{EPITECH}

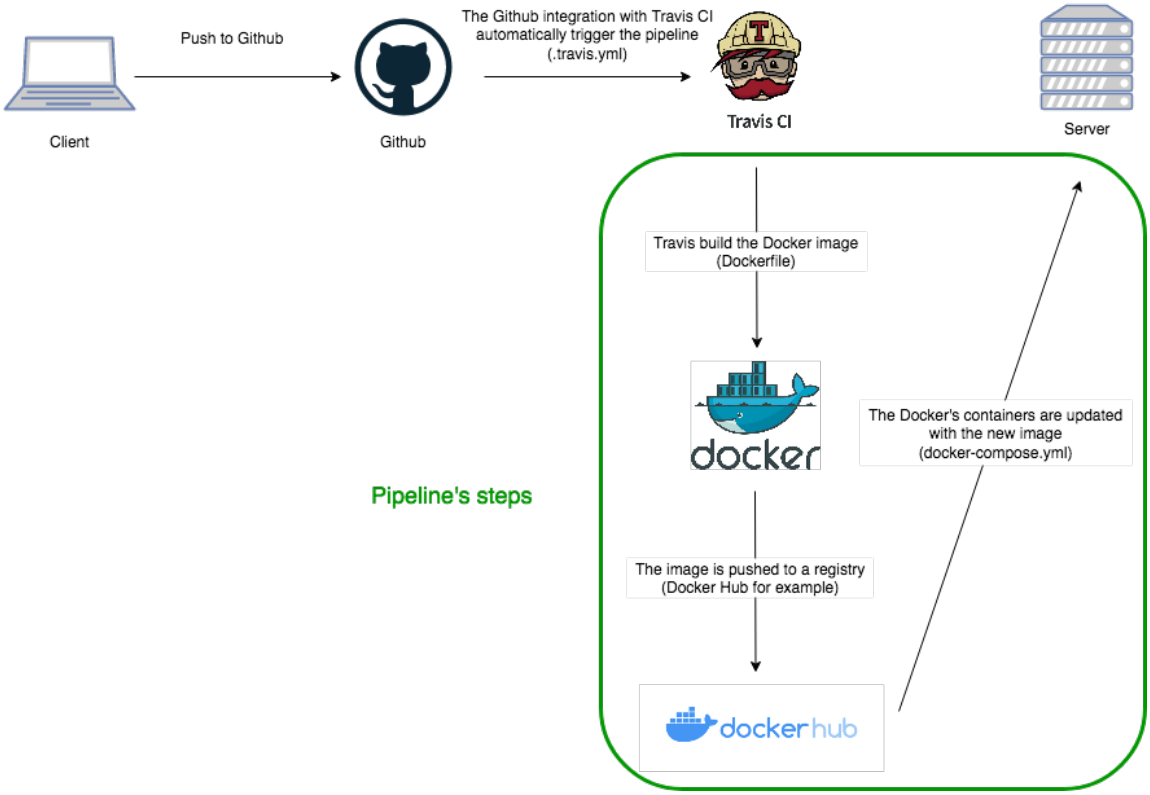
TIME MANAGER

BOOTSTRAP - DEVOPS

TIME MANAGER

During this bootstrap you will approach, in a simplified way, two principles of the devops : the continuous integration and the automated deployment.

CI will happen via a pipeline launched automatically upon each repository change.
This pipeline will build an image of your application, send your image to a registry and deploy the different Docker containers with the new image.

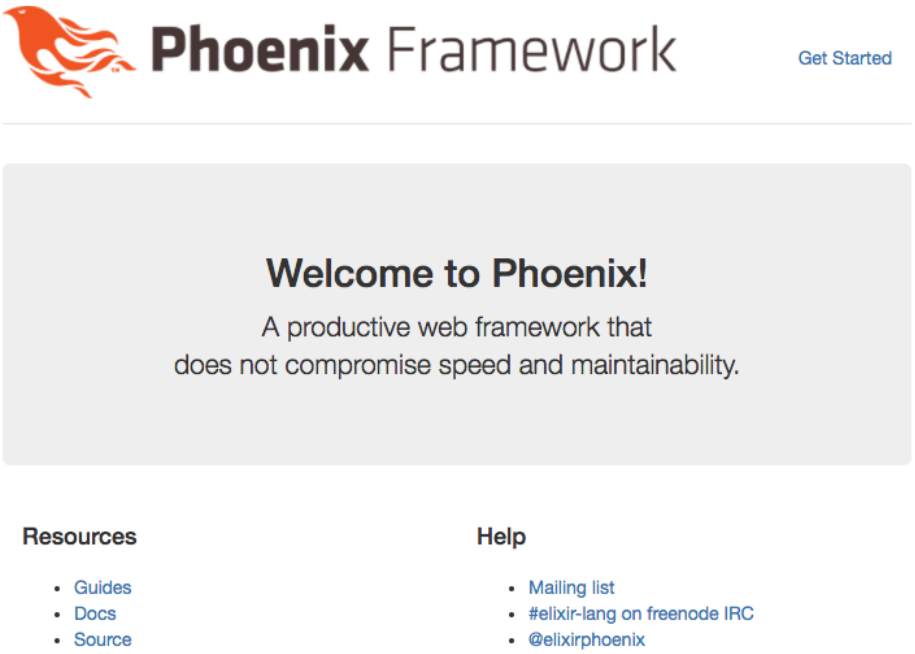


EPITECH

Hello world from Phoenix

Start by setting up a “hello world!” by following the Official Documentation Guide.

Check your localhost in your browser; it should look like:



Environment file

In a micro services architecture, each component of your project is separated.

This means that they can also change/evolve independently (to a certain extent) and in particular the database.

For instance, login credentials can change, or tables can be migrated to another database on a better provisioned server for example.

Starting from this assumption, it is wise to have a defined place that would allow us to declare variables that can change without having to directly modify part of the application.

Create an environment file, named `.env` at the root of your repository, that includes the following environment variables related to the database:

- ✓ `PGUSER`: the user name to connect to the database
- ✓ `PGPASSWORD`: the password to connect to the database
- ✓ `PGDATABASE`: the name of the database
- ✓ `PGPORT`: the port of the database
- ✓ `PGHOST`: the name of the host of your database

In order for these environment variables to be taken into account, you must also modify your Phoenix project for it to use. The configuration of the database is in the `config/dev.exs` file.

Dockerfile and entrypoint

Building a Docker image is done by creating a `Dockerfile`.

The entrypoint is a script usually called `entrypoint.sh` which is used to execute actions on initialization of the image in a container.

In short, the Dockerfile allows you to perform actions to build an image ; the entrypoint allows you to perform actions on initialization of the image in a container.

Here is the tasks list you need to implement in your Dockerfile:

1. Add dependencies for PostgreSQL and Node.
2. Copy your application inside the image.
3. Install `mix` (which is present in the image of Elixir).
4. Install the Phoenix dependencies.
5. Install dependencies of JS (node) and launch Webpack.
6. Execute the entrypoint.

For a given public Docker image, there are usually several “tags”.
These tags are a specific version of an image, for example, for the official image of Ubuntu the tags correspond to different versions of the distribution.

The steps of the entrypoint are quite simple but may require some research.
Test regularly the construction of your image to check consistency.

Here are the steps to follow in your script :

1. Check the presence of a `.env` and stop the container `phoenix` (see “Step 4 - Docker-compose”) in his absence.
2. Wait until the container of the database (cf “Step 4 - Docker-compose”) is usable.
3. Check the presence of the database entered in the `.env` on the container and create it if necessary.
4. Launch the application.

A Docker container is composed of an image, some options, and possibly a volume to ensure the persistence of the data.
The documentation is your friend and can greatly assist you. Also think about best practices.

Be sure to install all Phoenix dependencies.
Look carefully for error messages if there are any.

Docker-compose

Docker-compose is the tool that will allow you to set up your micro services architecture.

By filling in the dedicated configuration file (`docker-compose.yml`), you can create a set of containers that will allow you to have a complete application.

Your `docker-compose.yml` must have two services:

- ✓ one for your application called `phoenix`, which depends (the word is not chosen at random) on the other service
- ✓ one for the PostgreSQL database called `db`, that has a volume for persistence of data.

Do not forget to map ports for different services

We'll never say it enough but the documentation is your friend.

A volume is not a bind mount!

Choosing a web host

To automatically deploy your application on a server, you must choose a host that can provide you a server. Several solutions are possible.

The simplest is to rent a dedicated server at OVH. Otherwise you could also choose AWS (Amazon Web Services) or GCP (Google Cloud Platform).

No matter which host you choose, you must select an offer giving you access to a dedicated server.
At OVH, this is called a VPS, at AWS an EC2 and at GCP a VM Compute Engine instance.

You can access a free trial offer for AWS or GCP on their official website.

Do not forget to have port 4000 open or redirect the port of your application.

Travis CI

Several solutions exist to create pipelines.However here we will focus on Travis CI pipelines.

It is very easy to handle and easily integrates with Github. However, you have the choice to choose your registry where your image Docker built in your pipeline will be hosted.

The pipeline will be fairly simple and will perform the following steps:

- ✓ Connect to your host (to adapt according to the choice of your host)
- ✓ Use the service (the word is not chosen at random) `docker`
- ✓ Have a first `stage` called "**Build & Push Image**", that will:
  - connect to your registry where your image is/will be hosted
  - build this image
  - send the new image built on your registry
- ✓ Have a second `stage` called **Deploy** that will allow to:
  - copy your file to docker-compose.yml on your server
  - update the Docker containers that run on your server

If you are looking for a registry, Docker Hub allows you to host a private project for free.

Final Test

If everything went well, commit your Github repository to automatically launch your Travis CI pipeline.

The latter should:

1. build your image ;
2. upload it to your registry ;
3. send your docker-compose.yml (which will build your micro-services) ;
4. start the creation of containers on your server.

Congratulations! You have completed the bootstrap!

If something went wrong, sorry for you. Restart the boostrap from the beginning and be more cautious about the testing!

v2