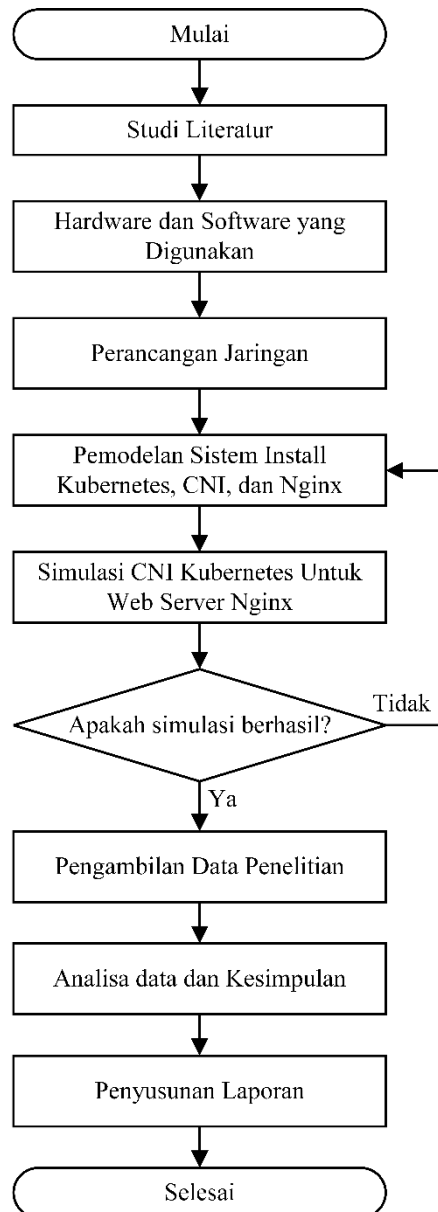


## BAB 3 METODE PENELITIAN

### 3.1 ALUR PENELITIAN

Penelitian yang akan dilakukan terdiri dari beberapa tahapan, mulai dari studi literatur hingga tahap penyusunan laporan. Gambar 3.1 adalah diagram alur yang digunakan pada penelitian ini.



Gambar 3.1 Alur Penelitian

### 3.2 ANALISIS KEBUTUHAN

Tahap analisis kebutuhan digunakan untuk menentukan kebutuhan apa saja yang diperlukan untuk menunjang penelitian. Penelitian ini membutuhkan dua jenis perangkat, yaitu perangkat keras (*hardware*) dan perangkat lunak (*software*). Pemilihan perangkat keras dan perangkat lunak didalam penelitian ini diambil berdasarkan hasil *reserch* kebutuhan minimum dalam Kubernetes *cluster* dan aplikasi *web server Nginx*, serta untuk *tools* pengujian yang digunakan berdasarkan hasil *reserch* aplikasi populer untuk melakukan *test* pengujian.

#### 3.2.1 Hardware dan Software yang Digunakan

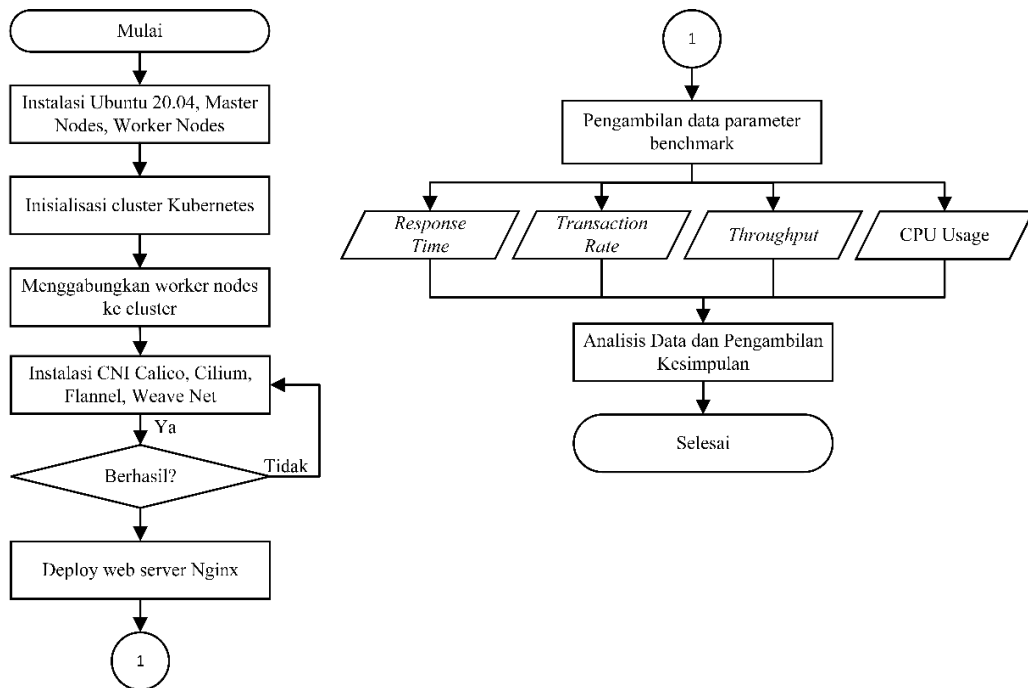
Perangkat-perangkat yang digunakan untuk membuat simulasi dalam penelitian ini terdiri atas perangkat keras (*hardware*) dan perangkat lunak (*software*) untuk *master nodes*, *worker nodes*, *web server*, dan *benchmark tools*. Tabel 3.1 adalah spesifikasi *hardware* dan *software* yang digunakan dalam penelitian ini.

Tabel 3.1 Spesifikasi *Hardware* dan *Software* yang Digunakan

Komponen	Spesifikasi	Keterangan
<i>Master Nodes dan Worker Nodes</i>	OS	<i>Ubuntu Server 20.04</i>
	CPU	2 vCPU
	RAM	8 GB
	<i>Tool dan Aplikasi</i>	1. <i>Kubelet version 1.22.1</i> 2. <i>Kubectl version 1.22.1</i> 3. <i>Kubeadm version 1.22.1</i> 4. <i>CNI Calico</i> 5. <i>CNI Flannel</i> 6. <i>CNI Cilium</i> 7. <i>CNI Weave Net</i>
<i>Web Server</i>	<i>Version</i>	<i>Nginx version 1.18.0</i>
<i>Benchmark tools</i>	<i>Apps</i>	<i>Siege version 4.1.3</i> <i>Htop version 2.2.0</i>

### 3.3 PERANCANGAN JARINGAN

Perancangan jaringan pada penelitian ini menggunakan *CNI Calico*, *Cilium*, *Flannel*, dan *Weave Net* untuk trafik *web server Nginx* dengan diagram alur perancangan program sebagai berikut. Gambar 3.2 merupakan alur perancangan jaringan yang digunakan pada penelitian ini.



Gambar 3.2 Alur Perancangan Jaringan

### 3.3.1 Perancangan Skenario

Skenario yang digunakan pada penelitian ini yaitu penggunaan CNI yang berbeda yaitu *Calico*, *Cilium*, *Flannel*, dan *Weave Net*. Skenario penelitian dimaksudkan untuk mengetahui pengaruh penggunaan CNI yang berbeda-beda untuk trafik *web server Nginx*. Jenis komunikasi yang digunakan yaitu *pod ke pod*, *pod ke service*, dan *client ke service*. Pada Tabel 3.2 adalah skenario yang digunakan dalam penelitian ini.

Tabel 3.2 Skenario Penelitian

Skenario	CNI	Jenis Komunikasi	Jumlah <i>simulated user</i>
1	<i>Calico</i>	<i>Pod ke pod</i>	10, 100, 200
		<i>Pod ke service</i>	
		<i>Client ke service</i>	
2	<i>Cilium</i>	<i>Pod ke pod</i>	10, 100, 200
		<i>Pod ke service</i>	
		<i>Client ke service</i>	
3	<i>Flannel</i>	<i>Pod ke pod</i>	10, 100, 200
		<i>Pod ke service</i>	
		<i>Client ke service</i>	
4	<i>Weave Net</i>	<i>Pod ke pod</i>	10, 100, 200
		<i>Pod ke service</i>	
		<i>Client ke service</i>	

Skenario pertama yang digunakan pada penelitian ini yaitu penggunaan CNI *Calico* dengan komunikasi *pod* ke *pod*, *pod* ke *service*, dan *client* ke *service*. Skenario pertama ini bermaksud untuk mengetahui pengaruh CNI *Calico* yang digunakan terhadap parameter yang diambil. Skenario kedua, ketiga dan keempat yaitu penggunaan CNI *Cilium*, *Flannel* dan *Weave Net* dengan komunikasi sama yaitu *pod* ke *pod*, *pod* ke *service*, dan *client* ke *service*.

### 3.3.2 Perancangan Parameter

Parameter yang akan dianalisis pada penelitian ini yaitu *response time*, *transaction rate*, *throughput*, dan *CPU usage*. Pada Tabel 3.3 adalah parameter simulasi yang digunakan dalam penelitian ini.

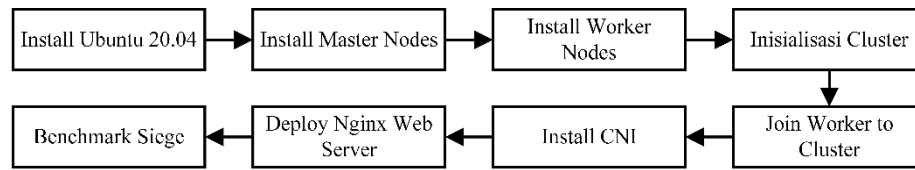
Tabel 3.3 Parameter Simulasi

Parameter Simulasi	Nilai
Waktu simulasi	60 detik
Jumlah <i>simulated user</i>	10, 100, 200

### 3.4 PEMODELAN SISTEM

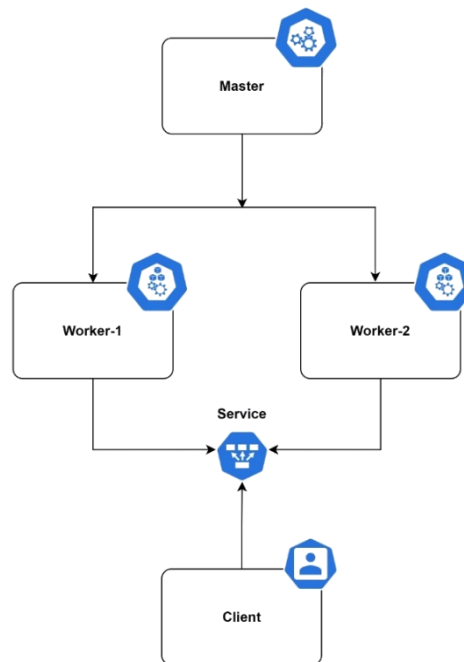
Pada tahap ini, akan dilakukan pembuatan simulasi yang dijalankan dengan skenario yang dirancang seperti pada Gambar 3.2. Langkah awal yaitu mempersiapkan *hardware* dan *software* yang digunakan untuk proses simulasi. Pada proses simulasi pertama meng-*install* sistem operasi *Ubuntu Server 20.04* yang digunakan sebagai *server* pada *google cloud platform*. Selanjutnya melakukan instalasi *master nodes* dan *worker nodes* pada Kubernetes. Lalu menginisialisasi *cluster* pada *master nodes* untuk mengumpulkan *nodes* yang akan menjalankan aplikasi dalam kontainer. Setelah *cluster* diinisialisasi, *worker nodes* akan digabungkan ke *cluster* Kubernetes. *Install* CNI berfungsi untuk *interface* antar *container* yang akan dijalankan, proses *install* CNI dilakukan secara berganti mulai dari *Calico*, *Flannel*, *Cilium*, *Weave Net*. Pada *deploy Nginx web server* dilakukan proses instalasi *web server Nginx* versi 1.18.0 dan proses pembuatan *service Nginx* pada Kubernetes *cluster*. Proses terakhir simulasi yaitu *benchmark* dengan

menggunakan *tools* Siege untuk mendapatkan hasil dari parameter yang diuji. Gambar 3.3 merupakan diagram blok sistem secara sederhana.



Gambar 3.3 Diagram blok sistem

Pada Gambar 3.4 yaitu desain Kubernetes *cluster* yang dibangun pada simulasi penelitian ini. Kubernetes *cluster* terdiri dari *service*, *master nodes*, dan 2 (dua) *worker nodes*. Kubernetes *cluster* juga sebagai skema skenario. Pada skenario *pod* ke *pod* dilakukan oleh *Worker-1* dan *Worker-2*. Skenario *pod* ke *service* dilakukan oleh pada salah satu *worker nodes* (*Worker-1* ataupun *Worker-2*). Skenario *client* ke *service* dilakukan oleh *Client* dan *Service*.



Gambar 3.4 Kubernetes *cluster*

### 3.4.1 Google Cloud Platform

*Google cloud platform* dapat digunakan secara gratis, dengan cara mendaftarkan akun *Google* yang dimiliki untuk menggunakan fasilitas akun percobaan senilai \$300 atau Rp. 4.392.225 yang dapat digunakan untuk seluruh fitur *Google Cloud Platform*. Proses penyiapan *Google Cloud Platform* dalam

penelitian ini menggunakan fitur *Compute Engine* atau dapat disebut juga mesin virtual *Google*. Sesuai dengan perancangan Kubernetes *cluster* yang akan dibuat yaitu menggunakan 3 buah *instance machine* sesuai dengan konfigurasi pada sub-subbab selanjutnya.

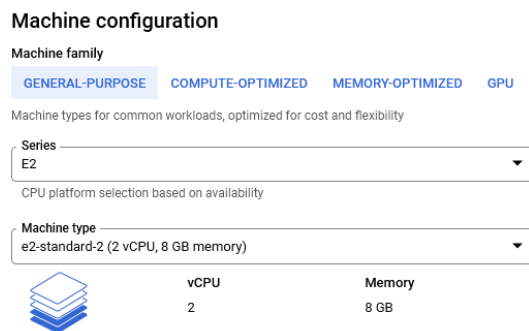
### A. Konfigurasi *Master Node*

Konfigurasi *master node* dengan nama *k8smaster* dengan kategori E2 dan tipe mesin 2 vCPU 8 GB *memory* sesuai pada Gambar 3.6. Konfigurasi *boot disk* menggunakan *image Ubuntu 20.04* dengan kapasitas disk 10GB sesuai pada Gambar 3.7.



Name \*  
k8smaster

Gambar 3.5 Nama *master node*



**Machine configuration**

Machine family


GENERAL-PURPOSE COMPUTE-OPTIMIZED MEMORY-OPTIMIZED GPU

Machine types for common workloads, optimized for cost and flexibility

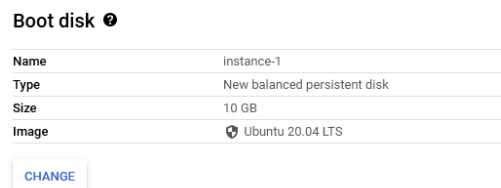
Series  
E2

CPU platform selection based on availability

Machine type  
e2-standard-2 (2 vCPU, 8 GB memory)

	vCPU	Memory
	2	8 GB

Gambar 3.6 Spesifikasi *master node*



**Boot disk**

Name	instance-1
Type	New balanced persistent disk
Size	10 GB
Image	Ubuntu 20.04 LTS

CHANGE

Gambar 3.7 *Boot disk master node*

### B. Konfigurasi *Worker Node*

Konfigurasi *worker node* satu dan *worker node* dua dengan nama *k8sworker01* dan *k8sworker02* sesuai pada Gambar 3.8. dengan kategori E2 dan tipe mesin 2 CPU 8GB *memory* sesuai pada Gambar 3.9. Konfigurasi *boot disk* menggunakan *image Ubuntu 20.04* dengan kapasitas *disk* 10GB sesuai pada Gambar 3.10.

Name \*  
k8sworker01

Name \*  
k8sworker02

Gambar 3.8 Nama *worker node*

**Machine configuration**

Machine family


GENERAL-PURPOSE COMPUTE-OPTIMIZED MEMORY-OPTIMIZED GPU

Machine types for common workloads, optimized for cost and flexibility

Series  
E2

CPU platform selection based on availability

Machine type  
e2-standard-2 (2 vCPU, 8 GB memory)

	vCPU	Memory
	2	8 GB

Gambar 3.9 Spesifikasi *worker node*

**Boot disk**

Name	instance-1
Type	New balanced persistent disk
Size	10 GB
Image	Ubuntu 20.04 LTS

Gambar 3.10 *Boot disk worker node*

### C. Konfigurasi SSH

Konfigurasi SSH *key* pada setiap *master* dan *worker* dilakukan agar seluruh *instance machine* dapat diakses secara *remote* dari jaringan luar (*client*). Konfigurasi dilakukan dengan cara men-*generate* SSH pada seluruh *node* dan menambahkan SSH *key* pada menu *edit* di setiap VM *instance* pada *Google Cloud Console* sesuai pada Gambar 3.11.

SSH key 1 \*  
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgQD2BSuUJaUkGumG121bUJ

Enter public SSH key

Gambar 3.11 SSH *key*

### D. Konfigurasi *firewall* *Google Cloud Console*

Konfigurasi *firewall* pada penggunaan *Google Cloud Console* untuk Kubernetes perlu diketahui, untuk penggunaan aplikasi *Nginx* harus membuat konfigurasi *firewall* baru. Konfigurasi tersebut meliputi *IP Range* yaitu 0.0.0.0/0, *action* menggunakan *allow*, *Protocols and ports* menggunakan *allow all* agar

aplikasi *Nginx* dapat diakses oleh *client* dari luar jaringan. Hasil konfigurasi *firewall* seperti yang terlihat pada Gambar 3.12.

<input type="checkbox"/>	Name	Type	Targets	Filters	Protocols / ports	Action	Priority	Network ↑	Logs	Hit count	Last hit	Insights
<input type="checkbox"/>	allow-all	Ingress	Apply to all	IP ranges: 0.0.0.0/0	all	Allow	1000	default	Off	--	--	

Gambar 3.12 Konfigurasi *firewall*

Ketika konfigurasi *master node*, *worker node*, *ssh*, dan *firewall* selesai maka dapat dilanjutkan ke tahap selanjutnya yaitu konfigurasi Kubernetes.

### 3.4.2 *Kubernetes*

Instalasi Kubernetes mempunyai beberapa tahapan utama yang harus dilakukan secara runtut. Dimulai dari instalasi *containerd*, *kubectl*, *kubelet*, *kubeadm*, *container network interface*, inialisasi *cluster* dan menggabungkan *worker node* ke dalam *cluster* Kubernetes. Proses pertama dimulai dengan pemetaan nama *hosts*. *File* konfigurasi tersebut berisi nama *host* setiap *node* dan diikuti dengan IP eksternalnya.

```
brica@k8smaster:~$ nano /etc/host
brica@k8smaster:~$ cat /etc/host
10.128.0.4 k8smaster
10.128.0.5 k8sworker01
10.162.0.3 k8sworker02
127.0.0.1 localhost
```

#### A. *Install paket containerd*

Instalasi *containerd* berguna untuk menyediakan antarmuka kontainer tingkat tinggi. Proyek perangkat lunak lain dapat menggunakan ini untuk menjalankan kontainer dan mengelola *image* kontainer.

```
brica@k8smaster:~$ cat <<EOF | sudo tee /etc/modules-load.d/containerd.conf
overlay
br_netfilter
EOF
brica@k8smaster:~$ sudo modprobe overlay
brica@k8smaster:~$ sudo modprobe br_netfilter
```



Berguna untuk menambahkan modul kernel yang dimuat pada *Ubuntu* dengan mengaktifkan modul `br_netfilter` untuk instalasi Kubernetes. Mengaktifkan modul *kernel* tersebut sehingga paket-paket yang melewati *bridge* akan di proses terlebih dahulu oleh *iptables* untuk *filtering* dan *port forwarding*. dan membuat *cluster* kubernetes dapat berkomunikasi satu sama lain.

Konfigurasi Kubernetes CRI, sistem ini untuk jaringan Kubernetes yang mengatur parameter `sysctl`, di mana ini akan bernilai tetap setiap kali penjalanan ulang.

```
brica@k8smaster:~$ cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF
```

Dilanjutkan dengan menyimpan konfigurasi yang dibuat pada *runtime Ubuntu*:

```
brica@k8smaster:~$ sudo sysctl --system
```

## **B. *Install containerd***

Melakukan *update* dan instalasi *containerd*.

```
brica@k8smaster:~$ sudo apt-get update && sudo apt-get install -y containerd
```

Menambahkan *directory* untuk *containerd* dilanjutkan dengan konfigurasi menyediakan *containerd*, beberapa konfigurasi dan mengatur entri *systemd* sehingga kita dapat memulainya secara otomatis saat *boot*. *Containerd* memiliki perintah praktis untuk menghasilkan konfigurasi *default*, jadi kita bisa menggunakannya.

```
brica@k8smaster:~$ sudo mkdir -p /etc/containerd
brica@k8smaster:~$ sudo containerd config default | sudo tee
/etc/containerd/config.toml
```

Melakukan perintah *restart containerd* dan mengaktifkan *containerd*.

```
brica@k8smaster:~$ sudo systemctl restart containerd
brica@k8smaster:~$ sudo systemctl enable containerd
```

Menonaktifkan SWAP

```
brica@k8smaster:~$ sudo swapoff -a
brica@k8smaster:~$ sudo sed -i ' swap / s/^(.*)$/#\1/g' /etc/fstab
```

Meng-*install* paket ketergantungan dan mencoba mengakses *packages cloud google*.

```
brica@k8smaster:~$ sudo apt update && sudo apt-get install -y apt-transport-https
curl
brica@k8smaster:~$ curl -s https://packages.cloud.google.com/apt/doc/apt-
key.gpg | sudo apt-key add -
```

Menambahkan *repository* Kubernetes dan meng-*update*-nya.

```
brica@k8smaster:~$ cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
sudo apt update
```

### C. *Install kubectl, kubelet, & kubeadm packages*

Meng-*install* *kubelet* versi 1.22.1, *kubeadm* versi 1.22.1, dan *kubectl* versi 1.22.1.

```
brica@k8smaster:~$ sudo apt-get install -y kubelet=1.22.1-00 kubeadm=1.22.1-00
kubectl=1.22.1-00
```

Perintah *apt-mark* akan menandai atau menghapus tanda paket perangkat lunak sebagai yang di-*install* secara otomatis dan digunakan dengan opsi tahan sehingga akan memblokir paket agar tidak di-*install*, ditingkatkan, atau dihapus.

```
brica@k8smaster:~$ sudo apt-mark hold kubelet kubeadm kubectl
```

#### D. Inisialisasi Cluster

Buat sebuah berkas bernama `kubeadm-config.yaml` yang digunakan untuk mengatur *node control plane* pertama dan membuat *certs* untuk *join* ke dalam *cluster*.

```
brica@k8smaster:~$ sudo nano kubeadm-config.yaml
brica@k8smaster:~$ cat kubeadm-config.yaml
apiVersion: kubeadm.k8s.io/v1beta2
kind: ClusterConfiguration
kubernetesVersion: stable
controlPlaneEndpoint: "fi-k8s-vrrp-master:6443"
networking:
  podSubnet: "10.244.0.0/16"
brica@k8smaster:~$ kubeadm init --config=kubeadm-config.yaml --upload-certs
```

Perintah untuk bergabung yang didapat dari keluaran ke dalam sebuah berkas teks untuk digunakan nantinya atau dapat dilihat ulang menggunakan perintah:

```
brica@k8smaster:~$ kubeadm token create --print-join-command
kubeadm join 10.128.0.4:6443 --token ifbwdp.sunt9nqhwd5wukcz --discovery-
token-ca-cert-hash
sha256:d0466dcadbd754d34f8ba09556ca2b2a9d338d80e062af3aca6ec2c954bfde
29
```

Agar *kubectl* berfungsi untuk pengguna *non-root*, dengan menggunakan perintah yang juga merupakan bagian dari keluaran *kubeadm* `init`:

```
brica@k8smaster:~$ mkdir -p $HOME/.kube
brica@k8smaster:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
brica@k8smaster:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

### E. *Worker nodes joint cluster*

Token digunakan untuk otentikasi timbal balik antara *node* bidang kontrol dan *node* yang bergabung. Token yang disertakan di sini adalah rahasia. Tetap aman, karena siapa pun yang memiliki token ini dapat menambahkan *node* yang diautentikasi ke *cluster*.

```
brica@k8smaster:~$ kubectl join 10.128.0.4:6443 --token
ifbwdp.sunt9nqhwd5wukcz --discovery-token-ca-cert-hash
sha256:d0466dcadb754d34f8ba09556ca2b2a9d338d80e062af3aca6ec2c954bfde
29
```

### 3.4.3 *Install Container Network Interface (CNI)*

Instalasi CNI dilakukan secara bergantian di mana jika salah satu CNI sudah ter-*install*, maka CNI tersebut harus terlebih dahulu dihapus untuk instalasi CNI lainnya, dimulai dari *Calico*, *Cilium*, *Flannel* dan *Weave Net*.

#### A. *Install CNI Calico*

Meng-*install plugin* jaringan *Calico*

```
brica@k8smaster:~$ kubectl apply -f
https://docs.projectcalico.org/manifests/Calico.yaml
```

Perintah *kubectl* mendeskripsikan *Pods* memberikan informasi rinci tentang setiap *pod* yang menyediakan infrastruktur Kubernetes.

```
brica@k8smaster:~$ kubectl get pods -n kube-system
```

#### B. *Install CNI Cilium*

Meng-*install plugin* jaringan *Cilium*

```
brica@k8smaster:~$ Cilium install
```

Memvalidasi bahwa *Cilium* telah di-*install* dengan benar.

```
brica@k8smaster:~$ Cilium status --wait
brica@k8smaster:~$ Cilium connectivity test
```

Memvalidasi setia *pod* telah menggunakan CNI yang sesuai.

```
brica@k8smaster:~$ kubectl get pods -n kube-system
```

### C. *Install CNI Flannel*

*Flannel* dapat ditambahkan ke *cluster* Kubernetes dengan menambahkan *Flannel* sebelum *pod* yang menggunakan jaringan *pod* dimulai.

```
brica@k8smaster:~$ kubectl apply -f https://raw.githubusercontent.com/Flannel-io/Flannel/master/Documentation/kube-Flannel.yml
```

Memvalidasi setiap *pod* telah menggunakan CNI yang sesuai.

```
brica@k8smaster:~$ kubectl get pods -n kube-system
```

### D. *Install CNI Weave Net*

Agar *Weave Net* berfungsi, perlu memastikan penerusan IP diaktifkan pada *node* pekerja. Aktifkan dengan menjalankan perintah berikut pada kedua pekerja.

```
brica@k8smaster:~$ sudo sysctl net.ipv4.conf.all.forwarding=1  
brica@k8smaster:~$ echo "net.ipv4.conf.all.forwarding=1" | sudo tee -a  
/etc/sysctl.conf
```

Meng-*install Weave Net* menggunakan konfigurasi dari *Weaveworks*

```
brica@k8smaster:~$ $ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-  
version=$(kubectl version | base64 | tr -d '\n')"
```

Memvalidasi setia *pod* telah menggunakan CNI yang sesuai.

```
brica@k8smaster:~$ kubectl get pods -n kube-system
```

### 3.4.4 *Nginx*

*Deployment Nginx* menggunakan versi 1.18.0 dengan menggunakan 3 *replicas/pod*. *Pod* tersebut akan masukan ke dalam kedua *worker node* secara otomatis.

### A. *Deployment Nginx*

Pembuatan *deployment Nginx* dengan menggunakan perintah *nano* untuk membuat *file* *yaml* yang berisi konfigurasi *Nginx* sesuai dengan kebutuhan. *Deployment* dibuat dengan menggunakan perintah *kubectl apply -f nginx-deployment.yaml*. Pengujian apakah *deployment* sudah berhasil dibuat dengan menggunakan perintah *kubectl get deployment*.

```
brica@k8smaster:~$ sudo nano nginx-deployment.yaml
brica@k8smaster:~$ cat nginx-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: default
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.18.0
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-index-file
          mountPath: /usr/share/nginx/html/
      volumes:
      - name: nginx-index-file
        configMap:
          name: index-html-configmap
```

## B. *Deployment Nginx Service*

Pembuatan *service Nginx* dengan menggunakan perintah *nano* untuk membuat *file yml* yang berisi konfigurasi *Nginx service* sesuai dengan kebutuhan. *Service* dibuat dengan menggunakan perintah *kubectl apply -f nginx-service.yml*. *Service* yang dibuat bertipe *loadbalancer* di mana akan membagi beban akses dengan seimbang untuk setiap *pod* yang bekerja. *Service* yang dibuat akan menggunakan IP eksternal dari *master node*. Pengujian apakah *service* telah berhasil dibuat dengan menggunakan *kubectl get service*.

```
brica@k8smaster:~$ sudo nano nginx-deployment.yml
brica@k8smaster:~$ cat nginx-deployment.yml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: default
spec:
  selector:
    app: nginx
  type: LoadBalancer
  externalIPs:
  - 34.123.204.161
  ports:
  - name: http
    port: 80
    nodePort: 30062
```

## C. *Testing Nginx*

*Testing Nginx* berguna untuk memastikan apakah *deployment* yang dibuat sudah dapat digunakan dan dapat diakses oleh *client* secara normal. *Testing Nginx* dapat menggunakan dua cara yaitu dengan perintah *curl* dilanjutkan dengan IP *service* atau pun dapat menggunakan *browser* pada client dengan mengakses IP dan

*port service* yang telah dibuat pada *deployment*. Halaman *web server Nginx* akan terlihat seperti pada Gambar 3.13.

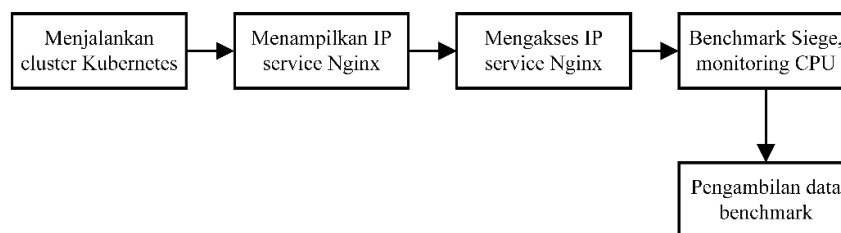


Gambar 3.13 Halaman *web server Nginx*

### 3.4.5 Siege Benchmark

Instalasi Siege v4.1.3 dilakukan pada *client* dan salah satu *worker node*. Instalasi pada *client* bertujuan untuk melakukan *benchmark* dari sisi *client* menuju *service*. Instalasi pada sisi *node* bertujuan untuk melakukan *benchmark* pada sisi *pod* menuju *service* dan juga *pod* menuju *pod*. Tahapan instalasi dilakukan dengan *download file repository* yang berada pada <http://download.joedog.org/siege/>. Siege di ekstrak pada *directory /opt/* dan dilakukan kostumisasi pada tampilan *output* agar menampilkan hasil yang lebih akurat pada *file /opt/src/main.c*. Selanjutnya dilakukan perintah `./configure` dan perintah `make install`. Untuk memastikan apakah Siege sudah terpasang dengan benar menggunakan perintah `siege -version`.

## 3.5 PROSES SIMULASI



Gambar 3.14 Alur proses simulasi

Gambar 3.14 merupakan proses simulasi yang dilakukan setelah proses instalasi seluruh komponen simulasi seperti *Google Cloud Platform*, *Kubernetes*, *Container Network Interface*, *Nginx*, dan *Siege benchmark* selesai. Pertama yaitu menjalankan *Kubernetes cluster* pada *Google Cloud Platform*. Pada Gambar 3.15, simulasi yang dijalankan yaitu komunikasi *client to service* dengan menampilkan *service* pada *master node* yang digunakan untuk mengakses *Nginx*.



```
brica@k8smaster:~$ kubectl get service
NAME          TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
kubernetes    ClusterIP     10.96.0.1       <none>           443/TCP          4d9h
nginx-service  LoadBalancer 10.111.159.39   104.154.243.246 80:32670/TCP     7s
```

Gambar 3.15 Tampilan *Nginx service*

IP address dan port diakses melalui browser pada client dan copy kan alamat URL service Nginx tersebut. Buka aplikasi Siege benchmark pada client. Jalankan perintah `siege http://ip-service:port service/ -c10 -t60s` seperti yang terlihat pada Gambar 3.16.

```
wahyu@BRI:~$ siege http://104.154.243.246:32670/ -c10 -t60s
** SIEGE 4.1.3
** Preparing 10 concurrent users for battle.
The server is now under siege...
```

Gambar 3.16 Syntax Siege benchmark

Gambar 3.17 menunjukkan hasil benchmark dengan alamat URL yang sudah ditentukan dengan jumlah *simulated user* sebanyak 10 user dan lama waktu yaitu 60 second.

```
wahyu@BRI:~$ siege http://104.154.243.246:32670/ -c10 -t60s
** SIEGE 4.1.3
** Preparing 10 concurrent users for battle.
The server is now under siege...
Lifting the server siege...
Transactions:          1182 hits
Availability:          100.00 %
Elapsed time:          59.43 secs
Data transferred:     0.09 MB
Response time:         0.50011 secs
Transaction rate:     19.89 trans/sec
Throughput:            0.00156 MB/sec
Concurrency:           9.95
Successful transactions: 1182
Failed transaction:    0
Shortest transaction:  0.47
```

Gambar 3.17 Hasil benchmark menggunakan Siege

Lakukan juga proses monitoring pada node master dengan menggunakan tool *htop* seperti pada Gambar 3.18. Simpan nilai average CPU pada master node ketika sedang menjalankan simulasi.

```
1  [||||]          4.0%]  Tasks: 56, 254 thr: 2 running
2  [||||]          6.0%]  Load average: 0.18 0.22 0.14
Avg[||||]          5.0%]  Uptime: 00:12:52
Mem[|||||]         748M/7.76G] Load: 0.18
Swp[ ]             0K/0K]

PID USER      PRI  NI  VIRT   RES   SHR   S  CPU%  MEM%  PTID#  Command
1250 root        20   0 1146M 365M 72176 S  4.0  4.6  0:40.58 kube-apiserver --advertise-address=10.128.0.11 --all
512  root        20   0 1191M 99776 62832 S  2.0  1.2  0:17.22 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes
531  root        20   0 1299M 63300 31156 S  2.0  0.8  0:05.48 /usr/bin/containerd
1372 root        20   0 1146M 365M 72176 S  1.3  4.6  0:07.11 kube-apiserver --advertise-address=10.128.0.11 --allc
1378 root        20   0 1146M 365M 72176 S  1.3  4.6  0:07.33 kube-apiserver --advertise-address=10.128.0.11 --allc
1379 root        20   0 1891M 99776 62832 S  1.3  1.2  0:04.03 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes
924  root        20   0 1299M 63300 31156 S  1.3  0.8  0:00.55 /usr/bin/containerd
776  root        20   0 1891M 99776 62832 S  0.7  1.2  0:04.22 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes
1164 root        20   0 10.7G 55228 23560 S  0.7  0.7  0:12.91 etcd --advertise-client-urls=https://10.128.0.11:237
1201 root        20   0 800M  99740 60820 S  0.7  1.2  0:11.39 kube-controller-manager --allocate-node-cidrs=true -
```

Gambar 3.18 Tampilan monitoring CPU usage

Untuk simulasi selanjutnya pada komunikasi pod to service yaitu masih menggunakan proses yang sama namun berbeda ketika melakukan proses benchmark, yaitu berada pada salah satu worker node. Pada simulasi komunikasi pod to pod di mana memiliki perbedaan pada URL yang digunakan untuk

*benchmark*. Simulasi komunikasi *pod to pod* menggunakan alamat IP *worker* salah satu *worker node* yang dilakukan *benchmark* dari sisi *worker node01* menuju *worker node02*. Ulangi ketiga tahapan berikut untuk *simulated user* 100 dan 200 *user*.

### 3.6 PENGAMBILAN DATA PENELITIAN

Pengambilan data dilakukan dengan menggunakan *tool* Siege dan *htop*. Tabel 3.4 merupakan data yang diambil untuk dianalisis yaitu parameter *response time*, *transaction rate*, *throughput*, dan *CPU usage* sebanyak 10 kali percobaan.

Tabel 3.4 Parameter Pengujian

<b>Parameter Pengujian</b>	<b>Satuan</b>
<i>Response time</i>	<i>second</i>
<i>Throughput</i>	Mbps
<i>Transaction rate</i>	<i>transaction/sec</i>
<i>CPU Usage</i>	%

### 3.7 ANALISIS DATA

Tahap ini dilakukan ketika pengambilan data dari hasil simulasi selesai, dan dapat menghasilkan keluaran untuk dianalisis. Analisis yang dilakukan yaitu dengan membandingkan hasil parameter *response time*, *transaction rate*, *throughput*, dan *CPU usage* berdasarkan skenario pada Tabel 3.2. Penjelasan masing-masing parameter telah dijabarkan pada BAB 2 subbab 2.2.8. Analisis dilakukan berdasarkan data parameter *benchmark* yang diperoleh dan divisualisasikan ke dalam bentuk grafik menggunakan *software* Matlab agar memudahkan proses analisis, selanjutnya akan dibahas pada BAB 4.