BAB 3
METODE PENELITIAN

3.1 ALAT YANG DIGUNAKAN

Penelitian ini menggunakan perangkat keras (*hardware*) berupa laptop dengan spesifikasi pada Tabel 3.1. Pada Tabel 3.2 perangkat lunak (*software*) yang digunakan, melalui layanan simulasi *Google Cloud Platform* untuk membuat virtual mesin yang akan mensimulasikan *server* Kubernetes. *Server* Kubernetes menggunakan 5 mesin virtual dari *Google Cloud Platform* yang akan dikonfigurasi menjadi 1 *master node* dan 4 *worker node*.

Tabel 3.1 Spesifikasi *hardware* yang digunakan

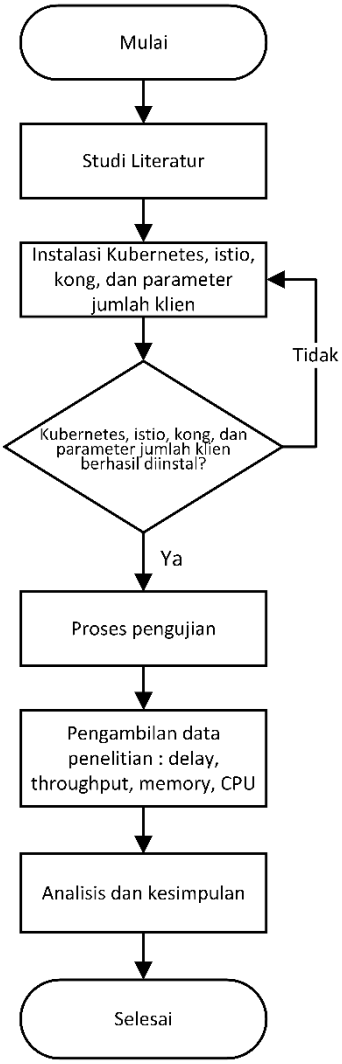
Perangkat	Spesifikasi	Keterangan
Laptop Lenovo Thinkpad T470	OS	Windows 10
	CPU	i5-7200U 4 CPU
	RAM	8 GB

Tabel 3.2 Spesifikasi *software* yang digunakan

Perangkat	Spesifikasi	Keterangan
Simulasi	Google Cloud Platform	VM instances
Master node	OS	Ubuntu Server 20.04
	CPU	2 vCPU
	RAM	8 GB
Worker node 1, 2, 3 dan 4	OS	Ubuntu Server 20.04
	CPU	2 vCPU
	RAM	8 GB

3.2 ALUR PENELITIAN

Penelitian dilakukan dalam beberapa tahap yaitu tahap perancangan sistem, tahap pembuatan simulasi, tahap pengujian simulasi, dan yang terakhir adalah tahap analisis dari hasil pengujian simulasi.



Gambar 3.1 Alur Penelitian

3.2.1 Studi Literatur

Studi literatur pada penelitian ini bertujuan untuk mencari referensi untuk dijadikan sebagai acuan dalam pelaksanaan penelitian. Adapun referensi utama yang digunakan yaitu jurnal yang memiliki kedekatan yang sama dari segi materi dan pembahasan dengan penelitian yang akan dilakukan. Sumber yang digunakan oleh penulis berupa jurnal, buku, dan *website* yang berhubungan dengan topik penelitian.

3.2.2 Perancangan Skenario

Pada perancangan scenario penelitian ini akan menggunakan dua jenis *inggres* yang berbeda yaitu kong *ingress controller* dan istio ingress *controller*.

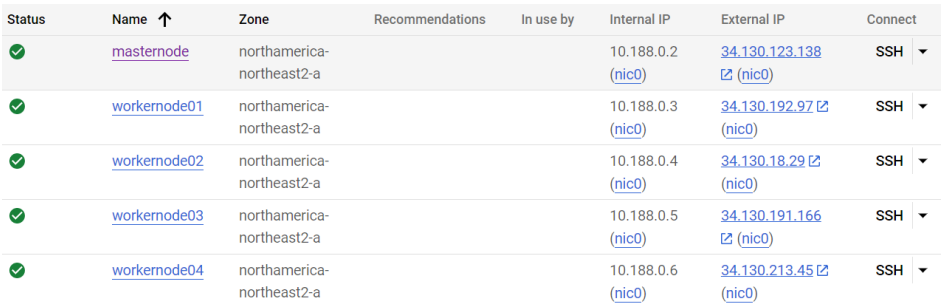
Controller tersebut merupakan aspek utama dalam penelitian ini, *controller* tersebut akan diakses untuk mengetahui seberapa besar performansinya menggunakan beberapa jumlah *klien* yaitu 50, 150, 250, 350, 450, 550, 650, dan 750 *klien*. Jumlah *klien* tersebut merupakan representasi jumlah *klien* pada penerapan nyata di mana biasanya untuk suatu layanan akan *web server* akan di akses kurang lebih antara 50 sampai dengan 750 *klien* per menit nya. Skenario penelitian ini dapat dilihat pada Tabel 3.3.

Tabel 3.3 Skenario Penelitian

Skenario	Ingress Controller	Jumlah *Klien*
1	Kong *Ingress Controller*	50, 150, 250, 350, 450, 550, 650, 750
2	Istio *Ingress Controller*	50, 150, 250, 350, 450, 550, 650, 750

3.2.2.1 *Setting Google Cloud Platform*

Simulasi dilakukan melalui *Google Cloud Platform* yang digunakan untuk membuat virtual mesinnya. Pada Gambar 3.2 terdapat 5 *server*, yaitu terdiri dari sebuah *master node*, *worker node* 1, *worker node* 2, *worker node* 3, dan *worker node* 4. Spesifikasi dari kelima *server* tersebut tertulis pada Tabel 3.2, yaitu menggunakan sistem operasi Ubuntu server 20.04, CPU berjumlah 2 vCPU, serta RAM sebesar 8 GB. Kelima server ini nantinya yang akan membentuk Kubernetes cluster setelah dikonfigurasi.

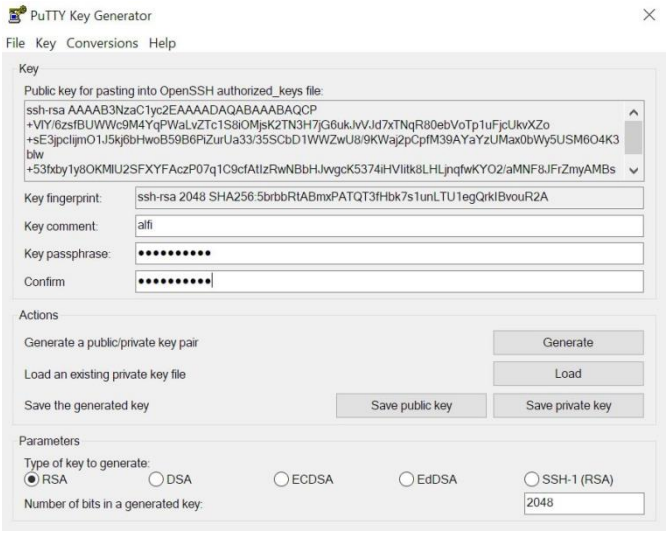


Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
✓	masternode	northamerica-northeast2-a			10.188.0.2 (nic0)	34.130.123.138 (nic0)	SSH
✓	workernode01	northamerica-northeast2-a			10.188.0.3 (nic0)	34.130.192.97 (nic0)	SSH
✓	workernode02	northamerica-northeast2-a			10.188.0.4 (nic0)	34.130.18.29 (nic0)	SSH
✓	workernode03	northamerica-northeast2-a			10.188.0.5 (nic0)	34.130.191.166 (nic0)	SSH
✓	workernode04	northamerica-northeast2-a			10.188.0.6 (nic0)	34.130.213.45 (nic0)	SSH

Gambar 3.2 VM (*server*) pada *Google Cloud Platform*

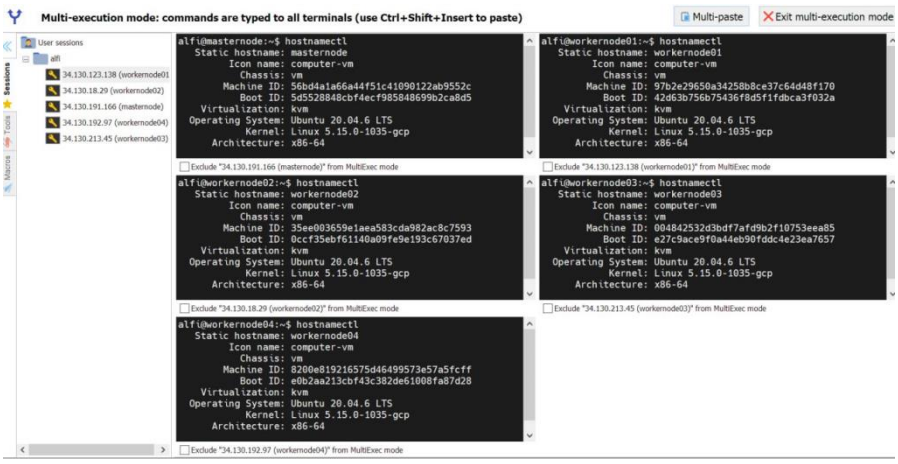
Konfigurasi SSH *key* pada *master node* dan *worker node* agar seluruh *server* dapat diakses secara *remote* dari jaringan luar (*klien*). Konfigurasi SSH dilakukan

menggunakan software PuTTY *Key Generator*. Setelah di-*generate* dan PuTTY menampilkan *public key* seperti yang terlihat pada Gambar 3.3, *public key* tersebut disalin ke pengaturan di *Google Cloud Platform* pada masing-masing *server* dan juga simpan sebagai *private key*. Setiap *server* memiliki *public key* yang berbeda.



Gambar 3.3 *Generate* SSH *key* untuk *remote server*

Untuk melakukan *remote* kelima server dari *Google Cloud Platform*, disini menggunakan *software* MobaXterm. Penggunaan *software* MobaXterm dilakukan karena memiliki fitur *multiple-execute* yang dapat mempermudah dalam konfigurasi pada server. Tampilan ketika remote kelima server menggunakan MobaXterm ditunjukkan pada Gambar 3.4.



Gambar 3.4 Tampilan *remote server* dengan MobaXterm

3.2.2.2 *Kubernetes Cluster*

Untuk membentuk Kubernetes *cluster* memiliki beberapa tahapan konfigurasi pada *master node* dan *worker node*. Tahapan awal dimulai dengan konfigurasi *upgrade* paket, instalasi *runtime* containerd, instalasi kubectl, kubelet, dan kubeadm, hingga *join* (menggabungkan) *worker node* ke *master no*de pada Kubernetes *cluster*.

A. Konfigurasi Pada Semua Node

Pada tahap ini konfigurasi dilakukan di seluruh node Kubernetes *cluster*, yaitu pada *master no*de, *worker node* 1, *worker node* 2, *worker node* 3, dan *worker node* 4. Beberapa hal yang di konfigurasi seperti *upgrade* paket, instalasi *runtime containerd*, pengaturan kernel, dan instalasi kubectl, kubelet, dan kubeadm.

1) Konfigurasi *Upgrade* Paket

Konfigurasi ini dilakukan untuk meng-*update* dan meng-*upgrade* paket yang telah terinstal pada sistem operasi Ubuntu server 20.04. Perintah yang digunakan untuk melakukan konfigurasi *upgrade* paket yaitu sebagai berikut.

```
alfi@workernode:~$ sudo apt update
alfi@workernode:~$ sudo apt upgrade -y
alfi@workernode:~$ sudo apt autoremove -y
```

2) Instalasi *Runtime* Containerd

Sebelum melakukan instalasi containerd itu sendiri, terdapat beberapa tahap konfigurasi paket yang harus dilakukan. Perintah yang digunakan untuk melakukan menginstal paket dependensi Kubernetes yaitu curl dan transport-https sebagai berikut.

```
alfi@workernode:~$ sudo apt install -y curl gnupg2 software-properties-common apt-
transport-https ca-certificates
```

Selanjutnya mengaktifkan *repository* dari Docker menggunakan perintah berikut.

```
alfi@workernode:~$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo gpg --dearmour -o /etc/apt/trusted.gpg.d/docker.gpg
alfi@workernode:~$ sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

Mengupdate paket karena repository baru telah ditambahkan dan juga untuk menginstal containerd menggunakan perintah sebagai berikut.

```
alfi@workernode:~$ sudo apt update
alfi@workernode:~$ sudo apt install -y containerd.io
```

Melakukan konfigurasi containerd agar mulai menggunakan *system* sebagai *control group* (*cgroup*). Di Linux, *cgroup* digunakan untuk membatasi sumber daya yang dialokasikan ke proses. Ketika *systemd* dipilih sebagai sistem init untuk distribusi Linux, proses init menghasilkan dan menggunakan *root control group* (*cgroup*) dan bertindak sebagai manajer *cgroup*. Perintah yang digunakan untuk konfigurasi containerd sebagai berikut.

```
alfi@workernode:~$ containerd config default | sudo tee /etc/containerd/config.toml
>/dev/null 2>&1
alfi@workernode:~$ sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g'
/etc/containerd/config.toml
```

Melakukan *restart node* lalu mengaktifkan layanan containerd menggunakan perintah sebagai berikut.

```
alfi@workernode:~$ sudo systemctl restart containerd
alfi@workernode:~$ sudo systemctl enable containerd
```

3) Konfigurasi Pengaturan Kernel

Menambahkan pengaturan kernel. Modul kernel overlay dan br_netfilter dimuat untuk mengaktifkan modul pada kernel Linux itu dan berfungsi untuk komunikasi pada Kubernetes cluster. Perintah yang digunakan untuk konfigurasi pengaturan kernel yaitu sebagai berikut.

```
alfi@workernode:~$ cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF
alfi@workernode:~$ sudo modprobe overlay
alfi@workernode:~$ sudo modprobe br_netfilter
```

Malakukan konfigurasi sistem jaringan Kubernetes dengan parameter *net bridge ip tables*, *bridge ipv6 tables*, dan *ipv4 forward*. Di Linux, antarmuka *sysctl* memungkinkan administrator untuk memodifikasi parameter kernel saat *runtime*. Perintah untuk melakukan konfigurasi sistem jaringan Kubernetes yaitu sebagai berikut.

```
alfi@workernode:~$ cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF
```

4) Instalasi kubectl, kubelet, dan kubeadm

Paket kubectl, kubelet, dan kubeadm diinstal pada Kubernetes *cluster* agar *cluster* dapat berjalan. Kubectl berfungsi untuk berinteraksi dengan *cluster* di Kubernetes. Kubelet berfungsi untuk *bootstrap* cluster di Kubernetes. Dan kubeadm berfungsi untuk menjalankan pod dan *kontainer* dan berjalan di semua mesin pada Kubernetes *Cluster*. Menerapkan (*apply*) pada sistem dengan membaca dan memodisikasi parameter kernel sistem tanpa me*restart* (*reboot*) sistem menggunakan perintah berikut.

```
alfi@workernode:~$ sudo sysctl --system
```

Menginstal paket transport-https dan paket dependensi dari *Google Cloud Platform* untuk menjalankan Kubernetes *cluster*. Lalu mengupdate paket yang telah ditambahkan. Setelah menginstal paket dependensi, selanjutnya menginstal paket kubectl, kubelet, dan kubeadm, serta melakukan konfigurasi menghapus tanda paket perangkat lunak yang diinstal secara otomatis dengan opsi tahan (*hold*) yang akan memblokir paket agar tidak diinstal, ditingkatkan, atau dihapus untuk terus berjalan pada aplikasi kubectl, kubelet, dan kubeadm. Perintah yang digunakan untuk melakukan install paket transport-https hingga melakukan opsi *hold* menggunakan perintah sebagai berikut.

```
alfi@workernode:~$ sudo apt install -y apt-transport-https; curl -s
https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
echo 'deb http://apt.kubernetes.io/ kubernetes-xenial main' > kubernetes.list

alfi@workernode:~$ sudo mv kubernetes.list /etc/apt/sources.list.d/kubernetes.list
alfi@workernode:~$ sudo apt update; sudo apt install -y kubectl kubelet kubeadm
alfi@workernode:~$ sudo apt-mark hold kubelet kubeadm kubectl
```

B. Konfigurasi Pada *Master Node*

Melakukan inisialisasi master dengan men-*stop* (menghentikan) *swap* terlebih dulu menggunakan perintah berikut.

```
alfi@masternode:~$ swapon -s
alfi@masternode:~$ sudo swapoff -a
alfi@masternode:~$ sudo kubeadm init --pod-network-cidr=10.244.XX.0/16
```

Membuat direktori *kube* pada direktori *Home* dan menyalin file admin.conf ke direktori kube/config dengan mengubah pengaturan *owner* untuk direktori *config*. Perintah yang digunakan untuk membuat direktori *kube* yaitu sebagai berikut.

```
alfi@masternode:~$ mkdir -p $HOME/.kube
alfi@masternode:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Menginstal paket *Container Network Interface* (CNI) Kubernetes yaitu Flannel yang digunakan untuk komunikasi antar *host* pada Kubernetes *cluster* menggunakan perintah sebagai berikut.

```
alfi@masternode:~$ wget
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-
flannel.yml
alfi@masternode:~$ kubectl apply -f kube-flannel.yml
alfi@masternode:~$ kubectl get pods --all-namespaces --watch
```

Melakukan verifikasi seluruh konfigurasi yang telah dilakukan dan verifikasi *cluster* yang telah dibuat menggunakan perintah sebagai berikut.

```
alfi@masternode:~$ kubectl config view
alfi@masternode:~$ kubectl cluster-info
```

C. Konfigurasi Worker Node Join Master Node

Menggabungkan *worker node* ke dalam *cluster* dengan men-*stop swap* terlebih dulu. Token disini bersifat rahasia dan digunakan untuk otentikasi *feedback* antara *node control plane* dan *node* yang bergabung. Perintah untuk melakukan *worker node join master node* yaitu sebagai berikut.

```
alfi@workernode:~$ swapon -s
alfi@workernode:~$ sudo swapoff -a
alfi@workernode:~$ sudo kubeadm join --token [TOKEN] [NODE-MASTER]:6443 --
discovery-token-ca-cert-hash sha256:[TOKEN-CA-CERT-HASH]
```

Melakukan verifikasi *node* yang ada di dalam Kubernetes *cluster* menggunakan perintah sebagai berikut.

```
alfi@masternode:~$ kubectl get nodes
```

3.2.2.3 *Kong Ingress Controller*

Pada instalasi Kong ingress controller, langkah pertama dilakukan dengan menginstal Kong pada Kubernetes menggunakan *tools* Kustomize. Perintah gambar 3.21 merupakan perintah untuk menambahkan kustomisasi dasar Kong untuk dibangun di atasnya dan menjadikan *suite* yang lebih baik dan *use-case*.

```
alfi@masternode:~$kustomize build github.com/kong/kubernetes-ingress-controller/config/base
```

Setelah Kong dibangun dengan Kustomize, atur variable *environment* dengan perintah sebagai berikut, $PROXY_IP dengan alamat IP Eksternal dari *kong-proxy service* di *namespace* kong.

```
alfi@masternode:~$export PROXY_IP=$(kubectl get -o jsonpath="{.status.loadBalancer.ingress
[0].ip}" service -n kong kong-proxy)
```

Install Kong dengan menggunakan *tools* helm dari *official repository* Helm *chart*, dan *update repository* dengan perintah sebagai berikut.

```
alfi@masternode:~$helm repo add kong https://charts.konghq.com
alfi@masternode:~$helm repo update
alfi@masternode:~$helm install kong/kong --generate-name --set
ingressController.installCRDs=false -n kong --create-namespace
```

Setelah Kong diinstal, atur variable *environment* dengan perintah sebagai berikut, $PROXY_IP dengan alamat IP Eksternal dari *kong-proxy service* di *namespace* kong.

```
alfi@masternode:~$export PROXY_IP=$(kubectl get -o jsonpath="{.status.loadBalancer.ingress
[0].ip}" service -n kong kong-proxy)
```

Lakukan tes menggunakan curl dengan menggunakan perintah berikut. Jika variabel environment PROXY_IP telah berhasil diatur untuk memuat alamat IP atau URL yang mengarah ke Kong Gateway, akan memberikan output dengan kode status HTTP 404 Not Found.

```
alfi@masternode:~$curl -i $PROXY_IP
```

Men-*deploy* aplikasi upstream untuk melakukan permintaan proxy dengan perintah sebagai berikut. Deploy server echo ini menyediakan aplikasi sederhana yang mengembalikan informasi tentang pod yang sedang dijalankannya.

```
alfi@masternode:~$echo "
apiVersion: v1
kind: Service
metadata:
  labels:
    app: echo
  name: echo
spec:
  ports:
  - port: 1025
    name: tcp
    protocol: TCP
    targetPort: 1025
  - port: 1026
    name: udp
    protocol: TCP
    targetPort: 1026
  - port: 1027
    name: http
    protocol: TCP
    targetPort: 1027
  selector:
    app: echo
---
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: echo
  name: echo
spec:
  replicas: 4
  selector:
    matchLabels:
      app: echo
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: echo
    spec:
      containers:
      - image: kong/go-echo:latest
        name: echo
        ports:
        - containerPort: 1027
        env:
          - name: node_alfi
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
          - name: pod_alfi
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: kong
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
          - name: POD_IP
            valueFrom:
              fieldRef:
                fieldPath: status.podIP
        resources: {}
" | kubectl apply -f -
```

Ingress controller memerlukan konfigurasi yang menunjukkan rangkaian konfigurasi routing mana yang harus mereka kenali dan memungkinkan banyak controller untuk hidup berdampingan di cluster yang sama. Sebelum membuat rute individual, perlu membuat konfigurasi *class* untuk mengaitkan rute dengan perintah konfigurasi. Distribusi resmi Kong *Ingress Controller* hadir dengan kong IngressClass secara *default*. Perintah yang digunakan untuk konfigurasi membuat rute individual yaitu sebagai berikut.

```
alfi@masternode:~$echo "
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: kong
spec:
  controller: ingress-controllers.konghq.com/kong
" | kubectl apply -f -
```

Membuat konfigurasi *routing* ke *proxy* atau permintaan *echo* ke *echo server* dengan perintah konfigurasi sebagai berikut.

```
alfi@masternode:~$echo "
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: echo
  annotations:
    konghq.com/strip-path: 'true'
spec:
  ingressClassName: kong
  rules:
  - host: kong.alfi.example
    http:
      paths:
      - path: /echo
        pathType: ImplementationSpecific
        backend:
          service:
            name: echo
            port:
              number: 1027
" | kubectl apply -f -
```

Setelah melakukan konfigurasi routing ke proxy, lakukan pengecekan proxy. Jika semua di-deploy dengan benar, akan tampil dengan output "you are connected to the node …". Di mana ini memverifikasi bahwa Kong *Gateway* dapat merutekan trafik dengan benar ke aplikasi yang berjalan di dalam Kubernetes. Perintah yang digunakan untuk mengecek proxy adalah sebagai berikut.

```
alfi@masternode:~$curl -i http://kong.alfi.example/echo --resolve kong.alfi.example:80:$PROXY_IP
```

Membuat pengujian certificate untuk hostname kong.example dengan perintah sebagai berikut.

```
alfi@masternode:~$openssl req -subj '/CN=kong.alfi.example' -new -newkey rsa:2048 -sha256 \
  -days 365 -nodes -x509 -keyout server.key -out server.crt \
  -addext "subjectAltName = DNS:kong.alfi.example" \
  -addext "keyUsage = digitalSignature" \
  -addext "extendedKeyUsage = serverAuth" 2> /dev/null;
  openssl x509 -in server.crt -subject -noout
```

Membuat Secret yang berisi certificate dan key certificate dengan perintah sebagai berikut.

```
alfi@masternode:~$kubectl create secret tls kong.alfi.example --cert=./server.crt --key=./server.key
```

Meng-update konfigurasi routing dengan menggunakan certificate dan secret dengan perintah sebagai berikut.

```
alfi@masternode:~$kubectl patch --type json ingress echo -p='[{
    "op":"add",
        "path":"/spec/tls",
        "value":[{
        "hosts":["kong.alfi.example"],
                "secretName":"kong.alfi.example"
    }]
}]'
```

Permintaan/request akan melayani certificate yang telah dikonfigurasi dengan perintah sebagai berikut.

```
alfi@masternode:~$curl -ksv https://kong.alfi.example/echo --resolve kong.alfi.example:443:$PROXY_IP
2>&1 | grep -A1 "certificate:"
```

Menyiapkan dan menambahkan sumber daya (resource) dari Kong Plugin dengan perintah sebagai berikut.

```
alfi@masternode:~$echo "
apiVersion: configuration.konghq.com/v1
kind: KongPlugin
metadata:
  name: request-id
config:
  header_name: my-request-id
  echo_downstream: true
plugin: correlation-id
" | kubectl apply -f -
```

Mengupdate konfigurasi rute untuk menggunakan plugin baru dengan perintah sebagai berikut.

```
alfi@masternode:~$kubectl annotate ingress echo konghq.com/plugins=request-id
```

Membuat sumber daya (resource) Kong plugin. Kong dapat menerapkan plugin ke service yang memungkinkan menjalankan konfigurasi plugin yang sama pada semua request ke service tersebut tanpa mengonfigurasi plugin yang sama pada beberapa ingress. perintah yang digunakan untuk membuat sumber daya Kong plugin yaitu sebagai berikut.

```
alfi@masternode:~$echo "
apiVersion: configuration.konghq.com/v1
kind: KongPlugin
metadata:
  name: rl-by-ip
config:
  minute: 5
  limit_by: ip
  policy: local
plugin: rate-limiting
" | kubectl apply -f -
```

Menerapkan anotasi Kong plugin ke service Kubernetes dengan memberikan rate-limiting (pembatasan rate) dengan perintah sebagai berikut.

```
alfi@masternode:~$kubectl annotate service echo konghq.com/plugins=rl-by-ip
```

Membuat pembatasan *rate* (*rate limit*) utnuk permintaan *proxy* ke *service* ini dengan perintah sebagai berikut.

```
alfi@masternode:~$curl -i http://kong.alfi.example/echo --resolve kong.alfi.example:80:$PROXY_IP
```

3.2.2.4 *Istio Ingress Controller*

Pada instalasi Istio *ingress controller*, langkah pertama yaitu dengan menginstal tools istioctl yang berfungsi untuk perintah konfigurasi dari istio,

perintah install istioctl menggunakan perintah sebagai berikut dengan men-*download*nya dari repository istio.

```
alfi@masternode:~$curl -L https://istio.io/downloadIstio | sh -
```

Setelah men-download istioctl dari repository, ekstrak paket tersebut, dan masuk ke dalam direktori istio. Lalu install istioctl dengan menggunakan konfigurasi profil demo dan menambahkan label namespace untuk menginstruksikan istio agar secara otomatik meng-injeksi proxy Envoy sidecar saat menerapkan aplikasi nanti. Perintah yang digunakan untuk menginstall istioctl adalah sebagai berikut.

```
alfi@masternode:~$cd istio-1.18.0
alfi@masternode:istio-1.18.0$istioctl install --set profile=demo -y
alfi@masternode:istio-1.18.0$kubectl label namespace default istio-injection=enabled
```

Mengecek daftar service dengan nama istio-ingressgateway, akan tampil juga daftar port yang tersedia dengan service istio-ingressgateway. Perintah yang digunakan untuk mengecek daftar service yaitu sebagai berikut.

```
alfi@masternode:~$kubectl get svc istio-ingressgateway -n istio-system
alfi@masternode:~$kubectl describe svc istio-ingressgateway -n istio-system |grep http2
```

Membuat file deployment untuk membuat aplikasi web server Nginx, file konfigurasi disimpan dengan nama my-nginx.yml yang berisi konfigurasi dengan perintah berikut.

```
alfi@masternode:~$sudo nano my-nginx.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: webserver
  name: my-nginx
  namespace: my-namespace
spec:
  replicas: 4
  selector:
    matchLabels:
      app: webserver
  template:
    metadata:
      labels:
        app: webserver
    spec:
      containers:
      - image: nginx
        name: my-nginx
        ports:
        - containerPort: 80
```

Membuat service untuk deployment, file konfigurasi disimpan dengan nama my-nginx-service.yml yang berisi konfigurasi dengan sebagai berikut.

```
alfi@masternode:~$sudo nano my-nginx-service.yml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: my-nginx
  name: webserver
  namespace: my-namespace
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: webserver
  type: ClusterIP
```

Membuat konfigurasi *ingress gateway* untuk *service*, file konfigurasi disimpan dengan nama my-nginx-gateway.yml yang berisi konfigurasi dengan perintah sebagai berikut.

```
alfi@masternode:~$sudo nano my-nginx-gateway.yml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: my-nginx-gateway
  namespace: my-namespace
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
      - "alfi.example.com"
```

Membuat konfigurasi virtual service untuk ingress gateway, file konfigurasi disimpan dengan nama my-nginx-virtualservice.yml yang berisi konfigurasi dengan perintah sebagai berikut.

```
alfi@masternode:~$sudo nano my-nginx-virtualservice.yml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: my-nginx-virtualservice
  namespace: my-namespace
spec:
  hosts:
  - "alfi.example.com"
  gateways:
  - my-nginx-gateway
  http:
  - match:
    - uri:
        prefix: /
    route:
    - destination:
        port:
          number: 80
        host: webserver
```

Membuat *namespace* untuk aplikasi dengan nama my-namespace dan mengaktifkan injeksi *proxy sidecar* otomatis dengan perintah sebagai berikut.

```
alfi@masternode:~$kubectl create namespace my-namespace
alfi@masternode:~$kubectl label namespaces my-namespace istio-injection=enabled
```

Meng-*apply* semua konfigurasi *file* yml yang telah dibuat dengan perintah sebagai berikut.

```
alfi@masternode:~$kubectl apply -f my-nginx.yml
alfi@masternode:~$kubectl apply -f my-nginx-service.yml
alfi@masternode:~$kubectl apply -f my-nginx-gateway.yml
alfi@masternode:~$kubectl apply -f my-nginx-virtualservice.yml
```

Setelah konfigurasi diterapkan (di-*apply*), maka dapat mengecek *ingress gateway* dan *virtual service* yang sedang berjalan (*running*), serta mengkonfirmasi bahwa *ingress gateway* melayani aplikasi *loadbalancer*. Perintah yang digunakan untuk mengecek *ingress* sebagai berikut.

```
alfi@masternode:~$kubectl get gateways.networking.istio.io -n my-namespace
alfi@masternode:~$kubectl get virtualservices.networking.istio.io -n my-namespace
```

3.2.3 Perancangan Parameter

Perancangan parameter dilakukan dengan mencari parameter yang sesuai untuk pengambilan data berupa performansi jaringan dan performansi hardware. Parameter tersebut didapatkan dengan mempertimbangkan parameter yang digunakan pada jurnal acuan. Maka didapatkan parameter yaitu berupa *delay*, *throughput*, *CPU usage*, dan *memory usage* yang tertulis di tabel 3.4.

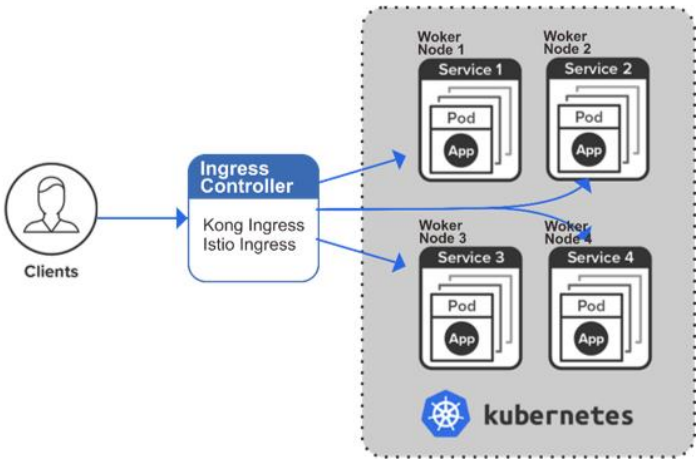
Tabel 3.4 Parameter Pengujian

Tools	Parameter Penelitian	Satuan
Wireshark	*Delay*	ms
	Throughput	Kbps
Htop	CPU *usage*	%
	Memory *usage*	MB

3.2.4 Proses Pengujian

Proses pengujian pada penelitian ini dimualai dengan menyiapkan system yang akan dugunakan untuk simulasi yaitu berupa mesin virtual pada *Google Cloud Platform* yang akan di *install* 5 sistem operasi Ubuntu Server 20.04 yang akan dijadikan sebuah Kubernetes *cluster*. Kelima sistem operasi tersebut berfungsi

sebagai Kubernetes *cluster* dengan *role* yaitu 1 *master node* dan 4 *worker node*. Ketika sudah selesai membuat sebuah *cluster* maka akan dilanjutkan dengan melakukan *deployment* aplikasi *web server* yaitu *Nginx* yang akan dapat diakses oleh *klien* melalui *ingress controller*. Proses pengujian akan dilakukan secara bergantian pada *ingress controller* menggunakan Kong *ingress controller* dan Istio *ingress controller*. Topologi Kubernetes *cluster* yang digunakan untuk pengujian dapat dilihat pada gambar 3.5.



Gambar 3.5 Topologi Kubernetes *Cluster*

3.2.5 Pengambilan Data

Pada tabel 3.5, proses pengambilan data dilakukan sebanyak 30 kali yang bertujuan agar data tersebut dapat dikatakan benar dan dapat dianalisis secara tepat. Pengambilan data menggunakan dua jenis aplikasi yaitu *Wireshark* dan *Htop*, dimana *Wireshark* berfungsi untuk mengambil data *Delay* dan *Throughput* sedangkan *Htop* berfungsi untuk mengambil data *Memory* dan CPU.

Tabel 3.5 Pengambilan Data

Parameter Penelitian	Banyaknya Pengujian	Waktu Pengambilan Data
Delay	30	60 *second*
Throughput	30	60 *second*
Memory	30	60 *second*
CPU	30	60 *second*

Pada pengambilan data, skenario jumlah klien yaitu 50, 150, 250, 350, 450, 550, 650, dan 750 *klien* disimulasikan menggunakan software Siege yang diinstal

pada Ubuntu klien, sehingga dapat merepresentasikan jumlah klien yang sebenarnya. Perintah pada Siege yang digunakan seperti pada gambar 3.6. Option -c berarti untuk jumlah klien, -t berarti untuk lama waktu simulasi, dan terhubung ke IP external master node dan port yang digunakan yaitu 30965. Pada Siege juga dapat menghasilkan output beberapa parameter lain seperti throughput, namun dalam penelitian ini parameter throughput diambil datanya menggunakan Wireshark.

Gambar 3.6 *Tools* Siege

3.2.5.1 Htop

Software Htop digunakan untuk mengambil data CPU *usage* dan *memory usage*. Langkah pertama dengan meng*install* Htop pada sistem operasi Ubuntu dengan menggunakan perintah sebagai berikut.

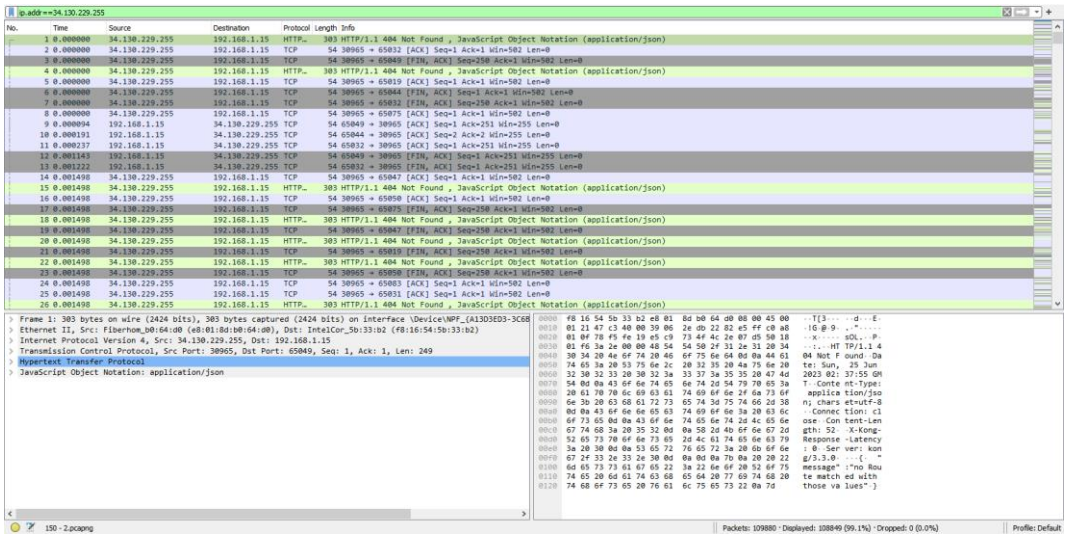
```
alfi@masternode:~$sudo apt-get install htop
```

Pada Gambar 3.7 merupakan output CPU pada Htop. Htop dijalankan hanya menggunakan perintah “htop”, lalu akan menampilkan output dari CPU dan memory. Output CPU ditandai dengan angka 1 dan 2, hasilnya didapatkan dengan mengambil nilai terbesar yang diamati selama 60 detik. *Output memory* ditandai dengan “Mem” di mana satuannya yaitu MB, hasilnya didapatkan dengan mengambil nilai terbesar yang diamati selama 60 detik.

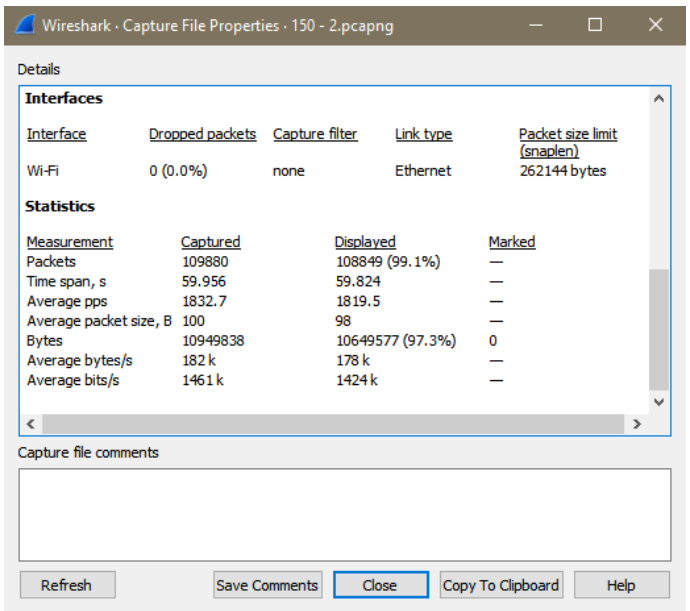
Gambar 3.7 *Output* CPU menggunakan Htop

3.2.5.2 Wireshark

Software Wireshark digunakan untuk mengambil data delay dan throughput. Langkah pertama dengan menginstall Wireshark pada sistem operasi klien, lalu trace melalui jaringan WiFi bersamaan dengan dijalankannnya Siege sesuai dengan jumlah klien. Gambar 3.8 merupakan output ketika trace jaringan menggunakan Wireshark. Dan gambar 3.9 merupakan hasil data yang diperoleh setelah proses trace selesai di mana data tersebut akan diolah untuk mendapatkan nilai delay dan throughput.



Gambar 3.8 Pengambilan data dengan Wireshark



Gambar 3.9 Hasil pengambilan data dengan Wireshark

3.2.6 Analisis Data

Analisis data merupakan proses yang terakhir setelah pengambilan data QoS berupa *Delay, Throughput, Memory* dan CPU. Pengambilan data QoS untuk parameter *Delay* dan *Throughput* dilakukan menggunakan *Software Wireshark* sedangkan untuk penggunaan *Memory* dan CPU menggunakan Aplikasi *Htop*. Data *Delay, Throughput, Memory* dan CPU akan dihitung menggunakan rumus yang terdapat pada dasar teori dan akan dianalisa untuk setiap percobaan yang dilakukan yaitu sebanyak 30 kali. Analisis yang dilakukan yaitu dengan membandingkan hasil keempat parameter tersebut.