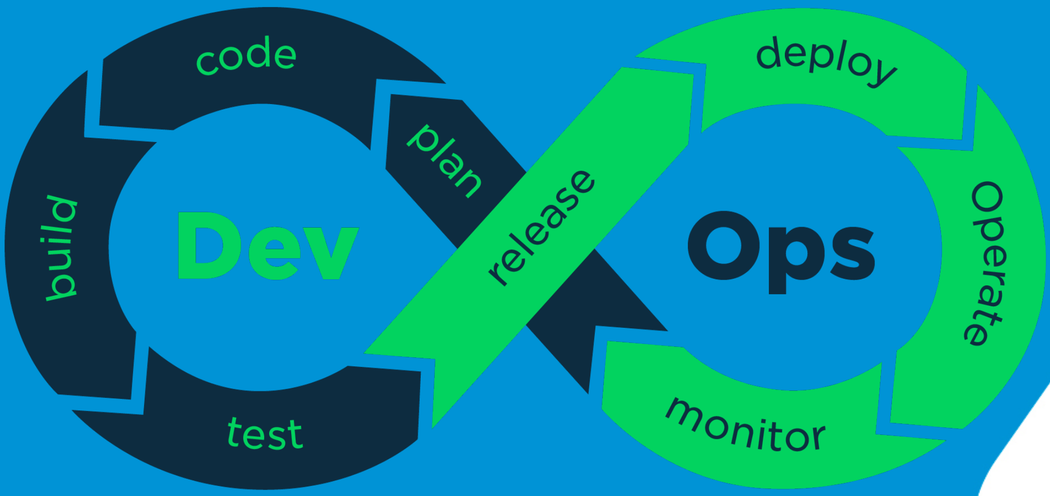
POPEYE - BOOTSTRAP

EAT SOME SPINACHES, BECOME A DOCKER SAILOR



POPEYE - BOOTSTRAP

Containerization is a great way to set up a runtime evironment 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 an important part of today's software engineering methods: running everything everywhere.

> 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, who thought it would be more classy to put that in a quote box —**

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

Step 0 – Hello Docker!

A little quiz to start with

Before starting to use Docker, it is important to gain some knowledge and to either acquire or clarify important 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?
- ✓ How much are 2 Docker containers separated when running on the same machine?
- ✓ In what way will spinach helps you to master DevOps?

If you answered those questions right, you then have all you need to start your journey.

Setup time

To use Docker, you need to have it installed.

First, check if you have Docker installed by running the following command:

```
~/B-DOP-200> docker --version
Docker version 27.5.0, build a187fa5
```

If Docker is installed, you should have a similar output to the one above, and you can directly proceed to the *Hands on the rudder* step.

If Docker is not installed, well, install it. ;)

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.
Follow the instructions from the official website to install the latest version of Docker for your OS.

Hands on the rudder

You are now going to be presented a few situations which each needs only **one Docker command** to be executed.
Take the time to **find and understand** them, as they will be very useful to you.

If you are using Docker Desktop, **refrain from using the GUI**, as you will not correctly learn the Docker commands that will be useful to you if you just click on buttons.

`docker --help`, and more importantly the Docker documentation, are your best friends.

Docker Inc. provides official images of popular softwares and applications on Docker Hub.
There are also other images, distributed by the community, on the same platform.

When importing (or ***pulling***) images in Docker, the Docker Hub is the default location used to look for desired images.

1. Find the official Nginx image and import (***pull***) it on your machine.

If you have an error message when trying to pull the image
(such as `You have reached your pull rate limit`),
you shoud create a Docker Hub account and log in with the `docker login` command.

2. ***List*** the ***images*** you have on your machine.
3. ***Run*** an Nginx container.
4. In another terminal window, while your container is running, ***list*** the ***containers*** running on your machine. Do you see the container ID of your Nginx container?
5. Now, ***stop*** your container.
6. Even if your container is stopped, it still exists on your system. ***Remove*** it.

There exists a nice option to automatically remove a container when it is stopped.

7. Bad news! The developper told you that their application is only compatible with Nginx 1.18. 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.

8. Nginx is a web server, which you can access from your usual browser. However, it is not possible yet to access it from your host, as its port is not ***published***. How to fix that?

Nginx listens on its port 80.
After publishing the port, you should be able to access the Nginx welcome page at `http://localhost:xxx`, where `xxx` is the ***host*** port you chose to publish the container's port 80 to.

Do not confuse the ***container*** port and the ***host*** port.
The first one is the port on which the application is listening inside the container, and the second one is the port on which the application is accessible from the host (and the outside world).

9. Can you find out which day it is in the container? ***Execute*** the command `date` into a **running** Nginx container.
10. 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.)

If you succeeded at finding the 10 commands above, congratulations! You earned a can of spinaches and a permit to proceed to the next step.

Step 1 – 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 is given to you on the intranet, and 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.

> To be clearer, 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. (It is similar to Virtual Machines' snapshots.)

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

- ✓ Based on the official `debian:bookworm-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 publish 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`.

Step 2 – Time to craft the crafter

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

Docker's associated tool, *Docker Compose*, 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 `docker-compose.yml` at the root of your repository which respects the following specifications:

- ✓ Has a `jigglypuff-server` service which:
  - builds its image by using the previous step's Dockerfile;
  - forwards container's port 3000 to host's port 8080;
  - make Jigglypuff shiny by setting the `JIGGLYPUFF` environment variable to `shiny`, using an environment variable or an environment file.
- ✓ Has a `nginx` service which:
  - uses the official Nginx image version 1.24 Alpine;
  - forwards container's port 80 to host's port 5000.

> If you are using an environment file, you must explicitly specify it in the `docker-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`.
You should now be able to access the `jigglypuff-server` service at `http://localhost:8080`.

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.

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

v 2.1

{EPITECH}