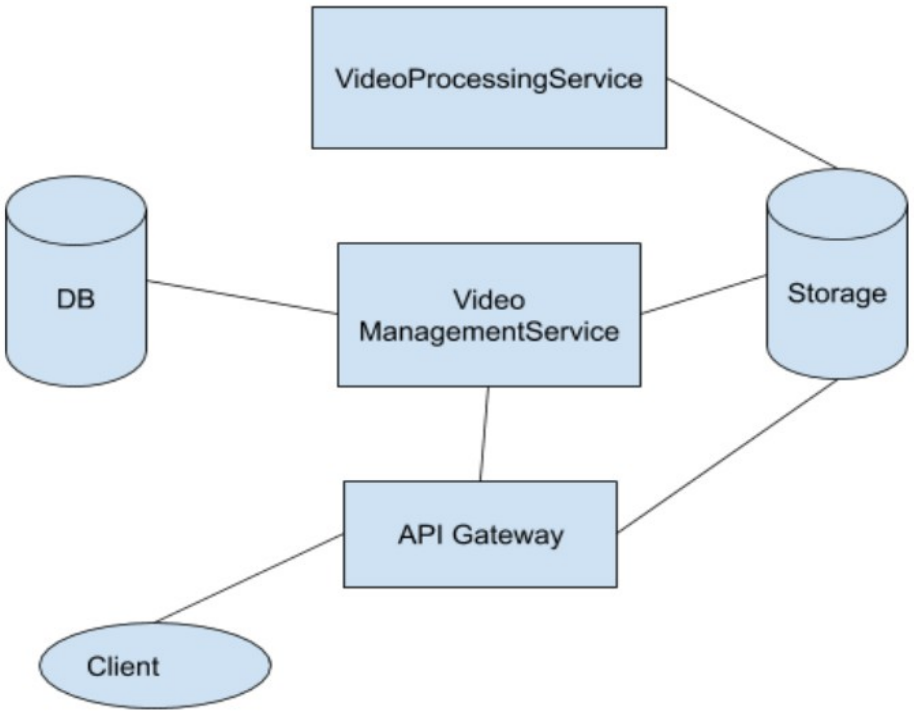
Distributed System And Big Data a.a. 2019/2020
Relazione Descrittiva HW – 1

Raiti Mario O55000434
Nardo Gabriele Salvatore O55000430

IL progetto preso in esame nella seguente relazione è la versione (a) , cioè il sistema Video Server. La variante presa in considerazione è (a).DB1.GW2.STATS1. Quindi si è scelto di utilizzare come DB MYSQL , implementare l'API GW usando Spring Boot , e le statistiche collezione sono quelle relative al modulo API GATEWAY.
L'architettura è descritta dalla figura in basso.

(a) Video Server

Implementare un sistema di gestione per video in formato MPEG-DASH.



Per la variante in esame non è previsto l'implementazione del client. Di seguito verranno fornite descrizioni dettagliate delle scelte implementative relative ai moduli presenti in figura. Come richiesto per ogni modulo è stato creato un apposito container , quindi uno specifico docker-file per ogni immagine. Inoltre ogni progetto Spring Boot utilizza anche Maven.

Video Management Service – VMS

Il VMS è stato realizzato come applicativo Spring Boot. Si occupa di gestire tutte le richieste inoltratagli da ApiGW , quindi il suo compito primario è quello da fungere come controller per le REST API richieste da progetto. Utilizza come dipendenze i seguenti moduli: Spring Security per la gestione della Basic Authentication realtiva alle due POST su /videos , data-jpa e mysql connector per la gestione della

persistenza e l'interazione col db MYSQL. All'interno del modulo troviamo tre tipologie di classi organizzate nei seguenti gruppi : Entity , Controller e Servcies. Le classi che fanno parte di entity sono state realizzate per definire gli oggetti che vengono mappati sul db attraverso le annotazioni di spring, le entità realizzate sono state user e video e i ripsettivi CRUD repository per l'interazione col db. La classe services contiene dei metodi che permettono di disaccoppiare le istruzioni di interazione con db dalla diretta gestione della chiamata REST. Tali istruzioni vengono poi richiamate all'interno dei controller. Come detto prima i controller si occupa della gestione delle API REST richieste da progetto. In particolare si prende in esame la classe video controller che si occupa di gestire le chiamate indirizzate al path /videos.
Di particolare interesse è la gestione della POST su videos/:id , che si occupa di creare il path nel volume condiviso e caricare tale video al path var/video , inoltre attraverso un RestTemplate e una PostForEntinty viene creata e inoltrata la POST al modulo VPS per l'elaborazione del video. Il servizio è esposto alla porta 8080 e l'immagine di riferimento utilizzata in docker è java:8.

Api Gateway – ApiGW

ApiGW è stato realizzato come applicativo Spring Boot , utilizzando per il routing la dipendenza Spring Cloud Gateway. Inoltre sono state inserite come dipendenze i moduli Actuator e io.micrometer al fine di raccogliere ed esporre le metriche e permettere la visualizzazione su prometheus. All'interno del file *GatewayRoutes* sono state definite le regole di routing custom (indicate nelle richieste di progetto) attraverso la funzione *RouteLocator.* All'interno del file application properties sono state definite le configurazioni sui vincoli temporali del modulo e la dimensione massima dei file in upload. Sono inoltre state definite un set di impostazioni per permettere l'esposizione delle metriche tramite prometheus.
IL servizio è esposto alla porta 8070 e rappresenta l'unico punto di ingresso al sistema dal mondo esterno. La porta 8070 viene infatti esposta nella clausola port del docker-compose file. Come detto precedentemente le metriche relative al modulo vengono esposte all'uri http://apigw:8070/actuator/prometheus , che viene scelto come target del server prometheus che si occupa di consumarle e visualizzarle tramite la sua dashboard , visitabile alla porta 9090 (porta esposta tramite compose).
Come immagine di riferimento per il dockerfile viene utilizzata java:8 , e l'immagine prom/prometheus per il server prometheus.

Video Processing Service – VPS

Il VPS è stato realizzato come applicativo Spring Boot. E' prevista una classe controller chiamata ScriptController, che si occupa di gestire le richieste provenienti dal VMS per quanto riguarda il processing dei video. Tali richieste vengono inoltrate dal VMS come POST al path /video/process con body un JSON con l'id del video da processare. La gestione del processamento avviene mediante la creazione di un thread (attraverso la classe ScriptExecutor che implementa Runnable), che lancia un process builder che esegue il comando shell di ffmpeg. Il thread attende l'intero

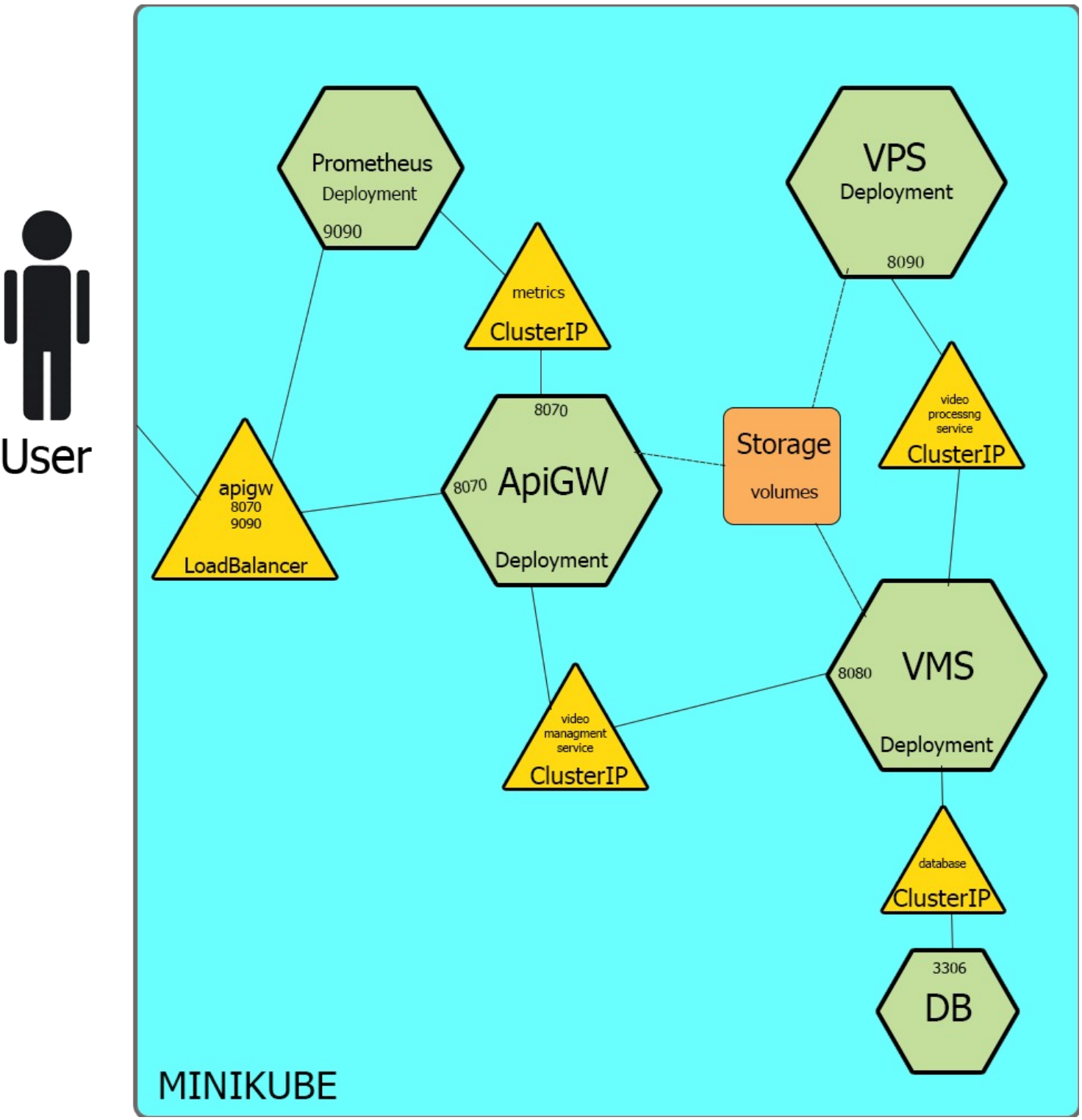
processamento prima di concludere e ritornare la risposta al VMS. I chunk prodotti dal processamento sono salvati sullo storage al path */app/var/videofiles/'id_video'*, che viene creato se non esiste. Per quanto riguarda il dockerfile come immagine di riferimento è stata scelta *adoptopenjdk/openjdk8:alpine* , per permettere di installare la versione 4 di ffmpeg per un corretto funzionamento dello script. Il servizio è stato esposto alla porta 8090.

Storage
Tale modulo è stato realizzato come volume condiviso tra i vari moduli, definito attraverso il docker-compose file. È stata montata la directory Storage al root path var. tale directory conterrà due sotto directory video e videofiles , che conterranno i video caricati e i file risultato dell'elaborazione.

È stato inoltre realizzato il porting su K8s. Sono stati definiti Service e Deployment object. I service definiti sono :

- **videoprocessingservice** : di tipo clusterIP , espone la porta 8090 di vps per renderlo accessibile alle richieste inoltrate dal modulo vms;
- **videomanagementservice** : di tipo clusterIP , espone la porta 8080 di vms per renderlo accessibile dalle richieste inoltrate da apigw;
- **metrics** : di tipo clusterIP , espone la porta 8070 di apigw per renderlo accessibile dal modulo prometheus per la raccolta delle metriche;
- **database** : di tipo clusterIP , espone la porta 3306 del database per renderlo accessibile del modulo vms.
- **apigw** : di tipo LoadBalancer , espone sia la porta 8070 del modulo apigw per renderlo accessibile dalle chiamate esterne, sia la porta 9090 di prometheus per visualizzare le metriche tramite il server.

È stato definito un oggetto di Deployment per ogni modulo precedentemente descritto. Anche in questo caso, in ogni modulo è stato montata la directory Storage come volumes condiviso. Il tipo di volume scelto è una semplice directory. Le configurazioni del DB e del VMS sono state passate in modo statico passando le variabili d'ambiente direttamente nel file di deploymnet senza usare configMap o secrets.


User
Prometheus
Deployment
9090
metrics
ClusterIP
VPS
Deployment
8090
video
processng
service
ClusterIP
apigw
8070
9090
LoadBalancer
8070
8070
ApiGW
Deployment
Storage
volumes
video
managment
service
ClusterIP
8080
VMS
Deployment
database
ClusterIP
3306
DB
MINIKUBE