Technical Report: Asynchronous HTTP Load Balancer with Docker

Project Name: Async-load-balancer

Developer: Romina Farhad

GitHub Repository: https://github.com/rominafarhad/Async-load-balancer.git

Date: February 2026

1. Introduction

This project implements a high-performance, asynchronous HTTP Load Balancer using Python. The primary objective is to distribute incoming network traffic across multiple backend servers to ensure high availability, scalability, and optimal resource utilization. The system is fully containerized using Docker, simulating a modern microservices architecture.

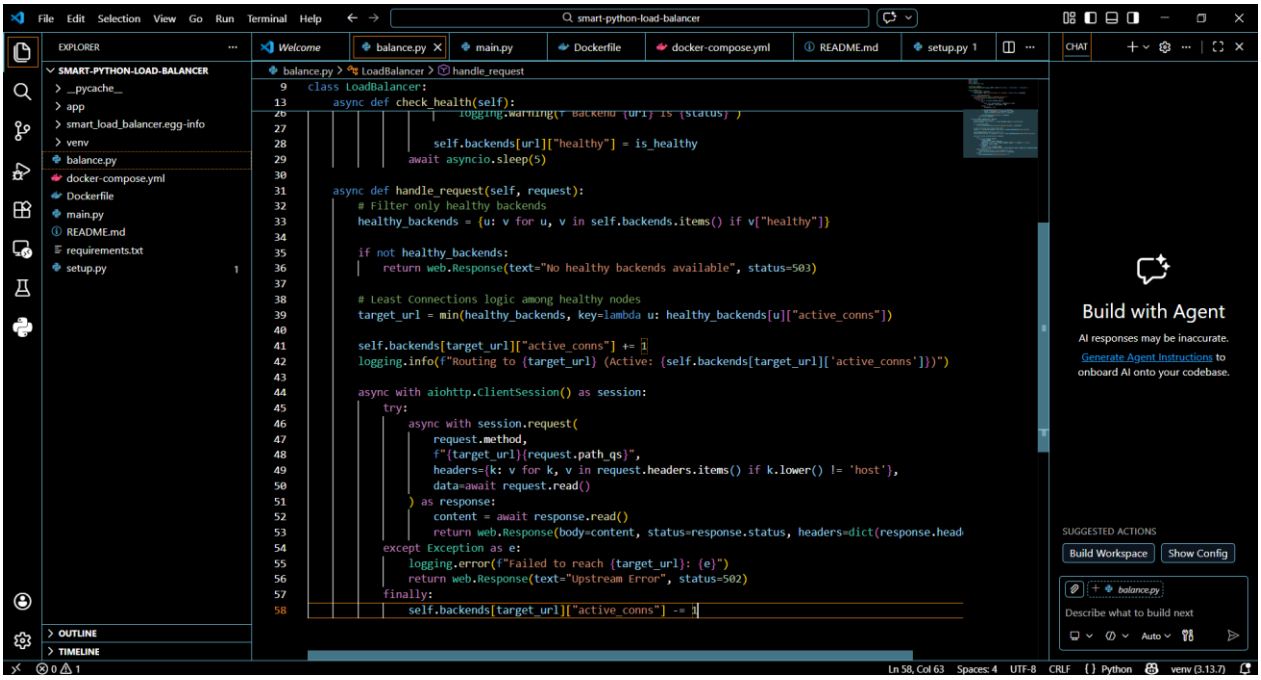
2. System Architecture

The architecture consists of three main components:

Load Balancer (Gateway): Built with aiohttp and asyncio, acting as the entry point (Port 8080).

Backend Servers: Two independent Python-based HTTP servers acting as upstream nodes.

Docker Network: A dedicated bridge network for secure inter-container communication.



3. Key Features

3.1. Least Connections Algorithm

Unlike basic Round-Robin, this project implements the Least Connections strategy. The load balancer tracks the number of active requests for each backend and routes new traffic to the server with the lowest current load.

3.2. Health Check Monitoring

The system features a background task that probes backends every 5 seconds. If a server becomes unresponsive, it is automatically flagged as DOWN and removed from the routing pool until it recovers.

3.3. Asynchronous Execution

By utilizing Python's asyncio loop, the load balancer can handle hundreds of concurrent requests without blocking, making it highly efficient for I/O-bound tasks.

4. Implementation Details

The core logic is divided into:

balance.py: Contains the LoadBalancer class and the selection logic.

main.py: Handles the web server lifecycle and background health checks.

docker-compose.yml: Orchestrates the deployment of the entire stack.

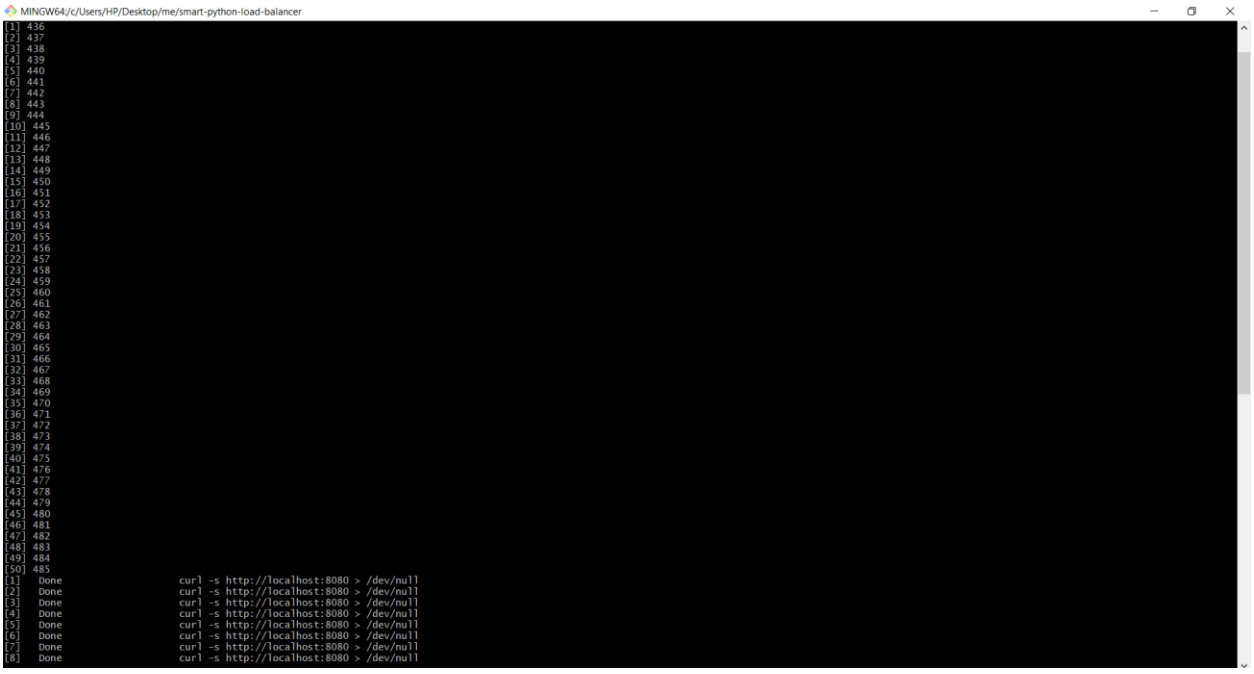
5. Testing and Results

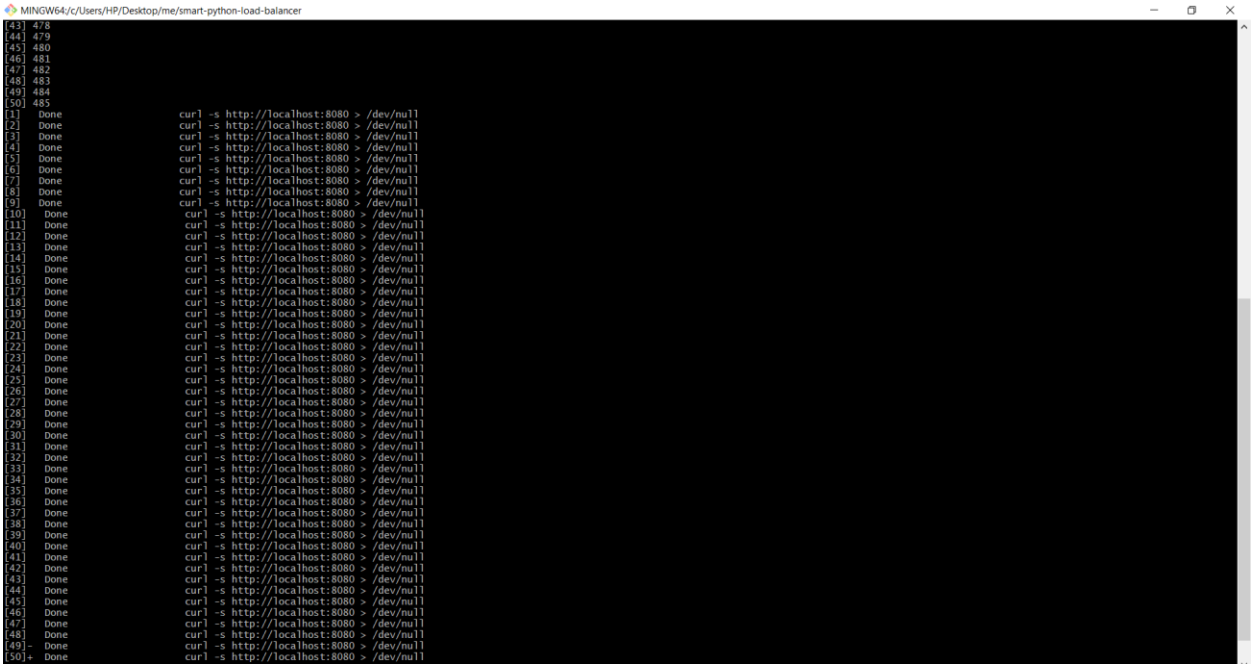
5.1. Deployment

The environment was deployed using Docker Compose. All services were initialized correctly within a virtual network.

5.2. Stress Testing (Concurrency)

To evaluate the system under heavy load, a stress test was performed by sending 50 concurrent HTTP requests using a Bash script. The load balancer successfully managed the traffic, maintaining a 100% success rate (HTTP 200).





5.3. Fault Tolerance Test

During the execution, one backend was manually stopped. The Load Balancer immediately detected the failure, logged a warning, and redirected all traffic to the remaining healthy node without any downtime for the end-user.

6. Conclusion

The Async-load-balancer successfully demonstrates the power of asynchronous programming and containerization. By implementing intelligent routing and health monitoring, the system provides a robust solution for managing web traffic in a distributed environment.