Load Balancer.
Reverse Proxy for a .NET Microservice.

Introduction

This document outlines the implementation of load balancing within a to-do list management system. The goal was to efficiently manage a high volume of incoming requests. To achieve this, a load balancer was integrated to evenly distribute workload across multiple servers. Leveraging the YARP (Yet Another Reverse Proxy) package in .NET, the system enhances resilience and fault tolerance while improving scalability and performance, particularly during peak usage periods.

Components

The application is built using a stack of cutting-edge technologies, including .NET, PostgreSQL, and Docker, to ensure robustness, scalability, and flexibility. Postman serves as the client application for testing and interacting with the backend APIs, providing a user-friendly interface for developers to validate functionality.

Additionally, pgAdmin4 is utilized as a powerful tool for viewing and managing the database's contents, offering insights into data structures and facilitating administrative tasks. Visual Studio 2022 [6] serves as the integrated development environment (IDE) for C#, offering a comprehensive suite of tools and features to streamline code development, debugging, and deployment processes. The command line plays an indispensable role in project management, enabling seamless startup and configuration procedures. Moreover, the entire development workflow is orchestrated using git, a distributed version control system, ensuring collaborative development, version tracking, and code integrity throughout the development lifecycle.

Startup Steps

1. Install every needed tool : Visual Studio, .NET 6, Docker, Postman, pgAdmin4.
2. Open the terminal at the docker compose file's location.

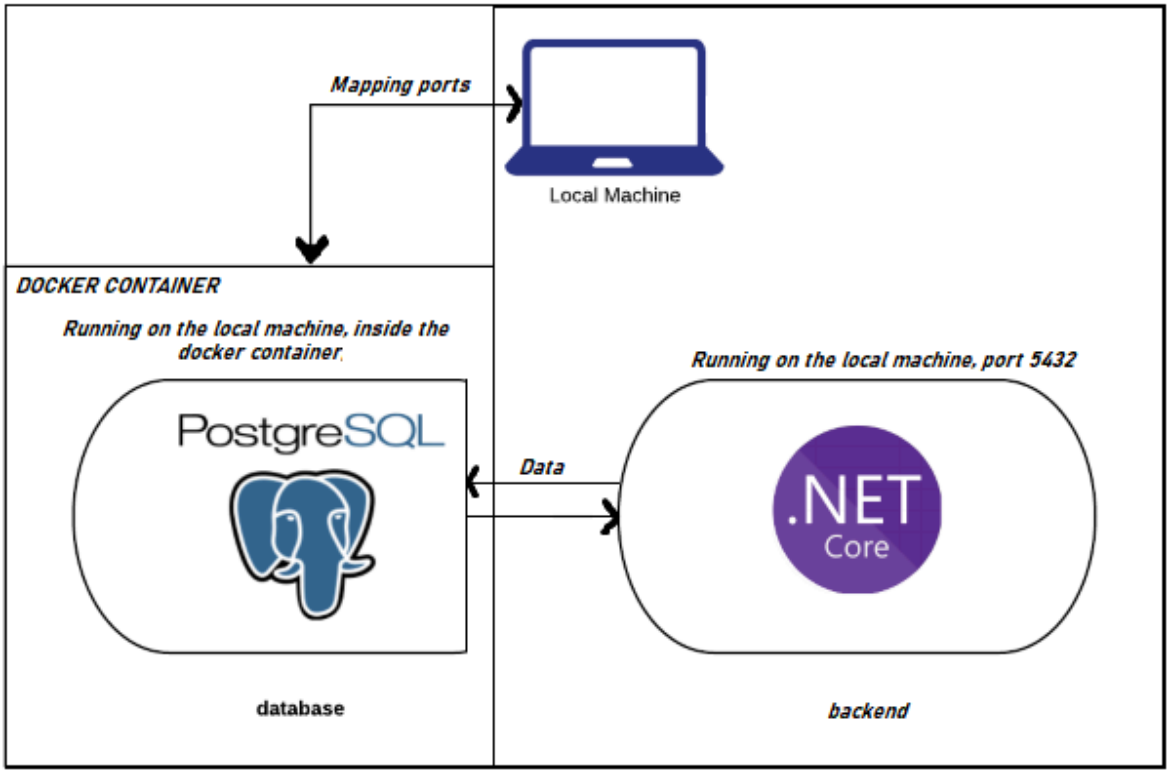
Run "docker-compose build".

3. Run "docker-compose up". (do not close the terminal)
4. Open Visual Studio solution for the YARP project. Start the YARP project.
5. Open pgAdmin4 to view the database' structure and present data.
6. Open the postman and make any get call (recommended getAll call).

Observation : *Make sure that the database is not empty.*

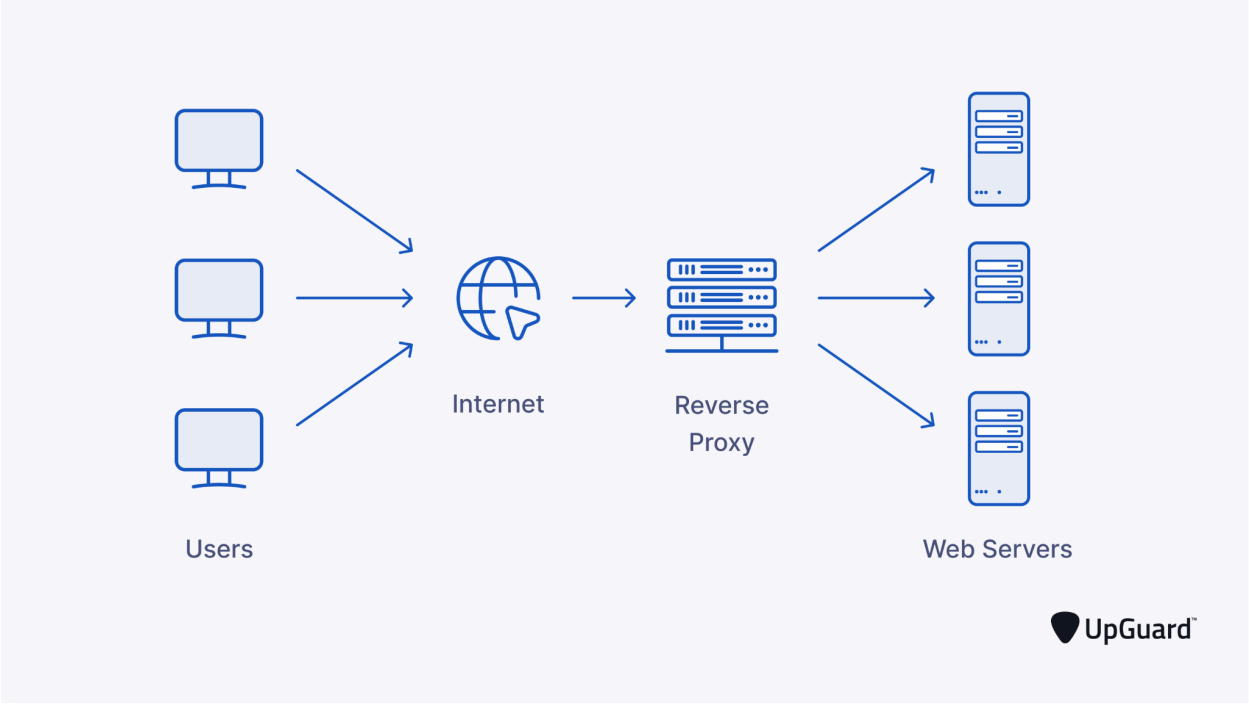
Load Balancer (YARP)

The microservice, built on *.NET 6*, provides CRUD operations for managing todo items, the most operations being GET methods. *PostgreSQL* serves as the backend's database, offering reliability and scalability, the fact that it is a very thought choice being defined by it's open source nature. *Docker* containers ensure consistency and portability across different environments, simplifying deployment and management. *The YARP package* in .NET facilitates load balancing, distributing incoming requests across multiple microservice instances. As for the frontend part, we can use any client (e.g Postman [[1]]) that allows us to send http calls to our service.



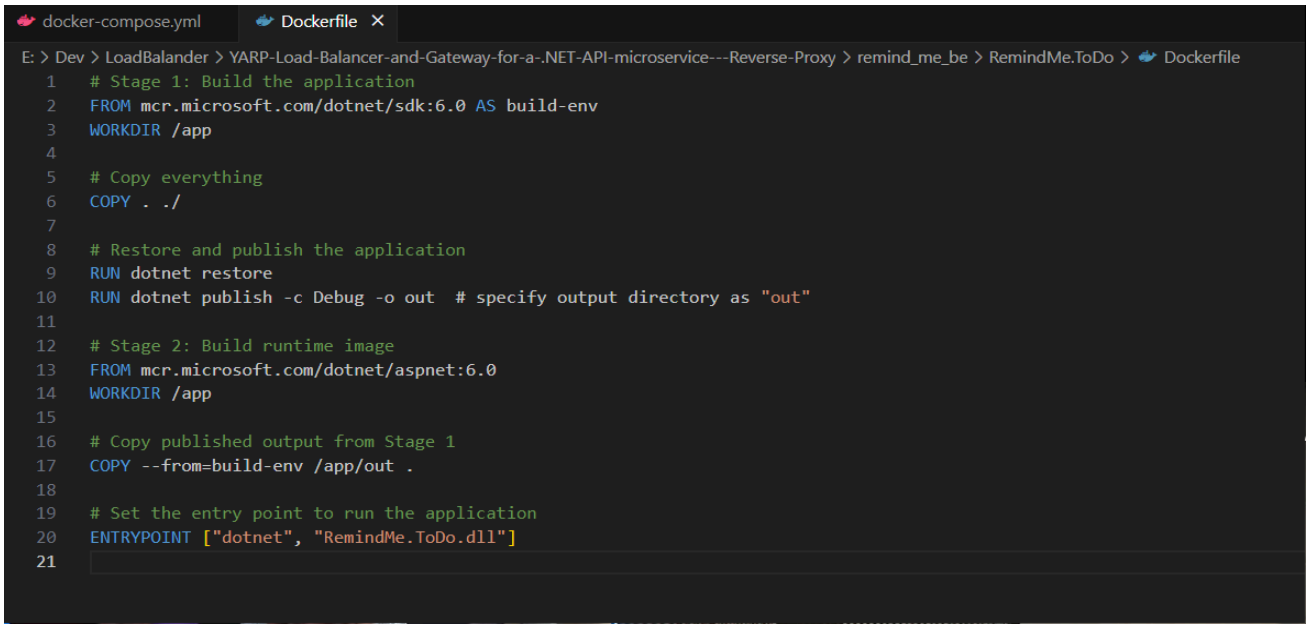
1. Project Architecture Overview

The distribution of HTTP requests is a fundamental aspect of modern web architecture, facilitated by reverse proxies that intelligently route incoming client requests across multiple backend servers. For this project, the architecture was designed following the next flow : Many users can access and manipulate data through the same application. It will efficiently handle all the workload, in order to have a fluent feeling for the end user, by duplicating the backend server using Docker. The distribution of workload is done using "*Round-robin scheduling algorithm*" [2]. The chosen algorithm empathizes that all instances have the same resources allocated and must take an equal amount of work.



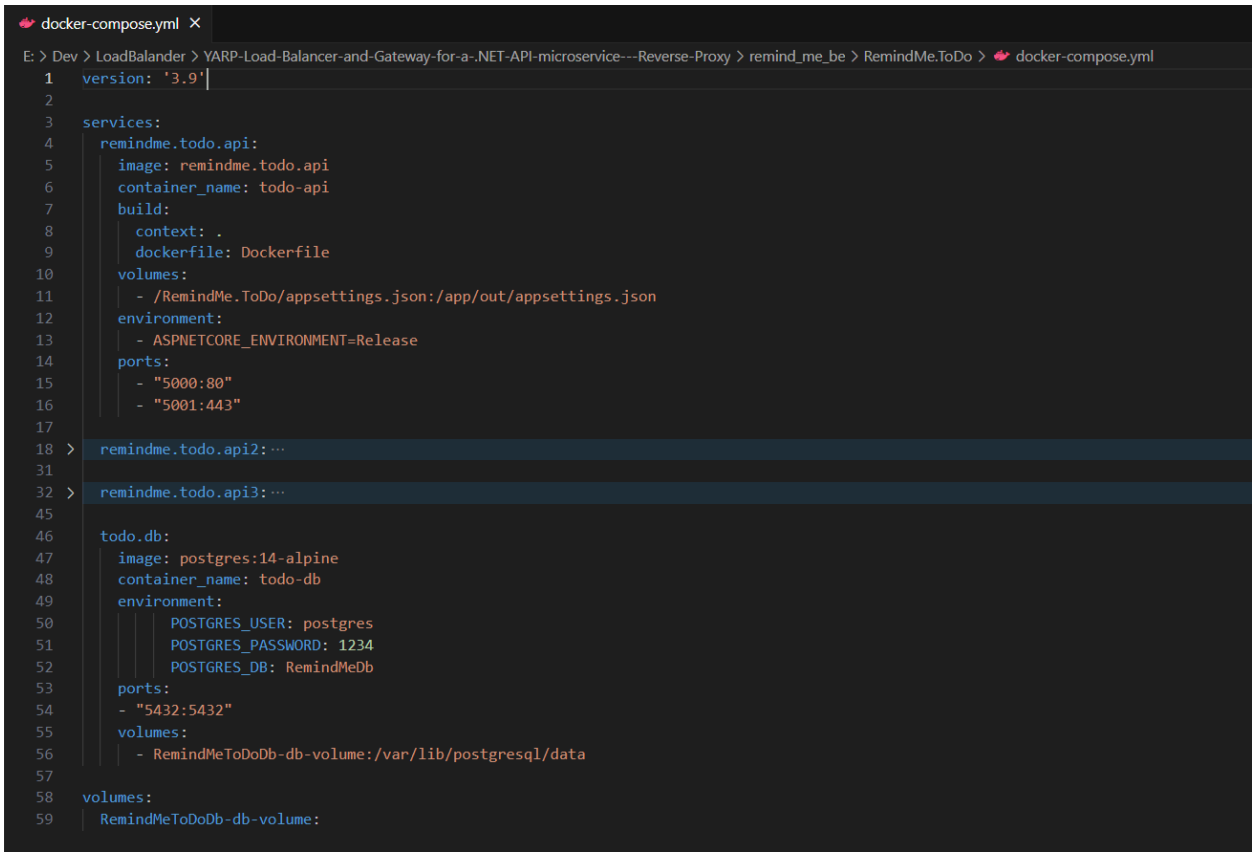
2. Reverse Proxy Server Illustration [3]

Three is the exact number of instances that are open, but it is as flexible as the need is. The setup file is called Dockerfile. A Dockerfile is a text file that contains a set of instructions used to build a Docker image. It serves as a blueprint for creating a Docker container, defining the environment, dependencies, and configurations required for the application to run. Dockerfiles typically include commands to specify the base image, copy files into the container, install software packages, set environment variables, and configure runtime settings. Once a Dockerfile is written, it can be used with the docker build command to generate a Docker image, which can then be instantiated as a container to run the application in an isolated and portable environment. This approach enables developers to package applications and their dependencies into lightweight, portable containers, making it easier to deploy and manage software across different environments.



3. Docker file [4]

The configuration file is a docker compose file. A Docker Compose file is a YAML (YAML Ain't Markup Language) file used to define multi-container Docker applications. It allows developers to specify the configuration of multiple services, networks, and volumes in a single, easy-to-read file. Docker Compose simplifies the process of managing complex applications by enabling users to define all the components required for their application to run in one place. Within a Docker Compose file, you can specify the services your application needs, including their Docker images, ports, environment variables, dependencies, and any other configuration options. Additionally, Docker Compose allows you to define networks and volumes for your services, enabling communication between containers and persisting data. Once the Docker Compose file is defined, you can use the docker-compose command-line tool to start, stop, and manage your multi-container application with a single command. This makes it easier to develop, test, and deploy complex applications in Docker environments.



4. Docker compose file [5]

Observation *: Visual Studio Code was used for handling docker file and docker compose file. [7]*

Conclusion

In conclusion, the implementation of load balancing, facilitated by YARP as a reverse proxy, significantly enhances the scalability and reliability of our microservice architecture. By evenly distributing incoming requests across multiple microservice instances, our system efficiently manages workload fluctuations, ensuring optimal performance and fault tolerance, particularly during peak usage periods. This approach not only improves the end-user experience by providing a smooth and responsive application but also enhances the overall resilience and robustness of our system. Thus, incorporating load balancing mechanisms like YARP is crucial for building robust microservice architectures, we have established a solid foundation for building scalable and high-performing microservices that meet the demands of modern web applications.

Bibliography

1. https://www.postman.com/
2. https://en.wikipedia.org/wiki/Round-robin_scheduling
3. https://www.upguard.com/blog/what-is-a-reverse-proxy
4. https://docs.docker.com/reference/dockerfile/
5. https://docs.docker.com/compose/
6. https://visualstudio.microsoft.com/downloads/
7. https://code.visualstudio.com/?wt.mc_id=vscom_downloads