Containerization (Report)

Name: Akram Jaghoub

Table of Contents

1.0 Introduction

System Overview

The "VideoGame Web Microservices" system is a web application built on microservices. It's like a puzzle, with Spring handling the backend magic and Thymeleaf making sure things look pretty.

Benefits of Dockerization

Dockerization revolutionizes deployment by packaging the application and its dependencies into portable containers. This approach offers:

Consistency: Ensures applications run consistently across different environments.

Efficiency: Optimizes resource utilization by sharing the host OS kernel.

Scalability: Enables easy scaling of components as demand changes.

Speed: Accelerates deployment through lightweight and reproducible containers.

Microservices

Microservices Our system is made up of four key microservices that we're going to build:

1. Enter Video Game
2. Authentication Service
3. Video Game Analytics
4. Show Video Game Analytics

And for our data storage needs, we'll use MySQL as our trusty database. Now, let's dive into each of these services and how we put them together. After that, I'll explain how I connected all these microservices with the database using Docker and Docker Compose.

2.0 Microservices

2.1 Authentication Service

The video game system uses the Authentication Service to make sure only the right people can use it. This service uses special codes called JWT tokens to check if someone is allowed in.

When someone wants to log in, they use the Login Endpoint (/auth/login). Here, they give their username and password. If they're right, the service gives them a special token. This token lasts for 30 minutes. Otherwise, the system would redirect users to the login page with an error message indicating “wrong credentials”.

Whenever users wish to access features within the system, they must utilize the token they were given at login. Every system request is accompanied by this token, requiring the system to validate its authenticity. The Validate Token Endpoint (/auth/validate-token) manages this task. If the system deems the token both valid and unexpired (within its 30-minute window), users are granted access. However, should the token be found invalid or expired, access is promptly denied.

To ensure this verification is consistently applied to every request, and to optimize system efficiency, interceptors are implemented. Within the framework of this video game system, one can liken the interceptor to a watchful security officer. This officer ceaselessly ensures that users present the correct "pass" before granting system access.
For safety, we should turn passwords into secret codes before saving them. We also have a "secret key" that helps make and check tokens. This key is very important, so we have to keep it safe. Also, because tokens only last 30 minutes, they can't be misused for long.

In simple terms, the Authentication Service is like a door guard. It checks if people have the right token before letting them use our game system.

2.2 Enter Video Game

The "Enter Video Game" microservice is a crucial component that seamlessly combines authentication and video game submission processes to offer users a smooth experience while upholding robust security measures.

Authentication Integration: The microservice integrates an authentication interceptor, building on principles from the "Authentication Service" section. This guarantees that only authorized users can access the feature.

Authentication and Video Game Submission Flow: Upon initiating access to the service, the authentication interceptor validates the user's authentication status. Successful authentication grants access to the video game form submission feature, while unsuccessful attempts redirect users to the login page, maintaining system integrity.

The core of the service relies on the VideoGameController class, which manages video game submissions through two key endpoints:

1. Show Video Game Form Endpoint (/addVideoGame): This endpoint displays the video game entry form, providing administrators and store owners a platform to add new games. The form initializes a VideoGame object for data input.

2. Add Video Game Endpoint (/addVideoGame): After form submission, collected video game details undergo processing. The VideoGameService adds the new game to the system, while the AnalyticsService triggers an HTTP POST request to notify the system of the addition to keep it well-informed and always up to date. A concise success message confirms the submission.

2.3 Analytics Service

The Analytics Microservice role in the "VideoGame Web Microservices" system is as follows, it meticulously analyzes data sourced from the MySQL database and seamlessly stores outcomes in MongoDB. The derived metrics encompass **maxPrice, minPrice, avgPrice, numberOfVideoGames, and lastUpdated**. These metrics collectively furnish a comprehensive overview of the video game data. Collaborating closely with the "Add Video Game" service, updates are selectively triggered only with the arrival of new data, thus preserving data accuracy, and keeping the system up to date.

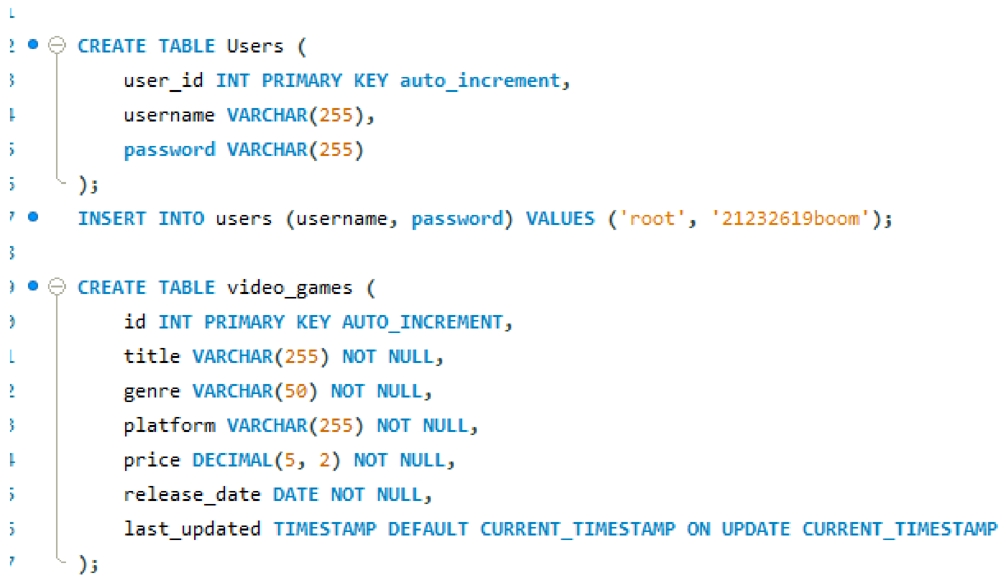
2.4 Show Analytic Results

The "Show Analytic Results" function operates similarly to "Add Video Game," relying on the Authentication Service for secure user access using JWT. The notable difference is that upon successful authentication, users are directed to the analytics view directly, eliminating unnecessary steps. This seamless transition provides users with valuable insights that are read from the MongoDB like **maxPrice, minPrice, avgPrice, numberOfVideoGames, and lastUpdated,** by using the existing authentication setup, this feature ensures consistency, simplifies user interaction, and maintains the overall system's reliability within the "VideoGame Web Microservices" environment.

3.0 Database

- MySQL

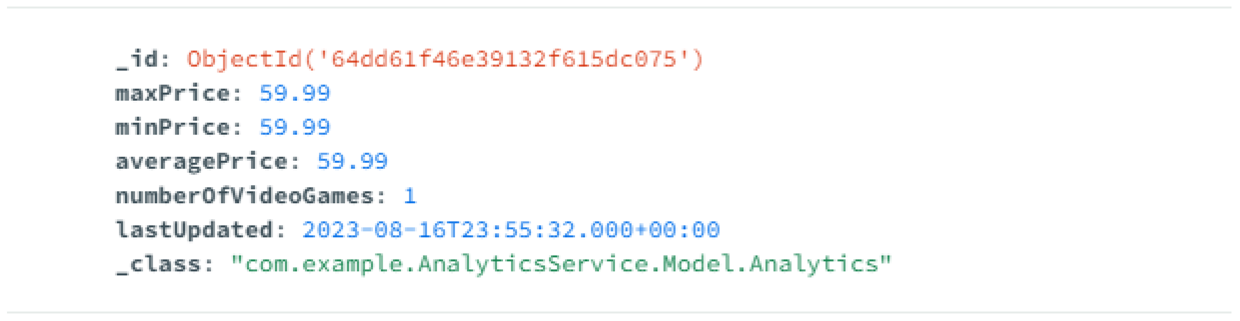
Within the database, we have two tables. The first one is dedicated to users(admins), who will access the system. This table includes the following attributes: username and password. The second table is meant for video games and contains attributes such as title, genre, platform, price, release date, and last update date. The "last update" attribute indicates the most recent modification date of the data.



```sql
CREATE TABLE Users (
    user_id INT PRIMARY KEY auto_increment,
    username VARCHAR(255),
    password VARCHAR(255)
);
INSERT INTO users (username, password) VALUES ('root', '21232619boom');

CREATE TABLE video_games (
    id INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    genre VARCHAR(50) NOT NULL,
    platform VARCHAR(255) NOT NULL,
    price DECIMAL(5, 2) NOT NULL,
    release_date DATE NOT NULL,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

- MongoDB



```
_id: ObjectId('64dd61f46e39132f615dc075')
maxPrice: 59.99
minPrice: 59.99
averagePrice: 59.99
numberOfVideoGames: 1
lastUpdated: 2023-08-16T23:55:32.000+00:00
_class: "com.example.AnalyticsService.Model.Analytics"
```

4.0 Docker and Docker Compose

We will rely on Docker to create each microservice independently. Initially, each microservice is accompanied by a JAR file, built using Maven packaging, and a corresponding Docker image that is launched using "docker compose up." Each service is equipped with a Dockerfile containing the following steps:

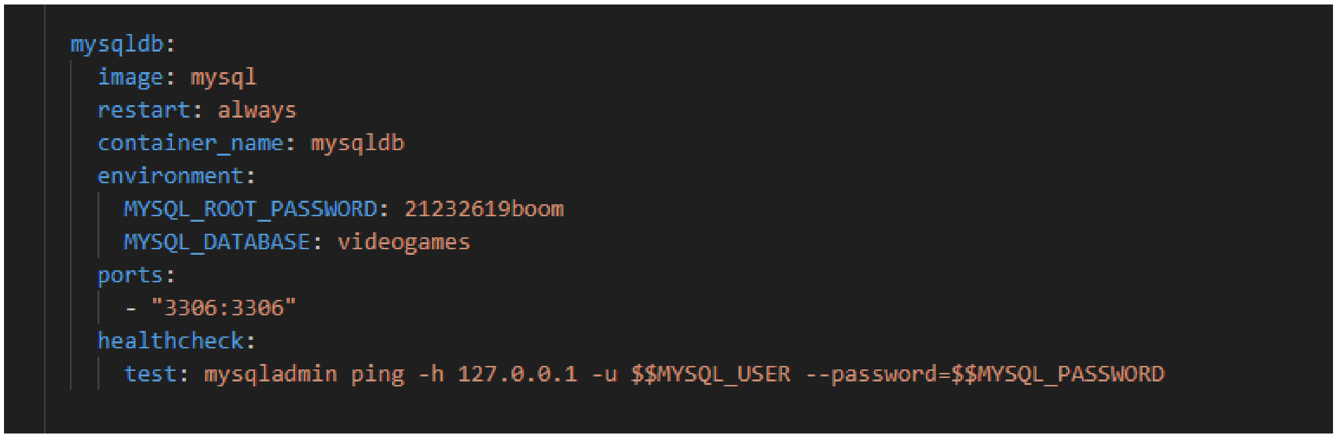
FROM "java jdk"

ADD "jar package location obtained from running Maven."

CMD "commands to execute the package (jar file) upon container startup."

In the Docker Compose configuration, we will define all desired services, including the databases (MySQL and MongoDB) as follows:

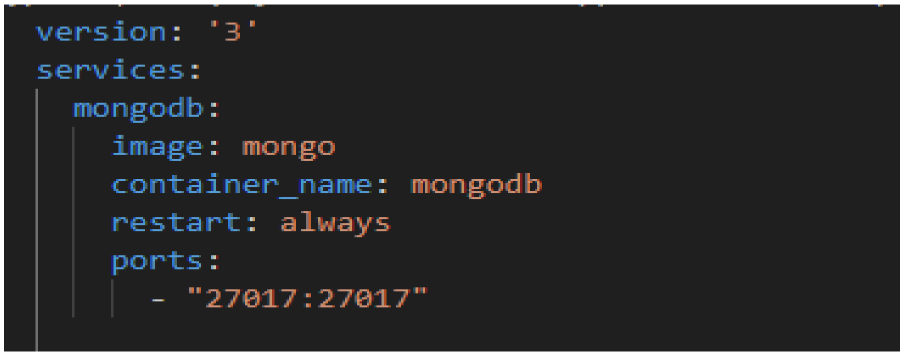
- MySQL Service:
  In this configuration, we define crucial details within the Docker Compose file, such as the service name, container name, image, root password, and the root database name for MySQL. Additionally, we establish a mapping for the MySQL port within the container, binding it to port 3306 externally. To address a potential concern where the database might not be operational when the Docker Compose starts, we employ a health check mechanism. This strategy involves a test that determines the availability of the database. This approach ensures that any dependent containers will wait until the health check successfully confirms the database's readiness before proceeding, thus enhancing the reliability and stability of the overall deployment process.



```yaml
mysqldb:
  image: mysql
  restart: always
  container_name: mysqldb
  environment:
    MYSQL_ROOT_PASSWORD: 21232619boom
    MYSQL_DATABASE: videogames
  ports:
    - "3306:3306"
  healthcheck:
    test: mysqladmin ping -h 127.0.0.1 -u $$MYSQL_USER --password=$$MYSQL_PASSWORD
```

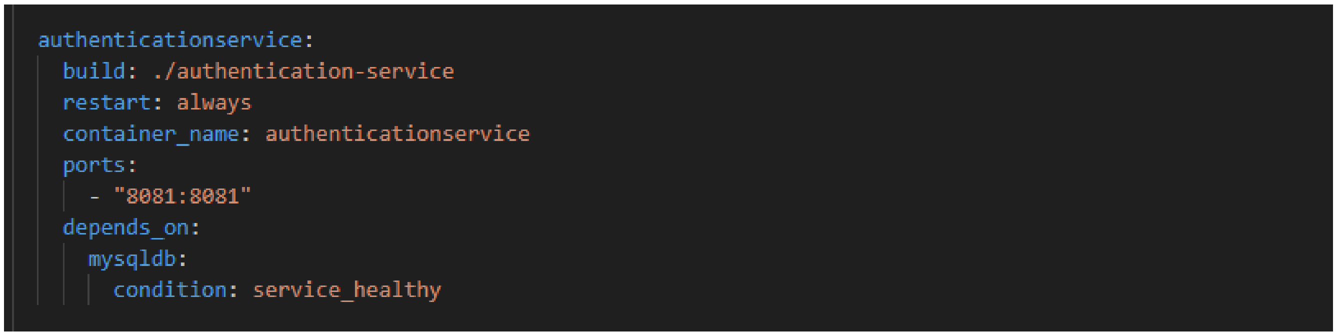
- MongoDB Service:

  In configuring the MongoDB service within the Docker Compose file, essential aspects are meticulously defined. The choice of the "mongo" Docker image, container naming as mongodb, and the specification for automatic restarts ensures consistent service availability. Through the port mapping mechanism, port 27017 within the container is externally accessible, enabling interactions.



```yaml
version: '3'
services:
  mongodb:
    image: mongo
    container_name: mongodb
    restart: always
    ports:
      - "27017:27017"
```

An example of a container that depends on the database (MySQL) service:



```yaml
authenticationservice:
  build: ./authentication-service
  restart: always
  container_name: authenticationservice
  ports:
    - "8081:8081"
  depends_on:
    mysqldb:
      condition: service_healthy
```

In this setup, the Authentication Service hinges on MySQL's health status, delaying its launch until MySQL is ready. The Docker Compose configuration also designates the Dockerfile location, enabling the service to build upon startup. By synchronizing dependencies and ports, the service is interwoven with the system. Moreover, the service relies on the "mysqldb" service's health check before initiating, ensuring a smooth and dependable deployment process.

5.0 Running the Project

Command to run docker compose : **docker-compose up**.

Services ports on host machine:

MySQL Service: **3306.**

MongoDB Service: **27017**.

Authentication Service: **8081**.

Enter video game service: **8000**.

Analytics Service: **9000**.

Show video game analytics service: **9500**.

The user credentials have to be entered manually when running the docker-compose up from scratch.

Example:

Username: **root**

Password: **21232619boom**

To run the **enter video game app** use this link:

http://localhost:8000/auth/login

And the **show video game analytics app**:

http://localhost:9500/auth/login