



Load Balancing Test Results as Management to Prevent Server Down

Muhammad Raihan¹

¹Engineering and Computer Science, Informatic Engineering, Universitas Harapan, Medan, Indonesia
muhammadraihan3101@gmail.com

Article Info

Article history:

Received August 27, 2025

Revised August 28, 2025

Accepted August 29, 2025

Keywords:

Docker
Load balancing
Nginx
Round robin
Server overload

ABSTRACT

Increased traffic on a website can lead to server overload, which in turn results in system downtime, slow response, and potential service disruptions that negatively impact user experience. This issue becomes particularly critical for systems that depend on a single main server without an efficient workload distribution mechanism. To address this challenge, this study aims to implement load balancing techniques using the round robin method to distribute incoming requests evenly across multiple backend servers. The research methodology involves system architecture design utilising Docker for containerisation and Nginx as a reverse proxy to manage traffic flow, combined with performance testing conducted using the K6 tool under various simulated traffic scenarios. Experimental results demonstrate that the round robin load balancing method successfully balances the distribution of user requests, reduces the risk of server overload, improves system reliability, and enhances response time performance. Therefore, the proposed approach proves effective in maintaining service availability and optimising overall system performance, especially under high and fluctuating traffic conditions.

This is an open-access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Muhammad Raihan
Universitas Harapan Medan
Email: muhammadraihan3101@gmail.com

1. INTRODUCTION

The use of websites as the primary means of disseminating information and services is increasing. With the high number of users, the challenges in managing website traffic are becoming increasingly complex. One of the problems often encountered is an increase in traffic on applications or web servers, which can result in a higher workload on the server, leading to decreased performance, response times, and even system failures.

Load balancing is a network concept that distributes the workload across two or more connections evenly so that the work runs optimally and does not experience connection overload [1]. Web servers can be maintained with the use of load balancing. When one server cannot serve a request, another server automatically replaces it [2]. Load balancing consists of virtual servers and real servers. Virtual servers are devices that distribute the load to servers. Generally, the targeted server is the one with the least load [3].

The Round Robin method in load balancing implementation is a scheduling algorithm. The basic concept of the Round Robin algorithm is based on time sharing, where the algorithm processes the queue in a sequential manner [4]. Each server receives requests in turn, forming a sequential and continuous cycle [5]. The mechanism is that each client's first request is directed to server 1, then to server 2, and subsequent requests are directed based on repetition until the request is completed [6].

The author decided to develop a system that implements load balancing technology using the round robin method, designed to distribute user requests evenly across available servers, so that each server receives balanced performance. It is hoped that website performance will improve, response times will be better, and website availability will be maintained even under high traffic conditions.

2. METHOD

2.1 Load Balancing

Load balancing is a process in a computer network system that aims to distribute the workload evenly across a number of available servers. In practice, when a user sends a request to a service such as accessing a website or application, the request is first received by a load balancer, which is a component that functions as a data traffic controller. The load balancer then analyzes the condition of each available server, for example by monitoring CPU load, number of active connections, and response time. Based on this information, the load balancer will determine which server is most suitable to handle the request [7]. The following is an overview of how load balancing works.



Figure 1. Load Balancing Work Process

Load balancing works to ensure that network traffic remains smooth and user requests are distributed evenly. Here's how it works:

1. Load balancing handles incoming requests from users for information and other services. The load balancer sits between the servers that handle these requests.
2. Once a request is received, load balancing first determines which servers are available and online, then routes the request to that server.
3. If a server goes down, the load balancer can redirect traffic to another available server. Conversely, the load balancer can also reduce server load as needed.

2.2 Round Robin

Round robin is a method widely used by load balancing devices. The round robin method works by dividing the load sequentially and in turn from one server to another. The basic concept of round robin is to use time sharing, in essence this algorithm processes queues in turn [8]. Round robin is one of the process scheduling algorithms in operating systems. Round robin is designed to divide the time for each process equally and in a circular order, running all processes without priority [9].

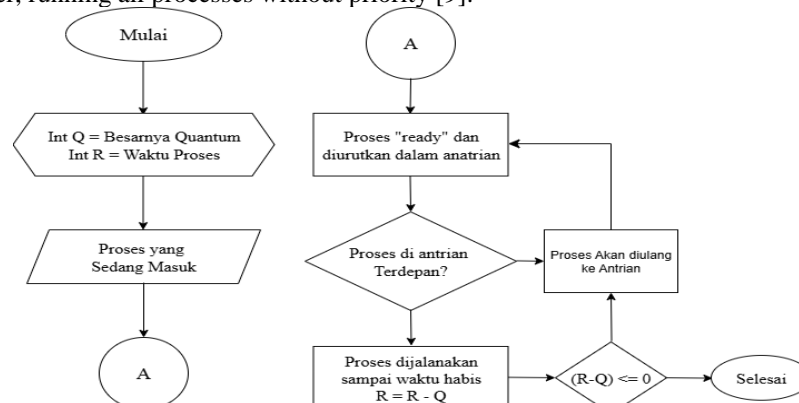


Figure 2. Load Balancing Algorithm Flowchart

The flowchart illustrates the Round Robin scheduling algorithm, which divides CPU time in turns using a time quantum (a specific amount of time given to each process to run before it is switched to another process) [10]. The flow is as follows:

1. Start = Initialize Variables
 - a. Q = Quantum Size (process running time)
 - b. R = Process Time (how long the process needs to run).
2. Process Entered? If there is a new process, it is added to the queue.
3. Processes in Queue = Processes are sorted according to user request.
4. Is process X at the front of the queue:
 - a. Yes = process runs until time expires
 - b. No = The process returns to the queue.
5. Recalculate the remaining time:

$R = R - Q$ (the process time is reduced by the quantum).
6. Is $R \leq 0$?
 - a. Yes = Process Complete
 - b. No = The process returns to the queue, for the next turn.

2.2.1 Round Robin Formula

The formula for round robin is as follows:

Initials / variables

N : (number of servers: S1,S2,S3)

R : Total requests

j : Request sequence number (starting from 1)

1. Calculate the average quota and remainder:

- $q = \left\lfloor \frac{R}{N} \right\rfloor$

- $r = R \bmod N$

2. Request mapping:

- $\text{server}(j) = ((j - 1) \bmod N) + 1$

3. RESULTS AND DISCUSSION

This stage is the first step in designing a system or application, which aims to identify the requirements, limitations, and relationships of the necessary parts of the system. The load balancing system for high traffic in a server environment uses the round robin method to distribute traffic or workload evenly among several resources or servers to prevent overload on server requests using the round robin method.

3.1 Network Topology

Topology describes the arrangement or architecture of devices and connections that are interconnected to form a network used for load balancing, as follows:

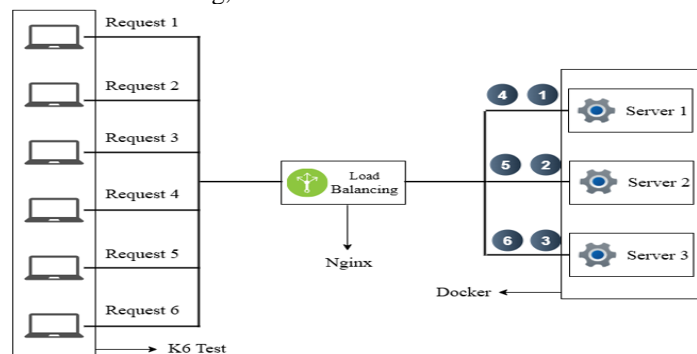


Figure 3. Round Robin Method Architecture

There are 5 clients making requests to the server. The load balancer uses the round robin method to select an available server according to the queue to send the request, as described below:

1. Client 1 sends a request. The load balancer sends the request to server 1 because it is first in the queue on server 1.
2. Client 2 sends a request, the load balancer will send the request to server 2 because the queue on server 2 is full and server 2 is still empty.
3. Client 3 sends a request, the load balancer will send the request to server 3 because the queue on server 2 is full and server 3 is still empty.
4. Client 4 sends a request, the load balancer will send the request to server 1 because the next queue is back to server 1.
5. Client 5 sends a request, the load balancer will send the request to server 2 because the next queue is server 2.
6. Client 6 sends a request, the load balancer will send the request to server 3 because the next queue is server 3.
7. And so on, if there is a new request, the load balancer will send the request to the server according to its queue.

3.2 Load Balancing Flowchart

Flowchart is used to illustrate the flow of how load balancing works, as shown in the following image:

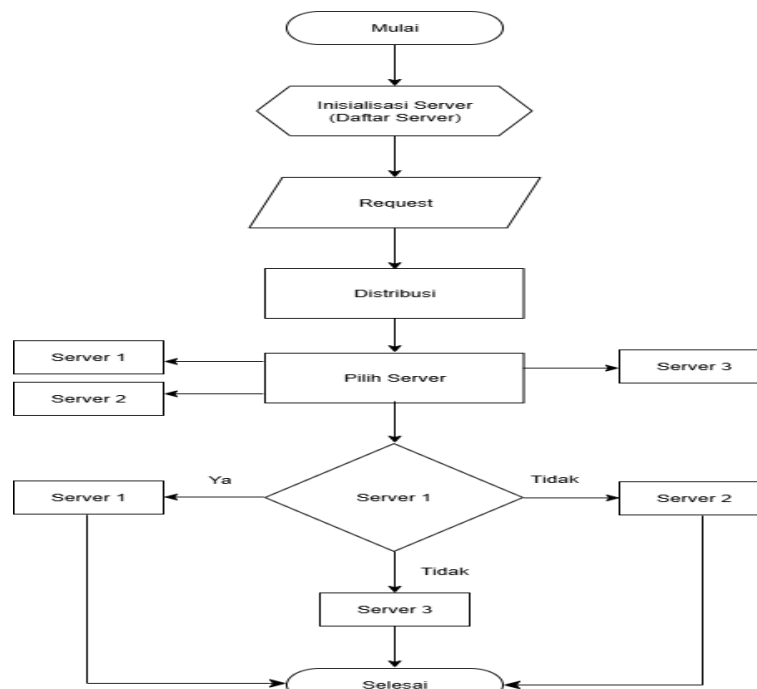


Figure 4. Load Balancing Flowchart

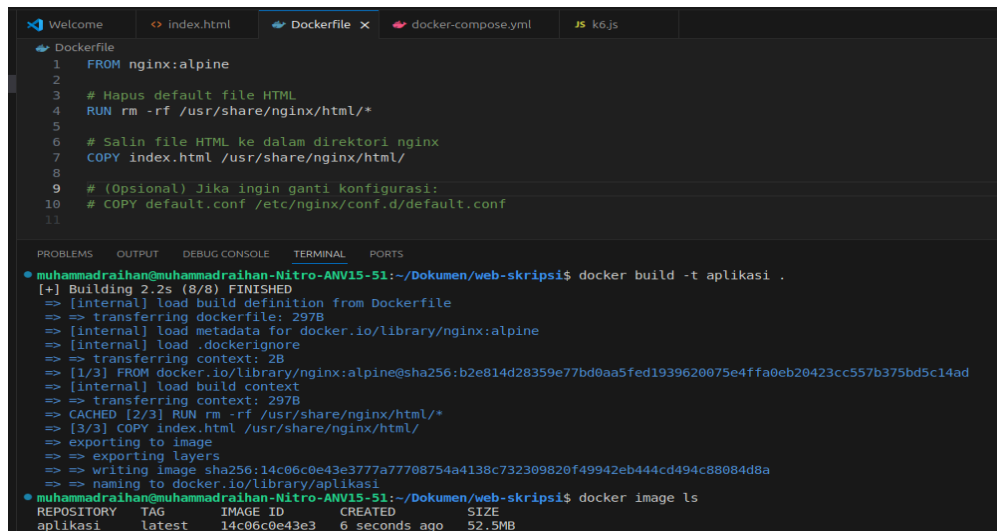
The flowchart illustrates how load balancing works, with the following details:

1. Start: This process describes the beginning of a load balancing system.
2. Initialize the list of available servers: Load balancing will create the names and numbers of available servers.
3. Store requests in a round robin queue: Load balancing will store all requests directed to the server in a queue using the round robin method.
4. Select Server: Load balancing selects available servers sequentially according to the round-robin algorithm.
5. Server availability check: Load balancing will check the server; if the server is available, the request will be forwarded to the server; if the server is unavailable, the request will return to the queue.

6. Send response: Load balancing sends the user's response according to the selected server.
7. Completion: This indicates that the system has finished.

3.3 Dockerfile Configuration

Dockerfile is a text file containing automated instructions for building a docker image, where a docker image is a package containing all the files, libraries, configurations, and instructions needed to run an application in a container. This ensures that developers can run applications with the same configuration.



The screenshot shows a code editor with a Dockerfile and a terminal window displaying the build process. The Dockerfile contains the following instructions:

```

1 FROM nginx:alpine
2
3 # Hapus default file HTML
4 RUN rm -rf /usr/share/nginx/html/*
5
6 # Salin file HTML ke dalam direktori nginx
7 COPY index.html /usr/share/nginx/html/
8
9 # (Opsional) Jika ingin ganti konfigurasi:
10 # COPY default.conf /etc/nginx/conf.d/default.conf
11

```

The terminal window shows the command `docker build -t aplikasi .` being executed, followed by the build process output:

```

[+] Building 2.2s (8/8) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 297B
=> [internal] load metadata for docker.io/library/nginx:alpine
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/3] FROM docker.io/library/nginx:alpine@sha256:b2e814d28359e77bd0aa5fed1939620075e4ffa0eb20423cc557b375bd5c14ad
=> [internal] load build context
=> => transferring context: 297B
=> CACHED [2/3] RUN rm -rf /usr/share/nginx/html/*
=> [3/3] COPY index.html /usr/share/nginx/html/
=> exporting to image
=> => exporting layers
=> writing image sha256:14c06c0e43e3777a77708754a4138c732309820f49942eb444cd494c88884d8a
=> naming to docker.io/library/aplikasi

muhammadraihan@muhammadraihan-Nitro-ANV15-51:~/Dokumen/web-skripsi$ docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
aplikasi latest 14c06c0e43e3 6 seconds ago 52.5MB

```

Figure 5. Dockerfile Configuration and Build Process

The process of configuring the creation of a Docker image to build a container that will run the configuration and instructions needed so that the application can be easily distributed.

1. The FROM variable initializes the new creation and sets the base image (a valid Dockerfile must start with the FROM instruction). Code FROM nginx: alpine. This describes the base image used in the Docker, which is the nginx image with the alpine version (a lightweight version of Linux).
2. The RUN variable is used to execute a new layer command on top of the current image. RUN rm -rf /usr/share/nginx/html/. This code serves to delete all default files from the nginx image, which will be replaced with the image created by the author.
3. The COPY variable is used to copy new files or folders into the docker image. COPY index.html /usr/share/nginx/html/. This is used to copy the author's index.html file into the default html folder so that it will be displayed on the server.

After completing the Dockerfile configuration process, proceed to the build process, which will create the docker image.

1. Script docker build -t application. (docker will build a new image, with the -t script serving to give it a name, the application script is the name of the image that has been created, and. is the location of the file in a folder).
2. Script docker image ls (command to display a list of docker images that have been created and stored in the local system).

3.4 Load Balancer Configuration With Nginx

This configuration sets up nginx and a load balancer, which receives requests from clients and forwards them to two backend servers running on ports 8085, 8086, and 8087. The load is then distributed (load balancing), and the client's original IP address is forwarded to the backend for logging.

```

GNU nano 7.2                                load-balancer
upstream backend {
    server 127.0.0.1:8085;
    server 127.0.0.1:8086;
    server 127.0.0.1:8087;
}

server {
    listen 80;
    listen [::]:80;
    server_name 127.0.0.1;

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

```

Figure 6. Load Balancer with Nginx

The image shows the Load Balancing configuration to determine how the server will respond to HTTP requests. Here is an explanation of each line:

1. `/etc/nginx/sites-available/load-balancer` (used to create load balancer configurations with Nginx).
2. Upstream block (The upstream block is used to define a group, which in this script the author has named server backend).
 - a. Three backend servers are defined: 127.0.0.1:8085, 127.0.0.1:8086, 127.0.0.1:8087.
 - b. Nginx performs load balancing between the three servers alternately.
3. Server block (used to determine how Nginx should handle requests based on domain, IP, and port).
 - a. `listen 80;` means that the Nginx server will listen for HTTP requests on port 80.
 - b. `listen [::]:80;` for IPv6 support.
 - c. `server_name 127.0.0.1;` means it only handles requests to 127.0.0.1.
4. Location / block (used to set how Nginx handles requests to specific paths on the server).
 - a. `location /` handles all requests to the root path (/).
 - b. `proxy_pass http://backend;` forwards requests to the defined backend.
 - c. `proxy_set_header` is used to forward important information from the client to the server.

3.5 K6 Test Configuration

K6 test configuration is a setting or script used to determine how to simulate load on a web application or API using the K6 tool. The purpose of this configuration is to test system performance when receiving a large number of requests simultaneously.

```

index.html  index-2.html  index-3.html  Dockerfile  docker-compose.yml  JS k6.js
s k6.js > default
1  import http from 'k6/http';
2  import { check, sleep } from 'k6';
3  import { scenario } from 'k6/execution';
4
5  const url = 'http://127.0.0.1/';
6
7
8  export const options = {
9    scenarios: {
10      per_vu_timeout: {
11        executor: 'per-vu-iterations',
12        vus: 100, // Jumlah VU
13        iterations: 1, // Iterasi per VU
14        //maxDuration: '120s', // Maksimal waktu per VU
15      },
16    },
17  };
18
19  export default function () {
20    const res = http.get(url);
21    check(res, {
22      'status is 200': (r) => r.status === 200,
23    });
24  }

```

Figure 7. k6 Test Configuration

Figure shows the contents of the k6 Test configuration. The following is an explanation of each line:

1. Import the http module to send HTTP requests.
2. Import the check function to validate responses and sleep if needed.
3. The test target is http://127.0.0.1/ (localhost).
4. Executor: 'per-vu-iterations' (Each virtual user will run a certain number of iterations. In this case, 1 iteration per VU).
5. Vus: 5450, Using 5450 virtual users (Indicates high load).
6. Iterations: 1 (Each virtual user will only make one request).
7. Main test function: Perform an HTTP GET to the target URL.

The main objectives are to:

1. Determine whether the server is capable of handling 5450 requests simultaneously.
2. Check whether all responses from the server return HTTP status 200 (OK).

If the server is powerful and well-configured, then:

1. All requests will be successful (status 200).
2. The output in the terminal will show a high success rate and stable response time.

If the server is unable to handle the load:

1. Errors such as timeout, connection refused, or a status other than 200 will occur.
2. This indicates that the server needs to be strengthened or optimized.

3.6 System Testing

At this stage, testing is conducted on the system that has been built, namely Load Balancing Implementation on a server to display the results of traffic load caused by user requests, using 1, 2, and 3 servers starting from 3000 requests on each server up to the maximum limit that the server can accommodate the load of requests received using a maximum of 4 CPU cores.

3.6.1 Testing 1 Server Without Load Balancer

Traffic load testing with 3000 requests per second using only 1 server, 1 container with a maximum of 4 CPU cores.

1. Traffic load testing with 3000 requests using 1 server with 1 container 4 core CPU:

```

TOTAL RESULTS

checks_total.....: 3000 14083.172447/s
checks_succeeded.....: 100.00% 3000 out of 3000
checks_failed.....: 0.00% 0 out of 3000

✓ status is 200

HTTP
http_req_duration.....: avg=18.06ms min=862.02µs med=12.76ms max=198.01ms p(90)=24.4ms p(95)=29.17ms
{ expected_response:true }.....: avg=18.06ms min=862.02µs med=12.76ms max=198.01ms p(90)=24.4ms p(95)=29.17ms
http_req_failed.....: 0.00% 0 out of 3000
http_reqs.....: 3000 14083.172447/s

EXECUTION
iteration_duration.....: avg=27.18ms min=1.27ms med=17.46ms max=208.64ms p(90)=37.78ms p(95)=109.33ms
iterations.....: 3000 14083.172447/s

NETWORK
data_received.....: 22 MB 103 MB/s
data_sent.....: 195 kB 915 kB/s
  
```

Figure 8. Testing Successful on 1 Server with 3000 Requests

The test was successfully conducted and produced the following data:

- a. Total number of checks = 3000 data
- b. Successful data = 100% (all data loaded successfully)
- c. Error data = 0% (no data failed)
- d. Average duration (avg) = 18.06ms
- e. Total duration = 14.083 requests per second

- f. Total data received = 22 mb
- g. Total data sent = 195 kb

The test results show that the server has excellent performance with all HTTP requests successful (status 200), no failures, fast response times (average 18ms), and high throughput reaching over 14,000 requests per second, indicating that the server is capable of handling loads efficiently and stably.

- 2. Traffic load testing with 10,000 requests using 1 server with 1 container and 4 core CPU:

```

TOTAL RESULTS
checks_total.....: 10000 8299.189444/s
checks_succeeded.....: 70.53% 7053 out of 10000
checks_failed.....: 29.47% 2947 out of 10000

x status is 200
  70% - x 7053 / x 2947

HTTP
http_req_duration.....: avg=125.76ms min=0s med=133.42ms max=324.46ms p(90)=226.2ms p(95)=235.14ms
{ expected_response:true }.....: avg=161.48ms min=66.67ms med=151.59ms max=297.06ms p(90)=229.86ms p(95)=239.11ms
http_req_failed.....: 29.47% 2947 out of 10000
http_reqs.....: 10000 8299.189444/s

EXECUTION
iteration duration.....: avg=505.08ms min=239.13ms med=427.64ms max=1.19s p(90)=901.58ms p(95)=935.88ms
iterations.....: 10000 8299.189444/s
vus.....: 9567 min=9567 max=9567
vus_max.....: 10000 min=10000 max=10000

NETWORK
data_received.....: 52 MB 43 MB/s
data_sent.....: 480 KB 398 KB/s

```

Figure 8. Failed Test of 1 Server With 10000 Requests

The test was successfully conducted and produced the following data:

- a. Total number of checks = 10,000 data points
- b. Successful data = 70.53% (7,053 successful data points)
- c. Error data = 29.47% (2,947 failed data)
- d. Average duration (avg) = 161.48ms
- e. Total duration = 8,299.18 requests per second
- f. Total data received = 52 mb
- g. Total data sent = 480 kb

The server experienced high pressure with 70.53% of requests successful and 29.47% failing, possibly due to excessive server load from 10,000 users. The response time increased to 125 ms and the execution duration reached more than 500 ms per iteration, indicating that the system was not yet stable enough for this amount of traffic.

4. CONCLUSION

The author has successfully built and implemented load balancing in a high-traffic server environment using containers provided by Docker to build a server. The implementation of load balancing using the round robin method significantly improved system performance and distributed the server's workload evenly in order to cope with increased traffic in the server environment. The round robin method successfully reduced response times and accelerated user request processes. Based on the test results by sending 3000 requests to the server without using the load balancing method, the first experiment produced the following data: Total duration = 14,083 requests per second. then testing using a load balancer with a division of 2 servers resulted in a total duration of 10,638 requests per second, which was faster than before, and the last test using 3 load balancer servers resulted in a total duration of 7,987 requests per second, resulting in an even better response speed.

ACKNOWLEDGEMENTS

First of all, the author would like to express gratitude for completing this article and research. The author would like to express his deepest gratitude to his supervisor for providing guidance and motivation so

that the author could complete this research well. The author realizes that this research still has many shortcomings and needs to be developed further.

REFERENCES

- [1] Ahmad, "Kenalan dengan Traffic Website Beserta Jenis-jenisnya," UNIVERSITASBAKRIE, 2023. [Online]. Available: <https://bakrie.ac.id/articles/751-kenalan-dengan-traffic-website-beserta-jenis-jenisnya-yuk.html>
- [2] Amira, "Pengertian Server: Jenis, Fungsi dan Cara Kerjanya," Gramedia Blog, 2021. [Online]. Available: <https://www.gramedia.com/literasi/media/>
- [3] Codingstudio, "Apa Itu Docker? Pengertian, Fungsi dan Cara Kerjanya," Codingstudio.id, 2023
- [4] DicodingIntern, "Apa itu Web Server dan Fungsinya," DICODING, 2021. [Online]. Available: <https://www.dicoding.com/blog/apa-itu-web-server-dan-fungsinya/>
- [5] Faradilla, "Web Server: Memahami Pengertian, Fungsi, dan Cara Kerjanya," HOSTINGER, 2024. [Online]. Available: <https://www.hostinger.com/id/tutorial/apa-itu-web-server#:~:text=Nah%2C>
- [6] Ikhtisar, "Docker memungkinkan Anda membuat, menguji, dan menyebarkan aplikasi dengan cepat," AWS, 2024.
- [7] M. Ilham, "Implementasi Sistem Load Balancing untuk Optimalisasi Kinerja pada Web Server Nginx Menggunakan Algoritma IP Hash," vol. 9, pp. 60–68, 2024.
- [8] A. M. Komaruddin, D. M. Sipitorini, and P. Rispian, "Load Balancing dengan Metode Round Robin Untuk Pembagian Beban Kerja Web Server," 2019.
- [9] A. N. Malik, "Simulasi Desain Load Balancing Dengan Menggunakan Metode NTH," Repository Dinamika, 2023. [Online]. Available: <https://repository.dinamika.ac.id/cgi/search/simple?q=load+balancing>
- [10] Meilinaeka, "Docker Adalah: Pengertian, Cara Kerja, Dan Empat Komponen Utama," IT Telkom University, 2023. [Online]. Available: <https://it.telkomuniversity.ac.id/docker-adalah-pengertian-cara-kerja-dan-empat-komponen-utama/>
- [11] R. Oktariyadi, I. Ruslianto, and S. Bahri, "Analisa Kinerja Load Balancing Menggunakan Metode Round Robin Dan Weighted Round Robin," Coding Jurnal Komputer Dan Aplikasi, vol. 9, no. 01, p. 131, 2021. <https://doi.org/10.26418/coding.v9i01.45871>
- [12] G. Rahayu, U. Sunarya, and A. Novianti, "Rancang Bangun Web Server Untuk Pemantauan Design Web Server for Monitoring the Cultivation of Vannamei Shimp Using," E-Proceeding Of Applied Science, vol. 3, no. 3, pp. 2066–2071, 2017.
- [13] Rian, "Apa Itu Traffic di Website? Jenis, Manfaat & Pentingkah?," Nextdigital, 2024. [Online]. Available: <https://nextdigital.co.id/apa-itu-traffic/>
- [14] M. F. Ridho, "Menguji Kinerja API Menggunakan K6," Medium, 2024. [Online]. Available: <https://blog.dot.co.id/menguji-kinerja-api-menggunakan-k6-a3b9d0f42549>
- [15] S. Safriadi and R. Rahmadani, "Analisis Kinerja Load Balancing Round Robin Pada Website Skalabel," Journal of Information System Management (JOISM), vol. 5, no. 2, pp. 227–232, 2024. <https://doi.org/10.24076/joism.2024v5i2.1441>