Homework 2

Progetto (a).DB1.GW2.STATS1

Questo progetto ha lo scopo di implementare un sistema di gestione per video in formato MPEG-DASH.

Si è deciso di realizzare un'applicazione SpringBoot che utilizza Apache Maven come tool per la gestione dell'applicazione basata su java.

Come database relazionale che interagisce con l'applicazione si è scelto di utilizzare MySql.

All'interno del POM vengono specificate tutte le dipendenze utilizzate, tra cui quelle che riguardano mysql, SpringSecurity che è il meccanismo di sicurezza implementato da spring, ecc.

Componenti:

VideoManagementService:

Componente che fornisce l'interfaccia REST che espone i diversi endpoint.

Qui si è scelto di creare le entità User e Video e i relativi Repository, tramite i quali si accede direttamente al database.

L'entità User è caratterizzata da un Id generato automaticamente, nome, email(nome utente univoco), pass e ruoli. I ruoli li useremo per la sicurezza, in base a ruoli, faremo un filtraggio delle operazioni che un determinato utente può fare.

L'entità Video è caratterizzata da un Id generato automaticamente, da un nome, author e da uno stato.

Vengono creati anche i relativi Controller e Service.

I controller useranno i servizi per poter accedere al db, tramite Repository.

In VideoManagementController e UserController abbiamo implementato diversi metodi, ognuno dei quali è in grado di gestire una richiesta a un determinato endpoint.

Abbiamo definito un path di base che è vms/, tutte le richieste che verranno effettuate a questo url, verranno gestite da questo controller.

Le richieste implementate nel VideoManagementController sono:

1) POST http://localhost:8080/vms/videos/
   - Prende in ingresso il payload json: *{nomevideo, autorevideo}*
   - L'utente loggato viene settato come riferimento al record.
   - Imposta lo stato del video su "WaitingUpload"
2) POST http://localhost:8080/vms/videos/{id}
   - Prende in ingresso un file video.mp4
   - Vengono fatti i controlli di esistenza del record con id *:id* nel db e che il record sia associato all'utente autenticato (400 se non esiste).
   - Esegue l'upload del video al path */vms/var/video/:id/video.mp4*
   - Scrive su una coda kafka "process|$ID", che sarà poi letto dal videoprocessing
   - Imposta lo stato del video come *"Uploaded"*
3) GET http://localhost:8080/vms/videos/
   - Ritorna una lista in json dei video disponibili interrogando il database
4) GET http://localhost:8080/vms/videos/{id}
   - Verifica se il record con id *:id* esiste nel db (404 se non esiste).

- Risponde con un HTTP Status code 301 puntando alla url *http://"+addr_api+":8080/videofiles/"+id+"/video.mpd*

Mentre nello UserController:

5) POST http://localhost:8080/register/
   - registra un utente per l'autenticazione e l'accesso alle chiamate POST.

Per le richieste 1) e 2) per le quali è richiesta l'autenticazione, viene implementata la sicurezza usando SpringSecurity che permette la basic authentication. Mentre le altre richieste le può fare chiunque senza autenticazione.

Inoltre è stato inserito anche un listener che si occupa di leggere la coda kafka che viene scritta anche da componente videoprocessing, se leggerà un valore "processed|$ID" allora il processamento del video è andato a buon fine e imposta lo stato su "Available", altrimenti su "NotAvailable".

Vengono creati i Dockerfile-dev e Dockerfile-prod e le relative immagini:

docker build -t vms:v1 . -f Dockerfile-dev

docker build -t vms-prod:v1 . -f Dockerfile-prod

VideoManagementService viene sviluppato tramite l'utilizzo dei file relativi a k8

VideoProcessingService:

Componente che legge dalla coda kafka l'id del video da processare, scritta in precedenza dal componente VideoManagementService e lancia un thread per eseguire l'encoding del video caricato tramite lo script bash allegato che fa uso di ffmpeg.

Viene implementato il service VideoProcessingService con il metodo *processaVideo*, che lancia il thread per l'encoding del video.

Vengono creati i Dockerfile-dev e Dockerfile-prod e le relative immagini:

docker build -t vps:v1 . -f Dockerfile-dev

docker build -t vps-prod:v1 . -f Dockerfile-prod

VideoProcessingService viene sviluppato tramite l'utilizzo dei file relativi a k8

APIGateway:

Componente implementato come una applicazione SpringBoot (Spring Cloud Gateway).

Viene realizzato il controller APIController al path di base /videofiles, che espone un'unica richiesta GET al path /{id}/video.mpd, che restituisce il file video.mpd.

Il suo compito è quello di instradare le richieste. Creiamo una classe SpringCloudConfig in cui creiamo la route per indirizzare le richieste al VideoManagementService su /vms/*.

L'API Gateway viene inoltre configurate in termini di timeout massimi e dimensione dei file massima in upload, nell'application.properties.

Viene creata la classe Filtro per fornire statistiche su:

- API richiesta (GET POST PUT ecc + URI)
- Payload size input
- Payload size output
- Tempi di risposta
- Eventuali errori
- Richieste al secondo

Vengono creati i Dockerfile-dev e Dockerfile-prod e le relative immagini:

docker build -t api:v1 . -f Dockerfile-dev

docker build -t api-prod:v1 . -f Dockerfile-prod

APIGateway viene sviluppato tramite l'utilizzo dei file relativi a k8

Prometheus:

È stato deciso di usare prometheus come mezzo per salvare le statistiche prodotte dal componente APIGateway, per far ciò abbiamo usato l'image prom/prometheus e aggiunto le relative dipendenze nel file pom di APIGateway. Abbiamo inoltre configurato prometheus in modo tale da avere come target l'APIGateway in modo da poterlo monitorare e catturare le statistiche.

Prometheus viene sviluppato tramite l'utilizzo dei file relativi a k8

Spout:

Componente che si occupa di richiedere i dati a prometheus relativi alle richieste al secondo e ai tempi di risposta, una volta ottenuti i dati può scriverli su una coda kafka in modo tale da farli leggere al componente SparkKafka.

Vengono creati i Dockerfile-dev e Dockerfile-prod e le relative immagini:

docker build -t spout:v1 . -f Dockerfile-dev

docker build -t spout-prod:v1 . -f Dockerfile-prod

Spout viene sviluppato tramite l'utilizzo dei file relativi a k8

SparkKafka:

Componente che legge dalla coda kafka che ha scritto il componente spout, tramite sparck legge uno stream di dati che andrà ad analizzare in base alle richieste.

docker build -t sparkstream:v1 . -f Dockerfile-dev

docker build -t sparkstream -prod:v1 . -f Dockerfile-prod

SparkKafka viene sviluppato tramite l'utilizzo dei file relativi a k8

Infine sono stati utilizzati kafka e spark con il master e i workers, sono stati sviluppati in kakfa tramite l'uso dei relativi file k8. Sono essenziali per il funzionamento di tutti i componenti che ne fanno uso. Inoltre è stato aggiunto un volume persistente che serve a mantenere i video in formato mp4 e mpd.