

# Docker Report

---

SEPTEMBER 12

---

Authored by: Deya' Aldeen AL-Bettar



# Table Content

---

<u>1: Introduction.....</u>	<u>3</u>
<u>2: Explanation.....</u>	<u>4</u>
<u>3: Testing.....</u>	<u>10</u>

# Introduction

---

This report presents the architecture and implementation of a containerized Video Streaming System composed of microservices for video uploading, video streaming, user authentication, file management, and database storage. Each service is containerized using Docker, and the system is orchestrated using Docker Compose to ensure the services run in an isolated, scalable, and manageable environment. The system provides a user interface for uploading and streaming videos while handling the backend processes of file storage, user authentication, and video metadata storage.

The services in this system include:

- Authentication Service: Validates user credentials.
- Upload Video Service: Allows users to upload videos after authentication.
- Video Streaming Service: Enables users to stream videos after authentication.
- File System Service: Handles reading and writing video files.
- MySQL Database Service: Stores metadata about videos and users.

Each microservice has its own Dockerfile to define its runtime environment, and they are all integrated using a Docker Compose file that defines the service interactions and dependencies.

# Explanation

---

The video streaming system is composed of several microservices that work together to provide authentication, video uploading, video streaming, and file storage. Each microservice has a dedicated responsibility and is containerized for easy deployment and scalability using Docker. Below is a detailed explanation of each service, its code, Dockerfile, and relevant sections of the `docker-compose` file.

## 1. Authentication Service

Authentication Service handle authentication requests not from users directly, but from other services like the Upload Video Service and the Video Streaming Service.

Code:

- **AuthController:** exposes a `/login` endpoint for users to submit credentials
- **AuthService:** contains the core authentication logic, checking whether the provided credentials match predefined values.

Dockerfile for Authentication Service:

```
FROM openjdk:20-jdk
EXPOSE 8001
COPY ./target/auth-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

- **Base Image:** Uses the OpenJDK 20 image for running a Java application.
- **Expose Port:** Exposes port `8001` for communication.
- **JAR File:** The compiled JAR file of the authentication service is copied into the container.
- **Entry Point:** Specifies the command to run the JAR file when the container starts.

Docker Compose Configuration:

```
auth-service:
  build: ./auth
  container_name: auth-service
  ports:
    - "8001:8001"
```

- **Service Name:** `auth-service` is built from the directory `./auth`.
- **Port Mapping:** Maps port `8001` of the container to port `8001` on the host machine.

## 2. File Service

The File Service is responsible for handling the uploading and serving of video files. It provides endpoints for file upload and makes the uploaded files accessible to other services (such as the Video Streaming Service) by configuring the file storage location.

Code:

- **FileController:** contains the `uploadFile` method, which handles file uploads from POST requests, saves the files to a specified directory, and returns a success or error message based on the upload result.
- **WebConfig:** configures the service to make uploaded files accessible via URLs by mapping requests to `/file-uploads/**` to the actual file storage directory on the file system. This allows files to be accessed directly through HTTP.

### Dockerfile for File System Service

```
FROM openjdk:20-jdk
EXPOSE 8003
COPY ./target/file-service-0.0.1-SNAPSHOT.jar app.jar
COPY ./target/file-uploads /file-uploads
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

- **Base Image:** Uses OpenJDK 20 for running the Java application.
- **Expose Port:** Exposes port `8003` for communication..
- **JAR File and Data:** Copies the JAR file and the file-uploads directory into the container.
- **Entry Point:** Runs the application on container startup.

### Docker Compose Configuration

```
file-system-service:
  build: ./file-service
  container_name: file-system-service
  ports:
    - "8003:8003"
  volumes:
    # - ./file-service/target/file-uploads:/file-uploads
    - file_data:/file-uploads
```

- **Service Name:** `file-system-service` is built from the `./file-service` directory.
- **Port Mapping:** Exposes port `8003` for communication with other services.

- Volume: The `file_data` volume is mounted to `/file-uploads`, ensuring that files uploaded to the File System Service are stored persistently and are accessible to other services.

### 3. MySQL Database Service

The MySQL Database stores metadata about the videos (such as video names, paths, and authors).

Docker Compose Configuration

```
mysql-db:
  image: mysql:5.7
  container_name: mysql-db
  ports:
    - "3306:3306"
  environment:
    MYSQL_ROOT_PASSWORD: rootpassword
  volumes:
    - mysql_data:/var/lib/mysql
    - ./mysql-db/init.sql:/docker-entrypoint-initdb.d/init.sql
```

- Image: The MySQL service uses the official `mysql:5.7` image.
- Port Mapping: Maps port `3306` for database access.
- Environment Variables: Sets the MySQL root password.
- Volumes: Mounts a persistent volume `mysql_data` for data storage and initializes the database with the provided `init.sql` file

### 4. Upload Video Service

The Upload Video Service provides a web interface for users to upload video files. The uploaded videos are stored in the File System Service, and the video metadata (such as file name, path, and author) is stored in the MySQL Database.

Code:

- AdminController: Manages views for login and upload pages.
- AuthController: Handles login requests by interacting with the Authentication Service to validate user credentials.

```
@RestController
@RequestMapping("/auth")
public class AuthController {

    @PostMapping("/login")
    public ResponseEntity<String> login(@RequestBody LoginDTO loginDTO) {
        RestTemplate restTemplate = new RestTemplate();
        String authServiceUrl = "http://auth-service:8001/auth/login";

        try {
            ResponseEntity<String> response = restTemplate.postForEntity(authServiceUrl, loginDTO, String.class);
```

- VideoController: Manages video uploads by interacting with both the File System Service to store video files and the Video Repository to save video metadata.

```
@RestController
public class VideoController {

    @Autowired
    private VideoRepository videoRepository;
    private final RestTemplate restTemplate;

    @Value("${file.server.url}")
    private String fileServerUrl;

    @Autowired
    public VideoController(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    @PostMapping("/video/create")
    public ResponseEntity<String> createVideo(
        @RequestParam("videoName") String videoName,
        @RequestParam("videoAuthor") String videoAuthor,
        @RequestParam("videoDescription") String videoDescription,
        @RequestParam("videoFile") MultipartFile videoFile) {

        String originalFileName = videoFile.getOriginalFilename();
        String uniqueFileName = UUID.randomUUID().toString() + "_" + originalFileName;

        try {

            MultiValueMap<String, Object> body = new LinkedMultiValueMap<>();
            body.add("file", new ByteArrayResource(videoFile.getBytes()) {
                @Override
                public String getFilename() {
                    return uniqueFileName;
                }
            });

            ResponseEntity<String> response = restTemplate.postForEntity(fileServerUrl, body, String.class);
        } catch (IOException e) {
            // Handle exception
        }
    }
}
```

In application.properties:

```
8 file.server.url=http://file-system-service:8003/file/upload
```

- VideoRepository: Provides CRUD operations for video entities in the database.

Dockerfile for Upload Video Service:

```
FROM openjdk:20-jdk
EXPOSE 8002
COPY ./target/upload-video-service-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java","-jar","/app.jar"]

#if you want to build, you can use this
#FROM maven:3-amazoncorretto-20 AS builder
#WORKDIR /app
#COPY pom.xml .
#RUN mvn dependency:go-offline -B
#COPY src ./src
#RUN mvn clean package -DskipTests
#FROM amazoncorretto:20
#WORKDIR /app
#COPY --from=builder /app/target/*.jar app.jar
#EXPOSE 8000
#CMD ["java", "-jar", "app.jar"]
```

- Base Image: Uses OpenJDK 20 for running the Java Spring Boot application.
- Expose Port: Exposes port 8002 for communication.
- JAR File: The compiled upload video service JAR file is copied into the container.
- Entry Point: Runs the JAR file when the container is started.

### Docker Compose Configuration:

```
upload-video-service:
  build: ./upload-video-service
  container_name: upload_video
  ports:
    - "8002:8002"
  environment:
    - spring.datasource.url=jdbc:mysql://mysql-db:3306/video_db
  depends_on:
    - auth-service
    - mysql-db
    - file-system-service
```

- Service Name: `upload-video-service` is built from the `./upload-video-service` directory.
- Port Mapping: Maps port 8002 of the container to the host.
- Environment Variables: Specifies the MySQL connection URL for the service to connect to the MySQL Database.
- Dependencies: The service depends on `auth-service`, `mysql-db`, and `file-system-service`, meaning these services must be running before the `upload-video-service` can start.

### Video Streaming Service

The Video Streaming Service allows users to stream videos that have been uploaded and stored in the system. Like the upload service, it requires user authentication before providing access to the video library.

Code:

- AuthController: Handles login requests, interacting with the Authentication Service to verify credentials.

```
@RestController
@RequestMapping("/auth")
public class AuthController {

    @PostMapping("/login")
    public ResponseEntity<String> login(@RequestBody LoginDTO loginDTO) {
        RestTemplate restTemplate = new RestTemplate();
        String authServiceUrl = "http://auth-service:8001/auth/login";

        try {
            ResponseEntity<String> response = restTemplate.postForEntity(authServiceUrl, loginDTO, String.class);
```



- **UserPagesController:** Manages view rendering for login and video listing pages.
- **VideoController:** Provides endpoints for fetching video metadata from the Video Repository.
- **VideoRepository:** Handles database operations for video entities.
- **Login Page:** Sends login requests to the `AuthController`, which then interacts with the Authentication Service to validate user credentials.
- **Videos Page:** Retrieves and displays a list of videos by making requests to the `VideoController`, which fetches video metadata from the `VideoRepository`.

## Dockerfile for Video Streaming Service

```
FROM openjdk:20-jdk
EXPOSE 8004
COPY ./target/video-streaming-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

- **Base Image:** Uses OpenJDK 20 for running the Java Spring Boot application.
- **Expose Port:** Exposes port `8004`.
- **JAR File:** Copies the video streaming service JAR file into the container.
- **Entry Point:** Executes the JAR file upon container startup.

## Docker Compose Configuration

```
video-streaming-service:
  build: ./video-streaming
  ports:
    - "8004:8004"
  environment:
    - spring.datasource.url=jdbc:mysql://mysql-db:3306/video_db
  depends_on:
    - auth-service
    - mysql-db
    - file-system-service
```

- **Service Name:** `video-streaming-service` is built from the `./video-streaming` directory.
- **Port Mapping:** Maps port `8004` to the host machine.
- **Environment Variables:** Sets the MySQL connection URL for video metadata retrieval.
- **Dependencies:** The service depends on `auth-service`, `mysql-db`, and `file-system-service`.



# Testing

To ensure that the architecture is functioning as expected, I performed the following testing steps:

- **Build docker-compose.yml file:** The `docker-compose build` command is used to build Docker images for the services defined in your `docker-compose.yml` file. Here's what it does:

```
D:\docker_task>docker-compose build
2024/09/12 22:26:50 http2: server: error reading preface from client //./pipe
d
[+] Building 6.7s (13/13)
=> [file-system-service internal] load build definition from Dockerfile
[+] Building 7.0s (13/13)
=> [file-system-service internal] load build definition from Dockerfile
[+] Building 13.8s (23/23)
```



































- The `docker-compose up` command is used to start and run the services defined in your `docker-compose.yml` file. Here's a detailed overview of what happens when you run `docker-compose up`:

```
D:\docker_task>docker-compose up
[+] Running 5/0
✓Container file-system-service          Created
✓Container auth-service                Created
✓Container mysql-db                   Created
✓Container upload_video                Created
✓Container docker_task-video-streaming-service-1 Created
Attaching to auth-service, video-streaming-service-1, file-system-service, mysql-db, upload_video
```




- **Images:**

<input type="checkbox"/>	Name <span>↑</span>	Tag	Status	Created	Size	Actions
<input type="checkbox"/>	<a href="#">docker_task-auth-service</a> b1083a74db68	latest	<a href="#">In use</a>	11 hours ago	513.18 MB	
<input type="checkbox"/>	<a href="#">docker_task-file-system-servi</a> dfdd05358411	latest	<a href="#">In use</a>	2 hours ago	515.24 MB	
<input type="checkbox"/>	<a href="#">docker_task-upload-video-ser</a> e4c210d9fc39	latest	<a href="#">In use</a>	2 hours ago	544.62 MB	
<input type="checkbox"/>	<a href="#">docker_task-video-streaming-</a> a5115b023fbf	latest	<a href="#">In use</a>	2 hours ago	542.04 MB	
<input type="checkbox"/>	<a href="#">mysql</a> 5107333e08a8	5.7	<a href="#">In use</a>	9 months ago	501.39 MB	

- Containers

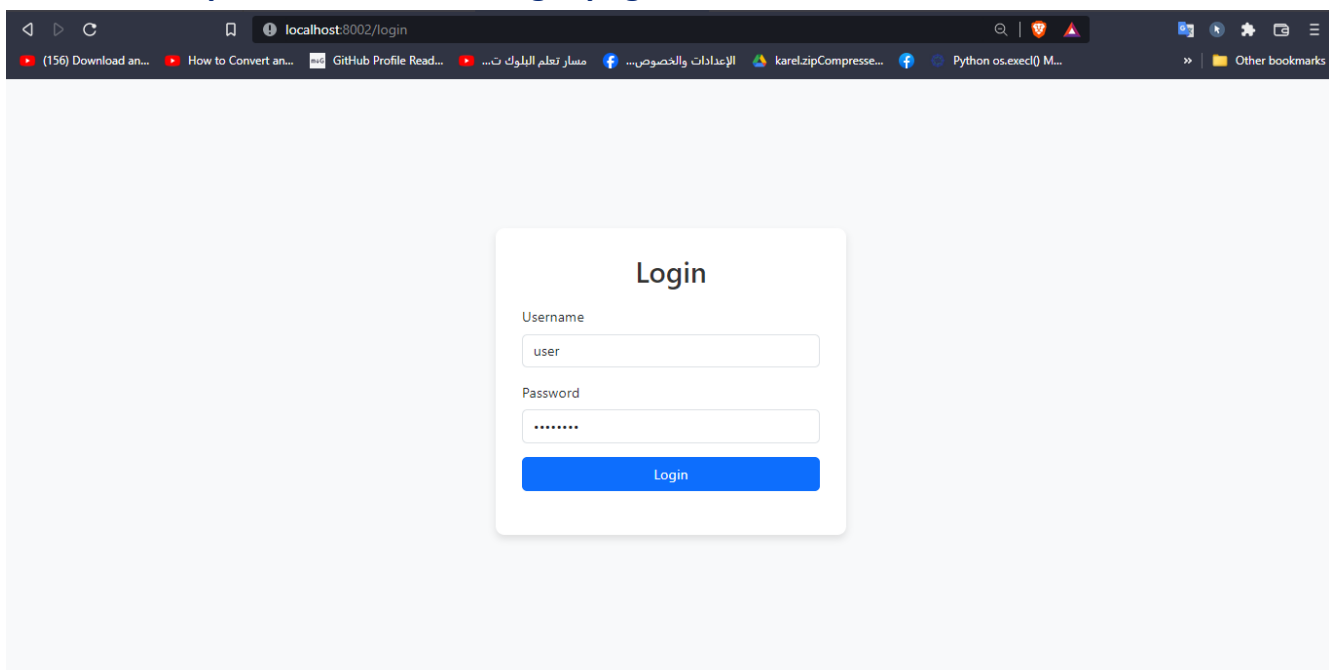
<input type="checkbox"/>	Name	Image	Status	CPU...	Port(s)	Last	Actions
<input checked="" type="checkbox"/>	 <a href="#">docker_task</a>		Running (5/5)	N/A		3 min	  
<input type="checkbox"/>	 <a href="#">upload_video</a> 3ee2b1a1a3d4 	<a href="#">docker_task-uploa</a>	Running	N/A	<a href="#">8002:8002</a> 	3 min	  
<input type="checkbox"/>	 <a href="#">video-streaming-service-1</a> 1e6cabf36529 	<a href="#">docker_task-vid</a>	Running	N/A	<a href="#">8004:8004</a> 	3 min	  
<input type="checkbox"/>	 <a href="#">mysql-db</a> 25d9861a6247 	<a href="#">mysql:5.7</a>	Running	N/A	<a href="#">3306:3306</a> 	3 min	  
<input type="checkbox"/>	 <a href="#">auth-service</a> 6395ca6e20cb 	<a href="#">docker_task-auth-</a>	Running	N/A	<a href="#">8001:8001</a> 	3 min	  
<input type="checkbox"/>	 <a href="#">file-system-service</a> 1b20be46933e 	<a href="#">docker_task-file-s</a>	Running	N/A	<a href="#">8003:8003</a> 	3 min	  

- Volumes

<input type="checkbox"/>	Name 	Status	Created	Size	Actions
<input type="checkbox"/>	<a href="#">docker_task_file_data</a>	in use	2 hours ago	2.3 MB	
<input type="checkbox"/>	<a href="#">docker_task_mysql_data</a>	in use	5 hours ago	210.1 MB	

## Work flow:

- Start upload-service with login page



- then upload video

localhost:8002/upload

### Upload Video

Video Name  
new video

Video Author  
deaa aldeen

Video Description  
good video

Video File  
Choose File Untitled video - Made with Clipchamp.mp4

Upload

Upload successfull! Video uploaded and metadata saved successfully:  
0ebdb187-a04d-4507-98cf-398cf31b8e0b\_Untitled video - Made with  
Clipchamp.mp4

- then go to video-streaming with login page

localhost:8004/login

### Login

Username  
user

Password  
\*\*\*\*\*

Login

- after logging will go to videos page

