Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 3, No. 1, Januari 2019, hlm. 925-930 http://j-ptiik.ub.ac.id

Load Balancing Server Web Berdasarkan Jumlah Koneksi Klien Pada *Docker Swarm*

Dimas Setiawan Afis[1], Mahendra Data[2], Widhi Yahya[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]dimassetiawanafis@gmail.com, [2]mahendra.data@ub.ac.id, [3]widhi.yahya@ub.ac.id

Abstrak

Sebagian besar server web yang digunakan saat ini masih banyak menggunakan arsitektur *single backend* server. Permasalahan yang muncul adalah bagaimana *single* server tersebut mampu menangani permintaan data yang sangat banyak. Kita harus mempertimbangkan menggunakan *clustering web server* untuk meningkatkan kehandalan server web. Kami membangun klaster ini menggunakan teknologi virtualisasi seperti kontainer virtual. Salah satu virtualisasi berbasis kontainer saat ini adalah Docker. Namun, mengelola beberapa kontainer untuk membuat layanan tunggal adalah tugas yang menantang. Docker memperkenalkan alat pengembangan sistem terdistribusi yang disebut Docker Swarm. Kami mengusulkan mekanisme *loadbalancing* pada Docker Swarm untuk menyeimbangkan beban internal, sehingga dapat mendistribusikan permintaan kepada server web. Selain itu, kami juga mencoba untuk mengetahui kinerja algoritme *roundrobin* dan *leastconn* pada mekanisme *loadbalancing* yang akan digunakan pada Docker Swarm. Hasil pengujian menunjukan *loadbalancing* yang menerapkan algoritme *least connection* memiliki *throughput* 15 Mbps pada 1000 request, 17 Mbps pada 3000 request, 17 Mbps pada 5000 request, sedangkan algoritme *round robin* memiliki *throughput* 15 Mbps pada 1000 request, 14 Mbps pada 3000 request, 15 Mbps pada 5000 request. Hasil menunjukan algoritme *least connection* memiliki kinerja yang lebih baik dari algoritme *round robin.* Selain itu, didapatkan hasil distribusi data secara seimbang pada setiap server web yang tersedia.

Kata kunci: *Docker, Swarm, Server web, Kontainer, Loadbalancer*

Abstract

Most web servers that are used today still use single backend server architectures. The problem that arises is how the single server is able to handle large data requests. We must consider using web server clustering to improve web server reliability. We built this cluster using virtualization technology such as virtual containers. One of the container-based virtualisations currently is Docker. However, managing multiple containers to make a single service is a challenging task. Docker introduces a distributed system development tool called Docker Swarm. We propose a load balancing mechanism on Docker Swarm to balance internal load, so that it can distribute requests to web server. In addition, we also try to find out the performance of roundrobin and leastconn algorithms on loadbalancing mechanisms that will be used on Docker Swarm. The test results show that loadbalancing applying the least connection algorithm has a throughput of 15 Mbps in 1000 requests, 17 Mbps in 3000 requests, 17 Mbps in 5000 requests, while the round robin algorithm has a throughput of 15 Mbps in 1000 requests, 14 Mbps in 3000 requests, 15 Mbps in 5000 requests. The results show the least connection algorithm has better performance than the round robin algorithm. In addition, the results of data distribution are balanced on each available web server.

Keywords: *Docker, Swarm, Web Server, Container, Loadbalancer*

1. PENDAHULUAN

Server web menjadi bagian penting dari infrastruktur internet saat ini. Sebagian besar dari arsitektur server web yang digunakan saat ini, bertujuan untuk meningkatkan kinerja server web hanya dengan menggunakan *single backend* server web. Permasalahan yang muncul adalah bagaimana *single backend* server web mampu menangani lonjakan permintaan data yang

sangat banyak. Itulah sebabnya, membangun infrastruktur server web yang andal dan sangat tersedia sangat penting. Server web tunggal tidak cukup untuk mendukung aplikasi web lalu lintas tinggi. Kita harus mempertimbangkan menggunakan *clustering web server* untuk meningkatkan keandalan dan ketersediaan server web (A. B. M. Moniruzzaman, 2014). Kami dapat membangun klaster ini menggunakan teknologi virtualisasi seperti mesin atau wadah virtual. Dibandingkan dengan mesin virtual, baru-baru ini, virtualisasi berbasis kontainer semakin populer. Salah satu virtualisasi berbasis kontainer yang paling banyak digunakan saat ini adalah Docker. Docker adalah mesin *open source* yang menerapkan secara otomatis aplikasi ke dalam *container* (Turnbull, 2014). Namun, mengelola beberapa kontainer untuk membuat layanan tunggal adalah tugas yang menantang. Docker memperkenalkan alat pengembangan sistem terdistribusi yang disebut Docker Swarm, yang dapat memperluas proses pengembangan perangkat lunak berbasis *container*.

Pada Docker Swarm terdapat dua jenis node, yaitu node manajer, dan node pekerja. Manajer node mengelola keanggotaan dan node pekerja dalam menjalankan layanan swarm di Docker Swarm. Node manager akan mengarahkan permintaan dari luar ke node pekerja dengan menggunakan mekanisme load balancing internal sendiri. Mekanisme penyeimbang beban internal Docker Swarm berfokus pada bagaimana mendistribusikan permintaan kepada node pekerja secara seimbang berdasarkan permintaan pengguna.

Load balancing dapat memaksimalkan *throughput*, mengurangi *latency*, dan memastikan *fault-tolerant*. Fokus utama dari *load balancing* dalam adalah dalam mengalokasikan beban secara dinamis di antara node untuk memenuhi kebutuhan pengguna layanan dan untuk pemanfaatan sumber daya secara maksimum dengan mendistribusikan beban keseluruhan ke beberapa node yang tersedia.

Penelitian ini bertujuan untuk mendistribusikan lalu lintas server web di dalam Docker Swarm dan menerapkan algoritma *Load Balancing* untuk menyeimbangkan beban request secara virtual. Selain itu, peneliti juga membandingakan kinerja algoritme *loadbalancing least connection* dan *round robin*. Harapannya dengan adanya penelitian ini dapat membantu menyelesaikan permasalahan.

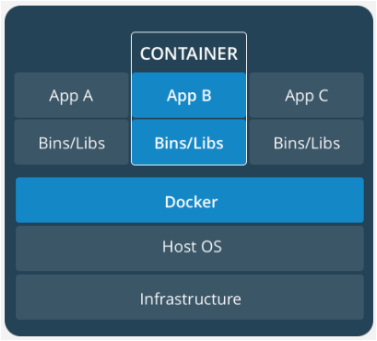
2. LANDASAN KEPUSTAKAAN

2.1 Virtualisasi

Virtualisasi merupakan teknik yang memungkinkan pengguna untuk membuat versi virtual perangkat atau sumber daya, seperti *server*, perangkat penyimpanan, jaringan, atau sistem operasi (Naik, 2016). 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 khusus mereka sendiri, di mana sistem operasi, *libraries*, dan program lain bersifat unik untuk sistem virtualisasi dan tidak terhubung ke sistem operasi *host* yang berada di bawahnya.

2.2 Docker

Docker adalah aplikasi *open source* yang mengotomatisasi penyebaran aplikasi ke dalam *container* (Naik, 2016). Docker merupakan salah satu platform yang dibangun berdasarkan teknologi *container*. *Container image* adalah paket perangkat lunak ringan yang dapat berdiri sendiri, yang mencakup semua yang dibutuhkan untuk menjalankannya: kode, *runtime*, alat sistem, *library* sistem, dan pengaturan. *Container* menggunakan komponen yang diperlukan untuk menjalankan perangkat lunak yang diinginkan. Komponen-komponen ini termasuk *file*, variabel lingkungan, dependensi dan pustaka. Perbedaan yang sangat terlihat dari *container* dibandingkan dengan virtualisasi adalah *container* memiliki ukuran file yang lebih kecil, karena *container* tidak perlu menyiapkan sistem operasi secara penuh.



Gambar 1. *Container*

2.3 Docker Swarm

Docker Swarm adalah *Docker native clustering solution*, yang dapat mengubah sekelompok *host* Docker terdistribusi menjadi

satu *server* virtual besar. *Swarm* terdiri dari beberapa *host* Docker yang berjalan dalam *mode swarm cluster* dan ada bertindak sebagai manajer (untuk mengelola keanggotaan) dan *worker* (yang menjalankan layanan swarm). Salah satu kelebihan utama layanan *swarm* adalah Anda dapat memodifikasi konfigurasi layanan, termasuk jaringan dan volume yang terhubung dengannya, tanpa perlu memulai ulang layanan secara manual.

2.4 *Load Balancing*

Load balancing merupakan teknik yang dapat mendistribusikan beban kerja, jaringan, atau lalu lintas aplikasi merata di beberapa *server* (chen, 2017). *Load balancing* meningkatkan fungsi sistem Anda dengan memungkinkan peningkatan keandalan dan kinerja karena menghilangkan terjadinya satu titik kegagalan.

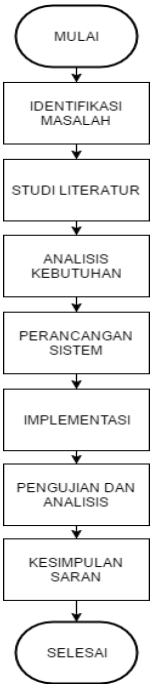
Ada beberapa algoritma yang umum digunakan yaitu *Round Robin* dan *Least Connections*. Algoritme penjadwalan *roundrobin* meneruskan setiap permintaan masuk ke server berikutnya dalam daftar (Mustafa, 2017). Algoritma *leastconn* adalah salah satu algoritma penjadwalan dinamis, karena algoritma *leastconn* perlu menghitung koneksi langsung untuk setiap *server* secara dinamis (Mustafa, 2017).

2.5 NGINX

NGINX merupakan aplikasi *open surce*, *server HTTP* yang berkinerja tinggi dan memiliki fitur *reverse proxy*, serta *server proxy IMAP / POP3*. NGINX dikenal dengan berkinerja tinggi, stabilitas, *set* fitur yang kaya, konfigurasi sederhana, dan konsumsi sumber daya yang rendah. NGINX adalah salah satu dari segelintir *server* yang dibuat untuk mengatasi masalah *C10K*. Masalah *C10k* adalah masalah mengoptimalkan soket jaringan untuk menangani sejumlah besar klien pada saat bersamaan.

3. METODOLOGI

Dalam bab ini akan menguraikan tahapan-tahapan yang akan dilakukan sebagai acuan dalam penulisan skripsi ini. Berikut merupakan alur metodologi yang digunakan:



Gambar 2. Metodologi penelitian

Dalam bab ini akan dijelaskan langkah-langkah yang akan dilakukan dalam penulisan skripsi ini, dari tahap pertama sampai akhir penelitian, yang digambarkan pada gambar 2. Langkah langkah yang akan dilakukan adalah sebagai berikut :

1. Identifikasi masalah

   Permasalahan yang dihadapi dalam penelitian ini adalah bagaimana menerapkan algoritma *load balancing* yang digunakan untuk menyeimbangkan beban request pada docker swarm.
2. Studi literatur

   Studi literatur dilakukan untuk mempelajari teori-teori yang menunjang penelitian, seperti *docker swarm*, dan *load balancing.*
3. Analisis kebutuhan

   Analisa kebutuhan dilakukan untuk mendapatkan komponen komponen yang dibutuhkan untuk penerapan *load balancing* pada *docker swarm*.
4. Perancangan sistem.

   Tahap perancangan vitualisasi ini menggunakan metode *routing mesh*, dimana *node-node* yang digunakan di virtualisasi ini di jadikan satu sebagai *swarm mode*.
5. Implementasi.

   Tahap implementasi dilakukan untuk menerapkan model dari sistem yang telah dirancang pada tahap perancangan menjadi sebuah sistem yang dapat memenuhi kebutuhan penelitian.
6. Pengujian dan analisis

Pengujian sistem dilakukan untuk mencari kesalahan atau error yang ada saat mendistribusikan beban ke beberapa server serta mengetahui apakah mekanisme yang telah diimplementasikan dapat berjalan dengan baik.

7. Kesimpulan dan saran

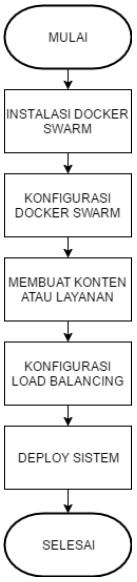
Tahap terakhir adalah memberikan kesimpulan dari hasil analisis yang telah dilakukan, serta dapat menjadi jawaban berdasarkan rumusan masalah yang telah ditentukan. Serta, saran yang berhubungan dengan hasil yang telah dicapai pada penelitian ini sehingga dapat memperbaiki kekurangan dan berguna untuk pengembangan lebih lanjut.

4. PERANCANGAN

Pada bab perancangan ini, akan dijelaskan langkah-langkah untuk merancang sebuah sistem *load-balancing* pada *docker swarm* dengan berpedoman pada metodologi penelitian yang telah dibahas. Sistem ini dibangun dengan menggunakan virtualisasi berbasis kontainer. Terdapat tiga node server *clustering* yang dikelola oleh docker swarm. Kemudian dilakukan *deploying* kontainer server web yang memanfaatkan image milik Nginx.

4.1. Perancangan alur implementasi sistem

Berikut merupakan perancangan sistem yang akan dibuat untuk membuat sebuah sistem *load balancing* server web berbasis kontainer pada Docker Swarm :

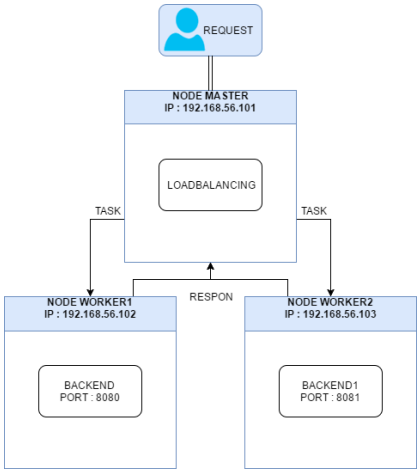


Gambar 3. Alur implementasi sistem

Pada gambar 3 menunjukan tahap-tahap yang akan dilakuakan untuk menerapkan *load balancing* pada Docker swarm. Pembahasan dari masing-masing tahapan akan dijelaskan sebagai sebagai berikut :

1. Download komponen yang dibutuhkan untuk membuat sistem *dockerswarm.* pada masing masing host.
2. Konfigurasi docker swarm dilakukan untuk menggabungkan beberapa node yang akan di jadikan *swarm*.
3. Proses membuat konten dan layanan dilakukan setelah seluruh node yang berada pada *swarm mode* dengan memanfaatkan image Nginx.
4. Proses konfigurasi *loadbalancer* dilakukan setelah layanan atau konten sudah di sebarkan ke dalam node yang tergabung ke dalam *swarm.*
5. *Deploy* sistem dilakukan setelah sistem sudah berjalan dengan benar dan sesuai yang diharapkan, lalu di uji untuk mengetahui jika terjadi error pada sistem.

4.2. Perancangan topologi sistem



Gambar 4. Topologi *load balancing* docker swarm.

Berdasarkan gambar 3, Peneliti menggunakan tiga node yaitu satu node sebagai master dan dua node sebagai *worker*. Node master memiliki alamat ip 192.168.56.101, node worker1 memiliki alamat ip 192.168.56.102, node, worker2 memiliki alamat ip 192.168.56.103. Node master adalah tempat perintah Swarm dijalankan. Swarm menangani penjadwalan, pencarian layanan DNS, penskalaan, dan *load balancing container* pada semua node.

5. IMPLEMENTASI SISTEM

Pada bab ini akan dibahas tentang langkah langkah mengimplementasikan *load balancing*

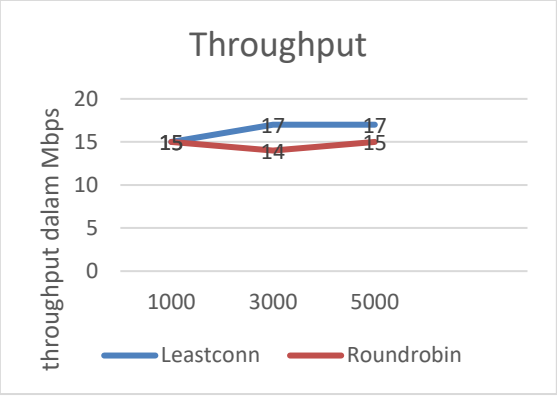
pada docker swarm. Implementasi merupakan proses realisasi dari perancangan sistem yang telah dirancang sebelumnya. Implementasi *load balancer* dilakukan setelah kita membuat *swarm mode*. Kemudian yang harus kita lakukan adalah membuat *container load balancer* untuk mendistribusikan request yang masuk ke *server cluster*. Penulis memanfaatkan *image* dari Nginx yang digunakan sebagai *loadbalancer*.

6. PENGUJIAN DAN ANALISIS

Tahap pengujian dilakukan untuk mengetahui apakah sistem yang sudah dibuat sesuai dengan kebutuhan, serta mengetahui apakah mekanisme yang telah diimplementasikan dapat berjalan dengan baik atau tidak. Ada 3 tahap pengujian yang akan dilakukan, yaitu pengujian performa, pengujian *load balancing*, pengujian fungsionalitas sistem.

6.1 Pengujian *throughput*

Pengujian *throughput* dilakukan untuk membandingkan performa antara *loadbalancer* yang menggunkan metode *leastconn* dan *roundrobin* dalam pengiriman data.



Gambar 5. Hasil pengujian *throughput*

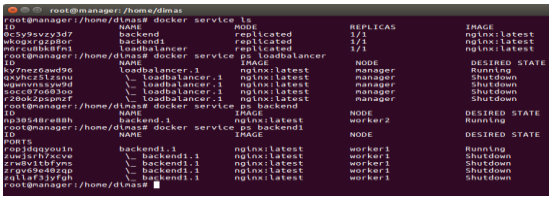
Pada gambar 5 menunjukan hasil yang didapatkan yaitu bahwa *load balancing* yang menggunakan algoritme *leastconn* memiliki *throughput* yang lebih baik dari pada algoritme *round robin.*

6.2 Pengujian *load balancer*

Pengujian *load balancing* dilakukan untuk mengetahui apakah algoritme yang di gunakan untuk *load balancing* berjalan dengan baik atau tidak. Pengujian *load balancing* dilakukan dengan menganalisis paket *request* yang masuk dengan menggunakan wireshark. Berdasarkan hasil yang didapatkan *load balancing* yang menggunakan algoritme *round robin* dapat membagi beban *request* dengan baik, sedangkan algoritme *leastconn* tidak.

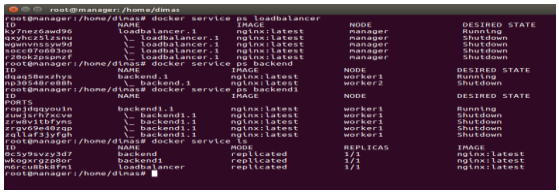
6.2 Pengujian fungsionalitas sistem

Pengujian fungsionalitas sistem dilakukan dengan menguji mekanisme *failover* yang merupakan suatu mekanisme yang ada pada docker swarm. Pengujian ini dilakukan dengan cara memetikan salah satu host pada *swarm cluster.*



Gambar 6. Letak *container* awal

Gambar 6 menunjukan Letak *container* pada masing masing *host* sebelum salah satu *host* di matikan.



Gambar 7. Letak *container* akhir

Gambar 7 menunjukan Letak *container* setelah salah *host* worker2 di matikan. Dapat dilihat bahwa *service* backend yang tadinya berada pada *host* worker2 berpindah ke *host* worker1. Hal ini menunjukan bahwa mekanisme *failover* bekerja dengan baik.

7. KESIMPULAN

1. Berdasarkan konsep perancangan, implementasi *load balancing* menggunakan algoritma *least connection* dan *round robin* pada docker swarm telah berhasil dilakukan. Selain itu, mekanisme *failover* yang terdapat pada docker swarm juga berjalan dengan baik, sehingga sistem memiliki ketersediaan tingkat tinggi.
2. Hasil throughput yang didapatkan dari *load balancer* yang menggunakan algoritme *least connection* dan *round robin* memiliki hasil yang berbeda. *Load balancing* yang menerapkan algoritme *leastconn* memberikan hasil *throughput* yang lebih baik dari pada algoritme *roundrobin*.

8. DAFTAR PUSTAKA

A. B. M. Moniruzzaman, M. W. (2014). A High Availability Clusters Model Combined with Load Balancing and Shared Storage

Technologies for Web Servers. *International Journal of Grid Distribution Computing*.

Affan, I. (2018, March 20). *Perkembangan Infrastruktur Saat ini*. Diambil kembali dari Mico Citra Utama: https://cv-mcu.co.id/2018/03/20/perkembangan-infra-saat-ini.html

B. Kostadinov, M. J. (2017). Cost-effective website failover through a CDN network and asynchronous replication. *IEEE EUROCON 2017 -17th International Conference on Smart Technologies*, 151-156.

chen, h. (2017, april 17). *IBM* . Diambil kembali dari IBM Blog: www.ibm.com

Chen, S.-L., Chen, Y.-Y., & Kuo, S.-H. (2016). CLB: A novel load balancing architecture and algorithm for cloud service. *Computers & Electrical Engineering (2017)*.

Constantinov, C., Potreas, C. M., & Mocanu, M. L. (2016). Performing real-time social recommendations on a highly-available graph database cluster. *journal of IEEE*, 116-121.

Docker. (2018). *Docker Web site*. Diambil kembali dari Docker Web site: https://docs.docker.com/engine/swarm/key-concepts/

docker. (2018). *what-container*. Diambil kembali dari docker.com: https://www.docker.com/what-container

eMarketer. (2018). *Pengguna Internet Indonesia Nomor Enam Dunia*. Diambil kembali dari Kementrian Komunikasi dan Informatika: https://kominfo.go.id/content/detail/4286/pengguna-internet-indonesia-nomor-enam-dunia/0/sorotan_media

Gopinath P P, G., & Vasudevan, S. K. (2015). An in-depth analysis and study of Load balancing techniques in the cloud computing environment.

Mustafa, D. M. (2017). Load balancing algorithms round-robin (rr), leastconnection, and least loaded efficiency. *GESJ: Computer Science and Telecommunications 2017|No.1(51)* .

Naik, N. (2016). Building A Virtual System of Systems Using Docker Swarm in Multiple Clouds.

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*.

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