



Linux Webservers

Milestone 1

Bachelor in Applied Computer Science

Rob Verbeek

Academic year: 2023-2024

Campus : Geel.

PREFACE

The goal of this document is to explain the steps I took to achieve the first milestone on the project for Linux Webservices. I will go through each component that was used to get the desired results and how generative AI helped me in this process. This includes virtual machines, Docker, Docker compose and containerization. For the virtualization of all this I used Docker engine. All AI prompts that aided me in the process of creating this project are listed at the end with a short description followed by the question I asked the AI and the response it generated. I was also able to include the optional assignment of adding horizontal scaling to the web service.

A YouTube video showcasing the project is included under the reference list.

TABLE OF CONTENTS

PREFACE2

TABLE OF CONTENTS.....	3
1 INTRODUCTION	5
1.1 The Goal.....	5
1.2 Steps to achieve the desired result	6
1.2.1 Setup an Apache container based on Ubuntu:22.04.....	6
1.2.2 Setup a database container to interact with Apache.....	6
1.2.3 Combine the containers with docker-compose	6
1.2.4 Add persistent data by creating volumes	6
1.2.5 Add web scaling with nginx	6
1.2.6 End result	6
2 APACHE.....	7
2.1 Dockerfile.....	7
2.2 Index.php	8
3 MYSQL DATABASE	10
3.1 MySQL container	10
4 HORIZONTAL SCALING OF WEB SERVICES.....	12
4.1 Nginx configuration file.....	12
5 DOCKER-COMPOSE.....	13
5.1 Docker-compose	13
5.2 Data persistence	16
6 END RESULT	18
REFLECTION ON AI.....	19
CONCLUSION.....	20
7 GENERATIVE AI USAGE	21
7.1 Introduction to ChatGPT	21
7.2 Dockerfile.....	21
7.2.1 Apache Dockerfile	21
7.2.2 Command parameter explanation.....	23
7.2.3 CMD ["apache2ctl", "-D", "FOREGROUND"].....	24
7.3 Index.PHP	25
7.3.1 PHP database connection	25
7.3.2 MySQLi	28
7.3.3 Generated IF statement	30
7.3.4 Fetch_assoc()	31
7.4 Data persistence	32
7.4.1 Transferring files to volumes	32
7.5 Docker-compose	34
7.6 Horizontal web scaling of web services.....	36
7.6.1 Load balancing with Nginx.....	36
7.6.2 Further explanation of Load balancing.....	39
7.6.3 Troubleshooting	43
7.7 Other.....	45
7.7.1 Interacting with containers via a terminal.....	45

REFERENCE LIST	47
-----------------------------	-----------

1 INTRODUCTION

This chapter goes over the concept of the project and the steps towards achieving it. While also explaining a few of the basic concepts.

1.1 The Goal

The goal of this project and milestone is to create a simulation of how a webservice can operate. To do this we have to setup a LAMP stack and containerize it with Docker. Here is a short description explaining what a LAMP stack is.

Linux (L): Linux is the operating system that forms the foundation of the LAMP stack. It is used as the base image for the Apache and PHP module.

Apache (A): Apache is a widely used open-source web server software. It handles incoming HTTP requests from clients (web browsers) and serves web content, such as HTML files, images, and other resources. In this project it is used to serve a PHP file to the web page.

MySQL (M): MySQL is a popular open-source relational database management system. It stores and manages data for web applications. And will contain data that has to be displayed on the web page.

PHP (P): PHP is a server-side scripting language designed for web development. It is used to create dynamic web pages and interact with databases like MySQL for this project.

To create a LAMP stack and containerize it with Docker, you would install Linux as your operating system, set up Apache as the web server, configure MySQL as the database, and use PHP for creating dynamic web applications. Docker is then employed to package and run these components within separate containers, allowing for easier deployment, scalability, and management of the LAMP stack.

Besides creating a functional LAMP stack web service, there must be redundancy to ensure a good uptime for the web service. This is achieved with a load balancer that distributes incoming requests to a multitude of Apache containers. For this load balancer nginx was used.

Nginx is a high-performance, open-source software that acts as a reverse proxy and load balancer. It distributes incoming network traffic or client requests across multiple backend servers to ensure even workload distribution, enhance system reliability, and improve web application performance.

For this project the help of generative AI was used, mainly ChatGPT. Because of this there will be ChatGPT prompts included for every step that they were used for. The desired end result is having a custom index page which displays the name stored in the database on a webpage explaining that they have reached milestone 1.

1.2 Steps to achieve the desired result

This segment gives a small description of every step taken to achieve the goal of the project.

1.2.1 Setup an Apache container based on Ubuntu:22.04

This Apache container uses a base image of Ubuntu:22.04. This base image will be used to create a custom image with the help of a Dockerfile. With this Dockerfile the modules Apache2 and PHP can be installed with the result being a containerized webservice.

1.2.2 Setup a database container to interact with Apache

For the database container, the official SQL image from Docker Hub is used, however a volume must still be created in order for any data stored in the database to persist when the SQL container is shutdown or removed.

1.2.3 Combine the containers with docker-compose

To combine the Apache image and have PHP pull data from the database they will be combined through the use of a "docker-compose" file. This file will make sure that these containers can communicate with each other and their respective volumes.

1.2.4 Add persistent data by creating volumes

Since by default the data is lost when containers are removed or shutdown, volumes must be created in order to save the data. This is where Docker-compose comes in handy once again.

1.2.5 Add web scaling with nginx

Nginx (pronounced "engine-x") is a popular open-source web server and reverse proxy server software. For this project it will be used as a reverse proxy to ensure redundancy in the web service that we are trying to create.

A reverse proxy is a server or software that acts as an intermediary between client requests and backend servers. It forwards client requests to the appropriate server, in this case multiple Apache container instances.

The concept is that if one of the Apache containers experiences high traffic loads or becomes unavailable for any reason, the system will automatically reroute traffic to the other instances, ensuring uninterrupted access to the web service

1.2.6 End result

After finishing all the steps above the end result should be a web service that is running on a functional LAMP stack which is containerized and virtualized by Docker and has redundancy built in with the use of Nginx. The web page will display a message: ***"your name" has reached milestone1!***.

2 APACHE

This chapter includes the code responsible for configuring the Apache container. We'll employ a Dockerfile to craft a customized Apache image, and this image will additionally have PHP installed.

2.1 Dockerfile

This section shows the Dockerfile code in its entirety. The explanation for what each line of code does is included between the code as comments.

```
# Use the ubuntu:22.04 image as the parent image
FROM ubuntu:22.04
# Update the repo cache and upgrade all packages, -y is used to automatically
confirm any messages that pop up.
RUN apt-get update && apt upgrade -y
# for whatever reason php needs a timezone so I installed tzdata to obtain
the timezone, otherwise this image won't build.
RUN apt-get install -y tzdata
# Install apache and php on the container, -y is used to automatically
confirm any messages that pop up.
RUN apt-get install -y apache2 php
# Install the php-mysql module, to be able to access the classes required in
the php file.
RUN apt-get install -y php-mysql
# Remove the default index file since it's being replaced by a php file
instead.
RUN rm /var/www/html/index.html
# Start Apache in the foreground when the container runs, for this
"apache2ctl" has to be used since the container doesn't have systemd on it.
# Meaning that apache won't start by itself, but the apache module comes with
it's own launch commands, those can be used instead.
# to specify the fact that it has to run in the foreground the "-D" command-
line option is used to add the parameter "foreground" to the CMD.
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

With this Dockerfile an image can be easily build for anyone that has access to this file. To do this the following command is used:

- docker build -t apachephp .

The "-t" tag is used to add a name to the image, in this case *apachephp*. The . At the end of the command is to tell docker what directory to build the container from. In this case it's telling docker that it has to be build from the current directory where the Dockerfile is located.

2.2 Index.php

In this section the code of the index page is shown in its entirety. Within the code every line of PHP code is explained as comments. This PHP file will replace the html file that was acting as the default page for the Apache webserver.

```
<!DOCTYPE html>
<html>
<head>
    <title>Milestone1</title>
</head>
<body>
    <h1><?php
// store the information required to access the database.
// servername, is the name of the SQL container.
$servername = "contsql-m1-rv";
// username = the username that is used to login to SQL.
$username = "my_user";
// password = the password that is used to login to SQL.
$password = "my_password";
// dbname = The name of the database that you want to pull information from.
$dbname = "my_database";

// Create a connection with the database, using the predefined variables.
mysqli is an extension for PHP which improves the interactions between PHP
and SQL.
// mysqli in this case is used to login to SQL and choose a database.
$conn = new mysqli($servername, $username, $password, $dbname);

// SQL query to retrieve a name from the "names" table.
$name = "SELECT name FROM names";

// store the outcome from a query into the result variable.
//To do this the connection object that was made above is used to execute the
query. "query" being another part of the mysqli extension.
$result = $conn->query($name);
// check to see if the query has any existing "rows", in other words any data
in the $result variable.
if ($result->num_rows > 0) {
    // Output data of each row, fetch_assoc() is another command from mysqli.
    It returns a row from $result and turns it into an associative array.
    // this will return an array with the ID's and corresponding names from
    the database.
    while ($row = $result->fetch_assoc()) {
        // echo will print the name that was retrieved from the database
        followed by " has reached milestone 1!". This is the end result that we are
        looking for.
```



```
        echo $row["name"] . " has reached milestone 1!" . "<br>";
    }
} else {
    // if no data is found, it will show 0 results on the page instead of the
    // aforementioned print.
    echo "0 results";}
// this closes the connection to the database when it is no longer needed.
$conn->close();
?>
</h1>
<h2>Hostname: <?php
// this will display the name of the apache container that is handling your
// request to the webservice.
echo gethostname(); ?></h2>
</body>
</html>
```

3 MySQL DATABASE

In this chapter, we provide an explanation of the MySQL database setup. We walk through all the commands used in this process, offering detailed explanations of the parameters used in each command.

3.1 MySQL container

Since for this project the official SQL image was used, there will be no dockerfile included for it. However there must still be persistent data after shutdowns or restarts. Before a volume can be created the SQL container must be created. For this a username, password, port and which version of MySQL that has to be used must be specified:

1. `docker run --name mysql-container -e MYSQL_ROOT_PASSWORD=my-secret-pw -d -p 3306:3306 mysql:latest`

docker run: runs a container

--name: Is used to give the container a name

-e: environmental variable to store the password

-d: docker detached mode, which means the container will run in the background

-p 3306:3306 : This maps the 3306 port from the host to the 3306 port from the container.

In order to mount data on the volume a SQL container was required. First launch an interactive terminal window for the container:

2. `docker exec -it contsql-m1-rv /bin/bash`

A connection must be made to the database in order to add any data to it. Since the SQL container already has SQL installed, connecting to it requires the password and username that was given during the creation of the sql container. After logging in a database can be created.

3. `MySQL -u my_user -p`

-u: Parameter to specify a username for the MySQL login.

-p: Parameter to provide a password, however it's insecure to write down the password straight away. This parameter will have a password prompt after entering the command.

4. `CREATE DATABASE mydatabase;`

This is a command to create a new database in MySQL. It requires a single parameter which is the name of the database.

5. USE mydatabase;

This command lets you specify which database you want to actively use.

After the creation of the database, data can be inserted. To get the desired result for this project a table with only a single entry is used. The entry in this table is the name that is supposed to show up on the webpage.

6. CREATE TABLE names (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255));

INT AUTO_INCREMENT PRIMARY KEY: Is used to add an id to the entries that are added to the table, because of AUTO_INCREMENT the id is increased automatically for every entry in the table.

VARCHAR(255): means that the maximum amount of characters for the entries can be 255 characters long.

7. INSERT INTO names (name) VALUES ('Rob Verbeek');

A name is inserted into the table "names" of which an entry is called "name" and the value being the name that will be displayed on the web page. This is all the configuration that has to be done and the database can now be exited. The data will be saved in a volume. As mentioned in 5.2 Data persistence.

Afterwards the database can be edited to change the name that will be displayed on the web page. In order to change the name that will appear on the webpage the following command can be used:

8. UPDATE names SET name = 'new_name';

Because in this table there is only 1 entry there is no reason to add any conditions to the update. It just updates the sole entry to the new name.

4 HORIZONTAL SCALING OF WEB SERVICES

This chapter talks about how nginx interacts with the docker-compose stack and the configuration that has to take place for this to happen.

4.1 Nginx configuration file

By default Nginx will serve as just another webserver just like Apache. Without any further configuration the creation of the Nginx container will showcase the default web page that comes with the image. However since that is not the goal further configuration must be done. The code in this section is the configuration file that is used to make sure it redirects requests towards the Apache containers that host the actual web service. The explanation of the code can be found within the comments in between the coding.

```
#This block is used to define HTTP-related directives for Nginx. Inside this
block, you configure how Nginx handles HTTP requests and responses.
http {
# upstream groups are used for load balancing. In this case the group exists
out of 3 Apache servers that we created with the docker-compose file.
# All of these servers are listening on port 80, which is the default port
for websites.
    upstream apache_backend {
        server apache1:80;
        server apache2:80;
        server apache3:80;
    }
#This block defines an HTTP server configuration. It specifies how Nginx
should handle requests for a specific server.
    server {
# Nginx is listening to port 80 which is the default port for websites.
        listen 80;
# Since everything is hosted on localhost in this project, the Nginx server
will be handling requests on localhost.
        server_name localhost;
#location tells Nginx how to handle requests to its root path, here
proxy_pass will redirect any incoming requests towards the apache_backend
group
#Which was created above. You can specify in what strategy the server should
handle load balancing but by default it's on round robin.
#For this project I thought round robin was fine since there was no strategy
requirement hence there being no further configuration for it.
        location / {
            proxy_pass http://apache_backend;
        }
    }
}
```

5 DOCKER-COMPOSE

In this chapter the code of the Docker-compose file will be shown in its entirety. The explanation of every line of code will also be shown as comments. Pictures of the results of composing the containers will also be shown. Furthermore this segment talks about the persistence of data with the help of volumes.

5.1 Docker-compose

```
# Version of the compose file.
version: '3.9'
# Milestone1 is a network that is created for the containers, with this
# network different containers are able to communicate.
networks:
  milestone1: {}
# Volumes are used for data persistence even if the containers get shutdown.
# "sql-data" is volume used to store the database data, "apache" is used to
# store the index.php page for the webpage.
# The nginx volume contains the config file that is required to use the nginx
# image as a load-balancer.
volumes:
  sql-data: {}
  apache: {}
  nginx: {}
# Within services you define which services you want to run, in this case
# there is an "apache" and a "mysql" service.
services:
#name of the service
  apache1:
#name of the image that is used for the service, apachephp was previously
#built by the dockerfile.
    image: apachephp
#name of the container that is used
    container_name: contapa2-m1-rv-1
# restart:always means that whenever the compose file is used, it will always
# restart the services even if they were still running from a previous boot.
    restart: always
# since the apache webpage requires data from the database, I added mysql as
# a dependency.
# Meaning that the apache service cannot start until the mysql service has
# started.
    depends_on:
      - mysql
# in order for this service to communicate with others it needs to be part of
# a network, in this case I added it to "milestone1" which was defined above.
    networks:
      - milestone1
```

```

# a volume is needed for data persistence, this line mounts a volume called
"apache" to the /var/www/html directory.
# This is where the index page for the website is located and how the website
will get access to the index.php file
  volumes:
    - apache:/var/www/html
# Duplicate of the Apache container, used for webscaling.
apache2:
  image: apachephp
  container_name: contapa2-m1-rv-2
  restart: always
  depends_on:
    - mysql
  networks:
    - milestone1
  volumes:
    - apache:/var/www/html
# Another duplicate for webscaling.
apache3:
  image: apachephp
  container_name: contapa2-m1-rv-3
  restart: always
  depends_on:
    - mysql
  networks:
    - milestone1
  volumes:
    - apache:/var/www/html
# nginx will be used as the load-balancer, this means that all 3 of the
apache images will be accessed on the port that is assigned to the
loadbalancer.
nginx:
  image: nginx
  container_name: load_balancer
  restart: always
# Bind the host port 8085 to port 80 on the container.
  ports:
    - "8085:80"
  volumes:
    - nginx:/etc/nginx
  depends_on:
    - apache1
    - apache2
    - apache3
  networks:
    - milestone1

```

```

mysql:
#name of the image used, mysql being a pre-built image from the Docker Hub.
  image: mysql
  container_name: contsql-m1-rv
  restart: always
#Defines environment variables within the container. These variables are used
to configure the MySQL server that is started within the container.
  environment:
    MYSQL_ROOT_PASSWORD: root_password
    MYSQL_DATABASE: my_database
    MYSQL_USER: my_user
    MYSQL_PASSWORD: my_password
# Just like the apache service, this one has to be part of the same network
for php to be able to communicate with the database.
  networks:
    - milestone1
# Mount the volume "sql-data" to the /var/lib/mysql directory, this directory
is where the mysql data is stored.
  volumes:
    - sql-data:/var/lib/mysql

```

With this Docker-compose file, the webservice can finally be created. To do this the command "docker compose up" is used and it will begin to launch the containers. The image below is the end result. [Figure 1:Docker-compose stack]

5.2 Data persistence

To keep data when the containers get shutdown/crash or are restarted, volumes are created. These volumes get created by the Docker-compose file and can then be accessed to add data to them.

- `docker cp index.php milestone1_apache:/var/www/html`

This command will copy the file *index.php* that was created on the host machine into the container that is running the Apache service. The name *milestone1_apache* is the name for the volume that was created by the Docker-compose file. [Figure 2:Volumes in Docker]

- `docker cp nginx.conf load_balancer:/etc/nginx`

This command copies the *nginx.conf* file from the current directory into the */etc/nginx* directory in the *load_balancer* container. The *load_balancer* container is the Nginx image that has the *milestone1_nginx* volume mounted on it. [Figure 1: Docker-compose stack, Figure 2: Volumes in Docker]

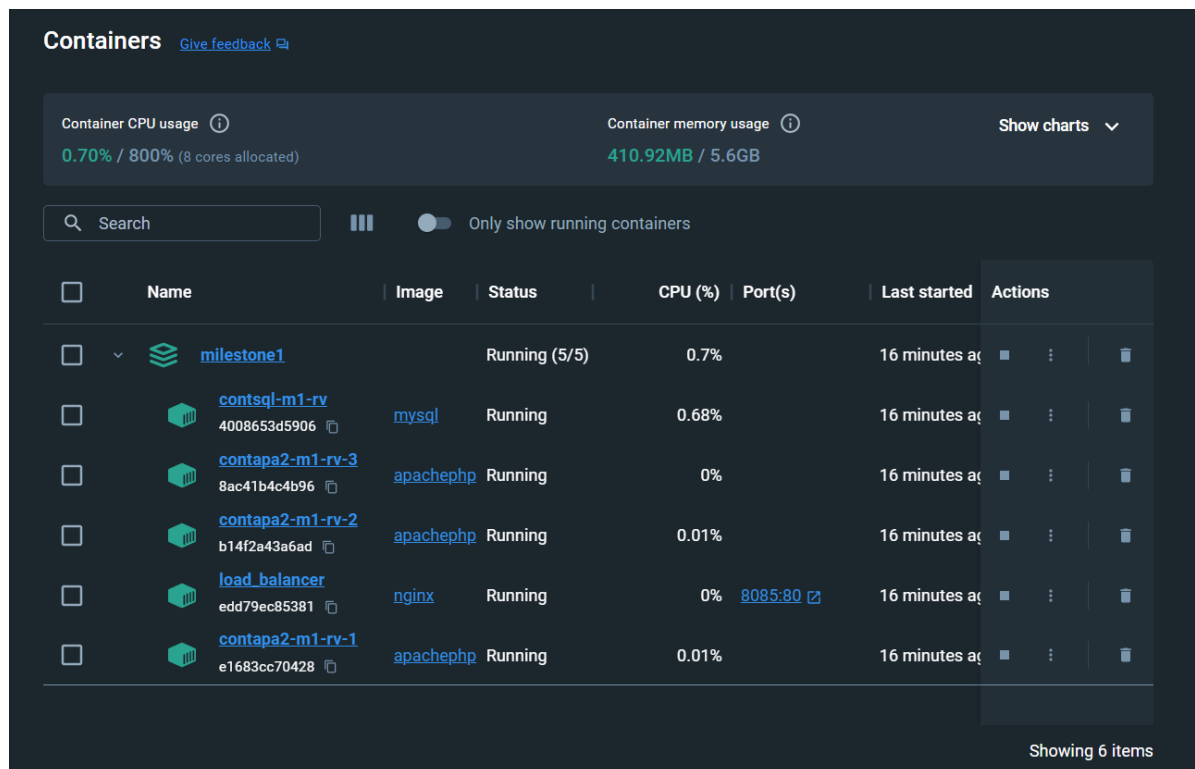


Figure 1: Docker-compose stack

Volumes

[Give feedback](#)Create +

Try the Volumes Docker extension to backup, restore and share volumes. [View details](#)

☐

Name

☐

milestone1_apache

☐

milestone1_nginx

☐

milestone1_sql-data

Status

in use

in use

in use

Created

2 days ago

2 hours ago

5 days ago

Size

12 kB

40 kB

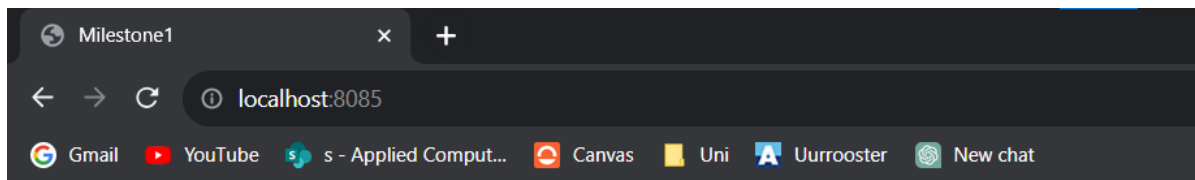
200.2 MB

Actions

Figure 2: Volumes in Docker

6 END RESULT

After going through all the steps listed in the introduction the end result is a fully functioning LAMP stack that is displaying a web page. On this web page is a title which retrieves data from the database and a secondary title that displays which container this is being hosted from. The hostname showcases that the load-balancer is working since refreshing can redirect you to one of the other Apache containers that are running this service.[Figure 3: End result, web page]. The refreshing of the hostnames is also showcased in the video.



Rob Verbeek has reached milestone 1!

Hostname: 8ac41b4c4b96

Figure 3: End result, web page

REFLECTION ON AI

During the making of this project AI was a huge help and time-saver. Instead of having to look up every problem you encounter online and looking through tons of stack overflow posts it's a lot easier to ask the AI a question and have a good response generated most of the time.

I used the AI to generate code for me and asked it to also explain to me what each part of the code does. It taught me how to structure a docker-compose file and how to use nginx as a load-balancing tool. Especially the load-balancing part would've been hard for me to figure out without its help since the coverage on web-scaling in the Linux webservices course was confusing to me.

I noticed that sometimes the AI does generate answers that don't make much sense or isn't what I asked for, this could be due to me giving bad prompts or the AI misinterpreting the question. Nonetheless telling the AI about its shortcomings or wrong answers did in most cases lead to an answer I could work with and use for this project. Experiences like this does teach you that AI isn't all knowing. It can still be wrong and you have to think critically about the answers that are generated instead of believing everything it tells you.

Using AI for troubleshooting whenever I got an error was extremely helpful. E.g I had an issue where the webpage was blank after using the AI to help me create the *index.php* page. After looking through the code I noticed that there were classes used I hadn't seen before so I asked ChatGPT what *mysqli* is and with its help I figured out that I hadn't installed the PHP extension on my Apache container that was necessary for this to work. When using AI for troubleshooting I noticed you have to give a lot of details about all the systems that are in place otherwise it won't be able to generate proper responses for the error you are experiencing.

By the end of this project I noticed that AI can be extremely useful when it comes to creating code however you have to understand what it's writing because sometimes it will generate code that doesn't do anything but cause bloat in your programming. Being good at creating prompts is an extremely useful skill when working with AI to avoid the immensely long responses and getting a response that you can learn from.

CONCLUSION

At the start I had no idea how to even begin creating this project. I'm not very good with Linux as it is so I found this quite challenging at the start. I started off with reading through the course material a few times, especially the Docker chapter to help understand it a bit better. With the help of ChatGPT this project ended up being quite doable in the end since whenever I had any questions about how something should be done I could simply ask ChatGPT to generate me code. Most of the time this didn't give me the complete picture but it was a good start nonetheless and was easy to adjust to what I wanted it to be. By using the AI more throughout this project I got better at making prompts to give me a better result from the AI.

I did notice that the AI always provided too much information and a lot of information I didn't ask for, sometimes this was actually helpful but other times it was a slog to read through all the information that wasn't relevant to my questions. Even on a few of my questions it was wrong and I had to inquire further or adjust the information it gave me in a way where it made sense or worked the way I intended it to.

All in all I found this project an interesting experience and learned a fair amount of new things about how web services are created, the one thing I found rather annoying about this project was documenting all the AI prompts since it has horrible formatting for documentation and adjusting it takes a lot of time. But learning to use AI and seeing how easy it is to quickly find information you are looking for is an amazing boon to have.

7 GENERATIVE AI USAGE

7.1 Introduction to ChatGPT

ChatGPT was used to answer all questions I had during the making of this project, it is a generative AI tool that can be used to ask questions and it will give responses. In this project I used it to give me code snippets and ask further explanation of those snippets. In this section a short description of why the question was asked is given followed up by the prompt I entered and the response that was generated by ChatGPT.

7.2 Dockerfile

7.2.1 Apache Dockerfile

I wanted to get ChatGPT to write me a dockerfile to save some time, however it didn't quite end up giving me everything I wanted since I didn't go into proper detail. It still ended up being a nice start however.

Me:

how to install apache with a dockerfile on ubuntu

ChatGPT:

To install Apache within a Docker container running Ubuntu, you can create a Dockerfile that uses an official Ubuntu base image and then installs Apache within the container. Here's how to do it:

1. Create a Dockerfile:

```
```Dockerfile
Use an official Ubuntu base image
FROM ubuntu:latest

Update the package list and install Apache
RUN apt-get update && apt-get install -y apache2

Expose port 80 (HTTP)
EXPOSE 80

Start Apache service when the container runs
CMD ["apache2ctl", "-D", "FOREGROUND"]
```
```

In this Dockerfile:

- We use the latest official Ubuntu base image.
- We update the package list and install Apache using the `apt-get` package manager.
- We expose port 80, which is the default port for Apache web server.
- Finally, we start the Apache service in the foreground using the `CMD` instruction.

2. Build the Docker image:

Save the Dockerfile in a directory, and then open a terminal or command prompt in that directory. Run the following command to build the Docker image:

```
```bash
docker build -t my-apache-ubuntu-image .
```
```

Replace `my-apache-ubuntu-image` with your desired image name.

3. Run the Apache container:

Once the image is built, you can run a container from the image with the following command:

```
```bash
docker run -d -p 80:80 --name my-apache-ubuntu-container my-apache-ubuntu-image
```
```

This command will start a Docker container named `my-apache-ubuntu-container` based on the `my-apache-ubuntu-image`, and it will map port 80 of the container to port 80 of the host machine. You can access the Apache web server by opening a web browser and navigating to `http://localhost`.

Your Apache web server is now up and running inside a Docker container based on an Ubuntu image. You can further customize the Apache configuration or serve your website content by adding files to the container or by modifying the Apache configuration within the Dockerfile.

7.2.2 Command parameter explanation

Since I'm not great with Linux I had no idea what the "-y" parameter meant in the command. Since this was a follow up question ChatGPT already had context for this prompt to work.

Me:

what does install -y mean

ChatGPT:

In the context of the `apt-get` command on Debian-based Linux distributions (such as Ubuntu), the `-y` option stands for "yes." When you use `apt-get install -y <package>`, you are telling the package manager to assume "yes" as the answer to all confirmation prompts.

Here's how it works:

1. Without `-y`: If you run `apt-get install <package>`, the package manager may prompt you with a confirmation message asking if you are sure you want to install the package. You would need to manually type "yes" or "y" and press Enter to proceed.
2. With `-y`: When you add the `-y` option, as in `apt-get install -y <package>`, you are telling the package manager to automatically answer "yes" to any confirmation prompts. This is useful when you want to automate installations or prevent manual intervention in scripts or Dockerfiles.

Using `-y` in the Dockerfile for package installation ensures that the installation process continues without waiting for manual input during the image building process. It's a common practice in Dockerfiles to use `-y` for package installations to avoid interruptions and streamline the build process.

7.2.3 CMD ["apache2ctl", "-D", "FOREGROUND"]

I wanted more information about this command line since it's not a part of normal Linux interactions. The reason this was used is because systemd and init aren't part of the container image so Apache has to be launched with commands that come with the installation.

Me:

CMD ["apache2ctl", "-D", "FOREGROUND"] what does the "-D" do

ChatGPT:

In the CMD instruction you provided, `["apache2ctl", "-D", "FOREGROUND"]`, the "-D" flag is used with `apache2ctl` to specify a command-line option. In this case, it sets the "D" flag to "FOREGROUND."

Here's what each part of the CMD instruction does:

1. `apache2ctl`: This is a command-line utility for controlling the Apache HTTP Server. It can be used to start, stop, and perform various other operations on the Apache server.
2. `-D`: This is a command-line option used with `apache2ctl` to specify a parameter or argument. In this case, it's set to "FOREGROUND."
3. "FOREGROUND": This parameter tells Apache to run in the foreground, which means that the Apache server process will stay in the foreground and log its output to the terminal where it was started. This is commonly used in containerized environments like Docker, where you want the container to stay running as long as the Apache server is active and to see its logs in the container's terminal.

Using the "-D" flag with "FOREGROUND" as an argument essentially instructs Apache to run in the foreground and log its output to the console, which is useful for monitoring and debugging within a Docker container.

Here's an example of how this CMD instruction might be used in a Dockerfile to start an Apache web server in a container:

```
`` `Dockerfile
FROM ubuntu:latest
# Install Apache and other dependencies
RUN apt-get update && apt-get install -y apache2
# Set the CMD instruction to start Apache in the foreground
CMD ["apache2ctl", "-D", "FOREGROUND"]
`` `
```

In this example, when you run a container from the image built using this Dockerfile, it will start the Apache web server in the foreground and display its logs in the terminal.

7.3 Index.PHP

7.3.1 PHP database connection

I wanted to know what the necessary code is to be able to retrieve data from a database with PHP since I hadn't learned this yet.

Me:

How do I use PHP to request a name from MySQL.

ChatGPT:

To use PHP to request a name from a MySQL database, you need to follow these general steps:

1. ****Connect to the MySQL Database****: First, you must establish a connection to your MySQL database. You can use PHP's ``mysql`` or ``PDO`` extension for this purpose. Here's an example using ``mysql``:

```
` ` ` php

$mysqli = new mysqli("localhost", "username", "password", "database_name");

// Check connection

if ($mysqli->connect_error) {

    die("Connection failed: " . $mysqli->connect_error);

}

` ` `
```

Replace ``"localhost"``, ``"username"``, ``"password"``, and ``"database_name"`` with your MySQL server details.

2. ****Execute a SQL Query****: After connecting to the database, you can execute an SQL query to request a name. For example, if you have a table named "users" with a column called "name," you can retrieve a name like this:

```
```php
$sql = "SELECT name FROM users WHERE user_id = ?";

$user_id = 123; // Replace with the actual user ID you want to retrieve

$stmt = $mysqli->prepare($sql);
$stmt->bind_param("i", $user_id);
$stmt->execute();
$result = $stmt->get_result();
```
```

This code prepares and executes an SQL query to retrieve the name of a user with a specific user ID.

3. ****Fetch Data****: Once you've executed the query, you can fetch the data returned by the database. If you expect only one result (as in this case), you can use ``fetch_assoc()`` to retrieve the name:

```
```php
if ($result->num_rows > 0) {
 $row = $result->fetch_assoc();
 $name = $row["name"];
 echo "User's name is: " . $name;
} else {
 echo "User not found.";
}
```
```

This code checks if there is at least one result and fetches the name. If no result is found, it prints a message indicating that the user was not found.

4. ****Close the Database Connection****: Don't forget to close the database connection when you're done:

```
```php  

$mysqli->close();

```
```

5. ****Error Handling****: Ensure you handle any potential errors, especially in database connection and query execution, using try-catch blocks or other error-handling mechanisms.

Make sure to replace the placeholders in the code with your actual database connection details, table name, and the specific data you want to retrieve. This is a basic example, and you can expand on it depending on your application's requirements.

7.3.2 MySQLi

Since the previous prompt showed a lot of unknown code to me I asked questions to help me understand them better. Of which an extension was being used that I hadn't heard of before namely mysqli.

Me:

what is mysqli

ChatGPT:

MySQLi (short for MySQL Improved) is a PHP extension that provides an improved and more feature-rich way to interact with MySQL databases from PHP scripts. MySQLi is an alternative to the older MySQL extension and is commonly used for connecting to MySQL databases in modern PHP applications.

Some key features and benefits of MySQLi include:

1. **Improved Security:** MySQLi supports prepared statements and parameterized queries, which help prevent SQL injection attacks by safely handling user input.
2. **Object-Oriented and Procedural Interfaces:** MySQLi provides both object-oriented and procedural programming interfaces, allowing developers to choose the style they are most comfortable with.
3. **Support for Multiple Statements:** MySQLi supports executing multiple SQL statements in a single query, which can improve performance and reduce the number of round-trips to the database.
4. **Enhanced Error Handling:** MySQLi offers improved error handling capabilities, making it easier to diagnose and troubleshoot database-related issues.
5. **SSL Support:** It includes support for secure connections to the database using SSL encryption.

Here's a basic example of how to use MySQLi to connect to a MySQL database and execute a simple query in PHP:

```
```php
// Create a connection to the MySQL database

$mysqli = new mysqli("localhost", "username", "password", "database_name");

// Check for connection errors
if ($mysqli->connect_error) {
 die("Connection failed: " . $mysqli->connect_error);
}

// SQL query
$sql = "SELECT id, name FROM your_table_name";

// Execute the query
$result = $mysqli->query($sql);

// Fetch and process the results
if ($result) {
 while ($row = $result->fetch_assoc()) {
 echo "ID: " . $row['id'] . " Name: " . $row['name'] . "
";
 } else {
 echo "Query failed: " . $mysqli->error;
 }
}

// Close the database connection
$mysqli->close();```
```

This example demonstrates how to establish a connection to a MySQL database, execute a SELECT query, fetch and display the results, and handle errors. The use of prepared statements with MySQLi is recommended for more secure database interactions.

### 7.3.3 Generated IF statement

I wanted further explanation of what happened in this IF statement that was generated since it contains code I hadn't seen before such as the `fetch_assoc()` command and the way `num_rows` interacts with the database.

**Me:**

```
if ($result->num_rows > 0) {
 // Output data of each row
 while ($row = $result->fetch_assoc()) {
 echo $row["name"] . " has reached milestone 1!" . "
";
 } else { echo "0 results";}
```

**ChatGPT:**

The code you've provided is a common pattern for fetching and displaying data from a MySQL database in PHP when using the MySQLi extension. Here's what this code does:

1. `if ($result->num_rows > 0)`: This condition checks if there are rows (records) in the result set returned by the SQL query. The `num_rows` property of the `$result` object returns the number of rows in the result set.

2. Inside the `if` block, you use a `while` loop to iterate through the result set with `$result->fetch_assoc()`. For each row (record) in the result set, you:

- Retrieve the "name" column value using `$row["name"]`.
- Construct and output a message indicating that the name has reached "milestone 1."

3. If there are no rows in the result set (i.e., the `if` condition evaluates to `false`), it means the query didn't return any results, and the "0 results" message is displayed.

This code is a basic example of how to retrieve data from a MySQL database and display it. It assumes that the SQL query retrieved data from a table, and it iterates through the result set to process each row.

Here's an important note: Always use prepared statements or input validation when constructing SQL queries to prevent SQL injection attacks, especially if user input is involved in building your SQL queries.

### 7.3.4 Fetch\_assoc()

**Me:**

what does the fetch\_assoc() line do

**ChatGPT:**

The `fetch_assoc()` function is a method provided by the MySQLi extension in PHP. It is used to fetch the next row from a result set and return it as an associative array. Each column in the row becomes a key in the associative array, and the corresponding column value becomes the associated value. Here's how it works:

1. When you execute a query using `query()` in MySQLi for a SELECT statement, it returns a result set, which is essentially a set of rows from the database that match the query criteria.

2. To retrieve the data from this result set row by row, you use the `fetch_assoc()` method on the result set object. Each call to `fetch_assoc()` fetches the next row from the result set, and it returns an associative array that represents that row.

For example, if you have a table with columns `id`, `name`, and `age`, and you execute a query to fetch all records, you might use `fetch_assoc()` like this:

```
```php
$result = $conn->query("SELECT id, name, age FROM my_table");
while ($row = $result->fetch_assoc()) {
    // $row is now an associative array with keys "id," "name," and "age"
    echo "ID: " . $row["id"] . ", Name: " . $row["name"] . ", Age: " . $row["age"] .
"<br>";}
```
```

In the example above, for each iteration of the `while` loop, `$row` will contain an associative array representing a single row from the result set. You can access individual column values using the keys in the associative array, such as `$row["id"]`, `$row["name"]`, and `$row["age"]`.

This method simplifies the process of working with query results, as you can directly access the data by column name in a user-friendly way.

## 7.4 Data persistence

### 7.4.1 Transferring files to volumes

I didn't know how to make use of volumes to make sure that my data ( index.php and SQL ) was persistent when the containers get shutdown or restarted. For this I asked ChatGPT to explain to me how file transfer works with docker volumes.

**Me:**

how do I put a file on a docker volume

**ChatGPT:**

To put a file on a Docker volume, you can follow these steps:

1. **\*\*Create a Docker Volume\*\*** (if not already created):

If you don't have a Docker volume created, you can do so using the ``docker volume create`` command. For example:

```
```bash
docker volume create my_volume
```
```

2. **\*\*Run a Docker Container\*\***:

You need to run a Docker container and mount the volume into it. You can do this using the ``-v`` (or ``--volume``) option when starting a container. For instance:

```
```bash
docker run -v my_volume:/path/in/container -it ubuntu
```
```

In the above command:

- ``-v my_volume:/path/in/container`` mounts the ``my_volume`` volume to the specified path inside the container.

- ``-it ubuntu`` starts an interactive Ubuntu container. You can replace ``ubuntu`` with the image name you need.



### 3. **\*\*Copy the File into the Container\*\***:

Once the container is running, you can copy the file into the container using a command like ``docker cp`` or any method you prefer. For example:

```
` `` `bash

docker cp your_file.txt <container_id>:/path/in/container/

` `` `
```

Replace ``your_file.txt`` with the actual path to the file you want to copy, ``<container_id>`` with the ID or name of your running container, and ``/path/in/container/`` with the path you mounted the volume to.

### 4. **\*\*Verify the File is in the Volume\*\***:

If you want to ensure that the file has been copied to the volume, you can run another container and mount the same volume to check:

```
` `` `bash

docker run -v my_volume:/path/in/container -it ubuntu

` `` `
```

Inside this new container, you should be able to see the file at the specified path in the volume.

Remember that Docker volumes are intended for data persistence, and the data stored in volumes can be shared among multiple containers. Make sure to clean up your containers and volumes when you're done, using ``docker rm`` and ``docker volume rm``, respectively.

## 7.5 Docker-compose

Since I already asked ChatGPT to give me some dockerfile code before I decided to ask it to combine that with an SQL database to give me a basic layout of what the Docker-compose file should look like.

**Me:**

write me a docker compose that uses the previous apache container and a sql container

**ChatGPT:**

Certainly! You can use Docker Compose to define and run multi-container applications. Here's a `docker-compose.yml` file that sets up an Apache container and a MySQL container, using the images and configuration we discussed earlier:

```
```yaml
version: '3'

services:
  apache:
    image: my-apache-image
    container_name: my-apache-container
    ports:
      - "80:80"
    # You can add additional configuration or volumes here if needed

  mysql:
    image: mysql:latest
    container_name: my-mysql-container
    environment:
      MYSQL_ROOT_PASSWORD: root_password
      MYSQL_DATABASE: my_database
      MYSQL_USER: my_user
      MYSQL_PASSWORD: my_password
    ports:
      - "3306:3306"
    # You can add additional configuration or volumes here if needed
```
```

In this ``docker-compose.yml`` file:

1. We define two services: ``apache`` and ``mysql``.
2. For the ``apache`` service:
  - We use the ``my-apache-image`` image, which should be the image you created earlier in your Dockerfile.
  - We specify a custom container name using ``container_name``.
  - We map port 80 of the host to port 80 of the container to access the Apache web server.
3. For the ``mysql`` service:
  - We use the official ``mysql:latest`` image.
  - We set environment variables like ``MYSQL_ROOT_PASSWORD``, ``MYSQL_DATABASE``, ``MYSQL_USER``, and ``MYSQL_PASSWORD`` to configure the MySQL server inside the container.
  - We map port 3306 of the host to port 3306 of the container to access the MySQL server.
4. You can add additional configurations, volumes, and other settings for each service as needed.

To use this Docker Compose configuration:

1. Save the ``docker-compose.yml`` file in a directory.
2. Open a terminal or command prompt in that directory.
3. Run the following command to start the services defined in the Compose file:

```
```bash
```

```
docker-compose up
```

```
```
```

This will create and start the containers for Apache and MySQL as specified in the Compose file. You can access the Apache web server on port 80, and the MySQL database on port 3306 as defined in the services.

## 7.6 Horizontal web scaling of web services

I had no idea how to start with horizontal web scaling. The only thing I knew so far was that nginx was used for it since that was included in the course material. Hence I asked ChatGPT the following.

### 7.6.1 Load balancing with Nginx

**Me:**

how do I load-balance multiple apache2+php containers and a SQL database with nginx

**ChatGPT:**

To load-balance multiple Apache2+PHP containers and a SQL database with Nginx, you can set up a typical LAMP (Linux, Apache, MySQL, PHP) stack with a reverse proxy for load balancing using Nginx. Here are the steps to achieve this:

#### 1. **\*\*Set Up Your Docker Containers:\*\***

First, you need to create multiple Apache2+PHP containers and a SQL database container (e.g., MySQL or PostgreSQL). You can use Docker or any container orchestration platform like Docker Compose, Kubernetes, or a Docker Swarm for this purpose.

#### 2. **\*\*Configure Apache2+PHP Containers:\*\***

Ensure that each Apache2+PHP container is configured to serve your PHP web application. You should have your web application files and configurations inside these containers.

#### 3. **\*\*Set Up Database Container:\*\***

Configure the SQL database container (e.g., MySQL or PostgreSQL) with the required databases, tables, and user accounts. Ensure that you can connect to this database from your Apache2+PHP containers.

#### 4. **\*\*Install and Configure Nginx:\*\***

Install Nginx on a separate server or container. This will be your load balancer.

## 5. **\*\*Nginx Configuration:\*\***

Configure Nginx to act as a reverse proxy and load balancer. You'll need to define the upstream server group for your Apache2+PHP containers and configure Nginx to distribute incoming requests among these containers. Here's a simplified Nginx configuration example:

```
```nginx
http {
    upstream backend {
        server apache1_container:80;
        server apache2_container:80;
        # Add more Apache containers as needed
    }

    server {
        listen 80;
        server_name yourdomain.com;

        location / {
            proxy_pass http://backend;
            # Other proxy settings (e.g., timeouts, caching) can be added here
        }
    }
}
```
```

Replace `apache1\_container`, `apache2\_container`, and `yourdomain.com` with the actual names or IP addresses of your Apache2+PHP containers and your domain.

## 6. **\*\*DNS Configuration:\*\***

Ensure that your domain's DNS records point to the IP address or host where Nginx is running.

## 7. **\*\*Testing:\*\***

Test the setup to make sure that Nginx is distributing requests among the Apache2+PHP containers. You should also check that the PHP application works correctly and can connect to the database.

## 8. **\*\*Monitoring and Scaling:\*\***

Implement monitoring tools and consider auto-scaling mechanisms to add or remove Apache2+PHP containers based on traffic demand.

## 9. **\*\*Security:\*\***

Implement security best practices, like firewall rules, secure database configuration, and SSL/TLS encryption for Nginx.

## 10. **\*\*Backup and High Availability:\*\***

Implement regular backups for your database and consider high availability solutions if required.

Remember to adapt this example to your specific use case and requirements. Additionally, consider using container orchestration tools like Docker Swarm or Kubernetes for better container management, scaling, and failover capabilities.

### 7.6.2 Further explanation of Load balancing

**Me:**

elaborate on the creation of an nginx container to act as a load-balancer for my Apache containers and SQL container that are docker-composed

**ChatGPT:**

Creating an Nginx container to act as a load balancer for your Apache containers and SQL container that are managed by Docker Compose involves several steps. In this example, I'll walk you through setting up a basic Docker Compose configuration for this scenario.

Here are the steps:

#### 1. **\*\*Docker Compose File (docker-compose.yml):\*\***

Start by creating a `docker-compose.yml` file in your project directory to define the services you want to run. In this case, you'll have three services: Nginx (load balancer), Apache (web server), and SQL (database). Here's a basic example:

```
```yaml
version: '3'

services:
  nginx:
    image: nginx
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    depends_on:
      - apache
  apache:
    image: your-apache-image
```

```
# Add your Apache configuration and volume mappings

sql:

  image: your-sql-image

  # Add your SQL container configuration
  ...
```

Make sure to replace `your-apache-image` and `your-sql-image` with the actual Docker image names for your Apache and SQL containers. Additionally, create an `nginx.conf` file in the same directory to configure Nginx.

2. **Nginx Configuration File (nginx.conf):**

Create an `nginx.conf` file in your project directory for Nginx configuration. Here's a basic example to configure Nginx as a load balancer:

```
`` `nginx
events {}

http {
    upstream apache_backend {
        server apache:80;

        # Add more Apache containers as needed
    }

    server {
        listen 80;

        server_name yourdomain.com;

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



```

    }
}
```

```

In the above example, `apache` refers to the service name defined in the Docker Compose file. Replace `yourdomain.com` with your actual domain or IP.

### 3. **Apache and SQL Container Configuration:**

Make sure you have appropriate configurations and Docker images for your Apache and SQL containers. Ensure that your Apache containers are listening on port 80 and your SQL container is correctly configured and reachable by the Apache containers.

### 4. **Start the Docker Compose Stack:**

Open a terminal in your project directory and run the following command to start your Docker Compose stack:

```

```bash
docker-compose up -d
```

```

This will start the Nginx, Apache, and SQL containers in detached mode.

### 5. **Testing:**

Test your setup by accessing your web application through the load balancer (Nginx) using the domain or IP you specified. Nginx will distribute requests to your Apache containers.

## 6. **\*\*Scaling:\*\***

You can scale your Apache containers up or down by adjusting the number of containers in the ``docker-compose.yml`` file, under the ``apache`` service. Docker Compose will manage the container instances.

## 7. **\*\*Security:\*\***

Ensure you implement appropriate security measures for your Nginx, Apache, and SQL containers. This includes firewalls, authentication, and encryption, especially for your SQL container.

By following these steps, you can set up a Docker Compose configuration with Nginx as a load balancer for your Apache and SQL containers. Remember to adapt the configuration and security measures to your specific requirements and the technologies you're using.

### 7.6.3 Troubleshooting

I was having trouble with the default page of Nginx showing up after setting up the config file and the docker-compose file. The problem ended up being that I used the container names instead of the service names in the upstream segment of the *nginx.conf* file.

**Me:**

this just shows me the default page of nginx instead of the webpage that is located on my Apache containers.

**ChatGPT:**

If you are seeing the default Nginx page instead of your web application hosted on your Apache containers, it may indicate an issue with your Nginx configuration or the way your Apache containers are set up. Here are some troubleshooting steps to help you resolve the issue:

1. **\*\*Check Nginx Configuration:\*\***

- Ensure that the `nginx.conf` file you created and mounted into the Nginx container is correctly configured.

- Verify that the `upstream` block in your `nginx.conf` file is correctly pointing to the names of your Apache services in your Docker Compose file.

2. **\*\*Check Apache Configuration:\*\***

- Verify that your Apache containers are correctly configured to serve your web application. Make sure they are listening on port 80 and serving your web content.

- Confirm that there are no issues with the volume mappings and paths to your web application files within the Apache containers.

3. **\*\*Check DNS and Hosts File:\*\***

- Ensure that your domain name or IP address is correctly resolved to the Nginx container's IP address. You can do this by checking your DNS configuration or updating your local host file if testing on your local machine.

4. **\*\*Docker Network:\*\***

- Confirm that your Nginx container can reach the Apache containers. All containers within the same Docker Compose setup should be on the same Docker network and be able to communicate with each other.

#### 5. **\*\*Logs:\*\***

- Check the logs of both the Nginx and Apache containers for any error messages or issues. You can use the following commands to view container logs:

```
```bash
```

```
docker-compose logs nginx
```

```
docker-compose logs apache1 # Replace with your Apache service names
```

```
```
```

#### 6. **\*\*Firewall and Security:\*\***

- Ensure that there are no firewall rules or security settings preventing Nginx from accessing the Apache containers. Make sure that the necessary ports are open.

#### 7. **\*\*Test Directly on Apache Containers:\*\***

- To isolate the issue, you can try accessing your web application directly from one of your Apache containers to ensure it is working as expected.

#### 8. **\*\*Check Container Names:\*\***

- Double-check that the names of your Apache services in the Docker Compose file match the names used in the Nginx configuration.

#### 9. **\*\*Clear Browser Cache:\*\***

- Sometimes, browser caching can lead to unexpected behavior. Clear your browser cache or try accessing your web application from a different browser to rule out caching issues.

#### 10. **\*\*SSL/TLS Considerations:\*\***

- If you are using SSL/TLS, ensure that your Nginx configuration includes the necessary SSL settings, and that your certificates and keys are correctly configured.

By following these troubleshooting steps and carefully reviewing your Nginx and Apache configurations, you should be able to identify and resolve the issue that is preventing Nginx from correctly proxying requests to your Apache containers.

## 7.7 Other

### 7.7.1 Interacting with containers via a terminal

To be able to add data to my database I had to be able to interact with the container which means opening it in a terminal. I didn't know how to do that so I asked ChatGPT.

**Me:**

interactive docker container terminal

**ChatGPT:**

To interact with a Docker container's terminal interactively, you can use the `docker exec` command. This command allows you to run commands inside a running Docker container, effectively providing you with a terminal interface to the container. Here's how you can do it:

1. First, make sure your Docker container is running. You'll need to know the name or ID of the container you want to access.
2. Open your terminal or command prompt.
3. Use the following command to start an interactive session inside the container:

```
```bash
```

```
docker exec -it <container_name_or_id> /bin/bash
```

```
```
```

Replace `<container_name_or_id>` with the name or ID of the Docker container you want to access.

- The `-it` flag stands for "interactive" and "tty," which allows you to interact with the container's terminal.

- `/bin/bash` is the shell you want to run inside the container. You can replace it with another shell (e.g., `/bin/sh`, `/bin/zsh`, or any other shell available in the container).

4. Once you run the command, you should be inside the container and have access to its terminal. You can run commands, install packages, and perform any other actions as if you were working directly on the container.

5. When you're done, you can exit the container's terminal by typing `exit`. This will return you to your host machine's terminal.

Here's a full example:

```
```bash
```

```
docker exec -it my_container /bin/bash
```

```
```
```

This will start an interactive terminal session in the Docker container named "my\_container" using the Bash shell. Remember to replace "my\_container" with the actual name or ID of your Docker container.

## **REFERENCE LIST**

### **Youtube Video:**

<https://www.youtube.com/watch?v=PEoTFBykW6A>