Atypon Training Assignment
Containerization

Yazan Abdullah

Contents

Chapter 1

Introduction

In this report, I will present my solution to the seventh assignment in Atypon Training.

I built the second option, a containerized video streaming system (see Figure 1.1), because I felt more curious about it.

Users can upload their videos through the management website and watch them on the streaming website.

I implemented a CI/CD pipeline (continuous delivery) that will automatically build and upload the new docker images to Docker Hub using GitHub Actions.

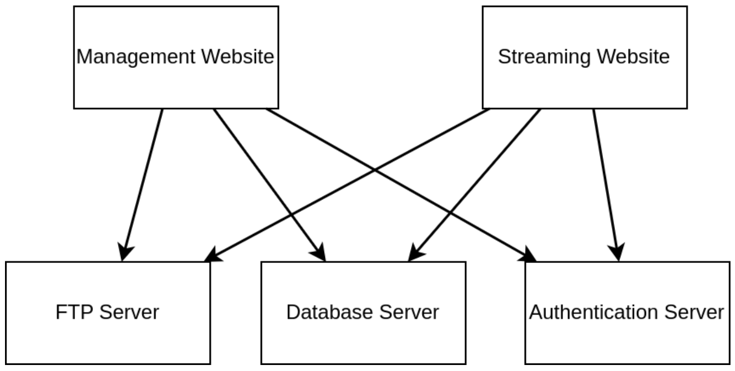

Figure 1.1: Video Streaming System.

Chapter 2

Some Technologies I Used

2.1 Maven

I used Maven to automate the building process and manage the dependencies in the project.

I set it up to be a multimodule project, such that each service is implemented in a separate module, and the common configurations between different services are inherited from the parent pom file.

This structure also allows me to build and package all the modules with a single command.

2.2 Jetty Server

Most of the communication in this project occurs over the `HTTP` protocol. This is why I used Java Servlets and JSP technology to handle the requests.

I used Eclipse Jetty to manage my servlets, due to its high performance and suitability to the microservices architecture.

2.3 Docker Maven Plugin

I delegated the responsibility of building docker images to maven by adding the docker maven plugin.

This plugin greatly simplifies building docker images and reduces it to running a single command.

Build configurations for this plugin, including the image name and tags, exist in POM files.

2.4 GitHub Actions

I used Git for version control and uploaded the project repository to GitHub.

Then, I used GitHub Actions to complete the CI/CD pipeline, as can be seen from the following code snippet:

```
name: Containerization CI/CD
on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up JDK 21
        # ...
      - name: Build with Maven
        # ...
      - name: login to docker hub
        # ...
      - name: push images to docker hub
        # ...
```

Chapter 3

Management Website

The purpose of this service is to provide an interface for users to manage their videos. Currently, It includes three pages: a login page, a home page, and a page for video upload.

I used Java servlets and JSP technology to build the website, then packaged it as a docker image so it could be deployed easily.

I defined the mapping between url paths and the servlets in the module in the web.xml file. I also configured the web application in the jetty-web.xml file.

This module communicates with the database to register new video metadata. It does so using the JDBC API and uses HickariCP as a Datasource.

It writes files to the FTP server using the Apache Commons Net library.

It also communicates with the authentication server using the HTTP protocol.

The docker file is fairly simple; it starts with the latest jetty docker image, then copies the war file to the proper directory; finally, it executes the jetty start mechanism in tool mode to set up the image. You can see the Dockerfile contents in the following code block:

```
FROM jetty
```

```
COPY maven/root.war /var/lib/jetty/webapps/root.war

RUN java -jar "$JETTY_HOME/start.jar"
   --add-modules=ee10-webapp,ee10-deploy,ee10-jsp,ee10-jstl,
     ee10-websocket-jetty,ee10-websocket-jakarta,server,http,deploy

EXPOSE 8080
```

Chapter 4

Streaming Website

Users can watch their videos on this website. Currently, It includes three pages: a login page, a home page, and a page for video streaming.

Just as in the previous chapter:

- I used Java servlets and JSP technology to build the website, then packaged it as a docker image.
- I used JDBC and HickariCP to communicate with the database server and retrieve video metadata.
- I used the Apache Commons Net library to connect to the FTP server and retrieve videos for streaming.
- I used the authentication server to authenticate users' credentials at login.

The Dockerfile is identical to the one in the previous chapter.

As you probably noticed, the only difference between the managing website and the streaming website, is that the former is used to upload videos and latter is used to watch them.

Chapter 5

Authentication Service

This service is responsible for registering new accounts and validating users' credentials.

This service can be accessed through the HTTP protocol by sending POST requests loaded with user credentials and other relevant data to `http://ip:8080/authenticator/authenticate`.

Figure 5.1 shows the class diagram for this service. I used the singleton design pattern and concurrent data structures while building the login manager to avoid race conditions and share the data between different instances.

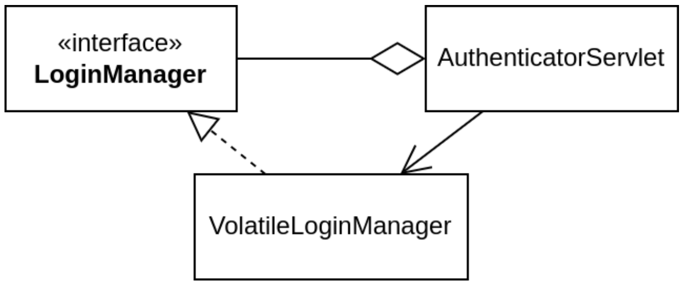

Figure 5.1: Authentication Service: Class Diagram.

To build the service, I packaged the app into a war file using `mvn package` command; then, I used a similar Dockerfile to the previous two services.

Chapter 6

Database Service

To store videos metadata, I used the latest mysql database docker image, and prepared its users and tables using the following script:

```
CREATE DATABASE IF NOT EXISTS videoStreaming;

USE videoStreaming;

CREATE TABLE video (
    owner varchar(20),
    name varchar(20),
    path varchar(250) PRIMARY KEY
);

CREATE USER 'managementApp'@'%' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON videoStreaming.* TO 'managementApp'@'%';

CREATE USER 'streamingApp'@'%' IDENTIFIED BY 'password';
GRANT SELECT ON VideoStreaming.* TO 'streamingApp'@'%';

FLUSH PRIVILEGES;
```

In the Dockerfile, I copied the script to `/docker-entrypoint-initdb.d`, such that it will be run when I start the containers.

Then, in docker-compose file, I mounted a directory to a volume in the container, so I can persist the data across multiple containers.

Chapter 7

File System Service

The file system service is responsible for reading and writing files to/from file storage.

To build this service, I used an FTP server docker image to manage communication and storage.

I chose FTP because it is designed to handle large file transfer, and because its interface is fairly simple.

I used docker volumes to persist the data. This provides me the required flexibility to vary the file storage.

The following code excerpt shows how I set up this service in my docker compose file:

```
filesystem:
  image: delfer/alpine-ftp-server
  container_name: fs
  environment:
    - USERS=webapp|1234|/home/videos|1001
  volumes:
    - ./videos:/home/videos
  networks:
    - back-tier
  restart: on-failure
```

Chapter 8

Docker Compose

I used docker compose to manage and coordinate all the containers in this project.

In the `docker-compose.yml` file, I defined the different services (five in total), their dependencies and configurations.

You can find an example of the services in the following code block:

```
streaming:
  image: yazannassr/streaming:latest
  container_name: strm
  depends_on:
    - authentication
    - filesystem
    - database
  ports:
    - "8080:8080"
  networks:
    - front-tier
    - back-tier
  restart: on-failure
```

As you can see, I specified the docker image, the container name, dependencies, networks, and restart policy. I also mapped the port number so I can access the app from my home machine.

End of the Report

Thank You