

ASSESSMENT 1:

Load Balancing

and Docker

COMP3004

Advanced Computing and

Networking Infrastructures

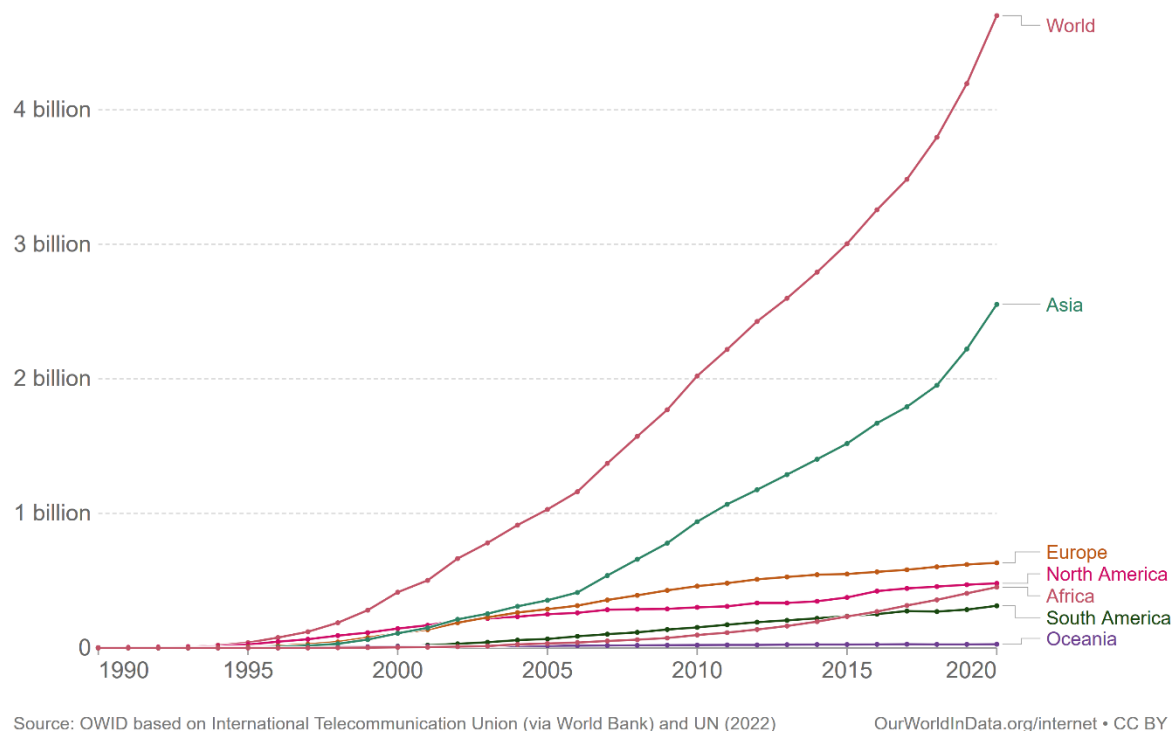
2022/2023

Table of Contents

1. Introduction	2
2. Experiment and Setup.....	4
Key Concepts.....	4
System Design	5
System Setup.....	5
File Structure	6
3. Automation Process	7
4. Results and Discussion	10
5. Conclusions	12
References	12

1. Introduction

The Internet is one of the fastest growing technologies, having amassed 4.7 billion regular users worldwide (Roser et al., 2015). Providing access to the World Wide Web, people across the world can access websites/platforms that serve a range of purpose (i.e., online shopping, entertainment, online communication).



1. **Internet user:** An internet user is defined by the International Telecommunication Union as anyone who has accessed the internet from any location in the last three months. This can be from any type of device, including a computer, mobile phone, personal digital assistant, games machine, digital TV, and other technological devices.

Figure 1: Number of people who used the Internet in the last three months (Roser et al., 2015)

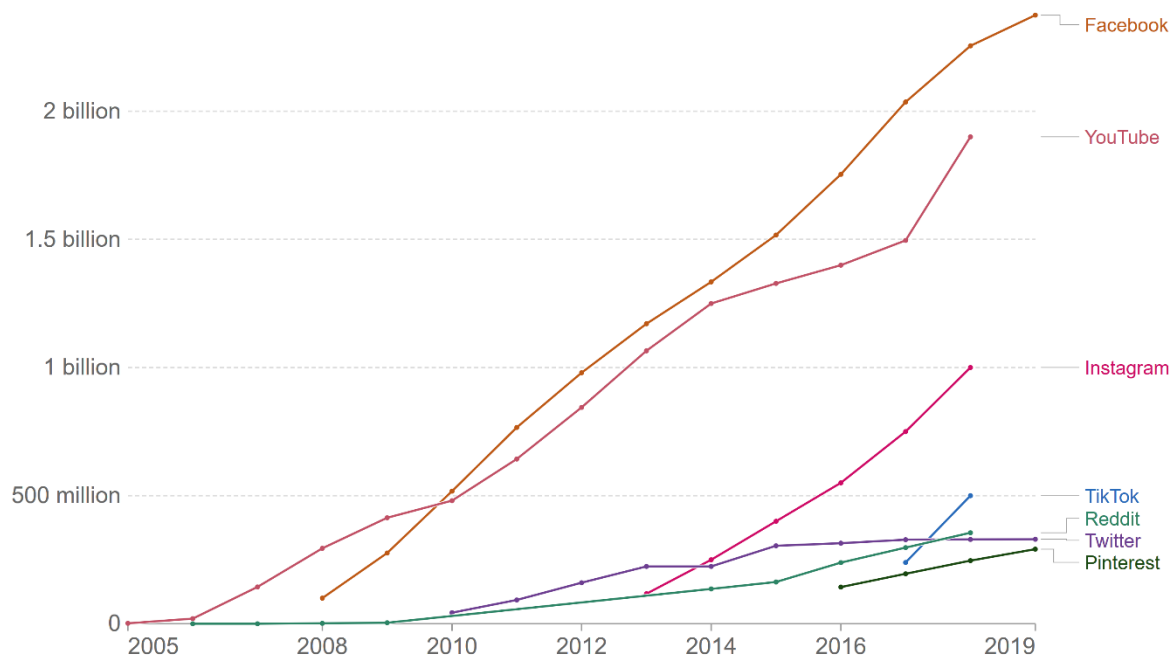
Media Service Providers are an example of the distribution of entertainment across the Web, wherein digital media (in the forms of video, image, and/or audio) is uploaded to the Internet, providing users global access to digital content. Notable examples of these services include Netflix, YouTube, and Spotify.

As the Internet grows, so do the media platforms that are hosting content, see Figure 2. With more and more users simultaneously using these platforms, performance becomes a priority, as the load on the backend web servers increases. HTTP requests become slow as response times increase. Errors also become more frequent, wherein HTML content might not be loaded fully. For platforms that are trying to entertain users, having seconds of 'delay' results in users moving to competitors' service.

The reason for surges in user activity could be related to content on the platform going 'viral' or because of global events like COVID-19 restricting individuals from in-person socialisation, resulting in them going online and using web platforms to entertain themselves.

Number of people using social media platforms, 2005 to 2019

Estimates correspond to monthly active users (MAUs). Facebook, for example, measures MAUs as users that have logged in during the past 30 days. See source for more details.



Source: Statista and TNW (2019)

OurWorldInData.org/internet • CC BY

Figure 2: Number of people using social media platforms, 2005 to 2019 (Roser et al., 2015)

When servers are under high stress and are performing very ineffectively, or even completely offline, the website will generate less revenue, and profits will decrease. Because of this loss, hackers might be motivated to try to intentionally take down these web servers via brute force DDoS attacks – wherein servers are flooded with ‘botted’ web traffic. Akamai (2018) reported that there was a 16% increase in recorded DDoS attacks between 2017 and 2018. Likewise, within the year 2022 alone, out of 573 UK-based businesses surveyed, 10% identified occurrences of denial-of-service attacks (“2022 in review: DDoS attack trends and insights”, 2023).

This threat of downtime incentivises companies to ensure that their websites have built in redundancy and systems to help servers maintain optimal throughput. This study aims to research one such method used to improve throughput – *Load Balancing*. This technology is used to distribute web traffic among hardware resources and the end goal is to reduce the stress on individual nodes (i.e. application servers). The benefit that this provides is that nodes are not overloaded as traffic will be distributed amongst less busy servers, which theoretically reduces response time (Zhang & Fan, 2008).

This report is organised as follows. Section 2 discusses the design and purpose of the experimental system. Section 3 highlights the process of automating *Load Balancing*. Section 4 outlines and justifies the results, explaining their findings. Section 5 concludes the experiment with a summary of the whole process.

2. Experiment and Setup

The purpose of this experiment is to demonstrate the usage of Load Balancing, Docker, and Automation methods. The goal of this demonstration is to overall determine the capabilities of using a Nginx web server as a load balancer and to evaluate the limitations of this approach. As mentioned within the Introduction of this report, this demonstration could potentially be applied to real-world systems to try to improve throughput of web servers and reduce latency for user clients.

Key Concepts

Within this project, these tools will be used to ensure that all initial packages and dependencies are installed on the host system, as well as to automate the process of creating containers, or to perform specific tasks.

Docker

A framework that enables platform developers to package their written code, and all required dependencies, into *containers* (*What is a container?*, 2023). Containerized applications are advantageous because packaging all dependencies alongside the code enables portability – applications can be run on different machines without needing additional setup. Docker will be used within this experiment to ensure that the same testing environment can be replicated across different machines.

Nginx

An open-source web server that will be used within this experiment for its HTTP *Load Balancing* functionalities. As a basic HTTP web server, nginx can be used to serve static html files (*What is Nginx?*, 2021).

Shell Scripting and Automation

Shell scripting is the process of writing shell commands within a .sh file that can then be compiled and executed to run all commands within the script file. Writing shell scripts reduces the time for developers to complete a task, and the scripts can be amended easily.

Whilst shell scripts can be used to automate the initial system setup, Docker can also be integrated into this process. *Dockerfile* is a .TXT document that contains all commands required to build an image. Docker will automatically build these images by reading the commands within this file. *Docker-Compose* is a .YAML file that is used to automatically define and run multiple docker containers.

System Design

Figure 3 presents the design of the system that this report discusses. In this design concept, the Load Balancer server is the primary method of access for the user to interface with the back-end of the application. In the experiment, this Load Balancer will be implemented via the open-source Nginx web server software. Workers 1, 2, and 3 are web application server nodes that the Load balancer will distribute incoming traffic towards. These three nodes will each be hosting their own instance of a web application. In this design, all three worker nodes connect to a database server wherein data can be stored. The database will be used to demonstrate dynamic html content.

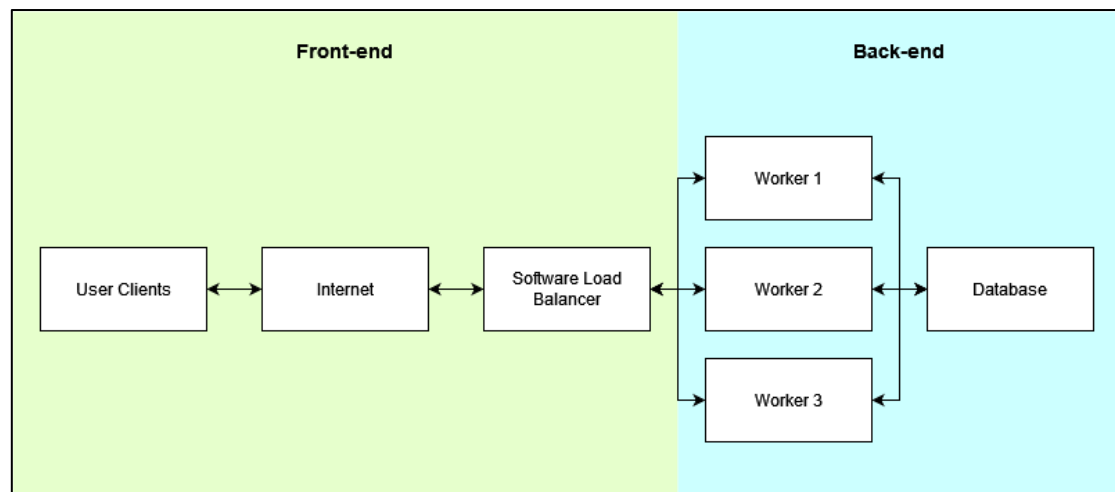


Figure 3: System diagram.

In this design, each server node will exist as its own Docker container, and each container will only provide one service. Therefore, this design ensures that one container is responsible for one specific component of the application.

System Setup

To carry out this experiment, an Ubuntu Linux Server (18.04) has been installed as a Virtual Machine.

To initially setup the system and ensure that all needed packages are installed onto the system, a shell script file has been supplied (Figure 4). This file contains commands to update and/or install packages that are needed to create the design in Figure 3.

The command **sudo apt-get update** refreshes the package lists to see if currently installed packages have newer versions.

The last three lines within the script are the most important as they install the three following packages:

- Docker
- Docker-Compose
- Nginx

```

GNU nano 2.9.3      init.sh

#!/bin/bash

sudo docker rmi $(sudo docker images -f "dangling=true" -q)

sudo apt-get update
sudo apt -y install curl
sudo apt-get -y install openssh-server

sudo apt-get -y install docker.io
sudo apt -y install docker-compose
sudo apt-get -y install nginx

```

Figure 4: init.sh shell script.

The script in Figure 4 can be compiled with the command **sudo chmod +x test.sh** and then executed by running **./init.sh**

Compiling and running the scripted code will run all the commands inside the file and all packages required for this experiment will be installed automatically.

File Structure

To ensure that the code is organised, the files have been organised into folders to create the resulting structure in Figure 5. The technicalities of each file will be explained within further sections of the report.

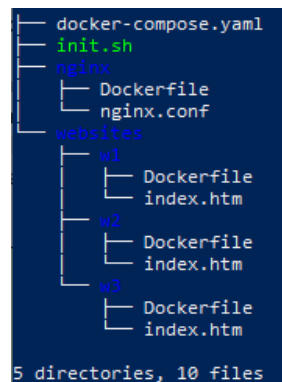


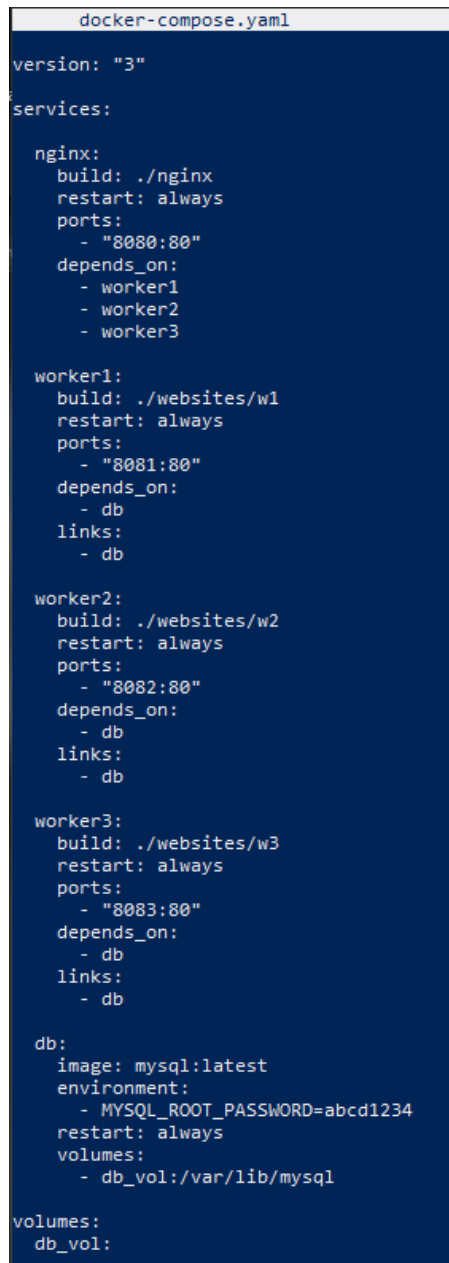
Figure 5: Project file structure.

Within the root of this repository, are the files which are used to build the project – **init.sh** and **docker-compose.yaml**

The **nginx** folder contains the files necessary to establish the load balancing aspect of the project, whilst the **websites** folder contains the files necessary to build the three backend web application servers that users will be distributed amongst. Each of the **index.htm** files contain the same app, but have a html <title> tag, which for the purpose of this experiment will highlight whether the load balancing implementation is running.

3. Automation Process

The docker-compose.yml file (Figure 6) can be found within the root of the project directory. The purpose of this file is to automatically build each of the 'services' within as Docker containers, based upon the attributes within the YAML file.

A screenshot of a code editor showing the content of a file named 'docker-compose.yml'. The file is displayed on a dark blue background with light blue text. The content is a YAML configuration for Docker Compose, defining five services: nginx, worker1, worker2, worker3, and db. Each service has specific build, restart, port, and dependency settings. The db service uses a pre-built image and has an environment variable and volume defined. A volume named db_vol is also defined at the bottom.

```
docker-compose.yml

version: "3"

services:

  nginx:
    build: ./nginx
    restart: always
    ports:
      - "8080:80"
    depends_on:
      - worker1
      - worker2
      - worker3

  worker1:
    build: ./websites/w1
    restart: always
    ports:
      - "8081:80"
    depends_on:
      - db
    links:
      - db

  worker2:
    build: ./websites/w2
    restart: always
    ports:
      - "8082:80"
    depends_on:
      - db
    links:
      - db

  worker3:
    build: ./websites/w3
    restart: always
    ports:
      - "8083:80"
    depends_on:
      - db
    links:
      - db

  db:
    image: mysql:latest
    environment:
      - MYSQL_ROOT_PASSWORD=abcd1234
    restart: always
    volumes:
      - db_vol:/var/lib/mysql

volumes:
  db_vol:
```

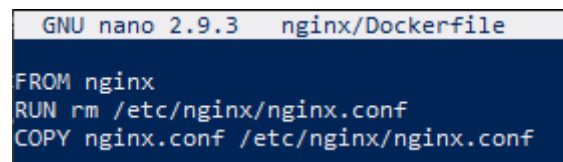
Figure 6: Project file structure.

In this file, there are five services, each of which will be built as its own container: nginx, worker1, worker2, worker3, and db. The first four services define a build attribute which specifies that the docker engine should build these services according to the Dockerfile within the specified directory. Dockerfiles are text files that contain commands which are used to build an image.

The docker-compose file also specifies the specific ports for each container, for example the nginx container can be accessed at port 8080, and workers on ports 8081, 8082, and 8083.

Each worker is linked to the db service which is built from the latest mysql image. Each worker also depends on this db service – if the db is not running, the worker containers will not be built. Each worker is also built from one of the website directories (see Figure 5).


The nginx container (which will be used as the load balancer) depends on the workers being running first. This service is built from the Dockerfile in Figure 7.

A screenshot of a terminal window with a dark blue background. The title bar at the top reads "GNU nano 2.9.3 nginx/Dockerfile". The terminal shows three lines of text: "FROM nginx", "RUN rm /etc/nginx/nginx.conf", and "COPY nginx.conf /etc/nginx/nginx.conf".

```
GNU nano 2.9.3 nginx/Dockerfile
FROM nginx
RUN rm /etc/nginx/nginx.conf
COPY nginx.conf /etc/nginx/nginx.conf
```

Figure 7: nginx/Dockerfile

These three commands specify that the default **nginx.conf** file on the system should be replaced with a new **nginx.conf** which specifically implements the load balancing behaviour (see Figure 8).

A screenshot of a terminal window with a dark blue background. The title bar at the top reads "GNU nano 2.9.3 nginx/nginx.conf". The terminal shows the following configuration: "events {worker_connections 768;}", "http {" followed by an "upstream loadbalancer {" block containing three "server" lines with IP addresses and ports 8081, 8082, and 8083. This is followed by a "server {" block with "listen 80;" and "root /nginx". Inside the server block is a "location / {" block with "proxy_pass http://loadbalance;". The configuration ends with closing braces for the location, server, and http blocks.

```
GNU nano 2.9.3 nginx/nginx.conf
events {worker_connections 768;}
http {
    upstream loadbalancer {
        server 192.168.127.128:8081;
        server 192.168.127.128:8082;
        server 192.168.127.128:8083;
    }

    server {
        listen 80;
        root /nginx

        location / {
            proxy_pass http://loadbalance;
        }
    }
}
```

Figure 8: nginx/Dockerfile

The **nginx.conf** file establishes that users accessing the app should be distributed amongst the services on ports 8081, 8082, and 8083, of which worker1, worker2, and worker3 are running. The users will each see the same app but be on a different server.


```

GNU nano 2.9.3 websites/w1/Dockerfile
FROM nginx
COPY index.htm /usr/share/nginx/html/index.html

```

Figure 9: websites/w1/Dockerfile (same for each worker).

worker1, worker2, and worker3 are all built from their corresponding Dockerfiles, but each file has the same commands. These commands specify that the nginx index.html should be replaced with the webapp page that is to demonstrate load balancing.

To build the application via the docker-compose file, the following command is input into the command line: **sudo docker-compose-up**

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
923fa7764cf3	coursework_nginx	"/docker-entrypoint...."	21 seconds ago	Up 20 seconds	0.0.0.0:8080->80/tcp, :::8080->80/tcp	coursework_nginx_1
318decf8c3c1	coursework_worker1	"/docker-entrypoint...."	24 seconds ago	Up 21 seconds	0.0.0.0:8081->80/tcp, :::8081->80/tcp	coursework_worker1_1
a11f26a12500	coursework_worker2	"/docker-entrypoint...."	24 seconds ago	Up 20 seconds	0.0.0.0:8082->80/tcp, :::8082->80/tcp	coursework_worker2_1
4cf852d01516	coursework_worker3	"/docker-entrypoint...."	24 seconds ago	Up 22 seconds	0.0.0.0:8083->80/tcp, :::8083->80/tcp	coursework_worker3_1
9Fcdc908f9c1	mysql:latest	"docker-entrypoint.s..."	25 seconds ago	Up 24 seconds	3306/tcp, 33060/tcp	coursework_db_1

Figure 10: containers built via sudo docker-compose up.

Figure 10 shows the containers that are automatically built by the docker engine. All five services that were written within the yaml file can be seen here.

4. Results and Discussion

375 words. Discuss: (evidence to show that the system works as you have designed/configured. This should include screenshots, plus descriptions/explanations, on e.g., load balancing and automation. Discussions should also cover what limitations are for the approaches taken)

The app can be accessed at the following URL, <http://192.168.127.128:8080/> and specifically accessing the website at port 8080 directs the user to the server managing the load balancing. Within each variation of the app, the <title> tag is different. The expected behaviour to be displayed is that upon page refresh, the title of the webpage changes.

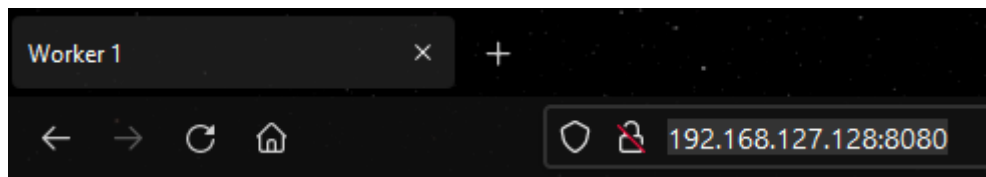


Figure 11: Website URL and title.

Upon refreshing the page, the title Worker 1 changes to Worker 2 and the url stays the same, proving that load balancing is in effect (Figure 12).

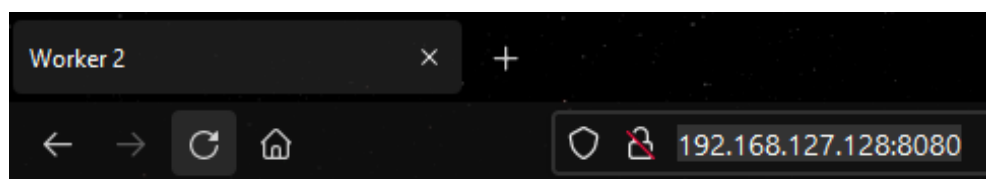


Figure 12: First refresh – load balancing changing server from worker1 to worker2.

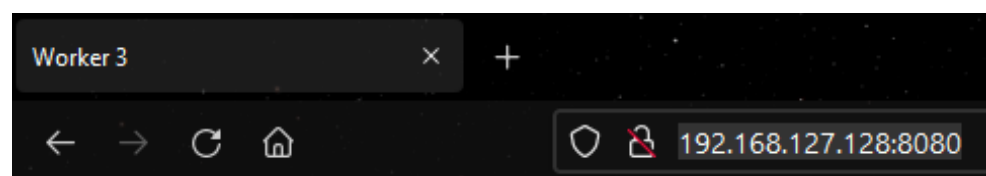


Figure 13: Second refresh – load balancing changing server from worker2 to worker3.

It can be understood that as the server changes from 1 to 2 to 3 and then back to 1, in a cyclical manner, that the load balancing is being implemented as Round-Robin.

Benchmarking

Figure 14 shows that when the app is tested to run 25 requests, with 5 per second that the application performs highly well. This remains consistent when a high load is tested, with 2000 total requests.

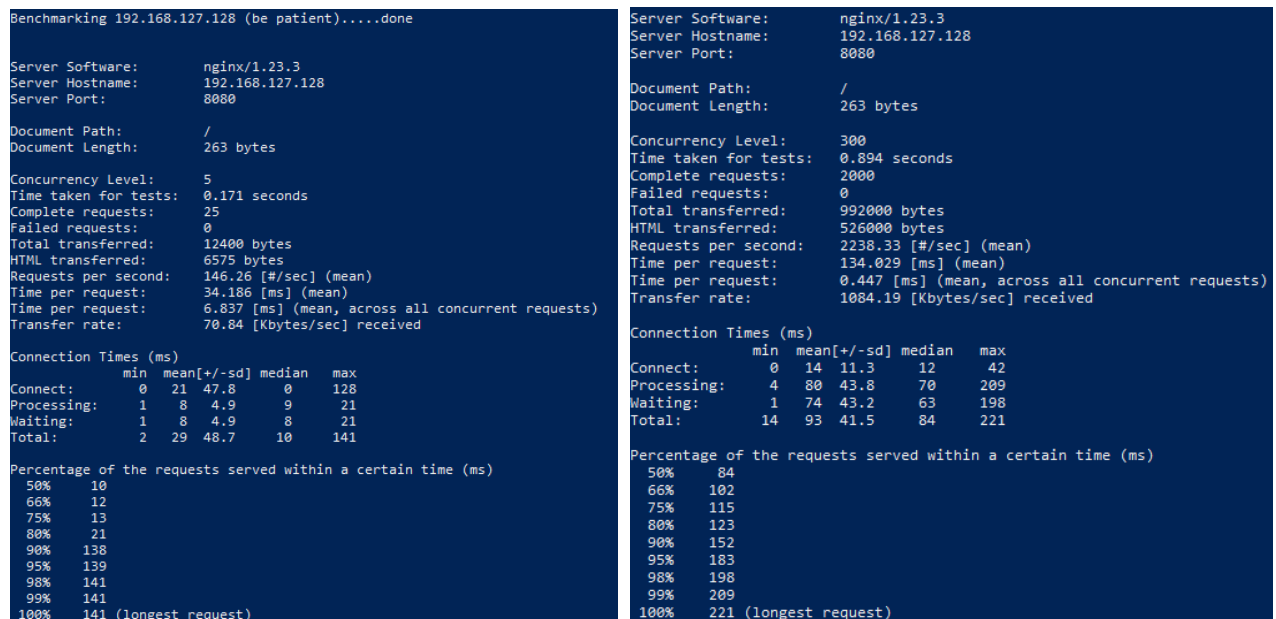


Figure 14: Benchmarking using APACHEBENCH; low stress (left); high stress (right).

Limitations

A limitation with the round robin approach is that the implementation is not making a decision according to server health and current load, but is instead assuming that all servers are equal. This method is highly simple but not as effective. A much better method would be to direct new users to the least populated backend server.

Another limitation with this experiment is that to amend the application (in this case index.html, see Figure 5), each directory would need to be updated and this could get difficult to maintain.

One theoretical limitation that has not been explored within this experiment is that the Nginx Load Balancer server is a single point of failure – if this server has technical issues, the entire application will be offline.

5. Conclusions

Overall, load balancing can be used to distribute users amongst backend servers, especially in circumstances wherein the same app is being hosted on a server. This could potentially help with reducing latency for end users and stopping servers from being overfilled with traffic, or even to help mitigate DDoS attacks.

References

- “Akamai State of the Internet / Security Summer 2018” (2018) *Akamai* [Preprint]. Available at: <https://www.akamai.com/newsroom/press-release/akamai-releases-summer-2018-state-of-the-internet-security-report> (Accessed: March 18, 2023).
- “Cyber security breaches survey 2022” (2022) *GOV.UK*. Available at: <https://www.gov.uk/government/statistics/cyber-security-breaches-survey-2022/cyber-security-breaches-survey-2022#chapter-5-incidence-and-impact-of-breaches-or-attacks> (Accessed: March 18, 2023).
- Roser, M., Ritchie, H. and Ortiz-Ospina, E. (2015) *Internet, Our World in Data*. Available at: <https://ourworldindata.org/internet> (Accessed: March 20, 2023).
- What is a container?* (2023) *Docker*. Available at: <https://www.docker.com/resources/what-container/> (Accessed: March 20, 2023).
- What is Nginx?* (2021) *NGINX*. Available at: <https://www.nginx.com/resources/glossary/nginx/> (Accessed: March 20, 2023).
- Yosifova, V. *et al.* (2019) “Trends review of the Contemporary Security Problems in the Cyberspace,” *Proceedings of the 9th Balkan Conference on Informatics* [Preprint]. Available at: <https://doi.org/10.1145/3351556.3351560>.
- Zhang, Z. and Fan, W. (2008) “Web server load balancing: A queueing analysis,” *European Journal of Operational Research*, 186(2), pp. 681–693. Available at: <https://doi.org/10.1016/j.ejor.2007.02.011>.