Weather Webapp (Containerization Assignment) / Atypon Java and Devops Bootcamp / Aug - 2022

Yazan Istatiyeh

Contents

1	Instructions to use	2
2	Introduction	2
3	Containers Diagram	2
4	Data Entry Service	3
5	Analytics Service	4
6	Show Results Service	4
7	Authentication Service	5
8	Connection (Between The Show/Entry Services And Authentication Service)	5
9	MySQL Service	6
10	Docker Files (All Services Except MySQL)	6
11	Docker Compose File	7
12	Used Dependencies	10

INSTRUCTIONS TO USE

Run the Docker Compose file with the following command: ***docker-compose up -d***, the app should run in detached mode.

Afterwards, you can head to localhost:8080/ or localhost:8080/home to start using the app.

The app should make 5 containers and it takes around 20 seconds to run if the MySQL image was already installed on the machine.

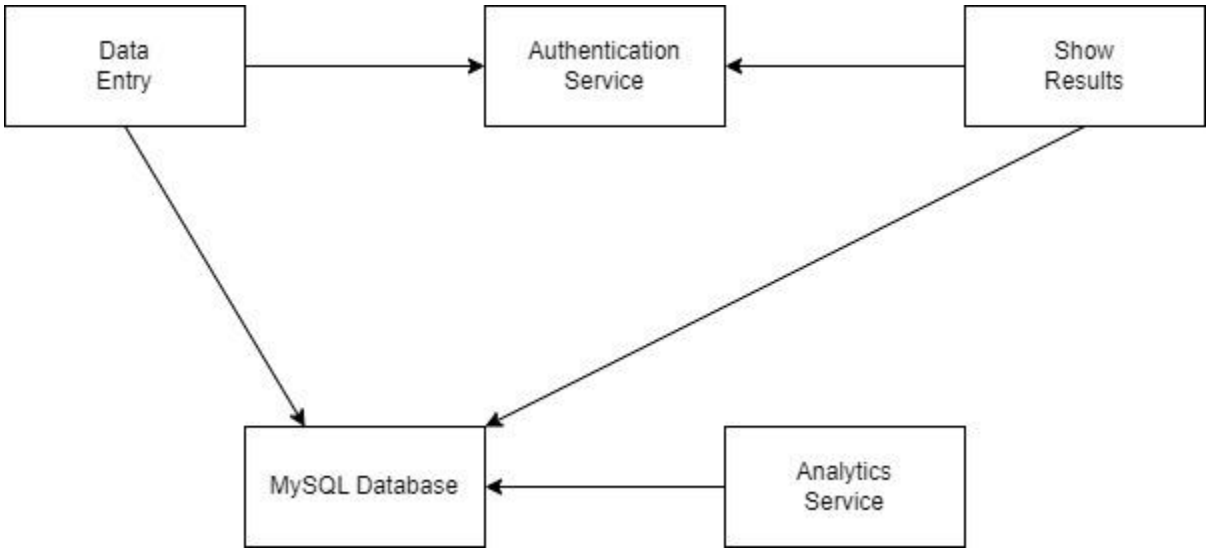
INTRODUCTION

The task was to build a group of containerized microservices to collect data and analyze them, containerization will be done using Docker.

There are many services done in the project, each one will be discussed by itself.

All apps and services were done using Spring Boot.

CONTAINERS DIAGRAM



DATA ENTRY SERVICE

This program's task is to let the user enter new data to the database, after authenticating the username and password inputted. In my app, there's a home screen (index page) this app is responsible for, it has choices to direct the user to enter new data, view all data, or view analyzed data, for the last two choices, it redirects the user to another host which the show results app is responsible for.

If the user chose to enter new data, the app asks the user to enter his/her credentials, if the entered credentials were incorrect, it states that with a message and lets the user either enter the credentials again or go back to the index page.

After entering correct credentials, the user can enter the data he/she would like to enter to the database, a data, morning temperature, and night temperature. If the entered day already existed in the database, it will state that with a message and let the user enter more data or return to the index page.

This app is responsible for making the database if it did not already exist, also, it is responsible for making the table of entries if it did not exist.

This app is connected directly to the database, so it can insert values into it using the Spring JDBC template.

Also, this app is connected to the authentication service, so it sends the user's credentials to authenticate them, and it receives the authentication status from the authentication service.

ANALYTICS SERVICE

This service reads data from the database, makes some simple analysis on it, like getting the minimum temperature, the maximum temperature, and the average temperature for different times through the day (Morning time, night time, and all-day), and inserts the analyzed data to a new table (which did not already exist, but this app is responsible for making) called analyzed_data.

This service is only connected to the database with the Spring JDBC template so it can insert and retrieve data into it and from it.

This service runs automatically again every 5 seconds, it reads the new data from the original table, clears the analyzed data table, and inserts new analyzed data into it. This was done by enabling scheduling for this app, and using the *EnableScheduling* annotation from Spring in the main class, and *Scheduled(fixedRate = 10000)* annotation in the *postAnalysis* method.

SHOW RESULTS SERVICE

This service is used to display all the data in the database, it can show the whole data or the analyzed data. But the user has to be authenticated first.

Like the data entry app, this app asks the user to enter his/her credentials before viewing the data. These credentials are sent to the authentication service to authenticate them, the authentication will return the authentication status to the current app, if the credentials were authenticated, it allows the user to view the data, otherwise, it indicates that with a message and lets them go back to the home page (redirect to the entry service) or enter the credentials again.

The data will be organized in HTML pages in tables.

Also, this app is connected to the database with Spring JDBC template, so it can retrieve data from the database, it has access to both the original data table, and the analyzed data table.

AUTHENTICATION SERVICE

This is a simple service to authenticate a given user and send a JSON file with the authentication status, so other apps can know if the entered credentials are valid or not.

This service is a very simple service that checks if the given username is “root” and the given password is also “root”, this service can be simply upgraded and connected to a database with a table that contains the usernames and passwords of users and check if the given credentials are valid. This is not the most “secured” solution, but it works.

CONNECTION (BETWEEN THE SHOW/ENTRY SERVICES AND AUTHENTICATION SERVICE)

This connection works by sending user’s credentials from the show and entry services to the authentication service using the RestTemplate and converting the user’s object to a JSON file so we can send it in HttpEntity. Also, the authentication service returns the authentication status after checking the user’s username and password in a JSON file that either contains “true” or “false”.

Also, after a user finishes entering or showing the data and return to the index page, it resets his/her authentication status to false, so if he/she wants to enter or show data again he/she has to enter his/her credentials and authenticate it again.

MySQL SERVICE

I will list the made tables in the MySQL database:

- ***weather_entries***: This table is made by the data entry app, it's attributes are:
  - **date**: An attribute of type date and is the primary key (so no days can be duplicated).
  - **morning_temp**: An attribute of type double.
  - **night_temp**: An attribute of type double.
- ***analyzed_data***: This table is made by the analytics app, it's attribute are:
  - **daytime**: An attribute of type varchar, indicates the time the analysis was done on (morning, night, allday, etc..) and it is the primary key. Also it can be changed to be of type time, so we can track data in each hour.
  - **lowest_temp**: An attribute of type double (the lowest found entry).
  - **avg_temp**: An attribute of type double (the average of all entries).
  - **highest_temp**: An attribute of type double (the highest found entry).

DOCKER FILES (ALL SERVICES EXCEPT MYSQL)

The docker files in general are files that build an image of an application automatically by reading some commands from ***Dockerfile***. This file in every app will contain all the commands we need to build an image from the app.

In every dockerfile for every app, we specify a parent image which our apps are based on, and in our situation it is ***openjdk:8***. That is done with the ***FROM*** command. Also we specify the directory with the .jar file (which is produced by maven in my project) to be moved into the image we are building, then we specify the port we are running our app on with the ***EXPOSE*** command. Finally, we use the ***ENTRYPOINT*** command to specify executable files that will always run when we make a container out of the built image.

DOCKER COMPOSE FILE

The Docker compose file is a ***YAML*** file that includes the needed services, networks, and other stuff needed to build a system.

I will discuss the services initiated in my Docker compose file, but first I will list the networks:

- **entry-mysql**: This network is used to connect the data entry app with the MySQL database so the entry app can insert data into the database.
- **auth-entry:** This network is used to connect the entry app with the authentication service, so the entry app can authenticate users' credentials when entered.
- **analy-mysql:** This network is used to connect the analysis service with the database so it can read the original data from it, and insert analyzed data into it.
- **show-auth:** This network is used to connect the show results app with the authentication service so the show app can authenticate users' credentials when entered.
- **show-mysql:** This network is used to connect the MySQL database with the show results app, so the app can read analyzed data and original entries from the database.

There are five services initiated in the Docker compose file:

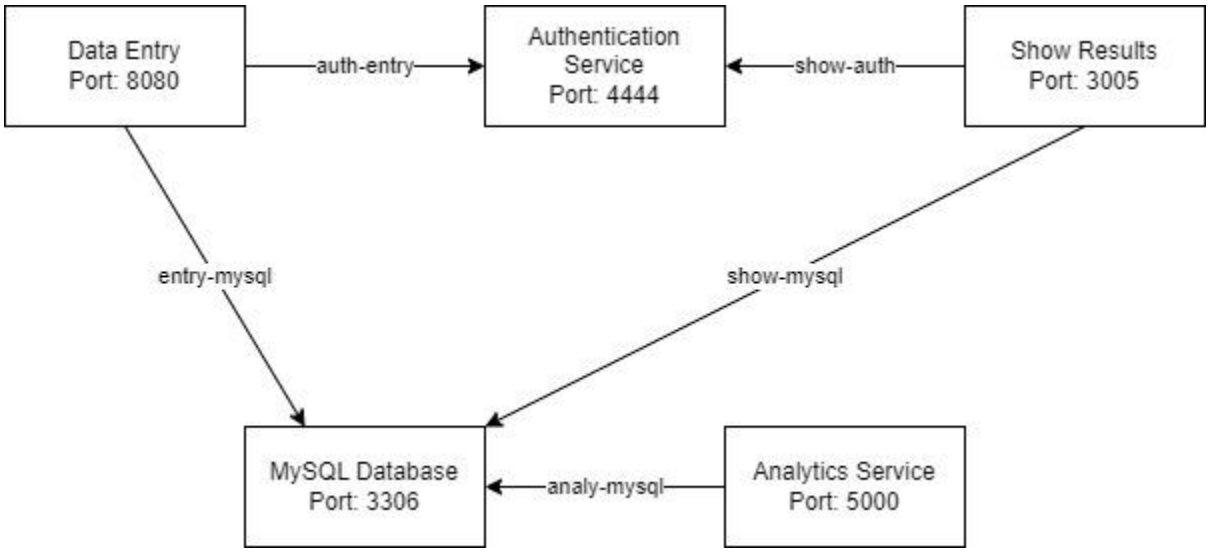
- **entryapp**: This service makes a container from the data entry app (builds it), and runs it on the port 8080, also it initializes the environment variables for it, so it specifies the URL to the database, and the password and username used in it. Also it lets the app create a database if it did not exist. In addition to giving him the permission to write new tables in the database. This container does not run immediately after being ready to run, instead, it waits for the ***mysqldb*** container to be healthy so it runs without any problems. The health check for the ***mysqldb*** container will be discussed next.
- **mysqldb**: This service makes a container from the basic MySQL image from Docker Hub, and runs on port 3306. It initializes environment variables for it too (the username). This service has a health check, which is a command that runs inside the container and the container's health is decided depending on this command result. In this container we run the command *curl* to check if the localhost (URL) this service is running on is active, if so, then the service is healthy and other services depending on it can run safely.

- **authenticationservice**: This service builds a container from the authentication service and runs it on port number 4444, this service can run anytime by itself because it does not depend on any other services, and only acts when another service interacts with it.
- **analyticsservice**: This service builds a container from the analytics service and runs it on port number 5000, and because this app is connected to the database, then we have to specify the database details in the environment variables, so we initialize the database URL, database username and password. Also, we gave the app the ability to create a table (the table of analyzed data).
- **show_results**: This service builds a container from the show results app and runs it on port number 3005, this service is also connected to the database so we initialized the environment variables with the database URL, username, and password. Also, this service cannot be ran before the analytics service, because the analytics service is responsible for creating the analyzed data table, and this service reads from that table and assumes that the table already exists.

The order of services running:

1. MySQL service (Almost all other services depend on it).
2. Data Entry Service (It is responsible for making the database and the main table).
3. Analytics Service (It is responsible for making the second table on the database).
4. Show Results App (It should run both Data Entry Service and Analytics Service because it reads from tables they are responsible for making)
5. Authentication Service (It does not matter when this service runs, because it is a light service and will be ready before really fast).

Another diagram with networks and ports' numbers:



USED DEPENDENCIES

I used many dependencies in my apps and will talk about some of them:

- Dependencies for databases:
  - Spring-boot-starter-data-jpa
  - Mysql-connector-java
  - Spring-boot-starter-data-jdbc

    The previous dependencies made some libraries available and made it easier to connect to the database, insert data into it, and retrieve data from it.

- Dependencies used for appearance (HTML pages):
  - Thymeleaf-layout-dialect
  - spring-boot-starter-thymeleaf

    These dependencies were used to build the interface for the system and being able to get input from HTML pages.

- Dependency for networking:
  - Json

    JSON dependency was used to be able to convert an object into a JSON file easily and send it to another service.

Thanks!