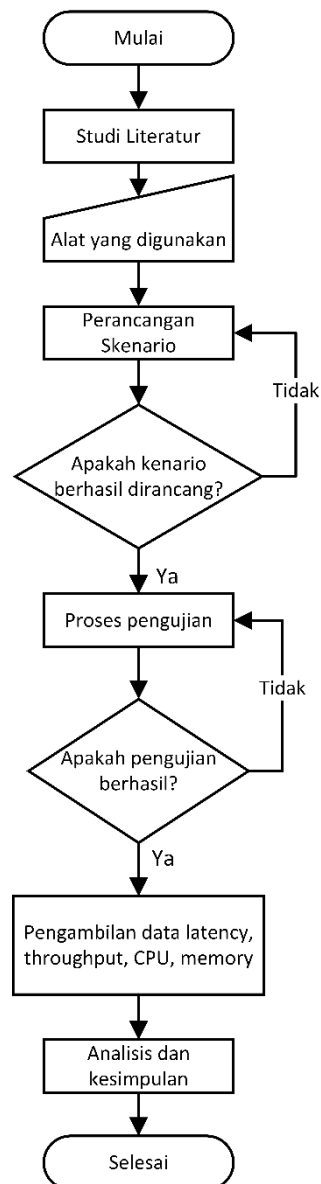


BAB 3 METODE PENELITIAN

3.1 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. Adapun penelitian ini akan dilakukan sesuai dengan urutannya yang digambarkan pada Gambar 3.1.



Gambar 3.1 Alur Penelitian

Alur penelitian dimulai dari studi literatur dengan mengkaji penelitian-penelitian sebelumnya yang relevan dengan penelitian penulis. Selanjutnya menentukan alat yang digunakan berupa perangkat keras dan perangkat lunak untuk membuat sistem. Pada perancangan skenario, dilakukan penentuan skenario sesuai dengan referensi yang digunakan. Perancangan skenario dianggap gagal jika skenario yang digunakan tidak dapat diterapkan atau tidak berjalan dengan lancar pada alat yang digunakan, maka akan dilakukan konfigurasi ulang. Konfigurasi ulang pada perancangan skenario tersebut akan digantikan dengan konfigurasi yang sesuai sampai skenario tersebut berhasil diterapkan dan tidak ada error. Sedangkan perancangan skenario berhasil dimana skenario tersebut dapat diterapkan pada alat yang digunakan dan juga sistem yang berjalan yaitu ketiga *container runtime* (*containerd*, CRI-O, dan kata *container*). Selanjutnya yaitu proses pengujian. Proses pengujian yang gagal yaitu ketika sistem yang dirancang tidak berjalan semestinya seperti sistem mengalami load yang berlebihan sehingga tidak dapat digunakan. Ketika proses pengujian mengalami kegagalan maka akan dilakukan *me-restart* aplikasi pengambilan data atau sistem. Sedangkan untuk proses pengujian yang berhasil yaitu ketika sistem yang dirancang untuk menjalankan simulasi dapat berjalan secara lancar dan optimal tanpa mengalami kendala yang berlebih. Setelah proses pengujian, didapatkan hasil data pengujian yang akan diolah, lalu hasil tersebut dapat dianalisis dan dibuat kesimpulan.

3.2 Studi Literatur

Studi literatur menjadi pada penelitian ini menjadi tahap yang bertujuan untuk mencari referensi untuk dijadikan sebagai acuan dalam pelaksanaan penelitian. Sumber yang digunakan oleh penulis berupa jurnal, buku, dan *website* yang berhubungan dengan topik penelitian.

3.3 Alat Yang Digunakan

Penelitian ini menggunakan suatu pemodelan *container* dalam menganalisis performansi *container runtime* (*Containerd*, CRI-O, dan Kata *Containers*) pada orkestrasi Kubernetes. Model *containerisasi* yang akan disimulasikan menggunakan *Google Cloud Platform*.

3.3.1 Perangkat Keras

Perangkat keras yang akan digunakan pada penelitian ini menggunakan satu buah laptop dengan spesifikasi sebagaimana terdapat pada tabel 3.1.

Tabel 3.1 Spesifikasi *Hardware*

Perangkat	Spesifikasi	Keterangan
Laptop Lenovo Ideapad Gaming 3 15IAH7	OS	Windows 11
	CPU	I5 12500H (16 CPU)
	GPU	RTX 3050 8GB
	RAM	16 GB

3.3.2 Perangkat Lunak

Perangkat lunak sebagai tool dan aplikasi yang digunakan pada penelitian ini dapat dilihat pada tabel 3.2.

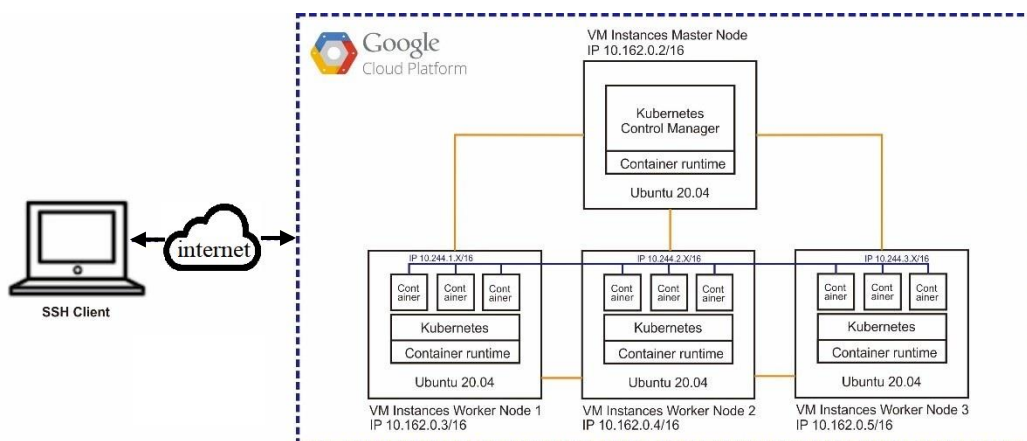
Tabel 3.2 Spesifikasi *server google cloud*

Perangkat	Spesifikasi	Keterangan	Fungsi
<i>Compute engine</i>	<i>Google Cloud Platform</i>	<i>VM Instances</i>	Membuat mesin virtual
1 <i>Master node</i> dan 3 <i>Worker node</i>	OS	<i>Ubuntu Server 20.04</i>	Sistem operasi
	vCPU	2 vCPU	Virtual CPU
	vRAM	8 GB	Virtual RAM
	Komponen	Containerd 1.6.21	<i>Container runtime</i>
		CRI-O 1.24.6	<i>Container runtime</i>
		Kata Containers 3,0	<i>Container runtime</i>
		Kubelet v1.22.1	Untuk Menjalankan perintah <i>manager</i>
Kubectl v1.22.1	Menjalankan <i>Container</i>		
Kubeadm v1.22.1	Untuk Membuat <i>cluster</i>		
<i>Analysis tools</i>	Aplikasi	Netperf 2.6.0	Mengambil data parameter <i>throughput</i>

Perangkat	Spesifikasi	Keterangan	Fungsi
		Ping s20190709	Mengambil data parameter <i>latency</i>
		y-cruncher v0.7.10.9513	Mengambil data parameter CPU
		STREAM 1.5.0	Mengambil data parameter <i>memory</i>

3.4 Perancangan Topologi

Gambar 3.2 merupakan topologi yang digunakan untuk pengujian. Topologi ini terbagi dalam area *Google Cloud Platform* dan area *client*. *Google Cloud Platform* berfungsi untuk membuat virtual mesin (VM) *instances*. Pada *Google Cloud Platform* area terdiri dari 4 VM *instances* yaitu satu *master node* dan tiga *worker node*, masing-masing *master node* dan *worker node* menggunakan sistem operasi Ubuntu *Server 20.04* serta spesifikasi yang dijelaskan pada Tabel 3.2.



Gambar 3.2 Topologi jaringan

Di dalam *master node* dan *worker node* masing-masing memiliki internal IP dan komponen seperti Kubernetes *control manager* dan *container runtime*, serta komponen *container* hanya pada *worker node*. *Master node* bertanggung jawab atas pemeliharaan dan pengelolaan *cluster*, sementara *worker node* menjalankan satu set *container* untuk aplikasi. Kubernetes *control manager* berfungsi untuk mengelola seluruh proses dan layanan ini berjalan bersama di *master* Kubernetes, *container runtime* bertanggung jawab menjalankan *container* dan mengelola *image container* pada *deployment node*, serta *container* sendiri berfungsi untuk menjalankan apa

yang ada di *microservice* kecil atau proses perangkat lunak hingga aplikasi yang lebih besar. Setiap *worker node* saling terhubung dan terhubung langsung ke *master node* melalui konfigurasi pada master node, konfigurasi worker node join master node dijelaskan pada subbab 3.5.2 poin H. Setiap worker node dan master node juga secara otomatis hidup dan terhubung langsung ketika seluruh VM dinyalakan di GCP. Lalu akan secara otomatis mati ketika VM tersebut dimatikan. Untuk komponen *container runtime*, pengujian dilakukan secara bergantian dengan menggunakan *container runtime* Containerd, CRI-O, dan Kata Containers.

Internal IP berfungsi sebagai IP *cluster* yang digunakan untuk komunikasi antar *worker node* dan *master node*. Masing-masing *container* juga memiliki IP yaitu 10.244.1.0/16, IP dalam *container* berfungsi untuk komunikasi antar *container* dalam mengirimkan data suatu layanan. *Container* akan memiliki IP yang berbeda dengan VM *instances*, dimana IP *container* akan ditentukan secara otomatis dan hanya menentukan *network* yang akan digunakan yaitu 10.244.0.0/16. IP *container* dikonfigurasi menggunakan *network* dan *subnetmask*-nya saja, sehingga sistem dapat menentukan IP yang tersedia secara otomatis. Garis berwarna biru merupakan jalur komunikasi antar container dalam satu cluster Kubernetes atau yang disebut komunikasi intra-cluster. Garis berwarna kuning merupakan jalur komunikasi antar master node dan worker node, di mana jalur komunikasi ini bersifat 2 arah yang dapat mengirim dan menerima informasi. Untuk internal IP (IP *cluster*) yang digunakan yaitu 10.162.0.0/16 dengan IP untuk masing-masing VM, yaitu *master node* memiliki IP 10.162.0.2, *worker node* 1 memiliki IP 10.162.0.3, *worker node* 2 memiliki IP 10.162.0.4 dan *worker node* 3 memiliki IP 10.162.0.5. Tabel 3.3 merupakan daftar IP *address* yang digunakan pada *master node* dan *worker node*.

Pada garis biru putus-putus merupakan area lingkungan *Google Cloud Platform* yang dapat diakses menggunakan protokol SSH oleh *client*. Pada SSH *client* area terhubung melalui internet ke *Google Cloud Platform* dengan mengakses menggunakan IP *external* dari VM *instances master node*. Dimana *external* IP ini didapatkan hanya ketika VM dinyalakan dan setiap kali VM nyala *external* IP tersebut akan berubah-ubah. Pengujian ini menggunakan topologi jaringan untuk mengetahui inisialisasi IP *host* untuk setiap VM *Instances* yang akan digunakan.

Topologi jaringan yang digunakan yaitu berupa topologi jaringan *mesh* sehingga setiap VM *instances* dapat terhubung secara langsung.

Tabel 3.3 Internal IP Master Node dan Worker Node

VM	IP Address
<i>Master node</i>	10.162.0.2/16
<i>Worker node 1</i>	10.162.0.3/16
<i>Worker node 2</i>	10.162.0.4/16
<i>Worker node 3</i>	10.162.0.5/16

3.5 Perancangan Skenario

Skenario yang digunakan pada simulasi dibagi menjadi dua bagian, yang pertama yaitu skenario jenis runtime dan skenario penambahan jumlah container. Pada skenario jenis runtime menggunakan tiga jenis *container runtime* yang berbeda Containerd, CRI-O, dan Kata Containers. Sedangkan pada skenario penambahan jumlah *container* yaitu sebanyak 10, 20, and 40 *container*. Skenario akan di uji coba secara bergantian mulai dari skenario pertama hingga skenario ketiga.

Pada Tabel 3.4 berisi skenario yang digunakan dalam penelitian. Skenario pertama yaitu akan menggunakan Containerd sebagai *container runtime* yang akan dikonfigurasi dengan jumlah *container* yang dipasangkan secara bergantian yaitu 10, 20, dan 40. Skenario kedua yaitu akan menggunakan CRI-O sebagai *container runtime* yang akan dikonfigurasi dengan jumlah *container* yang dipasangkan secara bergantian yaitu 10, 20, dan 40. Skenario ketiga yaitu akan menggunakan *kata container* sebagai *container runtime* yang akan dikonfigurasi dengan jumlah *container* yang dipasangkan secara bergantian yaitu 10, 20, dan 40. Skenario penelitian ini dilakukan percobaan dengan pengambilan data sebanyak 30 kali percobaan pada setiap skenario yang dijalankan.

Tabel 3.4 Skenario Penelitian

Skenario	Container Runtime	Jumlah Container	Jumlah Percobaan
1	Containerd	10	30
		20	
		40	
2	CRI-O	10	30
		20	
		40	
3	Kata Containers	10	30
		20	
		40	

Skenario pengujian yang digunakan bertujuan untuk mengukur kinerja dari masing-masing container runtime yang memiliki karakteristik berbeda-beda. Apakah dengan pengujian bertambahnya jumlah container akan mempengaruhi kinerja dari parameter yang diukur yaitu throughput, latency, CPU, dan memory. Misalnya dengan bertambahnya jumlah container yang semakin banyak, mungkin saja dapat mempengaruhi kinerja CPU dan memory menjadi berat, atau bertambahnya jumlah container akan mempengaruhi throughput yang menurun dan latency yang meningkat.

3.5.1 Google Cloud Platform Service

Penelitian ini dilakukan secara simulasi menggunakan *Google Cloud Platform (GCP)*. Pada topologi jaringan Gambar 3.2 diilustrasikan, penelitian ini memiliki 1 *master node* dan 3 *worker node*. Spesifikasi yang digunakan untuk masing-masing *master node* dan *worker node* tertulis pada Tabel 3.2, seperti menggunakan sistem operasi Ubuntu server 20.04, CPU sebesar 2 vCPU, serta RAM sebesar 8 GB. Gambar 3.3 merupakan tampilan pada GCP yang memiliki external IP yang berbeda dan dapat berubah-ubah.

Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP	Connect
✓	masternode	northamerica-northeast1-a			10.162.0.2 (nic0)	35.234.243.10 (nic0)	SSH ▾
✓	workernode01	northamerica-northeast1-a			10.162.0.3 (nic0)	34.152.32.85 (nic0)	SSH ▾
✓	workernode02	northamerica-northeast1-a			10.162.0.4 (nic0)	34.95.62.122 (nic0)	SSH ▾
✓	workernode03	northamerica-northeast1-a			10.162.0.5 (nic0)	34.118.142.107 (nic0)	SSH ▾

Gambar 3.3 Master Node dan Worker Node pada GCP

3.5.2 Kubernetes

Deployment pada Kubernetes memiliki beberapa tahap konfigurasi, yaitu konfigurasi yang dilakukan pada seluruh *node*, konfigurasi pada *master node*, dan konfigurasi pada *worker node*. Pada penelitian ini terdapat satu *master node* dan tiga *worker node*, sehingga jumlahnya empat *node*. Keempat *node* ini bergabung dan membuat sebuah *cluster* Kubernetes.

Proses pertama yaitu konfigurasi yang dilakukan pada seluruh *node*, yaitu *master node*, *worker node 1*, *worker node 2*, dan *worker node 3*. Seluruh *node* dikonfigurasi menggunakan perintah yang sama. Konfigurasi pertama dimulai dari *upgrade* paket Linux, *setting* kernel, *install* kubectl, kubelet, dan kubeadm.

A. Upgrade Package

Tahap konfigurasi *upgrade* paket bertujuan untuk *update*, *upgrade*, dan menghapus paket pada Linux yang telah usang.

```
sudo apt update
sudo apt upgrade -y
sudo apt autoremove -y
```

B. Setting Kernel

Konfigurasi untuk pengaturan kernel dengan mengaktifkan modul kernel yaitu overlay dan br_netfilter, serta menambahkan beberapa pengaturan ke sysctl.

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter
```


Menambahkan parameter sysctl yaitu bridge ip tables, bridge ipv6 tables, dan ipv4 *forward*.

```
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
```

Menerapkan parameter sysctl dengan memuat kembali (reload) sistem tanpa merestartnya.

```
sudo sysctl --system
```

C. *Install* Kubectl, Kubelet, dan Kubeadm

Menginstal beberapa paket dependensi tambahan melalui *curl*, yaitu paket *transport-http*, menambahkan GPG *key*, serta menambah *repository* Kubernetes.

```
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
```

Setelah *repository* dibuat, selanjutnya memindahkan *repositoty* *kubernetes.list* dan meng*update* paket, lalu menginstal *kubelet*, *kubectl*, dan *kubeadm*.

```
sudo mv kubernetes.list /etc/apt/sources.list.d/kubernetes.list
sudo apt update; sudo apt install -y kubectl kubelet kubeadm
sudo apt-mark hold kubelet kubeadm kubectl
```

Proses kedua yaitu konfigurasi yang dilakukan pada *master node* saja. Konfigurasi dimulai dari inisialisasi *master node*, konfigurasi admin, *install container network flannel*, serta verifikasi konfigurasi dan *cluster*.

E. Insialisasi Master Node

Konfigurasi pada master node dimulai dengan inisialisasi *master*, dimana diperlukan untuk menonaktifkan *swap* terlebih dulu pada *master node*.

```
swapon -s
sudo swapoff -a
sudo kubeadm init --pod-network-cidr=10.244.XX.0/16
```

F. Konfigurasi Admin

Konfigurasi admin dilakukan dengan membuat direktori baru yaitu kube untuk menampung *file* konfigurasi, lalu menyalin direktori tersebut, dan memberinya izin yang sesuai.

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

G. *Install Network Flannel*

Network Flannel berfungsi agar antar *host* dapat saling berkomunikasi didalam *cluster* Kubernetes. Perintah dibawah digunakan untuk menginstal *network Flannel* dengan mendownload *file *.yaml* dari *repository flannel*, meng-*apply* konfigurasi dari *file yml*, dan menerapkan konfigurasi ke setiap *pod*.

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

H. Verifikasi Konfigurasi dan Cluster

Melakukan verifikasi konfigurasi yang telah dilakukan dan disimpan, dan juga melakukan verifikasi *cluster* Kubernetes.

```
kubectl config view
kubectl cluster-info
```

Proses ketiga yaitu konfigurasi yang dilakukan pada *worker node* saja. Konfigurasi yang dilakukan yaitu menggabungkan *worker node* ke *master node* atau *cluster* Kubernetes.

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

Proses keempat pada *deployment* Kubernetes yaitu konfigurasi terakhir yang dilakukan pada *master node* saja. Konfigurasi yang dilakukan yaitu memverifikasi *node* yang telah dibuat pada *cluster* Kubernetes.

```
kubectl get nodes
```

Pada perancangan skenario, terdapat skenario skalabilitas *container* yang digunakan yaitu sebanyak 10, 20, dan 40 *container*. Perintah yang dijalankan untuk membuat beberapa *container* yaitu sebagai berikut.

I. Membuat Script YAML

Perintah dibawah berfungsi untuk membuat file *.yaml bernama deployment-container.yaml, yang di dalamnya berisi konfigurasi untuk membuat *container* dengan jumlah yang dibutuhkan.

```
sudo nano deployment-container.yaml
```

J. Deployment Container

Di dalam file *.yaml tersebut berisi konfigurasi untuk membuat *container* dengan ditandai oleh metrik “*replicas*” untuk menentukan jumlah *container*. Perintah berikut merupakan isi *file* konfigurasi untuk *deploy* container berjumlah 10 container, isi *file* konfigurasi ini sama untuk semua *container runtime* Containerd, CRI-O, dan Kata Containers.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: container
spec:
  selector:
    matchLabels:
```

```

  app: container
replicas: 10
template:
  metadata:
    labels:
      app: container
spec:
  runtimeClassName:
  containers:
  - name: container
    image: nginx

```

Setelah file *.yaml dibuat, lalu *file* tersebut disimpan dan di-*apply* pada Kubernetes dengan perintah berikut.

```
kubectl apply -f deployment-container.yaml
```

K. Pengecekan Container

Untuk dapat melihat daftar *container* yang telah dibuat, dapat menggunakan perintah berikut. Daftar *container* yang telah dibuat akan muncul seperti yang ditunjukkan pada Gambar 3.4 untuk 10 *container*, Gambar 3.5 untuk 20 *container*, dan Gambar 3.6 untuk 40 *container*.

```
kubectl get pod -o wide
```

```

pratama@masternode:~$ kubectl get pod -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE
GATES
containercoba-69485f45c9-4gxvz     1/1     Running   0           5h25m  10.244.2.26     workernode03
containercoba-69485f45c9-58ckh     1/1     Running   0           5h25m  10.244.2.24     workernode03
containercoba-69485f45c9-bzccp     1/1     Running   0           5h25m  10.244.3.24     workernode01
containercoba-69485f45c9-cfbk5     1/1     Running   0           5h25m  10.244.2.25     workernode03
containercoba-69485f45c9-lxpmz     1/1     Running   0           5h25m  10.244.1.22     workernode01
containercoba-69485f45c9-nxqm7     1/1     Running   0           5h25m  10.244.3.22     workernode02
containercoba-69485f45c9-rnnkx     1/1     Running   0           5h25m  10.244.1.21     workernode02
containercoba-69485f45c9-vv5s7     1/1     Running   0           5h25m  10.244.1.23     workernode02
containercoba-69485f45c9-wwc9m     1/1     Running   0           5h25m  10.244.2.23     workernode03
containercoba-69485f45c9-wwdbr     1/1     Running   0           5h25m  10.244.3.23     workernode01

```

Gambar 3.4 Daftar *container* berjumlah 10

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
containercoba-69485f45c9-4b15p	1/1	Running	0	4s	10.244.3.60	workernode01
containercoba-69485f45c9-4d95f	1/1	Running	0	3s	10.244.2.56	workernode03
containercoba-69485f45c9-587p8	1/1	Running	0	4s	10.244.3.59	workernode01
containercoba-69485f45c9-5fr4h	1/1	Running	0	3s	10.244.2.57	workernode03
containercoba-69485f45c9-6545v	1/1	Running	0	4s	10.244.2.61	workernode03
containercoba-69485f45c9-6pzsq	1/1	Running	0	4s	10.244.1.59	workernode02
containercoba-69485f45c9-6gg2l	1/1	Running	0	4s	10.244.3.56	workernode01
containercoba-69485f45c9-8xdpr	1/1	Running	0	4s	10.244.1.55	workernode02
containercoba-69485f45c9-bfspd	1/1	Running	0	3s	10.244.3.58	workernode01
containercoba-69485f45c9-blfgf	1/1	Running	0	4s	10.244.2.59	workernode03
containercoba-69485f45c9-cz6kx	1/1	Running	0	4s	10.244.1.56	workernode02
containercoba-69485f45c9-dtztz	1/1	Running	0	3s	10.244.1.61	workernode02
containercoba-69485f45c9-fgcsz	1/1	Running	0	4s	10.244.1.58	workernode02
containercoba-69485f45c9-fhpd7	1/1	Running	0	4s	10.244.3.61	workernode01
containercoba-69485f45c9-hk6x9	1/1	Running	0	4s	10.244.1.57	workernode02
containercoba-69485f45c9-jdhjk	1/1	Running	0	4s	10.244.2.55	workernode03
containercoba-69485f45c9-kwlll	1/1	Running	0	3s	10.244.1.60	workernode02
containercoba-69485f45c9-nrtwc	1/1	Running	0	4s	10.244.2.58	workernode03
containercoba-69485f45c9-q2dxn	1/1	Running	0	4s	10.244.3.57	workernode01
containercoba-69485f45c9-v99kl	1/1	Running	0	4s	10.244.2.60	workernode03

Gambar 3.5 Daftar *container* berjumlah 20

```
pratama@masternode:~$ kubectl get pod -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
containercoba-69485f45c9-244h7	1/1	Running	0	7s	10.244.2.47	workernode03
containercoba-69485f45c9-29xlk	1/1	Running	0	7s	10.244.1.49	workernode02
containercoba-69485f45c9-6v2qv	1/1	Running	0	7s	10.244.2.42	workernode03
containercoba-69485f45c9-84rrp	1/1	Running	0	6s	10.244.3.47	workernode01
containercoba-69485f45c9-88xmp	1/1	Running	0	7s	10.244.3.48	workernode01
containercoba-69485f45c9-9fvnh	1/1	Running	0	7s	10.244.3.50	workernode01
containercoba-69485f45c9-bg25s	1/1	Running	0	7s	10.244.1.41	workernode02
containercoba-69485f45c9-b15fm	1/1	Running	0	7s	10.244.3.39	workernode01
containercoba-69485f45c9-bn9j9	1/1	Running	0	7s	10.244.3.51	workernode01
containercoba-69485f45c9-cd797	1/1	Running	0	7s	10.244.1.40	workernode02
containercoba-69485f45c9-d7j6b	1/1	Running	0	7s	10.244.1.44	workernode02
containercoba-69485f45c9-dbq8z	1/1	Running	0	6s	10.244.2.50	workernode03
containercoba-69485f45c9-dj92f	1/1	Running	0	7s	10.244.3.52	workernode01
containercoba-69485f45c9-f9p9n	1/1	Running	0	7s	10.244.3.44	workernode01
containercoba-69485f45c9-h5z7f	1/1	Running	0	7s	10.244.2.39	workernode03
containercoba-69485f45c9-hn2zd	1/1	Running	0	7s	10.244.3.42	workernode01
containercoba-69485f45c9-hrs8t	1/1	Running	0	7s	10.244.1.50	workernode02
containercoba-69485f45c9-kfgj4	1/1	Running	0	7s	10.244.3.49	workernode01
containercoba-69485f45c9-l8hgb	1/1	Running	0	7s	10.244.3.41	workernode01
containercoba-69485f45c9-m657c	1/1	Running	0	7s	10.244.1.43	workernode02
containercoba-69485f45c9-mm756	1/1	Running	0	7s	10.244.2.51	workernode03
containercoba-69485f45c9-mvh4f	1/1	Running	0	7s	10.244.2.40	workernode03
containercoba-69485f45c9-nsx4c	1/1	Running	0	7s	10.244.2.49	workernode03
containercoba-69485f45c9-ntmmh	1/1	Running	0	7s	10.244.1.42	workernode02
containercoba-69485f45c9-p4xxh	1/1	Running	0	7s	10.244.1.48	workernode02
containercoba-69485f45c9-pghlc	1/1	Running	0	7s	10.244.2.48	workernode03
containercoba-69485f45c9-prs7h	1/1	Running	0	7s	10.244.2.45	workernode03
containercoba-69485f45c9-q8khh	1/1	Running	0	7s	10.244.2.44	workernode03
containercoba-69485f45c9-qvc77	1/1	Running	0	7s	10.244.1.47	workernode02
containercoba-69485f45c9-r6lr8	1/1	Running	0	7s	10.244.3.43	workernode01
containercoba-69485f45c9-s6rq5	1/1	Running	0	7s	10.244.1.39	workernode02

Gambar 3.6 Daftar *container* berjumlah 40

3.5.3 Containerd

Runtime Containerd merupakan runtime yang ada pada paket Kubernetes. Namun tetap membutuhkan beberapa konfigurasi paket tambahan. Konfigurasi dibawah ini berfungsi untuk konfigurasi dependensi Kubernetes dengan menginstall curl dan transport-http.

```
sudo apt install -y curl gnupg2 software-properties-common apt-transport-https ca-certificates
```

Konfigurasi untuk mengaktifkan *repository* Docker dan juga menambahkan paket dan *repository*.

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

Runtime containerd diinstal setelah menambahkan repository Docker. Paket harus *diupdate* terlebih dulu agar dapat terbaca oleh server. Setelah itu menginstal containerd.

```
sudo apt update
sudo apt install -y containerd.io
```

Setelah containerd terinstal, konfigurasi agar containerd dapat menggunakan systemd sebagai *cgroup (control group)* yang berfungsi sebagai sistem init Linux yang menggunakan *root cgroup* dan bertindak sebagai *manager*.

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

Instalasi containerd diakhiri dengan merestart containerd lalu mengaktifkannya melalui systemctl.

```
sudo systemctl restart containerd
sudo systemctl enable containerd
```

3.5.4 CRI-O

Langkah pertama dalam instalasi runtime CRI-O dilakukan dengan menambahkan repository kubic pada sistem operasi Ubuntu 20.04 yang didalamnya terdapat paket dari CRI-O.

```
echo "deb
https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable/xUb
untu_20.04/" | \
sudo tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list
```

Selanjutnya mengimpor *public key* dari kubic dengan mendownloadnya menggunakan perintah berikut.

```
curl -L
https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/xUbuntu_20.04/Release.key | \
sudo apt-key add -
```

Menambahkan *repository* CRI-O versi 1.23 pada sistem operasi Ubuntu 20.04 dengan mendownloadnya dari *repository* kubic.

```
echo "deb
http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:1.23/xUbuntu_20.04/" | \
sudo tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable:cri-o:1.23.list
```

Selanjutnya mengimpor *public key* dari CRI-O dengan mendownloadnya menggunakan perintah berikut.

```
curl -L
https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable:cri-o:1.23/xUbuntu_20.04/Release.key | \
sudo apt-key add -
```

Setelah menambahkan *repository* dan *public key*, diharuskan mengupdate paket, lalu menginstal CRI-O menggunakan perintah berikut.

```
sudo apt update
sudo apt install cri-o cri-o-runc cri-tools -y
```

Mengaktifkan CRI-O agar dimulai ketika menjalankan sistem operasi Ubuntu.

```
sudo systemctl enable crio.service
```

Setelah itu service CRI-O dapat dimulai dan dijalankan.

```
sudo systemctl start cri-o.service
```

CRI-O dapat di cek untuk memastikan jika service sudah siap berjalan.

```
sudo crictl info
```

3.5.5 Kata Containers

Proses pertama dalam instalasi runtime Kata Containers yaitu membuat dan menyediakan RBAC (*role-based access control*) *roles* yang berbeda untuk *pod* kata-deploy dengan perintah berikut. Role RBAC memiliki format *.yaml yang didapatkan dari repository kata containers. *Role* RBAC pada Kubernetes berfungsi untuk mengatur akses ke *resource* komputer atau jaringan berdasarkan *role* masing-masing *user* dalam organisasi.

```
kubectl apply -f https://raw.githubusercontent.com/kata-containers/kata-containers/main/tools/packaging/kata-deploy/kata-rbac/base/kata-rbac.yaml
```

Selanjutnya membuat *pod* kata-deploy dengan menerapkan versi stable yang didapatkan dari repository kata containers.

```
kubectl apply -f https://raw.githubusercontent.com/kata-containers/kata-containers/main/tools/packaging/kata-deploy/kata-deploy/base/kata-deploy-stable.yaml
```

Mengecek status *pod* kata-deploy di dalam namespace kube-system dengan kondisi Ready.

```
kubectl get pods -n kube-system  
kubectl -n kube-system wait --timeout=10m --for=condition=Ready -l name=kata-deploy pod
```

Mengecek label kata-containers pada node dengan mencari didalam file-file yang berisi kata “kata”.

```
kubectl get nodes --show-labels | grep kata
```


Melakukan konfigurasi class runtime untuk Kata Containers dengan membuat resource Kubernetes pada file runtimeclass.yaml.

```
sudo nano runtimeclass.yaml
```

Konfigurasi class runtime pada file runtimeclass.yaml dengan metrik yaitu kind: RuntimeClass.

```
kind: RuntimeClass
apiVersion: node.k8s.io/v1
metadata:
  name: kata-qemu
handler: kata-qemu
overhead:
  podFixed:
    memory: "160Mi"
    cpu: "250m"
scheduling:
  nodeSelector:
    katacontainers.io/kata-runtime: "true"
```

Meng-*apply* (menerapkan) konfigurasi runtimeclass.yaml yang telah dimodifikasi dengan kubectl.

```
kubectl apply -f runtimeclass.yaml
```

Mengecek semua informasi runtime class dengan deskripsi kata-qemu.

```
kubectl get runtimeclass
kubectl describe runtimeclass kata-qemu
```

3.6 Perancangan Parameter

Parameter yang akan digunakan pada analisis diambil berdasarkan tool analisis yang digunakan. Tools yang digunakan antara lain Netperf yang digunakan untuk mengambil data *latency* dan *throughput*. Tools y-cruncher yang akan digunakan untuk mengambil data penggunaan CPU. Dan tools STREAM yang akan

digunakan untuk mengambil data penggunaan memory. Pengujian yang akan dilakukan yaitu dengan metode benchmark menggunakan ketiga aplikasi yang sudah disebutkan. Benchmark tersebut akan dilakukan pada setiap container yang berjalan. Tabel 3.5 adalah tabel parameter yang diamati pada penelitian.

Tabel 3.5 Parameter Pengujian

Parameter Penelitian	Satuan	Tools
<i>Latency</i>	ms	ping
<i>Throughput</i>	<i>MBps</i>	Netperf
<i>CPU</i>	%	y-cruncher
<i>Memory</i>	<i>MB/s</i>	STREAM

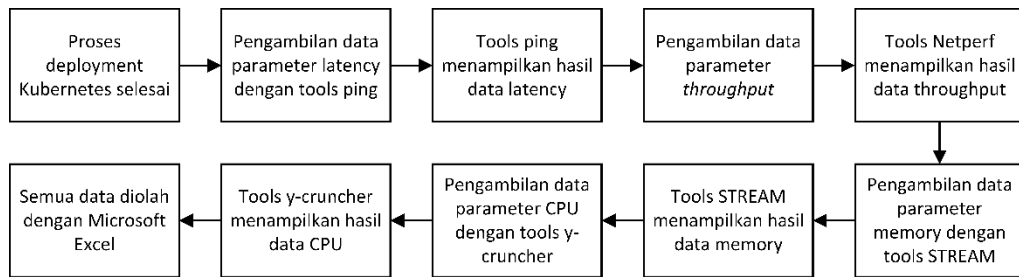
3.7 Proses Pengujian

Proses pengujian yang akan dilakukan dibagi kebeberapa tahapan yaitu persiapan sistem, konfigurasi sistem, dan pengujian sistem. Pada persiapan sistem akan menyiapkan empat Ubuntu server yang akan disimulasikan secara virtual pada *Google Cloud Platform*. Keempat sever tersebut dikonfigurasi sesuai dengan spesifikasi server yang terdapat pada sub bab alat yang digunakan. Selanjutnya masuk pada konfigurasi sistem, pada konfigurasi sistem akan dilakukan konfigurasi pembagian kategori server yang terdiri dari 1 *master node* dan 3 *worker node* yang setiap masing masing sever tersebut telah dilakukan pemasangan Kubernetes didalamnya. Proses terakhir yaitu pengujian sistem dimana pada pengujian ini akan dipastikan apakah seluruh server yang sudah dipasangkan kubernetes dapat berjalan dengan normal atau tidak, dengan cara melakukan pembuatan container sesuai dengan jumlah yang telah ditentukan.

3.8 Pengambilan Data

Proses pengambilan data digambarkan pada Gambar 3.7 menggunakan empat aplikasi (*tools*) yaitu ping, Netperf, y-cruncher, dan STREAM yang dilakukan secara berurutan (bergantian) setelah proses *deployment* pada Kubernetes selesai. Parameter data yang diambil yaitu *latency*, *throughput*, *CPU*, dan *memory*. Proses pengambilan data diulang untuk setiap *container runtime* sebanyak 30 kali

percobaan. Hasil data yang didapat akan diolah dan dibuat ke dalam bentuk grafik menggunakan *software* Microsoft Excel.



Gambar 3.7 Proses Pengambilan Data

3.8.1 Ping

Tools ping digunakan untuk mengambil data *latency*. Proses awal dilakukan dengan menginstal aplikasi terlebih dahulu pada salah satu *container* dengan perintah sebagai berikut. Ping IP container 10.244.1.74, option -c berarti menentukan jumlah paket atau *request* yang dijalankan, yaitu berjumlah 100.

```
ping 10.244.1.74 -c 100
```

Gambar 3.8 merupakan *output* yang dihasilkan oleh *tools* ping dengan data yang akan diambil sebagai *latency* yaitu nilai avg.

```

pratama@crio-master:~$ ping 10.244.1.74 -c 100
PING 10.244.1.74 (10.244.1.74) 56(84) bytes of data.
64 bytes from 10.244.1.74: icmp_seq=1 ttl=63 time=1.92 ms
64 bytes from 10.244.1.74: icmp_seq=2 ttl=63 time=0.361 ms
64 bytes from 10.244.1.74: icmp_seq=3 ttl=63 time=0.334 ms
64 bytes from 10.244.1.74: icmp_seq=4 ttl=63 time=0.337 ms
64 bytes from 10.244.1.74: icmp_seq=5 ttl=63 time=0.365 ms
64 bytes from 10.244.1.74: icmp_seq=6 ttl=63 time=0.484 ms
64 bytes from 10.244.1.74: icmp_seq=7 ttl=63 time=0.321 ms
64 bytes from 10.244.1.74: icmp_seq=8 ttl=63 time=0.294 ms
64 bytes from 10.244.1.74: icmp_seq=9 ttl=63 time=0.368 ms
64 bytes from 10.244.1.74: icmp_seq=10 ttl=63 time=0.552 ms
64 bytes from 10.244.1.74: icmp_seq=11 ttl=63 time=0.288 ms
64 bytes from 10.244.1.74: icmp_seq=12 ttl=63 time=0.363 ms
64 bytes from 10.244.1.74: icmp_seq=13 ttl=63 time=0.333 ms
64 bytes from 10.244.1.74: icmp_seq=14 ttl=63 time=0.354 ms
64 bytes from 10.244.1.74: icmp_seq=15 ttl=63 time=0.296 ms
64 bytes from 10.244.1.74: icmp_seq=97 ttl=63 time=0.351 ms
64 bytes from 10.244.1.74: icmp_seq=98 ttl=63 time=0.354 ms
64 bytes from 10.244.1.74: icmp_seq=99 ttl=63 time=0.505 ms
64 bytes from 10.244.1.74: icmp_seq=100 ttl=63 time=0.363 ms

--- 10.244.1.74 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101349ms
rtt min/avg/max/mdev = 0.262/0.373/1.920/0.168 ms
pratama@crio-master:~$
  
```

Gambar 3.8 Output pada ping

3.8.2 Netperf

Tools Netperf digunakan untuk mengambil data *throughput*. Proses awal dilakukan dengan menginstal aplikasi terlebih dahulu dengan perintah sebagai berikut.

```
sudo apt-get install netperf
```

Setelah itu netperf dijalankan dengan memasukkan IP *address* dari salah satu *container*. Gambar 3.9 merupakan daftar alamat IP *container*. Gambar 3.10 menunjukkan perintah menjalankan netperf pada *worker node* 1 menuju IP salah satu *container* pada *worker node* 3 yaitu dengan IP 10.244.2.145, lalu akan menghasilkan *output* data seperti *receiver socket time*, *send socket time*, *send message size*, *elapsed time*, dan *throughput*.

```
pratama@masternode:~$ kubectl get pod -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE
containercoba-69485f45c9-5bv26     1/1     Running  0           23m   10.244.2.145   workernode03
containercoba-69485f45c9-7jgsx     1/1     Running  0           23m   10.244.3.135   workernode01
containercoba-69485f45c9-8h792     1/1     Running  0           23m   10.244.1.138   workernode02
containercoba-69485f45c9-8x8sx     1/1     Running  0           23m   10.244.3.136   workernode01
containercoba-69485f45c9-dckxw     1/1     Running  0           23m   10.244.2.146   workernode03
containercoba-69485f45c9-lzmt7     1/1     Running  0           23m   10.244.1.140   workernode02
containercoba-69485f45c9-m688f     1/1     Running  0           23m   10.244.2.147   workernode03
containercoba-69485f45c9-npnnv     1/1     Running  0           23m   10.244.3.134   workernode01
containercoba-69485f45c9-nqddw     1/1     Running  0           23m   10.244.1.139   workernode02
containercoba-69485f45c9-wdbwt     1/1     Running  0           23m   10.244.1.137   workernode02
```

Gambar 3.9 Alamat IP *container*

```
pratama@workernode01:~$ netperf 10.244.2.145
Recv  Send  Send
Socket Socket Message Elapsed
Size  Size  Size  Time  Throughput
bytes bytes bytes secs.  10^6bits/sec
131072 16384 16384 10.00 17211.72
pratama@workernode01:~$
```

Gambar 3.10 *Output* pada Netperf

3.8.3 STREAM

Tools STREAM digunakan untuk mengambil data *memory*. Proses awal dilakukan dengan menginstal aplikasi terlebih dahulu pada salah satu *container* dengan perintah sebagai berikut. STREAM diinstal dengan men-*download* paket pada *website* hpcc, yang nantinya akan diekstrak dan menyalin (*copy*) file berekstensi *.h yang berisi *script* konfigurasi hpcc ke folder STREAM.

```
curl -O https://www.cs.virginia.edu/stream/FTP/Code/stream.c
curl -O https://hpcchallenge.org/projectsfiles/hpcc/download/hpcc-1.5.0.tar.gz
curl -O https://www.cs.virginia.edu/stream/FTP/Code/Versions/stream_mpi.c
curl -O https://www.cs.virginia.edu/stream/FTP/Code/Versions/stream_omp.c
tar -xvzf hpcc-1.5.0.tar.gz
sudo apt install gcc
```

```

cp "/home/pratama/hpcc-1.5.0/include/hpcc.h" /home/pratama/hpcc-
1.5.0/STREAM/
gcc -fopenmp -D_OPENMP stream.c -o stream
export OMP_NUM_THREADS=2
./stream

```

Gambar 3.11 menunjukkan perintah yang digunakan untuk menjalankan stream yaitu `./stream`. *Output* yang dihasilkan oleh STREAM berupa data *memory* yang terbagi menjadi *copy*, *scale*, *add*, dan *triad* yang masing masing memiliki nilai *best rate*, *average time*, *minimum time*, dan *maximum time*. Pada penelitian ini hanya mengambil data dari *best rate Copy* dan *Scale*.

```

root@containercrio-69485f45c9-26t9s:/# ./stream
-----
STREAM version $Revision: 5.10 $
-----
This system uses 8 bytes per array element.
-----
Array size = 10000000 (elements), Offset = 0 (elements)
Memory per array = 76.3 MiB (= 0.1 GiB).
Total memory required = 228.9 MiB (= 0.2 GiB).
Each kernel will be executed 10 times.
The *best* time for each kernel (excluding the first iteration)
will be used to compute the reported bandwidth.
-----
Number of Threads requested = 2
Number of Threads counted = 2
-----
Your clock granularity/precision appears to be 1 microseconds.
Each test below will take on the order of 17280 microseconds.
 (= 17280 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-----
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-----
Function Best Rate MB/s Avg time Min time Max time
Copy: 8742.7 0.018751 0.018301 0.021743
Scale: 8741.8 0.018524 0.018303 0.019447
Add: 11553.0 0.021054 0.020774 0.021521
Triad: 10594.7 0.023234 0.022653 0.027078
-----
Solution Validates: avg error less than 1.000000e-13 on all three arrays
-----

```

Gambar 3.11 Output pada STREAM

3.8.4 Y-Cruncher

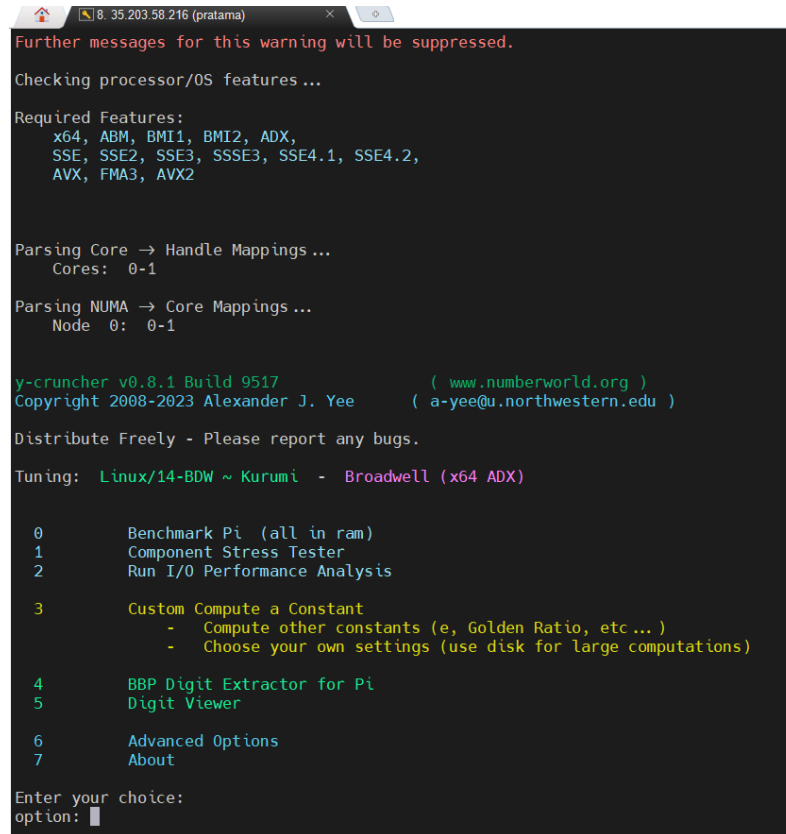
Tools y-cruncher digunakan untuk mengambil data CPU. Proses awal dilakukan dengan menginstal aplikasi terlebih dahulu pada salah satu *container* dengan menggunakan perintah berikut.

```

curl -O http://www.numberworld.org/y-cruncher/y-cruncher%20v0.7.10.9513-
static.tar.xz
tar -xf y-cruncher%20v0.7.1.9466-static.tar.xz
cd y-cruncher v0.7.1.9466-static
./y-cruncher

```

Tampilan awal y-cruncher ditunjukkan pada Gambar 3.12, terlihat beberapa option yang dapat dipilih untuk menghitung parameter yang dibutuhkan. Penelitian ini menggunakan *option 0* yaitu Benchmark Pi (*all in ram*) yang akan mengukur kinerja RAM.



```
Further messages for this warning will be suppressed.
Checking processor/OS features ...

Required Features:
x64, ABM, BMI1, BMI2, ADX,
SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2,
AVX, FMA3, AVX2

Parsing Core → Handle Mappings ...
Cores: 0-1

Parsing NUMA → Core Mappings ...
Node 0: 0-1

y-cruncher v0.8.1 Build 9517 ( www.numberworld.org )
Copyright 2008-2023 Alexander J. Yee ( a-yee@u.northwestern.edu )

Distribute Freely - Please report any bugs.

Tuning: Linux/14-BDW ~ Kurumi - Broadwell (x64 ADX)

0      Benchmark Pi (all in ram)
1      Component Stress Tester
2      Run I/O Performance Analysis

3      Custom Compute a Constant
        - Compute other constants (e, Golden Ratio, etc...)
        - Choose your own settings (use disk for large computations)

4      BBP Digit Extractor for Pi
5      Digit Viewer

6      Advanced Options
7      About

Enter your choice:
option: █
```

Gambar 3.12 Tampilan awal y-cruncher

Gambar 3.13 merupakan prose pemilihan besarnya ukuran Pi pada y-cruncher yaitu sebesar 25.000.000 digits. Gambar 3.14 merupakan *output* yang dihasilkan y-cruncher yaitu data CPU *utilization*, *multi-core efficiency*, dan juga *kernel overhead*. Pada penelitian ini hanya mengambil data dari CPU *utilization* dan *multi-core efficiency*.

```

0.35.203.58.216 (pratama)
5 Digit Viewer
6 Advanced Options
7 About
Enter your choice:
option: 0
Benchmark Pi:
Select a Benchmark Type:
0 Single-Threaded
1 Multi-Threaded
option: 0
Available Memory: 7.11 GiB
Option Decimal Digits Approx. Memory Needed
1 25,000,000 129 MiB
2 50,000,000 244 MiB
3 100,000,000 468 MiB
4 250,000,000 1.12 GiB
5 500,000,000 2.27 GiB
6 1,000,000,000 4.54 GiB
7 2,500,000,000 10.8 GiB
8 5,000,000,000 23.0 GiB
9 10,000,000,000 46.1 GiB
10 25,000,000,000 111 GiB
11 50,000,000,000 222 GiB
12 100,000,000,000 445 GiB
13 250,000,000,000 1.09 TiB
14 500,000,000,000 2.18 TiB
15 1,000,000,000,000 4.48 TiB
16 2,500,000,000,000 11.2 TiB
0 I prefer SuperPi sizes ... (1M, 2M, 4M ... )
option: 1

```

Gambar 3.13 Pemilihan Pi digits pada y-cruncher

```

Begin Computation:
Series CommonP2B3... 1,762,854 terms (Expansion Factor = 2.360)
Time: 6.538 seconds ( 0.109 minutes )
Large Division...
Time: 0.481 seconds ( 0.008 minutes )
InvSqrt(10005)...
Time: 0.291 seconds ( 0.005 minutes )
Large Multiply...
Time: 0.225 seconds ( 0.004 minutes )
Pi: 7.537 seconds ( 0.126 minutes )
Base Converting:
Time: 0.703 seconds ( 0.012 minutes )
Writing Decimal Digits:
Time: 0.110 seconds ( 0.002 minutes )
Start Time: Fri Aug 11 05:08:23 2023
End Time: Fri Aug 11 05:08:31 2023
Total Computation Time: 8.240 seconds ( 0.137 minutes )
Start-to-End Wall Time: 8.507 seconds ( 0.142 minutes )
CPU Utilization: 99.63 % + 0.00 % kernel overhead
Multi-core Efficiency: 49.82 % + 0.00 % kernel overhead
Last Decimal Digits: Pi
3803750790 9491563108 2381689226 7224175329 0045253446 : 24,999,950
0786411592 4597806944 2455112852 2554677483 6191884322 : 25,000,000
Spot Check: Good through 25,000,000
Version: 0.8.1.9517 (Linux/14-BDW ~ Kurumi)
Processor(s): Intel(R) Xeon(R) CPU @ 2.20GHz
Topology: 2 threads / 1 core / 1 socket / 1 NUMA node
Usable Memory: 8,332,730,368 (7.76 GiB)
CPU Base Frequency: 2,200,000,992 Hz
Validation File: Pi - 20230811-050831.txt
root@containercoba-69485f45c9-5bv26:/home/pratama/y-cruncher v0.8.1.9317-static#

```

Gambar 3.14 Output pada y-cruncher

3.9 Analisis Data

Analisis data merupakan tahap yang dilakukan ketika pengambilan data dari hasil simulasi selesai dan dapat menghasilkan keluaran yang dapat dianalisis. Analisis yang dilakukan yaitu dengan membandingkan hasil keempat parameter yaitu *latency*, *throughput*, *CPU*, dan *memory*. Menggunakan dua skenario yaitu *container runtime* (Containerd, CRI-O, dan Kata Containers) dan skalabilitas dengan jumlah container sebanyak 10, 20, dan 40 container.