Inés Larrañaga Fernández de Pinedo	CSC Jesuitas - Logroño
	DWES

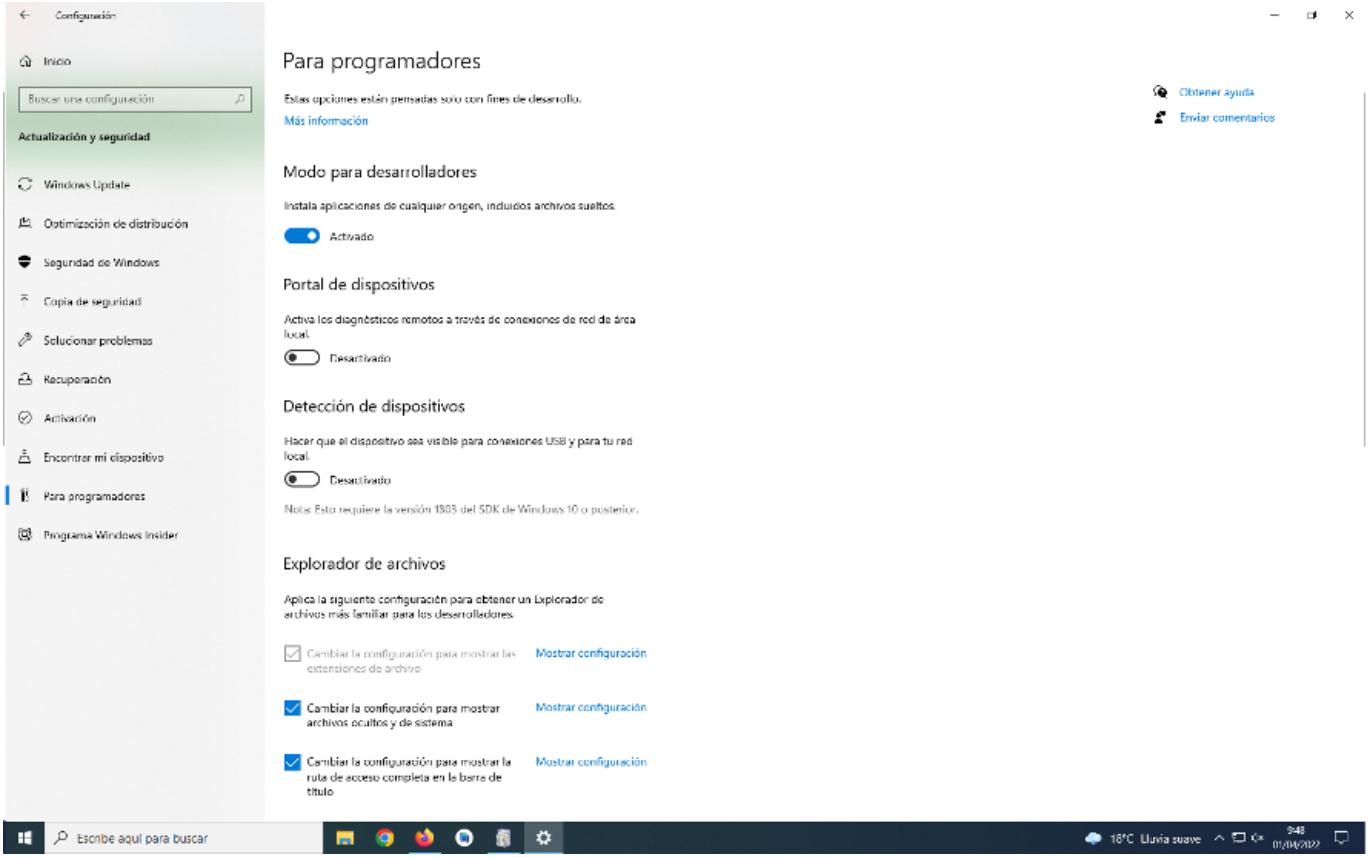
Table of Contents

Development environment setup

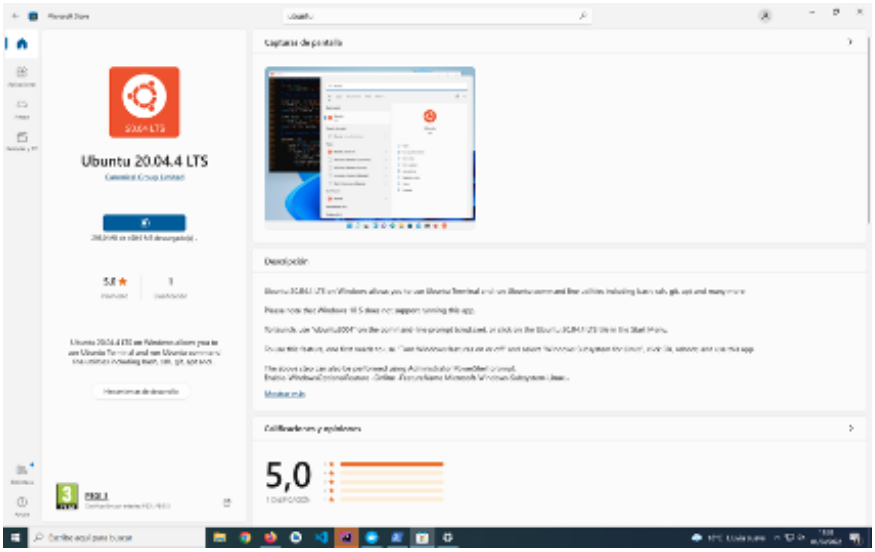
You should have this software/requirements installed and configured in your computers:

WSL (Windows subsystem for Linux)

Follow [these](Manual installation steps for older versions of WSL | Microsoft Docs) steps. Remember having the Windows `developers mode` activated:



Once you enable it, install through `Microsoft store`, ubuntu 20:

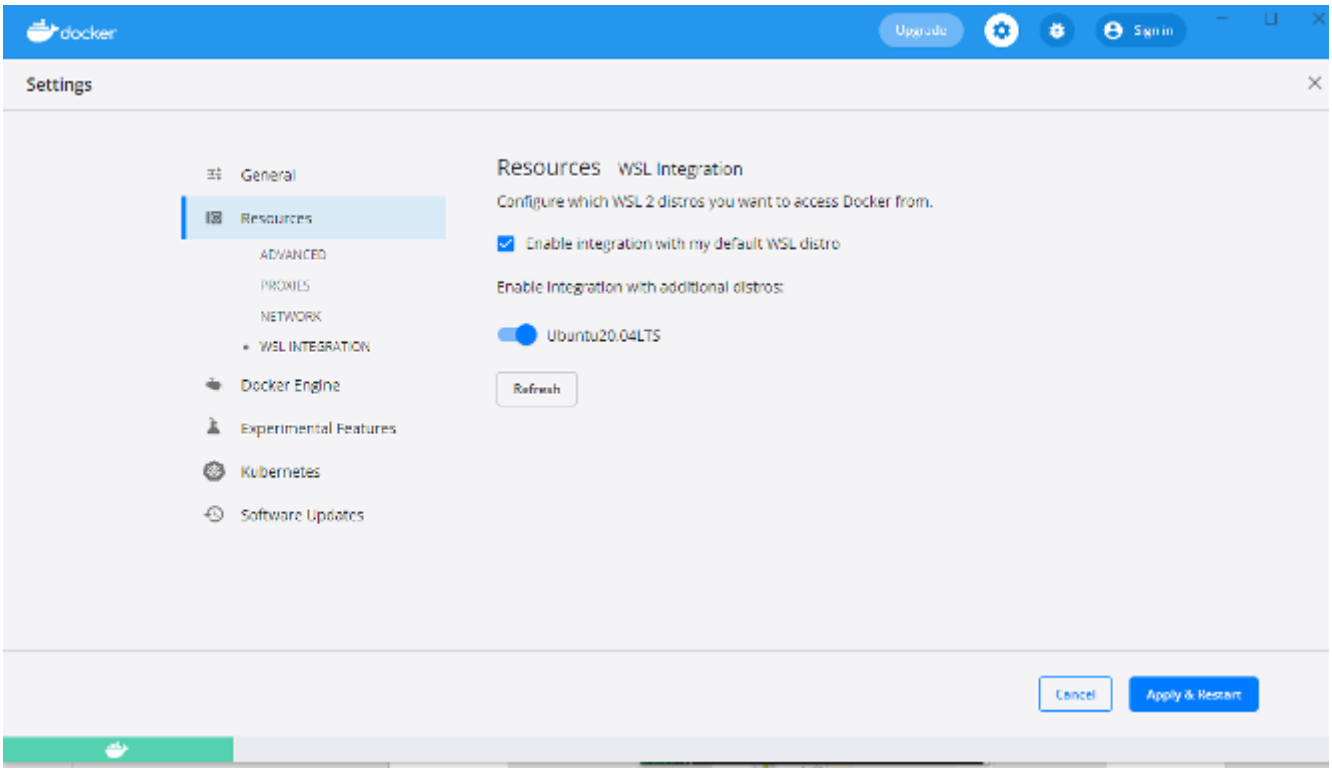


Open it, and a Powershell terminal should be openned. The first time, the ubuntu distribution will be installed properly in your machine.

Docker desktop.

Install it following this [link](Install Docker Desktop on Windows | Docker Documentation).

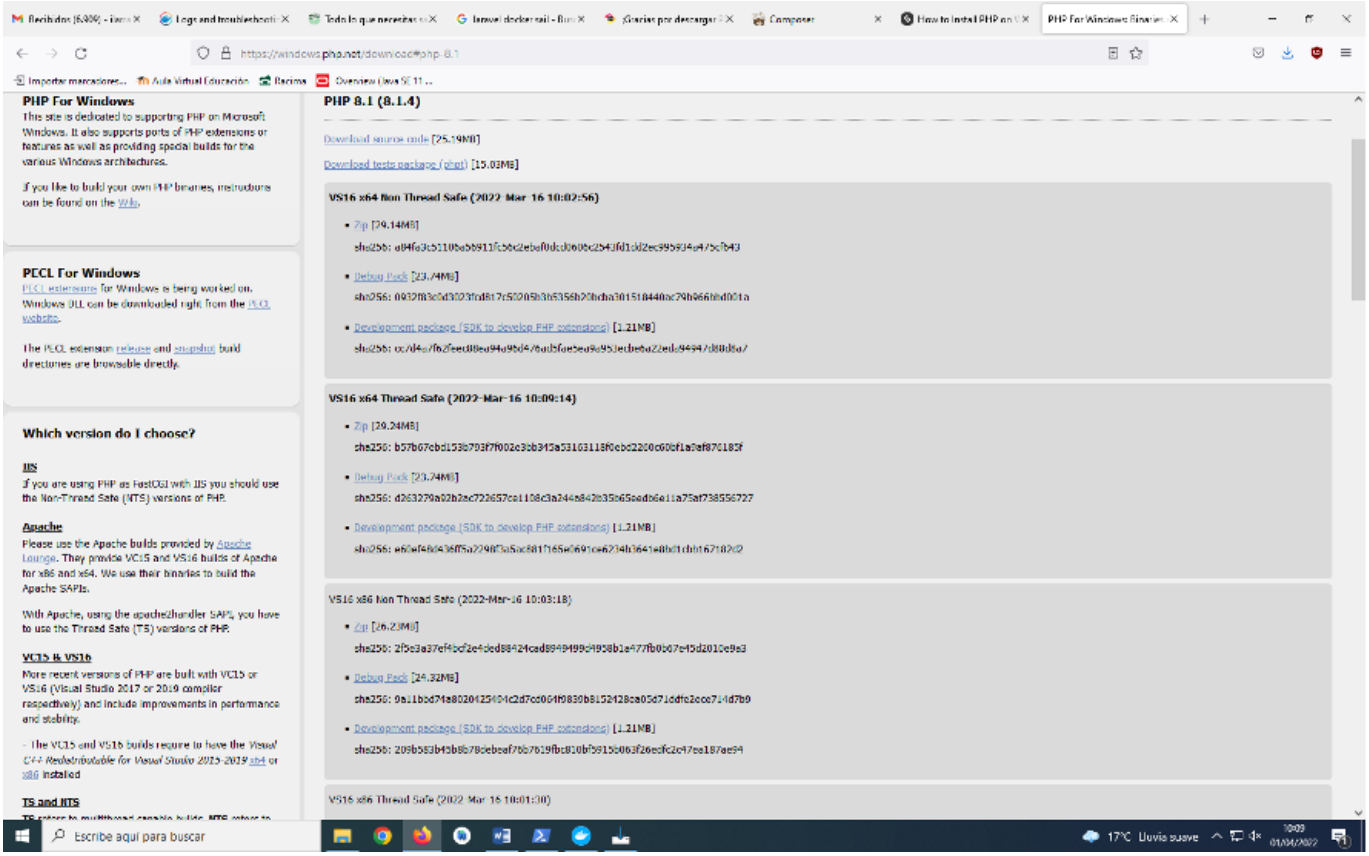
Remember setting your Docker desktop like this in order to integrate it with your ubuntu WSL:



PHP 8 windows installation

Step 1: Download the PHP files

You'll need the PHP Windows installer. There are a number of versions of PHP available. Make sure you get the latest PHP 8 **x64 Thread Safe** ZIP package from PHP: Downloads.



Step 2: Extract the files

Create a new php folder in the root of your `C:\` drive and extract the contents of the ZIP into it.

PHP can be installed anywhere on your system, but you'll need to change the paths referenced below if `C:\php` isn't used.

Step 3: Configure php.ini

PHP's configuration file is named `php.ini`. This doesn't exist initially, so copy `C:\php\php.ini-development` to `C:\php\php.ini`. This default configuration provides a development setup which reports all PHP errors and warnings.

There are several lines you may need to change in a text editor (use search to find the current value). In most cases, you'll need to remove a leading semicolon (;) to uncomment a setting.

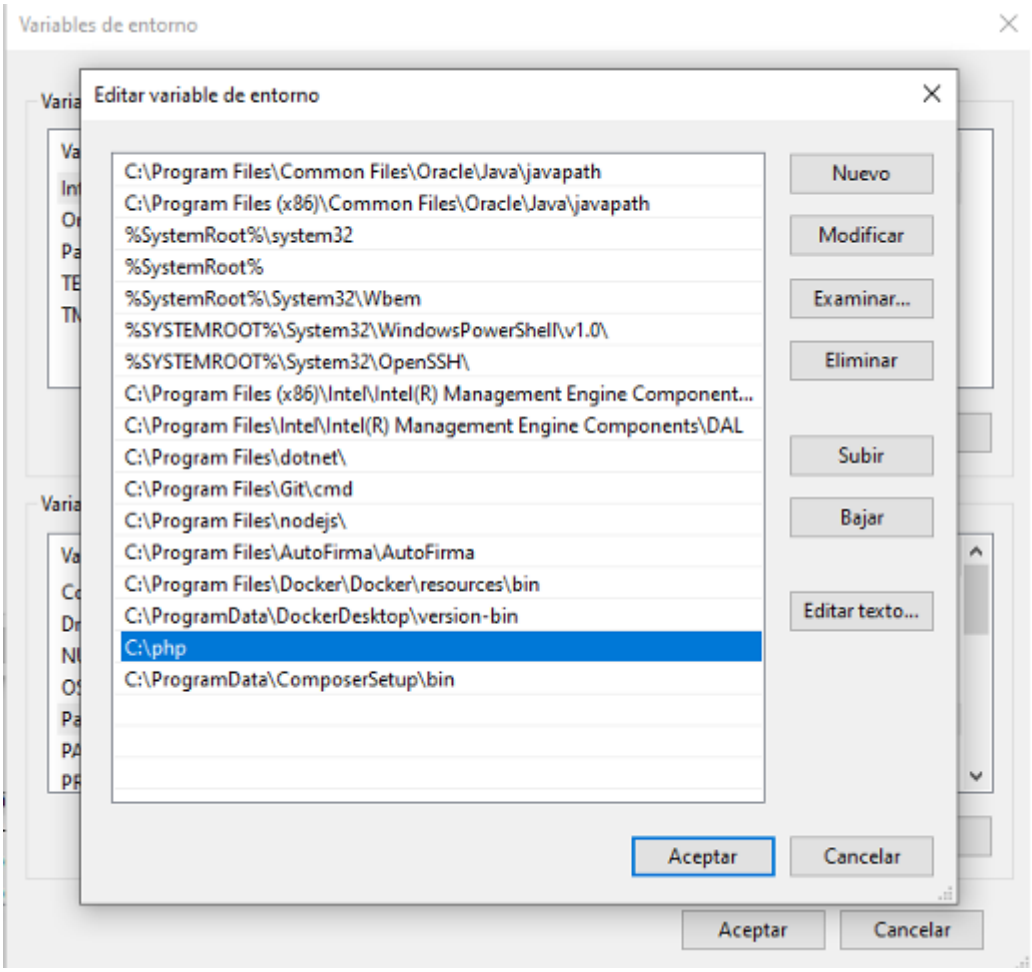
First, enable any required extensions. This will depend on the libraries you want to use, but the following extensions should be suitable for most applications:

```
php.ini
C: > php > php.ini
; 'extension='php_<ext>.dll') is supported for legacy reasons and may be
; deprecated in a future PHP major version. So, when it is possible, please
; move to the new ('extension=<ext>) syntax.
;
; Notes for Windows environments :
;
; - Many DLL files are located in the extensions/ (PHP 4) or ext/ (PHP 5+)
;   extension folders as well as the separate PECL DLL download (PHP 5+).
;   Be sure to appropriately set the extension_dir directive.
;
;extension=bz2
extension=curl
;extension=ffi
;extension=ftp
extension=fileinfo
extension=gd
;extension=gettext
;extension=gmp
extension=intl
;extension=imap
;extension=ldap
extension=mbstring
;extension=exif      ; Must be after mbstring as it depends on it
;extension=mysqli
;extension=oci8_12c  ; Use with Oracle Database 12c Instant Client
;extension=oci8_19   ; Use with Oracle Database 19 Instant Client
;extension=odbc
extension=openssl
;extension=pdo_firebird
extension=pdo_mysql
;extension=pdo_oci
;extension=pdo_odbc
extension=pdo_pgsql
;extension=pdo_sqlite
;extension=pgsql
```

Step 4: Add `C:\php` to the path environment variable

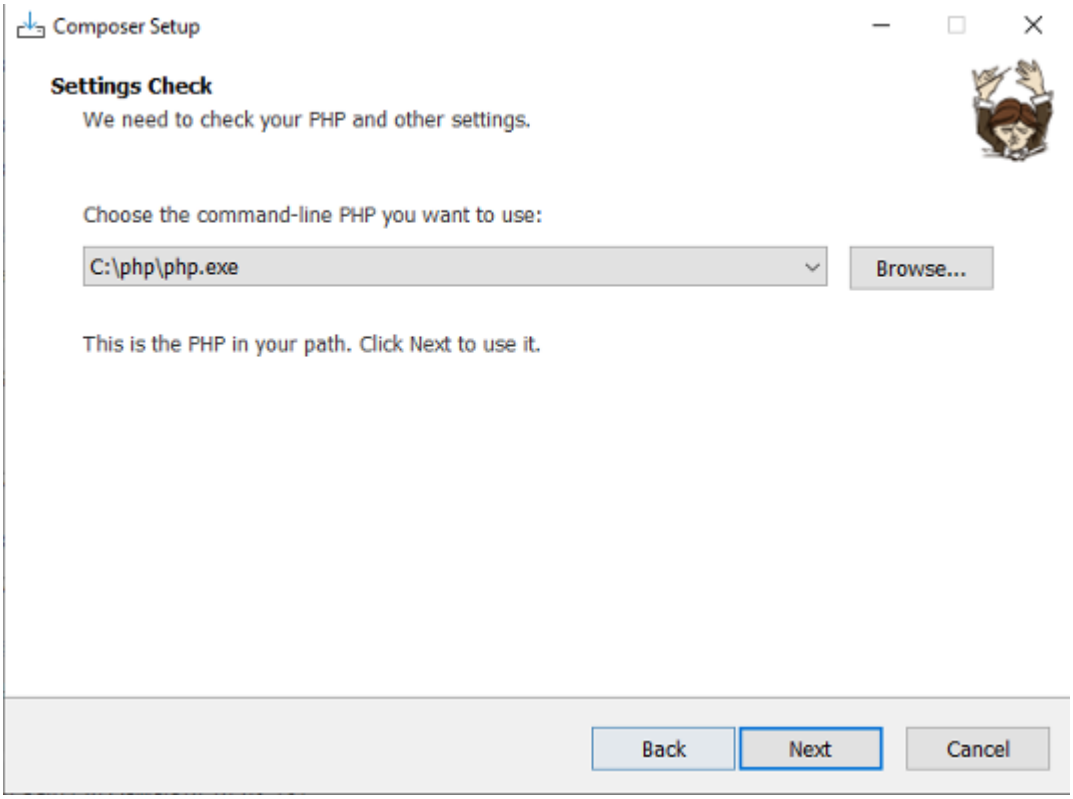
To ensure Windows can find the PHP executable, you need to change the PATH environment variable. Click the Windows Start button and type *"environment"*, then click **Edit the system environment variables**. Select the **Advanced** tab, and click the **Environment Variables** button.

Scroll down the **System variables** list and click **Path** followed by the **Edit** button. Click **New** and add `C:\php`:



Composer

Follow this link.



PHPStorm

Follow this [link](Download PhpStorm: Lightning-Smart PHP IDE). Remenber asking for the student license.

Github account

PHPStorm - a new laravel project

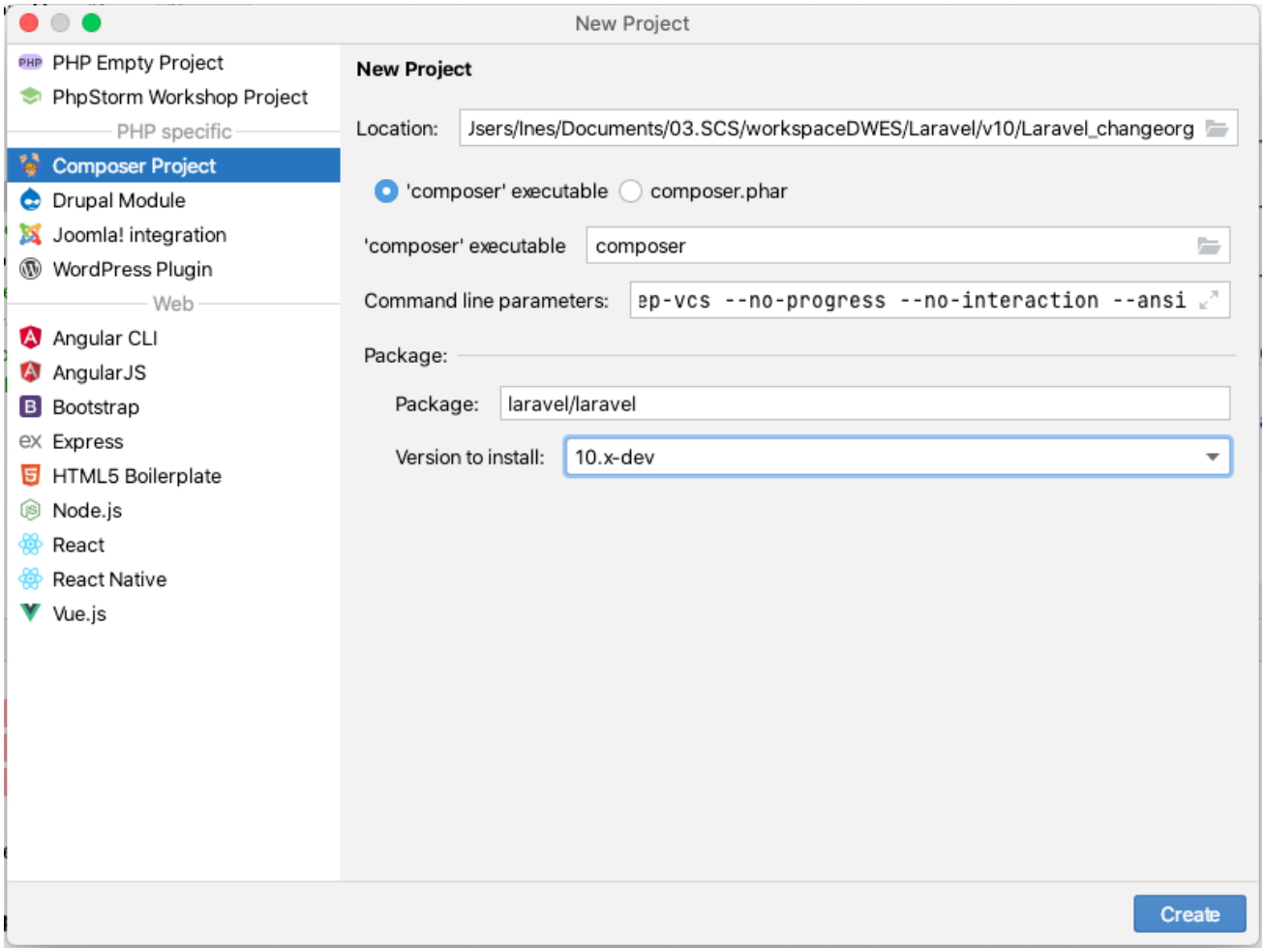
Step 1: Configure Composer in PHPStorm

- Go to Settings -> Languages & Frameworks -> PHP -> Composer
- Set the PHP executable path to `C:\php\php.exe`

Step 2: Create new Laravel project in PHPStorm (via Composer)

- Go to File -> New Project
- Select `Composer Project` on the left-hand side
- Enter the Location you want Laravel to be installed in your workspaceDWES folder with this name: `Laravel_changeorg`.
- Select `Use existing composer.phar` and ensure the path is there

- Search for `laravel/laravel` in `Filter packages` and select it.



- Ensure the path to the PHP executable is also set below (scroll down if you don't see it)
- Click `Create`
- Add `/.idea` to your .gitignore file

Laravel sail

When creating a new Laravel application (version 8 or up), [Laravel Sail](Laravel Sail - Laravel - The PHP Framework For Web Artisans) gets installed automatically. But if you already have an existing application, you will have to go through a couple of steps:

1. Require it using `composer` :

```
composer require laravel/sail --dev
```

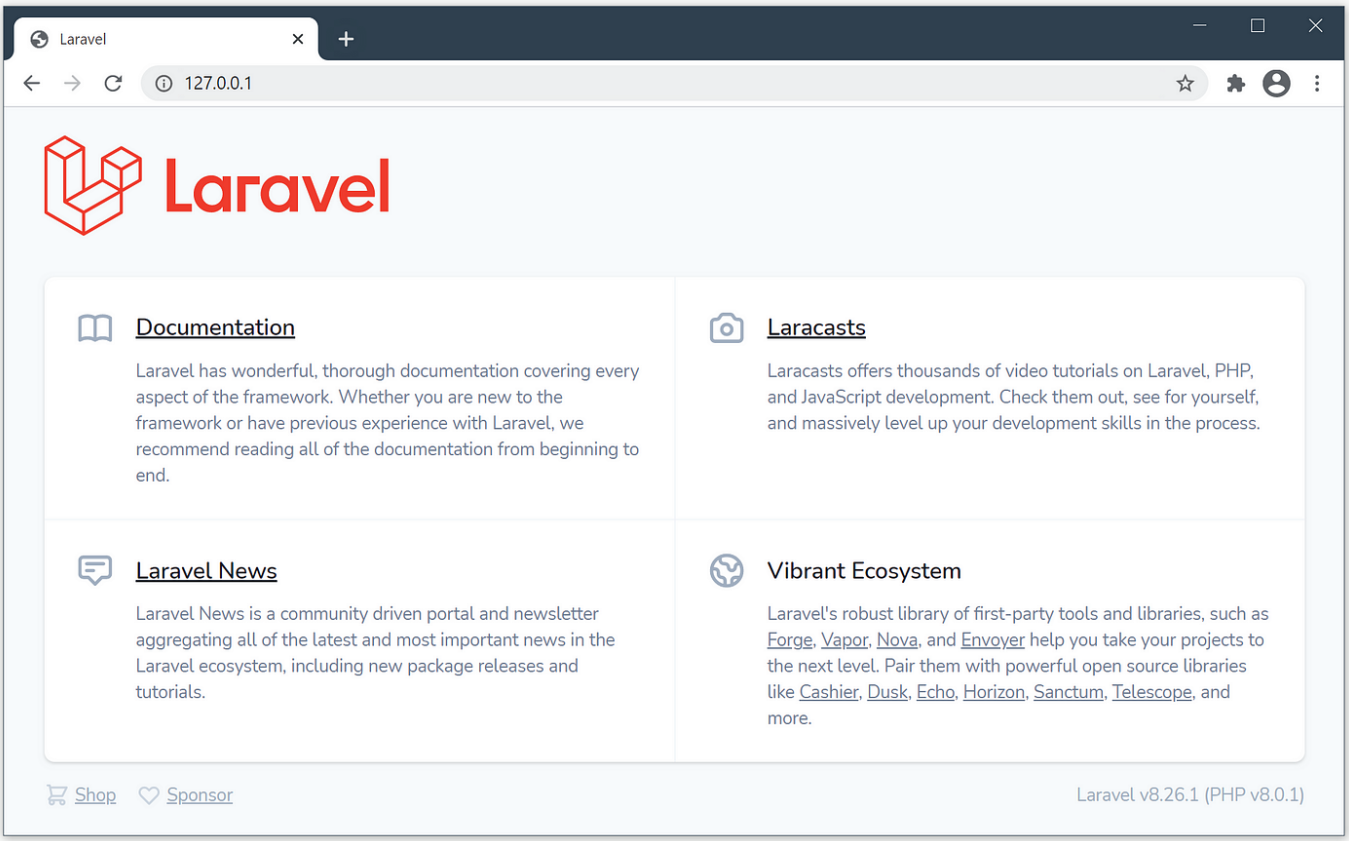
2. Run the `sail:install` Artisan command, which will publish the `docker-compose.yml` file to the root of your application:

```
php artisan sail:install
```

3. Run `sail up` which will start the container:

```
./vendor/bin/sail up -d
```

And that's it, now, if you visit `http://localhost` Laravel should present you with its default welcome view.



Project structure

When you are working with laravel framework it is important to have some knowledge regarding directory structure of the framework. You should know where to find files you are looking for and where to place classes or third party libraries.

When you download laravel framework for the first time you will see following directory structure:

```
laravel-9/
├── app
├── artisan
├── bootstrap
├── composer.json
├── composer.lock
├── config
├── database
├── docker-compose.yml
├── lang
├── package.json
├── phpunit.xml
├── public
├── README.md
├── resources
├── routes
├── storage
├── tests
├── vendor
└── webpack.mix.js
```

App directory

App directory contains additional directories:

- Console: where you will write artisan command classes
  - Kernel.php: this is where you schedule background jobs/commands
- Exceptions: this is where you can write customized exception logic
- Http: this directory contains
  - Controllers: laravel controller file lives
  - Middlewares: this is where you will create route middlewares
  - Requests: this is where you can write your request validation classes
- Models: this is where you will write database models
- Providers: providers helps you boot your custom modules or packages
  - RouteServiceProvider.php: loads laravel routes with defined prefix

Bootstrap directory

The bootstrap directory contains following files:

```
bootstrap/
├── app.php
└── cache
    ├── packages.php
    └── services.php
```

The bootstrap directory contains app.php file which bootstraps the laravel framework. This directory also contains cache directory where framework generated cached files are stored.

When you are running your application in production mode you would have to run some optimization commads like:

```
php artisan config:cache
php artisan event:cache
php artisan route:cache
```

All cached file generated because of above commands will be stored under bootstrap/cache folder.

Config directory

Laravel uses different types of configuration during runtime. All these configurations are fetched from config folders. It is a good practise to check the files withing this folder and familiarize yourself with different types of configurations as shown below:

```
config/
├── app.php
├── auth.php
├── broadcasting.php
├── cache.php
├── cors.php
├── database.php
├── filesystems.php
├── hashing.php
├── logging.php
├── mail.php
├── queue.php
├── sanctum.php
├── services.php
├── session.php
└── view.php
```

Database directory

The database directory contains different directories like:

- factories: where you can write database model factories
- migrations: actual database migration classes
- seeders: database seeder classes to load initial data

Let's look at the default directory structure:

```
database/
├── factories
│   └── UserFactory.php
├── migrations
│   ├── 2014_10_12_000000_create_users_table.php
│   ├── 2014_10_12_100000_create_password_resets_table.php
│   ├── 2019_08_19_000000_create_failed_jobs_table.php
│   └── 2019_12_14_000001_create_personal_access_tokens_table.php
└── seeders
    └── DatabaseSeeder.php
```

Public directory

The public directory contains `index.php` file which is main entry point for laravel application. This file loads all composer dependencies and then boots the laravel framework.

This is also your web root for your deployed production application. You can store images, css, js or font files in this directory. You can create more files here which are public facing.

One thing to make sure that you do not put any sensitive files in this directory because these files are accesible from anyone on internet.

```
public/
├── favicon.ico
├── index.php
└── robots.txt
```

Resources directory

The resources directory contains your uncompiled css or js files along with all laravel view files. You can store your application view files in this directory.

```
resources/
├── css
│   └── app.css
├── js
│   ├── app.js
│   └── bootstrap.js
└── views
    ├── errors
    └── homepage.blade.php
```

Routes directory

This is a very important directory in laravel framework. The route directory contains serveral files:

- `web.php`

    - If your application is not REST API based most likely you will use this file to define your application routes here.
    - All routes defined in this file goes through web middleware by default which means it provides:
        - session state
        - csrf protection
        - cookie encryption
- `api.php`
    - this file contains all middleware that goes through the api middleware by default if you are designing REST apis.
    - Routes defined in this file intended to be stateless therefore requests entering in the application should be authenticated based on token and do not have access to sessions.
- `console.php`
    - Generally, laravel allows you to create a console command class and then register this new command via `Kernel.php` file.
    - However, you can create a closure function to define your new artisan command rather then writing an entire new class in route/console.php file. It will be loaded by default by laravel.
- `channels.php`
    - The `channels.php` file is where you may register all of the event broadcasting channels supported by your application.

Storage directory

The storage directory contains:

- log files
- compiled blade templates
- file based sessions
- cached files

You can also create some other directories in this storage folder which you want to hide from public access on internet. You should create a symbolic link at `public/storage` which points to this directory. You may create the link using this command:

```
php artisan storage:link
```

Vendor directory

This directory contains all of your third-party libraries installed via composer command.

Other files

Aside from above directories you will also see some files in laravel application root folder. Following files can be found in your laravel root folder.

- composer.json: defines your composer dependencies
- composer.lock: defines locked versions for your third-party libraries defined in composer.json file
- artisan: bash file that boots your console application and runs artisan commands
- docker-compose.yml: docker dependencies for your local laravel project
- package.json: defines your javascript dependencies
- phpunit.xml: defines your unit test related phpunit configurations
- webpack.mix.js: defines your commands to mix or minify your js/css files