

Laporan Cloud Computing Tugas 2

Ricardo Supriyanto / 5025221218

Link Repository: <https://github.com/Ricardo08S/Task-CloudComputing/tree/main/Task-2>

1. Pendahuluan

Tugas ini bertujuan untuk mendalami implementasi containerization menggunakan Docker melalui eksplorasi tiga kasus yang terdapat pada repositori <https://github.com/rm77/cloud2023/tree/master/containers/docker>. Selain itu, tugas ini mengharuskan pengembangan tambahan berupa satu kasus baru (Case 4) berdasarkan kreasi sendiri, termasuk penyusunan arsitektur, pembuatan *Script*, serta dokumentasi lengkap berupa screenshot dan penjelasan.

2. Deskripsi Tugas

Tugas ini terdiri dari tiga kasus utama yang harus dijalankan menggunakan Docker. Setiap kasus berkaitan dengan konfigurasi dan pengelolaan container untuk layanan tertentu. Peserta diminta untuk mengikuti langkah-langkah dalam menjalankan container-container tersebut dan mendokumentasikan prosesnya beserta penjelasan rinci terkait implementasinya.

Case 1: Menggunakan background process di Docker untuk mengumpulkan dan menyimpan Chuck Norris jokes.

Case 2: Menjalankan web server Python menggunakan modul *http.server* untuk menyajikan file HTML dari direktori host.

Case 3: Menjalankan MySQL dan phpMyAdmin dalam container terpisah, serta menghubungkannya untuk pengelolaan basis data.

Case 4 (Case tambahan): Menggunakan Nginx sebagai Load Balancer untuk Menyebarakan Beban ke Beberapa Web Server

3. Arsitektur dan Implementasi Script

Case 1: Mengumpulkan Chuck Norris Jokes

1. Deskripsi

- Memanfaatkan skrip **getjokes.sh** untuk mengambil lelucon Chuck Norris dari API dan menyimpannya dalam file log di container. Skrip ini berjalan secara otomatis setiap 5 detik.

2. Tahapan

- Sebelum menjalankan skrip **run_process.sh**, pastikan untuk membuat direktori **files** di host sebagai lokasi penyimpanan lelucon.
- Skrip **run_process.sh** digunakan untuk mengatur dan menjalankan container berbasis image **alpine:3.18**.

- Menggunakan **curl** untuk mengambil data dari API, dan **jq** untuk memproses data JSON agar hanya lelucon yang diambil.
3. **Penjelasan**
 - Container akan menjalankan skrip di background untuk mengunduh lelucon secara berkala setiap 5 detik dan menyimpannya ke dalam file dengan format `/files/output_*.txt`.

Case 2: Menjalankan Web Server Python dengan `http.server`

1. **Deskripsi**
 - Menggunakan modul bawaan Python, yaitu **`http.server`**, untuk menyajikan file HTML dari direktori host yang di-mount ke container.
2. **Tahapan**
 - Skrip **`run_simple_web.sh`** digunakan untuk memulai web server yang melayani file **`index.html`** dari direktori **`./files`** pada host.
 - Server diatur untuk berjalan pada port 9999, sehingga dapat diakses melalui browser dengan URL <http://localhost:9999>.
3. **Penjelasan**
 - Web server berjalan di dalam container, dengan direktori **`./files`** pada host di-mount ke dalam container sebagai **`/html`**. File **`index.html`** disajikan melalui port 9999.

Case 3: MySQL dan phpMyAdmin dengan Docker

1. **Deskripsi**
 - MySQL dan phpMyAdmin dijalankan di dua container terpisah. Data MySQL disimpan secara persisten di host, dalam direktori **`./dbdata`**.
2. **Tahapan**
 - Skrip **`run_mysql.sh`** digunakan untuk memulai container MySQL dengan konfigurasi yang sesuai.
 - Skrip **`run_myadmin.sh`** digunakan untuk menjalankan phpMyAdmin dan menghubungkannya dengan MySQL, dengan phpMyAdmin dapat diakses pada port 10000.
3. **Penjelasan**
 - MySQL berjalan di container dengan direktori data yang dipetakan ke **`./dbdata`** pada host, sehingga data tetap tersimpan meskipun container dihentikan.
 - phpMyAdmin tersedia melalui port 10000, memberikan antarmuka untuk mengelola MySQL secara visual.

Case 4: Menggunakan Nginx sebagai Load Balancer untuk Web Server

1. **Deskripsi**
 - Menggunakan Nginx sebagai load balancer untuk mendistribusikan permintaan HTTP antara dua web server yang berjalan di container Docker.
 - Setiap web server menyajikan halaman HTML statis yang berbeda.
2. **Tahapan**

- Siapkan dua file HTML statis, yaitu **index1.html** dan **index2.html**, yang masing-masing akan dilayani oleh web server pertama dan kedua.
- Buat file konfigurasi **nginx.conf** untuk mengatur Nginx sebagai load balancer, dengan mengarahkan permintaan ke dua server backend (web1 dan web2).
- Jalankan skrip **load_balancer.sh** untuk:

```

1  docker rm -f web_server_1 web_server_2 nginx_load_balancer
2
3  docker container run \
4      --name web_server_1 \
5      -dit \
6      -v $(pwd)/index1.html:/usr/share/nginx/html/index.html \
7      nginx:alpine
8
9  docker container run \
10     --name web_server_2 \
11     -dit \
12     -v $(pwd)/index2.html:/usr/share/nginx/html/index.html \
13     nginx:alpine
14
15  docker container run \
16     --name nginx_load_balancer \
17     -dit \
18     -v $(pwd)/nginx.conf:/etc/nginx/nginx.conf \
19     -p 8080:80 \
20     --link web_server_1:web1 \
21     --link web_server_2:web2 \
22     nginx:alpine

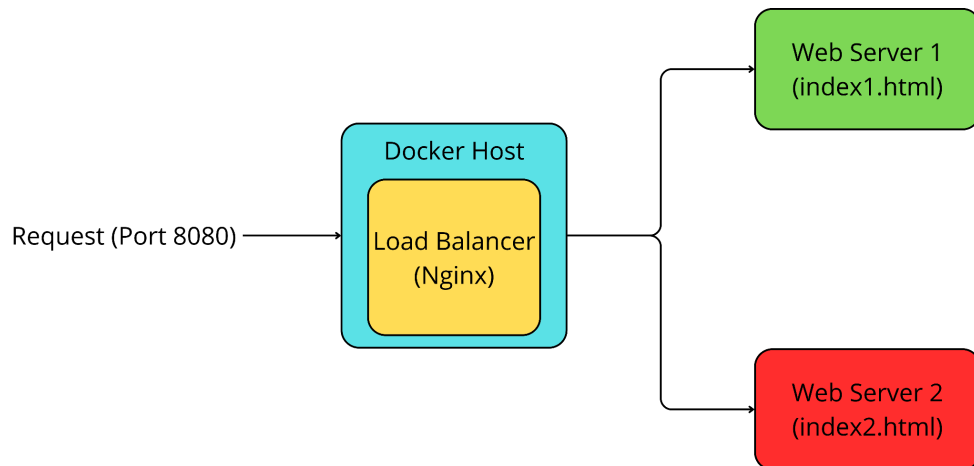
```

- Menghapus container lama (jika ada).
- Menjalankan dua web server berbasis **nginx:alpine** untuk menyajikan file HTML.
- Menjalankan container Nginx sebagai load balancer pada port 8080, dengan menghubungkan ke dua web server backend.

3. Penjelasan

- Dua web server menyajikan file HTML yang berbeda:
 - Web Server 1: Menampilkan "Welcome to Web Server 1" dari **index1.html**.
 - Web Server 2: Menampilkan "Welcome to Web Server 2" dari **index2.html**.
- Nginx bertindak sebagai load balancer dengan membagi permintaan HTTP ke kedua server secara merata.
- Saat mengakses **http://localhost:8080**, Nginx akan mendistribusikan permintaan secara bergantian antara kedua web server, menampilkan konten dari **index1.html** atau **index2.html** sesuai dengan load balancing.

Arsitektur Case 4:



1. Docker Host

- **Deskripsi:**

Mesin fisik atau virtual tempat container Docker dijalankan, berfungsi sebagai lingkungan untuk menjalankan dua web server dan load balancer.

- **Fungsi:**

Menyediakan sumber daya untuk container, termasuk jaringan, penyimpanan, dan pengolahan data.

2. Nginx Load Balancer Container

- **Deskripsi:**

Container ini menjalankan Nginx yang dikonfigurasi sebagai load balancer. Tugasnya adalah mendistribusikan permintaan HTTP ke dua web server backend (web1 dan web2) secara merata.

- **Detail Konfigurasi:**

- **Port:** Akses dilakukan melalui port 8080 di host.

- **Konfigurasi Backend:** Nginx diarahkan untuk menggunakan upstream backend yang terdiri dari dua server backend (web1 dan web2).

- **Akses:**

Permintaan HTTP dikirim ke <http://localhost:8080> dan diteruskan secara bergantian ke salah satu backend.

3. Web Server Backend Containers

- **Deskripsi:**

- **Web Server 1:** Container pertama menyajikan halaman HTML statis dari file **index1.html**, yang menampilkan "Welcome to Web Server 1".

- **Web Server 2:** Container kedua menyajikan halaman HTML statis dari file **index2.html**, yang menampilkan "Welcome to Web Server 2".

- **Mount:**

File HTML di-mount dari direktori host ke dalam container menggunakan volume Docker, sehingga dapat dengan mudah disesuaikan.

- **Port Internal:**

Kedua container berjalan di port 80 secara internal, tetapi hanya diakses melalui load balancer.

4. File Konfigurasi Nginx (nginx.conf)

- **Deskripsi:**

File konfigurasi Nginx yang menentukan pengaturan load balancing dan daftar server backend.

- **Detail Konfigurasi:**

- Mendefinisikan upstream bernama backend yang terdiri dari web1 dan web2.
- Menggunakan metode round-robin untuk mendistribusikan permintaan HTTP secara merata.

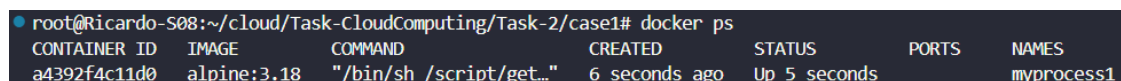
Kapan Skenario Ini Cocok Digunakan?

Skenario ini cocok digunakan untuk aplikasi yang membutuhkan skalabilitas dan distribusi beban secara efisien, seperti:

- Sistem berbasis web yang memiliki banyak pengguna dan perlu memastikan performa tetap stabil meskipun terjadi lonjakan lalu lintas.
- Aplikasi yang ingin memastikan waktu respons tetap konsisten dengan mendistribusikan permintaan ke beberapa server.
- Pengembangan aplikasi berbasis container yang memanfaatkan load balancing untuk simulasi lingkungan produksi.

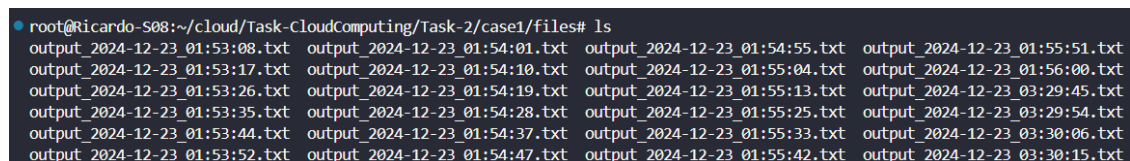
4. Screenshot Implementasi dan Penjelasannya

Case 1: Mengumpulkan Chuck Norris Jokes



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a4392f4c11d0	alpine:3.18	"/bin/sh /script/get..."	6 seconds ago	Up 5 seconds		myprocess1

Tampilan terminal yang menunjukkan proses **getjokes.sh** sedang berjalan di container Docker.



```
root@Ricardo-S08:~/cloud/Task-CloudComputing/Task-2/case1/files# ls
output_2024-12-23_01:53:08.txt  output_2024-12-23_01:54:01.txt  output_2024-12-23_01:54:55.txt  output_2024-12-23_01:55:51.txt
output_2024-12-23_01:53:17.txt  output_2024-12-23_01:54:10.txt  output_2024-12-23_01:55:04.txt  output_2024-12-23_01:56:00.txt
output_2024-12-23_01:53:26.txt  output_2024-12-23_01:54:19.txt  output_2024-12-23_01:55:13.txt  output_2024-12-23_03:29:45.txt
output_2024-12-23_01:53:35.txt  output_2024-12-23_01:54:28.txt  output_2024-12-23_01:55:25.txt  output_2024-12-23_03:29:54.txt
output_2024-12-23_01:53:44.txt  output_2024-12-23_01:54:37.txt  output_2024-12-23_01:55:33.txt  output_2024-12-23_03:30:06.txt
output_2024-12-23_01:53:52.txt  output_2024-12-23_01:54:47.txt  output_2024-12-23_01:55:42.txt  output_2024-12-23_03:30:15.txt
```

Isi folder **files** pada host yang menampilkan file-file berisi lelucon Chuck Norris.

Penjelasan:

Gambar menunjukkan bahwa proses pengambilan lelucon Chuck Norris sedang berjalan di dalam container Docker lokal. Lelucon-lelucon yang dihasilkan oleh proses ini disimpan secara otomatis ke dalam folder bernama **files** pada host. Hal ini membuktikan bahwa container dapat menjalankan proses secara otomatis di latar belakang.

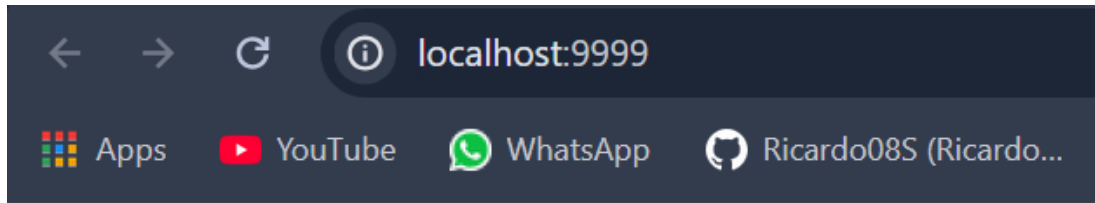
Case 2: Menjalankan Web Server Python dengan http.server

```

root@ricardo-508:~/cloud/Task-Cloudcomputing/Task-2/case2# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
0f91f12ce87c   python:3.13.0a1-alpine3.17   "python3 -m http.ser..."   3 seconds ago   Up 3 seconds   0.0.0.0:9999->9999/tcp, :::9999->9999/tcp   webserver1
root@ricardo-508:~/cloud/Task-Cloudcomputing/Task-2/case2#

```

Tampilan terminal yang menunjukkan container web server Python sedang berjalan.



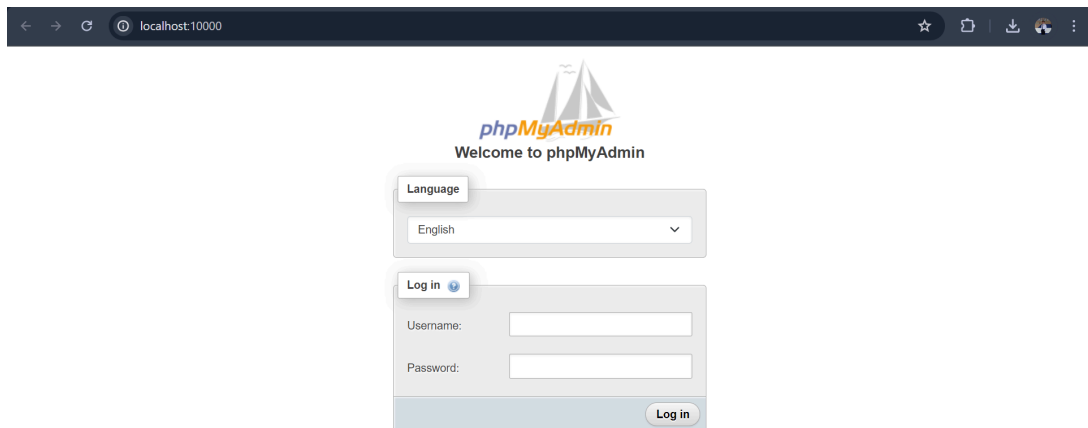
Hello Apa Kabar

Tampilan halaman web di browser yang menunjukkan isi file **index.html**, diakses melalui **http://localhost:9999**.

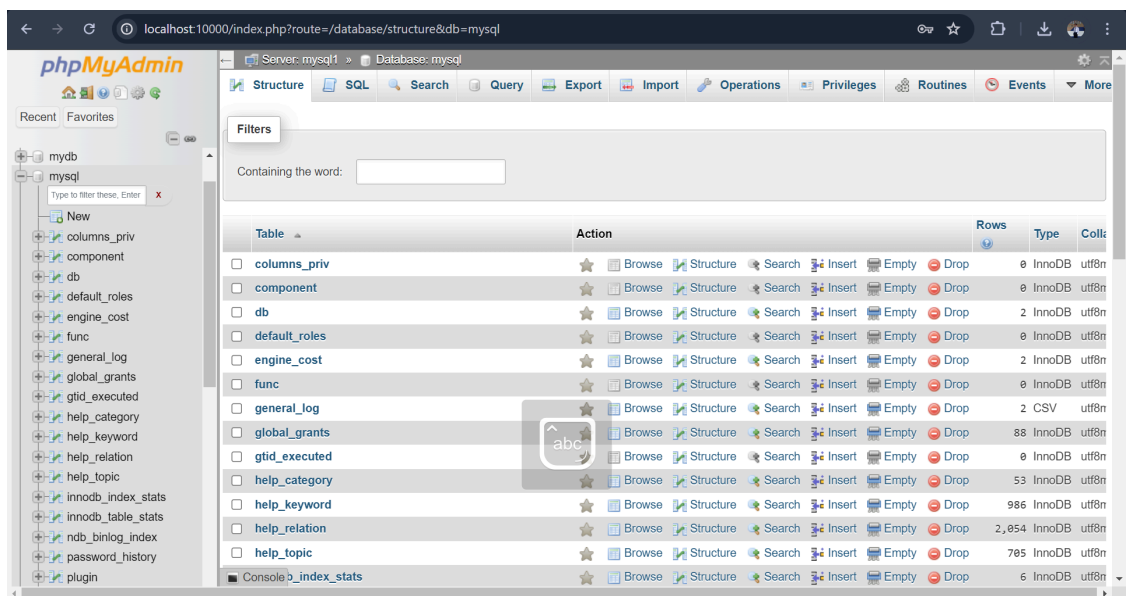
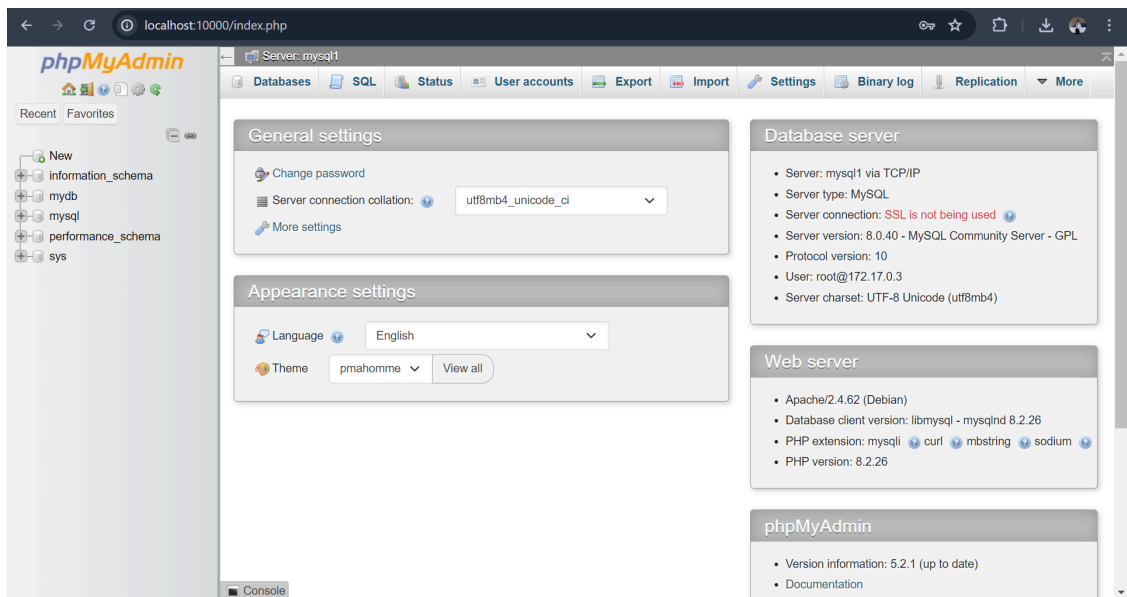
Penjelasan:

Gambar menunjukkan halaman web yang berhasil ditampilkan pada browser, menampilkan isi file **index.html** yang dilayani melalui web server Python. Halaman tersebut diakses pada port **9999**, sesuai dengan pengaturan container.

Case 3: MySQL dan phpMyAdmin dengan Docker



Tampilan antarmuka phpMyAdmin yang menunjukkan daftar database yang ada, memastikan koneksi ke MySQL server berhasil.



Tampilan halaman login phpMyAdmin dengan kredensial yang digunakan, serta bukti akses ke halaman utama phpMyAdmin setelah login.

Penjelasan:

1. Gambar pertama menunjukkan antarmuka phpMyAdmin setelah berhasil terhubung ke MySQL server yang berjalan di container. Ini memastikan bahwa MySQL server telah dikonfigurasi dan berjalan dengan benar.
2. Gambar kedua menunjukkan halaman login phpMyAdmin dengan kredensial yang sesuai dari skrip konfigurasi. Halaman ini memastikan bahwa phpMyAdmin telah berhasil digunakan untuk mengelola database MySQL.

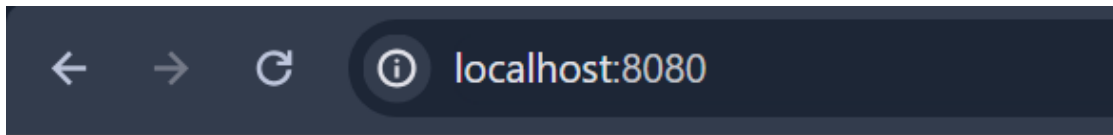
Case 4: Menggunakan Nginx sebagai Load Balancer untuk Web Server

```

root@Ricardo-S08:~/cloud/Task-CloudComputing/Task-2/case4# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
9e0179474aa1   nginx:alpine "/docker-entrypoint..." 49 seconds ago Up 49 seconds 0.0.0.0:8080->80/tcp, [::]:8080->80/tcp nginx_load_balancer
21e9c11bde2d   nginx:alpine "/docker-entrypoint..." 50 seconds ago Up 49 seconds 80/tcp                               web_server_2
e2ce3667b281   nginx:alpine "/docker-entrypoint..." 50 seconds ago Up 50 seconds 80/tcp                               web_server_1

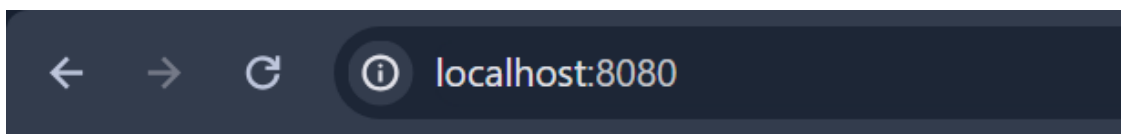
```

Tampilan terminal yang menunjukkan semua container (dua web server dan Nginx load balancer) sedang berjalan.



Welcome to Web Server 1

Tampilan browser yang menunjukkan isi halaman dari **index1.html** (Welcome to Web Server 1) diakses melalui **http://localhost:8080**.



Welcome to Web Server 2

Tampilan browser yang menunjukkan isi halaman dari **index2.html** (Welcome to Web Server 2) diakses melalui **http://localhost:8080**, memastikan load balancing berjalan dengan benar.

Penjelasan:

Gambar menunjukkan load balancer Nginx berhasil mendistribusikan permintaan HTTP secara bergantian ke dua web server yang berjalan di container. Hal ini terlihat dari tampilan halaman web yang menampilkan konten dari **index1.html** dan **index2.html** secara bergantian ketika diakses melalui port **8080**.

5. Kesimpulan

Dalam laporan ini, saya telah berhasil menjalankan tiga case sesuai dengan instruksi yang diberikan, yaitu mengumpulkan Chuck Norris jokes menggunakan background process, menjalankan web server Python, dan mengelola MySQL serta phpMyAdmin. Selain itu, saya

juga mengembangkan Case 4 untuk menerapkan load balancing menggunakan Nginx dalam Docker, yang mendistribusikan permintaan HTTP secara merata ke dua web server. Pendekatan ini dapat digunakan untuk berbagai aplikasi yang membutuhkan skalabilitas, distribusi beban, dan memastikan performa tetap stabil dalam menghadapi lalu lintas pengguna yang tinggi.