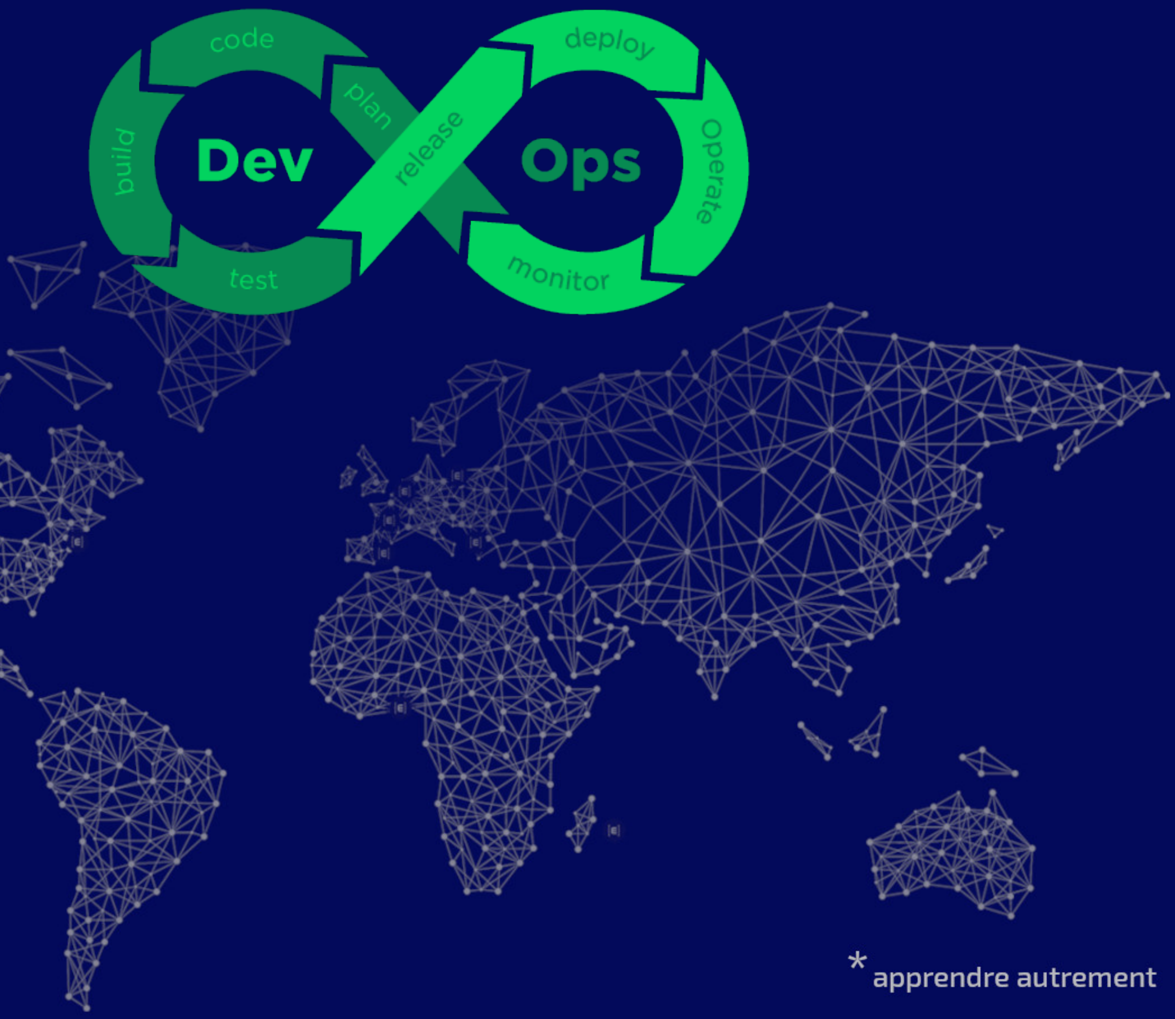
{EPITECH}
LEARN DIFFERENT*

POPEYE

BECOME A TRUE DOCKER SAILOR!



*apprendre autrement

POPEYE

Containerization consist in packaging various things inside one simple standard object, easier to work with. It eases many manipulations (carrying, moving, stacking, ...) and increases tremendously the productivity.

You're about to master the basics of containerizing applications and describing multi-containers infrastructures with Docker and Docker Compose.

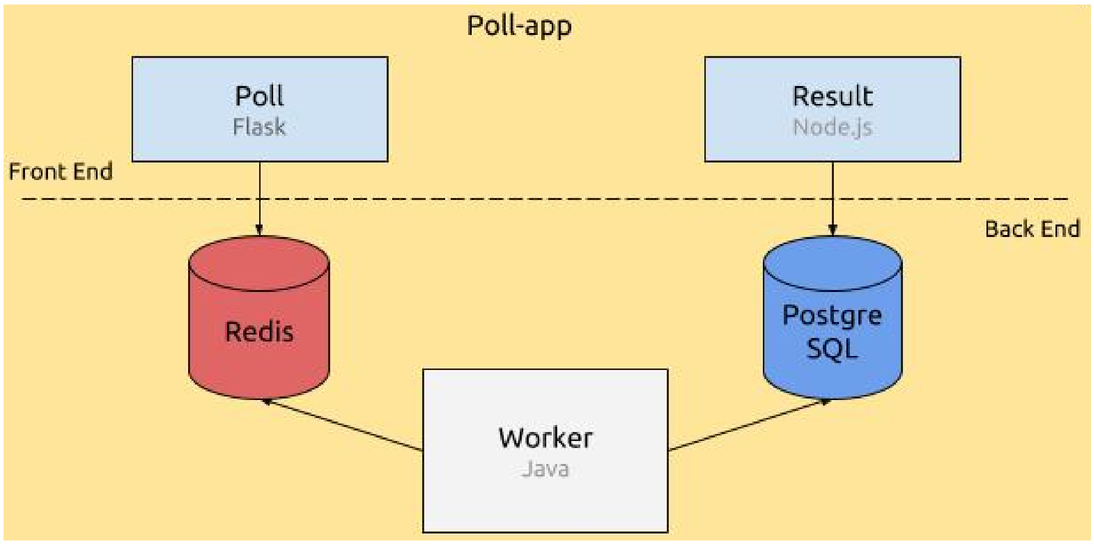
Like the brave sailor that Popeye is, containers can also confidently sail across the vast ocean of operating systems and configurations, being sure of working wherever they might end. As such, containers can be used on any host OS where Docker is installed.

General description

You will have to containerize and define the deployment of a simple web poll application.

There are five elements constituting the application:

- ✓ ***Poll***, a Flask Python web application that gathers votes and push them into a Redis queue.
- ✓ A **Redis queue**, which holds the votes sent by the Poll application, awaiting for them to be consumed by the Worker.
- ✓ The ***Worker***, a Java application which consumes the votes being in the Redis queue, and stores them into a PostgreSQL database.
- ✓ A **PostgreSQL database**, which (persistently) stores the votes stored by the Worker.
- ✓ ***Result***, a Node.js web application that fetches the votes from the database and displays the… well, result. ;)



Do not worry, you do not have to code them! Popeye is generous, the code of these applications is given to you on the intranet's project page.

In DevOps, it is especially important that you take the time to research and understand the technologies you are asked to work with, as you will need to understand how and by which way you can configure them as needed.

Docker images

You have to create 3 images, respecting the specifications described below.

Popeye does not like the `ENTRYPOINT` instruction, so you must not use it in this project.

It's part of your duties to find what versions the different elements of your infrastructure need.

Latest versions are not always compatible with everything.

Poll

- ✓ the image is based on a Python official image ;
- ✓ the apps' dependencies are installed with `pip3 install -r requirements.txt` ;
- ✓ the app exposes and runs on port 80 ;
- ✓ the app is started with `flask run --host=0.0.0.0 --port=80`.

Result

- ✓ the image is based on an official Node.js v12 Alpine image ;
- ✓ the app exposes and runs on port 80 ;
- ✓ the apps' dependencies are installed with `npm install`.

Be careful about the location where this command has to be run.

The `node_modules` folder must be excluded from the build context.

Worker

The image is built using a multi-stage build:

✓ First stage - compilation:

- is based on `maven:3.8.4-jdk-11-slim` and is named `builder`.
- is used to build (natuurlijk) and package the Worker application using:
  - `mvn dependency:resolve` from within the folder containing `pom.xml`;
  - then `mvn package` from within the folder containing the `src` folder.
- generates a file in the `target` folder named `worker-jar-with-dependencies.jar`

This folder path is relative to your `WORKDIR`.

✓ Second stage - run:

- is based on `openjdk:11-jre-slim` ;
- is the one really running the worker using `java -jar worker-jar-with-dependencies.jar`.

Your Docker images must be as simple and lightweight as possible.

Connection details (such as addresses, ports, credentials, . . .) are hard-coded into the apps. This is bad and nasty! Use some environment variables and update the apps' code.

Docker Compose

Now, you should have 3 Dockerfiles creating 3 isolated images.
It is now time to make them all work together using Docker Compose!

Create a `docker-compose.yml` file that will be responsible for running and linking your containers.
You MUST use version 3 of Docker Compose's syntax.

Your Docker Compose file should contain:

✓ **5 services**:

- `poll`:
  * builds your `poll` image ;
  * redirects port 5000 of the host to the port 80 of the container.
- `redis`:
  * uses an existing official image of Redis ;
  * opens port 6379.
- `worker`:
  * builds your `worker` image.
- `db`:
  * represents the database that will be used by the apps ;
  * uses an existing official image of PostgreSQL ;
  * has its database schema created during container first start.
- `result`:
  * builds your `result` image ;
  * redirects port 5001 of the host to the port 80 of the container.

Databases must be launched before the services that use them, because these services are *depending on* them.

✓ **3 networks**:

- `poll-tier` to allow `poll` to communicate with `redis`.
- `result-tier` to allow `result` to communicate with `db`.
- `back-tier` to allow `worker` to communicate with `redis` and `db`.

✓ **1 named volume**:

- `db-data` which allows the database's data to be persistent, if the container dies.

The path of data in the official PostgreSQL image is documented on Docker Hub.

Do not add unnecessary data, such as extra volumes, extra networks, unnecessary inter-container dependencies, or unnecessary container commands/entrypoints.

Once your `docker-compose.yml` is complete, you should be able to run all the services and observe the votes you submitted to the `poll`'s webpage on `result`'s webpage.

Your containers must restart automatically when they stop unexpectedly.

The environment variables or environment variables files (depending on what you choose to use) **have** to be defined in the `docker-compose.yml` file.
The environment variables defined by `ENV` instructions in Dockerfiles, or the environment variables files not specified in the `docker-compose.yml` file will **not** be taken into account.

Technical formalities

Your project will be entirely evaluated with Automated Tests, by analyzing your configuration files (the different Dockerfiles and `docker-compose.yml`).

In order to be correctly evaluated, your repository must at least contain the following files:

```
.
|-- docker-compose.yml
|-- poll
|    `-- Dockerfile
|-- result
|    `-- Dockerfile
|-- schema.sql
`-- worker
     `-- Dockerfile
```

`poll`, `result` and `worker` are the directories containing the provided applications.

v 1.5.1

*apprendre autrement