ENGINEERING SCHOOL
Creating the future together

REPORT DEVOPS

PART 01: DOCKER

WRITTEN BY :
ASSELE TSETSE Maria

PROFESSOR LEADER:
CHOUIB Chawki

Acknowledgements

I would like to express my deep gratitude to all the individuals who contributed to the organization and execution of the DevOps courses and practical work. Their sustained efforts, expertise, and dedication have been instrumental in helping me gain understanding.

I warmly thank my instructors for their enlightening teaching, insightful guidance, and availability throughout my learning journey. Their clear explanations, practical demonstrations, and passion for the subject greatly enhanced my understanding and strengthened my skills.

I also want to express my special gratitude to M. CHOUIB Chawki for their exceptional commitment and constant dedication to our learning. Their deep knowledge, invaluable support, and responses to all our questions have been sources of inspiration and motivation throughout this DevOps training. Their tireless efforts to guide us and help us understand complex concepts have had a significant impact on our progress and success. I am sincerely grateful to them for their valuable contribution to our training.

Introduction

Adopting DevOps practices in software development has become essential to ensure efficient, fast, and reliable application deployments. In this context, the use of tools such as Docker offers a modern approach to managing environments and containerizing applications. This report details the process of setting up a complete DevOps infrastructure by following the practical steps proposed in TP part 01 - Docker. From configuring the database to publishing Docker images, most of the steps are described to provide a complete overview of the process.

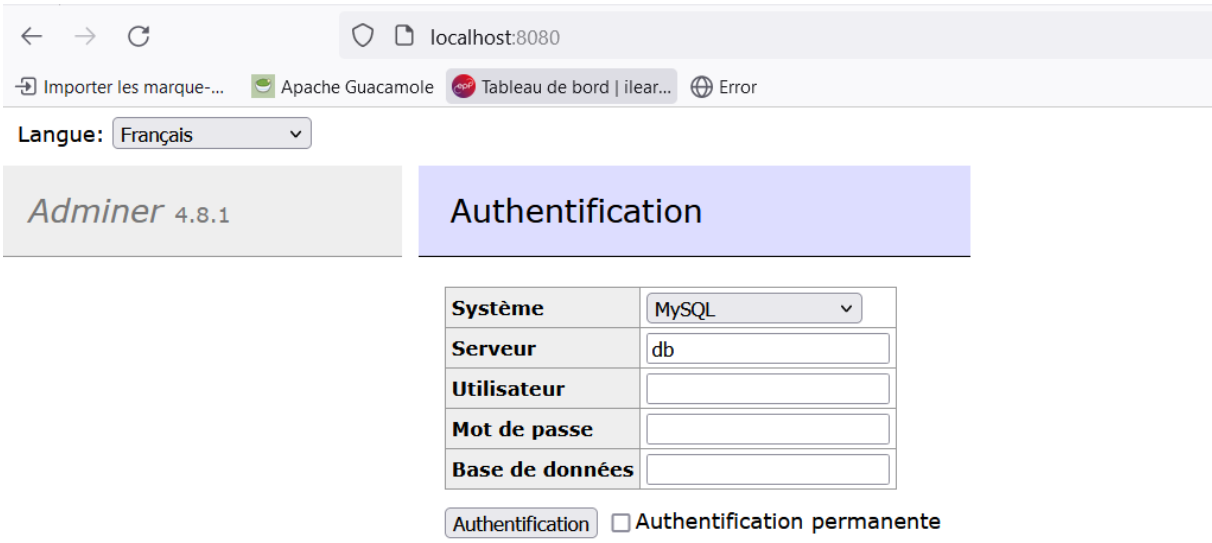
Table of contents

I- Database

Basics

To set up a PostgreSQL server, I used the postgres:14.1-alpine image. I started by creating a minimalistic Dockerfile to configure the PostgreSQL server with the necessary environment variables. Then I built the Docker image using the docker build command which builds the Docker image using the Dockerfile I created and labeling it with the name my-postgres:14.1-alpine.

I started a container from the built image making sure to bind the ports correctly to be able to access the database. To enable communication between PostgreSQL and Adminer, I used a Docker network. I created a network named app-network, on which I ran the PostgreSQL container and started Adminer.



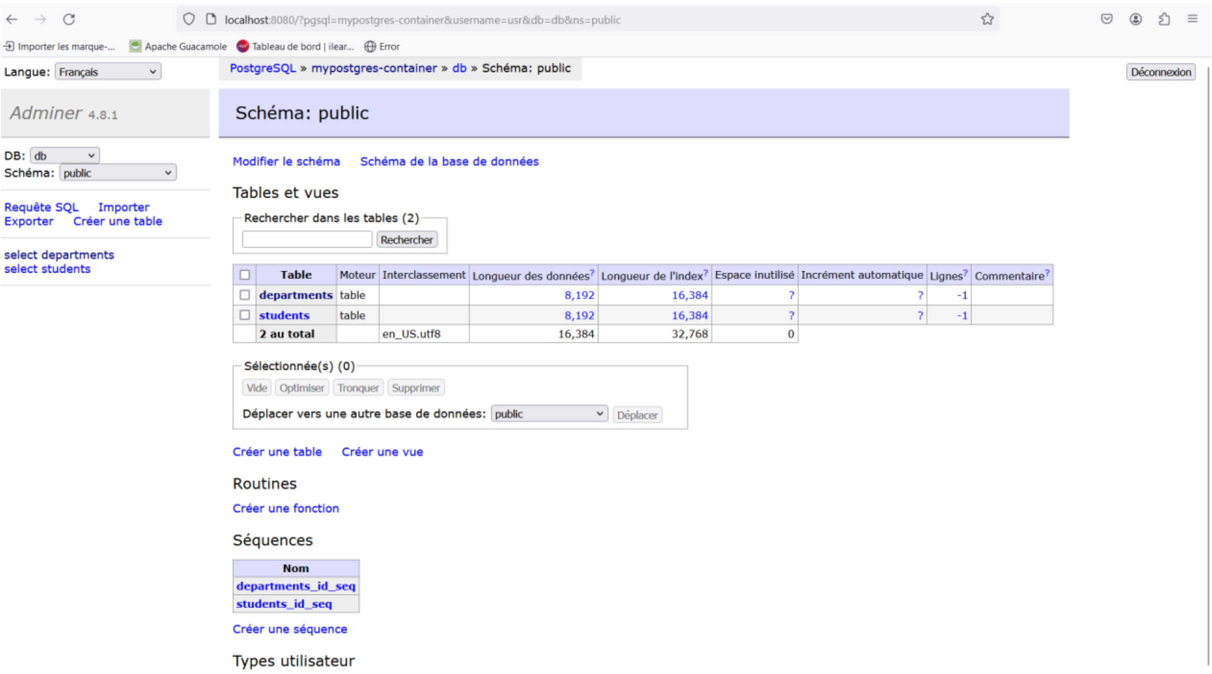
Init database

To initialize the database structure as well as some initial data, I created two SQL scripts for creating the database structure and inserting the initial data.

01-createSchema	27/05/2024 10:57	Fichier source SQL	1 Ko
02-InsertData	27/05/2024 11:15	Fichier source SQL	1 Ko

To make these scripts run when the PostgreSQL container starts, I added them to my Dockerfile. I then rebuilt the Docker image and started a container from that image.

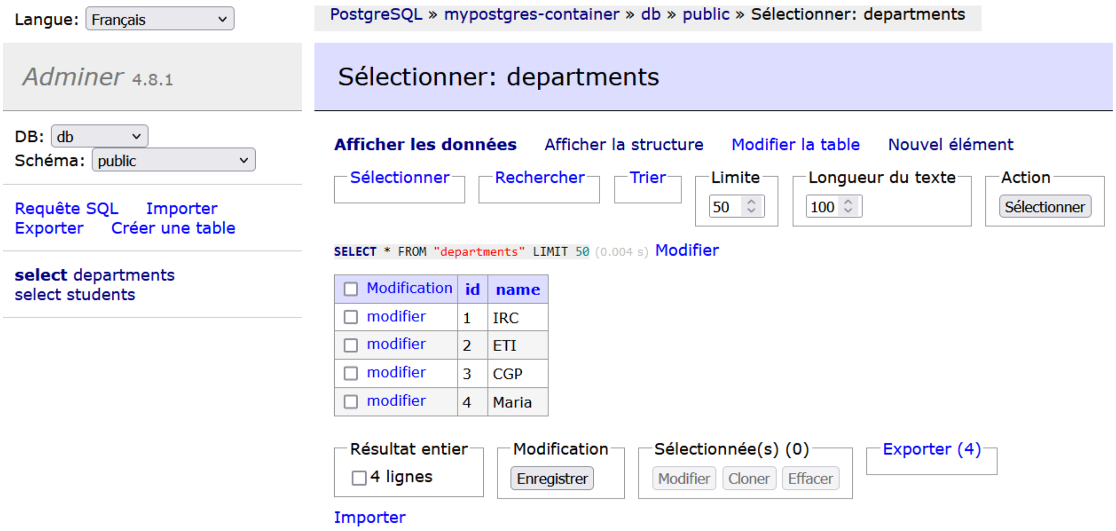
```
FROM postgres:14.1-alpine

ENV POSTGRES_DB=db \
   POSTGRES_USER=usr \
   POSTGRES_PASSWORD=pwd
COPY 01-createSchema.sql /docker-entrypoint-initdb.d/01-createSchema.sql
COPY 02-InsertData.sql /docker-entrypoint-initdb.d/02-InsertData.sql
```



Persist data

It is essential to ensure that database data persists even if the database container is destroyed or restarted. Without this persistence, all data would be lost every time the container is recreated. To avoid this, using Docker volumes allows database data to be stored on the host disk in a durable manner.

To ensure this, I used a Docker volume that mounts a host directory to the PostgreSQL data directory inside the container. By using a Docker volume for data persistence, database data is permanently stored on the host's disk, ensuring that information is not lost when the container is destroyed or restarted.



II- Backend API

1. Basics

A Main.java file was used for the initial example and the compilation was done with

```
PS C:\Users\assel\Documents\Java_api> java Main
Hello World!
```

I created a Dockerfile to run the compiled .class file. Then I built and ran the container. At this point I verified that the Hello World! displayed well in the console.

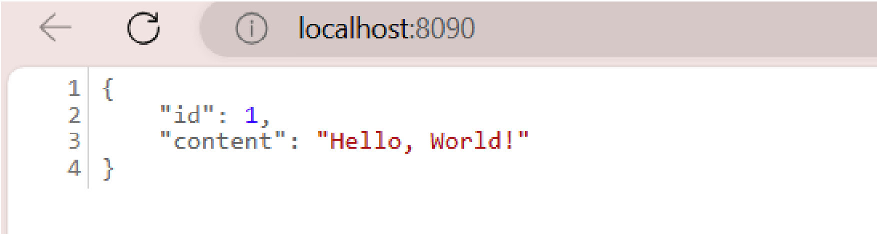
```
PS C:\Users\assel\Documents\Java_api> docker run java_api
Hello World!
```

2. Multistage build

To simplify and centralize compiling and running code in Docker, I used a multi-stage build, after using the Dockerfile to handle compilation and execution in separate stages, I built and ran the container

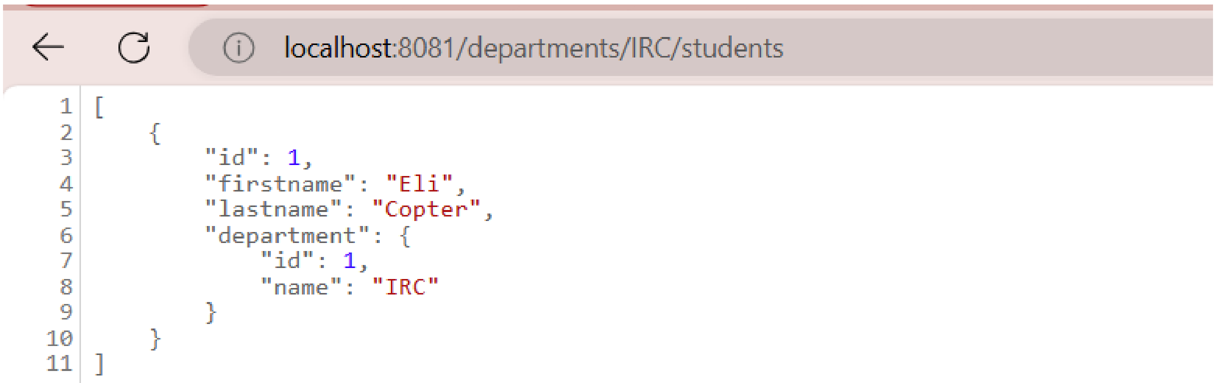
a. Backend simple api

I generated a Spring Boot application via Spring Initializer with the provided configurations, I then added the given GreetingController class and used the Dockerfile to build and run the Spring Boot application in Docker. I built and ran the container and checked the API by accessing the hotspot *http://localhost:8090* which returned a hello.



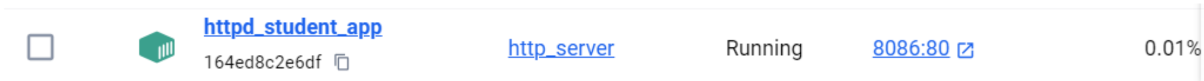
b. Backend API

For this part of the project, the goal was to build and run a backend API connected to a database. I modified the application.yml file to configure the application: to enable communication between the database container and the backend API container, I created a Docker network. I ran the Database and Adminer containers with the created Docker network and built the backend API Docker image using the provided multi-step Dockerfile. Once the container was running, I accessed the API to check how it was working.



III- Http server

I chose the httpd:2.4-alpine base image for the HTTP server. I created a simple index.html home page, I then added this into the container. I created a Dockerfile for the HTTP server, built the image and started the container.



I verified that the HTTP server is working correctly by going to http://localhost.

localhost:8086

```json
{
    "id": 3,
    "content": "Hello, World!"
}
```

localhost:8086/departments/IRC/students

```json
[
    {
        "id": 1,
        "firstname": "Eli",
        "lastname": "Copter",
        "department": {
            "id": 1,
            "name": "IRC"
        }
    }
]
```

I managed to setup a basic HTTP server and turn it into a reverse proxy to redirect requests to my backend application. This allows for a more modular and flexible architecture, making it easier to manage different parts of the application.

IV- Link application

I created a docker-compose.yml file that orchestrates the three containers (backend, database, httpd). Docker-compose is essential because it allows multiple Docker containers to be managed and orchestrated as services. This simplifies the process of starting, stopping and rebuilding containers, ensuring that all dependencies are respected and that services are correctly connected together. I started the services using the docker-compose up command and verified that the application is working correctly by accessing the API on localhost. Everything works as expected, with a three-tier application running orchestrated by Docker-compose.

docker-compose		Running (3/3)		1.06%	1 minute ago
database-container 6fae471c997f	docker-compose-data	Running		0.79%	1 minute ago
backend-container a5211ae3653f	docker-compose-back	Running		0.18%	1 minute ago
frontend-container 9dae552ec36c	docker-compose-http	Running	80:80	0.09%	1 minute ago

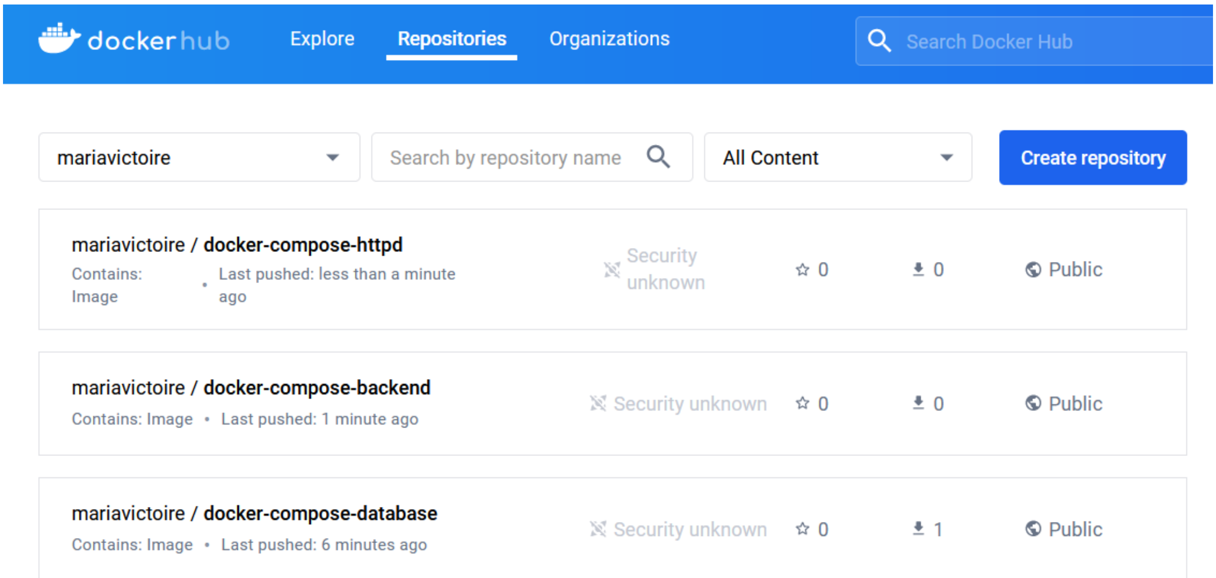
localhost/departments/IRC/students


```json
[
    {
        "id": 1,
        "firstname": "Eli",
        "lastname": "Copter",
        "department": {
            "id": 1,
            "name": "IRC"
        }
    }
]
```


V- Publish

To allow other team members or other machines to use the created Docker images, it is necessary to publish them to an online Docker image registry, such as Docker Hub. Publishing Docker images to Docker Hub is a crucial step to effectively sharing and managing images within a development team. This ensures constant availability and facilitates continuous integration and deployment of applications.

```
PS C:\Users\assel\Documents\Docker-compose> docker push mariavictoire/docker-compose-database:1.0
The push refers to repository [docker.io/mariavictoire/docker-compose-database]
0df33f8812ec: Pushed
db3995408cf4: Pushed
5b87e9731513: Mounted from library/postgres
176b9203da6e: Mounted from library/postgres
efb18f6577c9: Mounted from library/postgres
6c651825e7c4: Mounted from library/postgres
be6c168b4af5: Mounted from library/postgres
b737c2580132: Mounted from library/postgres
6cab14f8a434: Mounted from library/postgres
8d3ac3489996: Mounted from library/postgres
1.0: digest: sha256:52832df892927189ff0290c93ab9b0598e2994862a82352714014208b15cc66d size: 2399
```



Conclusion

This Part 01 - Docker provided a hands-on and informative experience in implementing DevOps infrastructure using tools like Docker. Through database configuration, backend API deployment, and setting up an HTTP server, I was able to gain an understanding of key DevOps concepts and their application in modern software development. By using Docker-compose to orchestrate containers and publishing Docker images to Docker Hub, I learned how to automate and standardize deployment processes, which contributes to better application efficiency and reliability. This report highlights the growing importance of adopting DevOps practices in contemporary software development to meet the demands of speed, flexibility and quality of software products.