# Lab: Docker Compose

Lab for the ["Containers and Clouds"](https://softuni.bg/trainings/4359/containers-and-cloud-january-2024) course @ SoftUni

In the exercises below we will learn how to **work with Docker images**, **Dockerfiles**, **volumes**, **networks**, **Docker Compose**, etc. to run **multi-container apps in Docker**.

## MyWebsite App: Building a Custom Image

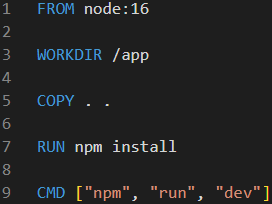
### Step 1: Create a Dockerfile

Our first task is to **create a Dockerfile for a Vue.js app**, which will allow us to **run it in a Docker container**.

First, we have to go to the **root** folder of the **Vue.js** app that we created and ran in our previous session.

Our next step is **creating a Dockerfile** in this **directory**. The **Dockerfile** **contains instructions** on how an **image for the app should be created**. As we know, **Dockerfiles** are **just text files**, so we can create our own and open it with a text editor of our choice. Note that the name of the file should be "**Dockerfile**" without any extensions.

The content should be as shown below:



Each Dokerfile starts with "**FROM**", so we start creating an image, based on the existing image **node:16.**

After that, we will **set** the **"app" folder** as the **current working directory** and we'll **copy** all of the **project files and folders to it.** This will add a layer.

Then, we run the **npm install** command in order to **install the necessary dependencies**, so that our app can run. This will form another layer, too.

Finally, we execute the **npm run dev** commands, in order to run the scripts that defined inside our app's **package.json** file.

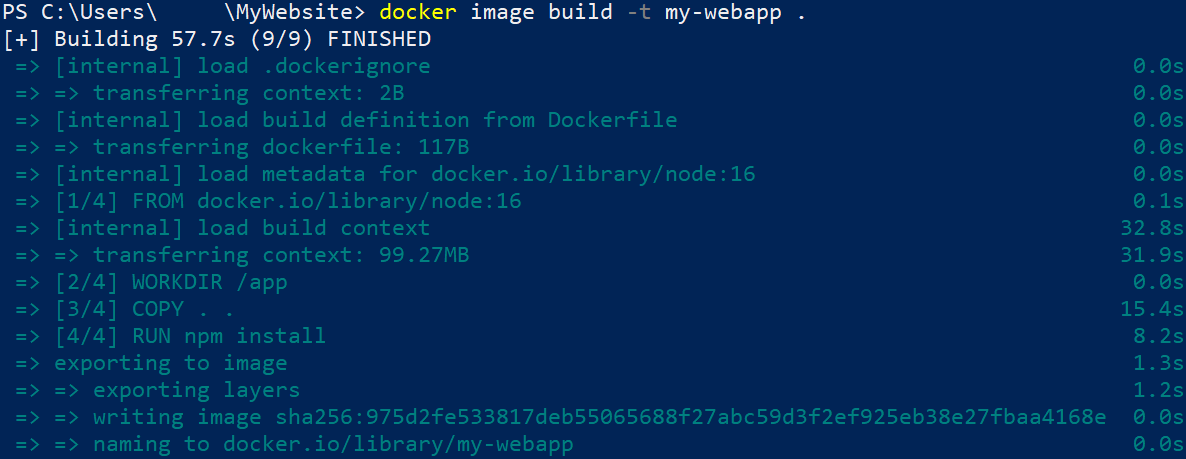
### Step 2: Build and Publish the Image to Docker Hub

#### Build the Image

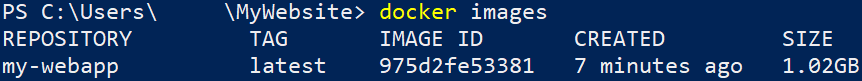
We can now **build a custom image** with this **Dockerfile**. Open a CLI, for example **Powershell**, and fulfill the **following steps** to do it:

* Navigate to the **MyWebsite** **directory**
* Use the **docker image** **build** command to **build the image**
* Set the **local directory** as the **working directory**
* With the **-f** option, set the **path to the Dockerfile**
* With the **-t** option, set the **name of the image** in format **{your\_docker\_hub\_username}/{app\_name}**, as we will later **add our image to Docker** **Hub**

The **whole command** should look like this:



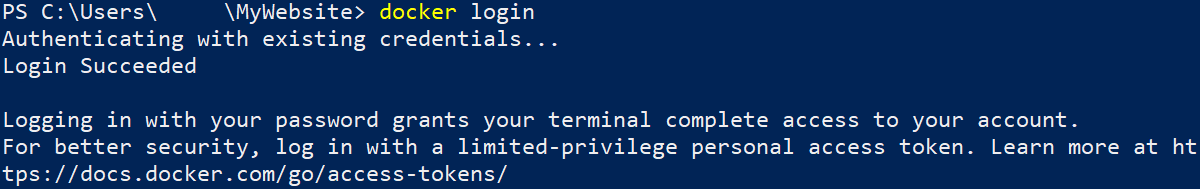
Note that we can examine how the **instructions from the Dockerfile** are executed to **build the image**.   
We can check the **ready image** with the **docker** **images** command.



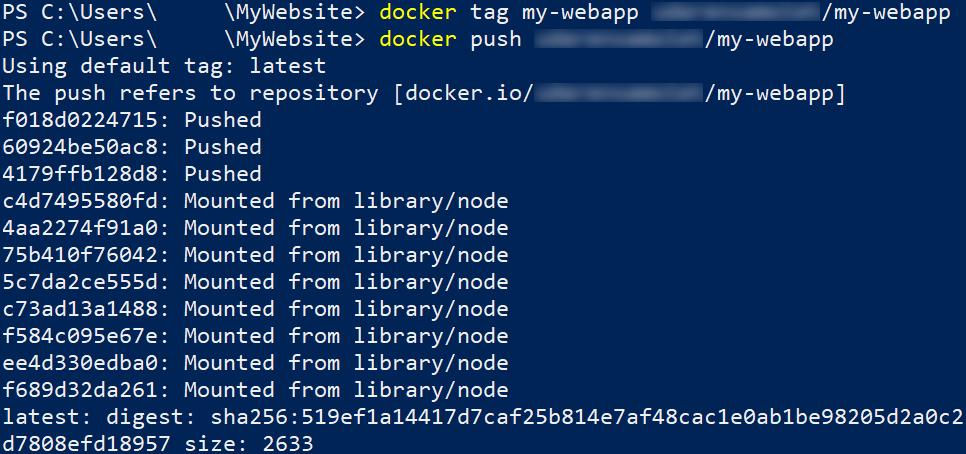
#### Publish the image

Now let's see how to **push our custom image** **to** **Docker** **Hub**. Note that this is **not needed** for running a container with that image – you can have the **image only locally** and still use it. However, it is good to know **how to push images**.

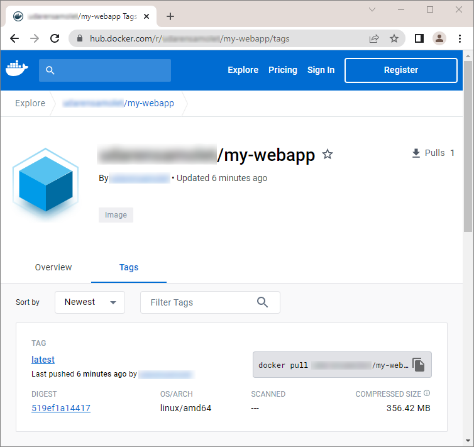
To **push our image to Docker** **Hub**, we should first **log-in to Docker Hub** with the command below. If this is the **first time** you log in, you should **enter your credentials**. Make sure that **login is successful**:



Now you should only **push the image**:



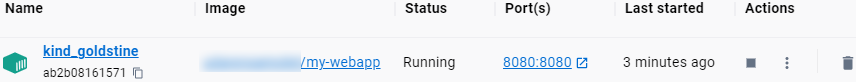
And it now is **available at Docker** **Hub** as a **public image**:



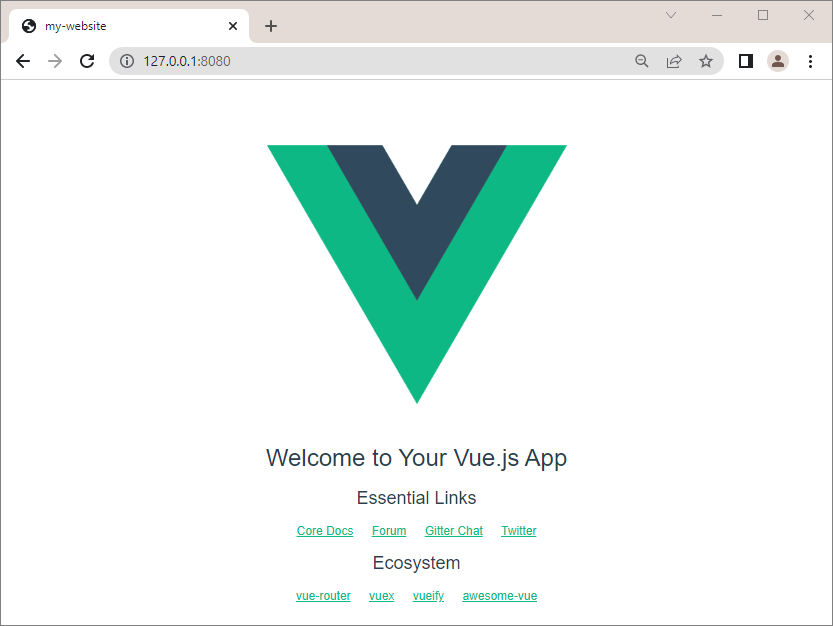
#### Run the Image as a Container

Finally, let's run the newly-created **image** as a **container** on the **right port**, using the command below:





Open your web browser and go to **127.0.0.1:8080**. You should be able to see the **running Vue.js app**:



## WordPress App with MySQL Database: Connecting Containers in a Network

In this exercise, our task is to **set up and run a WordPress container** in **Docker** with a **MySQL** **database** by connecting them in a network.

### Step 1: Create a Network

First, we have to create a network. Open a CLI and first create a new folder, which will contain the files for our app. Then, create a new network with the **docker network create {network\_name}** command.

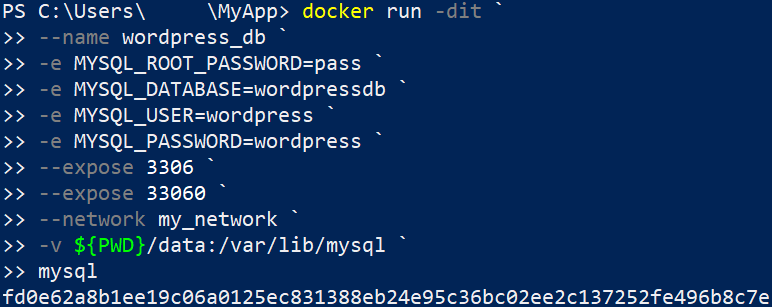


### Step 2: Add MySQL Container to Network

Our next step is **adding** the **MySQL** container to the network that we just created.

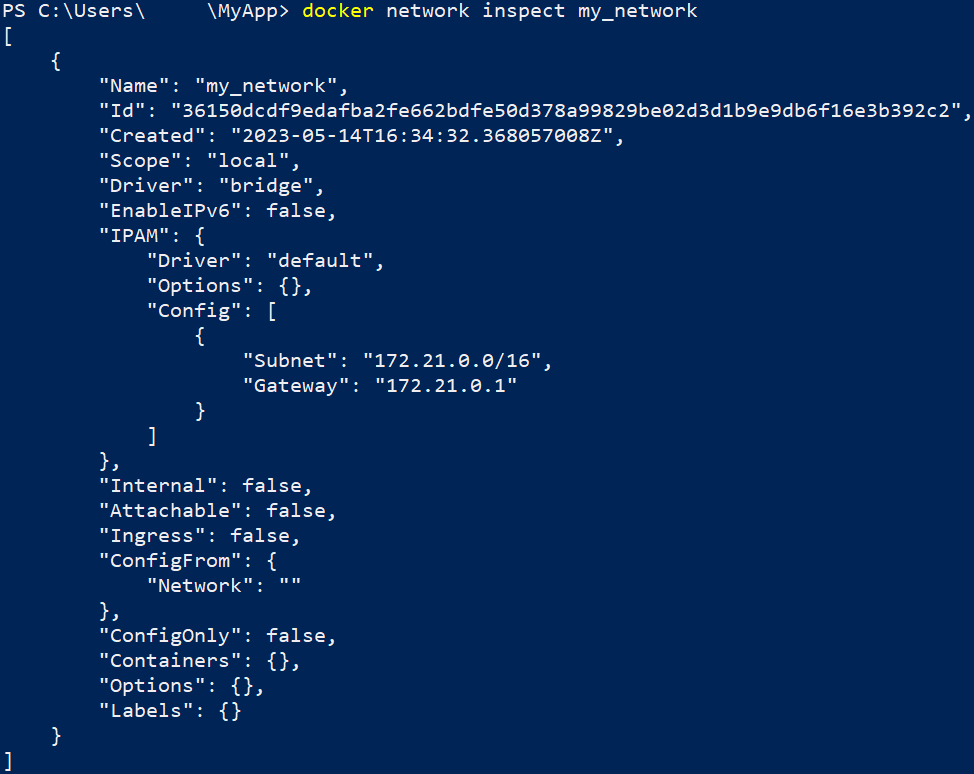
The **commands from the resources** are the following:

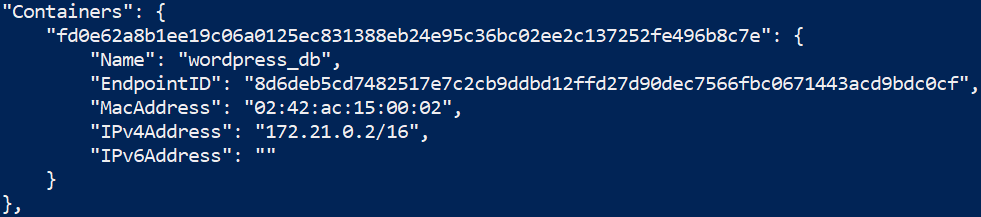
* **docker run -dit** → runs the image detached and in interactive mode;
* **--name wordpress\_db** → names the container **wordpress\_db**;
* **-e MYSQL\_ROOT\_PASSWORD=pass** → sets the password for the root MySQL user;
* **-e MYSQL\_DATABASE=wordpressdb** → sets the name of the MySQL database that we'll use for the WordPress installation;
* **-e MYSQL\_USER=wordpress** → sets the MySQL user that we'll use for the WordPress installation;
* **-e MYSQL\_PASSWORD=wordpress** → sets the password for that user;
* **--expose 3306** → sets the port of the container;
* **--expose 33060** → sets the SSL port of the container;
* **--network my\_network** → sets the network that we want to attach our container to;
* **-v ${PWD}/data:/var/lib/mysql** → maps the directory on our local machine to the directory of the container, so that we can store data outside of it;
* **mysql** → the name of the image.



### Step 3: Inspect Network

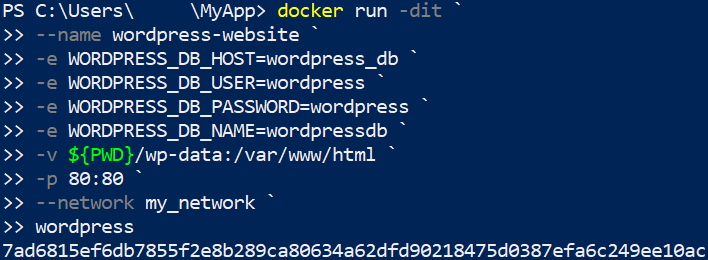
Now let's inspect our network in order to check if our **wordpress\_db** container is attached to it:





### Step 4: Add WordPress Container to Network

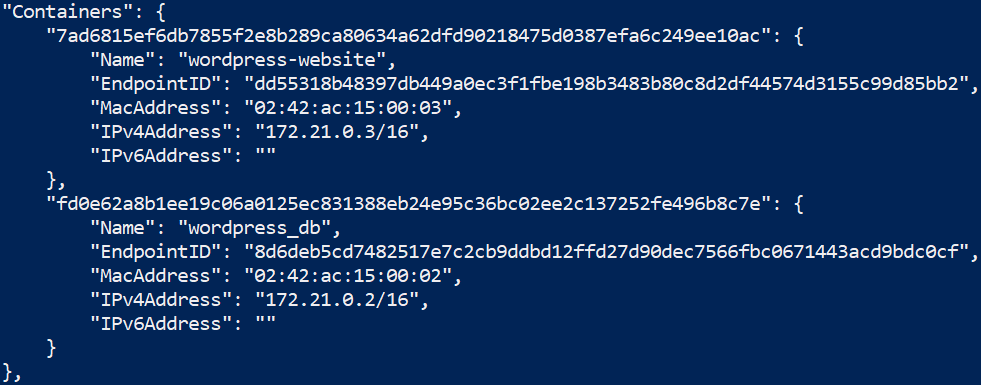
Our next step is adding the WordPress to our network. You can do it with the following command:



* **-e WORDPRESS\_DB\_HOST=wordpress\_db** → sets the WordPress database host, which matches the name of our MySQL container that we set up in Step 2;
* **-e WORDPRESS\_DB\_USER=wordpress** → sets the WordPress user that we previously set up;
* **-e WORDPRESS\_DB\_PASSWORD=wordpress** → sets the password for the user;
* **-e WORDPRESS\_DB\_NAME=wordpressdb** → sets the name of the WordPress database, that we created in Step 2.

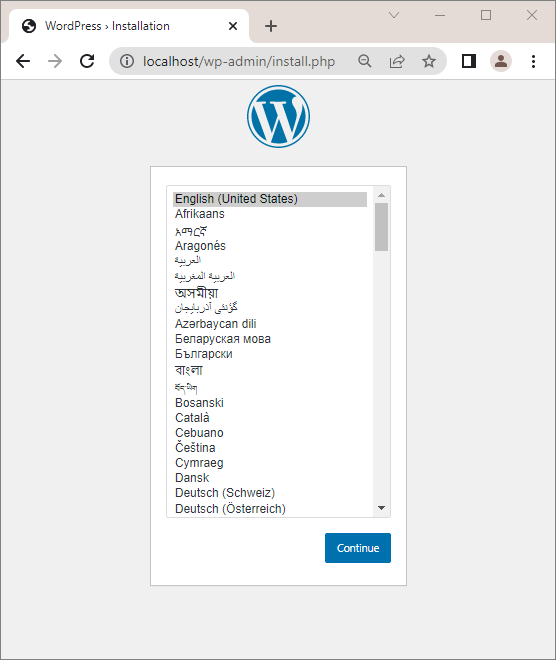
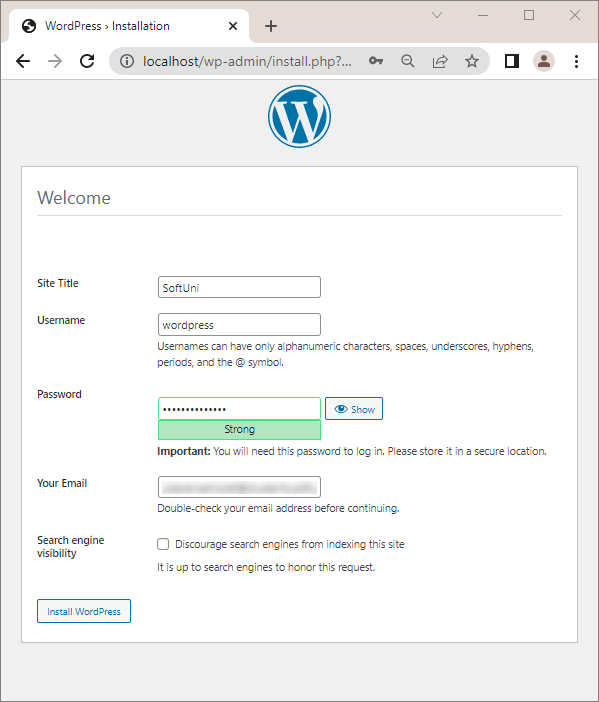
### Step 5: Inspect Network

Now, if we execute the command for inspecting our network, we should see that the two containers are attached to it:

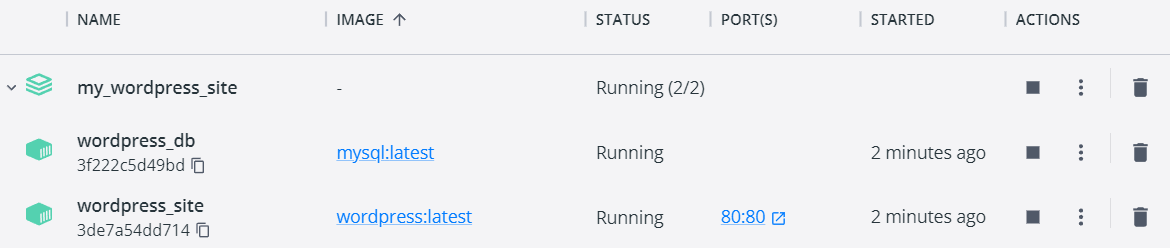


### Step 6: Run the App

You can **access the WordPress site** on **http://localhost:80** and you will see the **WordPress setup page:**

****

When you check **Docker Desktop**, you should be able to see the **two containers** combined in a **single network**:



## WordPress App with MySQL Database: Docker Compose YAML file

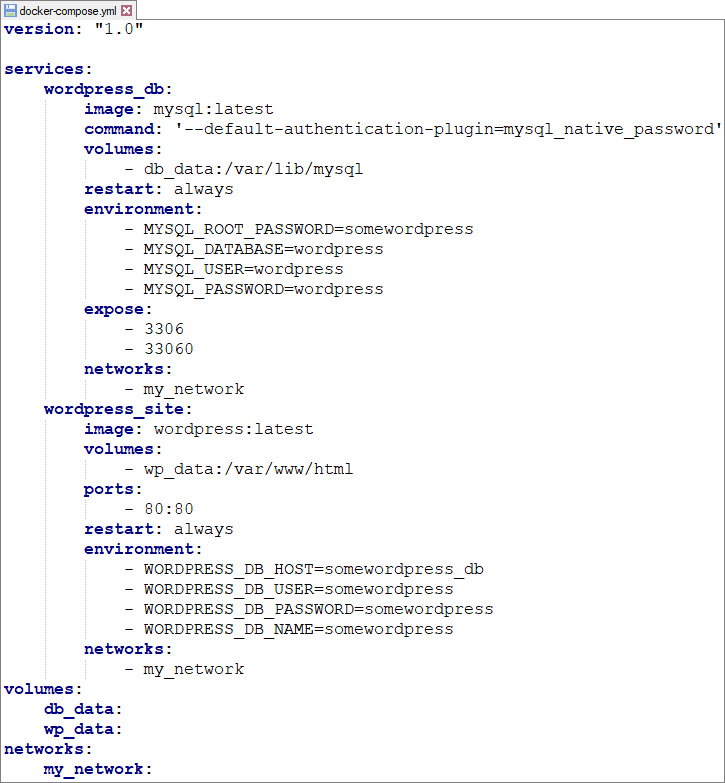
Our next task is, using **Docker Composer**, to **create a docker-compose.yaml file** with instructions for **creating the MySQL and WordPress containers** from the previous task**, together** in a single network, called **my\_network**.

### Step 1: Create a Network

Using the **docker network create**, create a new network, called **my\_network.**

### Step 2: Create the docker-compose.yml file

First, we have to create a **docker-compose.yml** file in the folder of our app. The docker file should look like this:

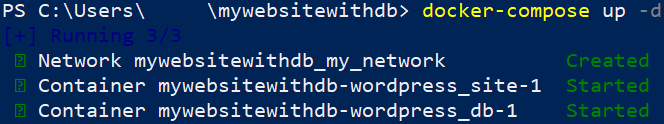


### Step 3: Build and Run the Multi-Container App

Next step is building and running our multi-container app. First, build all of the images with the **docker-compose** **build** command:

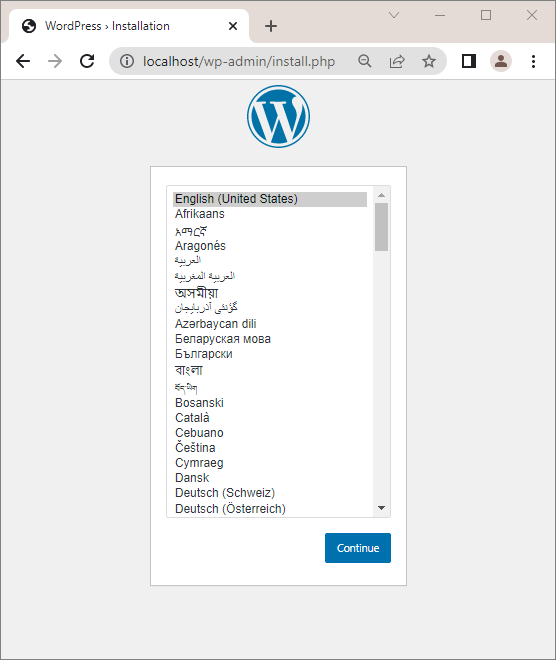


Then, run the containers with the **docker-compose** **up** or **docker-compose up -d command**.



### Step 4: Run the App

You can **access the WordPress site** on **http://localhost:80** and you will see the **WordPress setup page.**



You should be able to **configure and create your website**, following the guide:

