Load Balancing and Failover Mechanism Report

Dilbar Singh Lamba

1 Introduction

This report documents the implementation of a load balancing and automatic failover mechanism. The primary objective was to distribute incoming requests across multiple servers to improve reliability and prevent overload on a single server. The load balancing method used was round-robin, which cycles through available healthy servers in sequence to distribute requests evenly.

2 Implementation Details

The load balancing setup uses two main servers:

- `notdilbarsl-github-io.onrender.com`
- `notdilbarsl-github-io-1.onrender.com`

A third server, `notdilbarsl-github-io-3.onrender.com`, hosts the `config.js` file, which configures and manages both the load balancing and automatic failover. The load balancer script, written in JavaScript, checks the health of each server periodically, applies round-robin balancing among healthy servers, and handles failover if a server becomes unavailable.

2.1 Load Balancing Logic

The core of the load balancing logic is implemented in the `getNextServer` function within `config.js`. This function filters out unhealthy servers and cycles through healthy servers based on the round-robin method:

- A list of servers is maintained, each with a `healthy` status.
- Every 10 seconds, a health check runs on each server using an HTTP request to the `/health` endpoint. If the server responds with a 200 status, it is marked as healthy; otherwise, it is marked as unhealthy.
- The `getNextServer` function only considers healthy servers for load balancing. If no healthy servers are available, an error is thrown.
- For each incoming request, the selected healthy server's URL is returned, and the request is forwarded to that server.

The relevant code from `config.js` is shown below:

```
function getNextServer() {
    const healthyServers = servers.filter(server => server.healthy);
    if (healthyServers.length === 0) {
        throw new Error("No healthy servers available");
    }
    const server = healthyServers[currentServerIndex];
    currentServerIndex = (currentServerIndex + 1) % healthyServers.length;
    return server;
}
```

2.2 Automatic Failover Mechanism

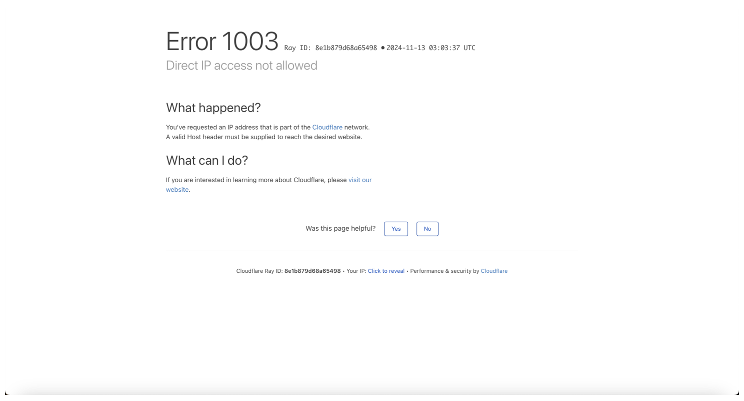
The `config.js` file also includes an automatic failover mechanism. If a server is detected as unhealthy during the periodic health checks, it is excluded from the pool of servers considered for load balancing. The following code snippet performs the health check every 10 seconds:

```
async function checkServerHealth() {
    for (const server of servers) {
        try {
            const healthResponse = await axios.get(\‘\${server.url}/health\‘);
            server.healthy = healthResponse.status === 200;
            console.log(\‘\${server.url} is healthy\‘);
        } catch (error) {
            server.healthy = false;
            console.error(\‘\${server.url} is unhealthy: \${error.message}\‘);
        }
    }
}
setInterval(checkServerHealth, 10000);
```

3 Challenges Encountered

Initially, I attempted to implement load balancing using NGINX. However, I encountered an error (Error 1003) stating ”Direct IP access not allowed” (see Figure 1). This issue occurred because Render required an explicit host header for each request on a server hosted on Render, which was not feasible in my setup, as it involved multiple backend servers.

To solve this, I developed a custom application with a single host header. This application would handle the selection of servers internally, based on the round-robin method. This approach avoided the need for NGINX's host header requirement and allowed the load balancer to forward requests correctly to each server.


Error 1003 Ray ID: 8e1b879d68a65498 • 2024-11-13 03:03:37 UTC
Direct IP access not allowed

What happened?

You've requested an IP address that is part of the Cloudflare network.
A valid Host header must be supplied to reach the desired website.

What can I do?

If you are interested in learning more about Cloudflare, please visit our website.

Cloudflare Ray ID: 8e1b879d68a65498 • Your IP: Click to reveal • Performance & security by Cloudflare

Figure 1: Error 1003 - Direct IP access not allowed

4 Conclusion

The load balancing and failover mechanism was successfully implemented using a round-robin method combined with periodic health checks. The primary improvements achieved are:

- Even distribution of traffic across multiple servers, preventing overload on any single server.
- Automatic failover that removes unhealthy servers from the load balancing rotation, improving the reliability of the application.
- Elimination of the need for NGINX by implementing a custom solution that meets the application's specific requirements for host header handling.

The current setup with two main servers and an additional server for configuration has proven to be effective in distributing requests and managing server availability. Future work could involve adding more servers to improve redundancy and further enhance reliability.