Course

scp -P 42 -r ~/Desktop/.env abboutah@127.0.0.1:/home/abboutah/project/srcs/

What is Docker?

Docker is a tool designed to allow you to build, deploy and run applications in an isolated and consistent manner across different machines and operating systems. This process is done using **CONTAINERS**. which are lightweight virtualized environments that package all the dependencies and code an application needs to run into a single text file, which can run the same way on any machine.
While Docker is primarily used to package and run applications in containers, it is not limited to that use case. Docker can also be used to create and run other types of containers, such as ones for testing, development, or experimentation.

How Does Docker Work?

Docker uses a client-server architecture, where Docker client talks to Docker daemon, which is responsible for building, running, and distributing all Docker containers. The Docker client and Docker daemon communicate using a REST API, over a UNIX socket or a network interface.

What is a Docker Image?

Docker Image is a lightweight executable package that includes everything the application needs to run, including the code, runtime environment, system tools, libraries, and dependencies.
Although it cannot guarantee error-free performance, as the behavior of an application ultimately depends on many factors beyond the image itself, using Docker can reduce the likelihood of unexpected errors.
Docker Image is built from a **DOCKERFILE**, which is a simple text file that contains a set of instructions for building the image, with each instruction creating a new layer in the image.

What is a Dockerfile?

Dockerfile is that ***SIMPLE TEXT FILE*** that I mentioned earlier, which contains a set of instructions for building a Docker Image. It specifies the base image to use and then includes a series of commands that automate the process for configuring and building the image, such as installing packages, copying files, and setting environment variables. Each command in the Dockerfile creates a new layer in the image.
Here's an example of a Dockerfile to make things a little bit clear:

This Specifies the base images for the container (in this case, it's the 3.14 version of Alpine)
FROM alpine:3.14
This Run commands in the container shell and installs the specified packages
(it will install nginx & OpenSSL, and will create the directory "/run/nginx" as well)

```
RUN apk update && \
    apk add nginx openssl && \
    mkdir -p /run/nginx

# This Copies the contents of "./conf/nginx.conf" on the host machine to the "/etc/nginx/http.d/"
# directory inside the container
COPY ./conf/nginx.conf /etc/nginx/http.d/default.conf

# This Specifies the command that will run when the container gets started
CMD ["nginx", "-g", "daemon off;"]
```

What is a Docker Compose?

Docker Compose is a powerful tool that simplifies the deployment and management of multi-container Docker applications. It provides several benefits, including simplifying the process of defining related services, volumes for data persistence, and networks for connecting containers. With Docker Compose, you can easily configure each service's settings, including the image to use, the ports to expose, and the environment variables to set...

Overall, Docker Compose streamlines the development process, making it easier for you to build and deliver your applications with greater efficiency and ease.

A Docker Compose has **3 important parts**, which are:

- **Services:** A service is a unit of work in Docker Compose, it has a name, and it defines container images, a set of environment variables, and a set of ports that are exposed to the host machine. When you run docker-compose up, Docker will create a new container for each service in your Compose file.

- **Networks:** A network is a way for containers to communicate with each other. When you create a network in your Compose file, Docker will create a new network that all the other containers in your Compose file will be connected to. This allows containers to communicate with each other without even knowing the IP of each other, just by the name.

- **Volumes:** A volume is a way to store data that is shared between containers. When you create a volume in your Compose file, Docker will create a new volume (a folder in another way) that all the containers have access to. This allows you to share data between the containers without having to copy-paste each and every time you want that data.

Here's an example of a Docker Compose to make things a little bit clear:

```
version: '3'

```

```
# All the services that you will work with should be declared under the SERVICES section!
services:

  # Name of the first service (for example nginx)
  nginx:

    # The hostname of the service (will be the same as the service name!)
    hostname: nginx

    # Where the service exists (path) so you can build it
    build:
      context: ./requirements/nginx
      dockerfile: Dockerfile

    # Restart to always keep the service restarting in case of any unexpected errors causing it to go down
    restart: always

    # This line explains itself!!!
    depends_on:
      - wordpress

    # The ports that will be exposed and you will work with
    ports:
      - 443:443

    # The volumes that you will be mounted when the container gets built
    volumes:
      - wordpress:/var/www/html

    # The networks that the container will connect and communicate with the other containers
    networks:
      - web
```

How Does Docker Compose Work?

Docker Compose uses a YAML file to configure all the services, networks, and volumes that your application needs to. You can then with just a single command create and start all the services from your configuration. That's it. As simple as that.

What is the difference between a Docker image used with Docker Compose and without Docker Compose?

The main difference between the two is that Docker Compose provides a high level of abstraction for defining and managing multi-container applications, and it also simplifies the process of building, and running all the services (containers) in one take, which means instead of building the image and running the containers one by one, Docker Compose allows you do that for all the containers that you have in just one single command.
Using Docker directly without Docker Compose gives you more fine-grained control over individual containers, but it requires more work on your end to manage and deal with multiple containers.

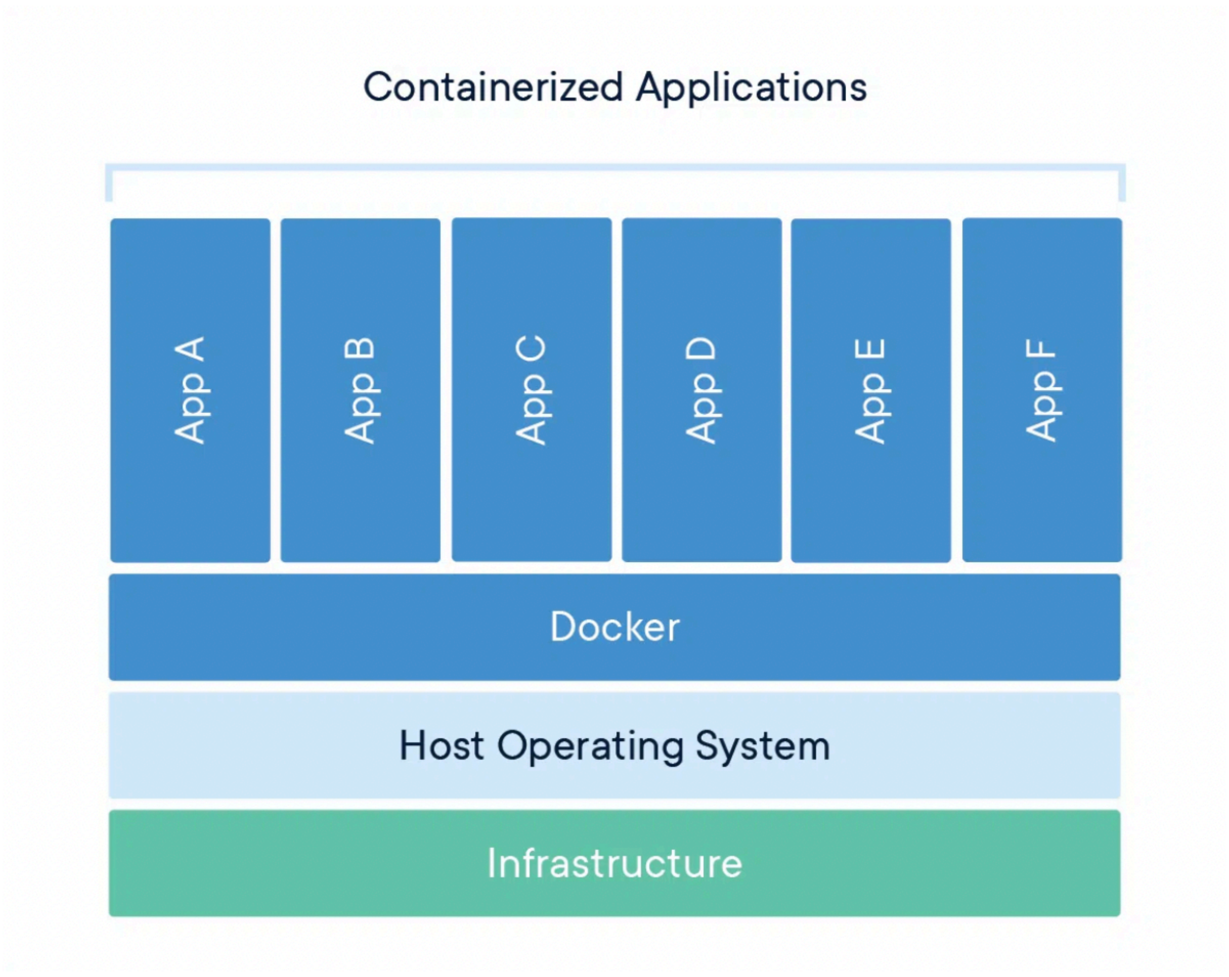
What is a Container?

A Container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

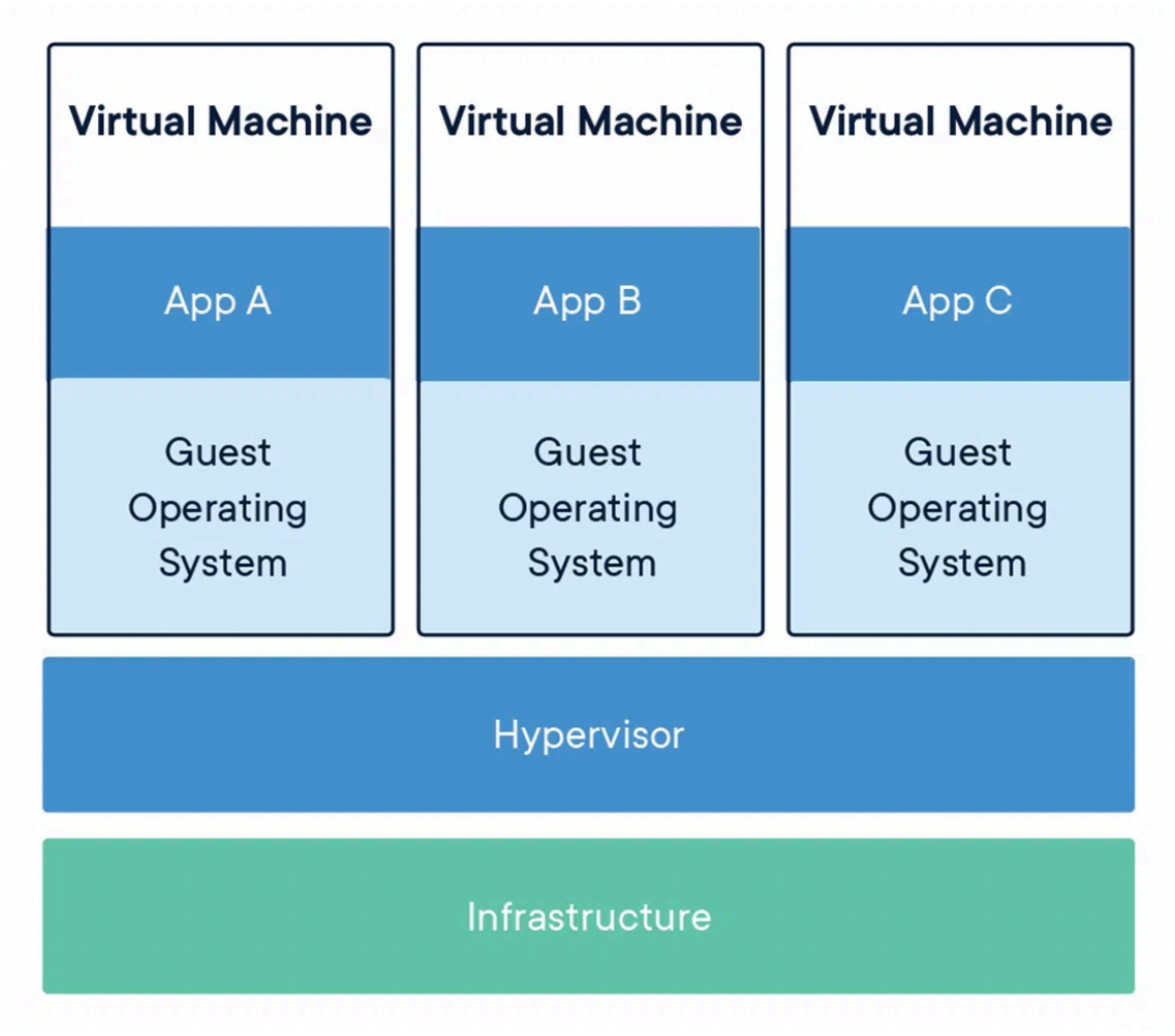
What is The Difference Between a Container and a VM?

Containers and Virtual machines have similar resource isolation and allocation benefits but function differently because containers virtualize the operating system instead of the hardware. Containers are more portable and efficient.

- **Containers** are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (talks about tens of MBs in size), can handle more applications, and require fewer VMs and Operating Systems.



- **Virtual Machines (VMs)** are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each machine includes a full copy of an operating system, the application, necessary binaries, and libraries. VMs take more space compared to containers (talking about tens of GBs in size), and they can be slow to boot.



What is PID 1?

In a Docker container, the PID 1 process is a special process that plays an important role in the container's lifecycle. This process is the identifier of the init process, which is the first process that is started when the system boots up, and it is responsible for starting and stopping all of the other processes on the system. And in Docker as well, the init process is responsible for starting and stopping the application that is running in the container.
PID 1 in a Docker container behaves differently from the init process in a normal Unix-based system. (they are NOT the same!)

Is the Daemon Process PID 1? And How Do They Differ From Each Other?

The daemon process is NOT the PID 1, the daemon process is a background process that runs continuously on a system and performs a specific task. In contrast, PID 1 is the first process that the kernel starts in a Unix-based system and plays a special role in the system.

What is WP-CLI?

WP-CLI is the command line interface for WordPress. It is a tool that allows you to interact with your WordPress site from the command line, it is used for a lot of purposes, such as automating tasks, debugging problems, installing/removing plugins alongside themes, managing users and roles, exporting/importing data, run database queries, and so much more...

Why Do You Need to Work With WP-CLI?

Since its a tool that allows you to interact directly with your WordPress site from the terminal, it can save you a lot of time that will take you to (for example) install a plugin/theme manually, moderate users and their roles, deploy a new WordPress website to a production server, etc...

WP-CLI will help you do all that in less time and is automated as well, so It's a really great tool that will help you react with your WordPress website.

Here's an example demonstrating how to run WP-CLI in a bash script to configure the WordPress environment:

```
#!/bin/sh

# We will first check if the "/var/www/html" folder exists or not, if not we create it
if [ ! -d "/var/www/html" ]; then
  mkdir /var/www/html
fi

# We will cd into the folder
cd /var/www/html

# This downloads the WordPress core files, the option ( --allow-root ) will run the command as root
# and ( --version:5.8.1 ) specifies the version of WordPress that will get downloaded
# and ( --local=en_US ) sets the language of the installation to US English
wp core download --allow-root --version=5.8.1 --locale=en_US

# This will generate the WordPress configuration file, and the options ( --dbname, --dbuser, --dbpass, --dbhost )
# are just placeholders that will get replaced once the script runs
wp config create --allow-root --dbname=${WP_NAME} --dbuser=${WP_ADMIN_USER} --dbpass=${WP_PASSWORD} --dbhost=${WP_HOST}

# This will then install WordPress, and again, all the options are just placeholders that will get replaced
```

```
wp core install --allow-root --url=${WP_URL} --title=${WP_TITLE} --admin_user=${WP_ADMIN_USER} --admin_password=${WP_ADMIN_PASSWORD} --admin_email=${WP_ADMIN_EMAIL}

# This create a new WordPress user, and sets its role to author ( --role=author )
wp user create "${WP_USER}" "${WP_EMAIL}" --user_pass="${WP_PASSWORD}" --role=author

# This is the command that will keep WordPress up and running
exec php-fpm7 -F -R
```

What is Redis Cache? And Why Do You Need it in WordPress?

Redis Cache, or Redis Object Cache, is an open-source, in-memory data structure store that can be used as a database, cache, or message broker. It's a plugin for WordPress that improves the performance of your website by storing accessed data in memory, rather than querying the database each and every time that data is needed.
You need Redis Cache plugin for your WordPress website because it can improve the performance of the website and reduce the time that it takes to load all the data from your databases, which will cause in a better user experience, plus it will help your website rank higher in search engines, etc...
In this project of Inception, in order for Redis Cache to work perfectly, you need to add this line in your WordPress script:

- If you're using Alpine: chown -R nobody:nobody *
- If you're using Debian: chown -R www-data:www-data *

What this basically does is that it changes the ownership of all files and directories recursively (-R) to the user and group nobody:nobody or www-data:www-data
Plus, you will also need to add the REDIS_HOST as long as the REDIS_PORT to your WordPress Configuration File wp-config.php
Here's an example:
sed -i "41 i define('WP_REDIS_HOST', 'redis');\ndefine('WP_REDIS_PORT', '6379');\n" wp-config.php
This will add 2 lines to the wp-config.php in the 41st line of the file, the first is define('WP_REDIS_HOST', 'redis'); and the second is define('WP_REDIS_PORT', '6379');. Those 2 lines will specify the host and port that Redis Cache will use to establish the connection.
Adding them using echo seems to cause some problems, and Redis Cache does not want to establish the connection!

What is FTP? And How Does it Work?

FTP or File Transfer Protocol is a protocol that's used for transferring files between a client and a server over TCP/IP network, such as the Internet. It provides a robust mechanism for users to upload, download, and manage files on remote servers.
FTP works by opening two connections that link the 2 hosts (client and server)

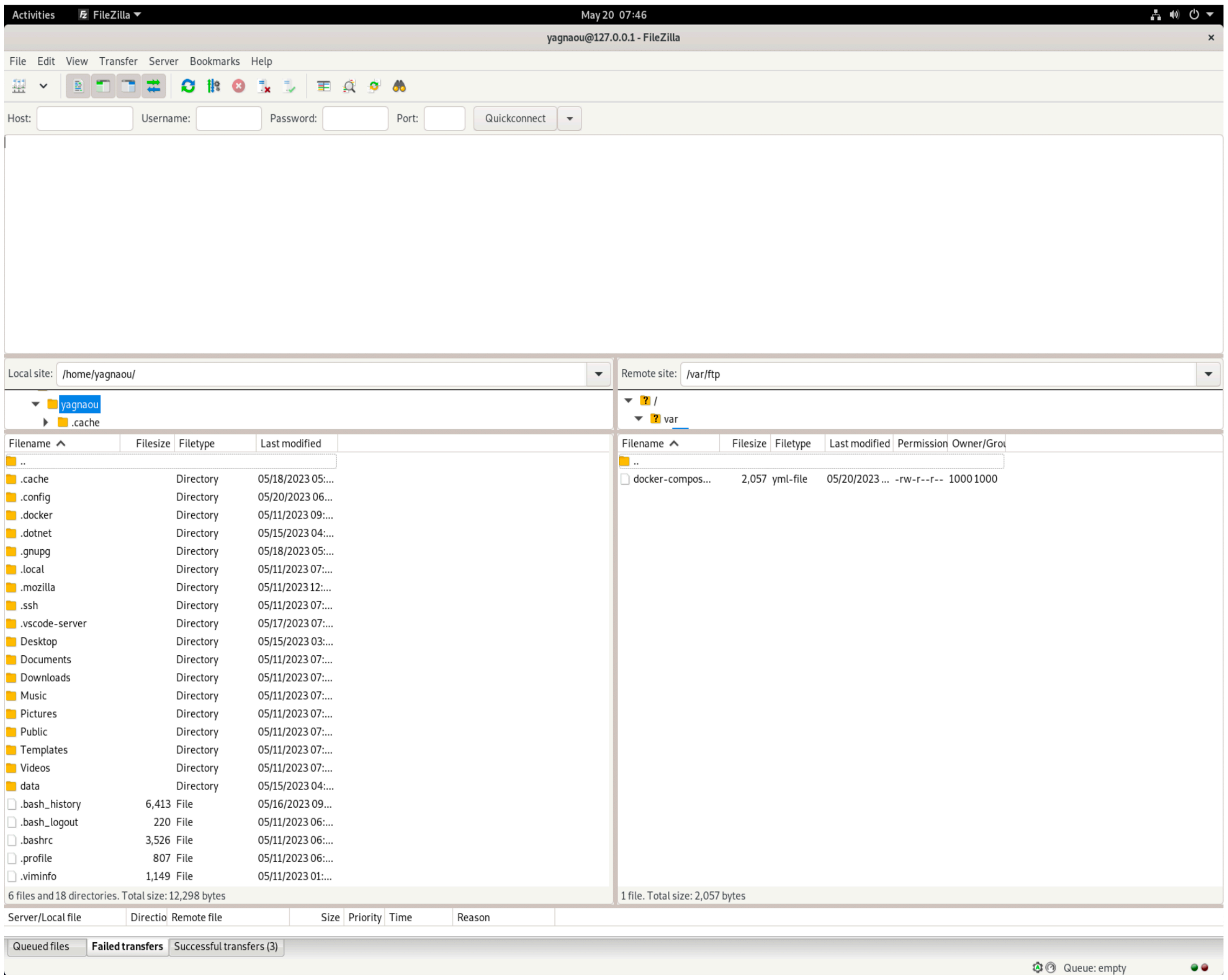
trying to communicate with each other, one connection is designed for the commands and replies that get sent between the two clients, and the other connection handles the transfer of the data.

How to Configure FTP? And How to Test If It's Working?

Before configuring and testing if FTP is working, you need to install FileZilla or other alternatives on your machine, which will allow you to transfer files between the two hosts easily and make sure that everything works as it should. I'll use FileZilla on my end

sudo apt-get install filezilla

Once installed, open it by typing the filezilla command in your terminal. A window like this will then popup


Activities
FileZilla
May 20 07:46
yagnaou@127.0.0.1 - FileZilla
File Edit View Transfer Server Bookmarks Help
Host:
Username:
Password:
Port:
Quickconnect

Local site: /home/yagnaou/

Filename	Filesize	Filetype	Last modified
..			
.cache		Directory	05/18/2023 05:...
.config		Directory	05/20/2023 06...
.docker		Directory	05/11/2023 09:...
.dotnet		Directory	05/15/2023 04:...
.gnupg		Directory	05/18/2023 05:...
.local		Directory	05/11/2023 07:...
.mozilla		Directory	05/11/2023 12:...
.ssh		Directory	05/11/2023 07:...
.vscode-server		Directory	05/17/2023 07:...
Desktop		Directory	05/15/2023 03:...
Documents		Directory	05/11/2023 07:...
Downloads		Directory	05/11/2023 07:...
Music		Directory	05/11/2023 07:...
Pictures		Directory	05/11/2023 07:...
Public		Directory	05/11/2023 07:...
Templates		Directory	05/11/2023 07:...
Videos		Directory	05/11/2023 07:...
data		Directory	05/15/2023 04:...
.bash_history	6,413	File	05/16/2023 09...
.bash_logout	220	File	05/11/2023 06:...
.bashrc	3,526	File	05/11/2023 06:...
.profile	807	File	05/11/2023 06:...
.viminfo	1,149	File	05/11/2023 01:...

6 files and 18 directories. Total size: 12,298 bytes

Remote site: /var/ftp

Filename	Filesize	Filetype	Last modified	Permission	Owner/Grou
..					
docker-compos...	2,057	yml-file	05/20/2023 ...	-rw-r--r--	1000 1000

1 file. Total size: 2,057 bytes

Server/Local file	Directio	Remote file	Size	Priority	Time	Reason

Queued files | Failed transfers | Successful transfers (3)

Queue: empty

Now you have to configure the FTP so you can connect to the FileZilla, then, and only then you can start testing and transferring the data.
First, you will begin by installing the package needed for the FTP and copying the config file to its proper directory. Here's the Dockerfile for the FTP

```
FROM alpine:3.14
```

```

# Installing the VSFTPD
RUN apk update && \
    apk add vsftpd

# Copying the config file to its proper directory
COPY ./conf/vsftpd.conf /etc/vsftpd/vsftpd.conf

# Here, you will create a new user, and give it the permissions needed so you can connect to FileZilla using it
RUN adduser -D -h /var/ftp USER_NAME && \
    echo "USER_NAME:USER_PASSWORD" | chpasswd && \
    mkdir -p /var/ftp && \
    chown -R USER_NAME:USER_NAME /var/ftp && \
    chmod 755 /var/ftp

CMD ["vsftpd", "/etc/vsftpd/vsftpd.conf"]
```

And here's the config file for the FTP

```
# This line specifies that the FTP server should listen for incoming connections
listen=YES

# This line disables anonymous access to the FTP server
# (ONLY authenticated users can get access)
anonymous_enable=NO

# This line enables local user access to the FTP server
local_enable=YES

# This line enables write access for authenticated users
# (authenticated users can upload/modify files on the FTP)
write_enable=YES

# This line tells the FTP to use the local system's time
local_umask=022

# This line enables the display of directory welcome messages
# (FTP will display a message when a user enters a directory)
```

```
dirmessage_enable=YES

# This line instructs the FTP server to use the local system's time settings
use_localtime=YES

# This line enables the logging of file transfer activities
# (FTP will create a log file containing info about data transferring)
xferlog_enable=YES

# This line specifies whether the FTP server should use port 20 for active data connections
connect_from_port_20=NO

# This line disables the use of a seccomp sandbox
# (Seccomp is a security mechanism that restricts the system calls available to a process)
seccomp_sandbox=NO
```

Now after setting everything up, open FileZilla, and connect using the following:

- Host: 127.0.0.1
- Username: USER_NAME
- Password: USER_PASSWORD
- Port: The one you specified in the Docker Compose, will usually be 21

If everything is done correctly, you will be able to connect to FileZilla without any error, and then you can start transferring data as you wish!

What is Adminer?

Adminer is a free open-source tool that allows you to easily view, edit, create, and modify databases through a user-friendly interface. It supports a wide range of database systems, such as MariaDB, MySQL, PostgreSQL, SQLite, and many more... It is a single file application that doesn't require any installation, and that is what makes Adminer stand out and be preferred as a database manager among all the other alternatives.