# Create the database.

## Basic

I create a project folder with 3 folders inside: backend, database and http-server

In the database folder, I added a Dockerfile with the following code inside:

FROM postgres:14.1-alpine

ENV POSTGRES\_DB=db \

POSTGRES\_USER=usr \

POSTGRES\_PASSWORD=pwd

I build this image with the command: docker build -t hberthon/docker\_tp\_db

And I run it with: docker run -p 5432:5432 --name=db hberthon/docker\_tp\_db

## Init Database

I then create a second container with an imported image with the following commands:

We also create a network to connect all our containers.

docker network create app-network

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

Adminer is a database management tool.

In our database folder, I created 2 files: **01-CreateScheme.sql** and **02-InsertData.sql.**

**In the first file:**

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

);

And in the second:

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');

The previous code allows us to create our tables for students and departments as well as to populate these tables.

## Persist data

In order to persist our data, I created a volume called persistence in our container in the container using the following command:

docker run -p 5432:5432 --net=app-network --name=db -v persistence:/var/lib/postgresql/data hberthon/docker\_tp\_db

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

docker rm -f db

docker ps -a

# Backend API

## Basic

* Create Dockerfile

FROM amazoncorretto:17

COPY . /usr/src/backend-api

WORKDIR /usr/src/backend-api

RUN javac Main.java

CMD ["java", "Main"]

* docker build -t hberthon/docker\_tp\_backend .

Une image contenant texte, capture d’écran

Description générée automatiquement

## Multistage build

* change dockerfile to do multistage build

# Build

FROM maven:3.8.6-amazoncorretto-17 AS myapp-build

ENV MYAPP\_HOME /opt/myapp

WORKDIR $MYAPP\_HOME

COPY pom.xml .

COPY src ./src

RUN mvn package -DskipTests

# Run

FROM amazoncorretto:17

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

* Create a controller

package fr.takima.training.simpleapi.controller;

import org.springframework.web.bind.annotation.\*;

import java.util.concurrent.atomic.AtomicLong;

@RestController

public class GreetingController {

private static final String template = "Hello, %s!";

private final AtomicLong counter = new AtomicLong();

@GetMapping("/")

public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {

return new Greeting(counter.incrementAndGet(), String.format(template, name));

}

record Greeting(long id, String content) {}

}

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

We need multistage build because we can leave behind everything we don't want in the final image. It allows us to separate the build process from the run process as well.

Une image contenant texte, Police, blanc, ligne

Description générée automatiquement

## Backend API

On dézippe l’archive simple-api.zip afin d’obtenir le backend de l’application web.

We change the config file (application.ym) in the backend with:

In the url, the first ‘db’ is the ip or name of the container and the second one is the name of the database

datasource:  
 url: jdbc:postgresql://db:5432/db  
 username: usr  
 password: pwd  
 driver-class-name: org.postgresql.Driver

In our browser, we obtain:

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

Description générée automatiquement

# Http server

## Basics

FROM httpd:2.4

COPY ./index.html /usr/local/apache2/htdocs/

docker build -t hberthon/docker\_tp\_http\_server .

docker run -d -p 8888:80 --net=app-network --name=docker\_tp\_httpd hberthon/docker\_tp\_http\_server

docker exec docker\_tp\_httpd cat /usr/local/apache2/conf/httpd.conf

to see in the terminal the conf file

or

docker cp docker\_tp\_httpd:/usr/local/apache2/conf/httpd.conf source.conf in order to copy in your local directory the conf file of your httpd server.

Docker stats to allow us to see the running containers as well as CPUs and MEM usage.

CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM % NET I/O BLOCK I/O PIDS

c66b110a549a docker\_tp\_httpd 0.00% 30MiB / 7.605GiB 0.39% 2.57kB / 668B 0B / 0B 82

c1140c98b8f8 docker\_tp\_backend 0.12% 325.7MiB / 7.605GiB 4.18% 37kB / 34.9kB 0B / 0B 46

522245a75bdc db 0.00% 29.84MiB / 7.605GiB 0.38% 34.4kB / 30.4kB 0B / 0B 17

## Reverse proxy

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

FROM httpd:2.4

COPY ./index.html /usr/local/apache2/htdocs/

COPY ./source.conf /usr/local/apache2/conf/httpd.conf

We build once again:

docker rm -f docker\_tp\_httpd

and now, we can access our backend using our httpd proxy on our frontend

<http://localhost:8888/departments/IRC/students>

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

# Link application

## Docker-compose:

version: '3.7'

services:

    backend:

        build: ./backend

        container\_name: docker\_tp\_backend

        networks:

            - my-network

        depends\_on:

            - database

    database:

        build: ./database

        networks:

            - my-network

        container\_name: db

        volumes:

            - persistence:/var/lib/postgresql/data

    httpd:

        build: ./http\_server

        container\_name: docker\_tp\_httpd

        ports:

            - "8888:80"

        networks:

            - my-network

        depends\_on:

            - backend

networks:

    my-network: {}

volumes:

  persistence:

docker compose up -d.

docker compose logs to see the logs that are not visible because of the -d.

docker compose down to shut down the running containers.

**Question** : 1-3 Document docker-compose most important commands. 1-4 Document your docker-compose file.

# Publish

I started by putting some tag on my images.

docker tag tp-httpd hberthon/tp-httpd:version1.0

docker tag tp-backend hberthon/tp-backend:version1.0

docker tag tp-database hberthon/tp-database:version1.0

I then pushed my images to docker:

docker push hberthon/tp-database:version1.0

docker push hberthon/tp-backend:version1.0

docker push hberthon/tp-httpd:version1.0

**Question:** 1-5 Document your publication commands and published images in dockerhub

# GitHub Actions

**Question**: 2-1 What are testcontainers?

testcontainers are lightweigh libraries for docker containers while testing.

mvn clean verify

## First steps into the CI World

in the actions file:

name: CI devops 2023

on:

#to begin you want to launch this job in main and develop

push:

branches:

- main

- develop

pull\_request:

jobs:

test-backend:

runs-on: ubuntu-22.04

steps:

#checkout your github code using actions/checkout@v2.5.0

- uses: actions/checkout@v4

#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@v3

with:

distribution: 'temurin'

java-version: '17'

cache: 'maven'

#finally build your app with the latest command

- name: Build and test with Maven

run: mvn -B package --file backend/pom.xml

## First steps into the CD World

  build-and-push-docker-image:

    needs: test-backend

    runs-on: ubuntu-22.04

    steps:

      - name: Checkout code

        uses: actions/checkout@v4

  # LOGIN

      - name: Login to Docker Hub

        uses: docker/login-action@v3

        with:

          username: ${{ secrets.DOCKER\_USERNAME }}

          password: ${{ secrets.DOCKER\_SECRET }}

  # BUILD AND PUSH IMAGES TO DOCKER

      - name: Build image and push backend

        uses: docker/build-push-action@v5

        with:

          context: ./backend

          tags:  ${{secrets.DOCKER\_USERNAME}}/tp-backend:v0.1

          push: ${{ github.ref == 'refs/heads/main' }}

      - name: Build image and push database

        uses: docker/build-push-action@v5

        with:

          context: ./database

          tags:  ${{secrets.DOCKER\_USERNAME}}/tp-database:v0.1

          push: ${{ github.ref == 'refs/heads/main' }}

      - name: Build image and push httpd

        uses: docker/build-push-action@v5

        with:

          context: ./http\_server

          tags:  ${{secrets.DOCKER\_USERNAME}}/tp-httpd:v0.1

          push: ${{ github.ref == 'refs/heads/main' }}

# Setup Quality Gate

We need to create an account on solar cloud and I added a sonar token into the secret setting of the git project to access sonar cloud.

We use sonar cloud to analyse staticly our codebase after

name: CI devops 2023

on:

#to begin you want to launch this job in main and develop

push:

branches:

- main

- develop

pull\_request:

jobs:

test-backend:

runs-on: ubuntu-22.04

steps:

#checkout your github code using actions/checkout@v2.5.0

- uses: actions/checkout@v4

#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@v3

with:

distribution: 'temurin'

java-version: '17'

cache: 'maven'

#finally build your app with the latest command

- name: Build and test with Maven

run: mvn -B verify sonar:sonar -Dsonar.projectKey=BerthonHugo\_cours\_docker -Dsonar.organization=berthonhugo -Dsonar.host.url=https://sonarcloud.io -Dsonar.login=${{ secrets.SONAR\_TOKEN }} --file backend/pom.xml

build-and-push-docker-image:

needs: test-backend

runs-on: ubuntu-22.04

steps:

- name: Checkout code

uses: actions/checkout@v4

# LOGIN

- name: Login to Docker Hub

uses: docker/login-action@v3

with:

username: ${{ secrets.DOCKER\_USERNAME }}

password: ${{ secrets.DOCKER\_SECRET }}

# BUILD AND PUSH IMAGES TO DOCKER

- name: Build image and push backend

uses: docker/build-push-action@v5

with:

context: ./backend

tags: ${{secrets.DOCKER\_USERNAME}}/tp-backend:v0.1

push: ${{ github.ref == 'refs/heads/main' }}

- name: Build image and push database

uses: docker/build-push-action@v5

with:

context: ./database

tags: ${{secrets.DOCKER\_USERNAME}}/tp-database:v0.1

push: ${{ github.ref == 'refs/heads/main' }}

- name: Build image and push httpd

uses: docker/build-push-action@v5

with:

context: ./http\_server

tags: ${{secrets.DOCKER\_USERNAME}}/tp-httpd:v0.1

push: ${{ github.ref == 'refs/heads/main' }}

Une image contenant texte, logiciel, Icône d’ordinateur, Système d’exploitation

Description générée automatiquement

After this analysis, we can see that there is two vulnerabilities and 3 hotspots in the codebase.

# Discover Ansible

In order to install ansible:

Apt-get update

Apt-get upgrade

Apt install python3-pip

python3 -m pip install --user pipx

python3 -m pipx ensurepath

apt install python3.10-venv

pipx install --include-deps ansible

Windows does not support permission:

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@ WARNING: UNPROTECTED PRIVATE KEY FILE! @

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

Permissions 0555 for '/mnt/c/Users/hugob/Downloads/id\_rsa' are too open.

It is required that your private key files are NOT accessible by others.

This private key will be ignored.

In order for the permission and ssh to work, run:

cp /mnt/c/Users/hugob/Downloads/id\_rsa .

This will copy the rsa key into the wsl ‘ubuntu distribution’.

Run:

chmod 400 id\_rsa

ssh -i id\_rsa centos@hugo.berthon.takima.cloud

we are now login.

## Setup an Apache Server

To set an Apache server with ansible, run:

ansible all -m yum -a "name=httpd state=present" --private-key=id\_rsa -u centos –become

ansible all -m shell -a 'echo "<html><h1>Hello World</h1></html>" >> /var/www/html/index.html' --private-key=id\_rsa -u centos –become

ansible all -m service -a "name=httpd state=started" --private-key=id\_rsa -u centos --become

## Inventories

To create an inventory, run:

mkdir -p ansible/inventories

cd ansible

nano inventories/setup.yml

and add to the setup.yml file the following code:

all:

vars:

ansible\_user: centos

ansible\_ssh\_private\_key\_file: ~/id\_rsa

children:

prod:

hosts: hugo.berthon.takima.cloud

Test your inventory with the ping command:

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

hugo.berthon.takima.cloud | SUCCESS => {

"ansible\_facts": {

"discovered\_interpreter\_python": "/usr/bin/python"

},

"changed": false,

"ping": "pong"

}

Question : 3-1 Document your inventory and base commands

In the ansible directory, run the following command to create a playbook for the different tasks and rules:

nano playbook.yml

in this file, we write our first task:

- hosts: all

gather\_facts: false

become: true

tasks:

- name: Test connection

ping:

to test, we run:

ansible-playbook -i inventories/setup.yml playbook.yml

## Advanced Playbook

To create a playbook to install docker on the server, run:

nano docker-playbook.yml

and add the following code to the playbook:

- hosts: all

gather\_facts: false

become: true

# Install Docker

tasks:

- 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

# Deploy your app

We create our roles using the command:

ansible-galaxy init roles/<name>

in playbook.yml:

---

- hosts: all

gather\_facts: false

become: true

tasks:

- name: Test connection

ping:

roles:

- docker

- network

- volume

- launch-database

- launch-app

- launch-proxy

in our

---

# tasks file for roles/launch-proxy

- name: Run HTTPD

docker\_container:

name: httpd

image: hberthon/tp-httpd

networks:

- name: network

ports:

- "80:80"

---

# tasks file for roles/launch-app

- name: Create a backend container

community.docker.docker\_container:

name: docker\_tp\_backend

image: hberthon/tp-backend

networks:

- name: network

---

# tasks file for volume

- name: Create a volume

docker\_volume:

name: persistence

---

# tasks file for roles/launch-database

- name: Create a database container

community.docker.docker\_container:

name: db

image: hberthon/tp-database

volumes:

- persistance:/var/lib/postgresql/data

networks:

- name: network

---

# tasks file for roles/network

- name: Create a network

community.docker.docker\_network:

name: network

# Front

## Continuous Deployment