TP

# TP 1

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

docker pull postgres

docker pull adminer

docker run -p "8090:8080" --network app-network --name=adminer -d adminer // run l'adminer docker build . -t mydb // build l'image en la nommant mydb

docker run -d --network app-network --name db mydb

Dans le Dockerfile on rajoute pour rajouter la bdd:

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

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

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Backend API

Java : copier les lignes dans un nouveau dockerfile

FROM openjdk:11

COPY . /usr/src/myapp

WORKDIR /usr/src/myapp

RUN javac Main.java

CMD ["java", "Main"]

Et on run

docker build -t my-java-app .

docker run -it --rm --name my-running-app my-java-app

1-2

télécharger le truc sur github ajouter le dockerfile (le même que celui d'avant dans intelliJ)

modifier l'url, le user et le password (jdbc:postgresql://db:5432/db pour l'url) (modifié)

commandes à effectuer :

docker build . -t spring-boot-app

docker run -d --network app-network -p 8888:8080 --rm --name my-running-app spring-boot-app

et après modifier le 8080 en 8888

1-3

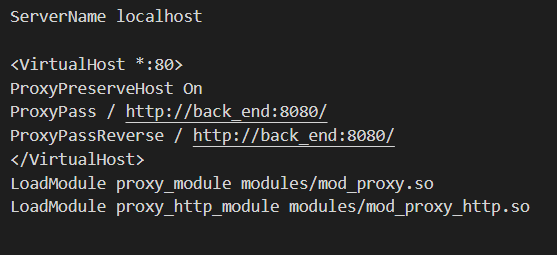
On pull httpd, et on écrit dans le Dockerfile

docker run --rm httpd:2.4 cat /usr/local/apache2/conf/httpd.conf > my-httpd.conf

pour avoir la config, on rajoute

COPY ./my-httpd.conf /usr/local/apache2/conf/httpd.conf

Dans le fichier conf, on rajoute :



Long pb de proxy qui ne voulait pas se connexter à my-java, parce que j’avais oublié de créer le network, on passe donc de :

docker run -dit --name my-running-app -p 8080:80 my-apache2

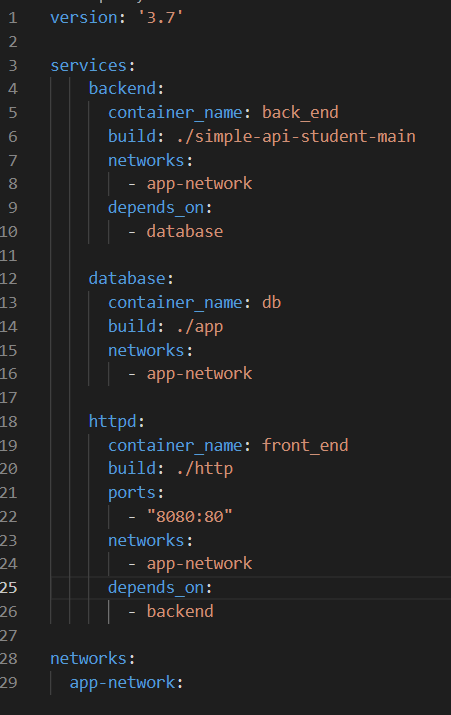
docker run -dit --name my-running-app -p 8080:80 --network app-network my-apache2

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Ca marche !!!

Compose up



Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Publish

docker tag devops-backend lydiebar/devops-backend:1.0

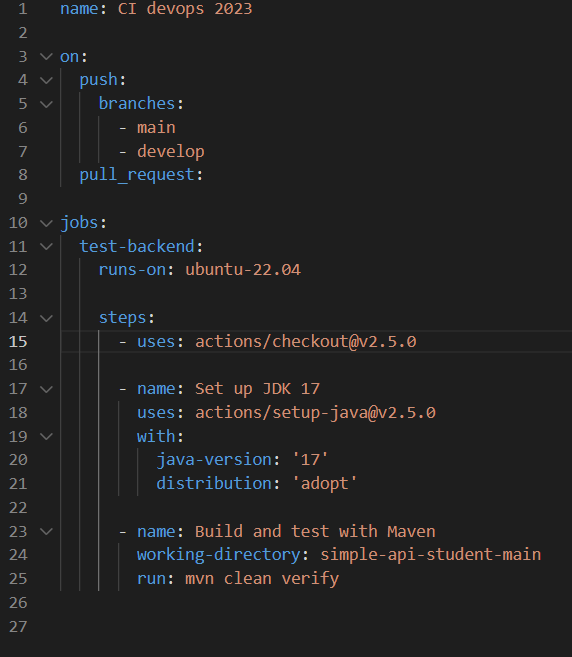
docker push lydiebar/devops-backend:1.0

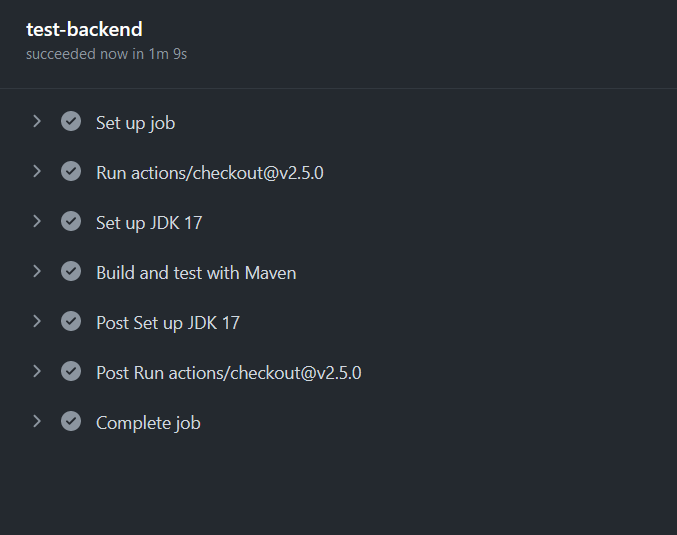
Pour chaque image

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

# TP 2





On crée des secrets :

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Il faut bien vérifier si les tokens ont été entrés correctement.

Sonarcloud :

On crée un compte sonarcloud et un projet en gardant bien les informations, et on crée un token qu’on rajoutera dans github.

On rajoute cette ligne

 - name: Build and test with Maven

        working-directory: simple-api-student-main

        run: mvn -B verify sonar:sonar -Dsonar.projectKey=devops-tp-lydiebar\_devops-tp -Dsonar.organization=devops-tp-lydiebar -Dsonar.host.url=https://sonarcloud.io -Dsonar.login=${{ secrets.SONAR\_TOKEN }} --file ./pom.xml

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

# TP3

Dans le fichier inventories on rajoute ce fichier setup.yml

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Et lorsqu’on ping on obtient :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

On a utilisé trois commandes : ping, setup et yum (et on définit)

Docker\_container configuration :

Install\_docker :

Une image contenant texte, capture d’écran

Description générée automatiquement

On reprend le même code que dans docker.

Create network :

Une image contenant texte, Police, capture d’écran, nombre

Description générée automatiquement

Launch\_database :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Launch\_app :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Launch\_proxy :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Playbook.yml :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Résultat :

Une image contenant texte, Police, ligne, nombre

Description générée automatiquement