**Optimized online proctoring using Amazon Rekognition &** **multi-region distribution processing**

An intelligent online proctoring solution is the need for the hour for conducting online exams via web & mobile devices. This solution should be capable of handling the invigilation of 50K students who takes exams at one time simultaneously, with enough intelligence to handle all sorts of malpractices and enforcing fair and well proctored examination system, for university and corporate examinations.

Need for such a cutting-edge solution became highly critical especially in times pandemic, which brought huge relief to students and test providers. Earlier, our client Ginger was using monitoring of students via screens, which was having issues of manual monitoring

**Use case overview**

At the time of student’s initial login, when authentication and authorization take place, students’ information is captured in form of multiple images at fractional time differences. At the time of student authorization, these images are sent to cloud for processing and training the rekognition model.

During the online examination, every 10 seconds student’s image is captured and sent to cloud. As Rekognition API has a region-specific Transactions per second (TPS) limit which was creating a huge problem i.e. North-Virginia has 50, Ireland has 50, Mumbai has 5 etc. and students were in tune of 50K so this problem resolved using a analytical function which uses various region wise weights and corresponding probabilities etc.

Using analytical function compare and detect requests of faces are distributed to multiple regions.

* In detect operation looks for multiple faces, if multiple faces found then displayed at web application window to proctor for further action.
* In compare operation looks for match with initial registered student, if not matched then this is displayed at web application window to proctor for further action.

**Solution overview**

Diagram

Description automatically generated

Fig 1. Architecture Diagram

* **Amazon Rekognition**

Amazon Rekognition is designed to work seamlessly with other AWS services like Amazon S3 and AWS Lambda. We pay for the images and videos that we analyze, and the face metadata that we store thus reducing the cost of using the AI services.

* **Retry and Exponential Backoff during any failure of an API request**

In case of any failed request with AWS, we can retry for several times if any request fails due to any internal error

* **AWS API Gateway secured single API request along with Cost saving**

API Gateway offers optimized serverless workloads and HTTP backends—they offer up to 71% cost savings and 60% latency reduction. It also helps us manage traffic to your backend systems, so we are free to focus on our business logic and services rather than maintaining infrastructure

* **AWS Lambda helped to create multiple API requests internally**

AWS Lambda lets us run code without provisioning or managing servers. We pay only for the compute time you consume. Also, connect with various AWS API’s to acquire various insights from the inputs

**Methodology**

Our Solution approach is multistep as follows,

1. We create a secure API using AWS API Gateway to receive two images from the user.
2. The two images are

* **Source image** – The image of the present frame.
* **Target image –** The image of the registered user.

1. We create a function as a service using AWS Lambda to receive the images from the API.
2. The Lambda function implements the logic of spreading out the requests to the AWS Rekognition API to different regions in order to handle the per region TPS limit of Rekognition API.

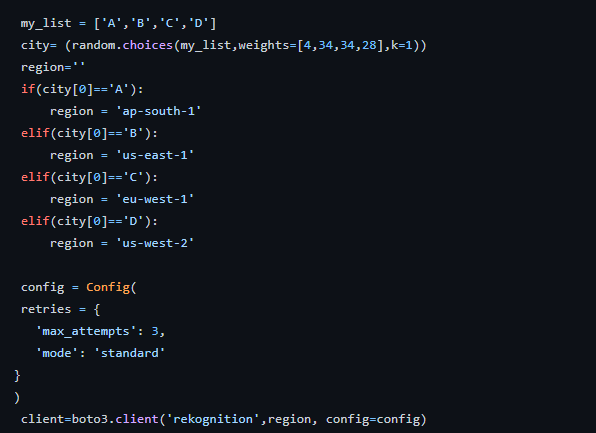


Fig 2. Weight distribution

1. First the source image received from the API to the lambda is sent to detect\_faces API.
2. The detect\_faces API counts the number of faces present in the source image.
3. The number of faces present in the source image decides how each of the case is handled.
   * **If there is one face –**

Then the compare faces API is called with the source and target image as the parameters.

* + **If there is no face –**

An error message “No Faces” is sent back to the API and the image is saved in a predefined S3 bucket.

* + **If there is more than one face –**

An error message “More than one Face” is sent back via the API and image is saved in predefined s3 bucket.

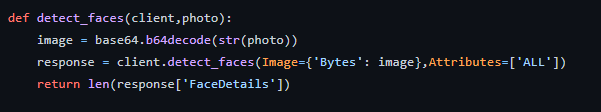


Fig 3. Calling the detect faces API

1. The compare faces API receives the source and target image to make sure the person presents in the frame i.e. the person taking the exam is same as the person who registered for the exam.
2. The faces between the source and target image are compared.
3. If the faces match, then “success faces match” is returned via the API.
4. If the faces do not match, then “error faces do not match” is sent back via the API and the source image is saved in a predefined s3 bucket.

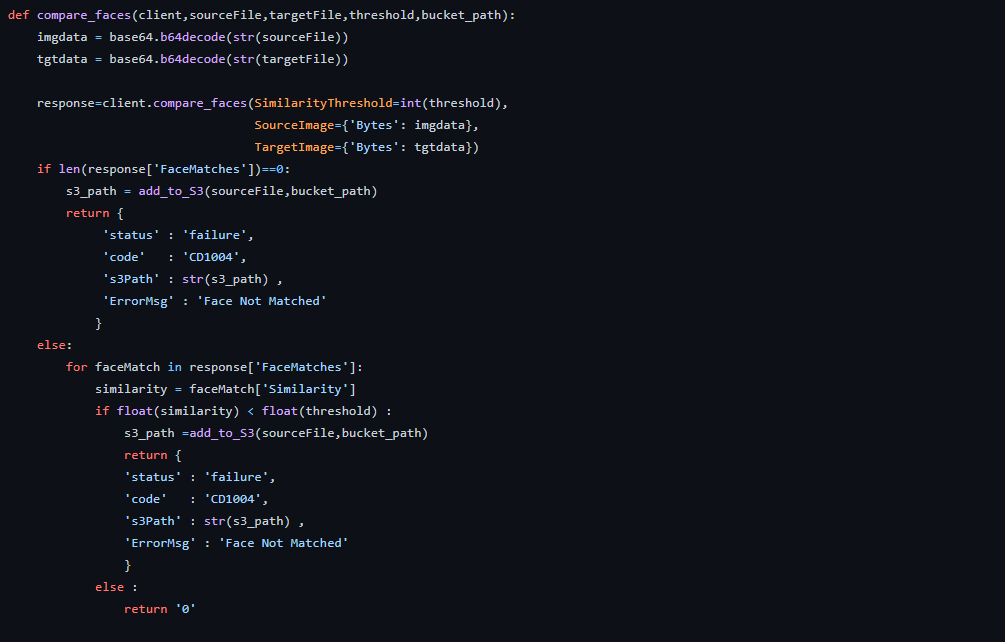


Fig 4. Calling compare faces API

**Additional Considerations**

The following are the major issues that we faced during the implementation of this solution and the below table of test cases make sure your deployment is robust.

* **Sending images to lambda –**

We had to find a way to send the source and the target image to the lambda function that can then be sent to compare faces api of AWS rekognition. We first sent the images in base64 encoded format to the lambda via API gateway. Later at the lambda we used base64 library and its base64.b64decode () function to convert it back to an image.

* **Overcoming Transactions Per Second(TPS) limit –**

In order to process a total of around 30 thousand images per second we had to find a way to overcome the per region limit of rekognition API. We created a weighted random generator to spread the requests between multiple AWS regions.

|  |  |  |  |
| --- | --- | --- | --- |
| **Description** | **Test Data** | **Expected Result** | **Actual Result** |
| When there is no person present in the frame | An image in which no person can be seen in the frame of the image | ‘No Face’ error message and the frame should be saved in s3 bucket | ‘No Face’ error and inside the s3 bucket the image that we passed |
| When there are more than one person present in the frame. | An image in which two persons can be seen in the frame of the image | ‘Multiple Face’ error message and the frame should be saved in s3 bucket | ‘Multiple Face’ error and inside the s3 bucket the image that we passed |
| When someone other than the person who registered for the exam is present in the frame | An image in which someone other than the registered person can be seen in the frame of the image | 'Face Not Matched 'error message and the frame should be saved in s3 bucket | 'Face Not Matched' error and inside the s3 bucket the image of the person currently giving the exam |

**Conclusion**

In this post, we showed how to use AWS services to successfully create a solution to help enforcing fair examination practices on students and test providers part and could enforce such fair practices and led enriched user experience. It reduces the count & need of invigilators, who used monitor the students live window which led to cost savings in tune 80%.