Containerization Project (Docker & CI\CD)

Video Streaming System

This system consists of five services working together using docker-compose file to achieve the purpose of the system.

In this report I will walk through these services and explain them one by one and how the interaction is done between them.

1) Authentication Service (Spring REST):

The goal of this service is to check if the user is authenticated to log in to the system or not. It is a simple one, just an API that will take an Id and password and based on that it will return TURE or FALSE.

2) MySQL Data Base Service:

A MySQL image from Docker Hub used in this service. All information about the name of the database, name of user and password are identified in the docker compose file.

```
mysqldb:
  container_name: mysqldb
  image: mysql
  environment:
    MYSQL_ROOT_PASSWORD: farooq
    MYSQL_DATABASE: video
  ports:
    - "3307:3306"
```

3) File System Service (Spring REST):

This service has two API's:

- The first one is to upload (write) video to the system storage, it takes the video file, name of the video and the path, and save this video in the given path.
- The second one is to read video from system storage, this service will take the path of the video and the name of the video and return the video itself as output.

4) Upload Video Service (Spring MVC):

This service needs to interact with the database, so in the docker-compose file information about the database must be identified:

```
upload-video:
  container_name: upload-video-service
  build: ./upload-video
  restart: on-failure
  ports:
    - "8081:8080"
  environment:
    SPRING_DATASOURCE_URL: jdbc:mysql://mysqldb:3306/video?allowPublicKeyRetrieval=true
    SPRING_DATASOURCE_USERNAME: root
    SPRING_DATASOURCE_PASSWORD: farooq
    SPRING_JPA_HIBERNATE_DDL_AUTO: update
  depends_on:
    - mysqldb
```

This service operates in the following sequence:

- When trying to upload video, the first thing is to authenticate your credentials, and this is done by interacting with authentication service.
- Second you must interact with the database because the information about the video will be saved in the database, so this service will interact with the database service.
- The last thing is interacting with file system service because we need to save the video in our system.

As an MVC application, this service will have layers as following:

- **Model layer:** this layer consists of Video class, and this video class represents the data in the database.
- **Repository layer:** this layer consists of VideoRepo class, and this class is responsible for interacting with database for saving and getting information.
- **Controller layer:** this layer is responsible for handling the interaction with the user.

NOTES:

- in this example, there is no service layer, and this is because the simplicity of this application also there is no business logic to put in the service layer.

About the interaction between the services, lets take the interaction between the upload video service and the authentication service:

- Using UriComponenetsBuilder to construct a URI string for making a request to an authentication service. The id and password are added as query parameters.

- Using RestTemplate to perform an HTTP GET request to the URI and expected a Boolean response from the authentication service.

```
private final String AUTHENTICATION_SERVICE = "http://authentication-service:8080/login";
```

```
@PostMapping("/login")
public String authenticate(@RequestParam String id, @RequestParam String password) {
    String uri = UriComponentsBuilder.fromUriString(AUTHENTICATION_SERVICE)
            .queryParam("id", id)
            .queryParam("password", password)
            .toUriString();
    Boolean result = restTemplate.getForObject(uri, Boolean.class);
    if (Boolean.TRUE.equals(result)) {
        return "redirect:/upload-video.html";
    } else {
        return "redirect:/index.html";
    }
}
```

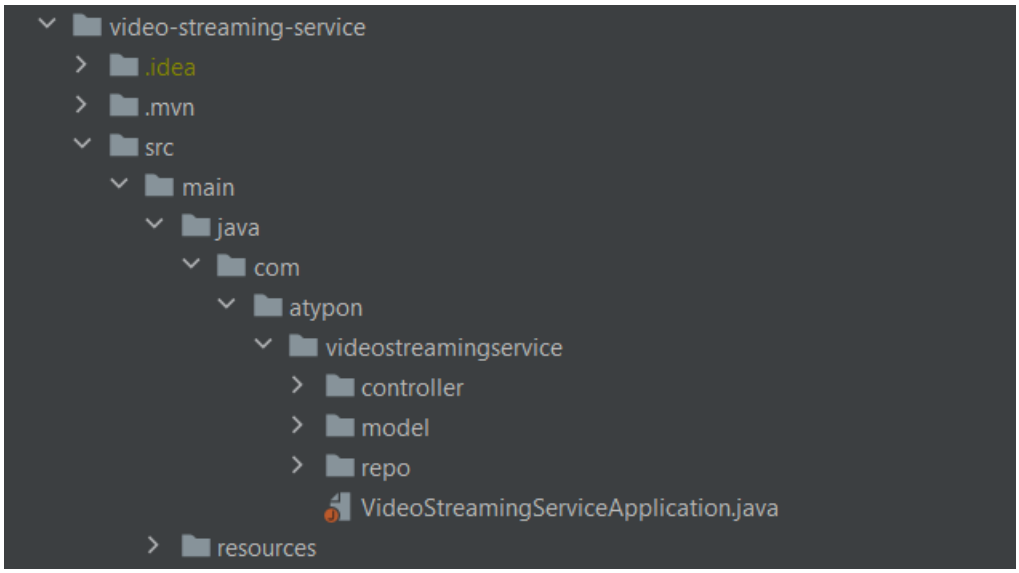
5) Video Streaming Service (Spring MVC):

This service operates in the following sequence:

- First, it will interact with the authentication service, to check if the user is authorized to view the videos or not, it will send the id and password of the user and get a Boolean value as response.
- Second, it will interact with the database service to get the paths and the names of the videos, the information about the database will be passed in the docker compose.

- Finally, it will interact with the file system service to read the videos, it will send the path of the video and the name of the video and get the video itself as response.

And this service also will have the same layers as explained in the upload video service:



Notes about the assignment:

- Each service will have its own Dockerfile.
- The docker-compose will generate images of each service by executing their Dockerfile.
- In the solution, two volumes are used:
  - The first volume for database information.
  - The second volume is for storing uploaded videos.
- Both MVC applications utilize JPA and interact with a shared database.
- Distinct ports are assigned to these services.

CI/CD Pipeline:

The continuous integration and continuous deployment (CI/CD) pipeline is designed to simplify the building, development and deployment process of a video streaming application.

Trigger Condition:

This pipeline is triggered by pushing to the master branch and pull requests from the same branch, this ensures that any changing to the master branch are subject to automated building and deployment.

Build Environment:

This pipeline runs on the latest version of Ubuntu OS, and it utilizes a MySQL service from Docker container to provide a database environment.

Build Steps:

- **Step 1 (Set up JDK 17):** Configures Java Development Kit (JDK) version 17 using Temurin distribution and cashes Maven dependencies.

- **Step 2 (Log in to Docker Hub):** Authenticate with Docker Hub to push Docker images to the Docker Dub.

- **Step 3 (Build and Push File System and Authentication Services):** This step rebuilds the services with Maven, builds the Docker images and pushing them to Docker Hub.

- **Step 4 (Build and Push Upload Video and Video Streaming Services):** This step rebuilds the services with Maven, builds the Docker images and pushing them to Docker Hub.
  Environment variables are used to configure database connection because these two services needed a database connection.

Conclusion

This CI/CD pipeline provides an automated and efficient workflow for the building, development, and deployment of a video streaming application. The using of Docker containers ensures consistency across different environments, while the use of CI/CD enhances collaboration the delivery of new features and improvements.