

TP 1 : Discover Docker

1-1 Document your database container essentials: commands and Dockerfile.

Dans un premier temps, on crée le fichier qui a pour rôle de créer les tables puis celui insérant les données

```
CREATE TABLE public.departments
(
  id      SERIAL      PRIMARY KEY,
  name    VARCHAR(20) NOT NULL
);

CREATE TABLE public.students
(
  id            SERIAL      PRIMARY KEY,
  department_id INT         NOT NULL REFERENCES departments (id),
  first_name    VARCHAR(20) NOT NULL,
  last_name     VARCHAR(20) NOT NULL
);
```

```
INSERT INTO departments (name) VALUES ('IRC');
INSERT INTO departments (name) VALUES ('ETI');
INSERT INTO departments (name) VALUES ('CGP');

INSERT INTO students (department_id, first_name, last_name) VALUES (1, 'Eli', 'Copter');
INSERT INTO students (department_id, first_name, last_name) VALUES (2, 'Emma', 'Carena');
INSERT INTO students (department_id, first_name, last_name) VALUES (2, 'Jack', 'Uzzi');
INSERT INTO students (department_id, first_name, last_name) VALUES (3, 'Aude', 'Javel');
```

On crée ensuite un dockerfile qui va créer une image Docker exécutant un serveur PostgreSQL avec ses identifiants et son contenu

```
FROM postgres:14.1-alpine

COPY ./db/sql /docker-entrypoint-initdb.d

ENV POSTGRES_DB=db \
    POSTGRES_USER=usr \
    POSTGRES_PASSWORD=pwd
```

On utilise ensuite les commandes suivantes

- « docker network create app-network » : crée le réseau gérant la connectivité entre les containers
- « docker build -t thibaultclt/database » : Construction du conteneur de la BDD
- « docker run -p "8090:8080" --net=app-network --name=adminer -d adminer » : Lancement du conteneur sur le port 8090 avec l'image Adminer

1-2 Why do we need a multistage build? And explain each step of this dockerfile.

Avec les multi-stage builds, on utilise plusieurs instructions FROM dans le Dockerfile. Chaque instruction FROM peut utiliser une image de base différente et commence une nouvelle étape de la construction. On peut copier des artefacts d'une étape à une autre, en laissant derrière tout ce dont on n'a pas besoin dans l'image finale.

Ça a plusieurs intérêts :

- Réduire l'image de la taille finale qui ne contiendra que les composants nécessaires
- Avoir une sécurité accrue du fait qu'on peut ne pas inclure les dépendances sensibles dans l'image finale de production.

- Chaque stage a un environnement clean et isolé ce qui peut éviter certains conflits ou problèmes pendant certains processus

Dans la première partie, on définit une image de base maven:3.8.6-amazoncorretto-17 pour la construction de l'application. Les étapes de cette partie sont les suivantes :

FROM maven:3.8.6-amazoncorretto-17 AS myapp-build : Cela configure la première étape et lui donne le nom "myapp-build".

ENV MYAPP_HOME /opt/myapp : On définit une variable d'environnement pour le répertoire de base de l'application.

WORKDIR \$MYAPP_HOME : On définit le répertoire de travail au répertoire de base de l'application.

COPY pom.xml . et COPY src ./src : On copie le fichier pom.xml et le code source de l'application depuis le répertoire local vers le répertoire de travail dans le conteneur.

RUN mvn package -DskipTests : On exécute la commande mvn package pour construire l'application. L'option -DskipTests indique de ne pas exécuter les tests pendant le processus de construction.

Dans la deuxième partie, on utilise une image de base plus légère, amazoncorretto:17, pour exécuter l'application. Les étapes de cette partie sont les suivantes :

FROM amazoncorretto:17 : On configure la deuxième étape en utilisant l'image de base Amazon Corretto 17.

ENV MYAPP_HOME /opt/myapp : On définit à nouveau la variable d'environnement pour le répertoire de base de l'application, pour garantir la cohérence avec la première étape.

WORKDIR \$MYAPP_HOME : On définit le répertoire de travail dans cette étape également.

COPY --from=myapp-build \$MYAPP_HOME/target/*.jar \$MYAPP_HOME/myapp.jar : On copie le fichier JAR de l'application construite dans la première étape vers cette étape, afin de l'inclure dans l'image finale.

ENTRYPOINT java -jar myapp.jar : On spécifie la commande à exécuter lorsque le conteneur démarre. Dans ce cas, il s'agit de l'exécution de l'application Java à l'aide du fichier JAR copié

1-3 Document docker-compose most important commands.

docker-compose up : permet de démarrer l'application docker compose, elle lit le fichier docker-compose.yml, crée et démarre tous les services définis en tant que conteneurs

docker-compose build : reconstruit les images de tous les services définis dans le fichier docker-compose.yml

docker-compose ps : affiche l'état des conteneurs gérés par Docker Compose, indiquant s'ils sont en cours d'exécution ou arrêtés

1-4 Document your docker-compose file.

```
version: '3.7'

services:
  backend:
    build:
      context: ./simple-api-student-main
      dockerfile: Dockerfile
    container_name: simpleapistudent
    networks:
      - app-network
    depends_on:
      - database

  database:
    build:
      context: ./db
      dockerfile: Dockerfile
    container_name: database
    networks:
      - app-network

  httpd:
    build:
      context: ./devops-front-main
      dockerfile: Dockerfile
    container_name: http
    ports:
      - "80:80"
    networks:
      - app-network
    depends_on:
      - backend

networks:
  app-network:
```

Ce docker-compose file permet de définir tous les services, leur chemin, container, réseaux et le service sur lequel ils dépendent s'ils en ont un

1-5 Document your publication commands and published images in dockerhub.

docker tag thibaultclt/database thibaultclt/database:1.0

docker push thibaultclt/database:1.0


docker tag thibaultclt/http thibaultclt/http:1.0

docker push thibaultclt/http:1.0

docker tag thibaultclt/simpleapistudent thibaultclt/simpleapistudent:1.0



docker push thibaultclt/simpleapistudent:1.0

Thibault Collet

thibaultclt/http •  6 •  0



By [thibaultclt](#) • Updated 6 days ago

 Image

thibaultclt/database •  6 •  0

By [thibaultclt](#) • Updated 6 days ago

 Image

thibaultclt/simpleapistudent •  6 •  0

By [thibaultclt](#) • Updated 6 days ago

 Image

TP 2: Discover Github Action

2-1 What are testcontainers?

Les testcontainers sont des librairies java qui permettent de run des container docker tout en faisant les tests d'intégration et les tests unitaires.

2-2 Document your Github Actions configurations.

```
name: CI devops 2023
on:
  #to begin you want to launch this job in main and develop
  push:
    branches:
      - master
  pull_request:

jobs:
  test-backend:
    runs-on: ubuntu-22.04
    steps:
      #checkout your github code using actions/checkout@v2.5.0
      - name: Checkout Repository
        uses: actions/checkout@v2.5.0

      #do the same with another action (actions/setup-java@v3) that enable to setup jdk 17
      - name: Set up JDK 17
        uses: actions/setup-java@v2 # Assurez-vous d'utiliser la version correcte de l'action
        with:
          java-version: 17
          distribution: 'adopt'

      #finally build your app with the latest command
      - name: Build and test with Maven
        working-directory: simple-api-student-main
        run: mvn -B verify sonar:sonar -Dsonar.organization=tp-devops-clt -Dsonar.host.url=https://sonarcloud.io -Dsonar.login=${{ secrets.SONAR_TOKEN }} --file ./pom.xml
```

```
# define job to build and publish docker image
build-and-push-docker-image:
  needs: test-backend
  # run only when code is compiling and tests are passing
  runs-on: ubuntu-22.04

  # steps to perform in job
  steps:
    - name: Checkout code
      uses: actions/checkout@v2.5.0

    - name: Login to DockerHub
      run: docker login -u ${ secrets.DOCKERHUB_USERNAME } -p ${ secrets.DOCKERHUB_PASSWORD }

    - name: Build image and push backend
      uses: docker/build-push-action@v3
      with:
        context: ./simple-api-student-main
        tags: ${ secrets.DOCKERHUB_USERNAME }/simpleapistudent
        push: ${ github.ref == 'refs/heads/master' }

    - name: Build image and push database
      uses: docker/build-push-action@v3
      with:
        context: ./db
        tags: ${ secrets.DOCKERHUB_USERNAME }/database
        push: ${ github.ref == 'refs/heads/master' }

    - name: Build image and push httpd
      uses: docker/build-push-action@v3
      with:
        context: ./frontend
        tags: ${ secrets.DOCKERHUB_USERNAME }/http
        push: ${ github.ref == 'refs/heads/master' }
```

✓ Ajout working-directory #5

Re-run all jobs

Summary

Jobs

✓ test-backend

Run details

Usage

Workflow file

Triggered via push last week

ThibaultClit pushed → ea8e6da master

Status

Success

Total duration

1m 16s

Artifacts

–

main.yml

on: push

✓ test-backend

1m 5s

🔄 – +

2.3 Document your quality gate configuration.

472 Lines of Code ?

Version 0.0.1-SNAPSHOT Last analysis 5 days ago be9cb3c2

✓

Quality Gate ?

New Code

Overall Code

Passed

New code: Since 7 days ago

Reliability

0 Bugs ?

A

Maintainability

0 Code Smells ?

A

Security

0 Vulnerabilities ?

A

Security Review

0 Security Hotspots ?

A

Coverage

— Coverage ?

on 0 New Lines to cover

Duplications

— Duplications ?

on 0 New Lines

TP 3: Ansible

3-1 Document your inventory and base commands

```
all:
  vars:
    ansible_user: centos
    ansible_ssh_private_key_file: ../../id_rsa
  children:
    prod:
      hosts: thibault.collet.takima.cloud
```

3-2 Document your playbook

```
- hosts: all
  gather_facts: false
  become: yes
  roles:
    - docker
    - network
    - database
    - app
    - proxy
```

Ce fichier playbook définit l'ensemble des rôles que l'on va utiliser pour créer nos containers

3.3 Document your docker_container tasks configuration.

```
- name: Clean packages
  command:
    cmd: yum clean -y packages

- name: Install device-mapper-persistent-data
  yum:
    name: device-mapper-persistent-data
    state: latest

- name: Install lvm2
  yum:
    name: lvm2
    state: latest

- name: add repo docker
  command:
    cmd: sudo yum-config-manager --add-repo=https://download.docker.com/linux/centos/docker-ce.repo

- name: Install Docker
  yum:
    name: docker-ce
    state: present

- name: Make sure Docker is running
  service: name=docker state=started
  tags: docker
```