.* Pada tahap ini, akan dilakukan instalasi dan konfigurasi kluster *Kubernetes* menggunakan *Container Network Interface Calico*. Setelah berhasil melakukan implementasi, tahap selanjutnya adalah melakukan pengujian performa kluster menggunakan alat uji *Iperf* dan juga ada tahap jika pengujian tidak berhasil maka dilakukan pengujian ulang.

Pada tahap pembahasan dan analisa hasil pengujian, akan dianalisis hasil pengujian yang telah dilakukan pada kluster *Kubernetes* berbasis *Container Network Interface Calico* menggunakan alat uji *Iperf*. Analisis ini akan mencakup performa jaringan, *throughput, latency, dan packet loss* pada kluster *Kubernetes* tersebut.

Setelah selesai melakukan analisis, tahap terakhir adalah penyusunan laporan yang akan berisi hasil penelitian yang telah dilakukan. Laporan ini akan membahas secara detail tentang perancangan arsitektur kluster *Kubernetes*, implementasi *Container Network Interface Calico,* dan pengujian performa kluster menggunakan alat uji *Iperf.*



Gambar 3.1 *Flowchart* Penelitian

3.2 ALAT DAN BAHAN

3.2.1 Perangkat Keras

Perangkat keras yang diperlukan dalam penelitian ini adalah :

1. Laptop

Laptop digunakan adalah *mackbook* M1 2020 sebagai perangkat untuk menghubungkan dengan *virtual machine* yang berada di *server*.

3.2.2 Perangkat Lunak

Tabel 3.1 merupakan spesifikasi perangkat lunak yang digunakan, Perangkat yang digunakan pada penelitian terdiri dari:

Tabel 3.1 Spesifikasi Perangkat Lunak

No	*Software*	Versi
1.	Ubuntu	20.04
2.	Macos	M1
3.	*Kubernetes*	1.26.2
4.	*Docker*	1.6.12
5.	KVM	4.2.1
6.	libvirtd	6.0.0
7.	*Terraform*	1.3.6
8.	MacTerminal	13.0
9.	*Iperf*	*iperf*3-netperf:v1.0

1. Terminal

MacOS adalah sistem operasi yang dikembangkan oleh *Apple* dan digunakan pada produk-produk komputer buatan mereka, termasuk *MacBook*. *Terminal* pada macOS versi 13.0 dengan build version 22A380 yang tertera di *MacBook* M1 2020 saya, adalah aplikasi bawaan yang berfungsi sebagai antarmuka untuk menggunakan perintah *Unix* dan mengakses berbagai fitur sistem operasi *macOS* melalui baris perintah. Pada skripsi saya terminal pada *macOS* versi tersebut digunakan untuk mengakses mesin *virtual* yang berjalan di atas sistem operasi *linux ubuntu* 20.04 yang digunakan untuk menjalankan skema *cluster kubernetes*.

2. *Linux* Ubuntu 20.04

Linux Ubuntu 20.04 dipilih sebagai sistem operasi yang digunakan karena kestabilannya dan dukungan yang luas dari komunitas. Selain itu, *linux ubuntu* 20.04 juga memiliki antarmuka yang mudah digunakan sehingga memudahkan pengguna untuk melakukan instalasi dan konfigurasi.

3. *Virtual Machine*

Virtual machine digunakan untuk memisahkan lingkungan kerja dari *host* dan membuat lingkungan kerja yang terisolasi. Pada skripsi saya, *virtual machine* digunakan untuk menjalankan beberapa *node* dalam *cluster Kubernetes* dan memastikan bahwa setiap *node* berjalan pada lingkungan kerja yang terpisah dari lingkungan kerja *host*. Saya menggunakan *virtual* box sebagai *tools* untuk membuat *virtual machine*, dengan version *virtual* box 6.1.42r155177

4. *Vagrant*

Vagrant digunakan untuk mempermudah proses pembuatan dan manajemen *virtual machine* dalam skripsi saya. *Vagrant* menyediakan banyak fitur seperti konfigurasi *virtual machine* dengan mudah, sharing folder antara *host* dan *guest*, serta *provisioning* yang otomatis sehingga menghemat waktu dalam melakukan instalasi dan konfigurasi. Saya menggunakan *vagrant version* 2.3.4 yaitu vesion latest saat ini.

5. Docker

Docker digunakan sebagai platform untuk menjalankan aplikasi dalam *container*. Pada skripsi saya, *Docker* digunakan untuk menjalankan *container* yang dijalankan dalam *cluster kubernetes*. *Docker* menyediakan fitur untuk mengelola dan mempercepat proses deployment aplikasi, serta memastikan konsistensi antara lingkungan kerja pada setiap *node* dalam *cluster*. Saya menggunakan version *Docker1*.6.18.

6. *Kubernetes*

Kubernetes adalah platform *open-source* untuk manajemen *container* yang dirancang untuk menjalankan aplikasi dalam lingkungan yang terdistribusi dan *scalable*. Pada skripsi saya, *Kubernetes* digunakan sebagai platform untuk mengatur dan mengelola *container-container* dalam *cluster*. *Kubernetes* menyediakan fitur untuk scaling, load balancing, dan manajemen *resource* yang memastikan aplikasi berjalan dengan performa yang optimal, saya menggunakan *version kubernetes* di *cluster kubernetes* .

7. *Iperf*

Iperf digunakan untuk mengukur performa jaringan dengan mengirimkan data secara terus menerus antara dua mesin. Dalam skripsi saya, *Iperf* digunakan untuk mengukur bandwidth pada setiap *container* yang terhubung ke jaringan menggunakan *calico* CNI. *Iperf* menyediakan fitur untuk mengukur performa jaringan seperti *throughput*, jitter, dan *packet loss*, saya menggunakan *image container iperf* gokulpch/*iperf3*-netperf:v1.0 untuk dijalankan di *pod kubernetes*.

3.3 PERANCANGAN DESAIN KLUSTER ARSITEKTUR *KUBERNETES*

Dalam perancangan arsitektur kluster *kubernetes* yang dilakukan, menggunakan sebuah *server* yang menggunakan teknologi kvm (*Kernel Virtual Machine*) dan *teraform* untuk membuat *virtual machine* yang akan digunakan. Kemudian ,membuat kluster *Kubernetes* berbasis *container network interface calico* dengan menggunakan *vagrant*, namun hanya menggunakan satu *node* dalam kluster tersebut. Setelah itu, membuat sebuah *pod nginx* untuk digunakan dalam skenario pengujian yang akan diakses oleh *client.*

Gambar 3.2 *Config Terraform* Pembuatan *Virtual Machine*

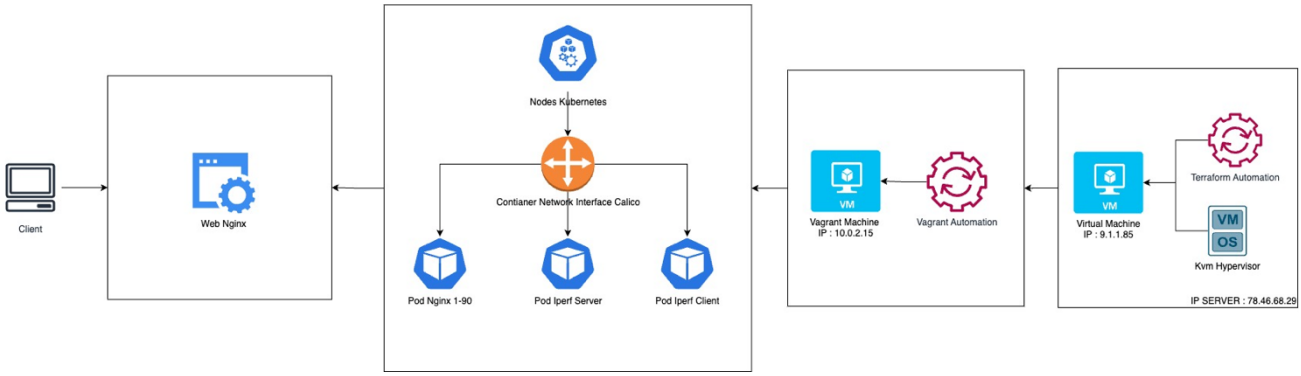
.

Gambar 3.2 merupakan *config terraform pembuatan virtual machine,* Dalam menggunakan teknologi kvm dan *terraform*, dapat dengan mudah dan efisien membuat *virtual machine* yang dibutuhkan untuk kluster *Kubernetes*. Penggunaan *vagrant* juga memudahkan dalam membuat kluster *Kubernetes* dengan cepat dan memastikan bahwa kluster tersebut berjalan dengan baik. Meskipun hanya menggunakan satu *node* dalam kluster *kubernetes* yang rancang,

namun memastikan bahwa *pod nginx* yang buat dapat digunakan sebagai sumber pengujian performa jaringan pada kluster tersebut. Tabel 3.1 merupakan spesifikasi dari *virtual mechine.*

Tabel 3. 1 Spesifikasi *Virtual* Mechine

Keterangan	Spesifikasi
Hypervisor Vendor	KVM (*Kernel*-Based *Virtual Machine*) Version .2.1
Nama VM	*Vagrant*
Sistem Operasi	Ubuntu 20.04.5 LTS
vCPU	8 Intel(R) Xeon(R) CPU E5-1650 v3 @ 3.50GHz
RAM	4 GB
Disk	40 GB



Gambar 3. 3 Topologi Jaringan

Gambar 3.3 merupakan topologi jaringan yang digunakan pada penelitian ini. Selanjutnya untuk Tabel 3.2 merupakan spesifikasi dari *vagrant mechine* dengan keterangan *Hypervisor Vendor*, Nama VM, Sistem Operasi, vCPU, RAM dan Disk

Tabel 3. 2 Spesifikasi *Vagrant Mechine*

Keterangan	Spesifikasi
Hypervisor Vendor	KVM (*Kernel*-Based *Virtual Machine*) Version 4.2.1
Nama VM	*vagrant*-kube-pikli
Sistem Operasi	Ubuntu 20.04.5 LTS
vCPU	13 Intel(R) Xeon(R) CPU E5-1650 v3 @ 3.50GHz
RAM	14 GB
Disk	70 GB

Berikut akan dijelaskan proses pembuatan *virtual machine* pada penelitianini. Pembuatan *virtual machine* menggunakan hypervisor KVM yang ada di *server* fisik dengan spesifikasi sesuai dengan tabel 3.2. Pada tahap ini, *source code* ditulis untuk membuat VM pada *Physical Server* sesuai dengan *source code* dibawah Nantinya untuk pembuatan VM.

```
[VM1]
NAME: vagrant-kube-pikli-2
OS: focal-server-cloudimg-amd64.img
NESTED: y
VCPUS: 16
MEMORY: 16G
DISK1: 70G
IFACE_NETWORK1: 9.1.1.0
IFACE_IP1: 9.1.1.85
CONSOLE: vnc
```

Setelah itu Membuat Konfigurasi *Vagrantfile* Yang akan digunakan untuk membuat *vagrant machine* didalam *vagrant*file akan mendefenisikan spesifikasi dari *vagrant machine* yang ingin dibuat dari cpu ram dan ip yang akan digunakan.

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/focal64"
  config.vm.box_version = "20230523.0.0"

  config.vm.network "private_network", ip:
"192.168.56.210"

  config.vm.provider "virtualbox" do |vb|
    vb.cpus = 13
    vb.memory = 14000
  end

  if Vagrant.has_plugin?("vagrant-vbguest")
    config.vbguest.auto_update = false
  end
  config.vm.provision "shell", privileged: false, inline:
<<-SHELL
    sudo snap install yq
    bash -eu /vagrant/install.sh 192.168.56.210
  SHELL
end
```

Selanjutnya setelah selesai mendeploy *cluster kubernetes* berbasis *container network interface calico* adalah membuat konfigurasi yaml *iperf* dengan membuat *deployment*, *service* dan *daemonset* untuk pengujian *iperf*.

Setelah mendeploy *iperf* didalam *cluster kubernetes* selanjutnya membuat *script* yang digunakan untuk mempermudah operational saat pengujian *iperf*, disini *script* nya diantaranya adalah untuk mendeploy automatis *iperf* , untuk menjalankan *iperf* setelah didepploy dan untuk menghapus *iperf* setelah melakukan pengujian *iperf* .

```
vagrant@ubuntu-focal:~/kubernetes-iperf3/steps$ cat
cleanup.sh
#!/usr/bin/env bash
set -eu

kubectl delete --cascade -f iperf3.yaml
vagrant@ubuntu-focal:~/kubernetes-iperf3/steps$ cat
setup.sh
#!/usr/bin/env bash
set -eu

kubectl create -f iperf3.yaml

until $(kubectl get pods -l app=iperf3-server -o
jsonpath='{.items[0].status.containerStatuses[0].ready}');
do
    echo "Waiting for iperf3 server to start..."
    sleep 5
done

echo "Server is running"
echo
vagrant@ubuntu-focal:~/kubernetes-iperf3/steps$ cat run.sh
#!/usr/bin/env bash
set -eu

CLIENTS=$(kubectl get pods -l app=iperf3-client -o name |
cut -d'/' -f2)

for POD in ${CLIENTS}; do
    until $(kubectl get pod ${POD} -o
jsonpath='{.status.containerStatuses[0].ready}'); do
        echo "Waiting for ${POD} to start..."
        sleep 5
    done
    HOST=$(kubectl get pod ${POD} -o
jsonpath='{.status.hostIP}')
    kubectl exec -it ${POD} -- iperf3 -c iperf3-server -T
"Client on ${HOST}" $@
```

```
    echo
done
```

Setelah itu membuat *script* yang akan menjalakan automatis dari *deploy iperf* sampai menghapus *iperf* setelah *iperf* nya dijalankan dan mendapatkan data dari *iperf* tersebut.

```
#!/usr/bin/env bash
set -eu

cd $(dirname $0)

steps/setup.sh
steps/run.sh $@
steps/cleanup.sh
```

Dan disini saya membuat *script* untuk mempermudah pengujian yang dimana di pengujian akan scale *pod nginx* dari 5 sampai 90 dan setelah scale *pod nginx* akan menjalankan *iperf* dengan parameter pengujian dari 5 sampai 90.

```
#!/bin/bash

kubectl scale --replicas=5 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 5 -P 5 -b 5k -u -t 5 > test-5.txt

kubectl scale --replicas=10 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 10 -P 10 -b 10k -u -t 10 > test-
10.txt

kubectl scale --replicas=15 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 15 -P 15 -b 15k -u -t 15 > test-
15.txt

kubectl scale --replicas=20 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 20 -P 20 -b 20k -u -t 20 > test-
20.txt

kubectl scale --replicas=25 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 25 -P 25 -b 25k -u -t 25 > test-
25.txt

kubectl scale --replicas=30 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
```

```
./iperf3.sh -p 5201 -M 30 -P 30 -b 30k -u -t 30 > test-
30.txt

kubectl scale --replicas=35 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 35 -P 35 -b 35k -u -t 35 > test-
35.txt

kubectl scale --replicas=40 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 40 -P 40 -b 40k -u -t 40 > test-
40.txt

kubectl scale --replicas=45 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 45 -P 45 -b 45k -u -t 45 > test-
45.txt

kubectl scale --replicas=50 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 50 -P 50 -b 50k -u -t 50 > test-
50.txt

kubectl scale --replicas=55 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 55 -P 55 -b 55k -u -t 55 > test-
55.txt

kubectl scale --replicas=60 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 60 -P 60 -b 60k -u -t 60 > test-
60.txt

kubectl scale --replicas=65 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 65 -P 65 -b 65k -u -t 65 > test-
65.txt

kubectl scale --replicas=70 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 70 -P 70 -b 70k -u -t 70 > test-
70.txt

kubectl scale --replicas=75 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 75 -P 75 -b 75k -u -t 75 >  test-
75.txt

kubectl scale --replicas=80 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 80  -P 80 -b 80k -u -t 80 >  test-
80.txt
```

```
kubectl scale --replicas=85 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 85  -P 85 -b 85k -u -t 85 >  test-
85.txt

kubectl scale --replicas=90 deployment/nginx-deployment
kubectl delete --cascade -f iperf3.yaml
./iperf3.sh -p 5201 -M 90  -P 90 -b 90k -u -t 90 >  test-
90.txt
```

Dengan demikian, penggunaan teknologi kvm, *terraform*, *vagrant*, dan *pod nginx* merupakan bagian penting dalam perancangan arsitektur kluster *Kubernetes* yang lakukan. Meskipun hanya menggunakan satu *node* dalam kluster *Kubernetes*, namun bahwa hasil pengujian yang lakukan dapat memberikan informasi yang bermanfaat terkait performa jaringan pada kluster yang dirancang.

3.4 SIMULASI *KUBERNETES* KLUSTER BERBASIS *CONTAINER NETWORK INTERFACE CALICO*

Setelah merancang arsitektur kluster *Kubernetes* yang dibutuhkan, Langkah selanjutnya adalah melakukan implementasi. Implementasi kluster ini dilakukan dengan menggunakan beberapa tahapan yang telah disiapkan sebelumnya. Pertama, membuat *virtual machine* dengan menggunakan teknologi kvm dan *terraform*. memilih menggunakan teknologi kvm karena dapat mengoptimalkan penggunaan re*source* pada *server* dan dapat mendukung pembuatan *virtual machine* dengan skala yang besar.

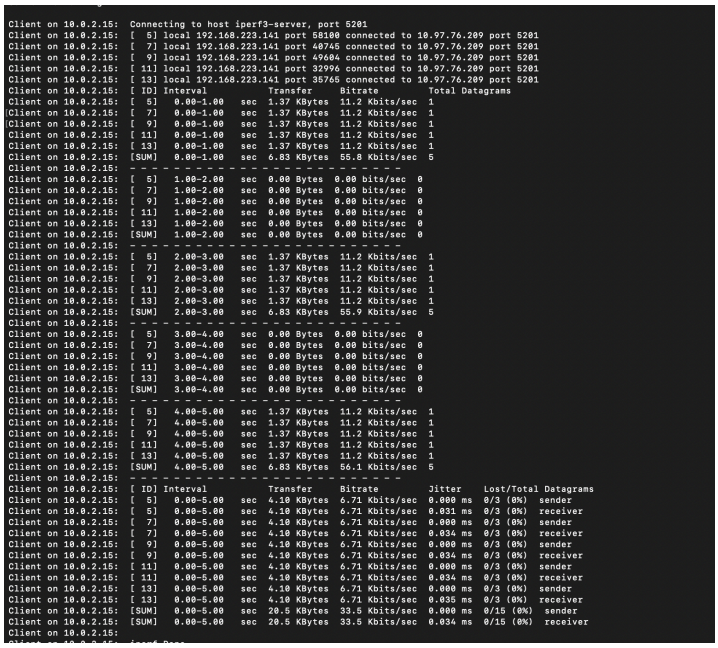
Setelah *virtual machine* berhasil dibuat, melanjutkan dengan membuat kluster *kubernetes* menggunakan *vagrant*. memilih menggunakan *vagrant* karena dapat mempercepat proses pembuatan dan konfigurasi kluster. Pada proses pembuatan kluster, mengimplementasikan *container network interface calico* sebagai salah satu komponen penting dalam kluster. *Container network interface calico* dipilih karena dapat memberikan kemudahan dalam mengatur jaringan antara *pod* dan *node* dalam kluster *kubernetes*.

Setelah berhasil membuat kluster, melakukan konfigurasi pada kluster *kubernetes* dengan menggunakan *container network interface calico*. Konfigurasi

dilakukan dengan memastikan setiap *node* dan *pod* dapat terhubung dengan baik dan jaringan antara *pod* dan *node* terkonfigurasi dengan baik. Hal ini dilakukan untuk memastikan bahwa kluster dapat berjalan dengan baik dan sesuai dengan yang direncanakan.

3.5 PENGUJIAN IPERF PADA KLUSTER KUBERNETES BERBASIS CONTAINER NETWORK INTERFACE CALICO

Setelah sukses mengimplementasikan kluster *kubernetes* dengan menggunakan *calico* sebagai *container network interface*, langkah selanjutnya adalah melakukan pengujian performa untuk memastikan bahwa kluster berfungsi secara optimal. Dalam proses pengujian ini, menggunakan alat bantu *Iperf* untuk mengukur *throughput* data, *packetloss* dan *delay* pada jaringan.

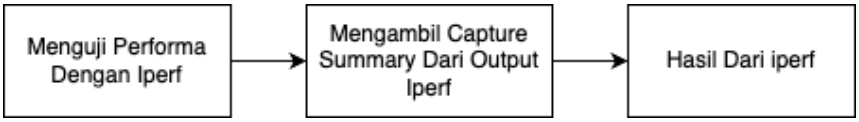


Gambar 3. 3 Contoh *Output* Dari Pengujian *Iperf*

Gambar 3.3 merupakan contoh *output* dari pengujian *iperf,* Proses pengujian lakukan dengan membuat beberapa *pod* sebagai *client* dan *server* pada kluster *kubernetes* yang telah diimplementasikan. Selama pengujian, mengirimkan data antara client dan *server* dengan menggunakan protokol *delay, packet loss* dan *throughput* di kluster *kubernetes* berbasis *container network interface calico*.

3.6 SKENARIO PENGUJIAN *IPERF* PADA KLUSTER *KUBERNETES* BERBASIS *CONTAINER NETWORK INTERFACE CALICO*

Pada tahap awal pengujian kinerja dalam penelitian ini, dilakukan proses konfigurasi yang sangat penting untuk pengujian *iperf.* Konfigurasi ini melibatkan pembuatan file YAML yang akan digunakan untuk mengatur *service, deployment,* dan *daemonset iperf*. Melalui konfigurasi ini, *service* dibuat untuk memastikan terjalinnya konektivitas yang optimal, *deployment* digunakan untuk mengontrol jumlah dan replikasi *pod iperf* yang akan diuji, sementara *daemonset* berperan penting dalam menjalankan *pod iperf* di setiap *node* yang ada. Dengan persiapan konfigurasi yang cermat ini, para peneliti dapat melakukan pengujian kinerja dengan kontrol yang tepat dan mampu melakukan pengulangan pengujian dengan mudah. Konfigurasi ini memberikan kebebasan untuk mengatur dan menyesuaikan parameter pengujian sesuai dengan skenario yang ditentukan. Gambar 3.4 merupakan diagram blok alur pengujian yang digunakan.



Gambar 3. 4 Diagram Blok Alur Pengujian

Untuk mempermudah dan menyederhanakan pelaksanaan pengujian, langkah selanjutnya yang diambil adalah membuat sebuah skrip bash perulangan khusus. Skrip ini telah dirancang secara khusus untuk memfasilitasi proses pengujian dengan efisiensi yang tinggi. Dengan menggunakan skrip ini, penguji tidak perlu lagi melakukan pengujian secara manual, yang membutuhkan waktu dan rentan terhadap kesalahan manusia. Melalui skrip bash perulangan ini, pengujian dapat dikonfigurasi dengan mudah, termasuk pengaturan jumlah iterasi atau repetisi pengujian yang diinginkan. Skrip ini akan menjalankan perintah pengujian secara berulang sesuai dengan konfigurasi yang telah ditentukan, sehingga penguji dapat fokus pada analisis dan interpretasi hasil yang diperoleh. Penggunaan skrip bash perulangan ini memberikan manfaat besar dalam mengotomatisasi proses pengujian, sehingga memungkinkan pengumpulan data dan hasil pengujian secara efisien dan konsisten. Dengan demikian, penggunaan skrip ini menjadi sebuah alat

yang sangat berharga dalam menjalankan pengujian dengan efektivitas dan memberikan kemudahan serta efisiensi yang signifikan dalam proses pengujian.

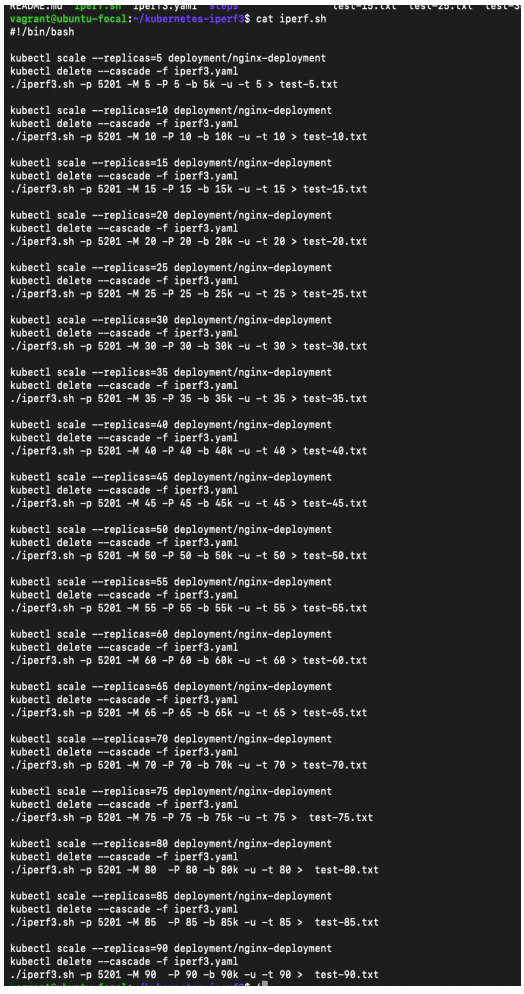
Kemudian untuk memulai pengujian digunakan perintah *Iperf* yang telah disesuaikan sesuai dengan skenario pengujian yang ditetapkan. Perintah yang digunakan untuk pengujian adalah sebagai berikut: "./*iperf*3.sh -p 5201 -M 2 -P 10 -b 2m -u -t 2". Dalam perintah tersebut, setiap parameter memiliki peran penting dalam proses pengujian. Argumen "-p 5201" digunakan untuk mengatur *port* yang digunakan dalam pengujian, sedangkan argumen "-M 2" menentukan ukuran maksimum segmen TCP yang akan digunakan. Selanjutnya, argumen "-P 10" menentukan jumlah sesi *paralel* yang dibuat selama pengujian, dan argumen "-b 2m" digunakan untuk mengatur tingkat laju *transfer* yang akan diuji. Penggunaan argumen "-u" menandakan penggunaan protokol UDP dan TCP dalam pengujian, sementara argumen "-t 2" menentukan durasi pengujian selama 2 detik. Kombinasi dari parameter-parameter ini memberikan fleksibilitas kepada peneliti untuk menguji kinerja sistem dengan pengaturan yang spesifik sesuai dengan skenario yang ditentukan.

Skenario pengujian yang akan dilakukan adalah menggunakan *pod iperf* untuk melakukan tes. Dalam skenario ini, terdapat *pod iperf-client* dan *iperf- server* yang harus diset *service* nya. Selama pengujian, akses client akan dilakukan menggunakan *pod web nginx*. Berikut adalah skenario lengkapnya:

1. Pertama, akan membuat 5 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 5, Setiap pengujian akan mengambil data *throughput*, *delay* dan *packet loss*.
2. Pertama, akan membuat 10 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 10, Setiap pengujian akan mengambil data *throughput*, *delay* dan *packet loss*.
3. Pertama, akan membuat 15 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 15, Setiap pengujian akan mengambil data *throughput*, *delay* dan *packet loss*.
4. Pertama, akan membuat 20 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 20, Setiap pengujian akan mengambil data *throughput*, *delay* dan *packet loss*.

5. Pertama, akan membuat 25 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 25, Setiap pengujian akan mengambil data *throughput*, *delay* dan *packet loss*.
6. Pertama, akan membuat 30 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 30, Setiap pengujian akan mengambil data *throughput*, *delay* dan *packet loss*.
7. Pertama, akan membuat 35 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 35, Setiap pengujian akan mengambil data *throughput*, *delay* dan *packet loss*.
8. Pertama, akan membuat 40 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 40, Setiap pengujian akan mengambil data *throughput*, *delay* dan *packet loss*.
9. Pertama, akan membuat 45 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 45, Setiap pengujian akan mengambil data *throughput*, *delay* dan *packet loss*.
10. Pertama, akan membuat 50 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 50, Setiap pengujian akan mengambil data *throughput*, *Delay* dan *Packet Loss*.
11. Pertama, akan membuat 55 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 55, Setiap pengujian akan mengambil data *throughput*, *delay* dan *packet loss*.
12. Pertama, akan membuat 60 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 60, Setiap pengujian akan mengambil data *throughput*, *delay* dan *packet loss*.
13. Pertama, akan membuat 65 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 65, Setiap pengujian akan mengambil data *throughput*, *delay* dan *packet loss*.
14. Pertama, akan membuat 70 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 75, Setiap pengujian akan mengambil data *throughput*, *delay* dan *packet loss*.

15. Pertama, akan membuat 80 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 80, Setiap pengujian akan mengambil data *throughput*, *delay* dan *packet loss*.
16. Pertama, akan membuat 85 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 85, Setiap pengujian akan mengambil data *throughput*, *delay* dan *packet loss*.
17. Pertama, akan membuat 90 *pod web nginx* dan melakukan pengujian *iperf* sebanyak 90, Setiap pengujian akan mengambil data *throughput*, *delay* dan *packet loss*.



Gambar 3. 5 *Script* Pengujian

Gambar 3.5 merupakan *script* yang digunakan untuk pengujian, Dalam melakukan pengerjaan skenario pengujian seperti dijelaskan diatas, dilakukan dengan menggunakan *script* untuk mempermudah dalam pengujian, didalam *script*

ada dua fungsi yang digunakan untuk melakukan pengujian diantaranya adalah untuk *scale pod web* ngnix yang otomatis dan setelah itu menjalankan *iperf* dengan parameter sesuai jumlah dari *pod web nginx* yang telah di *scale*

Dengan melakukan serangkaian pengujian ini, bertujuan untuk mengevaluasi kinerja *container network interface calico* dengan jumlah *pod* yang berbeda. ingin melihat bagaimana *throughput* berubah sesuai dengan skala infrastruktur yang digunakan. Setiap pengujian akan memberikan informasi penting tentang kemampuan dari kluster *kubernetes* dalam menghadapi beban lalu lintas yang semakin tinggi.

Tabel 3. 3 Skenario

Jumlah *Pod Web Nginx*	Jumlah Pengujian *Iperf*	Waktu *Transfer*	Parameter		
5	5	5	*PacketLoss*	*Throughput*	*Delay*
10	10	10			
15	15	15			
20	20	20			
25	25	25			
30	30	30			
35	35	35			
40	40	40			
45	45	45			
50	50	50			
55	55	55			
60	60	60			
65	65	65			
70	70	70			
75	75	75			
80	80	80			
85	85	85			
90	90	90			

Tabel 3.3 merupakan tabel skenario pengujian, memberikan penjelasan terinci tentang hasil pengujian kinerja yang melibatkan jumlah *pod web nginx*, jumlah pengujian *iperf*, waktu *transfer*, *throughput*, dan *delay*. Data dalam tabel ini

menggambarkan hasil pengujian yang dilakukan dengan memvariasikan jumlah *pod web nginx* dan jumlah pengujian *iperf*.

Dalam pengujian tersebut, terlihat bahwa ketika jumlah *pod web nginx* dan jumlah pengujian *iperf* ditingkatkan, terjadi peningkatan yang signifikan pada waktu *transfer* dan *throughput*. Sebagai contoh, pada pengujian awal dengan 5 *Pod Web Nginx* dan 5 pengujian *Iperf*, waktu *transfer* yang tercatat hanya sebesar 5 detik, dengan *throughput* mencapai 20.5 KBytes. Namun, saat jumlah *Pod Web Nginx* dan pengujian *Iperf* ditingkatkan hingga mencapai 90, waktu *transfer* yang tercatat mencapai 90 detik dengan *throughput* mencapai 87.0 MBytes. Pola peningkatan ini menunjukkan adanya hubungan yang kuat antara jumlah *pod web nginx*, jumlah pengujian *iperf*, waktu *transfer*, dan *throughput* dalam pengujian kinerja.

Selain itu, dalam setiap tabel entri, data juga menyajikan nilai *delay* yang diukur dengan presisi dalam milidetik (ms). Nilai *delay* ini memberikan gambaran tentang lamanya waktu tunggu yang terjadi selama proses *transfer* data antara komponen-komponen dalam lingkungan. Penting untuk dicatat bahwa dalam rangkaian pengujian ini, terdapat kecenderungan yang jelas di mana nilai *delay* cenderung meningkat sejalan dengan bertambahnya jumlah *pod web nginx* yang diaktifkan, serta dengan bertambahnya jumlah pengujian *iperf* yang dilakukan secara bersamaan.

Peningkatan *delay* ini mengindikasikan adanya potensi keterlambatan dalam komunikasi antar komponen sistem. Pertumbuhan nilai *delay* bisa disebabkan oleh beberapa faktor, termasuk pemrosesan yang lebih intensif di sisi *server* atau jaringan yang semakin padat karena adanya lebih banyak lalu lintas data. Pengamatan terhadap perubahan nilai *delay* ini memberikan wawasan penting tentang bagaimana kinerja sistem dapat dipengaruhi oleh skenario penggunaan yang berbeda. Analisis lebih lanjut perlu dilakukan untuk mengidentifikasi penyebab pasti dari peningkatan nilai *delay* dan mengambil langkah-langkah yang sesuai untuk mengoptimalkan performa keseluruhan sistem.

Dalam konteks keseluruhan, analisis yang mendalam dari tabel skenario yang disajikan memberikan wawasan yang sangat berharga terkait dampak dari variasi dua faktor utama, yaitu jumlah pod web nginx dan jumlah pengujian iperf,

terhadap sejumlah parameter kritis dalam pengujian kinerja. Informasi yang dihasilkan dari penelitian ini tidak hanya memiliki relevansi teoritis, tetapi juga memberikan panduan praktis yang dapat digunakan oleh para profesional di lapangan. Dengan memahami bagaimana perubahan jumlah pod web nginx berdampak pada waktu transfer, throughput, dan delay, serta bagaimana variabel jumlah pengujian iperf turut mempengaruhi parameter-parameter ini, praktisi dan pengembang sistem dapat mengambil keputusan yang lebih terinformasi untuk mengoptimalkan performa infrastruktur yang ada. Penelitian ini dapat menjadi p penting dalam upaya meningkatkan skalabilitas sistem secara efisien, sehingga mampu menghadapi tantangan pertumbuhan yang semakin kompleks di era teknologi saat ini.