Engineering 2: Automated Software Engineering

Exam number: Y3889274

## 2.1 Data-Intensive Systems

### 2.1.1 Architecture

A diagram of a person

Description automatically generated

[Insert UML Diagrams]

By storing the videos likes and dislikes as a set of users rather than an integer, we can scale up the system to recommend videos based on which videos a user has liked or disliked. This could also assist in recommending videos with specific hashtags. As well as subscribing to a hashtag, we could recommend videos based on which hashtags are assigned to a liked video.

Using a microservice architecture allows us to scale with increasing user demands as it allows different services to be scaled depending on its individual needs and allows us to independently split up different services, should they become too big, to handle each specific business capability. For example, within the video service, as the functionality of a video changes, e.g. a high volume of videos to store, we could scale this independently to the rest of the services. We could also remove the management of a user to a new user microservice.

The use of asynchronous communication using events allows for a complete decoupling of the microservices so as we scale and separate the microservices communication between the microservices will not be affected as they are not communicating directly with each other. Also, asynchronous communication allows for message queues so as traffic in the system increases, we can prioritize, and store messages prevent overloading the system.

### 2.1.2 Microservices

Document the high-level design of these microservices and the appropriate usage of your command-line client.

The system is split into three microservices: the video-microservice, trending-microservice and the subscription-microservice. These services all communicate with each other through kafka events.

The video microservice handles the management of videos, users, and hashtags. It allows users to create, view, like and dislike videos. Within the CLI, it contains 6 commands: post-video, watch-video, like-video, dislike-video, list-videos, and list-videos-by-user.

The post-video command takes the parameters: a title, a list of hashtags and a username. It then checks the user database for an existing user with the given username. If there is not a user created with the username, it will create a user, save it to the service’s user database and assign it to the new video as the poster. Given the hashtags in the request, which is a string of a comma-separated list, the post-video endpoint splits the string into the individual hashtags, and similarly to the username search, checks for a hashtag and creates one if there is not an existing hashtag, saving it to the service’s hashtag database, and adds it the videos hashtags. It then saves the video to the video database and publishes an event sending the new video-id and video. This event is then subscribed to by the trending and subscription service.

The decision to use a username rather than the user id in the request was made to encourage user friendly behaviour as it is easier to remember a username rather than an arbitrary number. For this reason, all endpoints that require a user, requests a username.

The video microservice also has the like-video, dislike-video, and watch-video request. All three of these commands take a video id and a username in the request. Similarly, to the post-video command, these commands check the user database for a user matching the given username and creates one if there is not an existing user, then adds the user to the video’s likes, dislikes, or views accordingly. All three of these endpoints publish an event with the video and user objects: the like-video event is consumed by the trending service and the watch-video event is consumed by the subscription service.

The list-video command takes no request parameters and returns the full list of videos posted. The list-video-by-user command takes a username and returns a 404 error message if the user is not found, otherwise, it returns the list of videos posted by the user.

The trending service subscribes to the post-video event and consumes these messages to generate a database of hashtags. The trending service also subscribes to the `like-video` event, consuming these messages to stream the trending hashtags in a rolling window of an hour.

The subscription service allows the user to subscribe to hashtags and list the next ten videos to watch. The service consumes the `post-video` and `watch-video` events from the video-microservice to create its database of videos, it then uses the watch-video event to determine the most popular videos. The `list-videos-to-watch` endpoint takes in a username and a hashtag and lists the most popular videos with that hashtag that the user has not already seen. The subscription service also has the endpoints `subscribe-to-hashtag` and `unsubscribe-to-hashtag` that given a hashtag and username, allow a user to subscribe and unsubscribe to a hashtag.

For the full list and usage of the commands, please refer to the ReadMe.md file within the top-level `microservices` directory.

### 2.1.3 Containerisation

Package and deploy the system as a set of Docker containers orchestrated through the technologies presented during the module’s lectures and practicals (e.g. Docker Compose). Discuss how the solution can scale up to larger numbers of users and be resilient to failures (e.g. of a container, or a node).

Each service has its docker container and within the docker-compose file we declare the number of clusters and partitions within the Kafka clusters. This allows for scaling as we increase the number of services, we can just increase the number of declarations of docker containers for the services. The use of docker-compose allows us to define the number of partitions of the Kafka clusters, so as the number of messages sent between services increases, we can increase the number of the partitions.

### 2.1.4 Quality Assurance

Perform testing for your microservices through an appropriate combination of approaches to manage risk. Give a brief report on the actual tests, including statistics of what tests were run and what results were achieved, with a clear statement of any tests that are failed by the current implementation. If some tests failed, explain why these do not or cannot be passed and comment on what is needed to enable all tests to be passed. If no tests failed, comment on the completeness and correctness of your tests instead. [12 marks, max 2 pages]

Inspect all Docker images you used (both your own, and any others you use) for security vulnerabilities. If you find any security issues, show how you have dealt with them, or justify why this was not feasible with the time available. [3 marks, max 1 page]

When inspecting my docker image for vulnerabilities, I found the highest number of them within the video-microservice image with 29 total vulnerabilities including 4 critical vulnerabilities and 18 high-level vulnerabilities. I reduced this to only two high-level vulnerabilities by updating the version of alpine used to build the image.

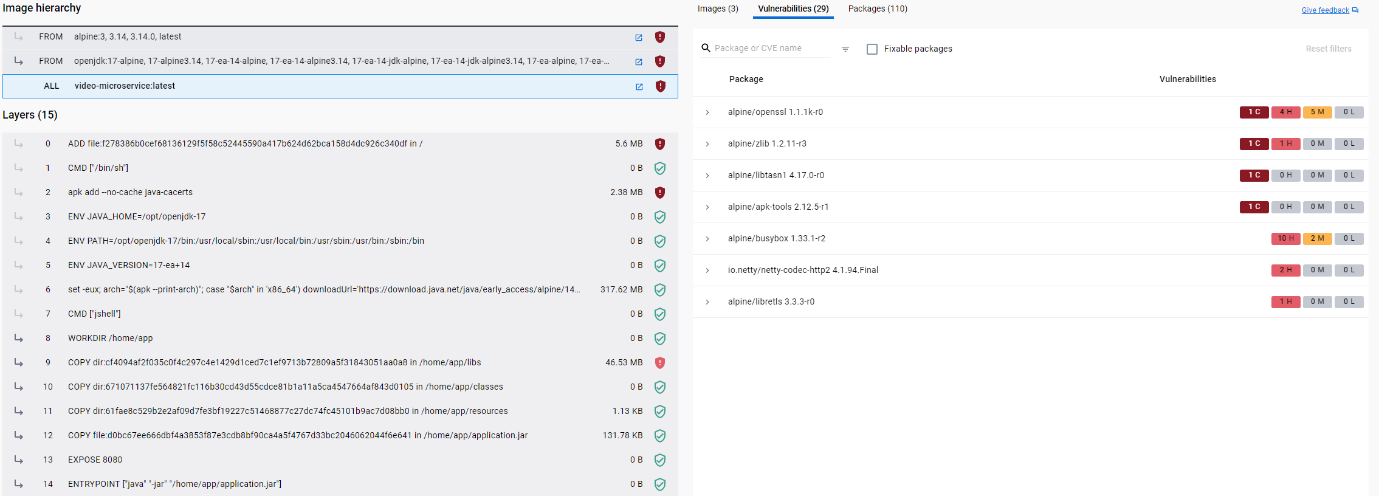


Figure : video-microservice image vulnerabilities prior to fix

A screenshot of a computer

Description automatically generated

Figure : video-microservice image vulnerabilities post fix

## 2.2 Application of Model-driven Engineering

### 2.2.1 Metamodel

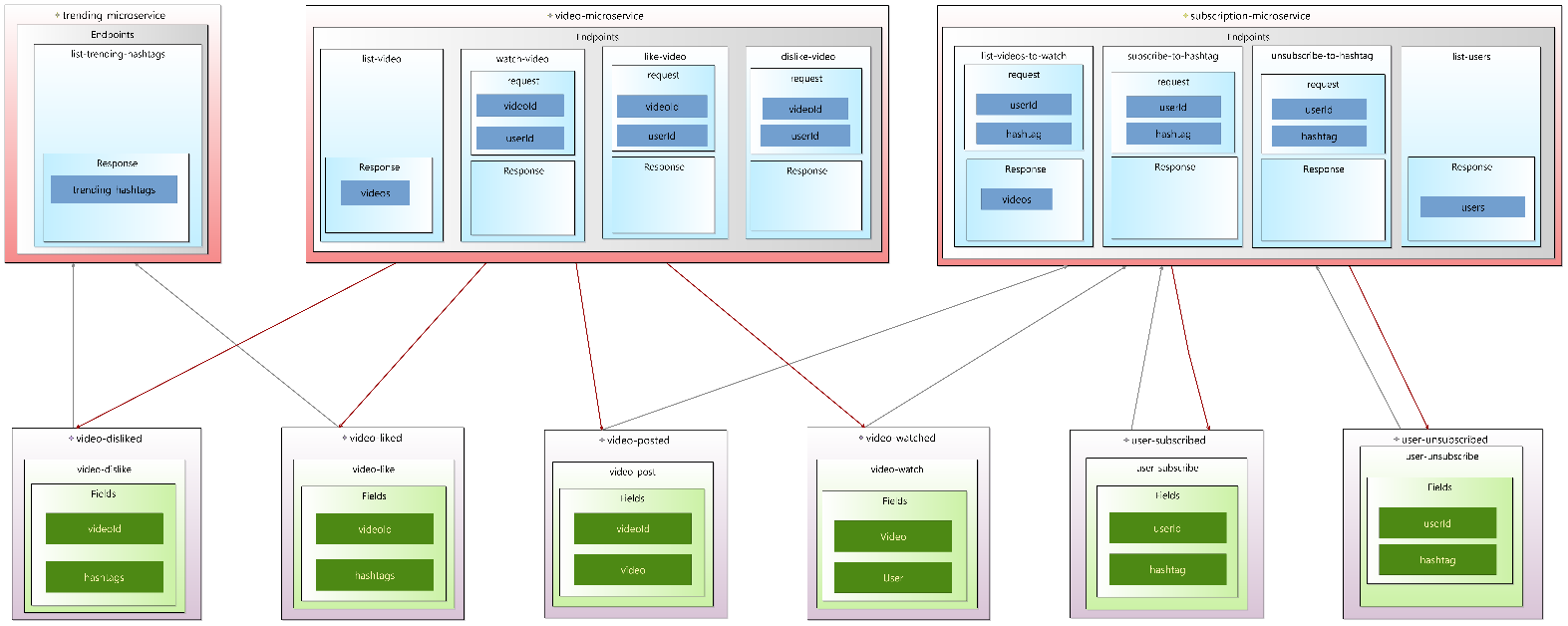
A diagram of a computer program

Description automatically generatedUse Emfatic/Ecore to define a metamodel for the domain-specific language described in Section 1.2. Include a class diagram of the metamodel in your report, discuss the metamodel, state any assumptions you have made, and explain any alternative design decisions that you have considered and discounted. [8 marks] (max 2 pages)

I decided to not include the Command Line Interface (CLI) in this model representation as the emf makes it clear that the microservices communicate using events and the endpoints are clearly stated within the microservices, so did not feel the need to include this.

### 2.2.2 Graphical Concrete Syntax

Use Eclipse Sirius or Picto to define and implement a graphical concrete syntax for the metamodel you defined in Question 2.2.1. Design the model of your architecture in your language, provide a screenshot of the model in your concrete syntax, discuss and justify your syntax design and implementation decisions, and reflect on the strengths and weaknesses of the selected concrete syntax compared to alternatives. [12 marks] (max 2 pages)



A weakness of the graphical syntax defined is that I have not represented the CLI as this would complicate the generation of the final code.

### 2.2.3 Model Validation

Use the Epsilon Validation Language to implement any validation constraints specified in Section 1.2, which cannot be expressed in the metamodel itself. Briefly explain the rationale and implementation of each constraint. [5 marks] (max 1 page)

One constraint I made was to check that all events defined in the model are used in at least one stream. This was to make sure that there would not be any redundant events defined in the model.

### 2.2.4 Model-to-Text Transformation

Discuss the model-to-text transformations and justify the organisation of the generated code. [15 marks] (max 2 pages)

I chose to generate the docker compose file. The EGX file takes a system and transforms each microservice into a docker image, it also takes in a defined number of Kafka clusters and generates the images for each Kafka node. A downside to passing the number of kafka nodes in the EGX file means that if we want to increase the number of nodes, we would have to manually change this in the EGX file. However, generating the docker compose file using the model to text transformations means that the docker file can be easily configured as we scale the system and increase the number of microservices, the docker file and images can be configured automatically.