UNIT 5: INSTALLATION

1. PHP FRAMEWORKS

A **framework** is a tool that provides a number of modules that help organize and develop a software product.
In the specific case of PHP frameworks, most of them provide a series of commands or tools to create projects with a certain structure (usually, following the MVC pattern that we will see later), so that they already give a base of work done, and facilities to be able to create the data model, the connection to the database, the paths of the different sections of the application, etc.

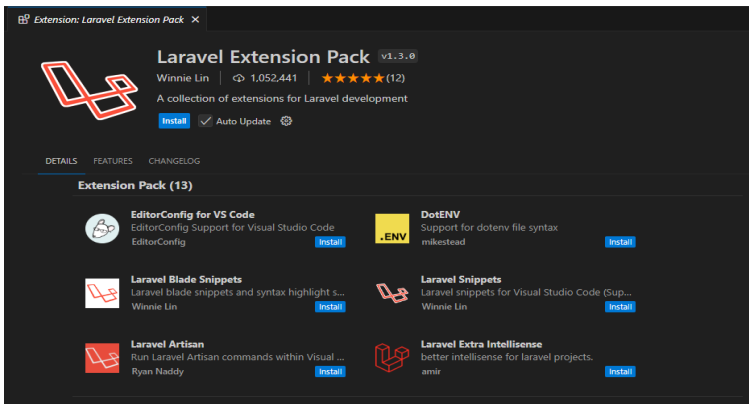
1.1. Previous resources

When working with Laravel, we need to have previously installed on our system a series of software resources, such as:
1. An IDE (development environment) with which to edit the code of our projects. We will use Visual Studio Code in these notes, although there are other similar alternatives, such as PHPStorm, Sublime Text, etc.
2. A web server that supports PHP.
3. A database server on which to store the information of our applications. We will use a MariaDB/MySQL server.
4. PHP
5. The Laravel framework itself.
6. In addition, we will need the 'npm' package manager to install client-side dependencies on Laravel projects.

In this document you have the steps to install each of these elements using a docker.

1.2. Visual Studio Code: Useful extensions

When we use Visual Studio Code to develop applications in a certain technology (Laravel, Node, etc.), it may be convenient to install a *plugin* or extension that facilitates the development of these applications. Through these extensions we can, for example, highlight the syntax of the files we edit, help us autocomplete code, etc.
In the case of Laravel, we can go to the *Extensions* section of the left menu of Visual Studio Code, and look for the extension *Laravel Extension Pack*. Then, click on the *Install* button to install this extension, which in turn contains a package of extensions useful for developing Laravel applications.



Specifically, we can highlight the syntax of the HTML views that we make with the Blade template engine, autocomplete certain parts of the code, auto-correct certain code errors, etc.

Also install the extensions: Dev containers and Docker if you want to manage docker from Visual Studio Code

2. ENVIRONMENT PREPARATION (optional)

1. Create an Ubuntu 22 virtual machine: ubuntu-22.04.2-desktop-amd64
2. Install Docker Engine

You can download an Ubuntu 22 virtual machine virtual from teams with Docker Engine installed. (user:marta and password: 123). So, you can skip this step and go directly to 3. INSTALLATION LARAVEL DOCKER AND CREATION OF CONTAINERS

2.1. Installation methods

Install Docker Engine on Ubuntu | Docker Documentation

You can install Docker Engine in different ways, depending on your needs:

1. Docker Engine comes bundled with Docker Desktop for Linux. This is the easiest and quickest way to get started.
2. **Set up and install Docker Engine from Docker's apt repository**.
3. Install it manually and manage upgrades manually.
4. Use a convenience script. Only recommended for testing and development environments.

We will use the 2n method.

2.1.1. Install using the apt repository

Install Docker Engine on Ubuntu | Docker Docs
Before you install Docker Engine for the first time on a new host machine, you need to set up the Docker repository. Afterward, you can install and update Docker from the repository.

Set up the repository

- Update the `apt` package index and install packages to allow `apt` to use a repository over HTTPS:

```
$ sudo apt-get update
```

```
$ sudo apt-get install ca-certificates curl gnupg
```

- Add Docker's official GPG key:

```
$ sudo install -m 0755 -d /etc/apt/keyrings
```

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg
--dearmor -o /etc/apt/keyrings/docker.gpg
```

```
$ sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

- Use the following command to set up the repository:

```
$ echo \
  "deb [arch="$(dpkg --print-architecture)" signed-
by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Note

If you use an Ubuntu derivative distro, such as Linux Mint, you may need to use UBUNTU_CODENAME instead of VERSION_CODENAME.

Install Docker Engine

- Update the `apt` package index:

```
$ sudo apt-get update
```

- Install Docker Engine, containerd, and Docker Compose.

To install the latest version, run:

```
 $ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-
plugin docker-compose-plugin
```

- Verify that the Docker Engine installation is successful by running the `hello-world` image.

```
$ sudo docker run hello-world
```

This command downloads a test image and runs it in a container. When the container runs, it prints a confirmation message and exits.
You have now successfully installed and started Docker Engine.

Post-installation

Linux post-installation steps for Docker Engine | Docker Documentation

Receiving errors when trying to run without root? If you execute "docker ps" ... You do not have permission

The `docker` user group exists but contains no users, which is why you're required to use `sudo` to run Docker commands. Continue to allow non-privileged users to run Docker commands and for other optional configuration steps.

To create the `docker` group and add your user:

- Create the `docker` group.

```
$ sudo groupadd docker (This step is not necessary because the group has already been created)
```

- Add your user to the `docker` group.

```
$ sudo usermod -aG docker $USER
```

- Log out and log back in so that your group membership is re-evaluated.

  If you're running Linux in a virtual machine, it may be necessary to restart the virtual machine for changes to take effect.

You can also run the following command to activate the changes to groups:

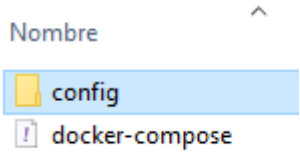
```
$ newgrp docker
```

- Verify that you can run `docker` commands without `sudo`.

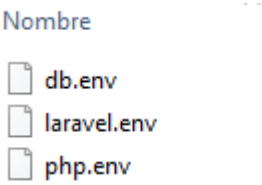
```
$ docker run hello-world
```

3. INSTALLATION LARAVEL DOCKER AND CREATION OF CONTAINERS:

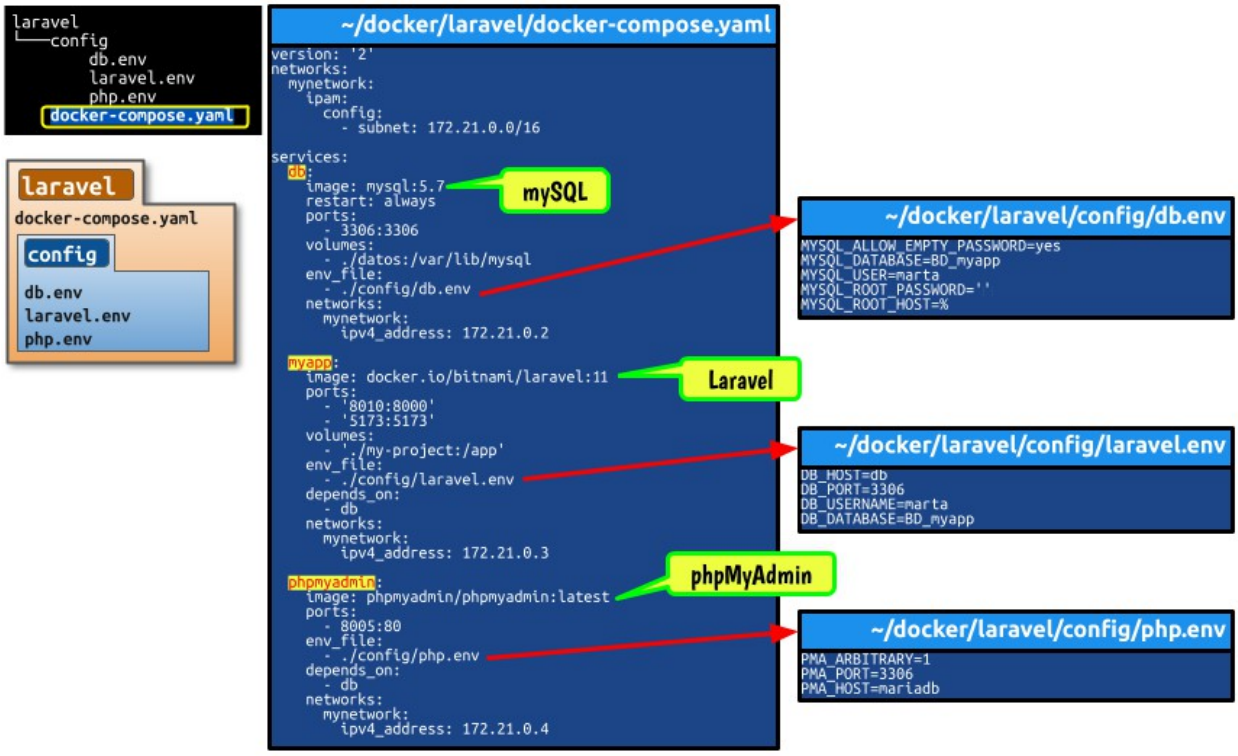
Create the folder laravel in /home/user and copy the docker files:



config folder with



Information docker-compose:



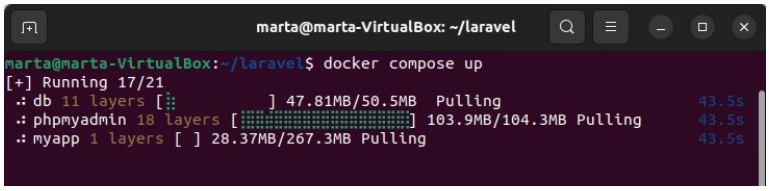
- db: network: 172.21.0.2
This is the container with the data in the DB.

- myapp: network: 172.21.0.3
This is the container with the Laravel project
It will be accessible in port http://localhost:8010

- phpmyadmin: network: 172.21.0.4
This is the container with phpMyadmin as database management system
It will be accessible in port http://localhost:8005

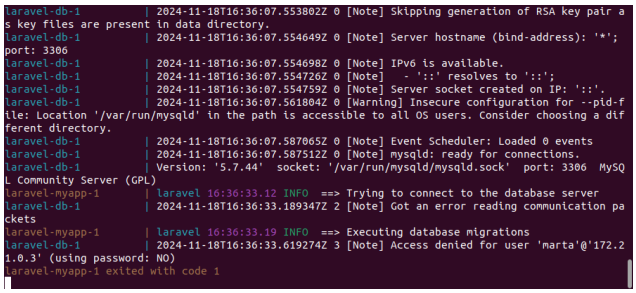
Open a terminal 1, Go to the laravel folder and execute:

```
$ docker compose up
```



It stops "trying to connect the database"
Access denied for user 'marta'@'172.21.0.3' (using password: NO) means that the machine Laravel is going to use the user marta to connect to the machine mysql.

It gives an access error for user marta, so let's fix that



Don't close terminal 1.

Open a new terminal 2,

Go to the laravel folder and enter the laravel container in interactive mode executing:

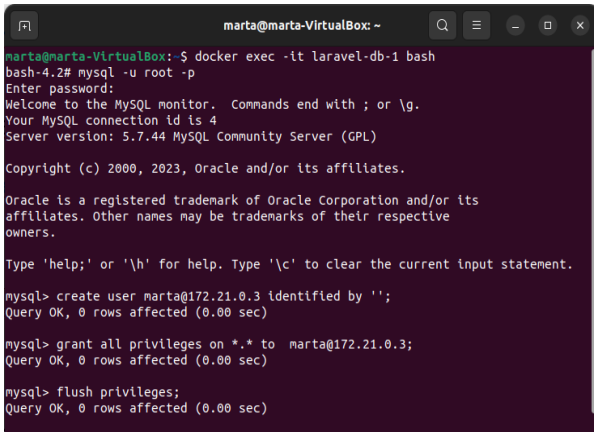
```
$ docker exec -it laravel-db-1 bash
```

and then....

```
bash-4.2#mysql -u root -p
```

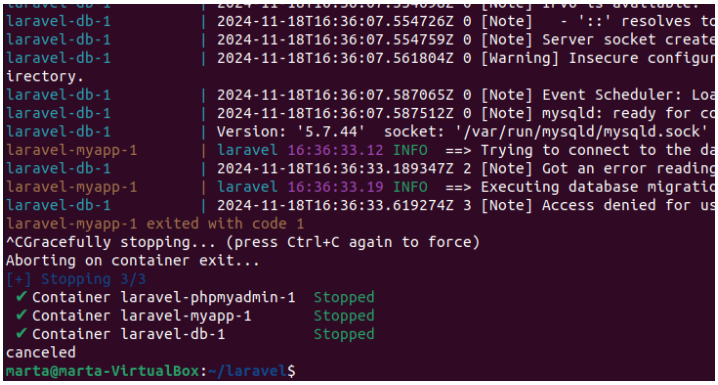
password= '' (like in db.env)

```
mysql>create user marta@172.21.0.3 identified by ''; //172.21.0.3 docker's ip

mysql>grant all privileges on *.* to  marta@172.21.0.3;

mysql>flush privileges;
```



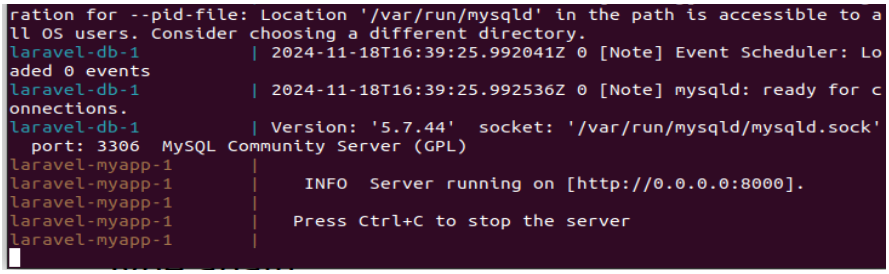
Close terminal 2

Return to terminal 1 press Ctrl + C in order to stop the containers.



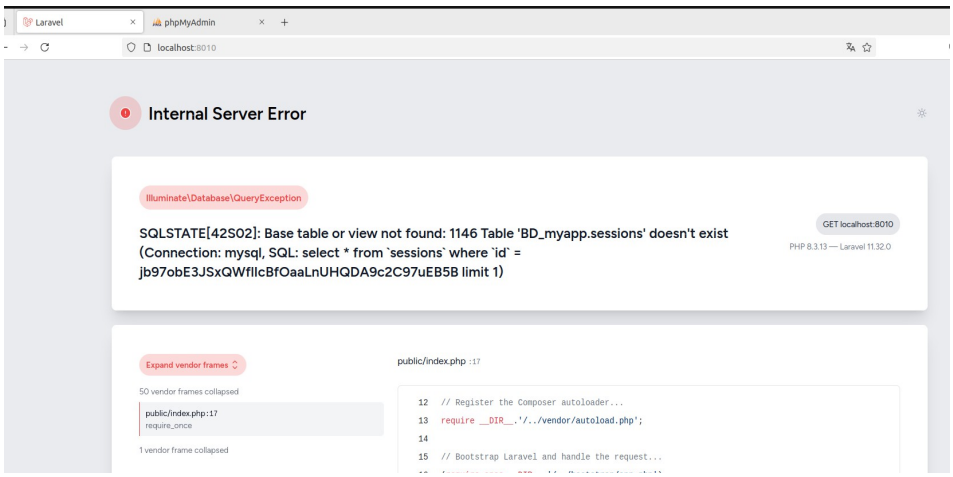
Type again:

```
$ docker compose up
```



Don't close this terminal, while you are working with your laravel project

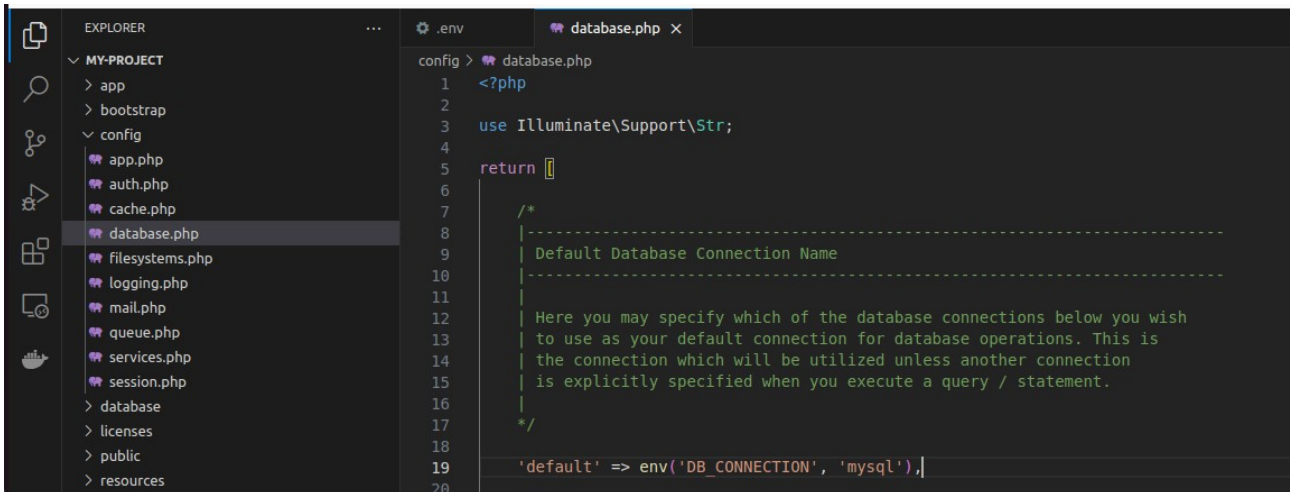
Our Laravel project is working at http://localhost:8010/
But with an error:



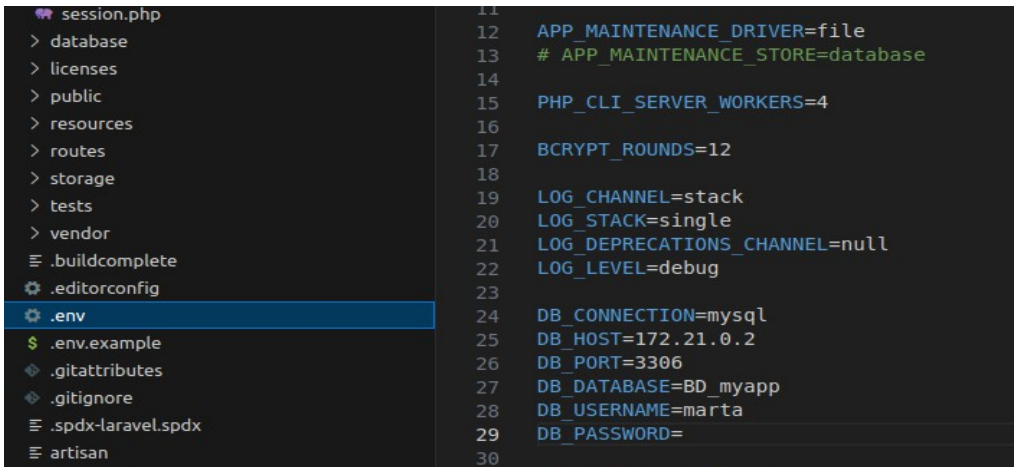
This is because we have to configurate the DB

Open your project folder “my-project” (inside laravel) with the Visual Studio Code.

In config/database.php
'default' => env('DB_CONNECTION', 'mysql'),

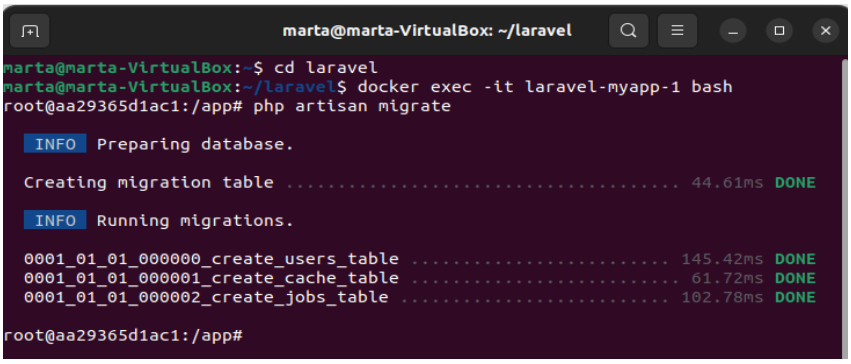


In .env
DB_CONNECTION=mysql
DB_HOST=172.21.0.2
DB_PORT=3306
DB_DATABASE=BD_myapp
DB_USERNAME=marta
DB_PASSWORD=

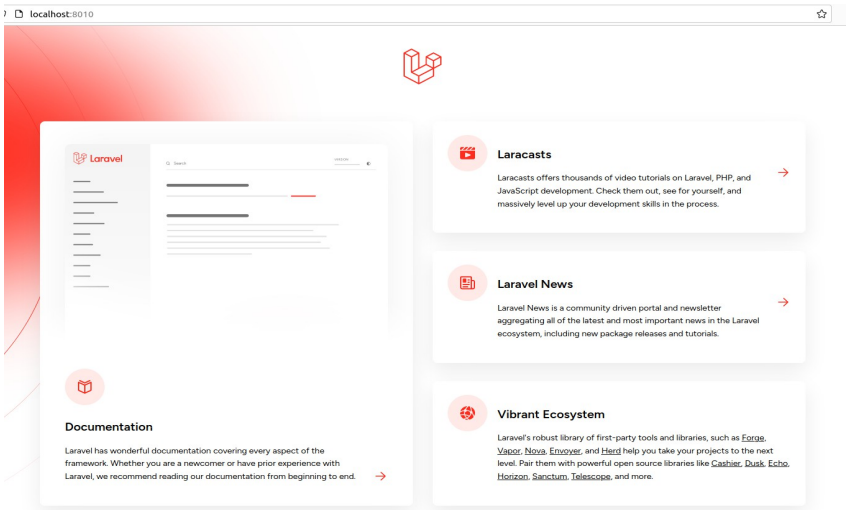


From another terminal:
docker exec -it laravel-myapp-1 bash
php artisan migrate

And you can close the terminal

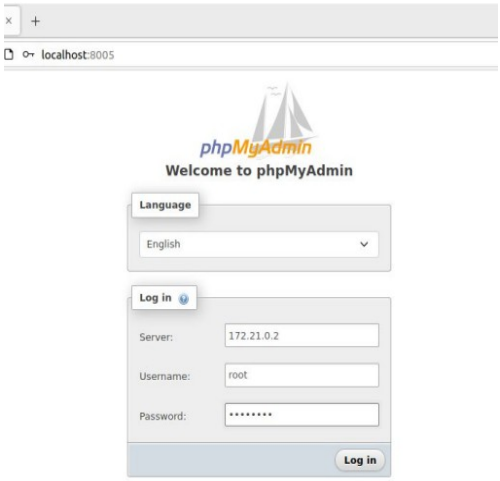


Then, your project is working:

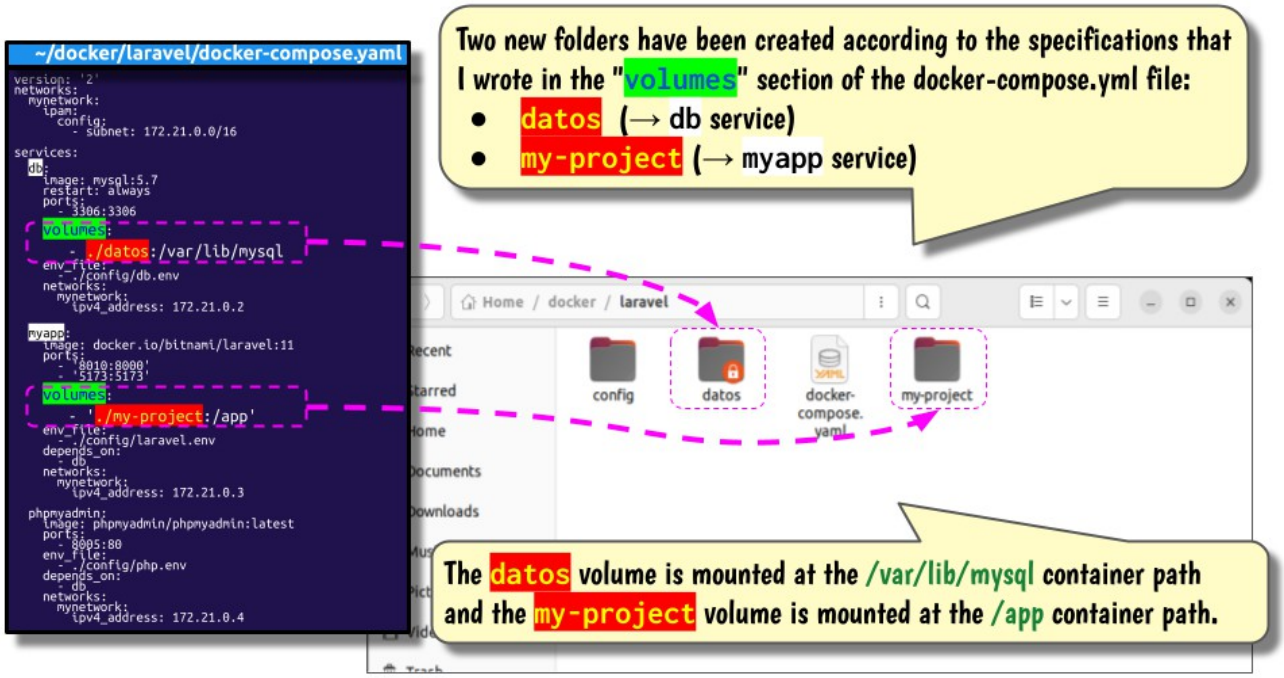


To access phpmyadmin: http://localhost:8005/

Server: 172.21.0.2
Username: root
Password: ''



If we open the folder laravel, we can see there are 2 new folders: datos and my-project. They are the volumes for the database and the Laravel project respectively.



To stop the project, from the terminal opened:

```
$ docker compose down
```

Rename my-project folder to library

Make a duplicate from folder library and name it blog

We are going to work with two projects during the course:

- library: Project about books and authors. We will do this project together. I will make a video for you with all the practices, and you will repeat it at home
- blog: This will be the project that you will deliver every fortnight.
  IMPORTANT: And you will have to bring it for the exam, since it will be a mandatory validation practice in the exam

To work with both projects, you must change the name of the project you want to work with in docker-compose.yml. For example if you want to work with library:

In docker-compose.yml change:

```
  volumes:
      - './my-project:/app'
```

For:

```
  volumes:
      - './library:/app'
```