VIDEO WORLD APP-A video search engine for students

"Video world" is an app I'm designing. The idea is to create a search video engine for students. So one of the requirement is that explicit video should not be uploaded. If so, the system will automatically block the video upload. Students will be able to search for their favourite videos by typing description and titles. The videos are only related with school subjects or culture matters.

Functional requirements:

1) Video description text and title storage (strored in a text-optimized cloud database)

2) Videos upload

3) Video research

4) Video resolution and conversion

5) explicit content checking.

6) video view

*Daily active users: more than 10 .000.000 students.

Non-functional requirements:

*resiliency

*scalability

*security

*cost effective

*fastness

Normally, all of these functional requirements should be processed in same time in parallel, excepted view feature that should be process by the final user. But in order to simplify, the process, we will design these features step by step.

1) Text description / Title storage

The producer of the video will be able to connect to the website, and enter in a form: the title, the description etc ...., in short all the details relating to the video. Once the form is submitted, an API will trigger a function that will store the data in a large database optimized for text search.

Since the application is global and consulted worldwide, the API should resist a significant demand, otherwise the system will suffocate. Also we will set a throttling limit in order to prevent any user to hit the button many times. Let's say a limit of 20. The API server should be automatically scalable, reliable. We will use Amazon API Gateway as API servers because it's cost efficient, reliable, serverless and

automatically scalable. We will configure a REST API to achieve this goal. We prefer a REST API because with REST APIs client and server are independent, the data storage along with the UI are independent, they are more scalable, cost efficient and flexible.

We will use a serverless function as service and write a python script that will copy the video details and store them in the Database. AWS lambda is a good fit, because it's simply of use , serverless, cost effective, fully managed, reliable and automatically scalable to process millions of demand in the same time.

The target database for text description and titles storage should be text-optimized. It should also be NoSQL because of the unpredictable traffic expected. It should also be fast, reliable and cost effective. For that reasons, we choose Opensearch Serverless to handle this feature.

For security purposes, we will use :

- Amazon cognito for session user and federation Identity
- AWS KMS for data security both in transit than at rest
- AWS IAM for roles and authorization to access different services.

The following design represents the above-mentioned module:

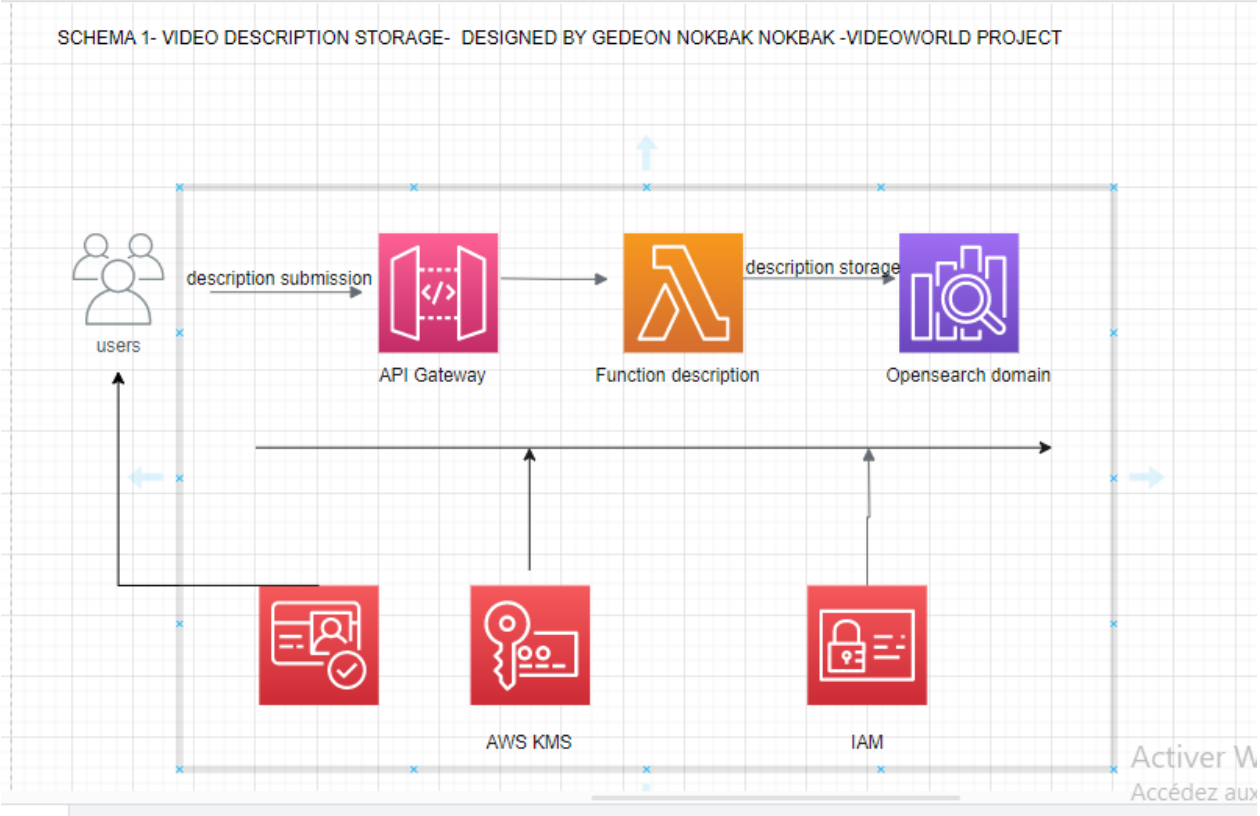

Fig.1 video description storage

2) video upload + explicit content checking

The first step is to create a video stream from a device such as a CCTV camera. The device will be linked to a cloud application such as Amazon kinesis Video Stream, which in turn will be connected to a durable, scalable, reliable and cheaper cloud storage system. We will choose Amazon S3 as the storage system, because it meets all the criteria mentioned above. We will call this bucket "storage1", we will define life policy rules that will send videos to a different storage class after a number of days. It is very important in order to limit expenses, because when a video is uploaded in this bucket it is no longer accessed. It's just an intermediate bucket.

Every new PUT in the previous bucket will trigger a lambda function called "lambda_to_recognition" that in turn will call a Machine learning service trained to recognize adult or explicit content. We will choose Amazon Recognition as the machine learning service because it is reliable, easy to use, have solid ML algorithms and scalable.

Recognition is connected to two lambda functions respectively called "lambda_recognition_fail" and "lambda_recognition_positive".

After analyzing the video and in the case of explicit content, Recognition will start the "lambda_recognition_fail " function. The latter will communicate with an API to inform the user of the explicit character of his video. The process is blocked and the video could not be uploaded to the system.

Otherwise, if the Recognition service doesn't detect anything bad, the "lambda_recognition_positive" function will trigger an orchestrator module that we will see next.

The following schema 2, shows how to design the module in the AWS cloud:

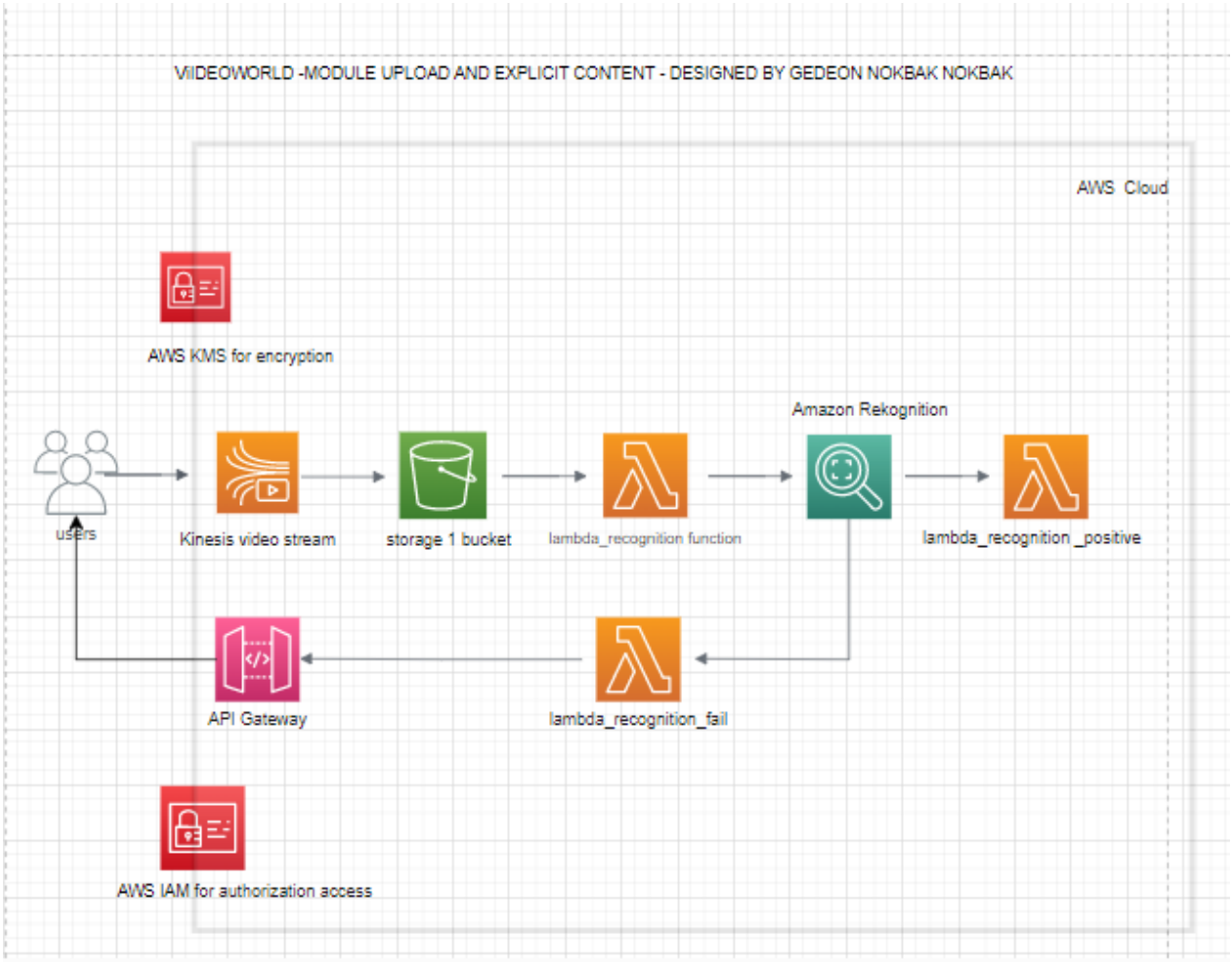

Fig.2. video upload part1 + explicit content checking

Let's continue with our video upload process, the second part. Assuming the recognition process is successful and the "lambda_recognition_positive" function is triggered. What next? Actually, we have to convert the 4k files format in many different resolution types (240p, 360p, 480p, 560p, 720p,930p, 1080p…..). This process should be orchestrated in parallel and at the same time. The different formats must be produced at the same time. It is important to produce different resolution formats because not all cameras are able to support the high resolution provided by cameras. For orchestration process, we will use the Amazon Step Functions service because it is serverless, fully managed and easy to use. For resolutions, we will use Amazon Elemental Media Converter.

Amazon Elemental Media Converter is very useful here, because apart the fact that he can convert video in different format, it can also support HLS encoding as input. We will deep dive in later.

At the end of the process, the converted videos are strored in an S3 bucket called "strorage final".We will define life cycle policies rules so that videos not frequently accessed should be moved in cheaper storage class. Let's set a threshold of 40 days. It's very important to convert video in different resolution styles

before stroring them, because if we start the conversion process when the video is needed, it will last long time and the user will get discouraged.

Now let's design the second part of the video upload module:

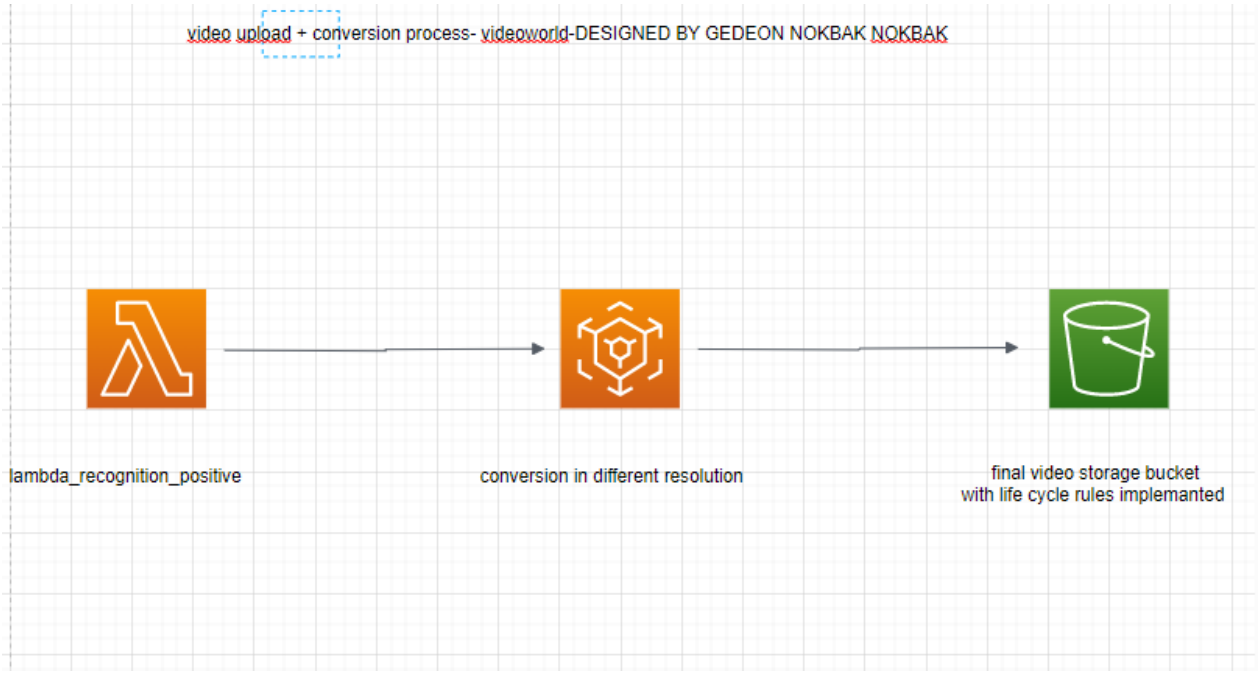

Fig.3. video upload and conversion processes

3) Video research and view

Now this is the main module of our project. The end user, our students should be able to search for a desired video. Let's say some one type in the search bar "thales theorem". An API will send the desired description to the system and a comparison should be made with the database opensearch.

But now because our application is global and worldwidely used, our user can make a request in a different AZ, we need to set a content-delivery network in order to deliver the video as close as possible the user AZ, why not in the same AZ. Also some video are frequently viewed, we can not allow the system to search for them each time they are asked otherwise the system could suffocate. The solution is to use a cache module before the video storage. If any user has already searched once for a search term, its description and path to the video are memorized in the cache. We will use Amazon CloudFront as content delivery network because it matches all the above-mentioned features. We will also set some distributions to many AZs.

If the tags are not already asked, an Api will trigger a lambda function that will get the videos IDs of all the videos which tags and description are closely related or equal to the student search within the same AZ. The CDN will store the data and will find in the bucket the videos desired and display them in the browser.

Let's design the previous module in the AWS Cloud:

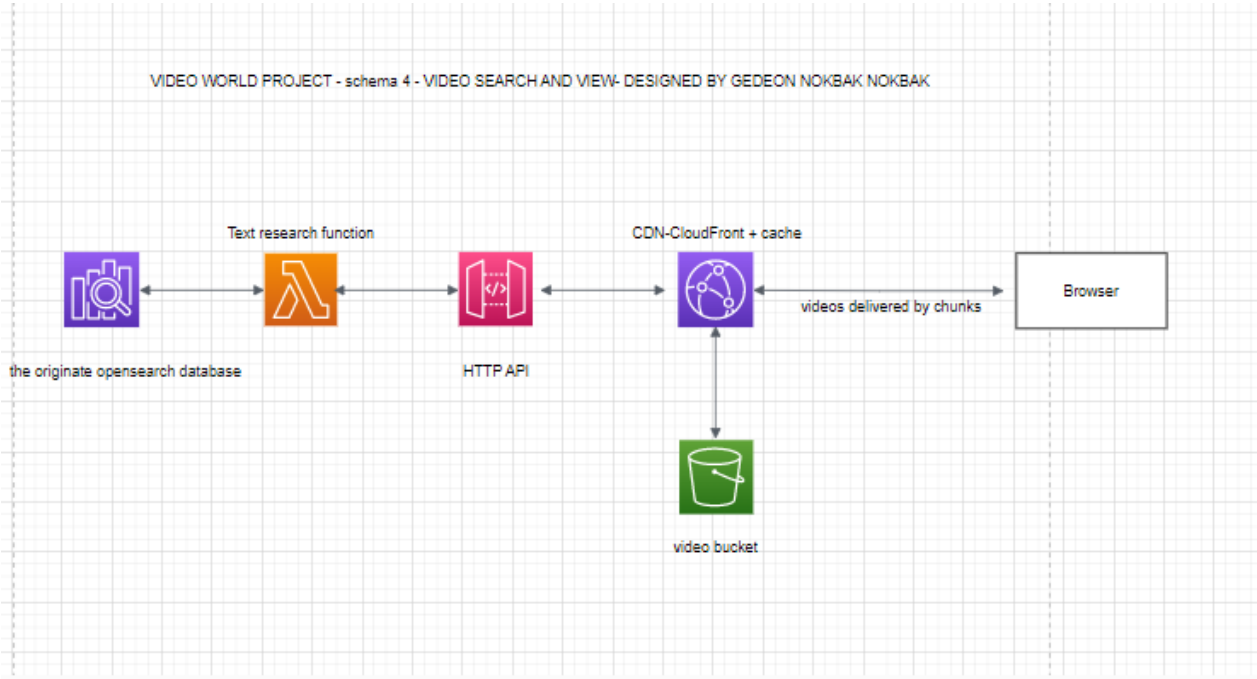

Fig.4. video search and view module