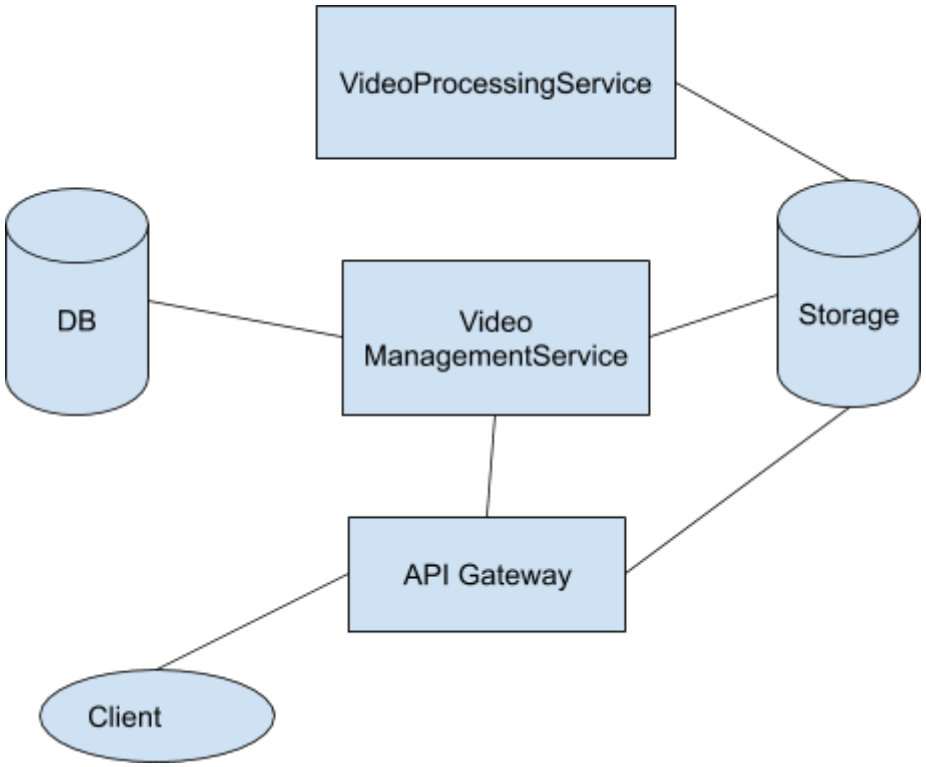
Progetto di Sistemi Distribuiti e Big Data
2019/2020

Homework 1

Per il seguente progetto si è scelto di implementare la variante (a):



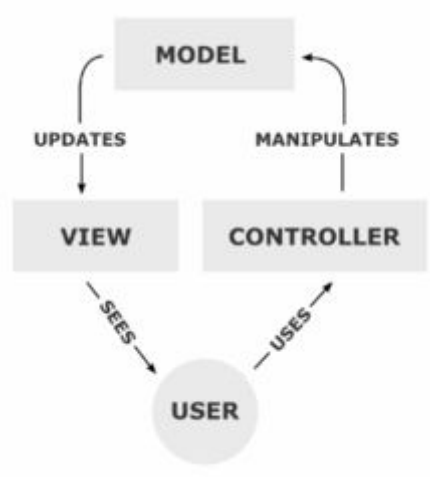
Di seguito vengono proposte le scelte sulle caratteristiche del sistema:

- DB - MongoDB
- API Gateway - Spring Cloud Gateway
- Statistiche raccolte: Stats 1 Prometheus
- Video Management Service - Spring boot
- Video Processing - Flask

Tutti i componenti sono stati implementati all'interno di container docker separati, quindi si verranno a definire 5 container differenti:

- VMS -> Video Management Service
- VPS -> Video Processing Service
- Gateway -> API Gateway
- Mongo -> MongoDB
- Prometheus

Inoltre è stato utilizzato il pattern MVC(Model View Controller), in cui la View è rappresentata dal formato dei messaggi HTTP scambiati dai vari componenti(JSON), il Controller risulta essere una classe definita all'interno di ogni componente e il Model sarà rappresentato dal Database MongoDB.



Video Management Service

Il componente Video Management Service risulta avere un'interfaccia REST che espone i seguenti URI:

1. POST /videos: prenderà in ingresso un messaggio in formato JSON in cui il client specifica il nome del video da postare e il nome dell'autore. Il servizio controllerà che l'utente risulta essere autenticato, che non esistano altri video con lo stesso nome; se le condizioni sono soddisfatte andrà a inserire all'interno del database le informazioni inerenti al video aggiungendo un riferimento all'utente che ha generato la richiesta. Nel caso in cui il video esista o l'utente non risulti essere autenticato il sistema risponderà con un messaggio di errore.
   Per implementare in Spring tutti i servizi REST si è scelto di utilizzare il RestController, all'interno del quale si è mappato un URI corrispondente a ciascun servizio. Una volta ricevuta la richiesta il

controller richiama una funzione inserita all'interno della classe *videoService* denominata *insertVideo*, la quale controllerà se esiste un video con lo stesso nome e in tal caso genererà un'eccezione custom chiamata *ExistingVideoNameException.*
Per la gestione del body della richiesta si è mappato il JSON in una classe Java chiamata VideoWrapper la quale servirà a memorizzare temporaneamente i dati del video che successivamente verranno salvati nella repository.
Per quanto concerne la sicurezza in questo URI come in tutti gli altri in cui si necessita autenticazione è stato utilizzato il plug-in fornito da Spring: *spring-boot-starter-security*, utilizzando una security basic in cui l'utente una volta effettuato il login utilizzerà un token per potersi autenticare alla richieste successive essendo quest'ultime stateless.

2. POST /videos/:id: prenderà in ingresso un file contenente un video MP4 controllerà se l'id presente nei parametri della richiesta risulta avere riscontri all'interno del database ed in questo caso salverà il video nel path indicato nella consegna.
   Una volta salvato il video il sistema invierà una richiesta a VPS passando come contenuto un JSON contenente l'id del video da elaborare, finita l'elaborazione cambierà lo stato del video in Uploaded.
   Una volta ricevuta la richiesta il controller richiama una funzione inserita all'interno della classe *videoService* denominata uploadVideo che prenderà come parametri un MultipartFile ed un videoId. La funzione controllerà se il file è vuoto e se esiste l'id del video all'interno del Database, in caso contrario genererà un'eccezione custom *NoVideoFileException*. Se il video esiste verrà controllato che l'utente che ha inviato la richiesta sia anche l'autore del video, nel caso in cui questa condizione non venga rispettata il sistema genererà un'altra eccezione *NoUserMatchException.* A questo punto il video verrà salvato nel path indicato dalla consegna, una volta salvata sarà inviata una richiesta a vps attraverso la funzione postForEntity contenente un JSON con l'id del video salvato. Se il vps risponde con un codice di successo lo status del video verrà cambiato in *uploaded* , in caso contrario verrà generata un'eccezione *VideoProcessingException.*
3. GET /videos: restituisce tutti i video disponibili nel database, non necessita autenticazione.
   Questo servizio non farà altro che richiamare la funzione *getVideos* della repository Video la quale ritornerà tutti i video presenti nel DB
4. GET /videos/:id : restituisce il file video.mpd del video corrispondente all'id passato nei parametri.
   Il sistema prenderà in ingresso l'id contenuto nei parametri della richiesta e chiamerà la funzione *getVideo* contenuta in *videoService* . GetVideo controllerà se il video è presente e se non lo è genererà

un'eccezione *NotExistingVideoException*, risponderà con un codice 301(Moved Permanently) indicando la locazione del video richiesto dall'utente.

5. POST /register registra l'utente che fornisce nome, email e password. Il sistema prenderà in ingresso un JSON contenente le informazioni dell'utente e richiamerà la funzione *createUser* contenunta in *UserService* che salverà l'utente nella repository User e quindi nel Database.

Video Processing Service

Il componente Video Processing Service risulta avere un'interfaccia REST che espone i seguenti URI:

1. POST /videos/process elabora il video passato nel body della richiesta.
   Per implementare questo componente si è scelto di utilizzare FLASK e per eseguire lo script di processamento del video si è scelto l'utilizzo di nodeJS. All'interno del container Video Processing Service si è inserita un'immagine Docker denominata package che contiene tutte le dipendenze necessarie per l'esecuzione dello script.
   Il sistema leggerà l'id passato nel body e lo inserirà come parametro dello script che elaborerà il video e lo salverà nel path indicato nella consegna.

API Gateway

L'API Gateway è stato implementato sfruttando Spring Cloud Gateway all'interno del quale è stato definito un route builder con 2 path: uno per reindirizzare tutte le richieste a vms e un altro per accedere allo storage condiviso a prelevare il video.mpd. Si è collegato Prometheus al container dell'API Gateway e attraverso la definizione di un GlobalFilter si sono esportate metriche custom per ogni URI. Come dimensione massima di file in upload si è scelto 10 MB.

Prometheus

Prometheus è stato utilizzato come strumento di storage per le statistiche.
Sono stati utilizzati tre tipi di metriche:

- Timer: risulta essere un timer per il calcolo delle latenze. Prometheus offre 3 diversi campi per ogni timer: un contatore, un campo in cui viene salvato la max latenza ed un campo in cui viene salvata la somma delle latenze

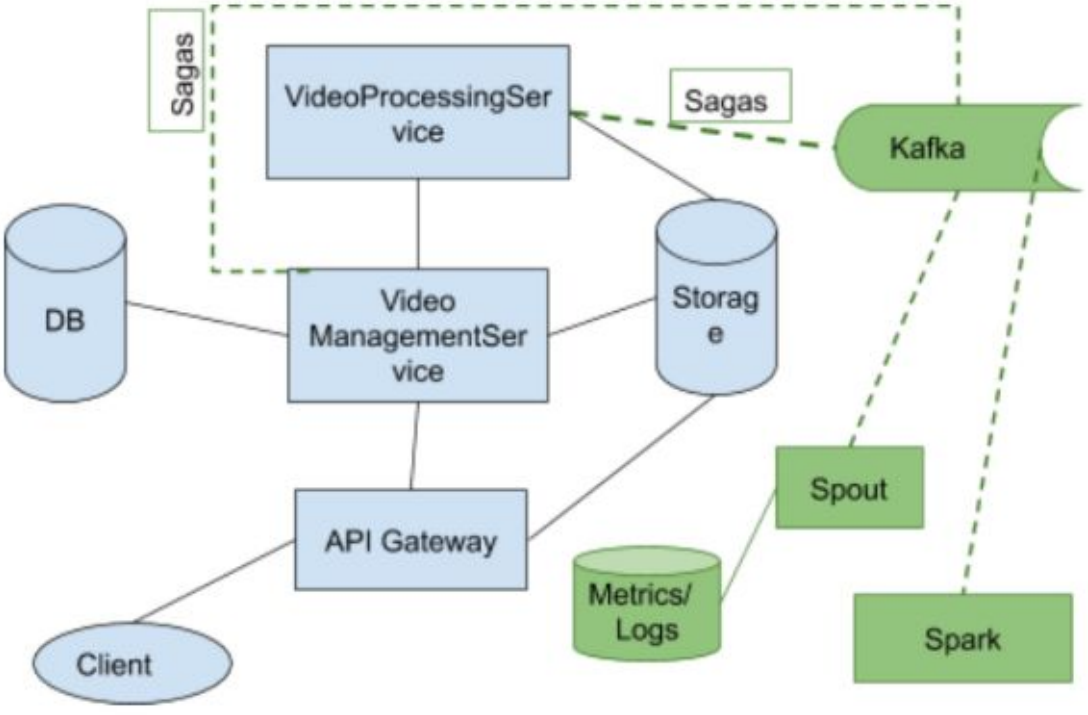
- Counter: un contatore per il conteggio delle richieste
- DistributionSummary: una metrica per salvare la dimensione delle richieste. Similmente a Timer offre 3 campi: un contatore, un campo in cui viene salvato la max dimensione ed un campo in cui viene salvata la somma delle dimensioni

Storage Condiviso

Per lo storage condiviso si è creato un volumes che conterrà due cartelle: var e videofiles. La prima conterrà i video in formato mp4 la seconda i video di output dello script di elaborazione. Nel docker-compose lo storage risulta essere condiviso del vari componenti che devono salvare dati permanenti.

Homework 2

Di seguito viene proposta la seconda parte della consegna:



Inizialmente si è effettuato il porting da Docker a Kubernetes inserendo ogni container in un Pod differente eccezion fatta per Prometheus, il quale condivide il Pod con l'API Gateway. Oltre ai Pod sono stati definiti i servizi per abilitare la comunicazione tra i vari componenti e la comunicazione tra il client e il punto di ingresso del sistema costituito dall'API Gateway.

Kafka

Per l'implementazione di Kafka è stato elaborato un nuovo pod all'interno del quale è stata inserita l'immagine docker di kafka: `wurstmeister/kafka:2.11-2.0.0`, insieme all'immagine dello zookeeper: `library/zookeeper:3.4.13`.
Una volta inserito kafka, lato vms grazie all'utilizzo delle dependencies: `spring-kafka`, si è elaborato un produttore kafka ed un consumatore su topic differenti, secondo le istruzioni dell' homework. Lato vps si è effettuato lo stesso procedimento grazie al modulo kafka-python. Per quanto concerne Sagas, lato Spring sono state utilizzate le annotazione @Transactional, lato VPS è stato definito un metodo di rollback che in caso di failure faccia tornare il sistema allo stato precedente l'elaborazione. Infine si è scelto di escludere dal vps FLASK, poiché il componente non comunicherà più attraverso API REST, ma sfruttando Kafka con una comunicazione asincrona.

Spout

Con il modulo Spout si effettua il retrieving delle statistiche per l'elaborazione attraverso Spark. E' stato utilizzato python come linguaggio e attraverso il modulo requests è stato possibile contattare le API esposte da Prometheus per riuscire ad estrarre le statistiche. Quest'ultime vengono inserite in una coda kafka sotto forma di stringa nel formato (name:example|uri:example|value:example,). I worker spark prenderanno i dati da questa coda ed eseguiranno le elaborazioni richieste.

Spark

Con il modulo Spark si effettuano le elaborazioni sulle statistiche. Attraverso Spark Streaming vengono prelevati i dati da una coda Kafka e generati Dstream di 30 secondi. Per ogni Dstream vengono effettuate le seguenti trasformazioni:

- Vengono separate le singole statistiche
- Viene effettuata la somma delle statistiche in cui è presente il tempo di risposta
- Attraverso le funzionalità di windowing viene fatta la differenza delle statistiche con il batch precedente in quanto incrementali(dato che le statistiche contengono la somma delle latenze per ottenere le latenze di un singolo batch sarà necessario effettuare la differenza con il batch precedente)
- Viene calcolato il tempo medio di risposta e le richieste al secondo
- Attraverso le funzionalità di windowing viene confrontato il valore calcolato al punto precedente con i valori dei 10 batch precedenti
- Se il confronto viola le regole imposte nell'homework viene inviato un alert in una coda kafka che verrà consumato dal componente AlertManagement