Video Streaming System

A report

Done by: Aya Almallah
Instructors:
Motasim Aldiab
Fahed Jubair

Contents

Introduction

This report introduces a Video Streaming System created using Docker and Docker Compose. during Atypon's Training in May 2023. The goal was to make uploading, authenticating, streaming videos, and storing files smoother. This system shows how using Docker and microservices can make sharing videos better. The report explains how everything is set up, what each part does, and how Docker is used.

System Architecture

The system's architecture revolves around a cohesive assembly of microservices aimed at enabling efficient video uploading and streaming functionalities. The key constituents encompass the Authentication Service, MySQL Database, File System Service, Upload Video, and Video Streaming.

- **Authentication Service:** This service safeguards the system's integrity by verifying user credentials before granting access. It guarantees secure user interactions by authenticating their identity against stored data.
- **MySQL Database:** Serving as the system's core repository, the MySQL Database stores essential metadata about videos, along with their corresponding storage paths. This centralized hub facilitates efficient data retrieval and management.
- **File System Service:** Responsible for managing file operations, the File System Service ensures the secure storage and retrieval of video files, thus maintaining data integrity and accessibility.
- **Upload Video Microservice:** The Upload Video microservice handles the addition of videos into the system. It validates user credentials through the Authentication Service and subsequently integrates video metadata into the MySQL Database. The actual video file is stored using the File System Service.
- **Video Streaming Microservice:** On the other end, the Video Streaming microservice empowers users to watch videos. Following authentication by the Authentication Service, this microservice retrieves video metadata from the MySQL Database and streams the video file, coordinated through the File System Service.

The orchestration of these elements is paramount to the system's functionality. The Authentication Service guarantees secure entry, while the MySQL Database and File System Service serve as data repositories. The Upload Video and Video Streaming microservices collaborate seamlessly with these components to fulfill their designated roles. This interdependence forms the backbone of successful video uploading and streaming.

In the ensuing sections, we'll delve into comprehensive insights about the functionalities and implementation specifics of each microservice, elucidating their collective contribution to an integrated Video Streaming System.

Microservice Descriptions

Authentication Service

The **Authentication Service** microservice manages user authentication and access control, safeguarding the Video Streaming System from unauthorized usage.

- **Functionality:**
    - The microservice encompasses user registration, login, and token-based verification mechanisms.
    - Users can register with a unique username, password, and name via the **register** endpoint.
    - The **login** endpoint enables users to authenticate using their username and password. Successful authentication results in the generation of a JSON Web Token (JWT).
    - The **verify** endpoint validates the authenticity and expiration of JWTs, ensuring authorized access to the system.
    - The service interfaces with the MySQL Database to store and retrieve user information.
- **Key Components:**
    - The microservice relies on the AppUser entity class, representing user details such as userId, username, password, and name.
    - It utilizes the AppUserRepo interface to interact with the database, performing operations like finding users by their username.
    - JWTs are employed to ensure secure user authentication and subsequent access to other system functionalities.
- **Interactions:**
    - Provides endpoints for user registration, login, and token verification.
    - Collaborates with the MySQL Database to store and retrieve user data.
- **Purpose:** The **Authentication Service** is instrumental in ensuring secure access to the Video Streaming System. It manages user authentication and authorization through token-based mechanisms, guaranteeing that only authenticated users can engage with the system.

File System Service

The **File System Service** microservice is responsible for handling file uploads and storage, utilizing Amazon S3 as the storage solution.

- **Functionality:**
    - The service configures Amazon S3 access using provided AWS credentials and region information.
    - It offers endpoints for uploading and deleting files.
    - Uploaded files are stored in the configured Amazon S3 bucket, and their URLs are returned.
    - Files can be deleted from the bucket based on their filename.
- **Key Components:**
    - The **StorageConfig** class configures the Amazon S3 client using AWS credentials and region settings.
    - The **StorageController** class defines endpoints for file uploading and deletion.
    - The **StorageService** class contains the core logic for uploading and deleting files in the Amazon S3 bucket.
- **Interactions:**
    - Provides endpoints for uploading and deleting files.
    - Collaborates with Amazon S3 for file storage and retrieval.
- **Purpose:** The **File System Service** ensures efficient and scalable file storage and retrieval using Amazon S3. It enables the system to manage uploaded videos, thumbnails, and other assets, providing reliable and high-performance file storage capabilities.

Amazon S3 and MySQL DB Service

The **MySQL DB Service** component of the system plays a crucial role in storing and managing video metadata, paths, and other relevant information. This service, in tandem with Amazon S3, ensures comprehensive data management and retrieval.

- **Functionality:**
    - The service interacts with the MySQL database to store metadata and paths of uploaded videos.
    - It interfaces with Amazon S3 for efficient storage and retrieval of video files and thumbnails.

- **Key Components:**
    - The **Video** entity class defines the structure of video metadata stored in the MySQL database.
    - The **VideoRepo** interface extends **JpaRepository** and provides methods for querying video metadata.
    - The service methods in **VideoService** interact with the MySQL database to manage video metadata and path information.
- **Interactions:**
    - When a video is uploaded, the **VideoService** communicates with both the MySQL database and Amazon S3:
        - The metadata, such as video name, author, and path, is stored in the MySQL database.
        - The video file and thumbnail are securely stored in Amazon S3, and the corresponding URLs are stored in the MySQL database.
    - When streaming a video, the service retrieves metadata and paths from the MySQL database and retrieves the actual video content from Amazon S3.
- **Advantages of Amazon S3 Integration:**
    - **Efficient Storage:** Amazon S3 handles the actual video and thumbnail storage, providing scalable and reliable storage.
    - **Offloading Database:** Storing files in Amazon S3 reduces the load on the MySQL database, optimizing its performance for metadata operations.
    - **Enhanced Accessibility:** Users can seamlessly access videos from anywhere, as Amazon S3 provides high availability and low-latency data access.
- **Impact on System:** The integration of Amazon S3 and the MySQL DB Service enables the system to efficiently manage video metadata and content. By leveraging Amazon S3's storage capabilities, the service ensures reliable and accessible video storage while the MySQL database focuses on metadata management.

This synergy between the Amazon S3 storage and the MySQL DB Service ensures an optimized approach to handling both metadata and content, contributing to a cohesive and high-performing video streaming system.

Upload Video (Web app):

The **Upload Video** microservice facilitates the seamless uploading of videos, ensuring user authentication and secure data storage.

- **Functionality:**
  - Upon receiving a video file, along with its associated thumbnail and name, the microservice initiates an authentication process via the Authentication Service.
  - The Authentication Service validates the user's credentials, ensuring authorized access to the system.
  - After successful authentication, the microservice interacts with the MySQL Database to store essential video metadata, including the video link, thumbnail link, name, author, and user ID.
  - Utilizing the File System Service, the **Upload Video** microservice ensures the secure storage of both the video file and its corresponding thumbnail for future retrieval.
- **Key Interactions:**
  - Interfaces with the Authentication Service for user credential verification.
  - Coordinates with the MySQL Database to store video metadata.
  - Utilizes the File System Service for secure video and thumbnail storage.
- **Purpose:** The microservice's purpose is to serve as a secure gateway for video uploads, encapsulating the process of user authentication, systematic metadata storage, and secure file storage within the system.

Video Streaming (Web app):

The **Video Streaming** microservice is dedicated to facilitating seamless video streaming for users, leveraging user authentication and optimized data retrieval.

- **Functionality:**
  - The microservice encompasses the rendering of a streaming interface for users to access videos.
  - It collaborates with the Authentication Service to validate user credentials, ensuring secure access.
  - The microservice interacts with the MySQL Database to retrieve video metadata, including video links, names, authors, and view counts.
  - Utilizing the File System Service, the microservice reads and streams video files for an uninterrupted playback experience.

- **Key Components:**
    - The streaming interface is designed to provide users with a visual representation of videos, including thumbnails, titles, authors, and view counts.
    - The Authentication Service guarantees secure entry to the streaming interface by validating user credentials.
    - The MySQL Database serves as a data repository for video metadata, offering essential information for the video streaming process.
    - The File System Service aids in the seamless retrieval and streaming of video files.
- **Interactions:**
    - Interfaces with the Authentication Service for user credential verification.
    - Collaborates with the MySQL Database to retrieve video metadata.
    - Utilizes the File System Service to read and stream video files.
- **Purpose:** The **Video Streaming** microservice ensures a smooth user experience by providing an interface for video playback. It establishes connections with the Authentication Service and MySQL Database to authenticate users and retrieve video data, while the File System Service ensures the efficient retrieval of video files for seamless streaming.

The interactions between different microservices, such as the **Authentication Service**, **File System Service**, **Upload Video**, and **Video Streaming**, create a coherent and integrated Video Streaming System, facilitating user authentication, video upload, storage, and streaming functionalities.

Dockerization

The Dockerization process involves packaging each microservice of the Video Streaming System into lightweight, isolated containers. This enables efficient deployment, scalability, and consistent environments across various stages of the development and deployment lifecycle. Below, we outline the Dockerization process for each microservice along with key points about their Dockerfiles:

Authentication Service:

- The Authentication Service, responsible for user authentication and JWT generation, is containerized using a Dockerfile that defines the base image and startup command.

File System Service:

- The File System Service, managing file uploads and Amazon S3 interactions, is containerized using a Dockerfile that specifies the base image and execution command.

Upload Video Microservice:

- The Upload Video Microservice, tasked with video uploads, is Dockerized using a Dockerfile that outlines the base image and startup instruction.

Video Streaming Microservice:

- The Video Streaming Microservice serves video content and user interactions. Its Dockerization involves a multi-stage build process:
    - The first stage, defined in a Dockerfile, builds the project and configures its dependencies.
    - The second stage, also defined in a Dockerfile, sets up a server to host the application.

Frontend Microservice:

- The Frontend Microservice, delivering the application's user interface, follows a similar multi-stage Dockerization process as the Video Streaming Microservice.

In conclusion, Dockerization streamlines the deployment process by packaging each microservice as a self-contained container. This approach enhances consistency, portability, and scalability of the Video Streaming System, making it easier to manage and deploy across different environments.

Docker Compose

The Docker Compose file plays a crucial role in orchestrating the deployment of the Video Streaming System. This file defines the services that make up the system, how they interact with each other, and various configurations that facilitate their seamless operation. Here is an overview of the Docker Compose file's structure and its significance in managing the deployment process:

- **Version Definition:** The file begins by specifying the version of the Docker Compose file format being used. This version helps ensure compatibility with the defined configurations.
- **Service Definitions:** Within the **services** section, each microservice is defined as a separate service. This includes details about building the service's Docker image, such as the location of its corresponding Dockerfile.
- **Microservice Configurations:** For each microservice, additional configurations can be specified. These may include details about the container's name, the ports it exposes, and dependencies it relies on.
- **Interconnections:** The Docker Compose file automatically establishes a private network for the services. This network enables seamless communication between services using their designated service names as hostnames. For instance, one service can communicate with another using its service name as an address.
- **Environment Variables:** Some services might require environment variables to function properly. The Docker Compose file allows the definition of these variables within the service configurations, providing necessary information for service initialization.
- **Volumes (Not Explicitly Shown):** Although not visible in the provided example, Docker Compose supports the configuration of volumes. Volumes allow data to persist across container restarts, contributing to data management and integrity.

Benefits:

- **Simplicity and Consolidation:** Docker Compose centralizes the configurations of all microservices, making it easier to manage and deploy the entire system. This consolidation streamlines the deployment process.
- **Service Interaction:** Through the private network created by Docker Compose, services can communicate seamlessly using service names as hostnames. This simplifies the communication setup between different parts of the system.
- **Scalability and Flexibility:** Docker Compose facilitates the scaling of individual services independently, offering flexibility in accommodating varying workloads.
- **Enhanced Data Management:** The ability to define volumes within the Docker Compose file ensures data persistence across container lifecycles, contributing to effective data management strategies.

In essence, the Docker Compose file acts as a blueprint for orchestrating the deployment of the Video Streaming System. By defining services, interconnections, configurations, and potential environment variables or volumes, it optimizes the process of setting up and managing the entire system.

Deployment and Running the System

Deploying and running the containerized Video Streaming System using Docker Compose is a straightforward process. Follow these step-by-step instructions to set up the system on your local environment:

1. **Prerequisites:**
    - Ensure you have Docker and Docker Compose installed on your machine.
    - Have the Docker Compose file ready, which defines the services and their configurations.
2. **Navigate to Project Directory:** Open a terminal window and navigate to the root directory of your Video Streaming System project.
3. **Build Docker Images:** Run the following command to build the Docker images for all microservices:

    ```
    docker compose build
    ```
4. **Start Containers:** Once the images are built, start the containers using Docker Compose:

    ```
    docker compose up
    ```
5. **Observing Initialization:** As the containers start, you'll see logs indicating the initialization process of each microservice. You'll notice services establishing connections and setting up resources.
6. **Accessing Services:**
    - The Video Streaming web application will be accessible at **http://localhost:3089**. You can use a web browser to navigate to this URL and interact with the application.
    - The Video Streaming backend services, authentication service, and file system service are accessible through their designated ports as defined in the Docker Compose file.

7. **Interacting with the System:**
    - Use the Video Streaming web application to view and stream videos.
    - Authenticate using the provided credentials or create new users using the authentication service.
    - Upload videos through the upload video service.
    - The system components will communicate seamlessly due to the private network established by Docker Compose.
8. **Shutting Down:** To stop and shut down the system, simply press **Ctrl + C** in the terminal window where Docker Compose is running. This will gracefully shut down the containers.
9. **Cleaning Up:** If you want to remove the containers and clean up resources, run the following command:

```
docker compose down
```