PRAKATA

Puji dan syukur penulis panjatkan kehadirat Allah SWT yang telah melimpahkan kasih dan sayang-Nya sehingga penulis dapat menyelesaikan skripsi yang berjudul "**Analisis Unjuk Kerja *Load Balancing Web Server* Menggunakan Virtualisasi Berbasis *Container Docker Swarm*** ". Maksud dari penyusunan skripsi ini adalah untuk memenuhi salah satu syarat dalam menempuh ujian sarjana Teknik Telekomunikasi pada Fakultas Teknik Telekomunikasi dan Elektro Institut Teknologi Telkom Purwokerto.

Dalam penyusunan skripsi ini, banyak pihak yang membantu penulis dalam berbagai hal. Penulis sampaikan rasa terima kasih yang sedalam-dalamnya kepada:

1. Orangtua dan kerabat yang telah memberikan dukungan berupa motivasi serta doa kepada penulis dalam penyusunan skripsi ini.
2. Bapak Dr. Arfianto Fahmi, S.T., M.T., IPM, selaku Rektor Institut Teknologi Telkom Purwokerto.
3. Ibu Dr. Anggun Fitrian Isnawati, S.T., M. Eng., selaku Dekan Fakultas Teknik Telekomunikasi dan Elektro.
4. Bapak Prasetyo Yuliantoro, S.T., M.T., selaku ketua Program Studi S1 Teknik Telekomunikasi.
5. Bapak Bongga Arifwidodo, S.ST., M.T.,selaku Dosen Pembimbing I.
6. Bapak Eka Wahyudi, S.T., M.Eng., selaku Dosen Pembimbing II.
7. Seluruh dosen, staf dan karyawan Program studi S1 Teknik Telekomunikasi Institut Teknologi Telkom Purwokerto.
8. Keluarga S1 Teknik Telekomunikasi serta angkatan 2015 khususnya kelas A yang berjuang menempuh pendidikan dan belajar bersama penulis.

Penulis menyadari bahwa dalam penulisan skripsi ini masih jauh dari sempurna, untuk itu semua pendapat dan masukan yang telah diberikan di harapkan dapat memberikan manfaat bagi penulis dan bagi pembaca.

Purwokerto, 11 Agustus 2022

(Arnanda Satria Wibawa)

ABSTRAK

Meningkatnya jumlah trafik data mempengaruhi kinerja *web server* sedangkan penggunaan *server* tunggal masih banyak digunakan. Untuk mengatasi permasalahan tersebut, dibutuhkan teknik *load balancing*. *Load balancing* yang dipakai adalah *zevenet ce*. *Load balancing* memiliki beberapa algoritma diantaranya adalah *round robin* dan *least connection*. Untuk mengetahui algoritma *load balancing* mana yang lebih baik dalam menangani koneksi dari *client* maka dibutuhkan pengukuran *response time, CPU Utilization* dan *Memory usage web server*. *Web server* dibangun menggunakan *docker swarm.* Pengujian dilakukan sebanyak 20 kali percobaan pada setiap parameter kemudian diambil rata-rata untul *response time* dan nilai *CPU Utilization* dan *memory usage* tertinggi selama percobaan. Tiap algoritma diberikan beban sebanyak 500 dengan 100 konkurensi, 2000 dengan 400 konkurensi, dan 5000 koneksi dengan 1000 konkurensi menggunakan *H2load Benchmark*. Hasil penelitian menunjukkan bahwa *round robin* bekerja lebih optimal dan cepat dibandingkan *least connection* karena memiliki nilai *response time* yang lebih kecil, *CPU Utilization* yang lebih besar dan *memory usage* yang lebih kecil dengan nilai *response time* pada 500 koneksi dengan 100 konkurensi sebesar 6,28 *second*, 19,83 *second* pada 2000 koneksi dengan 400 konkurensi dan 30,19 *second* pada 5000 koneksi dengan 1000 konkurensi. Persentase *CPU Utilization web server* pada koneksi 500-5000 *node manager* sebesar 62,56%-78,55% dan *node worker* sebesar 61,99%-62,32%. Nilai *Memory usage web server* pada koneksi 500-5000 sebesar 344,10MB-602,44MB pada *node manager* dan 319,72MB-586,70MB pada *node worker*.

Kata Kunci: *Web Server, Load Balancing, Zevenet, Least Connection, Round Robin, Docker Swarm*

ABSTRACT

The increasing amount of data traffic affects the performance of the web server while the use of a single server is still widely used. To overcome these problems, a load balancing technique is needed. The load balancing used is Zevenet CE. Load balancing has several algorithms including round robin and least connection. To find out which load balancing algorithm is better in handling connections from clients, it takes measurement of response time, CPU Utilization and Memory usage of the web server. The web server is built using docker swarm. The test was carried out 20 times for each parameter, then the average was taken for the response time and the highest CPU Utilization and memory usage values during the experiment. Each algorithm is given a load of 500 with 100 concurrency, 2000 with 400 concurrency, and 5000 connections with 1000 concurrency using H2load Benchmark. The results show that round robin works more optimally and faster than least connection because it has a smaller response time value, larger CPU Utilization and smaller memory usage with a response time value of 500 connections with 100 concurrency of 6.28 seconds, 19.83 second on 2000 connection with 400 concurrency and 30.19 second on 5000 connection with 1000 concurrency. The percentage of CPU Utilization of web servers on connection 500-5000 node manager is 62.56%-78.55% and node worker is 61.99%-62.32%. The memory usage value of the web server on a 500-5000 connection is 344.10MB-602.44MB on the node manager and 319.72MB-586.70MB on the worker node.

Keywords: *Web Server, Load Balancing, Zevenet, Least Connection, Round Robin, Docker Swarm*

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENDAHULUAN

1.1 LATAR BELAKANG

Dewasa ini, manusia semakin bergantung pada teknologi khususnya dalam bidang telekomunikasi dalam usaha mereka memperoleh berbagai macam informasi. Hal ini menyebabkan angka pertukaran data naik setiap harinya sehingga kecepatan dan kehandalan *web server* sangat dibutuhkan. Tetapi, saat ini *single server* masih banyak digunakan sedangkan sistem ini memiliki titik lemah dalam menangani permintaan data yang semakin bertambah banyak [1]. Oleh karena itu dibutuhkan teknik *load balancing* untuk menangani banyaknya permintaan data yang terjadi.

Load balancing menggabungkan beberapa *server* dan bekerja dengan membagi beban trafik secara merata kepada setiap *server* sehingga resiko terjadinya kemacetan *traffic* dan beban *server* yang berlebihan saat menangani *request* bersamaan dapat ditekan. Pada penelitian Faris Muslim Azmi [2] mengimplementasikan *load balancing* dalam sebuah sistem merupakan cara yang tepat dalam usaha membagi beban *server* saat banyaknya *request* yang masuk secara bersamaan. *Zevenet* merupakan salah satu *load balancer* yang bersifat *open source* dan dapat berjalan pada *virtual machine* sehingga biaya dapat ditekan. *Zevenet* memiliki banyak algoritma *load balancing* salah satunya *Least Connection* dan *round robin*. *Least Connection* bekerja dengan cara membagi beban sesuai dengan banyaknya koneksi yang sedang dilayani oleh *server*, beban koneksi yang masuk akan dilayani oleh *server* dengan jumlah koneksi yang lebih sedikit. Sedangkan *round robin* bekerja dengan cara membagi *request* yang masuk ke semua *web server* secara merata tanpa memperhatikan kapasitas *server* ataupun *request* yang diterima oleh *server*.

Docker merupakan salah satu virtualisasi berbasis kontainer. Namun, mengelola banyak kontainer untuk menjalankan sebuah *service* adalah tugas yang tidak mudah. Untuk itu, *docker* memperkenalkan *docker swarm*. *Docker swarm* dapat membantu mengelompokkan beberapa kontainer menjadi satu kelompok sehingga dapat diatur dalam tempat yang sama yaitu *swarm*. *Docker swarm*

mempunyai dua *node*, yaitu *node manager* dan *node worker*. *Node manager* berfungsi untuk mengatur keanggotaan sedangkan *node worker* berfungsi menjalankan *swarm service* di *docker swarm* [1].

Berdasarkan permasalan meningkatnya jumlah pertukaran data dan masih banyaknya penggunaan *single backend server*, maka penulis mengambil judul sebagai tugas akhir yaitu **"ANALISI UNJUK KERJA *LOAD BALANCING WEB SERVER* MENGGUNAKAN VIRTUALISASI BERBASIS *CONTAINER DOCKER SWARM*"**.

1.2 RUMUSAN MASALAH

Rumusan masalah pada penelitian ini adalah bagaimana unjuk kerja *response time*, *CPU utilization*, dan *memory usage* pada *load balancing web server* menggunakan algoritma *least connection* dan *round robin* pada virtualisasi berbasis *container docker swarm*?

1.3 BATASAN MASALAH

Batasan masalah pada penelitian ini adalah:

1. Implementasi *web server* menggunakan *apache*.
2. Implementasi *load balancing* menggunakan *zevenet*.
3. Implementasi menggunakan 2 *node docker swarm*, 1 buah *load balancer*, dan 1 buah *client*.
4. Implementasi *docker* sebagai *light-weight container virtualization*.
5. Penelitian menggunakan jaringan lokal berbasis IPv4.
6. Parameter yang diamati yaitu *memory usage, response time,* dan CPU *utilization*.

1.4 TUJUAN

Tujuan dari penelitian ini adalah untuk menganalisis unjuk kerja parameter *response time, CPU Utilization* dan *memory usage* yang dihasilkan dari pengujian *load balancing web server* menggunakan algoritma *least connection* dan *round robin* untuk mengetahui unjuk kerja *load balancing web server* menggunakan algoritma yang mana yang lebih baik.

1.5 MANFAAT

Penelitian ini diharapkan dapat memberikan gambaran mengenai penerapan sistem *load balancing web server* menggunakan algoritma *least connection* dan *round robin* untuk menekan resiko beban berlebihan dan antrian *traffic* yang masuk ke *server*.

1.6 SISTEMATIKA PENULISAN

Penelitian ini terbagi menjadi beberapa bab. Bab 1 berisi tentang latar belakang, rumusan masalah, manfaat dan tujuan penelitian, batasan masalah dan sistematika penulisan. Bab 2 berisi kajian pustaka serta dasar teori yang menjadi referensi penulis untuk menyusun penelitian ini. Cara penelitian seperti alat yang digunakan, topologi yang digunakan, spesifikasi perangkat yang digunakan, diagram alur penelitian akan dibahas pada bab 3. Bab 4 membahas tentang hasil simulasi dan analisis sistem berdasarkan hasil simulasi. Kesimpulan dan saran pengembangan untuk kedepannya dideskripsikan pada bab 5.

BAB 2
DASAR TEORI

2.1 KAJIAN PUSTAKA

Kajian pustaka yang digunakan pada penelitian ini terdiri dari beberapa penelitian yang sudah dilakukan sebelumnya. Penelitian [1] tahun 2019 mengungkapkan bahwa penggunaan *single backend server* belum mampu menangani permintaan data yang sangat banyak. Untuk itu, penulis menggunakan teknologi virtualisasi bernama *docker swarm* sebagai alat untuk membuat *clustering web server* untuk meningkatkan keandalan *server*. Untuk menyeimbangkan beban internal *server*, digunakan *load balancing* pada *docker swarm* sehingga permintaan dapat didistribusikan dengan baik ke *web server*. Algoritma load balancing yang digunakan yaitu *round robin* dan *least connection*. Sistem akan dibanjiri dengan 1000, 3000, dan 5000 *request* untuk mendapatkan nilai *throughput*. Hasil pengujian menunjukkan *load balancing* yang menerapkan algoritma *least connection* memiliki *throughput* 15 *Mbps* pada 1000 *request*, 17 *Mbps* pada 3000 dan 5000 *request*. Sedangkan algoritma *round robin* memiliki nilai *throughput* 15 *Mbps* pada 1000 *request*, 14 *Mbps* pada 3000 *request*, dan 15 *Mbps* pada 5000 *request*. Hasil menunjukkan bahwa algoritma *least connection* mempunyai kinerja yang lebih baik daripada algoritma *round robin*.

Penelitian [2] tahun 2019 tentang Perbandingan Kinerja *HAproxy* dan *Zevenet* dalam Pengimplementasian *Multi Service Load Balancing*. Dalam penelitian tersebut, penulis mengungkapkan bahwa setiap pengguna mempunyai kebutuhan layanan yang berbeda sehingga dibutuhkan sebuah sistem *load balancing multi-service* untuk menekan kemacetan *traffic* dan mengurangi beban kerja *server*. Aplikasi *load balancer HAproxy* dan *Zevenet* akan dibandingkan kinerjanya dalam pengimplementasian *multi-service load balancing* dengan mengirimkan 1000, 2500, dan 5000 koneksi HTTP dan FTP. Algoritma yang digunakan adalah *round robin* dan *least connection* dan parameter yang diuji adalah *response time* dan *resource utilization*. Dari pengujian tersebut diperoleh kesimpulan bahwa *HAproxy* mengungguli *Zevenet* pada parameter *response time*.

Sedangkan *Zevenet* lebih unggul dalam parameter *resource utilization* untuk layanan HTTP dan FTP.

Penelitian [3] tahun 2019 tentang Analisis Perbandingan Kinerja *HAproxy* dan *Zevenet* sebagai *Load Balancer Server*. Peneliti menemukan bahwa sering terjadi kemacetan *traffic*, *fail*, hingga kerusakan pada *server* yang disebabkan oleh peningkatan kebutuhan layanan dan tujuan yang beragam dari berbagai macam pengguna. Dengan *load balancer* kemacetan *traffic* dari pengguna terhadap *server* dapat ditekan dan membantu pembagian beban kerja *server*. Pengujian kinerja *HAproxy* dan *Zevenet* dilakukan dengan menggunakan algoritma *round robin* dan dengan cara mengirimkan 3000 koneksi ke *server* sehingga didapatkan nilai pada parameter uji rata – rata *request* dan tingkat *errors*. Hasil pengujian menunjukkan bahwa *Zevenet* memiliki nilai yang lebih baik daripada *HAproxy* dalam parameter rata – rata *request*. Sedangkan *HAproxy* lebih stabil karena tidak memiliki nilai *errors*. Rangkuman keterkaitan dengan penelitian sebelumnya dapat dilihat pada tabel 2.1

Tabel 2.1 Rangkuman Keterkaitan Dengan Penelitian Sebelumnya

Penelitian Oleh	**Parameter Penelitian**			
	Load Balancing	***Response Time***	**Docker Swarm**	**Web Server**
Dimas Setiawan Afis, Mahendra Data, Widhi Yahya	✓		✓	✓
Faris Muslim Azmi, Mahendra Data, Heru Nurwasito	✓	✓		✓
Jajang Khairil Azhar, Lutfhi Nurhakim	✓	✓		✓
Arnanda Satria Wibawa	✓	✓	✓	✓

2.2 DASAR TEORI

2.2.1 VIRTUALISASI

Virtualisasi adalah teknik yang memungkinkan pengguna untuk membuat versi virtual perangkat atau sumber daya seperti sistem operasi, jaringan, server dan perangkat penyimpanan. Virtualisasi memungkinkan sebuah komputer menjalankan beberapa sistem operasi secara serentak. Virtualisasi membagi sumber daya komputer utama menjadi beberapa sumber daya virtual dimana sistem operasi, *libraries*, dan program lainnya bersifat unik dan tidak terhubung ke sistem operasi komputer *host* dibawahnya [1].

Virtualisasi mengacu pada mesin virtual yang berperilaku seperti komputer nyata dengan sistem operasi. Perangkat lunak yang berjalan di mesin virtual yang terpisah dari sumber daya perangkat keras yang mendasarinya. Perangkat lunak yang mendasari mesin virtual pada perangkat keras *host* disebut *hypervisor* atau monitor mesin virtual. *Hypervisor* dalam dunia komputasi adalah *platform* dasar virtualisasi yang memungkinkan beberapa sistem operasi berjalan bersamaan dalam satu komputer. Ada dua jenis *hypervisor* yaitu:

1. Tipe 1

Hypervisor tipe 1 mengeksekusi perintah-perintah virtualisasi pada perangkat keras komputer yang bersangkutan secara langsung. Sistem operasi tamu (*guest OS*) dijalankan pada lapisan diatas lapisan perangkat keras. *XEN, TRANGO*, dan *Oracle VM* adalah contoh *hypervisor* tipe 1.

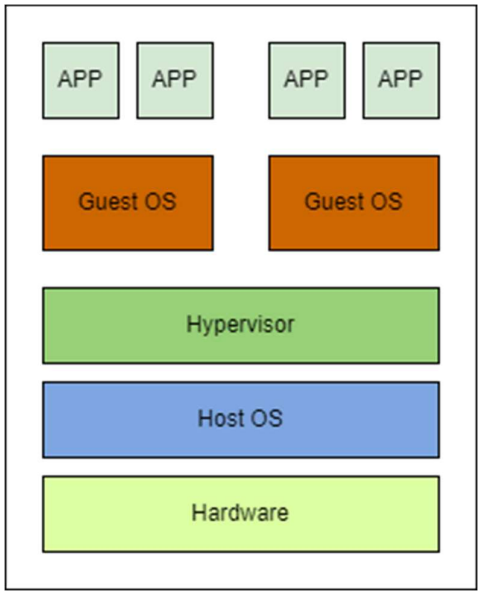
2. Tipe 2

Hypervisor tipe 2 merupakan sebuah aplikasi yang berjalan pada lapisan ketiga layaknya *guest os* pada *hypervisor* tipe 1. Ada tiga jenis virtualisasi yang termasuk pada tipe ini sebagai berikut:

a. Virtualisasi penuh (*Full Virtualization*)

Virtualisasi penuh menggunakan sebuah mesin virtual yang menjadi penyambung antara *guest OS* (sistem operasi yang dibuat oleh mesin virtual) dengan perangkat keras pada sebuah komputer. Instruksi – instruksi yang diproteksi dari sistem operasi harus dikerjakan didalam mesin virtual karena perangkat keras yang digunakan bukan menjadi wewenang sistem operasi tetapi dibagi-bagi antara sistem operasi dan mesin virtual. Dengan kata lain, instruksi yang akan dijalankan

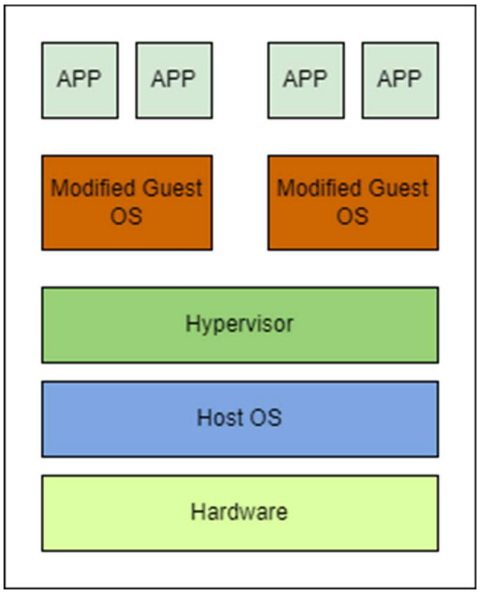
guest OS tidak di eksekusi langsung oleh perangkat keras tetapi disimulasikan ulang oleh mesin virtual yang menyebabkan unjuk kerjanya menurun. Keunggulan jenis virtualisasi ini adalah *guest OS* tidak memerlukan modifikasi. Ilustrasi virtualisasi penuh terdapat pada gambar 2.1.



Gambar 2.1 Arsitekutr Virtualisasi Penuh

b. *Paravirtualization*

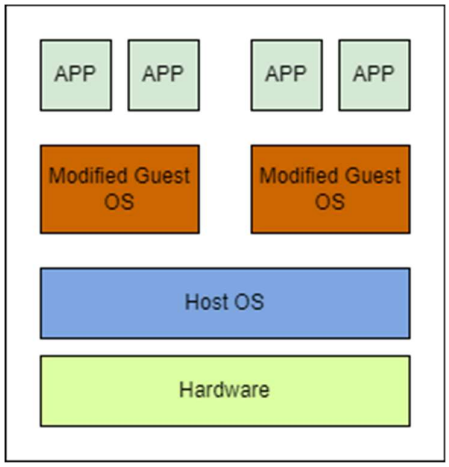
Paravirtualization menggunakan *hypervisor* untuk berbagi akses perangkat keras antara *guest OS* dengan *host OS* dan mengintegrasikan kode-kode virtualisasi kedalam sistem operasi itu sendiri (ke dalam *kernel OS*). Dengan demikian, *guest OS* tidak perlu melakukan simulasi instruksi di dalam mesin virtual karena sudah diatasi dengan kode-kode virtualisasi yang diintegrasikan dengan *kernel* sistem operasi. hal ini yang menjadikan *paravirtualization* memiliki unjuk kerja yang lebih unggul daripada virtualisasi penuh, bahkan mendekati unjuk kerja ketika memakai perangkat keras asli. Contoh aplikasi *paravirtualization* adalah *XEN, Virmware ESXi Server*, dll. Ilustrasi *paravirtualization* terdapat pada gambar 2.2.



Gambar 2.2 Arsitektur Paravirtualisasi

c. Virtualisasi Sistem Operasi (*Operating System Level Virtualization*)

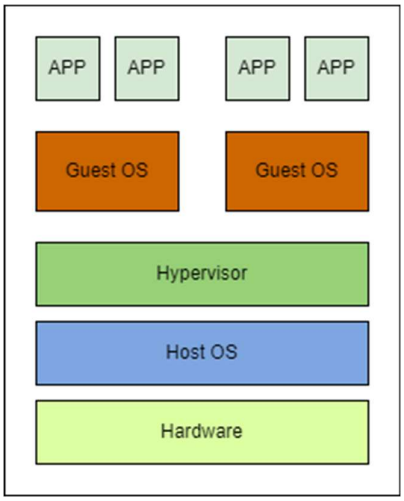
Virtualisasi sistem operasi memungkinkan server terisolasi dan aman untuk menjalankan beberapa mesin virtual pada server fisik tunggal. *Guest OS* yang diimplementasikan padavirtualisasi ini berbagi pakai *kernel OS* yang sama dengan *host OS*. Aplikasi yang berjalan didalam *guest OS* melihatnya sebagai sistem yang berdiri sendiri. Virtualisasi sistem operasi dirancang untuk memberikan isolasi dan keamanan yang diperlukan untuk menjalankan beberapa aplikasi atau salinan dari sistem operasi yang sama dengan distribusi yang berbeda pada satu server. Contoh aplikasi virtalisasi sistem operasi adalah *OpenVZ, Virtuozzo*, *Linux-VServer*, *Solaris Zones*, dan *Free BSD Jails* [4]. Ilustrasi virtualisasi sistem operasi terdapat pada gambar 2.3.



Gambar 2. 3 Arsitektur Virtualisasi Sistem Operasi

2.2.2 *VIRTUAL MACHINE*

Virtual machine merupakan virtualisasi tradisional dimana sebuah mesin terdiri dari komputer, OS standar dan berjalan diatas *layer hypervisor*. *Virtual machine* adalah komponen penuh dari sistem operasi yang disebut *guest* OS yang diinstall didalam sistem operasi lainnya yang disebut *host OS*. Dengan *virtual machine*, *user* dapat melakukan instalasi beberapa sistem operasi di dalam satu mesin secara independen [5]. Arsitektur *virtual machine* terdapat pada gambar 2.4.

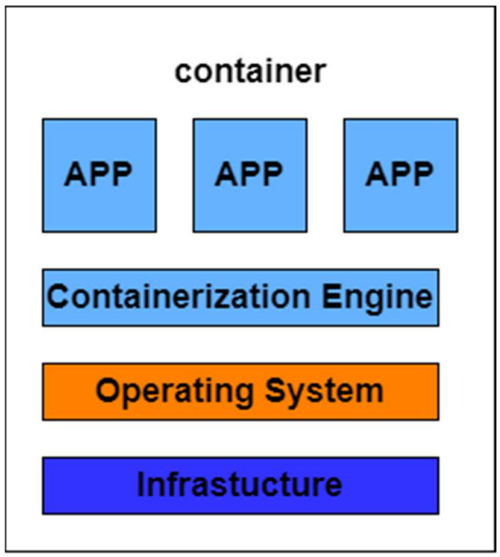


Gambar 2.4 Arsitektur Virtual Machine [5].

2.2.3 KONTAINERISASI

Virtualisasi berbasis *container* adalah cara terbaik dan mudah untuk membat *virtual user space*, ini dilakukan dengan memisahkan *kernel* dari sistem

operasi utama (*host*). *User space* sistem operasi utama atau *host* dibagi menjadi *isolated user space* yang disebut *container*. Model *container* ini memiliki keunggulan *overhead* yang lebih rendah dibandingkan dengan metode berbasis *hypervisor* karena tidak perlu menjalankan sistem operasi tamu (*guest OS*) dan mesin virtual untuk setiap *virtualized environment*. Selain itu virtualisasi berbasis *container* dapat memberikan lebih banyak *user space*. Virtualisasi berbasis *container* digunakan sebagai *container* tempat menjalankan sistem operasi, *web server*, dan alat *benchmarking* [6]. Arsitektur kontainerisasi terdapat pada gambar 2.5.



Gambar 2.5 Arsitektur Kontainerisasi

2.2.4 DOCKER

Docker merupakan salah satu *opensource platform* yang memanfaatkan teknologi virtualisasi berbasis *container*. Dengan docker, *developer* dapat dengan mudah untuk melakukan *deployment*, *build* dan manajemen *container* [7]. *Docker* mempunyai beberapa komponen sebagai berikut:

1. *Docker File*

Docker File merupakan dokumen berisi perintah untuk membangun *docker image*. Perintah dalam *docker file* berisi baris perintah *command-line interface* (*CLI*) yang akan dieksekusi secara berurutan oleh *docker engine* untuk membangun sebuah *image*.

2. *Docker Images*

Docker images berisi *source code* aplikasi yang dapat dieksekusi seperti *tools*, *libraries*, dan *dependency* yang dibutuhkan untuk dijalankan sebagai *container*. *Docker images* dapat dibangun, tetapi kebanyakan *developer* lebih memilih untuk mengunduhnya dari *repository* yang sudah ada. Beberapa contoh *images* seperti *image ubuntu*, *CentOS* dan sebagainya dapat diunduh dan dimodifikasi menjadi *image* baru dengan beberapa penambahan. *image ubuntu* yang sudah terinstall *Java* dan MySQL adalah contoh modifikasi dari sebuah *image*.

3. *Docker Container*

Docker container merupakan *running instance* dari *docker images*. *Docker images* hanya bersifat *read-only* sedangkan *container* dapat dieksekusi seperti merubah pengaturan dan kondisi menggunakan *docker command*. *Docker container* dapat diibaratkan sebagai *virtual machine* atau *guest OS* dalam virtualisasi pada umumnya.

4. *Docker Hub*

Docker Hub merupakan *repository* yang berisi *docker images*. Semua *user docker hub* dapat berbagi *image* sesuka hati. Mereka dapat mengunduh *base images* dari *docker filesystem* sebagai *starting point* untuk proyek kontainerisasi.

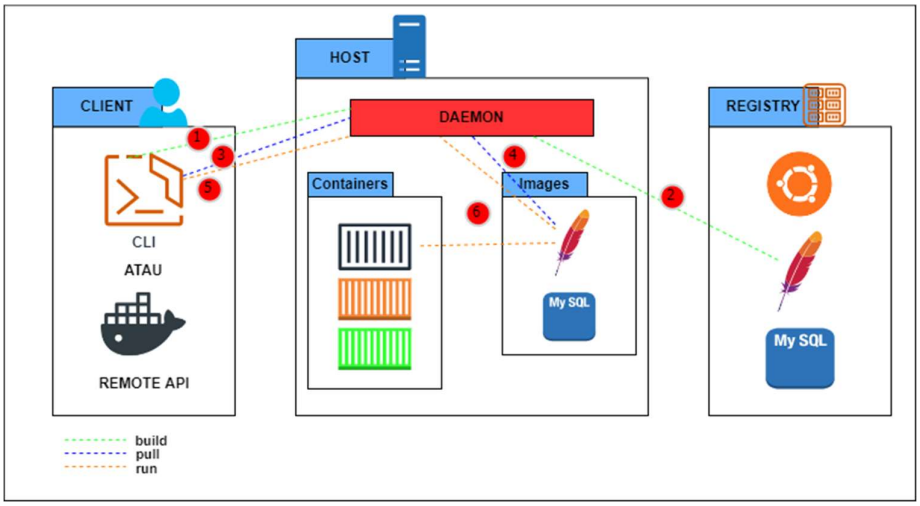
5. *Docker Daemon*

Docker Daemon merupakan *service* yang berjalan pada *OS* seperti *Microsoft Windows*, *MacOS* atau *Ubuntu*. *Service* ini membuat dan memanajemen *docker images* menggunakan *command* dari *client*.

6. *Docker Registry*

Docker registry merupakan penyimpanan *open-source* dan *distribution system* untuk *docker images*. *Registry* memungkinkan untuk mencari versi *image* di *repository* menggunakan *tagging* untuk identifikasi [7].

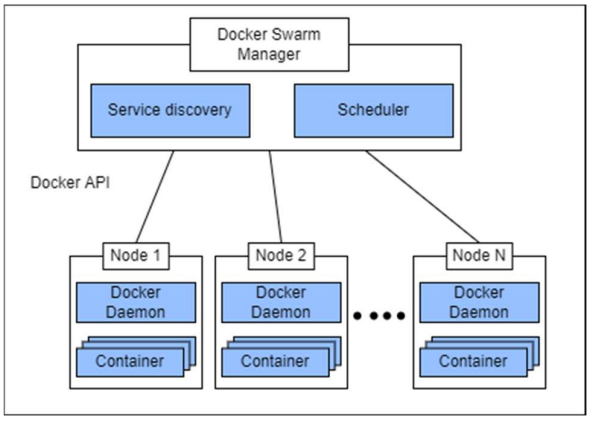
Arsitektur *docker* dapat dilihat pada gambar 2.6.



Gambar 2.6 Arsitektur Docker

2.2.5 DOCKER SWARM

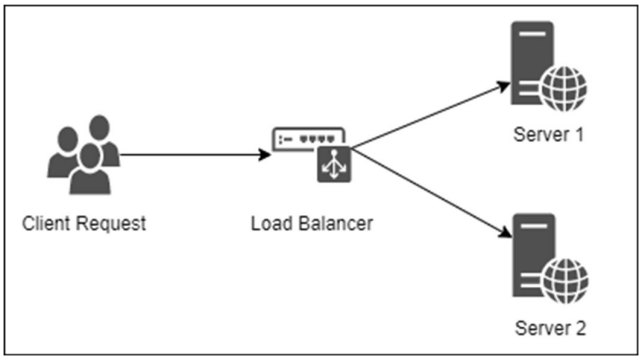
Swarm merupakan sekumpulan *docker host* atau *node* dimana saling terhubung melalui *overlay network*. Dengan demikian, pengguna dapat melakukan *deploy container* pada *multiple host* atau *node*. Dalam *docker swarm* terdapat *node manager* dan *node worker*. *Node manager* memiliki tugas untuk melakukan dan membuat *maintenance cluster service* sedangkan *node worker* berfungsi untuk menjalankan *container* atau layanan yang telah didefinisikan pada *node manager*. Kelebihan *docker swarm* adalah jika salah satu *node down* maka *layanan* akan dilakukan oleh *node* yang sedang aktif [8]. Arsitektur *docker swarm* terdapat pada gambar 2.7.



Gambar 2.7 Arsitektur *Docker Swarm* [9]

2.2.6 LOAD BALANCING

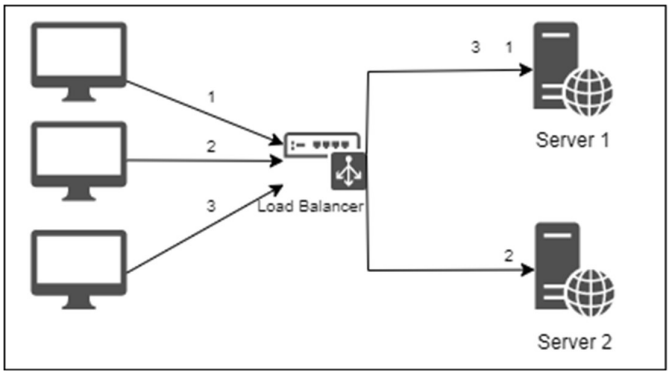
Load balancing merupakan teknik yang bekerja dengan cara mendistribusikan atau membagi beban kerja, jaringan, atau lalu lintas data ke beberapa *server*. Hal ini akan menjadikan *server* lebih andal dan kinerja meningkat karena titik kegagalan dapat ditekan [1]. Ada beberapa algoritma *load balancing* diantaranya adalah algoritma *round robin* dan algoritma *least connection*. Gambaran sistem *load balancing* terdapat pada gambar 2.8



Gambar 2.8 Sistem *Load Balancing* [10]

1. Algoritma Round Robin

Algoritma *round robin* bekerja dengan cara membagi beban ke seluruh *server* yang ada secara urut[10]. Misalnya, jika terdapat dua *server* di dalam *cluster* maka *request* pertama akan diteruskan menuju *server* 1, *request* kedua akan diteruskan menuju *server* 2, *request* ketiga akan diteruskan menuju *server* 1 lagi, dan seterusnya akan seperti itu siklusnya. Semua *server* diperlakukan setara tanpa memperhatikan jumlah koneksi yang masuk atau *response time* yang dialami setiap *server* [11]. Proses pembagian *request* algoritma *round robin* terdapat pada gambar 2.9.



Gambar 2.9 Proses Pembagian *Request Algoritma Round Robin* [10]

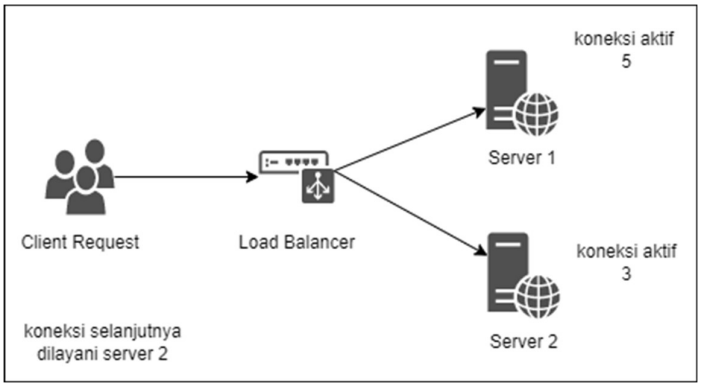
Perbandingan keunggulan dan kekurangan algoritma *round robin* dapat diketahui pada tabel 2.2.

Tabel 2.2 Perbandingan Keunggulan dan Kekurangan Algforitma Round Robin

Algoritma *Round Robin*	
Keunggulan	Kekurangan
Response time lebih lancar dan cepat untuk proses-proses kecil.	Waktu tunggu biasanya bisa sangat lama untuk proses yang lebih besar

2. Algoritma *Least Connection*

Algoritma *least connection* bekerja dengan membagi beban berdasarkan jumlah koneksi yang sedang dilayani *server*. *Server* dengan koneksi terendah akan melayani *request* yang masuk [12]. Algoritma ini termasuk salah satu algoritma yang dinamis karena perlu menghitung koneksi langsung untuk masing-masing server secara dinamis. Untuk *server* virtual, algoritma *least connection* baik untuk memperlancar distribusi beban ke *server* saat beban permintaan sangat bervariasi. Algoritma *leat connection* mengasumsikan bahwa kemampuan pemrosesan semua *server* setara dan memberikan permintaan masuk kepada *server* dengan koneksi paling sedikit. Namun, kinerja sistem tidak ideal saat kemampuuan pemrosesan *server* berbeda [11]. Proses pembagian *request* algoritma *least connection* terdapat pada gambar 2.10.



Gambar 2.10 Proses Pembagian *Request Algoritma Least Connection*

Perbandingan keunggulan dan kekurangan algoritma *least connection* dapat diketahui pada tabel 2.3.

Algoritma *Least Connection*	
Keunggulan	Kekurangan
Keseimbangan pembagian kerja didasarkan pada banyaknya koneksi aktif pada masing-masing server. Sehingga, lamanya koneksi aktif diperhitungkan.	Tidak mempertimbangkan kemampuan pemrosesan masing-masing server.

2.2.7 ZEVENET

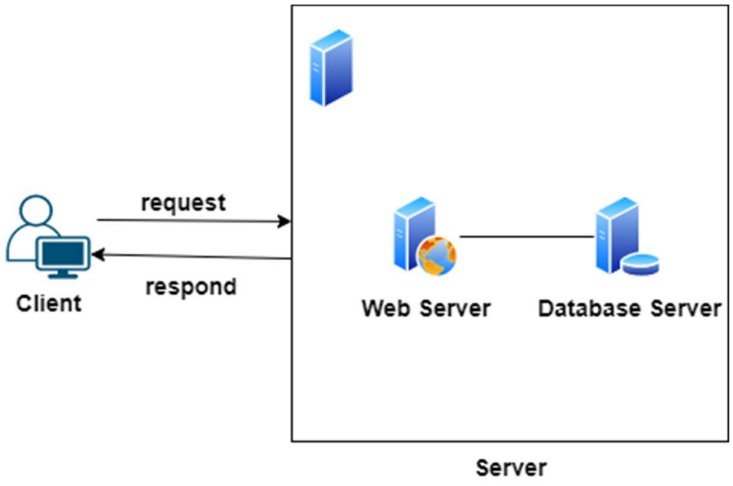
Zevenet merupakan perangkat lunak *load balancer opensource* yang berbasis HTTP dan L4xNAT. Dengan L4xNAT, memungkinkan untuk membuat profil L4 dengan kivvbbn nerja sangat tinggi dan *traffic* yang padat daripada *load balancer* di *layer 7* seperti TCP, UDP atau HTTP profil [13].

2.2.8 SERVER

Server adalah seperangkat sistem komputer yang meneyediakan berbagai jenis layanan (*service*) tertentu dalam sebuah jaringan komputer. Sebuah komputer *server* harus memiliki spesifikasi tinggi karena *server* membutuhkan banyak sumber daya dan agar bekerja secara optimal. Beberapa fungsi server secara umum adalah melayani dan bertanggung jawab penuh terhadap permintaan data dari komputer klien, bertanggung jawab dalam mengatur lalu lintas data, menyimpan berbagai file dan data untuk dapat diakses oleh komputer klien secara bersamaan, dll. Beberapa jenis *server* antara lain *web server* dan *database server* [14].

2.2.9 Web *Server*

Web server merupakan perangkat lunak yang melayani *request* HTTP dari *web browser user* dan mengirimkan kode-kode dinamis ke *server* aplikasi. *Server* aplikasi ini bertugas untuk menerjemahkan dan memproses kode-kode dinamis menjadi kode-kode statis HTML yang selanjutnya dikirimkan ke *browser* oleh *web server*. *Web server* menggunakan protokol HTTP [12].



Gambar 2.11 Ilustrasi Web Server dan Database Server yang berjalan pada Server tunggal

2.2.10 WORDPRESS

Wordpress merupakan salah satu aplikasi berbasis *web* yang popular digunakan saat ini. *Wordpress* memiliki *user interface* berupa *GUI (Graphical User Interface*) sehingga mudah untuk dikonfigurasikan [15].

2.2.11 H2LOAD BENCHMARK

H2load benchmark adalah salah satu alat pengujian *web server* baik untuk HTTP/2 maupun HTTP/1 dengan dukungan SSL atau TLS. H2load menggunakan *io non-blocking* dalam membuat koneksi bersamaan untuk menargetkan titik akhir GET/POST HTTP, sehingga tidak menyebabkan beban menjadi tinggi pada sistem yang berjalan karena *thread* tunggal dapat membuat ribuan koneksi dalam satu detik [16].

2.2.12 *CPU UTILIZATION*

CPU Utilization merupakan persentase banyaknya penggunaan sumber daya *CPU* yang digunakan oleh *server* ketika menangani *request* yang masuk. Peningkatan jumlah *request* mengakibatkan nilai *CPU utilization* ikut meningkat [17]. Lonjakan singkat dalam penggunaan CPU menunjukkan penggunaan sumber daya mesin virtual digunakan dengan sebaik-baiknya. Namun, jika penggunaan CPU untuk mesin virtual diatas 90% dan CPU *idle* diatas 20% akan memengaruhi kinerja mesin virtual tersebut [18].

2.2.13 *MEMORY* USAGE

Memory usage adalah banyaknya RAM yang digunakan oleh *server* ketika memproses *request* yang masuk [17].

2.2.14 RESPONSE TIME

Response time merupakan waktu yang dibutuhkan oleh *server* dalam menangani *request* dari *client*. Nama lain *response time* adalah *latency* yang merupakan total waktu tunggu dan waktu menjawab *request* [17].

Tujuan utama *load balancing* adalah untuk menyediakan *response time* terbaik kepada *client* dengan cara menyebarkan beban ke *server* dalam *cluster*. Jika *response time server* tinggi, akan ada *request* yang menunggu untuk ditangani *server*. Jika *response time server* rendah, *request* yang mennggu untuk ditangani juga rendah karena *server* melayani *request* dengan *rate* tinggi. Jika tiap *server* mempunyai *response time* yang berbeda-beda, *request* akan dikirimkan ke *server* dengan beban yang lebih sedikit [19].