Compte Rendu - DevOps

TP part 1 - Docker

> **Document your database container essentials: commands and Dockerfile**

Commandes:

`$ docker build -t julia/database ./database`

`$ docker run -d --name database -v database-data:/var/lib/postgresql/data --network=app-network julia/database`

`$ docker run -p "8090:8080" --network=app-network --name=adminer -d adminer`

Dockerfile:

```
FROM postgres:14.1-alpine

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

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

COPY ./InsertData.sql /docker-entrypoint-initdb.d
```

> **Why do we need a multistage build? And explain each step of this dockerfile**

Le multistage build permet de garder uniquement ce qui est utile lors de la phase de run et cela évite d'accumuler les éléments du docker qui ne sont plus utiles après l'initialisation du projet. Il y a donc le build et le run qui va se lancer à partir du build. Le container sera donc plus léger.

Le build installe le package maven avec les dépendances.

Le run récupère ensuite ce qui a été build et le copie.

Commandes:

`$ docker build -t julia/helloworldjava .`

`$ docker run -d --name helloworldjava julia/hellowordjava`

Dockerfile:

```
# Build
FROM maven:3.6.3-jdk-11 AS myapp-build
ENV MYAPP_HOME /opt/myapp
WORKDIR $MYAPP_HOME
COPY pom.xml .
COPY src ./src
RUN mvn package -DskipTests

# Run
FROM openjdk:11-jre
ENV MYAPP_HOME /opt/myapp
WORKDIR $MYAPP_HOME
COPY --from=myapp-build $MYAPP_HOME/target/*.jar $MYAPP_HOME/myapp.jar
ENTRYPOINT java -jar myapp.jar
```

> **Document docker-compose most important commands. Document your docker-compose file**

`$ docker compose -d up`

`$ docker compose -d down`

Docker-compose.yml:

On run le back, la database et le proxy. On précise pour chacun le dossier a build, le nom du container une fois build, le réseau. Pour certains, il faut préciser si on doit les restart ou pas.

```
version: '3.3'
services:
  backend:
    container_name: backend
    build: ./simple-api
    networks:
      - app-network
    depends_on:
      - database
    restart: on-failure
    volumes:
      - database-data:/var/lib/postgresql/data

  database:
    container_name: database
    restart: always
    build: ./database
    networks:
      - app-network
    env_file:
      - database/.env

  httpd:
    container_name: reverse_proxy
    build: ./httpd
    ports:
      - "80:80"
    networks:
      - app-network

volumes:
  database-data:

networks:
  app-network:
```

TP part 2 - Github actions

> **What are testcontainers?**

Il s'agit de bibliothèques Java qui permettent d'exécuter un certain nombre de conteneurs Docker pendant les tests.

> **Document your Github Actions configurations**

Dans le fichier ci-dessous, on fait un checkout sur notre code, on met en place la JDK 11 pour pouvoir compiler notre code, mise en cache des package sonar cloud, mise en cache des packages maven, on build le projet et on lance la vérification par sonarcloud.

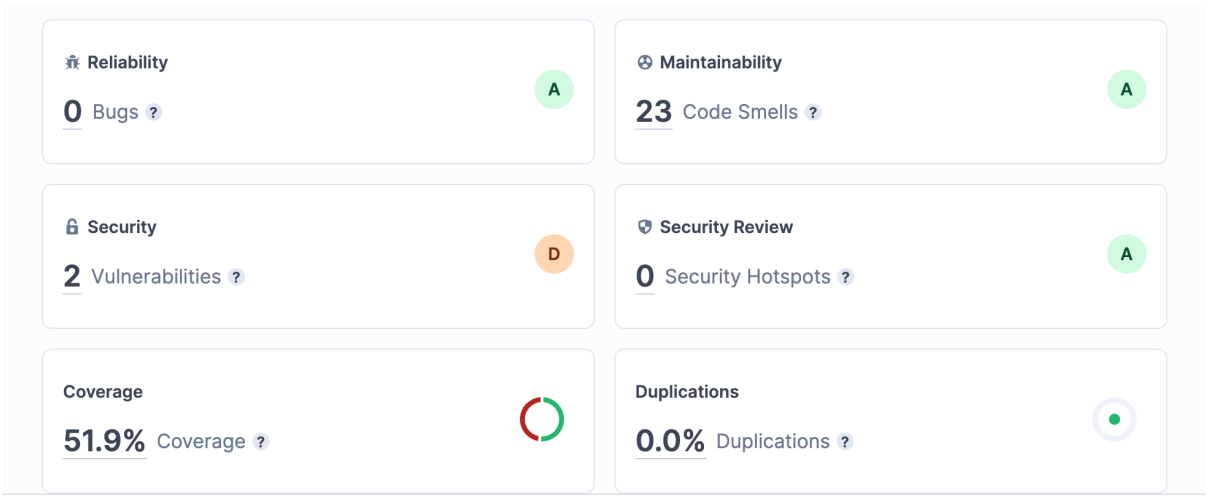
Une fois la partie test-simple-api terminée, on fait un checkout sur notre code, on se connecte à DockerHub avec notre identifiant et notre token, on build le backend et on le push sur dockerhub, de même pour la base de donnée et le proxy.

.github/workflows/Main.yml:

```
name: CI devops 2022 EPF
on:
  #to begin you want to launch this job in main and develop
  push:
    branches: main
  pull_request:

jobs:
  test-simple-api:
    runs-on: ubuntu-22.04
    steps:
      #checkout your github code using actions/checkout@v2.3.3
      - uses: actions/checkout@v2.3.3

      #do the same with another action (actions/setup-java@v2) that enable to setup jdk 11
      - name: Set up JDK 11
        uses: actions/setup-java@v2
        with:
          distribution: 'zulu'
```

```
          java-version: 11
          java-package: jdk

      #finally build your app with the latest command
      - name: Build and test with Maven
        run: mvn -B verify sonar:sonar -Dsonar.projectKey=juliaMasset_devops-tp2 -Dsonar.organization=juliamasset -Dsonar.host.url=htt

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

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

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

    - name: Build image and push backend
      uses: docker/build-push-action@v2
      with:
        # relative path to the place where source code with Dockerfile is located
        context: ./simple-api
        # Note: tags has to be all lower-case
        tags:  ${{secrets.DOCKER_HUB_JULIA}}/devops-tp2:simple-api
        push: ${{ github.ref == 'refs/heads/main' }}

    - name: Build image and push database
      uses: docker/build-push-action@v2
      with:
        # relative path to the place where source code with Dockerfile is located
        context: ./database
        # Note: tags has to be all lower-case
        tags:  ${{secrets.DOCKER_HUB_JULIA}}/devops-tp2:database
        push: ${{ github.ref == 'refs/heads/main' }}

    - name: Build image and push httpd
      uses: docker/build-push-action@v2
      with:
        # relative path to the place where source code with Dockerfile is located
        context: ./httpd
        # Note: tags has to be all lower-case
        tags:  ${{secrets.DOCKER_HUB_JULIA}}/devops-tp2:httpd
        push: ${{ github.ref == 'refs/heads/main' }}
```

Document your quality gate configuration



TP part 3 - Ansible

Document your inventory and base commands

Commandes:

`$ brew install ansible` (commande avec HomeBrew sur mac)

`$ ansible --version`

`$ chmod 400 ./id_rsa`

`$ ansible all -i inventories/setup.yml -m ping`

`$ ansible all -i inventories/setup.yml -m setup -a "filter=ansible_distribution*"`

`$ ansible all -i inventories/setup.yml -m yum -a "name=httpd state=absent" --become`

ansible/inventories/setup.yml:

```
all:
 vars:
   ansible_user: centos
   ansible_ssh_private_key_file: ./../id_rsa
 children:
   prod:
     hosts: julia.masset.takima.cloud
```

Document your playbook

Dedans, on retrouve l'ensemble des rôles définis dans Ansible. Cela permet de lancer toutes les tâches dans l'ordre donné. Ensuite, dans chacun des dossier de chaque rôle (database, httpd, simple-api...) on retrouve le détail des commandes que Ansible doit réaliser.

playbook.yml:

```
- hosts: all
  gather_facts: false
  become: yes

# Install Docker
  roles:
      - docker
      - network
      - database
      - proxy
      - app
```

Document your docker_container tasks configuration

- **clean package**: nettoie tous les packages dans le système
- **Install device-mapper-persistent-data:** permet de donner accès a de l'espace de stockage pour les futurs containers
- **Install lvm2:** installe lvm2 qui permet de gérer le volume physique
- **add repo docker:** ajoute un repository docker
- **Install Docker:** install docker
- **install python3: i**nstalle python 3
- **Pip Install:** installe pip
- **Make sure Docker is running:** on vérifie que docker run et que toutes les commandes se sont bien exécutées

roles/docker/tasks/main.yml:

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

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

- name: Install lvm2
```

```
  dnf:
    name: lvm2
    state: latest

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

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

- name: install python3
  dnf:
    name: python3

- name: Pip install
  pip:
    name: docker

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