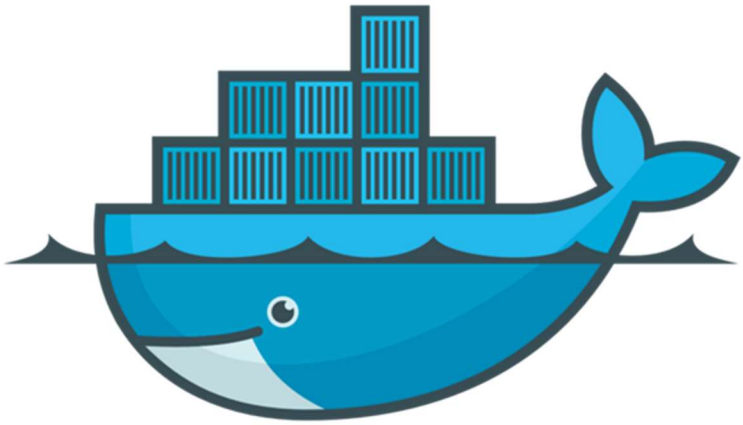
Car Rental Web app
- Containerization Project -

Atypon SDE Internship



docker

Yazan Yousef

Table Of Contents

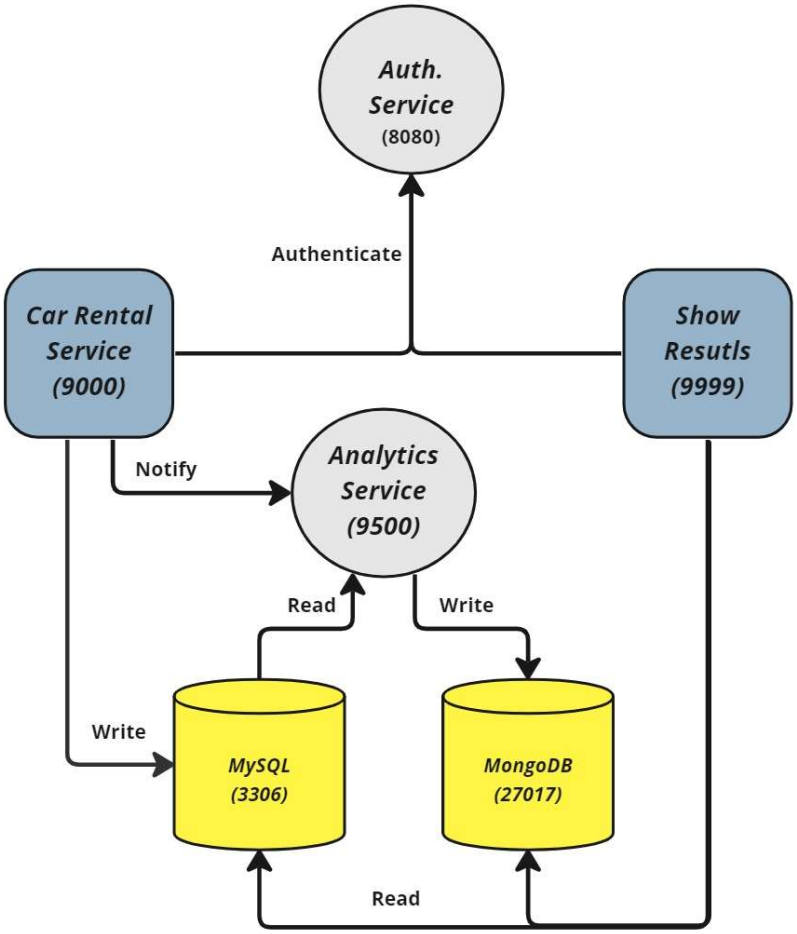
1 Introduction

This project focuses on using Docker to build a web application that collects data, stores it in a database, and performs analysis on the data. The idea is to create a simplified version of a **car rental web application** that utilizes six Docker containers running together as microservices.

The microservices architecture also allows for flexibility and easier maintenance of the application. The following sections will describe each of the microservices in more detail and how they work together to create the car rental application.

2 Microservices

The microservices makes the whole process a lot easier to maintain and scale, and also to extend, and we have 6 microservices each one is a Docker container.



For all services that I built, **Spring Boot** were used.

2.1 MySQL and MongoDB Containers

The MySQL container will store all the data related to car rental, while the MongoDB container will have the analytics stored inside it.

2.2 Authentication Service

This microservice will handle user authentication to ensure that only authorized personnel, such as admin or business owners of the car rental company, can access the system. It will be utilized by both the car rental and show results web applications.

2.3 Analytics Service

This microservice will analyze the data stored in the MySQL database and write the results to the MongoDB database. The show results web app will read the analyzed data from the MongoDB database, while the car rental service will notify the analytics service whenever a new car rental information is added. It's important to note that the analytics service will only update its data when **new information is added**, ensuring that the data is always **accurate** and **up to date**.

2.4 Car Rental Service

This web application will allow admins or business owners of the car rental company to enter data related to car rentals, such as the **customer's SSN**, **name**, and the **rented car model**. To ensure security,

users must pass through the authentication service before accessing the data entry form.

The entered data will be stored in the MySQL database, and the service will send a post request to the analytics service to trigger the analysis of the data. Once the analysis is complete, the results will be written to the MongoDB database.

The connection to the MySQL database was established using the **Spring Data JPA** framework, which provides an easy-to-use API for accessing relational databases.

2.5 Show Results Service

This web application is designed to provide admins or business owners of the car rental company with access to the analytical data stored in the MongoDB database. Users must pass through the authentication service to view the results. Additionally, this web app can also display all the data in the MySQL database, allowing users to view detailed information about car rentals.

I used HTML tables with some CSS styling to make it visually appealing.

The connection to the MongoDB database was established using **MongoTemplate**, a Java-based template that simplifies the interaction with MongoDB

3 Running the Project

Running the project is easy just clone the Git repository cd into it and run this command where **docker-compose** file is located:
docker-compose up –d

To access the **car rental web app** go to this link:

http://localhost:9000/login

And the **show results web app** on this link:

http://localhost:9999/login

The admin credentials:

- Email : admin1@gamil.com
- Pass : 123456

4 Docker Files

All the services have a Docker file except for the MySQL and MongoDb.

As all services were built using spring boot and maven, this docker file were shared among them:

```
FROM openjdk:17
WORKDIR /app
COPY target/<jar-file-name>.jar /app
EXPOSE <the-port>
CMD ["java", "-jar", "<path-to-the-jar>.jar"]
```

This Dockerfile is based on the OpenJDK 17 image and copies the executable jar file, located in the target directory, into the /app directory in the container. The **<jar-file-name>** and **<path-to-the-jar>** are specific to each service. Finally, the container's port is exposed and the command to run the application is specified which will be the running the jar file.

5 Docker-compose file

The docker-compose file is used to define and run multiple containers as a single service. In this project, the docker-compose file defines 6 services: MySQL, MongoDB, Authentication Service, Analytics Service, Car Rental Service, and Show Results Service.

Each service is defined using the **services** section, and each one has its own set of options.

- The **mysql** service is based on the **mysql** image, which provides a pre-built MySQL database server. It is given a name **mysql** for easy identification as well as that I will always refer to the database base as **mysql** for example when connecting to it in the properties file.

  And it is set to always restart in case of any issues. It is also configured with an environment variable **MYSQL_ROOT_PASSWORD** with a value of **123456**, which is used to set the root password for the MySQL database. The ports section maps the container's port 3306 to the host machine's port 3306, allowing connections to the MySQL database from outside the container.

- The **mongo** service is based on the **mongo** image, which provides a pre-built MongoDB server. It is given a name **mongo** for easy identification, and is set to always restart in case of any issues. The ports section maps the container's port 27017 to the host machine's port 27017, allowing connections to the MongoDB from outside the container.

- The **auth** service is built from the Dockerfile located in the **./AuthenticationService** directory, which defines the container's configuration. It is given a name **auth** for easy identification and I will access it using **auth** instead of localhost as it will go to different container from the other services, and is set to always restart in case of any issues. The ports section maps the container's port 8080 to the host machine's port 8080, allowing connections to the Authentication Service from outside the container. It also depends on the **mysql** service, which means that the **mysql** service must be started before the **auth** service.

- The **analytics** service is built from the Dockerfile located in the **./AnalyticsService** directory, which defines the container's configuration. It is given a name **analytics** for easy identification, and is set to always restart in case of any issues. The ports section maps the container's port 9500 to the host machine's port 9500, allowing connections to the Analytics Service from outside the container. It also depends on the **mysql** and **mongo** services, which means that both services must be started before the **analytics** service.

- The **car-rental** service is built from the Dockerfile located in the **./CarRentalService** directory, which defines the container's

configuration. It is given a name **car-rental** for easy identification, and is set to always restart in case of any issues. The ports section maps the container's port 9000 to the host machine's port 9000, allowing connections to the Car Rental Service from outside the container. It also depends on the **mysql**, **auth**, and **analytics** services, which means that all three services must be started before the **car-rental** service.

- The **show** service is built from the Dockerfile located in the **./ShowResults** directory, which defines the container's configuration. It is given a name **show** for easy identification, and is set to always restart in case of any issues. The ports section maps the container's port 9999 to the host machine's port 9999, allowing connections to the Show Results Service from outside the container. It also depends on the **auth**, **mysql**, and **mongo** services, which means that all three services must be started before the **show** service.

The order of running the Services:

1. MySQL container
2. MongoDB container
3. Authentication Service
4. Analytics Service
5. Car Rental Service
6. Show Results Service