Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 6, No. 5, Mei 2022, hlm. 2025-2033 http://j-ptiik.ub.ac.id

Implementasi *Load Balancing* dan *Failover* pada Proses Migrasi *Container Docker*

Shofura Naufal Rifiera[1], Heru Nurwasito[2]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]shofuranrifiera@student.ub.ac.id, [2]heru@ub.ac.id

Abstrak

Meningkat pesatnya penggunaan internet pada masa sekarang ini tentunya akan berdampak besar pula terhadap kinerja dari suatu web server. Web server tidak dapat merespon dengan cepat sehingga dapat menghambat proses komunikasi. Ketersediaan web server yang tidak didukung dengan metode dapat menghambat kinerja dari suatu server, dikarenakan terjadinya penumpukan request pada server dapat membuat kegagalan jaringan. Untuk itu maka dibetuklah perencanaan proses implementasi web server dinamis dengan menerapkan sistem docker dengan metode failover untuk dapat mengambil alih secara dinamis koneksi pada saat salah satu server down diusulkan dalama mengatasi masalah ini. Berdasarkan pada penelitian implementasi web server dinamis dengan mengimplementasikan Docker didapatkan kesimpulan yaitu, implementasi web server dinamis dengan docker sebagai manajemen kontainerisasi pada jaringan virtual. Dengan fasilitasi image web server dapan digunakan untuk menjalankan sebuah layanan open source sebagai web server dengan mengimplementasikan Bahasa PHP untuk ditampilkan pada halaman HTML yang langsung terhubung dengan data base sebagai media penyimpanan. Dengan mekanisme loadbalancing dan failover pada lingkungan docker sangat membantu dalam pendistribusian service web server dari node manager menuju node worker sebagai server cadangan pada saat terjadi kegagalan sistem pada salah satu server. Kemudian web server dinamis pada pengujian failover memiliki rata-rata waktu response time terendah berada pada node worker 2 dengan nilai 18,90ms, perolehan waktu yang didapat tentunya dipengaruhi oleh interval waktu delay yang telah ditentukan selama load testing service web server dijalankan.

Kata kunci: *load balancing, failover, docker, server, virtual*

Abstract

The rapidly increasing use of the internet today will of course have a major impact on the performance of a web server. The web server cannot respond quickly so that it can hamper the communication process. The availability of a web server that is not supported by a method can hamper the performance of a server, due to the accumulation of requests on the server which can cause network failure. For this reason, a dynamic web server implementation planning process was formed by implementing a docker system with the failover method to be able to dynamically take over the connection when one of the servers is down is proposed to overcome this problem. Based on the research on the implementation of dynamic web servers by implementing Docker, it can be concluded that the implementation of dynamic web servers with Docker as containerization management on virtual networks. With the facilitation of the web server image, it can be used to run an open source service as a web server by implementing the PHP language to be displayed on HTML pages that are directly connected to the data base as storage media. With loadbalancing and failover mechanisms in the docker environment, it is very helpful in distributing web server services from node manager to node worker as a backup server in the event of a system failure on one of the servers. Then the dynamic web server in failover testing has the lowest average response time at node worker 2 with a value of 18.90ms, the time obtained is certainly influenced by the delay time interval that has been determined during load testing of the web server service.

Keywords: *load balancing, failover, docker, server, virtual*

1. PENDAHULUAN

Baru-baru ini, virtualisasi berbasis *container* semakin populer. Salah satu *virtualisasi* berbasis *container* yang paling banyak digunakan saat ini adalah Docker. Docker merupakan sebuah aplikasi *platform virtualisasi container* yang bersifat *open source*, yang menyediakan *platform* terbuka untuk developer maupun sistem admin untuk dapat membangun, mengemas, dan menjalankan aplikasi dimanapun sebagai sebuah *container* (Docker, 2018). Docker dan mesin virtual memiliki persamaan sumber daya dan alokasi yang sama, tetapi berfungsi secara berbeda karena *docker* menggunakan sistem operasi tunggal pada *host* sehingga dapat menjalankan banyak aplikasi yang berbeda, sedangkan virtual mesin dapat menjalankan sistem operasi yang berbeda. *Docker* lebih efektif dalam hal penggunaan sumber daya mesin *host* dibandingkan dengan *virtual machine*, karena dalam proses deployment-nya, *docker* akan menjalankan sebuah *container* menggunakan *base image* dengan metode *file system as a layer* yang berarti *docker* hanya akan menyalin lapisan perubahannya saja untuk dijalankan sebagai duplikasi *container* yang berbeda dengan *base image* yang sama, berbeda dengan virtual mesin dimana kita harus menyalin seluruh co*ntainer* beserta *kernel* dan librari yang berada di dalamnya, hal ini mengakibatkan adanya pengehematan penyimpanan dan memory pada *docker*.

Docker dapat melakukan migrasi *container* dari *docker* mesin satu ke *docker* mesin dua dengan *ip address* atau *domain* yang berbeda nantinya. Namun ditemukan permasalahan dimana saat melakukan migrasi *container* dari satu *docker* mesin ke *docker* mesin yang lain koneksi *user* akan terputus karena alamat *ip address* atau *domain docker* mesin berbeda. Oleh karena itu untuk mendukung penggunaan secara *real-time* diperlukannya penerapan *load balancing* sebagai alternatif untuk mengambil alih secara dinamis koneksi TCP dan permintaan HTTP yang tertunda dari koneksi *server* asli. Migrasi bisa digunakan untuk meningkatkan kehandalan *server* Web dengan hanya sedikit kesalahan dalam kinerja (Rawal et al., 2011). *Load balancing* dapat memaksimalkan *throughput*, mengurangi *latency* dan memastikan *fault-tolerant.* Load balancing sendiri bekerja sebagai pengalokasian beban secara dinamis untuk memenuhi kebutuhan penggunaan layanan dan sumber daya secara maksimal dengan mendistribusikan semua beban ke beberapa node yang tersedia (Setyawan et al., 2014).

Web server merupakan penyedia layanan yang ditampilkan pada halaman web atau dokumen *html* sesuai dengan permintaan dari pengguna. Web server sendiri memiliki 2 kategori yaitu statis dan dinamis, web server statis dilakukan konfigurasi secara statis yang hanyak menampilkan informasi yang terkandung pada *website* yang tidak dapat diupdate melalui aplikasi website sehingga mengharuskan menggubah script. Sedanngkan pada web server dinamis dapat dilakukan update pada informasi-informasi yang ditampilkan dengan menerapkan aplikasi dinamsi yang mendukung web server dinamis. Masalah utama pada web server yaitu pada sisi ketersediaan yang dimana apabila web server hanya menyediakan beberapa halaman web statis seperti bantuan *online* sederhana dan pengenalan sistem maka web server statis dapat dimanfaatkan, namun apabila diperlukan web server yang dapat meningkatkan ketersediaan dan dapat berkomunikasi secara *real-time* maka web server dinamis dapat dijadikan solusi dalam kondisi ini (Liu & Cheng, 2010)

Penelitian ini akan berfokus kepada bagaimana *user* tetap dapat menerima layanan yang diinginkan meskipun terjadi *down* pada *server* yang pertama kali diakses yang akan diteruskan langsung ke *server backup* tanpa harus melakukan *request* ulang sehingga system yang dibuat akan lebih efisien dari pada user harus melakukan *request* ulang ke alamat *server* yang lain. Penelitian ini diharapkan mengurangi tingkat kegagalan yang terjadi pada web server pada saat server menerima ataupun mengirim balasan terhadap permintaan *user*. Berdasarkan penjabaran diatas, peneliti memiliki hipotesis bahwa pengunaan docker dalam penerapan web server dinamis memiliki kinerja yang lebih baik dalam kesiapan pembuatan sistem dengan level yang lebih tinggi, dibandingkan dengan web server yang tidak menggunakan *docker container*.

2. DASAR TEORI

2.1 Virtualisasi

Virtualisasi merupakan teknik yang memungkinkan pengguna untuk membuat

perangkat virtual atau sumber daya, seperti server, perangkat penyimpanan, jaringan, atau sistem operasi. Virtualisasi mengacu pada sebuah komputer yang menjalankan beberapa sistem operasi secara bersamaan. Aplikasi yang berjalan di mesin virtual dapat berjalan seolah-olah mereka berada di mesin mereka sendiri, dimana sistem operasi, librari, dan program lain bersifat unik untuk sistem virtualisasi dan tidak terhubung ke sistem operasi *host* yang berada di bawahnya. Untuk pengguna *desktop*, penggunaan yang paling umum adalah untuk dapat menjalankan aplikasi yang dimaksudkan untuk sistem operasi yang berbeda tanpa harus berpindah komputer atau *reboot* ke sistem yang berbeda (Naik, 2016). Virtualisasi juga menawarkan kemampuan untuk menjalankan sistem operasi yang berbeda dan memiliki keunggulan yaitu, dapat membagi sistem yang besar menjadi beberapa bagian yang lebih kecil, memungkinkan *server* untuk digunakan secara lebih efisien oleh sejumlah pengguna yang berbeda atau aplikasi dengan kebutuhan berbeda. Ada begitu banyak teknologi virtualisasi populer yang tersedia seperti VMWare, VirtualBox, Parallels, QEMU, UML dan Xen.

2.3 VirtualBox

VM VirtualBox adalah perangkat lunak virtualisasi, yang dapat digunakan untuk menjalankan suatu sistem operasi didalam sistem operasi. VirtualBox merupakan produk virtualisasi x86 dan AMD64 / Intel64 yang bagus untuk perusahaan maupun penggunaan di pribadi. VirtualBox memiliki sangat banyak fitur, dan juga memiliki performa yang bagus dan cocok untuk di gunakan di suatu perusahaan. VirtualBox juga menawarkan solusi profesional yang tersedia secara bebas sebagai Perangkat Lunak Open Source dibawah ketentuan GNU General Public License (GPL) versi 2.

Saat ini, VirtualBox dapat berjalan pada Windows, Linux, Macintosh, dan Solaris *host*. VirtualBox mendukung sejumlah besar *guest operating systems* tetapi tidak terbatas pada Windows (NT 4.0, 2000, XP, Server 2003, Vista, Windows 7, Windows 8, Windows 10), DOS / Windows 3.x, Linux (2.4, 2.6, 3.x dan 4.x), Solaris dan OpenSolaris, OS / 2, dan OpenBSD. VirtualBox sedang dikembangkan secara aktif dan memiliki daftar fitur yang terus berkembang, mendukung *guest operating systems* dan *platform* yang dijalankannya.

2.4 Web Server

Web server merupakan software yang menyediakan layanan berbasis data dan berfungsi sebagai penerima permintaan dari HTTP atau HTTPS pada klien, untuk mengirimkan kembali yang hasilnya dalam bentuk halaman web yang umumnya berupa dokumen HTML. Fungsi utama sebuah web server yaitu mentransfer berkas atau permintaan pengguna melalui protokol komunikasi yang telah ditentukan, salah satu *web server* yang sering digunakan adalan *apache* karena sudah didukung terhadap kontrol akses PHP dan SSL. *Web server* juga berfungsi mengelolah transfer seluruh aspek pemberkasan dalam sebuah halaman web terkait termasuk didalamnya teks, gambar, video, dan lain-lain (Solichin, 2005).

2.5 Docker

Docker adalah teknologi perangkat lunak yang menyediakan virtualisasi tingkat sistem operasi pada Windows dan Linux. Pengaturan dan hal pendukung lainnya akan dibangun menjadi sebuah *image* yang dimana akan didapat suatu hasil instansiasi dari *image* tersebut yang dinamakan *container*. Docker menggunakan fitur isolasi sumber daya dari kernel Linux seperti *group* dan *namespace* kernel sehingga proses dari *container* tidak akan mengganggu *host* dan *container* lain. *Container* juga memiliki pengaturan jaringan tersendiri sehingga sebuah *container* tidak akan memiliki hak akses dari *socket* atau *interface container* lain (Turnbull, 2014). Docker merupakan arsitektur klien server sehingga *docker container* akan didistribusikan oleh *docker daemon* pada saat dihubungi oleh *docker* klien dengan jalur komunikasi menggunakan *socket* API yang sudah disediakan oleh *docker*, keduanya akan berjalan secara bersamaan.

Perbedaan antara *virtual machine* dengan *docker* sangat terlihat dari segi strukturnya, *virtual machine* sendiri membutuhkan ruang yang banyak untuk menyimpan *guest* OS.

2.6 Docker Container

Docker *container* merupakan komponen untuk menjalankan *docker* sama seperti direktori. *Container docker* memegang segala sesuatu yang diperlukan untuk menjalankan aplikasi. Setiap *container* lahir dari *docker Image*. *Container docker* dapat dijalankan,

dimulai, berhenti, dipindah, dan dihapus. Setiap *container* merupakan *platform* aplikasi yang terisolasi secara aman. Sebuah *container* memberikan solusi untuk mengisolasi suatu proses pada keseluruhan sistem (Dua et al., 2014).

2.7 Docker Swarm

Load balancing meningkatkan distribusi beban kerja di beberapa sumber komputasi, seperti komputer, klaster komputer, tautan jaringan, unit pemrosesan pusat, atau *disk drive*. *Load balancing* bertujuan untuk mengoptimalkan penggunaan sumber daya, memaksimalkan throughput, meminimalkan waktu respon, dan menghindari kelebihan sumber daya tunggal. Menggunakan beberapa komponen dengan *load balancing*. *Load balancing* biasanya melibatkan perangkat lunak atau perangkat keras khusus, seperti *switch multi layer* atau proses *server* DNS(*Domain Name System*) (Setyawan et al., 2014).

Algoritma *load balancing* yang digunakan menentukan yang mana dari server yang bekerja secara optimal pada backend yang akan dipilih. Ada beberapa algoritma yang umum digunakan yaitu *Round Robin* dan *Least Connections*.

Algoritma penjadwalan *roundrobin* meneruskan setiap permintaan masuk ke *server* berikutnya dalam daftar. Jika kita memiliki 4 *server cluster* (*server* A, B, C dan D) *request* 1 akan pergi ke server A, *request* 2 akan pergi ke *server* B, *request* 3 akan pergi ke *server* C, *request* 4 akan masuk ke *server* D, dan *request* 4 akan masuk ke *server* A sehingga *packet request* akan menyelesaikan siklus *roundrobin* pada *server*. Algoritma penjadwalan *leastconn* mengarahkan koneksi jaringan ke *server* dengan jumlah *koneksi* yang paling sedikit. Algoritma *leastconn* adalah salah satu algoritma penjadwalan dinamis, karena perlu menghitung koneksi langsung untuk setiap *server* secara dinamis (Mustafa & Affiliation, 2017).

2.10 Apache

Apache adalah *web server* paling sederhana dan populer di internet. Digunakan untuk melayani request-response HTTP dan *logging* informasi secara *detail* dari semua situs web yang aktif, meskipun ada banyak *web server* layak yang akan melayani suatu konten. *Apache* dapat dijalankan pada kebanyakan website daripada kombinasi server lainnya karena *Apache* merupakan software yang baik. Adapun keuntungan dari *Apache* adalah tidak dikenai biaya dalam menggunakan software ini (Coar & Bowen, 2004).

3. METODOLOGI

3.1. Rekayasa Kebtuhan

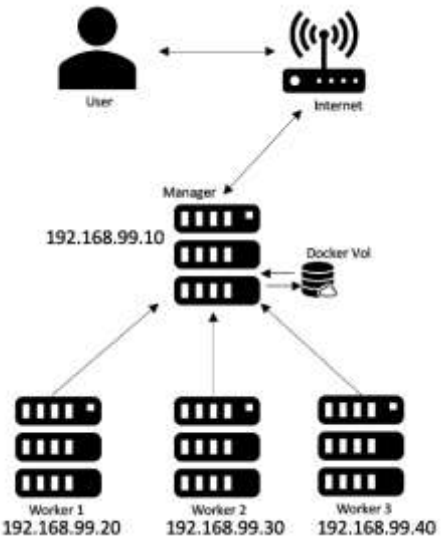
Tahap ini merupakan tahapan yang dilakukan untuk merancang kebutuhan apa saja yang dibutuhkan dalam pembangunan sistem, kebutuhan yang menunjang pada penelitian ini terdapat 2 kebutuhan yaitu kebutuhan fungsional dan kebutuhan non-fungsional..

Kebutuhan yang harus dapat dilakukan oleh sebuah layanan sistem yang dikembangkan merupakan kebutuhan fungsional. Berikut beberapa kebutuhan fungsional pada implementasi *load balancing* dan *fail over* pada proses migrasi *containrer docker:*

1. Sistem dapat menyediakan layanan yang nantinya bisa diakses oleh *user,* penelitian ini dibangun dengan memanfaatkan sistem operasi *linux*, *apache* dan *php* menggunakan *docker swarm.*
2. Sistem dapat berjalan seperti apa yang diinginkan dan dapat merespon *request* dari *user* meskipun mengalami *down* pada *server* yang menerima *request*.

Kebutuhan non fungsional mengarah pada kebutuhan perangkat keras dan perangkat lunak yang dibutuhkan.

3.2. Perancangan



Gambar 1. Perancangan Sistem

Sistem yang akan dibuat pada penelitian yang dilakukan. diluar 4 komponen yang menjadi pendukung sistem seperti yang sudah dijelaskan terdapat docker volume yang menjadi wadah untuk melakukan sharing data antar kontainer yang diinginkan. Supaya komponen-komponen bekerja dengan baik maka dibutuhkan koneksi jaringan internet yang stabil tentunya. Web server tentu tidak dapat langsung dijalankan, maka dari itu dibutuhkan sebuah image yang dibuat dengan docker file, didalam docker file itulah berisikan tools-tools yang dibutuhkan seperti apache2, mysql dan hph yang berupa kode program HTML.

3.2. Implementasi

Implementasi dimulai dengan menyiapan layanan virtualisasi dengan virtualbox. Setelah konfigurasi alamat IP maka mengimplementasikan Docker dengan beberapa implementasi docker seperti dockerfile, docker compose, docker swarm init dan join, docker pull image, dan docker swarm deploy.

Setelah semua kebutuhan dan perancangan sudah berhasil diimplementasikan, maka dari itu pada proses implementasi akan dilakukan percobaan pada sistem yang sudah dibuat apakan implementasi yang sudah dibuat sudah mengacu terhadap kebutuhan sistem dan perancangan sistem. Berikut hasil dari implementasi yang sudah dilakukan seperti pada Gambar 2

Gambar 2. Hasil Implementasi

4. HASIL DAN PEMBAHASAN

4.1 Pengujian Fungsional

Melakukan pengecekan pada fungsional sistem satu persatu untuk menjawab rumusan masalah yang hasilnya akan ditampilkan pada tabel keberhasilan. Pengujian fungsional terdiri dari pengujian web server dan pengujian *failure*.

Skenario yang sudah dilakukan pada pengujian web server dapat mengakes halaman web dengan mengunakan alamat IP pada masing-masing node. Keempat node tersebut berjalan sesuai dengan perannya, node pertama sebagai manager, node kedua sampai dengan keempat sebagai worker, berikut tampilan halaman utama web server seperti pada Gambar 3.

Gambar 3. Hasil Pengujian Web Server

Hasil yang didapat dari pengujian web server dengan menjalankan masing-masing skenario didapatkan hasil yang sesuai, dimana web server dapat diakses melalui browser sehingga dapat menampilkan homepage. Maka dari itu dapat dikatakan pengujian pada web server berhasil dan hasil yang didapatkan valid sesuai dengan yang diharapkan.

Selanjutnya pengujian failure pengujian pertama mematikan layanan pada worker 1 maka layanan web server akan secara otomatis berpindah secara acak terhadap worker yang masih menyala. Hasil dari pengujian yang dilakukan layanan pada worker 1 yang sudah dimatikan diambil alih secara otomatis oleh worker 3 walaupun pada worker 3 mengalami penumpukan layanan sehingga layanan yang ingin diakses pada worker 1 tetap akan direspon oleh worker 3 sehingga diketahui pada saat worker 1 mati dan worker 3 dengan otomatis mengambil alih layanan sehingga tetap dapat diakses oleh node manager seperti pada Gambar 4.

Gambar 4. Hasil Pengujian Worker 1 Down

Pengujian kedua yang sudah dilakukan dengan mematikan layanan pada worker 3 maka layanan web server akan secara otomatis secara acak akan diambil alih oleh worker yang masih

menyala. Hasil dari pengujian yang dilakukan layanan pada worker 1 dan worker 2 yang menyala secara otomatis akan mengambil alih dari layanan worker 3 dan worker 3 walaupun pada worker 1 mengalami penumpukan layanan sehingga layanan yang ingin diakses dengan otomatis diambil alih worker 1 sehingga tetap dapat diakses oleh node manager seperti pada Gambar 5.

Gambar 5. Hasil Pengujian Worker 1 dan Worker 2 Down

Pengujian ketiga yang sudah dilakukan dengan mematikan layanan pada worker 3 maka layanan web server akan secara otomatis ke worker 1. Hasil dari pengujian yang dilakukan layanan pada worker 1 yang menyala secara otomatis akan mengambil alih dari layanan worker 2 dan worker 3 walaupun pada worker 1 mengalami penumpukan layanan sehingga layanan yang ingin diakses dengan otomatis diambil alih worker 1 sehingga tetap dapat diakses oleh node manager seperti pada Gambar 6

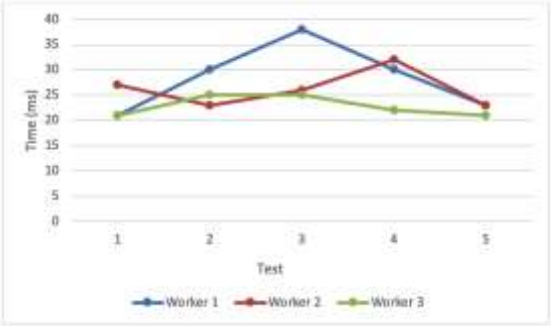
Gambar 6. Hasil Pengujian Worker 3 Down

Hasil yang didapat dari pengujian failur untuk mengetahui apakah worker dapat dengan otomatis mengambil alih layanan dari suatu worker yang mati secara otomatis dengan mengakses alamat IP yang aktif. Sistem yang berjalan dapat memberikan respon layanan walaupun terdapat worker yang down dengan memanfaatkan mekanisme failover.

Maka dari itu dapat dikatakan pengujian pada web server berhasil dan hasil yang didapatkan valid sesuai dengan yang diharapkan.

4.2 Pengujian Failover

Langkah pengujian failover akan dilakukan pengujian sebanyak 5 kali percobaan, 10 kali percobaan, 15 kali percobaan dan 20 kali percobaan menggunakan *web server stress tools*.



Gambar 7. Hasil Pengujian 5 Percobaan

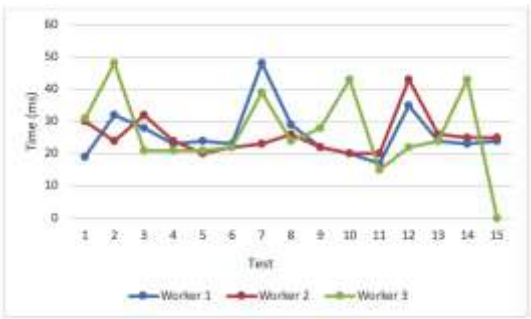
Grafik yang ditunjukkan Gambar 7 menampilkan hasil dari lima kali percobaan setelah dilakukan pengujian parameter nilai waktu respon dari masing-masing worker pada setiap percobaan terjadi kenaikan dan penurunan waktu respon disetiap percobaan. Pada worker 1 waktu terendah terdapat pada percobaan pertama dengan waktu 21ms dan tertinggi pada percobaan ketiga yaitu 38ms, sedangkan pada worker 2 waktu terendah terdapat pada percobaan kedua dan kelima yaitu 23ms dan waktu tertinggi pada percobaan keempat dengan waktu respon 32ms. Percobaan pertama dan terakhir memiliki waktu terendah pada worker 3 dengan waktu 21ms sedangan percobaan kedua dan ketiga memiliki waktu respon 25ms sebagai percobaan dengan waktu tertinggi.

Data yang bervariatif didapatkan setelah melakukan lima kali percobaan pada masing-masing worker. Pada percobaan kelima didapatkan waktu respon yang hampir sama dengan waktu 23 ms dan 21 ms. Waktu respon tertinggi adalah 38 yang diperoleh worker 1 sedangkan yang terendah dengan waktu respon 21ms diperoleh woker 1 dan worker 3. Dalam satu menit waktu percobaan pada lima kali percobaan yang telah dilakukan semua web server dapat melaksanakan tuugasnya dengan baik. Mekanisme *load testing* pada lima kali percobaan telah bekerja dengan baik.

Gambar 8. Hasil Pengujian 10 Percobaan

Grafik yang ditunjukkan Gambar 8 menampilkan bahwa waktu respon dari masing-masing worker dari 10 kali percobaan mengalami peningkatan dan penurunan disetiap percobaannya. Waktu terendah pada worker 1 diperoleh percobaan kelima dengan waktu 21 ms dan waktu tertingginya pada percobaan ketujuh yaitu 37 ms. Percobaan ketujuh juga menjadi waktu tertinggi untuk worker 2 dengan waktu respon 48 ms sedangkan percobaan pertama menjadi waktu terndah untuk worker 2 yaitu 20 ms. Sedangkan untuk worker 3 waktu terendah pada percobaan keempat dengan 19ms dan waktu tertinggi pada percobaan ketujuh seperti pada worker 1 dan worker 2 dengan waktu respon 30 ms.

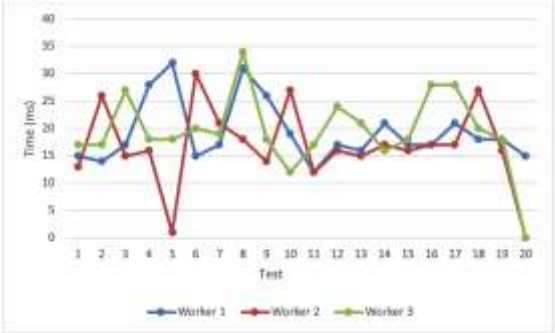
Setelah 10 kali percobaan pada masing-masing worker didaptkan hasil yang beragam. Pada percobaan ketujuh dapat dikatakan menjadi percobaan dengan waktu tertinggi pada seluruh worker. 48 ms adalah waktu tertinggi dari 10 kali percobaan yang terjadi pada percobaan ketujuh diperoleh worker 2 sedangkan waktu respon terendah adalah 19 ms diperoleh worker 3 pada percobaan keempat. Terjadi peningkatan waktu yang drastis pada percobaan 5, percobaan 7 dan percobaan 9 terutama pada worker 2. Walaupun terjadi peningkatan yang drastis pada beberapa percobaan, percobaan 10 kali pada masing-masing worker dapat berjalan telah bekerja dengan baik dalam satu menit setiap percobaannya.



Gambar 9. Hasil Pengujian 15 Percobaan

Gambar 9 merupakan grafik dari hasil waktu respon dari 15 kali percobaan pada masing-masing worker yang mengalami perubahan waktu respon disetiap percobaan. Worker 1 dan worker 3 memperoleh waktu respon 48 ms sebagai waktu tertinggi dimana worker satu mendapatkannya pada percobaan ketujuh sedangkan worker 3 pada percobaan kedua. Untuk waktu terendah worker 1 adalah 17 ms pada percobaan ke-10 dan worker 3 adalah 0 ms pada percobaan ke-15. Percobaan ke-12 menjadi waktu respon tertinggi untuk worker 2 dengan waktu respon 43 ms dan percobaan kelima, ke-10 dan ke-11 menjadi waktu respon terendah dengan waktu respon 20 ms.

Dari 15 kali percobaan kedua dan ketujuh memiliki waktu respon tertinggi dengan waktu respon 48 ms yang diperoleh worker3 dan worker. Sedangkan percobaan ke 15 memiliki waktu respon terendah yaitu 0 ms yang diperoleh worker 3. Perubahan waktu yang drastis terjadi kembali dalam 15 kali percobaan, perubahan drastis sering terjadi pada worker 3. Setiap worker menjalankan simulasi selama satu menit berulang hingga 15 kali dan worker dapat melaksanankan tugasnya tetapi pada percobaan ke-15 pada worker 3 memilki waktu respon 0 ms atau tidak mendapatkan waktu respon dikarekan kehabisan waktu respon yang telah diberikan saat melaksanakan tugasnya.15 percobaan dapat berjalan dengan baik.

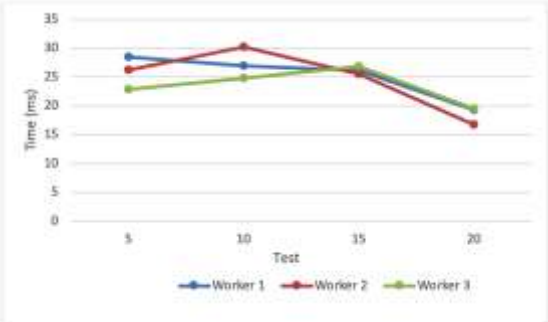


Gambar 10. Hasil Pengujian 20 Percobaan

Gambar 10 menapilkan grafik dari percobaan yang telah dilakukan sebanyak 20 kali pada masing-masing worker. 32 ms pada percobaan kelima adalah waktu respon tertinggi dan 12 ms pada percobaan ke-11 adalah waktu terendah dari worker 1. Worker 2 dan worker 3 memiliki waktu respon yang sama yaitu 0 ms pada percobaan ke-20 dimana menjadi waktu terendah worker 2 dan worker 3. Sedangkan untuk waktu tertinggi worker 2 adalah 30 ms pada percobaaan ke 6, pada percobaan kedelapan

menjadi waktu tertinggi worker 3 dengan waktu respon 34 ms.

Waktu respon tertinggi dalam 20 kali percobaan dimiliki worker 3 dengan 34 ms pada percobaan kedelapan. Sedangkan waktu respon terendah diperoleh worker 2 dan worker 3 dengan waktu respon 0 ms pada percobaan ke-20. Pada percobaan ke 20 worker 2 dan worker 3 memiliki waktu respon 0 ms yang berarti worker 2 dan worker 3 kehabiskan waktu respon yang diberikan untuk memberikan respon. Percobaan 20 kali pada mekanisme failover dengan menggunakan *load testing* berjalan dengan baik dimana setiap percobaan memiliki durasi satu menit untuk melaksanakan tugas masing masing worker.



Gambar 10. Hasil Pengujian Failover

Gambar 11 menampilkan grafik dari rata-rata hasil percobaan uji parameter waktu respon dari setiap percobaan pada masing-masing worker yang telah dilakukan. Untuk hasil rata-rata terendah pada 5 kali percobaan diperoleh worker 3 dengan rata-rata 22,8 ms dan worker 1 memiliki rata-rata tertinggi dengan rata-rata waktu respon 28,4ms. Pada pengujian 10 kali percobaan didapatkan rata-rata terendah pada worker 3 dengan rata-rata waktu respon 24,8 ms sedangkan untuk rata-rata waktu respon tertinggi diperoleh worker 2 dengan rata-rata waktu respon 30,2 ms. Rata-rata tinggi dari pengujian 15 kali percobaan adalah worker 3 dengan rata-rata waktu respon 26,8 ms dan yang terendah adalah worker 2 dengan rata-rata waktu respon 35,47 ms. 19,5 ms adalah rata-rata worker 3 pada pengujian 20 kali percobaan dan menjadi yang rata-rata waktu respon tertinggi. 16,7 ms adalah waktu rata-rata waktu terendah pada pengujian 20 kali percobaan yang diperoleh worker 2, namun rata-rata waktu tersebut tidak dapat dikatakan *valid* dikarekan pada pengujian 20 kali percobaan saat percobaan yang ke-20 kali didapatkan nilai waktu respon 0 ms pada worker 2 dan worker 3 dimana artinya worker 2 dan worker 3 tidak dapat menyelesaikan tugasnya karena kehabiskan waktu respon yang telah diberikan. Maka rata-rata waktu respon yang dapat dikatakan *valid* adalah 19,3 ms pada worker 1.

Rata-rata hasil waktu yang didapatkan sangat beragam dan setiap worker memiliki rata-rata waktu yang berselisih. Faktor perbedaan rata-rata waktu respon dari setiap worker adalah pengaruh dari proses *load testing* yang terjadi sedangkan faktor selisih didapatkan dari interval waktu jeda yang diberikan pada setiap percobaan yang dilakukan dan juga pengujian trafik internet yang digunakan dapat mempengaruhi saat mengalami kenaikan saat terjadinya proses *load testing* maka dari itu hasil dari setiap percobaan yang didapatkan sangat terpengaruh.

Berdasarkan grafik pada Gambar6.17 dapat disimpulkan rata-rata waktu respon keseluruhan yang telah didapatkan waktu respon terendah adalah 19,3 ms pada worker 1 dalam pengujian 20 kali percobaan. Rata-rata hasil uji parameter waktu respon didapatkan setelah melakukan *load testing* dengan durasi satu menit pada setiap percobaan yang dipengaruhi oleh interval jeda yang telah ditentukan.

5. PENUTUP

5.1 Kesimpulan

Berdasarkan pada penelitian implementasi web server dinamis dengan mengimplementasikan Docker didapatkan kesimpulan sebagai berikut:

1. Implementasi web server dinamis dengan docker sebagai manajemen kontainerisasi pada jaringan virtual. Dengan fasilitasi image web server dapan digunakan untuk menjalankan sebuah layanan open source sebagai web server dengan mengimplementasikan Bahasa PHP untuk ditampilkan pada halaman HTML yang langsung terhubung dengan data base sebagai media penyimpanan. Dengan mekanisme loadbalancing dan failover pada lingkungan docker sangat membantu dalam pendistribusian service web server dari node manager menuju node worker sebagai server cadangan pada saat terjadi kegagalan sistem pada salah satu server,

2. Web server dinamis pada pengujian failover memiliki rata-rata waktu response time terendah berada pada node worker 2 dengan nilai 18,90ms, perolehan waktu yang didapat tentunya dipengaruhi oleh interval waktu *delay* yang telah

ditentukan selama *load testing service web* server dijalankan.

5.2 Saran

Berdasarkan penelitian yang dilakukan terdapat aspek-aspek yang harus di kembangkan untuk mendukung kinerja dari sistem webserver sehingga didapatkan hasil yang lebih maksimal. Berikut merupakan saran yang dapat disampaikan antara lain:

1. Mengimplementasikan sistem perhitungan waktu untuk mendapatkan nilai yang lebih akurat.

2. Memberika sistem keamanan yang handal untuk diterapkan pada sistem server seperti mengimplementasikan *cryptografi*.

3. Mengembangkan kontainerisasi ke kehidupan nyata untuk membantu server dalam memanajemen permintaan *user*.

6. DAFTAR PUSTAKA

(2021, Januari 17). Retrieved from HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer: http://www.haproxy.org/#desc

(2021, Januari 5). Retrieved from NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy: https://www.nginx.com/resources/wiki/

7. Abdullah, A., Simamora, S. N., & Andrian, H. R. (2010). Implementasi dan Analisa Load-Balancing pada suatu Web Server Lokal.

Afis, D. S., Data, M., & Yahya, W. (2019). oad Balancing Server Web Berdasarkan Jumlah Koneksi Klien Pada Docker Swarm. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*.

Coar, K., & Bowen, R. (2004). *Apache Cookbook.* O'Reilly Media, Inc.

Dani, R., & Suryawan, F. (2017). Perancangan dan Pengujian Load Balancing dan Failover Menggunakan NginX. *Khazanah Informatika Jurnal Ilmu Komputer dan Informatika*.

Dua, R., Raja, A. R., & Kakadia, D. (2014). Virtualization vs Containerization to Support Paas. *IEEE International Conference on Cloud Engineering*, 610-614.

Liu, Y., & Cheng, X. (2010). Design and implementation of embedded Web server based on arm and Linux. *The 2nd International Conference on Industrial Mechatronics and Automation*.

Mustafa, D. E., & Affiliation, H. (2017). Load Balancing Algorithms Rond-robin (RR), Least-Connection, and Least Loaded Efficiency. *GESJ: Computer Science and Telecommunications* .

Naik, N. (2016). Migrating from Virtualization to Dockerization in the Cloud: Simulation and Evaluation of Distributed Systems. *2016 IEEE 10th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments (MESOCA)*.

Nurwarsito, H., & Sejahtera, V. B. (2020). Implementation of Dynamic Web Server Based on Operating System-Level Virtualization using Docker Stack . *2020 12th International Conference on Information Technology and Electrical Engineering (ICITEE), Yogyakarta, Indonesia*.

Rawal, B. S., Karne, R. K., & Wijesinha, A. L. (2011). Splitting HTTP requests on two servers. *2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011)*.

Rouse, M. (2021, Januari 19). *What is Docker Swarm?* Retrieved from https://searchitoperations.techtarget.com/definition/Docker-Swarm

Setyawan, R. A., Muttaqin, A., Razak, A. A., & Risman, L. (2014). Analisis Mekaniseme Multi Server Load Balancing pada Server SIAKAD Universitas Brawijaya. *Jurnal EECCIS Vol. 8, No. 1*.

Solichin, A. (2005). *Pemrograman Web dengan PHP dan MySQL.* Jakarta: Universitas Budi Luhur.

Sumbogo, Y. T., Data, M., & Siregar, R. A. (2018). Implementasi Failover Dan Autoscaling Kontainer Web Server Nginx Pada Docker Menggunakan Kubernetes. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*.

Turnbull, J. (2014). *The Docker Book.* Turnbull Press.