Perancangan Dan Implementasi Load Balancing Menggunakan Algoritma Least Connection Dan Ip Hash Pada Kubernetes

1st Hanif Luthfi
Fakultas Ilmu Terapan
Universitas Telkom
Bandung, Indonesia
haaanifluthfiiii@student.telkomuniversity.ac.id

2nd Rohmat Tulloh
Fakultas Ilmu Terapan
Universitas Telkom
Bandung, Indonesia
rohmatth@telkomuniversity.ac.id

3rd Muhammad Iqbal
Fakultas Ilmu Terapan
Universitas Telkom
Bandung, Indonesia
miqbal@telkomuniversity.ac.id

***Abstrak*— Saat ini dunia telah memasuki Industry 4.0, dimana teknologi sudah banyak dapat ditemukan dalam kehidpan sehari-hari. Bahkan teknologi tidak bisa lepas begitu saja, karena teknologi telah dapat mempermudah dan membantu manusia dalam mengerjakan segala sesuatu, baik itu dalam mencari suatu informasi atau berkomunikasi dengan jarak jauh. Teknologi sudah dapat terintegrasi secara penuh dengan internet, sehingga sistem maupun perangkat dapat melakukan komunikasi dan saling bertukar informasi melalui internet. Bahkan, layanan yang disediakan saat ini sudah massive dan banyak ditemukan dalam jaringan online. Dengan perkembangan tersebut, tentunya skalabilitas dari layanan tersebut sangat besar dan semakin kompleks. Bahkan server dapat menampung layanan tersebut dengan jumlah data yang besar dan sangat cepat. Trafik yang diterima oleh server harus memiliki spesifikasi hardware yang mumpuni untuk mengatasi overload. Jika sewaktu-waktu server tidak dapat menangani jumlah trafik yang sangat besar, maka server akan mengalami down dan tidak dapat melayani berbagai aplikasi yang disediakan oleh server. Pada Implementasir ini dilakukan perancangan dan implementasi load balancing menggunakan dua algoritma yaitu Least Connection dan IP Hash pada layanan Kubernetes yang disediakan oleh Oracle Cloud Infrastructure. Kemudian, dilakukan analisis untuk perbandingan dari penggunaan dua Algorima tersebut. Perancangan ini menggunakan platform Kubernetes yaitu sebuah platform untuk membuat kluster server dengan konfigurasi 3 virtual server. Kemudian dilakukan untuk mengukur performa proses dari aplikasi yang dijalankan berdasarkan parameter response time, throughput, request loss, serta performa penggunaan sumber daya komputasi dari server seperti CPU Utilization.**

***Kata kunci*—Load Balancing, Least Connection, IP Hash, Kubernetes, Cloud Computing**

I. PENDAHULUAN

Seiring berjalan nya waktu, *internet* memiliki perkembangan yang sangat pesat dari tahun ke tahun. Layanan yang ditawarkan sudah sangat kompleks dan meluas, seperti layanan aplikasi *web*, layanan media penyimpanan *online*, layanan *cloud computing*, *streaming* video, *game online*, dan masih banyak lagi. Tentu saja dengan berkembangnya *internet* dapat mempengaruhi dampak suatu infrastruktur dan skalabilitas jaringan yang kompleks pula, dimana paket data melewati berbagai banyak trafik yang dikirimkan ke *server* dalam waktu sangat cepat dan bahkan dapat menampung jumlah data yang sangat besar. Dalam hal ini perlu dipertimbangkan jika suatu kasus pada *server* sewaktu-waktu mengalami kondisi *overload* ataupun *down*. Jika *server* mengalami kondisi *down*, maka *server* tidak dapat menjalankan aplikasi-aplikasi yang disediakan oleh *server* tersebut tidak tersedia. Oleh karena itu, perlu dibutuhkan solusi yang tepat untuk mengatasi masalah tersebut dengan cara menggunakan metode *load balancing* dan membangun suatu klaster *server* menggunakan platform *Kubernetes* agar dapat mengurangi penggunaan sumber daya perangkat berlebih dan mampu meringkas konfigurasi terhadap aplikasi agar dapat berjalan dengan baik.

Terdapat pada penelitian [1] penggunaan *Load Balancing Web Server* menggunakan teknologi *Software Defined Network (SDN)* dan menggunakan algoritma yaitu *Source IP Hash* dengan jumlah request yang lebih sedikit sehingga tidak memberikan hasil yang maksimal pada pengujian nya. Pada penelitian [2] menggunakan algoritma *Least Connection* dan *Round Robin* sebagai pembanding dan diterapkan pada *Docker Swarm* sebagai klaster *server*, namun untuk hasil pengujian yang didapat belum maksimal pada *Least Connection*. Pada penilitan [3] menganalisis performansi *Least Connection* dan *IP Hash*, namun tidak melakukan pengukuran dari penggunaan *CPU*, sehingga tidak dapat mengetahui berapa penggunaan *CPU* ketika menggunakan algoritma tersebut. Pada Penelitian [4] dilakukan pembuatan aplikasi layanan web E-Commerce pada platform *docker*, selanjutnya akan dilakukan simulasi untuk melihat performa dari sistem yang telah dikembangkan berdasarkan parameter *throughput*, *response time*, *CPU utilization*, dan *memory utilization*.

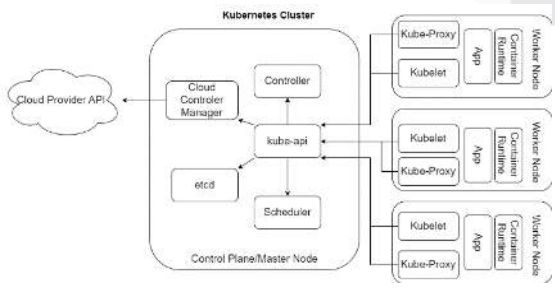
Perancangan serta implementasi ini dilakukan dua algoritma sebagai pembanding, yaitu algoritma *Least Connection* dan *IP Hash* yang akan diimplementasikan pada platform *kubernetes* menggunakan salah satu layanan *cloud computing* yaitu *Oracle Cloud Infrastrucutre*. Klaster tersebut terdiri dari 3 *virtual server* sebagai *worker node* atau *backend server* dimana melakukan *processing* dari layanan aplikasi *web* Wordpress sekaligus *MySQL* yang diisolasi dalam bentuk objek *Pod* pada masing-masing *worker node*. Selanjutnya dilakukan dengan kombinasi layanan penyimpanan pada *oracle cloud* menggunakan *File Storage* sebagai media penyimpanan data dari aplikasi pod pada kubernetes cluster. Hal tersebut dilakukan untuk mengurangi penggunaan disk serta dapat mencegah terjadinya hilang data

pada saat suatu kasus server tersebut atau *pod* tidak berfungsi dengan baik. Kemudian, dibutuhkan *virtual client* yang terinstall pada *Vmware Workstation* sebagai wadah untuk menjalankan simulasi *load testing* untuk membangkitkan trafik beban *request HTTP* dengan jumlah yang besar menuju ke *load balancer* dengan bantuan aplikasi yaitu *Apache JMeter*. Kemudian *load balancer* melakukan pembagian beban trafik permintaan dari client berdasarkan algoritma yang digunakan sehingga request tersebut akan diteruskan menuju *worker node* untuk melayani permintaan dari client selanjutnya. Setelah melakukan pengujian pada dua algoritma tersebut, dilakukan analisis lebih lanjut terkait dengan penggunaan algoritma *Least Connetcion* dan *IP Hash* dengan mengukur performa *server* seperti *CPU Utillization* pada masing-masing *worker node.* Karena hal tersebut merupakan metric pengukuran paling penting untuk mengukur kinerja dan dapat menguji performa *server*. Kemudian dilanjut dengan mengukur performa aplikasi yang digunakan dengan pengukuran *response time*, *throughput*, dan *request loss* agar mengetahui seberapa baik masing-masing *worker node* memberikan layanan aplikasi web saat sedang proses permintaan oleh client.

II. DASAR TEORI

A. Kubernetes

Kubernetes merupakan salah satu platform *open-source container orchestration* yang digunakan untuk membantu melakukan pengelolaan *workloads* aplikasi yang telah dikontainerisasi dalam skala besar. *Kubernetes* telah menyediakan berbagai macam konfigurasi dan *automation* secara deklaratif untuk mendukung pengembangan aplikasi berbasis *container*. *Kubernetes* berada di dalam ekosistem yang besar dan berkembang cepat karena berbagai *service*, *support*, *addons*, dan peralatan lainnya tersedia secara meluas [5]. Dengan adanya *Kubernetes* memungkinkan pengguna dengan mudah untuk mengembangkan serta meluncurkan aplikasi web yang kompleks terdiri dari beberapa cluster *container* yang saling terintegrasi seolah-olah meluncurkan hanya 1 program aplikasi saja. Fitur yang ditawarkan oleh *Kubernetes* sudah sangat lengkap, seperti *deployment*, *scaling, load balancing*, *logging*, dan *monitoring* sehingga dapat mempermudah *Development* dan *Operation* untuk mengembangkan, mengatur dan men-deploy aplikasi kedalam *server.*



GAMBAR 2. 1
Arsitektur Kubernetes

Gambar 2.1 merupakan arsitektur secara dasar terkait dengan komponen dalam *Kubernetes*. *Kubernetes* dapat membentuk suatu kluster *server*, yang dimana terdapat *Master Node* dan *Worker Node* yang saling terhubung. Berikut adalah penjelasan dari masing-masing fungsi dari node:

1. Master Node

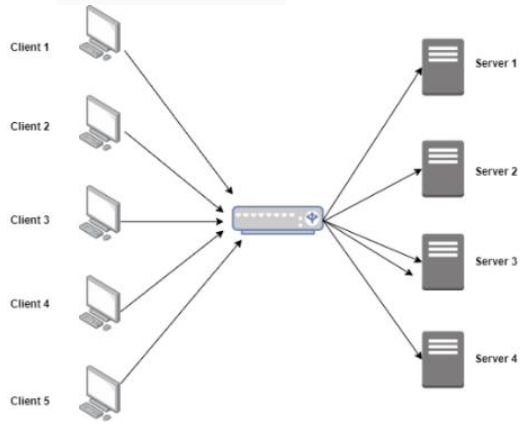
Master Node atau dapat didefiniskan sebagai *control plane* yang dapat mengatur serta mengarahkan seluruh *worker node* yang terhubung pada cluster *server*. *Master node* mampu mengambil proses deteksi serta pemberian respons terhadap *log event* yang sedang berlangsung pada cluster. Komponen yang terdapat pada master node yaitu *kube-apiserver*, *kube-controller-manager*, *kubescheduller*, *cloud-controller-manager*, dan *etcd* [6].

2. Worker Node

Worker Node merupakan node yang mampu menjalankan *workloads runtime* aplikasi dan service yang diisolasi dengan menggunakan *container* kedalam *Pod*. *Pod* merupakan objek unit terkecil dari *Kubernetes* yang membungkus satu atau lebih *container* dan dapat mengatur seluruh proses dari *container*. Komponen yang terdapat pada *worker node* yaitu *kubelet*, *container-runtime*, dan *kube-proxy*. Setiap *daemon* pada *kubelet* akan menjalankan pod, tergantung dengan informasi dari spesifikasi pod yang diperoleh dari api*server* pada master node [7].

B. Load Balancing

Load balancing merupakan metode untuk mendistribusikan beban trafik jaringan yang masuk kedalam *server* aktif dan membagi trafik data tergantung algoritma yang diterapkan pada *load balancer*. Dengan menggunakan *load balancing*, dapat mengurangi overload terhadap sumber daya komputasi yang digunakan sehingga membuat *server* menjadi lebih efisien, mempercepat kinerja aplikasi dan mengurangi *latency* [8].

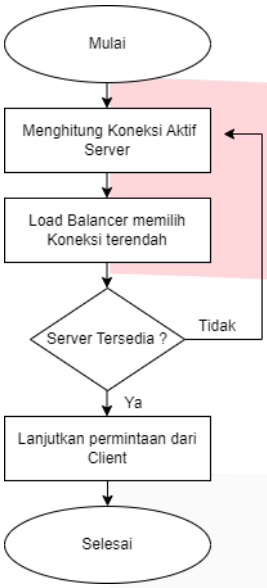


GAMBAR 2.2
Ilustrasi Load Balancing

Gambar 2.2 merupakan ilustrasi dari penggunaan *load balancing* secara mendasar. Pada umumnya, aplikasi pada masing-masing *server* memiliki konten serta data yang sama, sehingga sesuai dengan fungsi dari *load balancer* untuk melakukan pendistribusian beban kerja *request* dari *client* dengan cara mengoptimalkan setiap *resource server*, mencegah terjadinya *overload* pada seluruh *server*, dan dapat meningkatkan respon yang memungkinkan untuk *client*.

C. Least Connection

Least Connection merupakan salah satu algoritma yang populer digunakan dalam pemrosesan sistem *load balancing*. *Least Connection* akan mendistribusikan serta mengarahkan koneksi dari client menuju *server* berdasarkan jumlah koneksi yang dibuat paling sedikit [9], [10]. Ketika salah satu *server* mengangani banyak koneksi yang masuk, maka koneksi akan dipindahkan menuju *server* yang menangani koneksi yang paling sedikit/kecil. Tentu saja dengan algoritma ini dapat mempertahankan distribusi beban koneksi aktif yang setara dengan *backend server*.

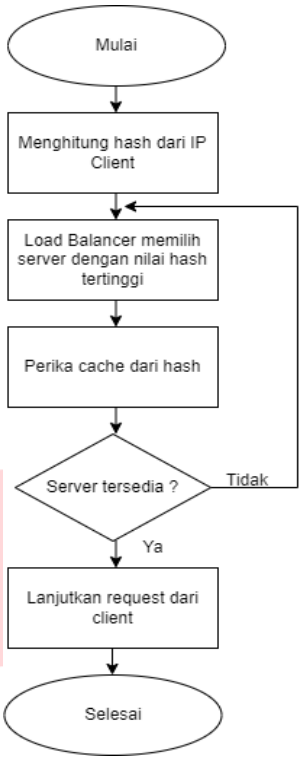


Gambar 2.3
Diagram Alir Least Connection

Least Connection akan melakukan perhitungan untuk koneksi aktif dari *server* yang terhubung dengan *Load Balancer*. Kemudian, *Load Balancer* akan melakukan pengambilan koneksi yang paling terkecil/rendah pada masing-masing *server*. Jika *server* dengan koneksi terkecil tersedia, maka *load balancer* akan memilih *server* tersebut untuk menerima paket *request* dari *client* dan jika *server* tidak tersedia, maka *load balancer* akan melakukan perhitungan ulang kembali untuk mencari koneksi aktif *server* yang paling terkecil.

D. IP Hash

IP Hash merupakan algoritma load balancing yang menggunakan *IP Address* dari sumber, kemudian dilakukan metode *hashing* sebagai kunci untuk mengarahkan trafik beban menuju *server* yang sama dan *server* tersebut akan menerima respon request tersebut. Hal tersebut dapat memastikan bahwa alamat *IP client* yang sama akan mencapai *server* yang sama selama tidak terjadi apablia *server* yang mengalami kondisi padat atau *down* [11].



Gambar 2.4
Diagram Alir IP Hash

IP Hash melakukan perhitungan terhadap nilai *hash* dari *IP Client* yang ingin melakukan *request*. Kemudian, *load balancer* akan memilih *server* dengan nilai hash yang tertinggi untuk dicocokkan dengan hash dari *IP Client*. *Load balancer* akan melakukan pemeriksaan terhadap *cache* cari nilai *hash* tersebut, jika *server* tersedia maka *load balancer* akan melanjutkan *request* dari *client* dan jika tidak maka *load balancer* akan melakukan perhitungan kembali untuk mendapatkan nilai *hash* yang sesuai. Perlu diketahui bahwa *client* yang terhubung dengan jaringan *internet* memiliki *IP Address* yang sama. Jika menerapkan algoritma *IP Hash* pada *backend server*, *load balancer* akan merutekan *traffic* berdasarkan *IP Address* yang masuk dan mengirimkan *request client* dari *internet* ke *server backend* yang sama.

E. Cloud Computing

Cloud Computing adalah teknologi jaringan komputasi berbasis *internet* yang berkembang pesat dari tahun ke tahun dengan memberikan layanan kepada pelanggan berbagai sumber daya komputasi dan utilitas seperti penyediaan *Compute, Network*, *Database*, *Object Storage*, *Monitoring*, *Analytics*, dll [12]. Pelanggan dapat menyewa dan menggunakan berbagai sumber daya sesuai dengan kebutuhannya sesuai dengan ketentuan dan kebijakan layanan dari masing-masing penyedia layanan *Cloud Computing*. *Cloud Computing* juga dapat memangkas biaya awal dan meminimalkan penggunaan biaya operasional, karena menggunakan konsep *Pay as You Go* yang artinya pengguna dapat membayar sesuai dengan pemakaian yang dibutuhkan. Keunggulan *cloud computing* antara lain sebagai berikut [13]:

1. Mengurangi kebutuhan daya komputasi PC.

2. Peningkatan fault tolerance dan security.
3. Mampu meningkatkan kecepatan pemrosesan data berkali-kali lipat.
4. Biaya perangkat keras, perangkat lunak, maintenance, daya berkurang, dan dapat menghemat penggunaan disk.

Cloud Computing termasuk kedalam komputasi yang bersifat terdistribusi secara dinamis mampu menyediakan sumber daya komputasi dengan jarak jauh, mudah, sekuritas yang tinggi, serta terukur dalam pembayaran per penggunaan layanan. Karena skalabilitas yang tergolong mudah dan *high availability*, penggunaan biaya pada *cloud computing* telah menjadi suatu tren teknologi yang signifikan, sehingga *cloud computing* tidak hanya menyediakan layanan komputasi saja, namun mampu menyediakan utilitas yang sesuai dengan kebutuhan *customers*.

F. Virtualization

Virtualization merupakan teknik untuk melakukan emulasi yang dapat mereplika bentuk mesin komputasi secara *virtual* dan memisahkan penggunaan sistem operasi, data, jaringan, memori, I/O diatas *hardware* fisik. Dapat diasumsikan bahwa *virtualization* dapat melakukan sharing sumber daya dari perangkat fisik tanpa menggangu sistem komputasi atau aplikasi lain [14]. Tentunya penggunaan *virtualization* dapat membagi dan memanfaatkan sumber daya berlebih yang dimiliki oleh perangkat fisik tanpa perlu membangun perangkat fisik baru dengan spesifikasi yang sama. *Virtualisasi* dilakukan untuk platform perangkat keras tertentu melalui perangkat lunak khusus (aplikasi kontrol) yang menciptakan lingkungan komputer yang disimulasikan (mesin virtual) untuk *host* perangkat lunak, dimana perangkat lunak tersebut berjalan seolah-olah diinstall pada platform perangkat keras secara terpisah [15]. *Virtualisasi* dapat dikategorikan memjadi 2, yaitu virtualsiasis perangkat keras dan *virtualisasi* perangkat lunak.

G. Container

Container adalah standard dalam suatu unit pada software yang mengemas code dan seluruh depedensinya sehingga aplikasi tersebut dapat berjalan dengan cepat dari satu lingkungan komputasi ke lingkungan komputasi lain [16]. Artinya, *container* dapat menampung berbagai banyak data seperti library, code, registry images dalam wadah yang terisolasi berbentuk *container* dan bisa dijalankan pada komputasi manapun dikarenakan *container* bersifat praktis dan ringan. *Container* mampu menyediakan peningkatan pada virtualisasi OS dengan menggunakan fitur kernel untuk mengisolasi terhadap batas proses penggunaan sistem sumber daya komputasi seperti *CPU*, *Memory*, *Disk*, serta Jaringan.

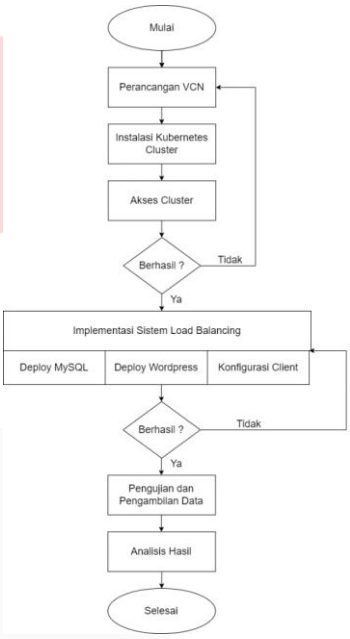
H. Apace JMeter

Apache JMeter adalah tools perangkat lunak *opensource* dibuat dengan bahasa Java didesain untuk melakukan *test workloads* dan performa dari layanan yang diberikan oleh *server*. Layanan tersebut mampu menguji performa dari *web server*, *FTP server*, *VoIP*, dan lain-lain. Tools ini digunakan untuk melakukan simulasi terhadap performa aplikasi web dengan membangiktan trafik *request* jumlah yang sangat banyak dan didistribusikan menuju *server* [17].

III. PERANCANGAN SISTEM

A. Diagram Alir Perancangan

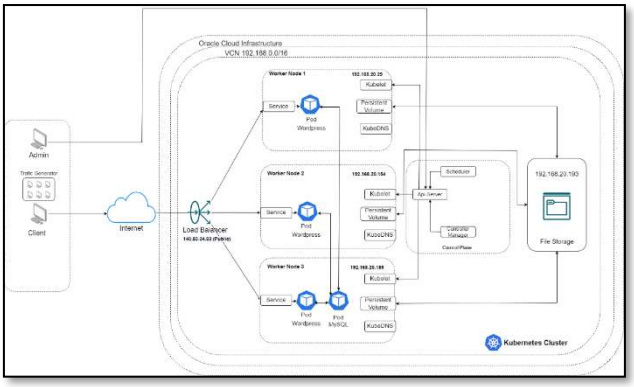
Perancangan sistem ini memiliki tahapan-tahapan pendukung dalam proses perancangan load balancing. Berikut merupakan diagram alir dari perancangan dan implementasi sistem load balancing pada Kubernetes.



GAMBAR 3.1
Diagram Alir Perancangan Sistem

Gambar 3.1 merupakan diagram alir dari perancangan sistem load balancing pada kubernetes. Sistem ini akan menjelaskan penggunaan teknologi *cloud computing*, dimana *cloud computing* merupakan teknologi yang saat ini banyak digunakan dalam infrastruktur jaringan. Algoritma dari *load balancer* yang digunakan adalah algoritma *Least Connection* dan algoritma *IP Hash* yang akan diimplementasikan pada *Kubernetes Cluster*. Penggunaan dari kedua algoritma ini dipilih untuk melakukan ujicoba terkait pengimplementasian pada *Oracle Cloud* menggunakan platform *Kubernetes* sebagai *clustering server* agar dapat terintegrasi satu sama lain. Pengujian sistem menggunakan layanan aplikasi berupa *Wordpress* sebagai *front-end system* yang terintegrasi dengan *MySQL* sebagai *back-end system* untuk melakukan penyimpanan *database* dari *wordpress*. Jika proses implementasi telah berhasil, maka dilakukan untuk pengujian sample serta pengambilan data berdasarkan parameter seperti *Response time*, *throughput,* serta *request loss*. kemudian penggunaan *resource* disetiap masing-masing *server* berupa penggunan *CPU*.

B. Desain Sistem



GAMBAR 3.2
Desain Sistem Perancangan

Gambar 3.2 merupakan desain sistem dari perancangan sistem Load Balancing pada Kubernetes yang diimplementasikan pada layanan Oracle Cloud Infrastructure. *Kubernetes* dijadikan sebagai *cluster server* yang telah disediakan oleh *Oracle Cloud Infractruture*, sehingga dapat mempermudah perancangan dan instalasi. Setiap *Node* yang terdapat pada *Kubernetes Cluster* menggunakan *Virtual Machine* dengan sistem operasi yang dijalankan adalah *oracle linux*. *Control Plane* akan mengatur seluruh operasi kerja dari masing-masing *Worker Node*. *Admin* melakukan perancangan terhadap seluruh komponen di *Kubernetes Cluster* dengan cara melakukan *deploying* berupa file dengan format yaml, dimana terdapat berbagai script seperti perancangan berupa *Pod* untuk *web server* dan *services load balancer*. File yaml akan di deploy melalui *Control Plane*, kemudian akan diproses pada *apiserver* dan diteruskan pada *kubelet* yang merupakan perantara komunikasi dengan *Control Plane*. *File Storage* digunakan sebagai konfigurasi *persistent volume* agar masing-masing *node* memiliki penyimpanan untuk data aplikasi. *Load Balancer* dihubungkan pada masing-masing *Worker Node* sebagai *backend server* agar trafik data yang masuk kedalam *load balancer* dapat terdistribusi dan seimbang menuju *backend server*. Perancangan sistem ini menggunakan dua algoritma *Least Connection* dan *IP Hash* sebagai pembanding dengan pengujian secara bergiliran yang telah terkonfigurasi pada *Load Balancer* yang disediakan oleh *Oracle Cloud Infrastructure*. *Load Balancer* akan dikonfigurasi dalam mode *public*, sehingga *client* dapat melakukan akses web *server* melalui jaringan *internet*. Pada sisi client menggunakan peralatan pendukung seperti traffic generator yaitu Apache JMeter sebagai pembangkit beban trafik data dari client dalam jumlah yang banyak menuju ke Load Balancer.

C. Penggunaan Infrastruktur Sistem

Penggunaan infrastruktur pendukung sistem dibagi menjadi dua bagian, yaitu penggunaan perangkat keras dan perangkat lunak. Infrastruktur sistem yang digunakan dalam implementasi sistem dirincikan sebagai berikut:

TABEL 3. 1
Spesifikasi Worker Node

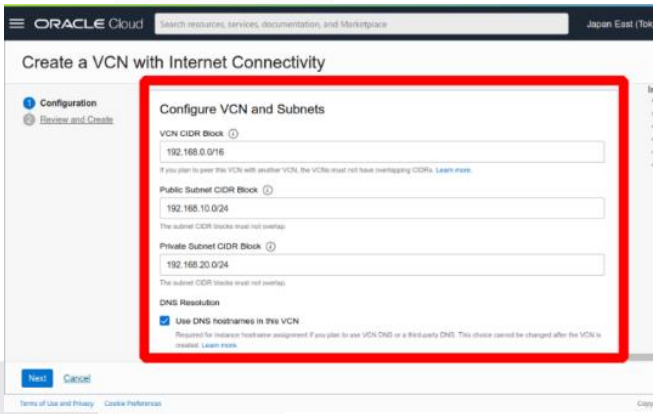
Type	Spesifikasi
Central Processing Unit	1 OCPU AMD EPYC 7742. Base frequency 2.25 GHz
Memory	8 GB
Disk	Block Storage Only
Operating System	Oracle Linux 8

TABEL 3. 2
Spesifikasi Client

Type	Spesifikasi
Central Processing Unit	2 Core
Memory	4 GB
Disk	50 GB
Operating System	Ubuntu Linux 18.04

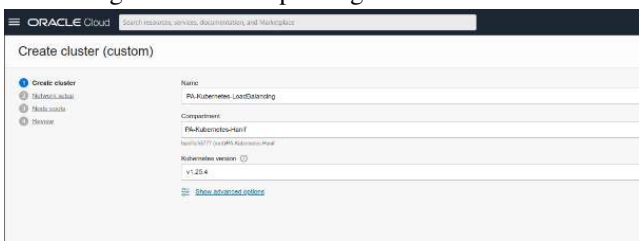
D. Instalasi dan Konfigurasi

Pada bagian ini akan dijelaskan mengenai tahapan instalasi dan konfigurasi dari sistem *Virtual Cloud Network*, *Oracle Kubernetes Engine* (*OKE*), Deployment untuk *Wordpress* dan *MySQL*, dan *Client* sebagai wadah untuk mejalankan simulasi *load testing* menggunakan *Apache JMeter*



Gambar 3.3
Pembuatan Virtual Cloud Network

Gambar 3.3 merupakan konfigurasi VCN pada *Oracle Cloud Infrastructure*. Konfigurasi ini berisi Blok CIDR VCN sebagai blok alamat jaringan, kemudian VCN dibagi menjadi dua jenis blok subnet, yaitu subnet publik, dan subnet pribadi. Tujuannya agar dapat membentuk infrastruktur jaringan yang tertata dengan baik sesuai pembagian blok subnet.

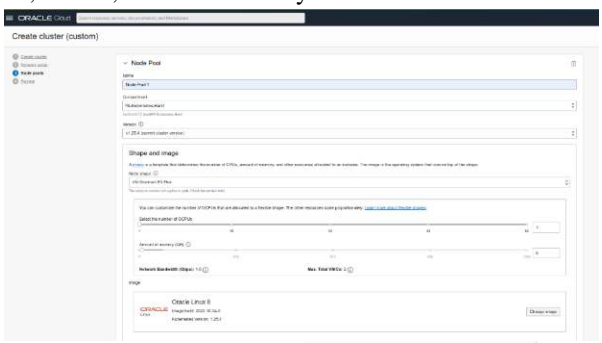


GAMBAR 3.4
Perancangan Kubernetes Custom Cluster

Gambar 3.4 adalah bagian pertama dari proses konfigurasi *cluster Kubernetes*. Versi yang digunakan untuk *Kubernetes Cluster* adalah v1.25.1 yang merupakan versi terbaru dari *Kubernetes Cluster* untuk meminimalisir bug yang terjadi pada versi sebelumnya.

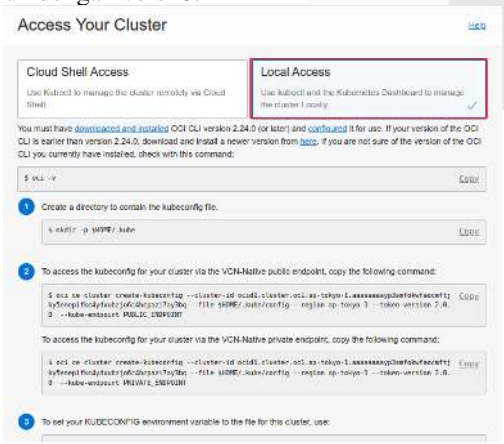
GAMBAR 3. 5
Network Setup Kubernetes

Pada Gambar 3.5 merupakan bagian Network Setup, menggunakan jaringan tipe Flannel Overlay, dimana fungsi nya sebagai komunikasi antara pods satu dengan pods lain yang berbeda Node melalui jaringan overlay. VCN yang digunakan adalah Kubernetes-VCN serta beberapa subnet yang dibutuhkan sesuai dengan permintaan konfigurasi. Kemudian, bagian Node Pool dilakukan konfigurasi terhadap node yang akan digunakan. Konfigurasi tersebut mencakup spesifikasi dari node, sistem operasi yang digunakan, jumlah node, subnet, serta Availability Domain.



GAMBAR 3.6
Penentuan Spesifikasi Node

Gambar 3.6 adalah bagian terakhir dari proses konfigurasi cluster Kubernetes. Shape yang digunakan adalah VM.Standard.E3.Flex dengan spesifikasi 1 OCPU dan memori 8GB dengan sistem operasi yang dijalankan adalah Oracle Linux dengan versi 8.



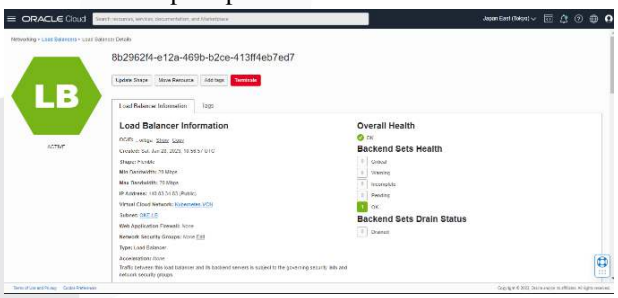
GAMBAR 3.7
Tahapan akses kluster

Gambar 3.7 merupakan metode untuk mengakses cluster Kubernetes yang telah dibuat. Ada dua cara yang bisa dilakukan yaitu melalui Cloud Shell Access dan Local Access. Perancangan kali ini menggunakan metode akses lokal sehingga lebih fleksibel menggunakan perintah dari *kubernetes*. Proses *deployment* membutuhkan file script dengan format yaml. Perintah ini dapat digunakan di terminal Linux dengan cara eksekusi script tersebut dengan perintah "*kubectl apply -k ./*" hingga script tersebut berhasil melakukan *deploy* pada cluster dengan *status created*. Kubectl adalah salah satu perintah yang digunakan untuk mendeklarasikan berbagai fungsionalitas pada cluster *kubernetes*. Gambar 3.8 merupakan eksekusi terhadap aplikasi web pada *kubernetes cluster*.

```
hanifluthfi@Amethyst:~
$ kubectl apply -k ./
storageclass.storage.k8s.io/mysqldb-kube-storage created
storageclass.storage.k8s.io/wp-kube-storage created
secret/mysql-pass-7bh672g2d5 created
service/mysqldb-service created
service/wordpress-lb created
persistentvolume/mysql-db-persistent-volume created
persistentvolume/wordpress-persistent-volume created
persistentvolumeclaim/mysqldb-pvc created
persistentvolumeclaim/wp-pvc created
deployment.apps/mysql-databases created
deployment.apps/wordpress-web created
hanifluthfi@Amethyst:~
```

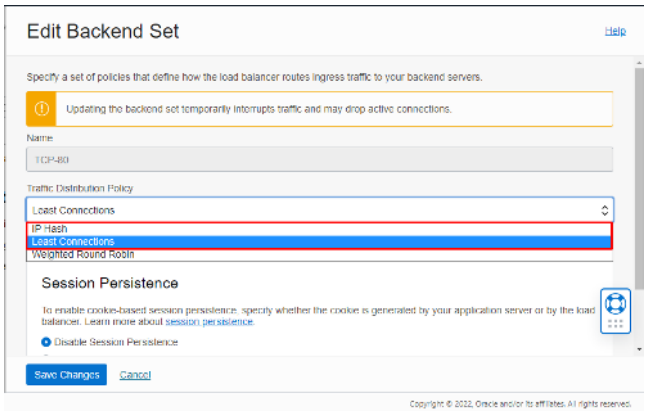
GAMBAR 3. 8
Eksekusi file script YAML

Pada Oracle Cloud, akan muncul service dari Load Balancer beserta dengan IP Public agar dapat diakses secara *external cluster*. Proses deploy load balancer sudah otomatis terpasang pada cluster serta *backend server*. Pastikan status health "OK" dan backend sets health berada pada posisi "OK" yang menandakan backend server dalam kondisi baik. Subnet yang digunakan akan otomatis menggunakan subnet Load Balancer seperti pada Gambar 3.9.



GAMBAR 3. 9
Informasi Load Balancer

Jika melihat lebih detail, Layanan load balancer tersebut memberikan opsi terhadap algoritma load balancing yang sesuai dengan kebutuhan perancangan.



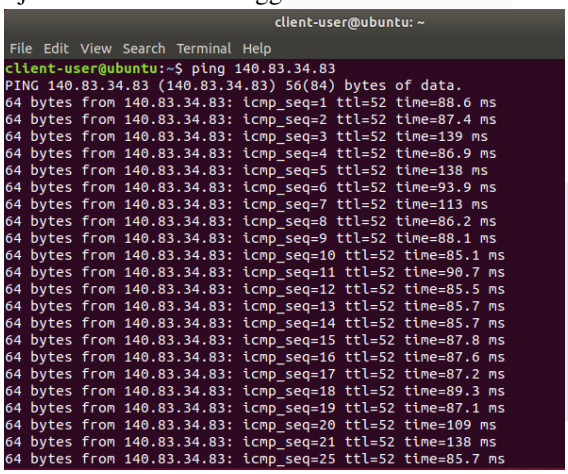
GAMBAR 3. 10
Konfigurasi Algoritma pada Load Balancer

Gambar 3.10 merupakan konfigurasi penggunaan kebijakan distribusi traffic, dimana penggunaan distribusi traffic masuk ke dalam load balancer. Oracle cloud menawarkan 3 algoritma, yaitu Weighted Round Robin, Least Connection, dan IP Hash. Pengujian kali ini akan menggunakan 2 algoritma sebagai acuan perbandingan performa server yang akan diuji menggunakan layanan aplikasi web. Opsi tersebut dapat dilakukan pada bagian backend server, kemudian pilih Traffic Distribution Policy.

IV. PENGUJIAN DAN ANALISIS

A. Pengujian Konektivitas

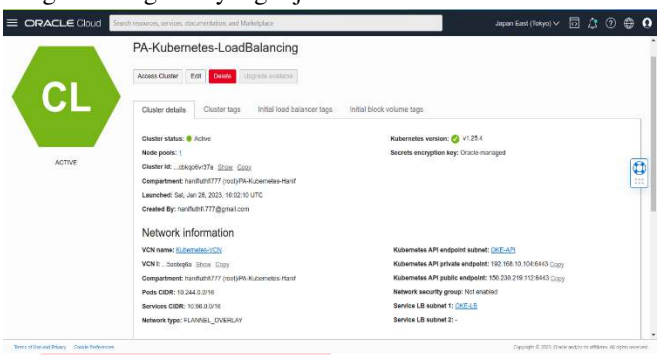
Pengujian konektivitas dilakukan dengan cara untuk melakukan test ping. Pada skenario ini dilakukan ping menuju *Load Balancer* dari Oracle Cloud. Gambar 4.1 merupakan hasil dari pengujian konektivitas dari client menuju Load Balancer menggunakan *IP Public*.



GAMBAR 4.1
Pengujian Konektivitas Load Balancer

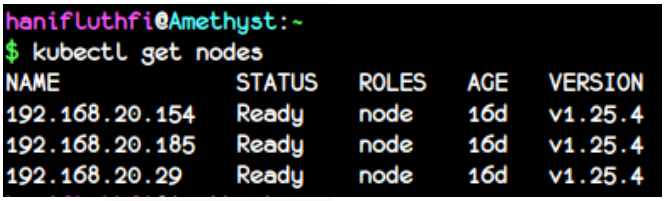
B. Pengujian Fungsionalitas Sistem

Pengujian fungsionalitas sistem dilakukan untuk melakukan identifikasi terhadap fungsi dari sistem yang telah diimplementasikan agar dapat berjalan serta berfungsi dengan baik. Bagian ini akan memaparkan beberapa tahapan pengujian fungsionalitas terhadap sistem dari perancangan load balancing yang mencakup berupa pengecekan terhadap *kubernetes cluster* yang telah diimplementasikan pada *Oracle Cloud Infrastructure*. Kemudian dilakukan pemeriksaan terhadap masing-masing *worker node* serta *deployment* dari komponen *pod* aplikasi wordpress berfungsi dengan konfigurasi yang dijalankan.



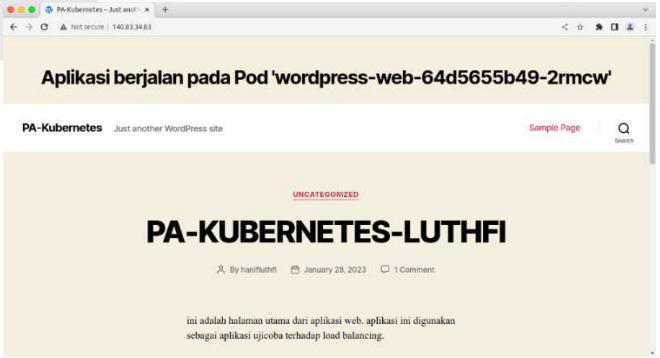
GAMBAR 4.2
Informasi Kubernetes Cluster pada Oracle Cloud

Gambar 4.2 merupakan informasi *kubernetes cluster* yang diimplementasikan pada *Oracle Cloud* secara detail sesuai dengan tahap perancangan sebelumnya. Status yang ditampilkan *kubernetes cluster* sudah aktif sehingga dapat melakukan tahap pemeriksaan terhadap masing-masing *worker node* pada terminal menggunakan perintah "*kubectl get nodes*". Perintah tersebut akan menampilkan berupa list dari penggunaan *worker node* dalam *kubernetes cluster*.



GAMBAR 4.3
Pengujian Fungsionalitas Worker Node

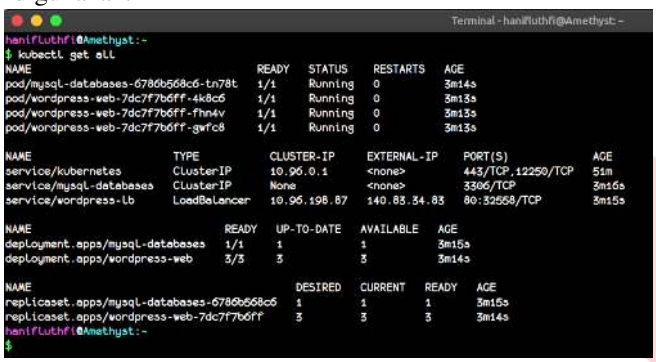
Gambar 4.3 menampilkan daftar worker nodes dari kubernetes cluster yang dilakukan instalasi cluster pada *Oracle Cloud Infrastructure*. Masing-masing *worker node* mendapatkan *IP Address* bersifat *private* sesuai *subnet network* yang dipilih menggunakan subnet *worker node* dengan status "*Ready*" sehingga menandakan node tersebut sudah dapat melakukan proses sesuai dengan tugas serta peran nya untuk mengelola aplikasi yang berjalan didalamnya.



GAMBAR 4.4
Tampilan Aplikasi Web

Gambar 4.4 merupakan tampilan utama dari halaman aplikasi web yang digunakan untuk pengujian sistem *load*

balancing pada kubernetes. Web tersebut dapat diakses menggunakan alamat IP Address publik yang didapatkan pada layanan *load balancer* dari *Oracle Cloud*, kemudian trafik diarahkan ke *worker node* sesuai dengan algoritma yang digunakan. Pada halaman web dapat menampilkan penggunaan dari nama *pod* aplikasi sehingga dapat mengetahui *pod* berapa yang sedang digunakan. Skenario tersebut dilakukan untuk mengetahui permintaan client yang diarahkan load balancer menuju *pod container* yang sedang digunakan.



GAMBAR 4.5
Pengujian Fungsionalitas Pod Kubernetes

Gambar 4.5 merupakan pengujian fungsionalitas terhadap *pod* serta *service* dari kubernetes. menunjukkan beberapa informasi yang didapatkan seperti pod yang digunakan yaitu Wordpress dan *MySQL*. Service untuk Wordpress dijadikan sebagai *Load Balancer* agar aplikasi wordpress dapat diakses menggunakan *IP Public* dengan konfigurasi port 80 yang merupakan port default dari *HTTP* dan kemudian dilakukan port mapping menuju ke port 32558, dimana port tersebut sebagai komunikasi antara Load Balancer dengan worker node. Service untuk *MySQL* dikategorikan sebagai ClusterIP dan digunakan untuk komunikasi antara pod Wordpress sebagai *frontend* aplikasi dan pod *MySQL* sebagai *backend* database. Status dari *pod* Wordpress serta *MySQL* dapat dikatakan berjalan dengan baik pada masing-masing worker node.

1. Pengujian Least Connection

Dengan metode least connection, load balancer akan me lakukan perbandingan jumlah koneksi yang aktif untuk diteruskan menuju server selanjutnya yang akan menerima koneksi tersebut.

GAMBAR 4.6
Metric Inbound Request Least Connection

Gambar 4.6 merupakan merupakan grafik metrik pengujian dari algoritma *least connection* dengan skenario 500 user, 1000 user, 1500 user, 2000 user, dan 2500 user yang didapatkan pada metrik *inbound request* atau permintaan yang diterima oleh *load balancer* dari layanan *Oracle Cloud Infrastructure*. Grafik tersebut menjelaskan bahwa masing-masing dari *backend server* atau *worker node* akan mendapatkan request yang masuk pada *load balancer* secara merata sesuai dengan request dari client.

2. Pengujian IP Hash

Algoritma IP Hash akan melakukan deteksi terhadap IP dari client yang masuk kedalam load balancer. Kemudian, algoritma ini melakukan hashing untuk mendapatkan kunci unik, dimana kunci unik tersebut digunakan untuk melakukan alokasi terhadap server tententu. Kunci unik dapat diperbarui jika session yang telah dibuat telah rusak. Metode IP Hash mampu untuk memastikan bahwa client akan diarahkan ke server yang sama yang digunakan sebelumnya.

GAMBAR 4.7
Metric Inbound Request IP Hash

Gambar 4.7 Berikut merupakan data grafik pengujian dari algoritma IP Hash dengan skenario 500 user, 1000 user, 1500 user, 2000 user, dan 2500 user yang didapatkan pada metrik *inbound request* yang diterima oleh *load balancer* dari layanan *Oracle Cloud Infrastructure*. Grafik tersebut menjelaskan bahwa masing-masing dari *backend server* atau *worker node* akan mendapatkan request yang masuk pada load balancer secara merata sesuai dengan request dari client.

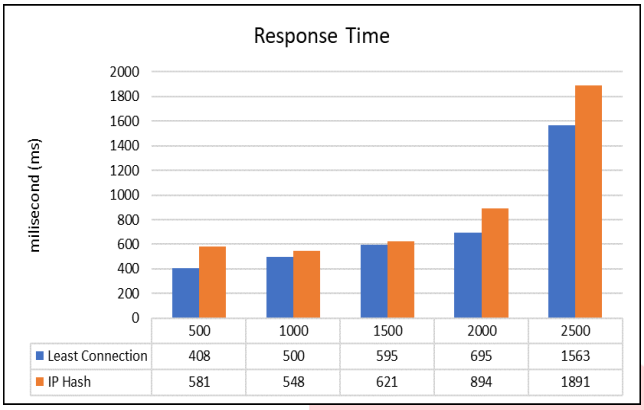
C. Hasil Pengukuran Sistem Perancangan

Pada proses skenario yang dilakukan pada implementasi sistem load balancing dengan algoritma *Least Connection* dan *IP Hash* menggunakan kubernetes cluster. Hasil ujicoba tersebut dilakukan analisis berdasarkan parameter pengukuran *response time*, *throughput*, dan *request loss* pada performa aplikasi web *server* serta pengukuran penggunaan CPU pada masing-masing *server*. Berikut merupakan grafik dari pengukuran sistem load balancing dengan algoritma *Least Connection* dan *IP Hash* pada *Kubernetes.*

1. Response Time

Response Time merupakan metode pengukuran yang digunakan untuk mengukur seberapa lama paket-paket tersebut yang direspon oleh suatu server agar dapat memenuhi request client dalam rentang waktu tertentu. Satuan yang digunakan dalam pengujian kali ini adalah *millisecond*. Jika semakin kecil dari response time tersebut, maka server tersebut dapat dikategorikan baik dalam

menangani request dari *client.* Berikut merupkan grafik perbandingan terhadap pengukuran response time pada *load balancing* dengan algoritma *Least Connection* dan IP Hash pada Kubernetes.

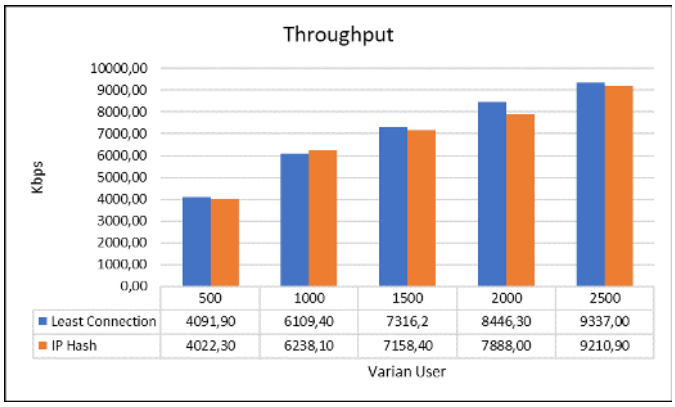


GAMBAR 4.8
Grafik Perbandingan Response Time

Gambar 4.8 merupakan grafik perbandingan terhadap parameter response time dari masing-masing penggunaan algoritma load balancing yang diimpelemtasikan pada kubernetes. Berdasarakan pengujian yang telah dilakukan, didapatkan data *response time* dengan beberapa varian user pada implementasi *load balancing* menggunakan algoritma *Least Connection* dan *IP Hash* pada *Kubernetes*. Setiap varian user memiliki peningkatan dari masing-masing ujicoba skenario yang dilakukan. Saat melakukan pengujian varian user 2500 terjadi selisih peningkatan cukup tinggi, Hal tersebut disebabkan keterbatasan terhadap spesifikasi perangkat yang digunakan pada worker node sehingga worker node menjadi kurang responsif saat melakukan proses permintaan dari banyak client. Pada Grafik menunjukkan bahwa *response time* pada algoritma *Least Connection* lebih rendah serta responsif dalam proses penanganan request jika dibandingkan dengan algoritma *IP Hash* disetiap masing-masing varian user. Hal tersebut terjadi karena algoritma *IP Hash* melakukan proses hashing terhadap paket request terlebih dahulu pada *load balancer* sehingga membutuhkan cukup waktu untuk menuju ke *worker node*. sedangkan algoritma *Least Connection* tidak membutuhan hal tersebut sehingga dapat menyeimbangkan beban setiap *worker node*.

2. Throughput

Throughput dilakukan untuk mengukur kemampuan dari server yang menerima paket-paket request dari client. Paket tersebut akan dihitung oleh server untuk mengetahui seberapa banyak paket yang diterima. semakin besar nilai dari troughput yang didapatkan, maka semakin baik kinerja yang dihasilkan oleh server. Berikut merupkan grafik perbandingan dari pengukuran data throughput algoritma *Least Connection* dan *IP Hash* pada *Kubernetes*.

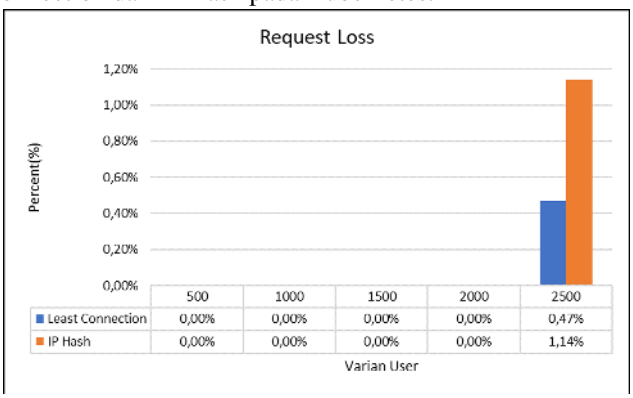


GAMBAR 4. 9
Grafik Perbandingan Throughput

Gambar 4.9 merupakan grafik perbandingan terhadap parameter throughput dari masing-masing penggunaan algoritma load balancing yang diimpelemtasikan pada kubernetes. Berdasarkan pengujian yang telah dilakukan, didapatkan data *throughput* dengan beberapa varian user pada implementasi *load balancing* menggunakan algoritma *Least Connection* dan *IP Hash* pada *Kubernetes*. Grafik menunjukkan bahwa nilai *throughput* mengalami kenaikan karena penambahan jumlah varian user yang otomatis mampu melakukan peningkatan jumlah *request* menuju web *server* dan trafik dapat terdistribusi dengan baik. Pada Grafik menunjukkan nilai *throughput* pada algoritma *Least Connection* dapat memberikan throughput yang lebih baik dibandingkan dengan algoritma *IP Hash* disetiap masing-masing varian user. Hal tersebut disebabkan bahwa *Least Connection* memberikan kapasitas koneksi yang stabil dan maksimal tanpa mengurangi koneksi yang terjadi.

3. Request Loss

Request Loss merupakan parameter yang merepresentasikan kondisi yang dimana menunjukkan jumlah total dari request yang gagal disebabkan *server* tidak dapat melayani permintaan dari client. Dengan *request loss* dapat mengetahui seberapa kemampuan server untuk menangani *request* dari client, jika semakin kecil *request loss* yang didapatkan, maka dinyatakan bahwa algoritma tersebut dapat meminimalisir terhadap terjadinya *request loss*. Berikut merupakan grafik perbandingan request loss pada masing-masing penggunaan load balancing dengan algoritma Least Connection dan IP Hash pada Kubernetes.

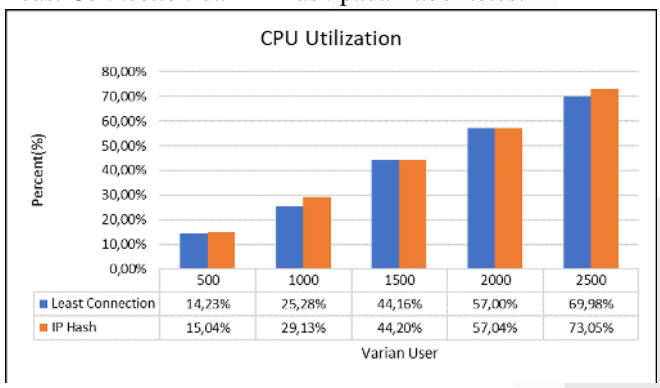


GAMBAR 4.10
Grafik Perbandingan Request Loss

Gambar 4.10 merupakan grafik perbandingan terhadap parameter request loss. Berdasarkan pengujian yang telah dilakukan, didapatkan data *request loss* dengan beberapa varian user pada implementasi *load balancing* menggunakan algoritma *Least Connection* dan *IP Hash* pada *Kubernetes*. Pada pengujian terhadap varian 500 user hingga 2000 user tidak mengalami kendala ketika melakukan proses request dari client menuju *worker node*, hal tersebut disebabkan karena spesifikasi dari worker node serta *pod* yang digunakan mampu melakukan layanan request dengan optimal pada batas permintaan dari user tersebut. Namun berbeda dengan user 2500 memiliki *request loss* yang berbeda pada masing-masing penggunan algoritma sehingga aplikasi dari *pod* tidak mampu memberikan layanan secara maksimal kepada permintaan dari client. *Request loss* yang pada algoritma *Least Connection* yang didapatkan selama pengujian dilakukan adalah sebesar 0,47%, sedangkan algoritma *IP Hash* didapatkan sebesar 1,14% sehingga *Least Connection* mampu mengurangi lebih sedikit jika terjadinya *request loss* dibandingkan dengan *IP Hash*.

4. CPU Utilization

CPU Utilization merupakan parameter proses penggunaan dari kinerja pada suatu server untuk mengetahui persentase beban kerja yang diproses. Semakin tinggi beban kinerja yang diproses oleh CPU, maka server akan memberikan limit process sehingga *server* tidak mengalami *overload*. Berikut merupakan grafik perbandingan CPU pada masing-masing penggunaan load balancing dengan algoritma *Least Connection* dan *IP Hash* pada *Kubernetes*.



GAMBAR 4. 11
Grafik Perbandingan CPU Utilization

Gambar 4.11 merupakan grafik perbandingan terhadap parameter *CPU Utilization* dari masing-masing penggunaan algoritma *load balancing* yang diimpelemtasikan pada kubernetes. Berdasarkan pengujian yang telah dilakukan, didapatkan data persentase CPU yang digunakan oleh *worker node* dengan beberapa varian user pada implementasi *load balancing* menggunakan algoritma *Least Connection* dan *IP Hash* pada *Kubernetes*. Pada Grafik menunjukkan bahwa penggunaan *CPU* dapat meningkat karena penambahan jumlah varian user. Algoritma *IP Hash* memiliki penggunan CPU yang tinggi disebabkan oleh proses yang dilakukan pada saat load balancing, yaitu pembuatan kunci hash yang unik untuk mengarahkan trafik menuju *worker node*. Hal tersebut dapat meningkatkan kinerja *CPU* dari masing-masing *worker node*. Dapat diasumsikan bahwa algoritma *Least Connection* mampu mengurangi beban penggunaan sumber daya komputasi tiap *worker node* sehingga dapat lebih optimal dibandingkan dengan algoritma *IP Hash*.

V. KESIMPULAN

A. Kesimpulan

Berdasarkan hasil perancangan, pengujian dan analisa yang telah dilakukan maka dapat diambil beberapa kesimpulan sebagai berikut:

1. Berdasarkan hasil pengujian dari perancangan sistem load balancing dengan algoritma *Least Connection* dan *IP Hash* yang diimplementasikan pada Kubernetes dalam cloud environment, didapatkan bahwa *Least Connection* mampu memberikan performa yang baik untuk mendistribusikan paket permintaan dari client sesuai dengan pengukuran pada parameter *Response Time*, *Throughput* dan *Request Loss*.
2. Dari hasil implementasi dan pengujian sistem dapat diakses seluruh kalangan melalui jaringan internet sehingga dapat diakses kapanpun dan dimanapun.
3. Berdasarkan analisis yang didapat bahwa nilai perbandingan pada performa *CPU Utilization* sudah sesuai dengan standar kurang dari 80% dari resource yang terpakai pada masing-masing *server*.
4. Nilai *throughput* mengalami kenaikan karena penambahan jumlah varian user yang otomatis mampu melakukan peningkatan jumlah request menuju web *server*.
5. Aplikasi layanan web salah satunya adalah *wordpress* dapat diimplementasikan pada platform *Kubernetes*, yaitu dengan cara membuat berupa script file dan kemudian dilakukan deployment agar aplikasi tersebut dapat berjalan pada masing-masing *worker node* dengan integrasi database menggunakan *MySQL* sebagai *database* pada aplikasi wordpress.

B. Saran

Berdasarkan hasil pembangunan perancangan sistem ini, dapat disampaikan beberapa saran untuk pengembangan selanjutnya yaitu:

1. Menggunakan variasi layanan aplikasi agar dapat melakuakan perbandingan untuk melakukan penelitian *load balancing*.
2. Merancang *Kubernetes Cluster* pada layanan *Cloud Computing* lainnya, seperti AWS, Google Cloud, Microsoft Azure.
3. Melakukan ujicoba dengan varian user lebih banyak dan menggunakan spesifikasi yang mumpuni agar dapat

mendapatkan hasil yang lebih baik.

4. Melakukan kombinasi dengan layanan yang ditawarkan pada Cloud Computing serta fitur kubernetes.

REFERENSI

[1] A. Sumiati, P. H. Trisnawan and M. A. Fauzi, "Implementasi Load Balancing Web Server dengan Algoritma Source IP," *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer,* vol. 4, pp. 919-928, 2020.

[2] A. D. Setiawan, Y. Widhi and M. Data, "Load Balancing Server Web Berdasarkan Jumlah Koneksi Klien Pada Docker Swarm," *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer,* vol. 3, pp. 925-930, 2019.

[3] I. P. A. Suwandika, Analisis Performansi Load Balancing menggunakan Algoritma Least Connection dan IP Hash melalui jaringan SDN pada Web Server, Bandung: Universitas Telkom, 2018.

[4] M. Fihri, "Implementasi & Analisis Performansi Layanan Web Pada Platform Berbasis Docker," 2019.

[5] Kubernetes, "What is Kubernetes?," 2020. [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.. [Accessed February 2022].

[6] N. Nguyen and T. Kim, "Toward Highly Scalable Load Balancing in Kubernetes Clusters," *IEEE Communications Magazine,* vol. 58, pp. 78-83, 2020.

[7] K. Takahashi, "A Study on Portable Load Balancer", The Graduate University for Advanced Studies (SOKENDAI), 2019.

[8] Cloudflare, "What is load balancing?," [Online]. Available: https://www.cloudflare.com/learning/performance/what-is-load-balancing. [Accessed January 2022].

[9] D. M. E. Mustafa, "LOAD BALANCING ALGORITHMS ROUND-ROBIN (RR), LEAST CONNECTION, AND LEAST LOADED EFFICIENCY," *GESJ: Computer Science and Telecommunications,* pp. 25-29, 2017.

[10] G. Singh and K. Kaur, "An Improved Weighted Least Connection Scheduling Algorithm for Load Balancing in Web Cluster Systems," *International Research Journal of Engineering and Technology (IRJET),* vol. Vol.5, pp. 1950-1955, 2018.

[11] J. P. Putra, "Kajian Web Load Balancing Berbasis Round Robin Dan IP Hash," Institut Teknologi Sepuluh Nopember, 2018.

[12] S. Afzal and G. Kavitha, "Load balancing in cloud computing – A hierarchical taxonomical classification," *Journal of Cloud Computing: Advances, Systems and Applications,* 2019.

[13] V. N. Volkova, L. V. Chemenkaya, E. N. Desyatirikova, M. Hajali, A. Khodar and A. Osama, "Load balancing in cloud computing," *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus),* pp. 387-390, 2018.

[14] V. G. d. Silva, M. Kirikova and G. Alksnis, "Containers for Virtualization: An Overview," *Applied Computer Systems,* vol. 23, pp. 21-27, 2018.

[15] M. Klement, "Models of integration of virtualization in education: Virtualization technology and possibilities of its use in education," *Computers & Education,* pp. 31-43, 2017.

[16] Docker, "What is Container," 2018. [Online]. Available: https://www.docker.com/resources/what-container.

[17] Kamarudin, Kusrini and A. Sunyoto, "Uji Kinerja Sistem Web Service Pembayaran Mahasiswa Menggunakan Apache JMeter (Studi Kasus: Universitas AMIKOM Yogyakarta)," 2018.