Report of TP Part 01 Docker

Author: Jiaen LIU

Database container setup

Create a folder (database) which contains the Dockerfile for this part and enter this folder (database)

```
# The content of the Docker file
FROM postgres:14.1-alpine

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

```
#Create the network and volume we will use later
docker volume create my-vol
docker network create app-network
```

```
# Under the folder database
# Using the following commands to build the docker image and run the docker container.
# Build the docker image for the database
docker build -t johojakusha/tp_database .
# Create the docker container from the image file we just created.
# Setting the expose port to 5433, name to postgresql, inside the app-network network
docker run -p 5433:5432 --name postgresql --net app-network -v my-
vol:/var/lib/postgresql/data  johojakusha/tp_postgres

```

```
# run the adminer container to have an interface
docker run \
    -p "8090:8080" \
    --net=app-network \
    --name=adminer \
    -d \
    adminer
```

Create file **01-CreateScheme.sql** & **02-InsertData.sql** and modify the Dockerfile to import the data to the container

```
# The content of the Docker file
FROM postgres:14.1-alpine

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

# copy the init script
COPY CreateScheme.sql /docker-entrypoint-initdb.d
COPY InsertData.sql /docker-entrypoint-initdb.d
```

Then recreate the image file and run the container again.

```
docker build -t johojakusha/tp_database .
docker run -p 5433:5432 --name postgresql --net app-network -v my-
vol:/var/lib/postgresql/data  johojakusha/tp_postgres
```

Question 1-1

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

Backend API

Basisc

Use the docker container to run the java program to print the hello world

Create the Main.java

```
public class Main {

   public static void main(String[] args) {
       System.out.println("Hello World!");
   }
}
```

Then compile it by javac Main.java

Create the docker file for this app

```
FROM openjdk:17

COPY Main.class /Main.class
# run the application
CMD ["java", "Main"]
```

Then lunch the app

```
# Build the docker image
docker build -t johojakusha/tp_java .
# Build the run the docker container
docker run -t johojakusha/tp_java
```

Backend simple api (Multistage build)

Create the Springboot application on Spring Initializer.

Use the following config:

- Project: Maven
- Language: Java 17
- Spring Boot: 2.7.5
- Packaging: Jar
- Dependencies: **Spring Web**

Generate the project and give it a simple GreetingController class:

```java
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));
   }

   class Greeting {

       private final long id;
       private final String content;

       public Greeting(long id, String content) {
           this.id = id;
```

```
            this.content = content;
        }

        public long getId() {
            return id;
        }

        public String getContent() {
            return content;
        }
    }
}
```

Create the dockerfile inside the folder demo(the folder of created project)

```
# 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
```

Then run the following commends to build the image file and container to run the simple api

```
docker build -t johojakusha/tp_multi .
docker run  -p 8080:8080 -t johojakusha/tp_multi
```

Question 1-2

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

Answer:

With the multistage build, we can just keep the unnecessary parts to run the app. For example, in a java application, when we do one build, it will contain the parts that compile the java and the runtime. If we use the multistage, we can use one container to build the app, then put it inside of the second container with the minium runtime components.

Backend API

Download the backend api and unzip it.

Adjust teh configuration in simple-api/src/main/resources/application.yml

```
spring:
  jpa:
    properties:
      hibernate:
        jdbc:
          lob:
            non_contextual_creation: true
    generate-ddl: false
    open-in-view: true
  datasource:
  # There needs a jdbc url
    url: jdbc:postgresql://postgres:5432/db # I set the hostname of the postgres database
to postgres.
    username: usr # the username of the database
    password: pwd # the password of the user
    driver-class-name: org.postgresql.Driver
management:
 server:
   add-application-context-header: false
 endpoints:
   web:
     exposure:
       include: health,info,env,metrics,beans,configprops
```

For the docker file, it is the same as previous one.

Run the following commands to build the image and run the backend in docker container

```
docker build -t johojakusha/simple-api .
docker run -p 8080:8080 -t johojakusha/simple-api
```

Http server

Basics

Choose an appropriate base image

Create a folder and then inside this folder create a simple landing page: index.html and put it inside the container.

Then create the dockerfile and write with content below.

```
FROM httpd:2.4
COPY index.html /usr/local/apache2/htdocs/
```

Then we create the docker image and run it.

```
docker build -t johojakusha/tp_http_server
docker run -p 8080:80 -t --name http_server johojakusha/tp_http_server
```

Now I can execute some check commands to see whether everything is running properly.

```
docker stats
docker inspect http_server
docker logs http_server
```

Now we need change some configuration inside our container.

```
# Use docker cp to get the httpd.conf file
docker cp . http_server:/usr/local/apache2/conf/httpd.conf
```

Then modify the httpd.conf to set the reverse proxy.

```
# Uncomment these four lines
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_connect_module modules/mod_proxy_connect.so
LoadModule proxy_ftp_module modules/mod_proxy_ftp.so
LoadModule proxy_http_module modules/mod_proxy_http.so
# Add the configuration here
<VirtualHost *:80>
ProxyPreserveHost On
ProxyPass / http://docker_tp-backend-1:8080/ # docker_tp-backend-1 is the name of my
backend docker container.
ProxyPassReverse / http://docker_tp-backend-1:8080/
</VirtualHost>
```

After that, we need to put this httpd.conf back to the container. For the convenience, I modify the dockerfile to automatically add the configuration.

```
FROM httpd:2.4
COPY index.html /usr/local/apache2/htdocs/
# Add the modified configuration file back to the container.
COPY httpd.conf /usr/local/apache2/conf/httpd.conf
```

Then run the following commands to recreate the docker image and run it.

```
docker build -t johojakusha/tp_http_server
docker run -p 8080:80 -t --name http_server johojakusha/tp_http_server
```

Question:

Why do we need a reverse proxy?

We do not want to just expose our backend service directly to the users. We want an front-end to interact with the users and with that structure, it is more convenient for the scalability. For example, we can use an front-end to separate the load equally to the backend services.

Link application

Docker-compose

we want to make the above three things work together. Docker-compose can help us to do that.

Create a docker-compose.yml file with the following contents

```
# The docker-compose.yml file for the simple-api-student project
# This file is used to configure the services that make up the project.
# For more information on the Compose file, see https://docs.docker.com/compose/compose-
file/
# For more information on Docker Compose, see https://docs.docker.com/compose/


# The version of the Compose file format
version: '3.7'

# The services that make up the project
services:
    # The backend service
    backend:
        # The build context for the backend service
        build:
          # The location of the Dockerfile for the backend service
          context: ./simple-api-student-main
        # The networks that the backend service will be connected to
        networks:
          ["my-network"]
        # Only after the database service is ready, the backend service will start
        depends_on:
          ["database"]

    # The database service
    database:
        # The build context for the database service
        build:
          # The location of the Dockerfile for the database service
          context: ./postgresql/
        # The networks that the database service will be connected to
        networks:
          ["my-network"]
        # The hostname of the database service
        hostname: postgres

    # The http_server service
    httpd:
        # The build context for the http_server service
        build:
          # The location of the Dockerfile for the http_server service
          context: ./http_server/
        # The ports that the http_server service will expose
        ports:
         ["80:80"]
        # The networks that the http_server service will be connected to
        networks:
          ["my-network"]
        # Only after the backend service is ready, the http_server service will start
        depends_on:
          ["backend"]

# The networks that will be created
networks:
    # The network will be created with the name "my-network"
    my-network:
```

The docker-compose will handle the three containers and a network for us.

Run the following command and then you can access the API on localhost:80

```
docker-compose up
# when you need to rebuild your docker run this one.
docker compose up --build --force-recreate
```

Questions:

1-3 Document docker-compose most important commands.

1-4 Document your docker-compose file.

Publish

Create a Docker Hub account

1- Connect the docker hub account by

```
docker login
```

2- Tag your image. For now, all the images I created only using the latest tag, now, for the publish, it is important to add a meaningful version information to images.

```
# Just an example for this part
# tag means to tag the image file from source to target
docker tag johojakusha/tp_java johojakusha/tp_java:1.0
```

3- Then push this tagged image to docker hub:

```
# This document will push the tagged images to the docker hub
docker push johojakusha/tp_java
```

The reason why put images into an online repo is that we can access it when we need it.