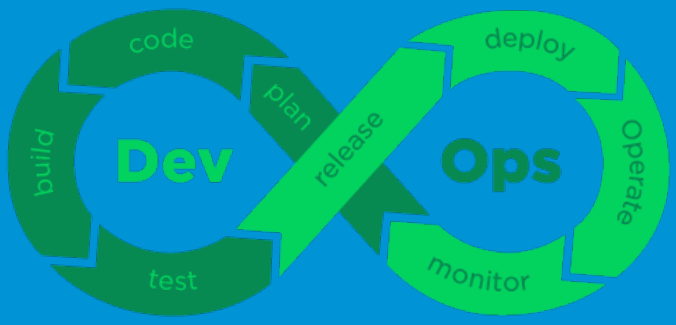
{EPITECH}

# POPEYE

EAT YOUR SPINACHES!



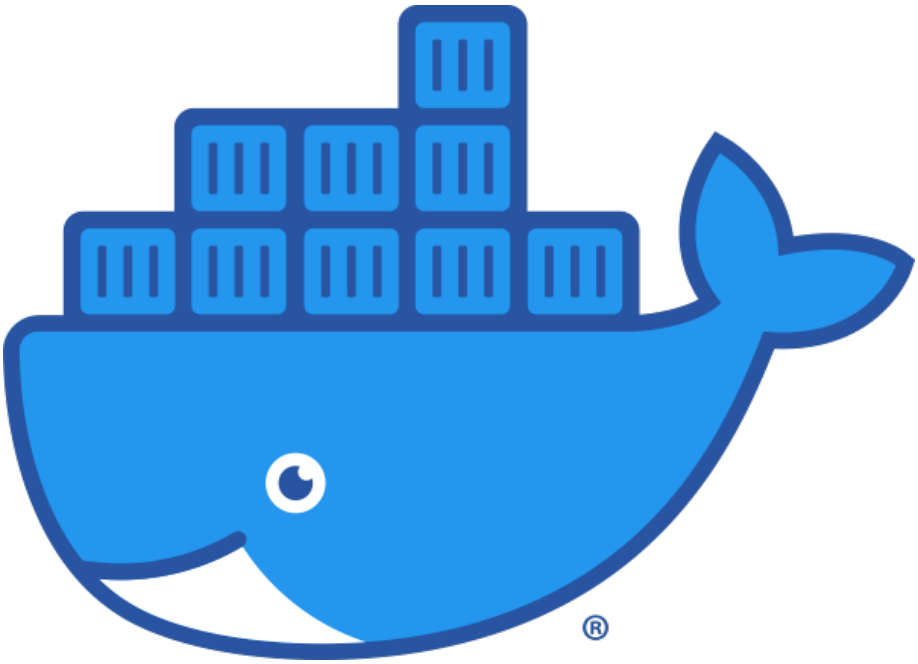
# POPEYE

**Running everything everywhere** is an important part of software engineering methods.

Containerization is a great way to set up a runtime environment for any application, down to the operating system it needs. But above all, it is not dependent on the host OS on which the containerization service is running.

> This is like spinaches for programmers: at first, it feels strange and does not taste good, but in the end, you become an incredibly strong sailor that loves those tools and is a master at using them.
>
> **— The module's designer —**

During this bootstrap, you are going to become familiar with one of the most popular containerization technologies: *Docker*, and its associated tool *Compose*.



# Hello Docker!

## A little quiz to start with

Before starting to use Docker, it is important to clarify some points. Try to answer those questions with your friends:

- ✓ What is Docker?
- ✓ What is the difference between Virtual Machines and Docker containers?
- ✓ What is the difference between a Docker container and a Docker image?
- ✓ In what way will spinach helps you to master DevOps?

If you answered those questions right, you got everything to start your journey.

## Setup time

Before using something, you need to install it (natuurlijk !). Install *Docker* and *Compose*.

It is strongly discouraged to install the Docker packages that might exist in the default package repositories of your operating system, as they are often outdated.

It is strongly advised to follow the instructions from the official website.

For this bootstrap, you will also need a PostgreSQL client. There exists a command-line tool (`postgresql`) as well as a GUI (`pgadmin4`).

## Hands on the rudder

Docker Inc. provides official images of popular softwares and applications on Docker Hub. When importing (or *pulling*) images in Docker, the Docker Hub is the default location used to look for desired images.

For each situation below, only one Docker command needs to be executed. Take the time to find and understand them, as they will be very useful:

- ✓ Find the official PostgreSQL image and import it on your machine.

- ✓ Show how many images you have on your machine.
- ✓ Run a PostgreSQL container.
- ✓ Show containers running on your machine. Do you see the container ID of PostgreSQL?
- ✓ Is your previous PostgreSQL container still running? Fine, stop it.
- ✓ Even if your container is stopped, it still exists on your system. Remove it.
- ✓ Bad news! The developer told you that their application is only compatible with PostgreSQL 9.4. Anyway, run a container with this specific version.

When running a container with an image that is not present on your machine, Docker will search for it and pull it automatically.

- ✓ You want to list tables, but you cannot connect to this PostgreSQL instance from your host. It seems like the needed port is not exposed on your machine. How to fix that?

```
psql postgres://username:password@address:port/dbname
```

- ✓ In the official PostgreSQL image, the default username / password / database are: `postgres`/`postgres`/`postgres`. This is uncreative, but above all unsecure! Find how to change PostgreSQL's password (when executing `docker run`).
- ✓ At the moment, data is not persistant and vanish from hard drive when its associated container is removed. Be kind to your data and offer them a place to stay on your hard drive.
- ✓ Can you find out which day it is in the container? Execute the command `date` into a **running** PostgreSQL container.
- ✓ Because you absolutely love reading lines of information written on your screen, find how to display the logs of your container. (Bonus point if you find how to display them in real time.)

```
docker --help
```

Once you succeeded at finding the 12 commands above, congratulations! You earned a can of spinaches and are allowed to proceed to the next step.

## Time to craft

In the previous section, you used an official (prebuilt) Docker image.
It is now time for you to do a real sailor job: creating your own image!

A very simple Node.js application should be provided. Your task is now to make an image out of it.

To do so, you are going to use what is called a *Dockerfile*. This is a recipe, consisting of one or several instructions, that Docker follows to create a custom image. Such operations can include copying files from the host to the container, running commands, etc.

The construction of an image with a Dockerfile is simply the launch of a base container, the execution of operations in it, and the save of the state of the container at the end, which then allows to launch it multiple times without needing to redo all the previous operations to set up the environment.

Write a file named `Dockerfile` at the root of your repository which respects the following specifications:

- ✓ Based on the official `debian:buster-slim` image.
- ✓ Installs Node.js and its associated package management tool with `apt-get`.
- ✓ Copies the source code of the application into the image.
- ✓ Installs the application's dependencies with Node.js' package management tool.
- ✓ Exposes port 3000.
- ✓ Sets the command (not the entrypoint, beware) to start the application so that the application is run when running a container based on the image.

The above instructions can be executed with one Dockerfile instruction each.

When you have finished, you can build your new image with an easy command: `docker build`!

You should now be able to run a container with your freshly built image, and have the application launched automatically (do not not forget to set up a port, so you can access it from your host).

For debugging, you can run your container with Bash:`docker run -it <my-image> /bin/bash` or execute a command inside a running container with:`docker exec -it <my-image> /bin/bash`

## Time to craft the crafter

As you might have seen, proper `docker run` commands are often quite long.

Docker's associated tool, *ompose*, allows to describe a set of containers with the different properties you want to give them in just a single YAML file. It is very useful as it simplifies a lot the management of the lifecycle of Docker containers' architectures.

Write a file named `compose.yml` at the root of your repository which respects the following specifications:

- ✓ Has a `hello-world` service which:
  - builds its image by using the previous step's Dockerfile;
  - forwards container's port 3000 to host's port 8080.
- ✓ Has a `db` service which:
  - uses the official PostgreSQL image version 15.2 Alpine;
  - forwards container's port 5432 to the same host's port;
  - sets the superuser's password to `popeye` using an environment variable or an environment variables file.

If using an environment variables file, you must explicitly specify it in the `compose.yml` file.

When you have finished, take a deep breath, and prepare to launch a very simple command that will ignite for good in yourself the flame of DevOps passion: `docker compose up`.

Congratulations! You now have a full infrastructure of a Node.js app (with its dependencies) and a database up and running just with one command, which can be run and work on any OS. Jij bent heel getalenteerd!

You are now ready to sail and to tackle the project! Good luck!
(And do not forget to eat your spinaches.)

v 1.3.1