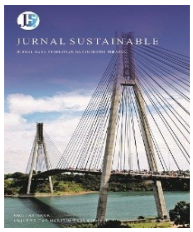


Jurnal Sustainable: Jurnal Hasil Penelitian dan Industri Terapan
Vol. 11, No. 01, hal. 16- 25, Mei 2022
Jurnal Sustainable: Jurnal Hasil Penelitian dan Industri Terapan
ISSN 2615-6334 (Online)
ISSN 2087-5347 (Print)

Analisis Perbandingan Performa *Load Balancer Nginx* Dan *Haproxy Pada Docker*

Fernando Mardi Nurzaman[1], Ferdi Chahyadi[2]*, Muhamad Radzi Rathomi[3],
[1,2,3]Jurusan Teknik Informatika, Fakultas Teknik, Universitas Maritim Raja Ali Haji
[1,2]Jl. Politeknik Senggarang, Tanjungpinang 29100
*Corresponding Author: ferdi.chahyadi@umrah.ac.id

Abstract— The development of computer and network technology is so rapid, along with the increasing demand for information needs on the internet, the traffic load for Webserver data is increasing, due to too many requests. To implement clustering on the Webserver, use the loadbalancing method so that the traffic load for each Webserver can run optimally. The Round Robin Algorithm divides each process equally to the Webserver. Implementation of Nginx and Haporxy Load Balancers is carried out by container virtualization on Docker. Docker is an application to manage resources on the server and make it easier to deploy applications. The test results show that the Haporxy Load Balancer on Docker using the Round Robin algorithm is better than other Load Balancers. It is evident from the performance test of the average Throughput value obtained 8418.6 KB/s with an average response obtained of 2.2 ms with an average CPU usage of 42.0% and an average memory usage of 0.4% of Load Haporxy Balancer in Docker is more stable than other Load Balancers. On tight (crowded) connections and the availability of large requests it can be done with the Haporxy Load Balancer on Docker without any errors, while on the Nginx Load Balancer on Docker there is an average error of 739.8 connection time out.

Keywords— *Load Balancer*, *Nginx*, *Haporxy Docker*

Intisari— Perkembangan teknologi komputer dan jaringan begitu pesat, seiring dengan itu meningkatnya permintaan akan kebutuhan informasi dalam internet menyebabkan jalannya beban *traffic* data *Webserver* semakin banyak akibat terlalu banyak *request*. Untuk mengimplemtasikan *clustering* pada *webserver* menggunakan metode *loadbalancing* agar beban *traffic* setiap masing-masing *Webserver* dapat berjalan dengan optimal. Algoritma *Round Robin* membagi rata setiap proses kepada *Webserver*. Implementasi *Load Balancer Nginx* dan *Haporxy* dilakukan secara *virtualisasi container* pada *Docker*. *Docker* merupaka aplikasi untuk *memanagement resource* pada *server* dan memudahkan saat mendeploy aplikasi. Hasil pengujian menunjukan bahwa *Load Balancer Haporxy* pada *Docker* dengan menggunakan algoritma *Round Robin* lebih baik dibandingkan dengan *Load Balancer* lainnya. Terbukti dari pengujian performa dari nilai rata-rata *Throughput* yang didapatkan 8418,6 KB/s dengan response rata-rata yang didapatkan sebesar 2.2 *ms* dengan rata-rata penggunaan *CPU* 42,0 % dan rata-rata penggunaan memory 0,4 % dari *Load Balancer Haporxy* pada *Docker* lebih stabil dibandingkan dengan *Load Balancer* lainnya. Pada koneksi yang pandat (ramai) serta ketersedian permintaan yang besar dapat di kerjakan dengan *Load Balancer Haporxy* pada *Docker* tanpa adanya *error* sedangkan di *Load Balancer Nginx* pada *Docker* terdapat rata-rata error sebesar 739,8 *connection time out*.

Kata kunci— *Load Balancer, Nginx, Haporxy, Docker*

ISSN 2615-6334 (Online)
ISSN 2087-5347 (Print)

I. PENDAHULUAN

Perkembangan teknologi komputer dan jaringan begitu pesat, seiring dengan itu meningkatnya permintaanakan kebutuhan informasi dalam internet menyebabkan jalanya beban traffic pada *webserver* semakin besar, akibat terlalu banyaknya request. Dengan demikian menyebabkan beban kerja pada suatu layanan *webserver* mengalami down (overload). Untuk menanggulangi permasalahan beban yang berlebihan dalam suatu server maka diterapkan konsep clustering yaitu menggambungkan beberapa server untuk bekerja secara bersamaan tetapi seperti sistem tunggal. Salah satu metode clustering yaitu *loadbalancing.*

Load balancing merupakan teknologi pembagian beban kepada beberapa *server,* memastikan tidak terjadinya kelebihan beban pada salah satu *webserver. Load balancing* bertujuan untuk memaksimalkan *Throughput*, memperkecil waktu tanggap (*Response Time*) dan menghindari overload pada salah satu server [1]. Dalam menerapkan *loadbalancing* ini menggunakan aplikasi proxy *opensource* yaitu *Nginx* dan *Haproxy*, tidak hanya *opensource* melainkan aplikasi yang disebutkan memiliki fitur-fitur yang berbeda dalam menangani load balancing dengan unggulan masing-masing.

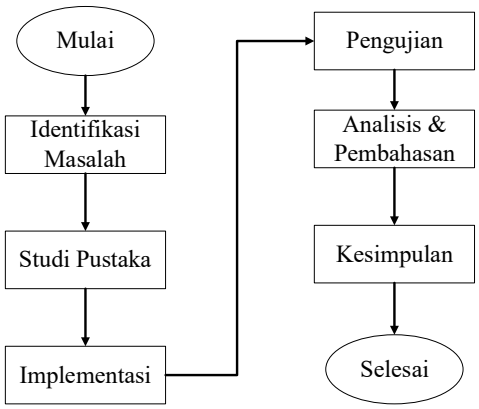
Nginx adalah *software webserver* yang *opensource*, ketika pertama kali dirilis hanya berfungsi sebagai HTTP *webserver* saja. Namun sekerang berperan sebagai reserver *proxy*, HTTP *Load Balancer* dan email proxy untuk IMAP, POP3 dan SMTP. *Nginx* merupakan *Webserver* yang ringan dan memiliki performa yang cepat, *Nginx* mampu menerima banyak *traffic* dengan menggunakan *Load Balancer Nginx* [2]. Begitu juga dalam menerapkan *Load Balancer* dengan menggunakan *Haporxy*. *Haporxy* adalah produk *opensource* yang digunakan untuk menciptakan sistem *load balancing* dan *Failover* dari aplikasi yang berbasis *FTP* dan *HTTP* dan bisa menjadi penghubung antara *client* dan server-server *multi service* serta mendistribusikan *traffic* [3]. Perangkat lunak ini sangat cocok digunakan untuk website yang *traffic* hariannya tinggi. *Haporxy* sangat diperlukan aplikasi menurut SLA yang cukup ketat dan tidak mentolerir adanya *downtime*. Selain itu *Haporxy* memungkinkan adanya *backup* dari *Load Balancer* yang menjadi *backup* dalam singkat secara otomatis akan menggantikan *Load Balancer* yang mati tersebut. dalam penerapan *Load Balancer* tersebut dilakukan dengan menggunakan *virtualisasi container* yang ada pada *Docker*.

Docker merupakan *software* yang bersifat *opensource* yang dibuat untuk memudahkan deploy aplikasi, serta pengembangan aplikasi dengan menggunakan *containerization*. *Docker* juga memiliki mekanisme *failover* dimana jika salah satu *node*/*host* mengalami kegagalan maka akan langsung ditangani oleh *node/host* yang aktif [4]. *Container* merupakan unit standar dari *software* yang memaketkan kode dan semua depensinya, agar aplikasi dapat berjalan dengan cepat dan terpercaya melalui suatu lingkungan computing menunju ke lingkungan yang lainnya [5]. dalam mengimplementasikan *Load Balancer* maka membutuhkan beberapa *Docker*-mechine untuk menjalankan *Load Balancer* pada *Docker*. Untuk membangun cluster web dapat menggunakan teknologi *virtualisai* seperti virtual mesin atau container dengan menggunakan *Docker Swarm* [6]. *Docker Swarm* dapat menghubungkan container pada beberapa perangkat *server* yang berbeda sehingga dapat mempermudah proses perawatan dan deployment suatu website.

Berdasarkan uraian diatas, penulis memulai penelitian dengan judul "Analisis Perbandingan Performa *Nginx* dan *Haporxy* pada *Docker*". Besar harapan penulis terhadap penelitian ini agar dapat bermanfaat khususnya bagi penulis sendiri dan penelitian penelitian selanjutnya.

II. METODE PENELITIAN

Dalam bab ini akan menjelaskan tahapan-tahapan yang akan dilakukan sebagai acuan dalam penulisan penelitian ini. Berikut ini merupakan alur metodologi yang digunakan pada penelitian ini.



Gambar 1. Alur metode penelitian

A. Indetifikasi masalah

Permasalahan yang ada pada penelitian ini adalah performa manakah yang baik anatara *Nginx* dah *Haproxy*, dalam menangani *Load Balancer* dengan menggunakan *Docker mechine* saling terhubung dengan menggunakan *Docker Swarm*.

B. Studi pustaka

Studi pustaka dilakuakn untuk mempelajari teori-teori yang menunjang penelitian yang berasal dari jurnal-jurnal penelitian & buku-buku yang berkaita dengan *Nginx, Haproxy, Load Balancer, dan Docker.*

C. Implemetasi

Implementasi dilakukan dengan *virtual private server (VPS)* dan *Docker Swarm* sebagai penghubung antar *Docker-mechine,* dengan menggunakan *Nginx* dan *Haproxy* sebagai *Load Balancer.*

D. Pengujian

Pada penelitian ini dilakukan dengan menguji performa *Load Balancer Nginx* dan *Haproxy* pada *Docker* dengan menggunakan *webstrees tools* yaitu *Httperf*. Serta membandingkan *Load Balancer Nginx* dan *Haproxy* pada komputer *konvesional*, dengan parameter yang dilihat yaitu *Throughput*, *Response Time*, *CPU Usage* dan *Memory Usage*.

E. Analisis & Pembahasan

Analisis yang dilakukan hanya pengujian performa yang dilihat dari hasil pengujian parameter *Throughput*, *Response Time*, *CPU Usage* dan *Memory Usage* pada masing-masing *Load Balancer*

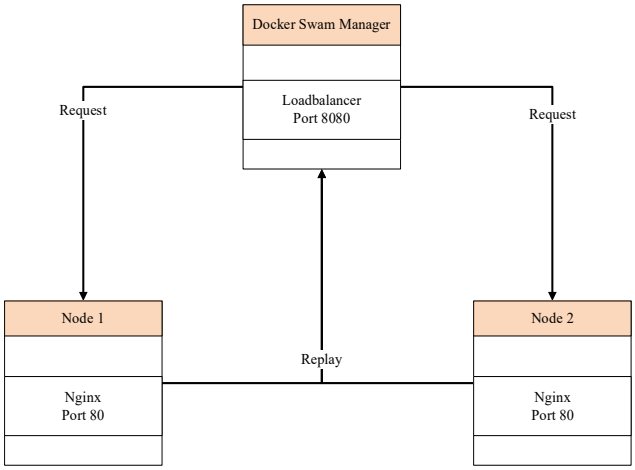
F. Kesimpulan

Kesimpulan merupakan tahap terakhir yang memberikan kesimpulan dari hasil analisis & pembahasan yang telah dilakukan, serta menjadi rumusan masalah yang telah didapat pada penelitian ini. Serta saran yang berhubungan dari hasil analisis & pembahasan, sehingga dapat memperbaiki kekurangan dan berguna untuk pengembangan lebih lanjut.

III. PERANCANGAN

Pada bab ini mengambarkan rancangan arsitektur *Load Balancer* yang akan dibangunkan yaitu *Load Balancer Nginx* dan *Haproxy* pada *Docker* dan racangan *Load Balancer Nginx* dan *Haproxy* pada komputer *konvesional* sebagai perbandingan antara penerapan *Load Balancer* yang dibangun dengan menggunakan *virtualisasi* container dengan yang tidak menggunakan *Docker*/ *virtualisasi* container.

A. Arsitektur Load Balancer pada Docker

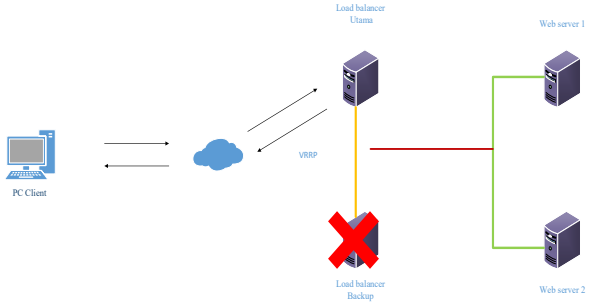


Gambar 2. Arsitektur *Load Balancer* dengan *Docker Swarm*

Pada Gambar 2. Arsitektur *Load Balancer* pada *Docker* menggunakan tiga node server *clustering* yang dikelola oleh *Docker Swarm Manager*. *Docker Swarm Manager* juga menjadi tempat *virtualisasi* untuk *Load Balancer Nginx* dan *Haproxy*, sedangkan pada node 1 dan node 2 bekerja sebagai *webserver* dengan menggunakan *Nginx webserver*.

ISSN 2615-6334 (Online)
ISSN 2087-5347 (Print)

B. Arsitektur *Load Balancer* pada komputer konvesional



Gambar 3. Arsitektur *Load Balancer* dengan komputer konvesional

Pada Gambar 3. Arsitektur *Load Balancer* pada komputer *konvesional* menggunakan empat node server *clustering*, *Load Balancer Nginx* dan *Haproxy*, dengan komputer *konvesional* membagikan beban yang diterima ke masing-masing *Nginx webserver*.

IV. IMPLEMENTASI

Pada bab ini akan membahas tentang implementasi *Load Balancer Nginx* dan *Haproxy* pada *Docker* dengan menggunakan *Docker Compose*.

```
upstream backend {
    server 172.10.10.10:80
    server 172.10.10.11:80
}

server {
    listen 8080;

    location / {
     proxy_redirect      off;
     proxy_set_header    X-Real-IP
$remote_addr;
     proxy_set_header    X-Forwarded-For
$proxy_add_x_forwarded_for;
     proxy_set_header    Host $http_host;
      proxy_pass http://backend;
    }
}
```

Gambar 4. konfigurasi *Load Balancer Nginx* pada *Docker*

Pada gambar 4 merupakan konfigurasi *Load Balancer Nginx* yang dimasukkan pada konfigurasi *Dockerfile*, dari *Dockerfile* dibuat menjadi *images* setelah itu images *Load Balancer Nginx* digunakan pada konfigurasi *Docker-compose.yml* seperti pada gambar 5.

```
version: "3.8"
services:
  react-app:
    image: nando2302/tangkasNginx:1.0
    ports:
      - target: 80
        published: 80
        protocol: tcp
        # mode: host
    deploy:
      mode: replicated
      replicas: 2
      update_config:
        parallelism: 1
        delay: 10s
        failure_action: rollback
        order: start-first
      restart_policy:
        condition: on-failure
      placement:
        constraints: [node.role ==
worker]
  Load Balancer:
    image: nando2302/lb-react:1.0
    ports:
      - target: 8080
        published: 8080
        protocol: http
    deploy:
      # mode: global
      update_config:
        parallelism: 0
        delay: 10s
        failure_action: rollback
      restart_policy:
        condition: on-failure
      placement:
        constraints: [node.role ==
manager]
```

Gambar 5. Konfigurasi *Docker-compose Nginx Load Balancer*

Untuk mendeploy sistem *Load Balancer Nginx* pada *Docker* menggunakan *Docker stack* yang dijalankan pada *Docker Swarm manager*, dan *Docker Swarm Manager* juga yang akan membagikan *images webserver* menjadi container-container dan dibagikan pada masing masing node yang ada dengan *replicated* yang sudah disesuaikan pada konfig.

```
global

defaults
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms

frontend public
    bind *:8080
    default_backend apps

backend apps
 balance roundrobin
 mode http
 server worker-1 172.104.176.176:80 check
 server worker-2 139.162.8.11:80 check
```

Gambar 6. Konfigurasi Load Balancer Haproxy pada Docker

Pada gambar 6 merupakan konfigurasi *Load Balancer Haproxy* yang dimasukkan pada konfigurasi *Dockerfile*, dari *Dockerfile* dibuat menjadi *images* setelah itu images *Load Balancer Haproxy* digunakan dalam konfigurasi *Docker-compose.yml* seperti pada gambar 7.

```
version: "3.8"
services:
  react-app:
    image: nando2302/tangkasNginx:1.0
    ports:
      - target: 80
        published: 80
        protocol: tcp
        # mode: host
    deploy:
      mode: replicated
      replicas: 2
      update_config:
        parallelism: 1
        delay: 10s
        failure_action: rollback
        order: start-first
      restart_policy:
        condition: on-failure
      placement:
        constraints: [node.role ==
worker]
  Load Balancer:
    image: nando2302/lb-react:1.0
    ports:
      - target: 8080
        published: 8080
        protocol: http
    deploy:
      # mode: global
      update_config:
        parallelism: 0
        delay: 10s
        failure_action: rollback
      restart_policy:
        condition: on-failure
      placement:
        constraints: [node.role ==
manager]
```

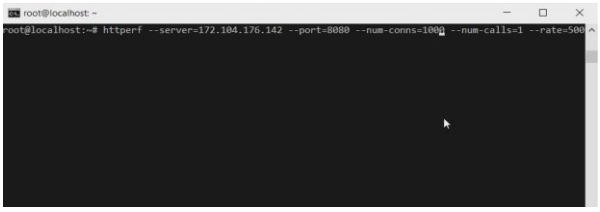
Gambar 7. Konfigurasi *Docker-compose Haproxy Load Balancer*

Untuk mendeploy sistem *Load Balancer Haproxy* pada *Docker* menggunakan *Docker stack* yang dijalankan pada *Docker Swarm manager*, dan *Docker Swarm Manager* juga yang akan membagikan images *webserver Nginx* menjadi container-container dan dibagikan pada masing-masing node yang ada dengan *replicated* yang sudah disesuaikan.

V. PENGUJIAN

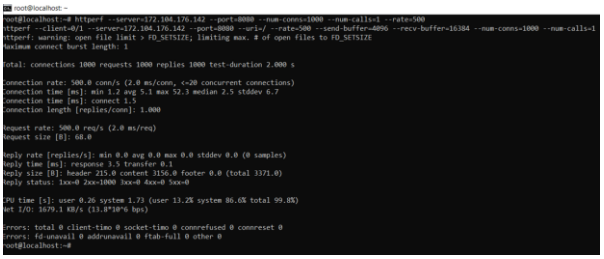
Tahap pengujian dilakukan untuk menguji performa dari masing-masing *Load Balancer*, dengan menggunakan parameter-paramater *Throughput, Response Time, CPU Usage* dan *Memory Usage.* Dengan menggunakan *webstress tools* yaitu dengan *httperf* dengan koneksi seperti koneksi 1000 dengan *calls*s 1,3,5 dan 10 dan *rate* 100 & 500, koneksi 3000 dengan *calls*s 1,3,5 dan 10 dan *rate* 100 & 500, koneksi 5000 dengan *calls*s 1,3,5 dan 10 dan *rate* 100 & 500, dan koneksi 10000 dengan *calls*s 1,3,5 dan 10 dan *rate* 100 & 500.

A. Pengujian performa dengan httperf



Gambar 8. Pengujian *Load Balancer* dengan koneksi yang sudah ditentukan

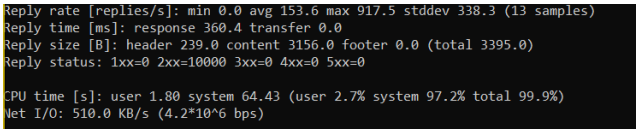
Gambar 8 menampilkan cara pengujian dengan menggunkan htttperf dengan memberikan koneksi, *calls*s dan *rate*, yang telah di tentukan. Setelah pengujian selesai maka akan memberikan hasil seperti pada gambar 9.



Gambar 9. Hasil pengujian *Load Balancer* dengan koneksi yang sudah ditentukan

B. Pengujian hanya dengan menggunakan 1 webserver/node

Pengujian ini dilakukan untuk mengujia apakah layanan yang ada pada *Load Balancer* masih bisa diakses dan apakah fungsional *Load Balancer* masih bekerja hanya dengan satu node. Terdapat pada gambar-gambar berikut ini:

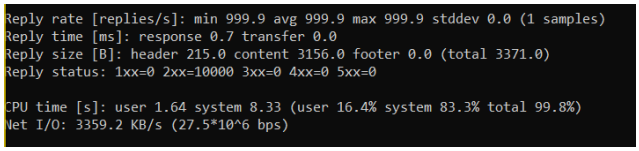


Gambar 10. Pengujian *Load Balancer Nginx* pada *Docker* dengan 1 node

Hasil pengujian menggunakan satu node server pada *Load Balancer Nginx* pada *Docker* didapatkan bahwa layanan masih dapat diakses

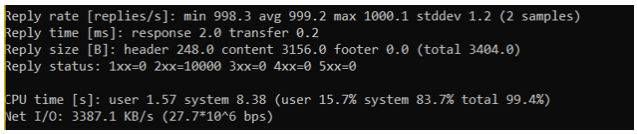
ISSN 2615-6334 (Online)
ISSN 2087-5347 (Print)

tetapi dengan *response* yang sangat lambat dan fungsional *Load Balancer* tidak dapat berkerja karena *webserver* yang tersedia hanya satu *webserver*.



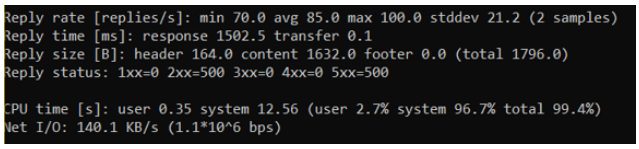
Gambar 11. Pengujian *Load Balancer Haproxy* pada *Docker* dengan 1 node

Hasil pengujian menggunakan satu node server pada *Load Balancer Haproxy* pada *Docker* didapatkan bahwa layanan masih dapat diakses dengan *response* yang diberikan sangat cepat tetapi untuk fungsional *Load Balancer* tidak dapat berkerja karena hanya menggunakan satu *webserver*.



Gambar 12. Pengujian *Load Balancer Nginx* pada komputer dengan 1 node

Hasil pengujian menggunakan satu node server pada *Load Balancer Nginx* pada komputer *konvesional* didapatkan bahwa layanan masih dapat diakses dengan *response* yang diberikan sangat cepat tetapi fungsional *Load Balancer* tidak dapat bekerja karena *webserver* yang tersedia hanya satu *webserver*.



Gambar 13. Pengujian *Load Balancer Haproxy* pada Komputer dengan 1 node

Hasil pengujian menggunakan satu node server pada *Load Balancer Haproxy* pada komputer *konvesional* didapatkan bahwa layanan masih dapat diakses dengan *response* yang diberikan sangat lambat tetapi fungsional *Load Balancer* tidak dapat berkerja karena *webserver* yang tersedia hanya satu *webserver*.

VI. ANALISIS & PERBANDINGAN

Pada bab ini merupakan analisis data yang didapatkan dari pengujian dengan menggunakan *tools httperf* dengan parameter-parameter yang sudah tentukan. Dari data tersebut dibuatkan menjadi rata-rata untuk dibuatkan menjadi grafik perbandingan dengan berdasarkan koneksi dan *rate*.

A. Analisis Hasil Pengujian

1) Analisis parameter Throughput: *Analisis parameter thorughput* terbagi menjadi 2 bagian yaitu pada rate 100 dan 500 disajikan pada tabel 1 dan tabel 2.

Tabel 1. Rata-rata Pengujian Throughput dengan rate 100

Rata Rata Data *Throughput* dengan *rate* 100 (Kb/s)				
Koneksi	*Load Balancer Nginx Docker*	*Load Balancer Haporxy Docker*	*Load Balancer Nginx Non-Docker*	*Load Balancer Haporxy Non-Docker*
1000	1606,2 Kb/s	1594,8 Kb/s	1609,6 Kb/s	1598,9 Kb/s
3000	1606,3 Kb/s	1595,2 Kb/s	1610,3 Kb/s	1599,2 Kb/s
5000	1606,3 Kb/s	1595,1 Kb/s	1610,4 Kb/s	1599,3 Kb/s
10000	1606,3 Kb/s	1595,2 Kb/s	1610,4 Kb/s	1599,4 Kb/s
Rata-rata	1606,3 Kb/s	1595,1 Kb/s	1610,2 Kb/s	1599,2 Kb/s

Data dari tabel 1. Dapat disimpulkan bahwa rata-rata nilai *Throughput* tertinggi terdapat pada *Load Balancer Nginx* pada *Non – Docker* sebesar 1610,2 Kb/s. sedangkan nilai rata-rata terendah terdapat pada *Load Balancer Haproxy* pada *Docker* sebesar 15,95,1 Kb/s.

Tabel 2. Rata-rata pengujian Throughput dengan rate 500

Rata Rata Data *Throughput* dengan *rate* 500 (KB/s)				
Koneksi	*Load Balancer Nginx Docker*	*Load Balancer Haporxy Docker*	*Load Balancer Nginx Non-Docker*	*Load Balancer Haporxy Non-Docker*
1000	6750,3 Kb/s	7916,9 Kb/s	5710,9 Kb/s	5666,9 Kb/s
3000	6592,8 Kb/s	7962,2 Kb/s	5467,0 Kb/s	5692,5 Kb/s
5000	5780,8 Kb/s	7966,6 Kb/s	5361,7 Kb/s	5589,4 Kb/s
10000	5672,2 Kb/s	9828,6 Kb/s	5154,2 Kb/s	5716,5 Kb/s
Rata-rata	6199,0 Kb/s	8418,6 Kb/s	5423,5 Kb/s	5666,3 Kb/s

Dari tabel 2 dapat simpulkan bahwa rata-rata nilai *Throughput* tertinggi terdapat pada *Load Balancer Haproxy* pada *Docker* sebesar 8418 Kb/s. Sedangkan rata-rata nilai terendah terdapat pada *Load Balancer Nginx* pada *Non – Docker* sebesar 5423,5 Kb/s.

2) Analisis parameter Response Time: *Analisis parameter Response Time* dibagi menjadi 2 bagian yaitu pada rate 100 dan rate 500. Disajikan pada tabel 3 dan tabel 4.

Tabel 3. Rata-rata pengujian *Response Time* dengan *rate* 100

Rata-rata Hasil Pengujian *Response Time* dengan *Rate* 100 (*ms*)				
Koneksi	*Load Balancer Nginx Docker*	*Load Balancer Haporxy Docker*	*Load Balancer Nginx Non-Docker*	*Load Balancer Haporxy Non-Docker*
1000	1,5 *ms*	1,6 *ms*	1,8 *ms*	1,7 *ms*
3000	1,3 *ms*	1,5 *ms*	1,8 *ms*	1,7 *ms*
5000	1,4 *ms*	1,5 *ms*	1,8 *ms*	1,8 *ms*
10000	1,4 *ms*	1,5 *ms*	1,7 *ms*	1,8 *ms*
Rata-rata	1,4 *ms*	1,5 *ms*	1,8 *ms*	1,7 *ms*

Dari tabel 3. Dapat disimpulkan bahwa nilai rata-rata *Response Time* terendah terdapat pada *Load Balancer Nginx* pada *Docker* sebesar 1,4 *ms*. Sedangkan nilai rata-rata *Response Time* tertinggi terdapat pada *Load Balancer Nginx* pada *Non – Docker* sebesar 1,8 *ms*.

Tabel 4. Rata-rata pengujian *Response Time* dengan *rate* 500

Rata-rata Hasil Pengujian *Response Time* dengan *Rate* 500 (*ms*)				
Koneksi	*Load Balancer Nginx Docker*	*Load Balancer Haporxy Docker*	*Load Balancer Nginx Non-Docker*	*Load Balancer Haporxy Non-Docker*
1000	13,7 *ms*	2,4 *ms*	34,6 *ms*	29,3 *ms*
3000	28,8 *ms*	1,7 *ms*	50,3 *ms*	32,0 *ms*
5000	41,6 *ms*	2,4 *ms*	53,6 *ms*	35,3 *ms*
10000	35,1 *ms*	2,2 *ms*	57,0 *ms*	43,0 *ms*
Rata-rata	29,8 *ms*	2,2 *ms*	48,9 *ms*	34,9 *ms*

Dari tabel 4 dapat disimpulkan bahwa nilai rata-rata *Response Time* terendah terdapat pada *Load Balancer Haproxy* pada *Docker* sebesar 2,2 *ms*. Sedangkan nilai rata-rata tertinggi terdapat pada *Load Balancer Nginx Non - Docker* sebesar 48,9 *ms*.

3) Analisi parameter CPU Usage : *Analisis parameter CPU Usage* dibagikan menjadi 2 bagian yaitu dengan rate 100 dan rate 500. Disajikan pada tabel 5 dan tabel 6.

Tabel 5. Rata-rata pengujian *CPU Usage* dengan *rate* 100

Rata-rata hasil pegujian *CPU Usage* dengan *rate* 100 (%)				
Koneksi	*Load Balancer Nginx Docker*	*Load Balancer Haporxy Docker*	*Load Balancer Nginx Non-Docker*	*Load Balancer Haporxy Non-Docker*
1000	12,3%	10,4%	13,0%	13,3%
3000	13,5%	11,4%	13,3%	13,3%
5000	13,2%	11,1%	13,0%	14,6%
10000	14,0%	11,2%	13,65	15,0%
Rata-rata	13,2%	11,0%	13,2%	14,1%

Dari tabel 5 bisa disimpulkan bahwa nilai rata-rata *CPU Usage* terendah terdapat pada *Load Balancer Haproxy* pada *Docker* sebesar 11,0%. Sedangkan nilai rata-rata tertinggi terdapat pada *Load Balancer Haproxy Non-Docker* sebesar 14,1%.

Tabel 6. Rata-rata pengujian *CPU Usage* dengan *rate* 500

Rata-rata hasil pegujian *CPU Usage* dengan *rate* 500 (%)				
Koneksi	*Load Balancer Nginx Docker*	*Load Balancer Haporxy Docker*	*Load Balancer Nginx Non-Docker*	*Load Balancer Haporxy Non-Docker*
1000	43,0%	38,1%	24,9%	33,9%
3000	59,4%	43,9%	31,8%	36,4%
5000	58,5%	45,2%	29,5%	39,0%
10000	69,7%	40,7%	29,8%	37,9%
Rata-rata	57,6%	42,0%	29,0%	36,8%

Dari tabel 6 dapat disimpulkan bahwa nilai rata-rata *CPU Usage* tertinggi terdapat pada *Load Balancer Nginx* pada *Docker* sebesar 57,6%. Sedangkan nilai rata-rata terendah terdapat pada *Load Balancer Nginx Non – Docker* sebesar 29,0%.

4) Analisis parameter Memory Usage: *Analisis parameter Memory Usage* terbagi menjadi 2 bagian yaitu pada rate 100 & 500. Disajikan pada tabel 7 dan tabel 8.

Tabel 7. Rata-rata pengujian Memory Usage dengan rate 100

Rata-rata Penggunaan Memory dengan *Rate* 100 (%)				
Koneksi	*Load Balancer Nginx Docker*	*Load Balancer Haporxy Docker*	*Load Balancer Nginx Non-Docker*	*Load Balancer Haporxy Non-Docker*
1000	0,2%	0,4%	0,4%	0%
3000	0,2%	0,4%	0,4%	0%
5000	0,2%	0,4%	0,4%	0%
10000	0,2%	0,4%	0,4%	0%
Rata-	0,2%	0,4%	0,4%	0%

ISSN 2615-6334 (Online)
ISSN 2087-5347 (Print)

rata

Dari tabel 7 dapat disimpulkan bahwa nilai rata-rata *Memory Usage* tertinggi terdapat pada *Load Balancer Haproxy* pada *Docker* & *Load Balancer Nginx* pada *Non – Docker* sebesar 0,4%. Sedangkan nilai rata-rata terendah terdapat pada *Load Balancer Haproxy Non – Docker* sebesar 0%.

Tabel 8. Rata-rata pengujian *Memory Usage* dengan *rate* 500

Rata-rata Penggunaan Memory dengan *Rate* 500 (%)				
Koneksi	*Load Balancer Nginx Docker*	*Load Balancer Haporxy Docker*	*Load Balancer Nginx Non-Docker*	*Load Balancer Haporxy Non-Docker*
1000	0,2%	0,4%	0,4%	0,1%
3000	0,2%	0,4%	0,4%	0,1%
5000	0,2%	0,4%	0,4%	0,1%
10000	0,2%	0,4%	0,4%	0,1%
Rata-rata	0,2%	0,4%	0,4%	0,1%

Dari tabel 8 dapat disimpulkan bahwa nilai rata-rata *Memory Usage* tertinggi terdapat pada *Load Balancer Haproxy* pada *Docker* & *Load Balancer Nginx* pada *Non – Docker* sebesar 0,4%. Sedangkan nilai rata-rata terendah terdapat pada *Load Balancer Haproxy Non – Docker* sebesar 0,1%.

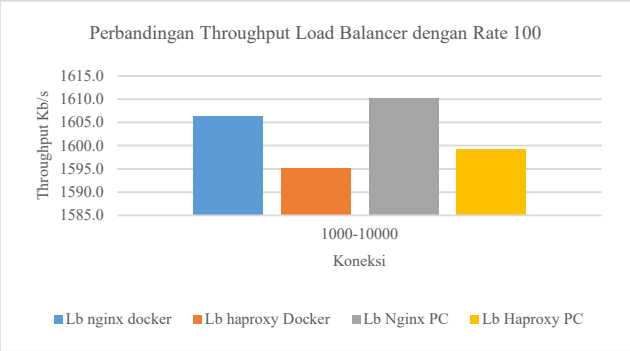
5) Hasil pengujian dengan mengggunakan 1 node webserver: *Hasil dari pengujian dengan satu webserver* pada masing-masing Load Balancer disajikan pada tabel 9 dimana untuk mengakses layanan masih berfungsi dengan semestinya tetapi untuk teknik Load Balancer tidak berfungsi semestinya.

Tabel 9. hasil pengujian dengan 1 node server

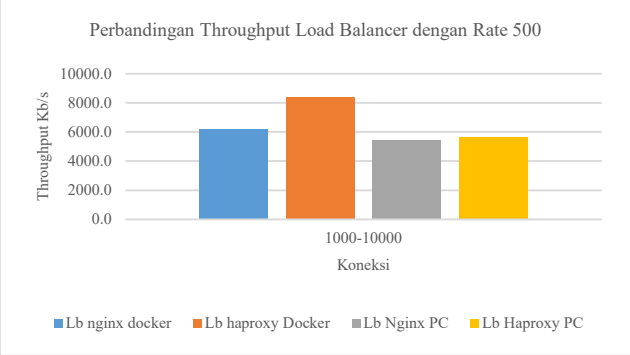
Pengujian	***Load Balancer* dengan satu *Webserver***	**Mengakses layanan pada *Load Balancer***
Load Balancer Nginx* pada *Docker	Tidak Bekerja	Bekerja
Load Balancer Haporxy* pada *Docker	Tidak Bekerja	Bekerja
***Load Balancer Nginx* pada Non – Docker**	Tidak Bekerja	Bekerja
***Load Balancer Haporxy* pada Non – Docker**	Tidak Bekerja	Bekerja

B. Perbandingan

1) Parameter Throughput : *Perbandingan* parameter Throughput

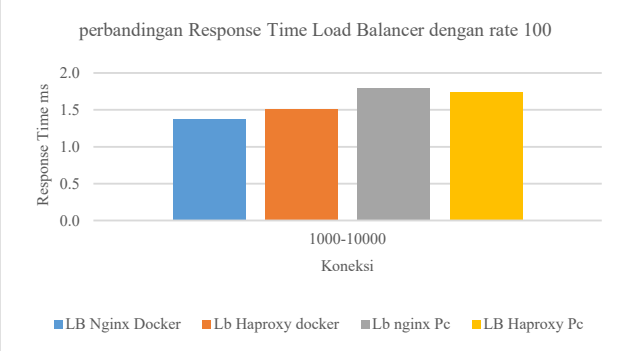


Gambar 14. Perbandingan Throughput dengan rate 100

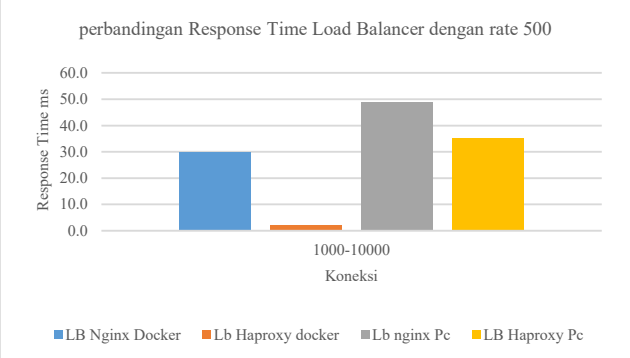


Gambar 15. Perbandingan Throughput dengan rate 500

2) Parameter Response Time : *Perbandingan* parameter Response Time

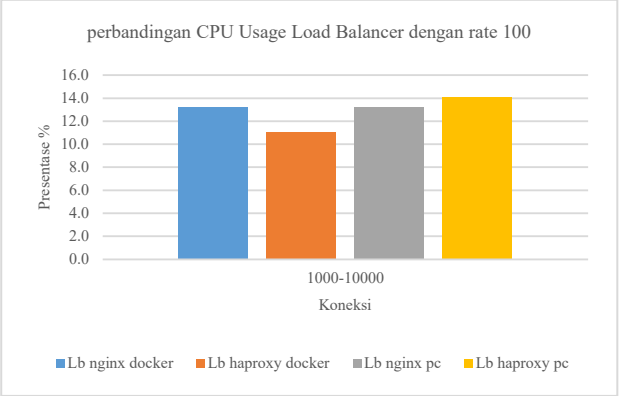


Gambar 16. Perbandingan Response Time dengan rate 100

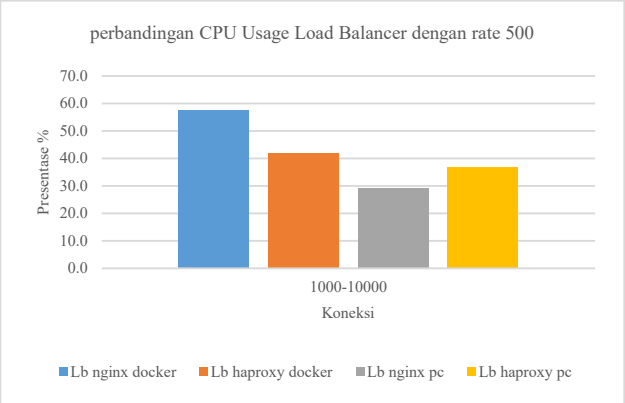


Gambar 17. Perbandingan reponse time dengan rate 500

3) Parameter *CPU Usage: Perbandingan* parameter *CPU Usage*

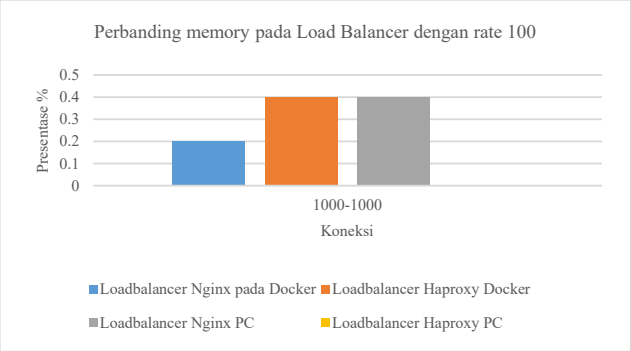


Gambar 18. Perbandingan CPU Usage dengan rate 100

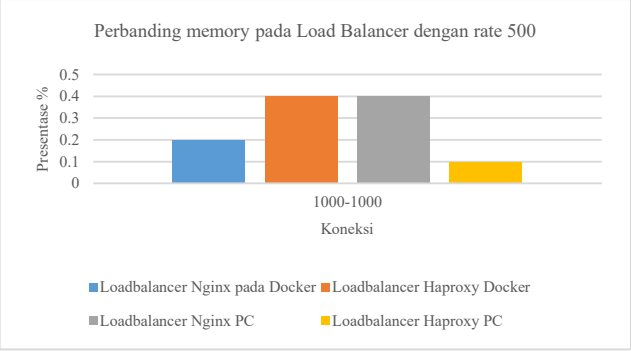


Gambar 19. Perbandingan CPU Usage dengan rate 500

4) Parameter Memory Usage: *Perbandingan* parameter *Memory Usage*



Gambar 20. Perbandingan Memory Usage dengan rate 100



Gambar 21. Perbandingan Memory Usage dengan rate 500

Pada perbandingan dengan *rate* 100 masing-masing *Load Balancer* masih mampu bekerja dengan sangat baik karena hanya menjalankan 100 per-koneksi, tetapi pada saat menjalankan pengujian dengan 500 per-koneksi hanya *Load Balancer Haproxy* pada *Docker* mampu menjalankan semua dengan baik dari rata-rata *Throughput* yang didapat 8418,6 Kb/s dengan rata-rata *Response Time* yang didapat 2,2 *ms* serta rata-rata penggunaan *CPU Usage* hanya 42% dan rata-rata penggunaan memory 0,4%.

VII. KESIMPULAN

Pada penelitian ini layanan dapat diakses pada saat dengan satu *webserver* tetapi fungsional *Load Balancer* tidak dapat bekerja. Untuk penerapa *Load Balancer* pada *Docker* yang paling bagus menggunakan *Haproxy* karena dari penelitian yang ditelah dilakukan *Throughput* yang didapat lebih besar serta response yang kecil sehingga dengan keneksi dan permintaan yang besar *Load Balancer Haproxy* pada *Docker* dapat melayani tanpa adanya error sedangkan di *Load Balancer Nginx* pada *Docker* terdapat rata-rata error sebesar 739,8 *connection time out*.

VIII. SARAN

Saran untuk penelitian selanjutnya berdasarkan kesimpulan terhadap hasil penelitian ini sebagi berikut:

1. Menggunakan tiga atau lebih *Webserver* sebagai pengujian pembagian beban, serta menguji fungsional Load Balancer dengan cara memtikan salah satu *Webserver*/node.
2. Menggunakan *Load Balancer Kubernetes*s dan membandingkan dengan *Docker Swarm* atau komputer *konvesional.*
3. Menambahkan penggunaan *database* untuk sistem *Load Balancer* yang lebih komplek.
4. Membandingkan algoritma yang ada pada *Load Balancer.*
5. Menambahkan pengujian *Quality of service,* untuk pengujian dan perbandingan yang lebih detail.

REFERENSI

[1] D. Rahmana, R. Primananda, and W. Yahya,

ISSN 2615-6334 (Online)
ISSN 2087-5347 (Print)

"Analisis Load Balancing Pada Web Server Menggunakan Algoritme Weighted Least Connection," *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 2, no. 3, pp. 915–920, Mar. 2018, [Online]. Available: https://j-ptiik.ub.ac.id/index.php/j-ptiik/article/view/1010/382.

[2] F. Apriliansyah, I. Fitri, and A. Iskandar, "Implementasi Load Balancing Pada Web Server Menggunakan Nginx," *J. Teknol. dan Manaj. Inform.*, vol. 6, no. 1, 2020, doi: 10.26905/jtmi.v6i1.3792.

[3] F. M. Azmi, M. Data, and H. Nurwasito, "Perbandingan Kinerja Haproxy dan Zevenet Dalam Pengimplementasian Multi Service Load Balancing," *J. Pengemb. Teknol. Inf. dan Ilmu Komput. Univ. Brawijaya*, vol. 3, no. 1, pp. 253–260, 2019.

[4] D. S. Afis, M. Data, and W. Yahya, "Load Balancing Server Web Berdasarkan Jumlah Koneksi Klien Pada Docker Swarm," *J. Pengemb. Teknol. Inf. dan Ilmu Komput. Univ. Brawijaya*, vol. 3, no. 1, pp. 925–930, 2019.

[5] W. Chen, A. Noertjahyana, and J. Andjarwirawan, "Analisis Perbandingan Kinerja Algoritma Load Balancer NGINX pada Studi Kasus PRS," *J. Infra*, vol. 7, no. 2, pp. 60–64, 2019.

[6] I. Farid, M. Idhom, and H. E. Wahanani, "Web Server Load Balancing Pada Arsitektur," vol. 1, no. 3, pp. 775–782, 2020.