T6 - DevOps

T-DOP-600

DevOps

Docker Containerization

1.3

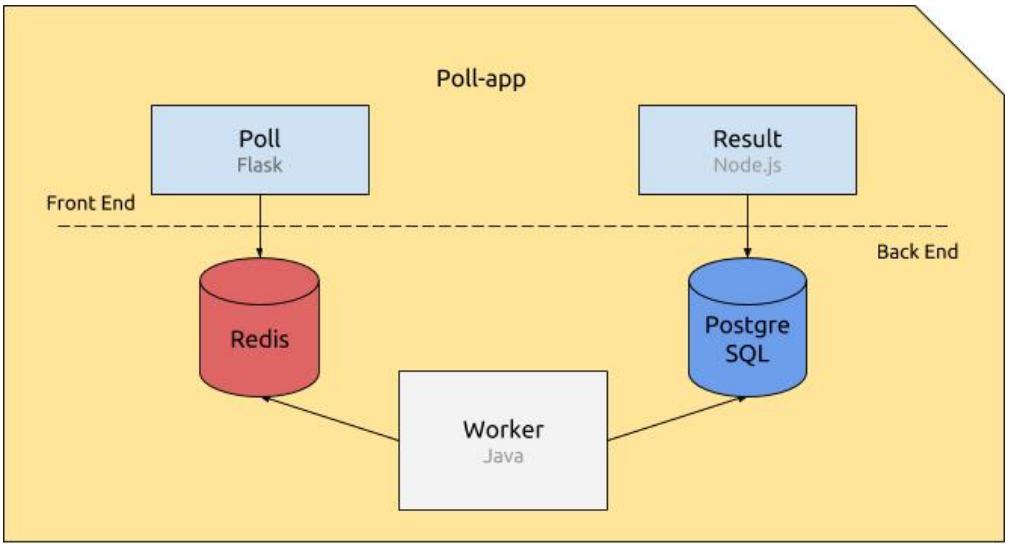
DevOps

repository name: DOP_docker_*$ACADEMICYEAR*
repository rights: ramassage-tek

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.

This project aims to teach you the basics of containerization using docker and docker-compose.

The application you are working on during this project is a simple poll web application.
Poll is a Python Flask web application that gathers the votes to push them into a Redis queue.
The Java Worker consumes the votes stored in the Redis queue, then pushes it into a PostgreSQL database.
Finally, the Node.js Result web application fetches the votes from the DB and displays the result.



Start by becoming familiar with the concept of Docker and the Docker Hub.

DOCKER

You are to create three docker containers.
First, there is one for the poll application itself.

- it must be based on one of python' official container.
- the application must be run on the port 80

The dependencies of this project can be installed using this command (python related): `pip install -r requirements.txt`

The version of python installed within your container must match the one used by the application.

The application can be started with `flask run --host=0.0.0.0 --port=80`.
Then, one for the "result" application.

- it must be based on one of node' official container.
- the application must use the port 80

The dependencies of this project can be installed using the command: `npm install` from within the folded containing `package.json`.
Finally, one container for the worker using multistage build.

- The first stage must be based on `maven:3.5-jdk-8-alpine` and be named `builder`.
  You must use it to build the worker application and package it using the following commands:
  - `mvn dependency:resolve`, from within the folder containing `pom.xml`.
  - Then, `mvn package`, from within the folder containing the `src` folder.
    It generates a file in `target/worker-jar-with-dependencies.jar` (relative to your `WORKDIR`).
- The second stage must be based on `openjdk:8-jre-alpine`.
  This is the one really running the worker using `java -jar worker-jar-with-dependencies.jar`.

Your Docker images must be simple, lightweight and not bring too much things.

Databases addresses and credentials are hard-coded into applications. You must use environment variables. Update application code if necessary.

DOCKER-COMPOSE

You now have 3 dockerfiles that create 3 isolated images. Let's 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 should use docker-compose version 3 syntax.

Your docker-compose file should contain:
5 services:

- `poll`:
  - build your `poll` image and redirect port 5000 of the host to the port 80 of the image.
- `redis`:
  - use an existing image of `redis` and open port 6379
- `worker`:
  - build your `worker` image.
- `db`:
  - representing the database that will be used by the apps. You must use an existing image of `postgres`.
- `result`:
  - build your `result` image and redirect port 5001 of the host to the port 80 of the image.

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 volume:

- `db-data` will make the database data persistent, if the container dies.

The path of data in the official postgres image, is documented on Docker hub.

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

Your containers must restart automatically when they stop unexpectedly.

In the end, your repository must at least contain the following files:

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

`poll`, `result` and `worker` being the directories containing the apps, given to you on the intranet.

GOING FURTHER

- `flask run` is not recommanded for production environments. Find out why, and fix that.
- We would like to use your Docker image in a development environment, without rebuilding images after every code change.
- If you write different configurations for production in `docker-compose.prod.yml`, find a way to limit duplication of `docker-compose.yml`.
- Argh, Docker logs are filling up my hard drive…
- One of NodeJS dependencies is eating to much memory (memory leaks ?). Please limit resources!