JURNAL MEDIA INFORMATIKA BUDIDARMA
Volume 4, Nomor 3, Juli 2020, Page 529-534
ISSN 2614-5278 (media cetak), ISSN 2548-8368 (media online)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/mib
DOI 10.30865/mib.v4i3.2147

Perancangan Multi Node Web Server Menggunakan Docker Swarm dengan Metode Highavability

Ahmad Rivaldi*, Ucuk Darusalam, Deny Hidayatullah

Fakultas Teknologi Komunikasi dan Informatika, Teknik Informatika, Universitas Nasional, Jakarta, Indonesia
Email: [1,*]ahmadrivaldi21@gmail.com, [2]ucuk.darusalam@gmail.com, [3]deny@civitas.unas.ac.id
Email Penulis Korespondensi: ahmadrivaldi21@gmail.com

Abstrak−Virtualisasi berbasis container sangat popular dikalangan programming development di karenakan virtualisasi yang ringan dimana kernel Linux dapat membagi penggunakan resource antar container bertujuan agar kinerja tidak saling terganggu antar lainnya serta sebagai pembagi beban dalam menanggulangi banyaknya bandwith yang masuk. Salah satu virtualisasi berbasis container yang sering digunakan adalah Docker. Docker sendiri merupakan open source software yang dapat di ubah sesuai dengan keinginan. Container Docker dapat digunakan untuk clustering web server. Hal ini bertujuan untuk mengurangi ”a single point of failure” (SPOF) dalam web server. Bagaimanapun mengatur container dalam jumlah banyak sangatlah rumit, tetapi Docker memiliki engine untuk mengaturnya yang disebut Docker Swarm. Dengan di bantu management NGINX sehingga dapat mengakibatkan resource antar host tidak terbagi secara rata. Oleh karena itu penelitian ini bertujuan untuk mendistribusikan traffic web server secara rata antar host dengan loadbalancing berdasarkan sumber daya monitoring dan failover berdasarkan waktu. Dengan memenfaatkan rendahnya resource atau sumber daya yang di gunakan oleh docker dalam pengoperasian sebuat aplikasi yang dapat di virtualisasikan hanya apa yang di butuhkan.

Kata Kunci: Docker, load balancer, swarm, web cluster

Abstract−Container-based virtualization is very popular among programming development in a lightweight virtualization because where the Linux kernel can divide resource-using containers to prevent uninterrupted performance between And as a burden divider in tackling the many incoming bandwidth. One of the most commonly used container-based virtualization is Docker. Docker itself is an open source software that can be changed to your liking. Docker containers can be used for clustering Web servers. It aims to reduce "a single point of failure" (SPOF) in a Web server. However, arranging a lot of containers is very complicated, but Docker has an engine to set it up called Docker Swarm. With it in the NGINX management help so that it can cause the resource between hosts is not divided on average. Therefore the research aims to distribute Web server traffic across the host with loadbalancing based on time-based monitoring and failover resources. By winning the low resources or resources used by Docker in the operation of an application that can be virtualized only what is needed.

Keywords: Docker, Load Balancers, Swarm, Web Clusters.

1. PENDAHULUAN

Seiring dengan perkembangan teknologi komputer yang sangat pesat salah satunya pada bidang jaringan *virtualisasi web server*, *system administrator* ditantang untuk membuat sebuah sistem *virtualisasi* yang mengkombinasikan antara fleksibilitas dan penggunaan sumber daya yang minimum untuk proses kemudahan *deployment* (penyebaran) [1]. Teknik *virtualisasi* adalah teknik dimana mengisolasi suatu sistem sehingga tidak mengganggu sistem yang lainnya. Salah satu teknik *virtualisasi* adalah *virtualisasi* berbasis *container*. *Virtualisasi* berbasis *container* adalah teknik yang tidak menerapkan *Hypervisior* yang hanya mengisolasi proses tanpa mengisolasi perangkat keras, *kernel* dan *operating system* sehingga dapat mengurangi *overhead* pada *hardware* juga performanya lebih baik darii *virtulasi* mesin [2]. Teknologi *virtualisasi* memiliki beberapa perangkat lunak yang sering digunakan seperti *proxmox, vmware, xen, virtualbox,* dan *hyper-v*, perangkat lunak tersebut adalah jenis *hypervisor*. *Hypervisor* adalah sebuah teknik *virtualisasi* yang memungkinkan beberapa *operating system* untuk berjalan bersamaan pada sebuah *host, hypervisor* memiliki dua kategori native *hypervisor* dan *hosted hypervisor*, *native hypervisor* merupakan *virtual machines* yang berjalan atau diinstal langsung pada *hardware* atau PC seperti *proxmox, vmware ESX*, dan *hyper-v*, sedangkan *hosted hypervisor* adalah *virtual machines* yang diinstal diatas *operating system* atau OS yang kemudian *guest OS* berjalan pada proses aplikasi *virtual machines* tersebut seperti *virtual box, vmware player, parallels desktop* dan *qemu* [3]. Salah satu *virtualisasi* yang berbasis *container* adalah *Docker*.

Docker adalah *open platform* yang memungkinkan sebuah *service* dalam *environment* terisolasi, yang bisa disebut dengan *container* [4]. Sebuah *container* dapat mengemas *library* maupun *dependency* yang dibutuhkan oleh suatu layanan (aplikasi), sehingga *share kernel* memungkinkan berbagi *library* dan *dependency* yang dibutuhkan, dan sebuah layanan (aplikasi) dapat dijalankan tanpa harus menghidupkan *guest OS* terlebih dahulu, proses tersebut dihilangkan, sehingga kebutuhan *resource* (CPU, RAM) berkurang dan waktu yang dibutuhkan untuk menjalankan layanan semakin cepat [2,5]. Salah satu teknik untuk mengatasi masalah tersebut adalah menggunakan teknik *web cluster*. Teknik ini adalah menggunakan beberapa mesin dapat berupa *virtualisasi host* atau *host* asli lebih dari satu dalam satu jaringan yang saling terhubung. Jika salah satu *host* mati maka terdapat *host* lainnya untuk menangani *request* dari *client* sehingga tercapai fungsi ketersediaan (*availability*) [8]. Bagaimanapun mengatur banyak *host* sangatlah kompleks, tetapi *Docker* memiliki modul untuk menangani masalah tersebut yang disebut *Docker Swarm*.

Submitted: **1/05/2020**; Accepted: **09/05/2020**; Published: **20/07/2020**

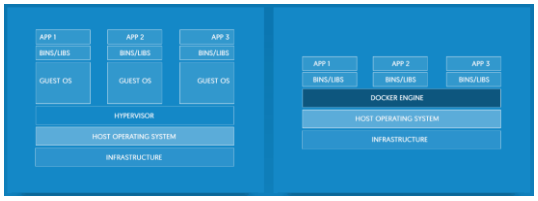
JURNAL MEDIA INFORMATIKA BUDIDARMA
Volume 4, Nomor 3, Juli 2020, Page 529-534
ISSN 2614-5278 (media cetak), ISSN 2548-8368 (media online)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/mib
DOI 10.30865/mib.v4i3.2147

Load balancer terbagi menjadi 2 yaitu *loadbalancer software* dan *hardware.* Terdapat banyak *software load balancing* diantaranya yaitu HAproxy, Nginx, dan Zevenet. Pada penelitian sebelumnya yang berjudul *"Load Balancing* Server *Web* Berdasarkan Jumlah Koneksi Klien Pada Docker Swarm" bahwa load balancing pada *Nginx* dapat diterapkan pada *Docker Swarm* (Dimas Setiawan Afis, 2018). Kelebihan menggunakan *Nginx* dalam *Docker* adalah terdapat modul *load balancer* yang tersedia langsung di dalamnya sehingga mengurangi penggunaan *resource* pada *host* dan ketergantungan modul lainnya. *Nginx* terdapat pada repository *Docker* yang telah di *package* menjadi *container* sehingga mudah untuk di implementasikan[9].

Dari referensi diatas dapat disimpulkan bahwa penggunaan *load balancing* dapat mengoptimalkan suatu *web server* dengan membagi beban *traffic* dengan berbagai algoritma dan metode yang berbeda. Oleh karena itu, peneliti menggunakan metode *load balancing* berbasis sumber daya metode ini dipilih karena dengan mengetahui penggunaan *memory web server* kita bisa mengetahui *Node worker* mana yang bebannya sedikit memproses suatu *request* sehingga beban dapat didistribusikan dengan baik dan dapat mendeteksi *host* yang *down* ketika bekerja. Serta kemudahan dan fleksibilitas yang di tawarkan juga sangat menguntunkan para *developer* dan *user* sebagai pengguna dengan minimnya *resource* yang di punya perangkatnya. Dengan *virtualisasi* yang menghasilkan peningkatan kinerja dari *docker*, dapat menghemat sumber daya yang di gunakan, dan dengan adanya *load balancing* yang dapat meringankan kinerja *server* sebagai penyedia sumber daya perangkat. Metode ini juga dapat di terapkan sebagai online *library* dan r*epository* yang dapat mengurangi kinerja *server* ketika *access user* sedang *peak*. Ada beberapa hal yang menunjang untuk lebih majunya *trend virtualisasi*. Dengan semakin banyaknya *physical server* juga akan semakin banyak sumber tenaga listrik yang digunakan, ini juga yang membuat semakin populernya *virtualisasi*.

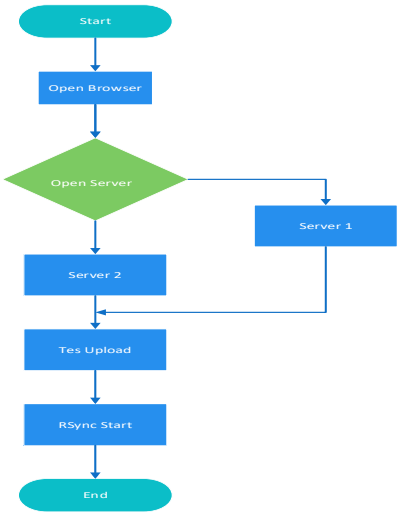
2. METODOLOGI PENELITIAN

Penelitian ini berfokus pada meringankan sumber daya yang digunakan oleh *server host* dengan metode *virtualisasi* yang berbeda dengan biasanya disediakan oleh beberapa aplikasi seperti *proxmox*, *virtualbox*, *vmware*, dll. Dengan rendahnya sumber daya yang digunakan dapat menunjang kinerja *server* sebagai perangkat *multitasking*. Dibawah ini diberikan perbandingan yang dikutip dari jurnal sebelumnya ketika menggunakan *virtualisasi* selain *Docker*. Pemanfaatan serta peningkatan kapabilitas di sisi pengguna yang juga sangat menguntungkan dengan rendahnya sumber daya yang terpakai.



Gambar 1. Tabel Perbandingan

Dapat dilihat dari tabel perbandingan antara *docker* dan *virtualbox* dalam menjalankan *virtualisasi* sebuah aplikasi atau *system operasi*. Di *virtualbox* ketika membutuhkan sebuah aplikasi yang akan dijalankan di *guest OS*, diperlukan instalasi untuk *guest OS* tersebut. Berbeda dengan *docker* yang hanya divirtualisasikan *binary* yang terkait oleh aplikasi atau *system operasi* yang akan dijalankan dan dapat meringankan kinerja *host* serta infrastruktur.

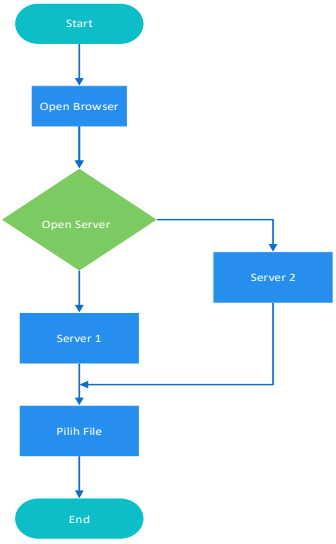


Gambar 2. Flowchart Upload

Submitted: **1/05/2020**; Accepted: **09/05/2020**; Published: **20/07/2020**

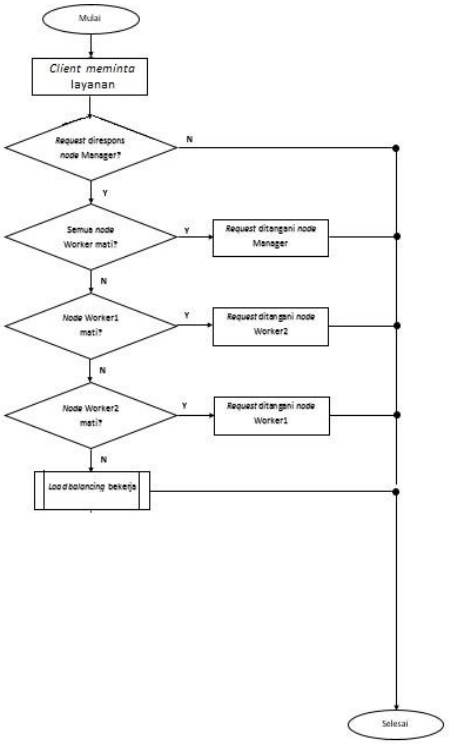
JURNAL MEDIA INFORMATIKA BUDIDARMA
Volume 4, Nomor 3, Juli 2020, Page 529-534
ISSN 2614-5278 (media cetak), ISSN 2548-8368 (media online)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/mib
DOI 10.30865/mib.v4i3.2147

Flowchart diatas sebagai gambaran untuk memberikan penjelasan proses berjalannya upload data ke *server* menggunakan *docker*. Dimulai dari *user* membuka *browser*, setelah itu akan tampil halaman *home browser* , kemudian *user* melakukan *upload* data. Setelah *user upload* data di *server*, sistem melakukan *RSync* yang artinya data yang tadi di *upload* oleh *user* di sinkronasikan ke *server* berikutnya. *RSync* adalah *tools* yang berfungsi untuk memindahkan dan mensinkronisasikan file secara efektif antara perangkat lokal, *remote server*, atau perangkat lainnya yang serupa. Bagi *user* sistem berbasis Linux, *command* ini sangat disarankan karena memungkinkan mereka untuk mengatur dan mengelola file atau direktori. *Sinkronisasi* folder atau penyalinan file secara manual akan sangat membuang-buang waktu. *Rsync* dapat melakukan banyak hal, termasuk menambahkan beberapa fitur canggih untuk menghemat waktu. Bahkan jika koneksi hilang saat memindahkan file, *tools* ini akan memulai dari proses pemindahan yang sempat terhenti ketika koneksi sudah kembali.



Gambar 3. Flowchart Download

Flowchart diatas sebagai gambaran untuk memberikan penjelasan cara download data dari *server* menggunakan *docker*. Dimulai dari user membuka *browser* dilanjutkan dengan muncul tampilan halaman *home* yang berisi data-data yang ada di *server*. Setelah itu, user mendownload data tersebut. Jika dalam mendownload data yang ada di *server* kemudian ada *user* lain mendownload data yang ada di *server* tersebut dengan berbarengan, maka user tersebut akan mendownload dari *server* yang satunya.

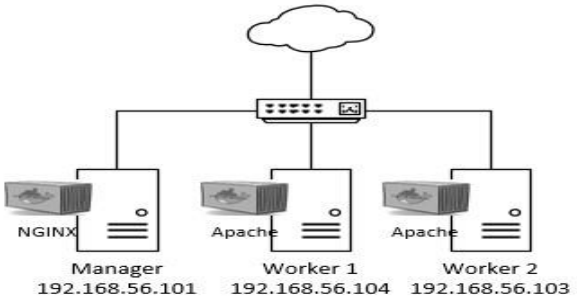


Gambar 4. Flowchart Perancangan

Submitted: **1/05/2020**; Accepted: **09/05/2020**; Published: **20/07/2020**

JURNAL MEDIA INFORMATIKA BUDIDARMA
Volume 4, Nomor 3, Juli 2020, Page 529-534
ISSN 2614-5278 (media cetak), ISSN 2548-8368 (media online)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/mib
DOI 10.30865/mib.v4i3.2147

Dari gambar *flowchart* diatas ini adalah cara kerja dari perancangan ini, dimana pada saat *client* meminta layanan atau akses itu akan dialihkan ke dalam *Nginx node manager*. *Node manager* yang berupa aplikasi *Nginx* akan meneruskan request untuk melihat *server* mana yang sedang hidup serta dengan penggunaan memory terendah agar tidak membebani *server* yang sedang melakukan proses untung melayani kebutuhan *user*. Dengan alur kerja seperti ini tingkat pembenan terhadap *server* akan di bagi jadi beberapa titik tidak berpusat sehingga dapat membuat path yang lebih cepat dan responsive.

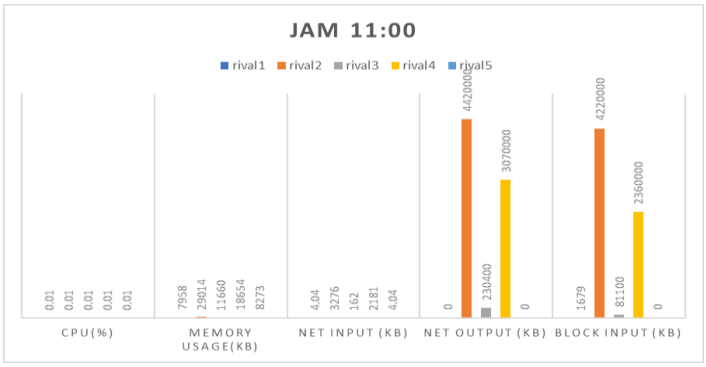


Gambar 5. Topology Yang Digunakan

Ini adalah bagaimana *topology* yang akan di gunakan dalam penelitian ini. Ada 3 komponen utama yaitu manager sebagai pembagi pembebanan dan *worker* sebagai *urldir app server*. Semua interkoneksi yang di terima akan melalui manager baru di teruskan ke masing masing *worker*. Pembagian pembebanan itu juga di konfigurasi di *node manager*.

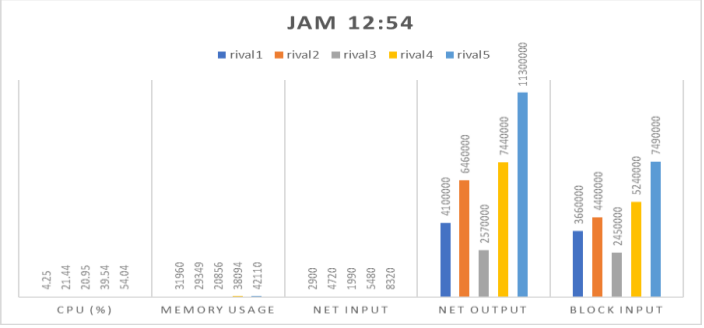
3. HASIL DAN PEMBAHASAN

Setelah penulis melakukan rangkaian pengujian pada 3 buah perangkat sebagai *client* yang terdiri dari *user* 1 adalah laptop yang mendownload menggunakan *software* pihak ketiga yaitu *Internet Download Manager* (IDM), dan *user* 2 adalah *handphone* yang berjumlah 2 unit *handphone* yang mendownload menggunakan aplikasi *Google Chrome*. Ketiga perangkat tersebut, masing-masing melakukan download file berkapasitas 8,8 GB yang ada di *server*. Dibawah ini adalah *timestep* dari hasil pengujian yang dilakukan secara serentak :



Gambar 6. Timestep pertama

Gambar 6 menjelaskan tentang *timestep usage server* yang sedang digunakan ketika download file. Pada gambar di atas, menunjukkan terdapat 5 *server* yang ke lima-limanya berjalan bersamaan. Pada grafik di atas penggunaan *usage* pada rival1, rival2, rival3, rival4 dan rival5 CPU, Memory Usage, Net Input, Net Output, Block Inputnya masih *relative* rendah.

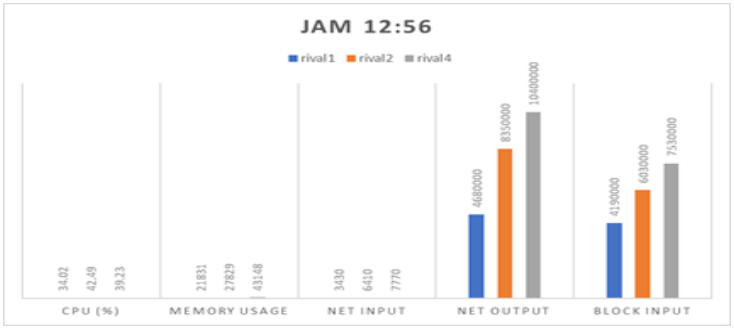


Gambar 7. Timestep kedua

Submitted: **1/05/2020**; Accepted: **09/05/2020**; Published: **20/07/2020**

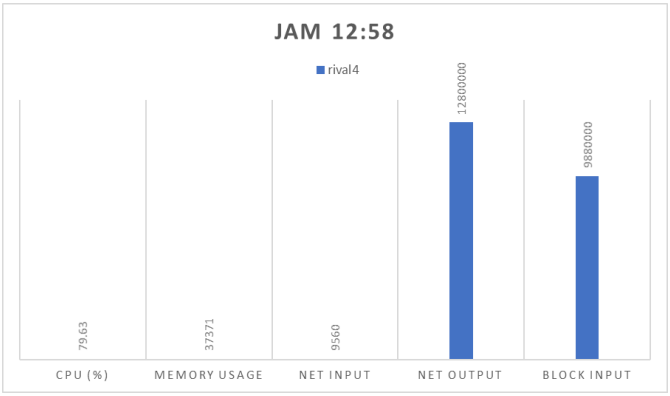
JURNAL MEDIA INFORMATIKA BUDIDARMA
Volume 4, Nomor 3, Juli 2020, Page 529-534
ISSN 2614-5278 (media cetak), ISSN 2548-8368 (media online)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/mib
DOI 10.30865/mib.v4i3.2147

Gambar 7 menunjukkan adanya peningkatan *traffic* pada Net Output dan Block Input. Sedangkan pada CPU, Memory Usage dan Net Input *usage* yang digunakkan masih rendah.



Gambar 8. Timestep ketiga

Gambar 8 menjelaskan uji coba menggunakan 3 *server*. Pada grafik di atas terdapat kenaikan penggunaan *usage server* yang signifikan pada Net Output dan Block Input.



Gambar 9. Timestep keempat

Gambar 9 menjelaskan uji coba menggunakan 1 *server*. Dan terlihat pada grafik tersebut *usage* Net Output dan Block Input masih sangat tinggi. Pada pengujian di atas, nilai penggunaan *resource* yang didapat dengan menggunakan aplikasi *Docker Start* dan ntop yang terpakai pada *server*. Semakin kecil jumlah penggunaan *resource*, maka semakin baik sistem tersebut dan juga semakin ringan sistem tersebut untuk dijalankan pada *server*. Pada pengujian di atas terlihat bahwa satu *host* menggunakan jumlah *resource* lebih tinggi dibanding dengan sistem yang menggunakan lima *host* pada *server*. Hal tersebut disebabkan karena penggunaan dalam *single server* mendapatkan beban kerja yang jauh lebih tinggi dibandingkan dengan menggunakan algoritma *Upstream Module, Weight, Hash, dan Max Fails*. Terjadi penurunan pada sistem *load balancing* dengan menggunakan algoritma *Upstream Module, Weight, Hash, dan Max Fails* karena beban kerja dibagi secara merata pada *server* lainnya dan tidak terpusat pada satu buah *web server*. Maka menggunakan *web server cluster* pada *virtualisasi container docker* akan meghemat penggunaan *resource* yang ada.

4. KESIMPULAN

Berdasarkan pengujian yang dilakukan, dapat disimpulkan bahwa:

1. Performa dan kemampuan *load balancing* dengan lima buah *web server cluster* dalam melayani *request client* jauh lebih bak dibandingkan *single server*, karena mengacu pada konsep *cluster computing* dimana penggunaan *cluster* dalam sebuah *server* tidak membebankan atau terpusat pada satu *server* saja sehingga tidak memberatkan *server*.
2. Pada sistem *load balancing* Nginx, algoritma *Upstream Module* berguna untuk menyeimbangkan penggunaan sumber daya atau *resource server* yang digunakan. Algoritma *Weight* melakukan perhitungan perbedaan kemampuan *processing* dari masing-masing *server* anggota *cluster*. *Administrator* memasukan secara manual parameter beban yang akan ditangani oleh masing-masing *server* anggota *cluster*, kemudian *scheduling sequence* secara otomatis dilakukan berdasarkan beban *server*. Algoritma *IP Hash* ini akan mendistribusikan *traffic* berdasarkan *IP* dari *client*. Nginx akan mencocokkan *web server backend* dengan *IP* yang dimiliki oleh *client*, sehingga apabila seorang *client* mendapatkan *respone* dari rival1 pada pertama kali, maka *request* selanjutnya juga akan mendapat *respone* dari rival1 dan seterusnya sampai *server* tersebut dianggap *down*. Jika *down*, maka *client* akan diarahkan ke rival2 dan seterusnya. Algoritma *Max Fails* adalah parameter untuk mengindikasikan berapa kali *server* tersebut diberi kesempatan jika terjadi gagal respon. Dan *Max Fails* akan

Submitted: **1/05/2020**; Accepted: **09/05/2020**; Published: **20/07/2020**

JURNAL MEDIA INFORMATIKA BUDIDARMA
Volume 4, Nomor 3, Juli 2020, Page 529-534
ISSN 2614-5278 (media cetak), ISSN 2548-8368 (media online)
Available Online at https://ejurnal.stmik-budidarma.ac.id/index.php/mib
DOI 10.30865/mib.v4i3.2147

dihitung apabila sudah melampaui batas *timeout* sejak *server* tersebut gagal merespon. Apabila *Max Fails* sudah tercapai maka Nginx akan menganggap *server* tersebut *down* yang kemudia Nginx akan mengarahkan *client* ke *server* lain dan mengantisipasi agar tidak ada *request client* yang mengarah ke *server* tersebut. Nginx akan tetap mencoba untuk memberikan *request* ke *server* tersebut, apabila sudah mendapat *respone* kembali maka *client* yang diarahkan ke *server* lain akan diarahkan kembali ke *server* tersebut.

3. Dengan menggunakan berbagai macam metode yang dapat digunakan untuk membantu membagi pembebanan pada setiap *server*, akan dapat mempengaruhi kinerja pada setiap *server*. Kinerja yang di hasilkan dari metode pembagian pembebanan yang beragam dapat menghasilkan pemrosesan yang lebih cepat.

REFERENCES

[1] Chung, M. T., Quang-Hung, N., Nguyen, M. T., & Thoai, N. (2016, July). Using docker in high performance computing applications. In *Communications and Electronics (ICCE), 2016 IEEE Sixth International Conference on* (pp. 52-57). IEEE.
[2] Adiputra, F. (2015). Container dan Docker teknik Virtualisasi dalam Pengelolaan Banyak Aplikasi Web.
[3] Jurenka, V. (2015). Virtualization using Docker Platform. Faculty of Informatics Masaryk University.
[4] Liu, D. &. (2014). The Research and Implementation of Cloud Computing Platform Based on Docker.
[5] Bik, M. F. (2017). Implementasi Docker Untuk Pengelolaan Bayak Aplikasi Web. Universitas Negri Surabaya.
[6] Fateh, J. R. (2018). Implementasi Web Server Dengan Pemanfaatan Virtualisasi Docker.
[7] Mohammad, R. M. (2019). Implementasi Load Balancing Server Web Berbasis Docker Swarm Berdasarkan Penggunaan Sumber Daya Memory Host.
[8] Singh, H. (2015). *WSQ:* Web Server Queueing Algorithm for Dynamic Load Balancing.
[9] NGIX Software.INC. (2014, January 1). NGIX. (NGIX Software.INC) Retrieved February 15, 2018, from NGIX:https://www.nginx.com/blog/.
[10] M. Aldi Aditia Putra. (2020). Implementasi High Availability Cluster Web Server Menggunakan Virtualisasi Container Docker.
[11] D. S. Afis, M. Data, dan W. Yahya, "Load Balancing Server Web Berdasarkan Jumlah Koneksi Klien Pada Docker Swarm,"vol. 3,no. 1,hal. 925-930, 2019.
[12] T. P. Kusuma, R. Munadi, dan D. D. Sanjoyo, "Implementasi dan Analisis Computer Clustering System dengan Menggunakan Virtualisasi Docker," *e-Proceeding Eng.*, vol. 4, no. 3,hal, 1-6, 2017.
[13] Dwi S. H. (2016). Perancangan Virtualisasi Sistem Operasi Dengan Menggunakan Docker Berbasis Terminal Emulator Untuk Peningkatan Performa.

Submitted: **1/05/2020**; Accepted: **09/05/2020**; Published: **20/07/2020**