VideoServer_MPEG-DASH

This project implements a management system for videos in MPEG-DASH format. It was created for educational purposes at the **University of Catania**. We faced some of the most prominent technologies in the field of distributed systems and big data analysis, like **Docker** for containerization, **Kubernetes** as Orchestrator, **Apache Kafka** for building real-time data pipelines, and**Apache Spark** for big data processing. We realized the whole project using the potential of **Spring Boot**, an open-source Java-based framework used to create microservices. The goal is to realize a distributed Video Server application that exposes REST API that allows users to authenticate, upload a *video.mp4*, and get the URL to stream it using**ffplay** or **vlc** networks feature.

Getting Started

These instructions will get you a copy of the project up and running for development and production purposes. See deployment for notes on how to deploy the project.

Prerequisites

- **Docker** daemon
- **Docker-compose**
- **Minikube** to run a single-node Kubernetes cluster inside a VM on your laptop with 5GB of Memory and kvm2 as Driver.
- **Apache Spark**
- **Curl** or **Postman** (up to you)
- **FFplay** or **VLC** (up to you)

Installing

Clone the repository

1) HMW-1

Go to videoserver-HMW1 folder and type:

```
$ docker-compose -f "docker-compose.production.yml" up
```

After you finished to play on it you can type:

```
$ docker-compose down
```

2)HMW-2

Go to videoserver-HMW2 folder and type:

To start Minikube VM on your laptop

```
$ minikube start --memory=5096
```

Then, enable nginx ingress controller

```
$ minikube addons enable ingress
```

To link your host docker daemon with Minikube

```
$ eval $(minikube docker-env)
```

Make sure you haven't any service or deployment up yet

```
$ kubectl get all
```

Go to *k8s/production/* folder and run**create-configmaps-secrets.sh**

```
$ ./create-configmaps-secrets.sh
```

If you don't have permission type:

```
$ chmod 755 create-configmaps-secrets.sh
```

Now that you have created ConfigMaps and Secrets, run the following to deploy the microservices on the cluster

```
[production] $ kubectl apply -f kafka/ -f videomanagementservice/ -f videoprocessingservice/ -f spout/ -f proxy/
```

To deploy **spark** create a service account named *'spark'*. *Alert*: The service account credentials used by the driver pods must be allowed to create pods, services, and ConfigMaps.

```
[production] $ kubectl apply -f spark-on-k8s-rbac.yaml
```

After this run *spark_exec.sh* in *videoserver-HMW2/spark/* folder

```
$ ./spark_exec.sh
```

When you want to shut-down the cluster:

```
[production] $ kubectl delete -f kafka/ -f videomanagementservice/ -f videoprocessingservice/ -f spout/ -f proxy/
```

To stop the spark component, type

```
CTRL^C on spark terminal window
```

Finally

```
$ minikube stop
```

How to use

- **POST /register** wants a json as body like this: {"email": "MJ@gmail.com", "name": "Michael", "surname": "Jordan", "password": "1234"}
- **POST /videos** wants a json as body like this: {"video_name": "Cat", "author_name": "MJ"}

Deployment

We faced the whole project by splitting it into two phases:

1. Deploying the main components (nginx, vms, vps, vsDB, vsStorage, vsStats) using docker-compose. FOLDER: videoserver-HMW1
2. Porting the project on Kubernetes and adding the remaining components (Kafka, Spout, Spark). FOLDER: videoserver-HMW2

1)

Imgur

- We used **nginx** like API Gateway to route the REST API requests from the client to vms and the **vsStorage**. The latter has two main paths inside: */var/video/** and */var/videofiles/**. To be specific nginx route all the traffic on */vms/** to vms and all the requests on */videofiles** to the root dir */var/videofiles**. Furthermore, we set up nginx to accept videos not larger than 5MB in upload with a timeout of **90m**.
- **vms** exposes the following REST API:
    - 1 GET */ping*
    - 2 POST */register* [no-auth]
    - 3 POST */videos* [need-auth]
    - 4 POST */videos/:id* [need-auth]
    - 5 GET */videos*
    - 6 GET */videos/:id* (returns the encoded video URL)
    - 7 GET */videofiles*
    - 8 GET */videofiles/:idvideo_folder* (to access video encoded files)
- **vps** exposes the following REST API to vms:
    - POST */videos/process*

The latter starts a thread that calls the encode() method which runs **FFmpeg** (videoEncoder) script that encode the video .mp4 uploaded.

- We used Mysql for the **vsDB**, which stores all the users and the video's metadata.
- *vsDBclient* collects the statistics and saves them in a *.txt* file named *stats.txt* to the **vsStats** storage. The previous step was made using **stats.sh** bash script, which interrogate *vsDB* every 10 seconds.

The **statistics** collected are:

- Query type
- Query Latency
- Errors Occurred
- Query per second
- Resources usage (CPU usage time - Memory usage)

- Payload Size input/output

HMW1 - Project Choices

We used custom image as a base to build *vps* Dockerfile m1c0l/alpine-openjdk-ffmpeg

Containers properly isolation ensured by networks concept introduction. To be specific we decided to realize different networks

- ***apigw* network** links *nginx* and *vms*
- ***db* network** links *vsDBClient*, *vsDB*, *vms*
- ***vpsnet* network** links *vms* and *vps*

We decide to add a further volume named **vsStats** to store *vsDB* stats.

To allow the correct execution of vms, we used wait-for-it.sh script.

- We realize the thread using a *taskExecutor*

2)

Imgur

We ported *HM1-project* on **Kubernetes**. In this version *vms* and *vps* talks to each other through a **Kafka** queue with "**main-topic**" as topic.

The **spout** component reads *stats_$.txt* in input from vsStats, filters the following stats:

- Query Type (Com_select, Com_delete, Com_update, Com_insert)
- Queries per Second (Queriespersec)

and sends them in output on a Kafka queue with topic equal to "**stats-topic**", every 10 seconds.

The output is something like this:

> numfile|stats_n|query|value

- **Spark** component is a subscriber of "**stats-topic**". It consumes the stream in input at batch pace. Each batch is 30 seconds large.

HMW2 - Project Choices

- Ingress linked directly to **nginx** service instead of *vms* service.
- **vs-db** pod contains two containers:
  - mysql
  - mysqlclient
- We decide in order to handle stats through **Kafka** queue to create one **stat_***$number* **.txt** file every 10 seconds.

- We used Java CompletableFuture to manage in an asynchronous and concurrent way, the threads used to encode the video uploaded.

Issue - notes

Even if *spark* folder is present inside the project, this component isn't functional at all. It's only a functional subscriber of "stats-topic". To be precise, following the instructions we made, the driver and the executors are created in the correct way and you can see Kafka messages on logs, but we faced some issues in testing DStreams functionality. Between these, we had hardware limitations that caused crashes during deployment. Even though we didn't face any issue in printing only the DStream received, we faced a crash when we added even the simplest DStream management because of a lack of CPU.

Nonetheless, we tried to write some code that seemed to us logically correct, but we didn't test it.

Built With

- SpringBoot - Java-based framework
- Maven - Dependency Management
- Docker - Used for Containerization
- Kubernetes - The Orchestrator
- Apache Kafka - Used to create streams
- Apache Spark - Used for processing statistic streams

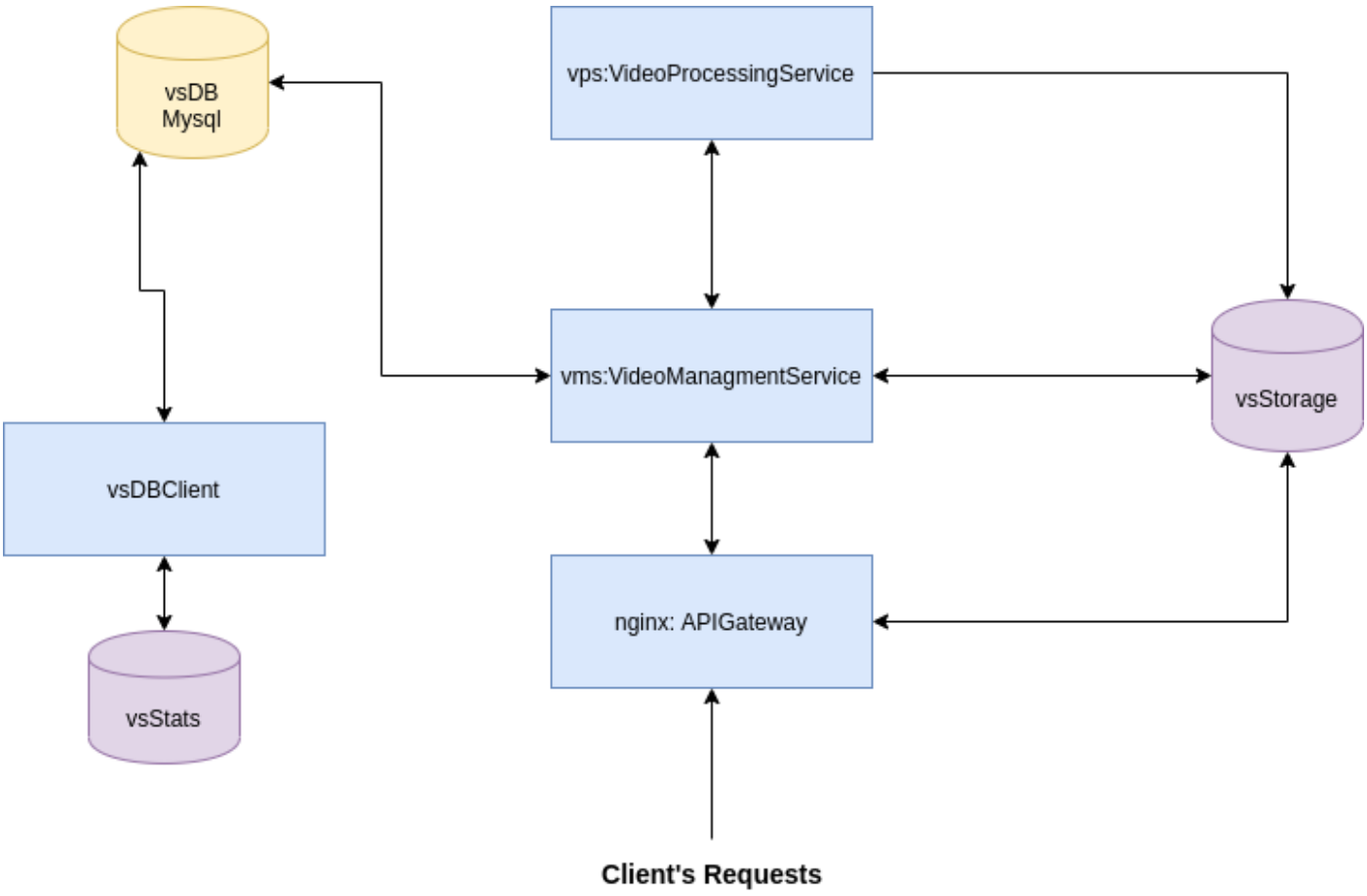
All the docker images used can be found onm1c0l's DockerHub

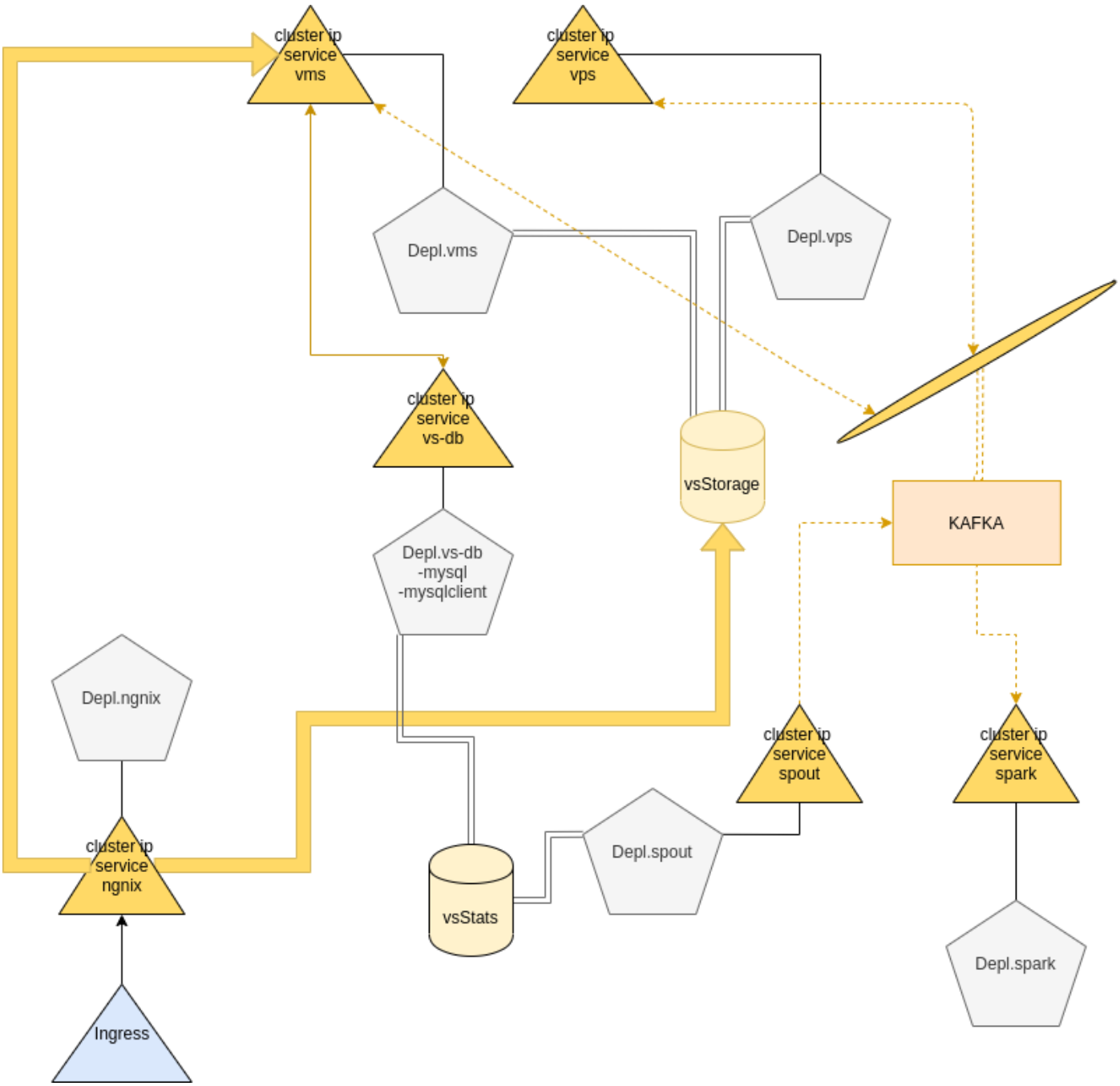
Versioning

We used Git for versioning. For the versions available, see thetags on this repository.

Authors

- **Michele Cannizzaro**
- **Michele Grasso**


vsDB
Mysql
vps:VideoProcessingService
vms:VideoManagmentService
vsStorage
vsDBClient
vsStats
nginx: APIGateway
Client's Requests


cluster ip
service
vms
cluster ip
service
vps
Depl.vms
Depl.vps
cluster ip
service
vs-db
vsStorage
KAFKA
Depl.vs-db
-mysql
-mysqlclient
Depl.ngnix
cluster ip
service
spout
cluster ip
service
spark
cluster ip
service
ngnix
Depl.spout
vsStats
Depl.spark
Ingress