# Load-Balanced Node Apps with NGINX and Docker

## Overview

This project demonstrates a basic load distribution strategy using:

- Two Node.js applications (app1 and app2)

- An NGINX reverse proxy

- Docker containers (to run apps without pm2)

- A basic threshold-based routing logic

- A simple load-testing script (test-req) to validate request routing

When total requests cross a defined threshold (e.g., 10), traffic is automatically rerouted from Node App 1 to Node App 2.

## Project Structure

project-root/

├── app1/ # Node App 1

│ ├── Dockerfile

│ └── index.js

├── app2/ # Node App 2

│ ├── Dockerfile

│ └── index.js

├── nginx/

│ ├── Dockerfile

│ └── default.conf # NGINX reverse proxy config

├── docker-compose.yml # Multi-container orchestration

└── test-req/

└── test.js # Axios script to simulate multiple requests

## Setup Instructions

1. Clone / Prepare Your Project

Ensure your folders/files are laid out as shown above.

2. Build and Run the Containers

From the root of the project, run:

docker-compose up --build -d

This will:

- Build Docker containers for both apps and NGINX

- Start all containers in detached mode

3. Verify All Services

Run:

docker ps

Expected services:

- node-app1

- node-app2

- nginx

## How It Works

Node App 1:

- Listens for incoming HTTP requests

- Maintains a request counter (in-memory)

- If request count is ≤ threshold (e.g., 10), it responds itself

- If request count is > threshold, it proxies the request to App 2

Node App 2:

- Just handles and responds to incoming requests

- Used when load crosses threshold

NGINX:

- Reverse proxies all requests to App 1 at the start

- Could be extended to handle load balancing as needed

Note: The request counter resets if the container restarts. This is acceptable for demo/testing purposes.

## How to Test (Using Axios in test-req/)

1. Navigate to Test Folder

cd test-req

2. Install Dependencies

npm install axios

3. Run the Load Simulation Script

node test.js

Example Output:

[Request 1] Response from App 1

[Request 2] Response from App 1

...

[Request 11] Response from App 2

[Request 12] Response from App 2

This verifies:

- First 10 requests are handled by App 1

- Requests beyond the threshold go to App 2

## Optional Enhancements

- Use Redis or shared memory to manage request counters across instances

- Integrate NGINX’s built-in load-balancing for true horizontal scaling

- Add metrics/logging to monitor which app served the request