Face and Liveness Verification for Identity Verification with Amazon Rekognition

First published June 30, 2022

Last updated Mar 12, 2024

aws

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Table of Contents

1. Introduction

The digital shift has accelerated the adoption of online services for banking, shopping, and other day-to-day services, and people have become accustomed to near-frictionless user experiences. With this digital shift comes an increased need to protect users' identity businesses from fraud. Identity verification and step-up authentication services are simplifying user experiences, making services more accessible, combating fraud, preventing identity theft, and addressing regulatory requirements across industries.

Identity verification is the process of verifying a person's identity by matching a user to a trusted reference; for example, verifying a person is a real genuine user and matching their face to an ID document like a driver's license, a student ID, or a passport. Step-up authentication is the process of requiring additional levels of authentication to increase security for high-risk and high-value transactions; for example, asking a user for a one-time password or verifying their face before wiring money to an external account.

In this white-paper, we focus on the face verification aspects of identity verification and step-up authentication. We present commonly used digital identity workflows such as new user registration (onboarding) and existing user login (authentication), and how you can implement liveness detection, face detection and analysis, face matching (1:1), and face searching (1:N). We also present a scalable reference architecture for implementing these workflows. We share best practices around Amazon Rekognition Liveness and Face APIs, and provide guidance on storage and security. Finally, we present several business metrics you can implement to track the health of your identity verification or step-up authentication solution.

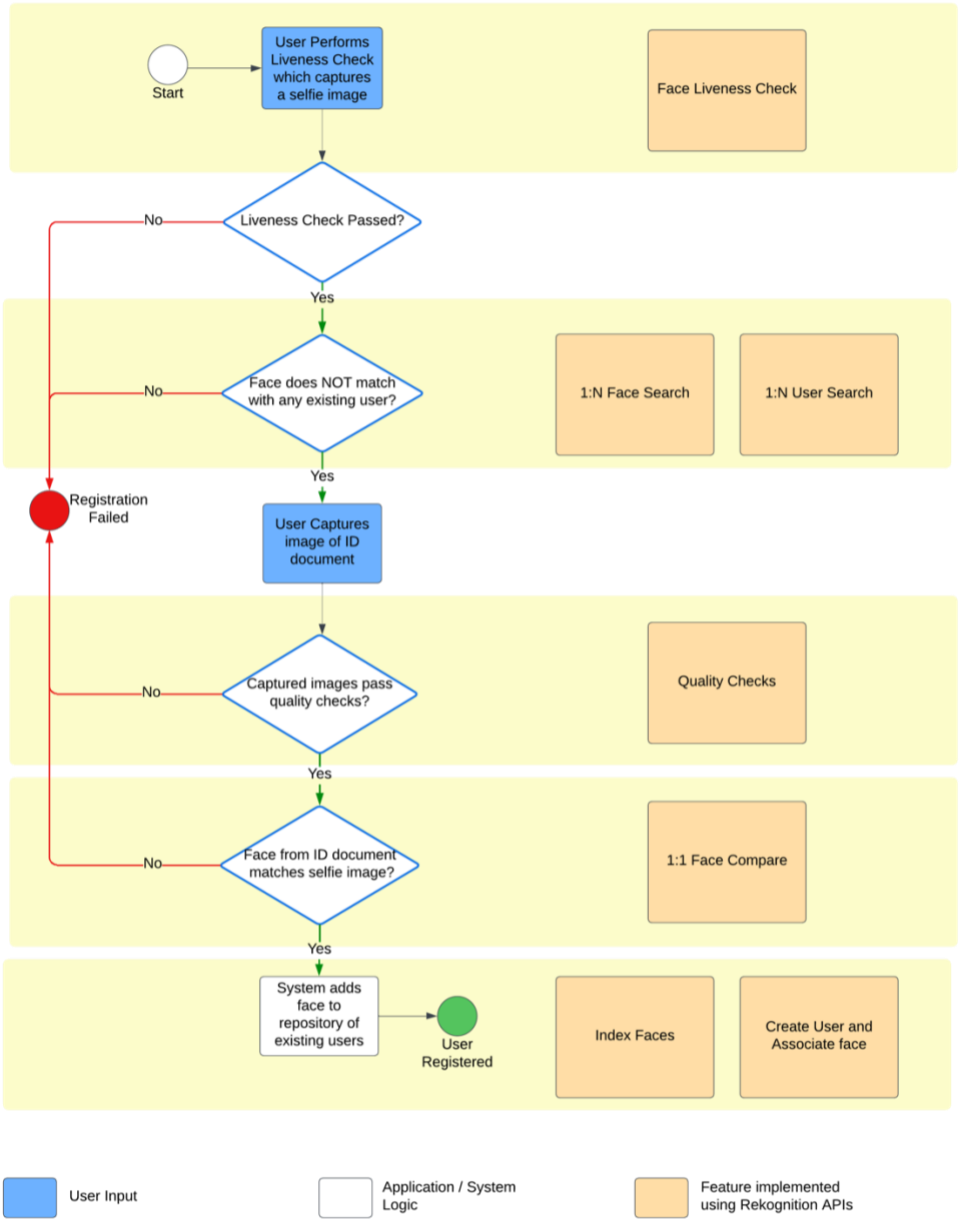
2. Digital Identity: Feature Implementations

2.1. Reference Workflow

The following figure shows a sample workflow for new user registration. The inputs required from the user are minimal to minimize friction and are limited to:

1. A user performs a liveness check
2. A user captures or uploads an image of an ID document

Notice that the logical decisions made by the application is enabled by face analysis and face recognition technology (FRT) features implemented using Rekognition.



New user registration

Using the Rekognition APIs as software building blocks allows you to customize the user registration flow according to your business process and gives you the flexibility to add or remove functionality into the workflow as required. These same features can be applied to other workflows such as existing user logins, bad actor lookups, and step-up authentication among others, bringing technical agility to your organization.

3. Feature Implementations using Rekognition APIs

3.1. Key Concepts

Before we dive into the common flows and best practices, it is important to know the following concepts and Rekognition capabilities:

Face vectors: A mathematical representation of a detected face from an image that is used when performing face matching.

User vectors: An aggregation of multiple face vectors from the same person to represent a single user. User vectors can be used to improve search performance.

Collections: Server-side repositories of face vectors and user vectors. Actual images files are never stored in a collection and the values of face vectors are not exposed.

Storage API operations: Amazon Rekognition may store content. For example, for Face Search, detected facial information, called face vectors, are stored in containers known as collections. Amazon Rekognition provides additional API operations you can use to search the persisted face vectors for face matches. For more information, see Storage-based API operations.

Non-storage API operations: These are referred to as non-storage API operations because when an opted-out customer makes the operation call, Amazon Rekognition does not persist your content that is processed by Rekognition. Opted out means a customer that instructs AWS to not use and store such content to develop and improve Rekognition by opting out through AWS Organizations.

Face liveness checks: Amazon Rekognition Face Liveness helps you verify that only real users, not bad actors using spoofs, can access your services. You can detect spoofs presented to the camera, such as printed photos, digital photos, digital videos, or 3D masks, as well as spoofs that bypass the camera, such as pre-recorded or deepfake videos.

Validate selfie picture: Amazon Rekognition Face Detection helps you detect that the user's selfie picture is captured correctly. You can detect if a face is present in the picture. You can also use predicted attributes such as bounding box size, pose, brightness, sharpness, eyes open, mouth open, and eyeglasses worn to determine picture quality.

Compare selfie picture with user ID: Amazon Rekognition Face Comparison helps you measure the similarity of two faces to help you determine if they are the same

person. You can receive a similarity score prediction for a user's selfie picture against their identity document picture in near real time.

Detect duplicate users: Amazon Rekognition Face Index and Search enables you to create a face collection of existing users and search new user selfie picture against all faces in your collection to detect duplicate or fraudulent account creation attempts.

3.2. Face Liveness Checks

Amazon Rekognition Face Liveness helps you verify that a user going through facial verification is physically present in front of a camera. It detects spoof attacks presented to a camera or trying to bypass a camera. Users can complete a Face Liveness check by taking a short video selfie to verify their presence.

The face liveness feature can be integrated to your application using both Rekognition APIs and the Amplify SDK provided the hardware you consider meets the following minimum specifications. The Amplify SDK is approximately 1-3MB in size and is available in React, native iOS, and native Android. If you are using frameworks other than the ones listed, you may be able to implement workaround solutions such as this example for Angular. Note however, that Amplify does not officially support any custom implementation of the Amplify SDK.

To configure your application to integrate with face liveness, you will utilize 3 APIs:

CreateFaceLivenessSession	Initiates a Face Liveness session and returns a SessionId, which can only be used for one check and is only valid for 3 minutes.
StartFaceLivenessSession	Starts a real time video stream and liveness detection prediction process for a valid session. You will not directly implement this API call. Rather you will use the FaceLivenessDetector component from the AWS Amplify SDK which calls this API.
GetFaceLivenessSessionResults	Retrieves the results of a specific Face Liveness session. This includes the corresponding Face Liveness confidence score, a reference image that includes a face bounding box that you can use for face matching, and audit images that also contain face bounding boxes. You will apply a threshold on the liveness confidence score to give a pass/fail decision.

To integrate Face liveness into your application, you need to perform the following:

1) Configure a backend to call the **CreateFaceLivenessSession** API. This API will return a SessionId that is valid for 3 minutes (i.e. session TTL = 3 minutes). The backend infrastructure could be implemented using Lambda functions, and a sample implementation can be referred to in this git repo.

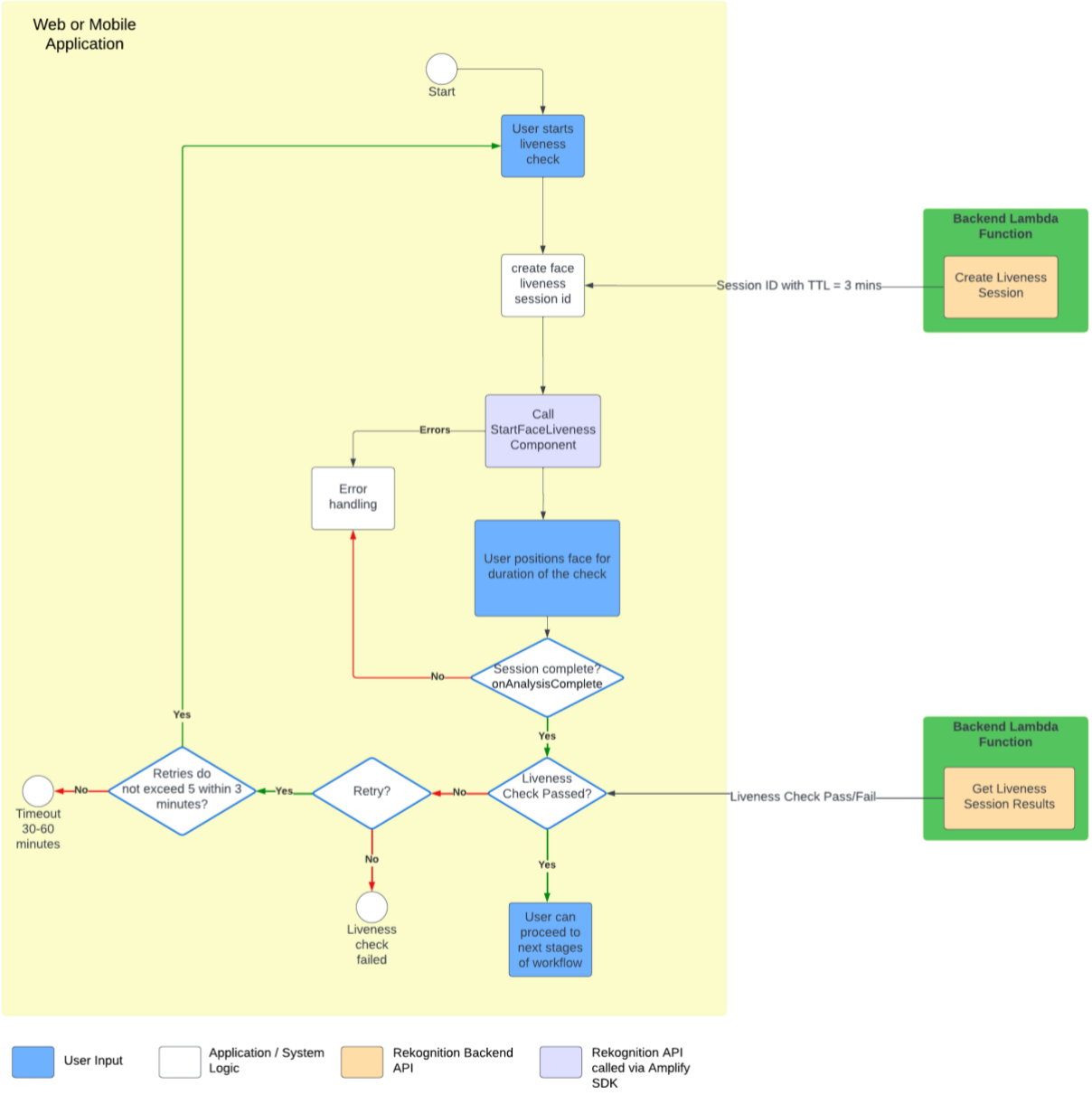
2) In your application, configure the AWS Amplify UI **FaceLivenessDetector** which provides a UI component that:

    a. Provides a user interface and provides real-time feedback for users during the liveness check.

    b. Starts an event stream from the app to the Rekognition service which analyzes the stream data.

3) Setup a backend to call the **GetFaceLivenessSessionResults** API. This API will return the results of the Face Liveness session, including a Face Liveness Confidence score, as well as a reference image and audit images.

4) Apply a threshold on the Face Liveness Confidence score to provide a pass/fail decision.



The above figure illustrates the flow of a face liveness implementation using the Rekognition Face Liveness APIs.

Note the following best practices when using Rekognition Face Liveness:

- Users should complete the Face Liveness check in environments that aren't too dark or too bright and have fairly uniform lighting.
- Web SDK users should increase their display screen's brightness to its maximum level when making checks on web browsers. Mobile Native SDKs adjust the display brightness automatically.
- Regularly run human review checks on audit images to make sure that spoof attacks are mitigated at the confidence threshold you set.
- Offer an alternative face liveness verification path to your users if they are photo-sensitive or do not want to verify their face liveness using Rekognition.
- Allow only five failed liveness checks in three minutes from a single device. After five fails, timeout the user for 30–60 minutes. If the pattern is seen 3–5 times repeatedly, block the user device from making additional calls.
- Implement the get-ready screen in your workflow so that users can more easily pass the Face Liveness checks.
- You are responsible for providing legally adequate privacy notices to, and obtaining any necessary consent from, your End Users for the processing, storage, use, and transfer of content by Face Liveness.

For further guidance on the responsible design and use of the service, see the Service Card for Amazon Rekognition Face Liveness.

3.3. Quality checks

Quality checks can be implemented using the following API:

DetectFaces	Detect 100 largest faces in an image and return 'face details' for each detected face. A face detail object comprises of analysis metadata including the bounding box of the face, a confidence value, and a set of attributes such as facial landmarks (for example, coordinates of eye and mouth), pose, age estimation, and presence of facial occlusion.

3.3.1. Face attribute threshold checks

Good image quality is an important consideration when working with AI systems as it can enable more accurate feature extraction and analysis, leading to more reliable predictions. To enable accurate face matching, apply a quality filter on all input images. If you are using a digital identity workflow that uses front facing poses, it is recommended to use the values of the facial attributes returned by the face APIs and apply thresholds on those values. The recommended thresholds are the following:

- Yaw between -30 to 30 degrees
- Pitch between -30 to 30 degrees
- Sharpness > 25
- Brightness > 25
- FaceOccluded = False
- Size of the face in an image is > 50x50 pixels

For 1:1 face comparison, use the DetectFaces API before calling the CompareFaces API. This can be described by the following:

1. **Index faces in an image:**
   Use the DetectFaces API to detect faces and facial metadata from in an input image
2. **Filter faces by quality:**
   Analyze the response from the IndexFaces API to assess each detected face against the recommended quality attributes (such as FaceOccluded, brightness, or pose).

For 1:N search, after indexing an image, apply the filter on the response of the IndexFaces API. Delete any faces that do not pass the quality check using the DeleteFaces API. (Note: There is no cost to call the DeleteFaces API). While you can also use the DetectFaces API before you index an image, this would require an additional API call for DetectFaces. This can be described by the following:

1. **Index faces in an image:**
   Use the IndexFaces API to detect and add faces from in an input image to a collection

2. **Filter faces by quality:**
   Analyze the response from the IndexFaces API to assess each detected face against the recommended quality attributes (such as FaceOccluded, brightness, or pose).

3. **Delete low-quality faces:**
   Remove faces from the collection that do not meet the quality threshold using the **DeleteFaces** API, based on the face IDs received from the IndexFaces API.

3.4. 1:1 Face Compare

A 1:1 Face compare feature is implemented using the following API:

CompareFaces	Compares the largest face in a *source* input image with each of the 100 largest faces detected in a *target* input image. For each face match, the response provides a bounding box of the face, facial landmarks, pose details (pitch, roll, and yaw), quality (brightness and sharpness), and confidence value (indicating the level of confidence that the bounding box contains a face). The response also provides a similarity score, which indicates how closely the faces match.

A 1:1 face compare feature would be implemented as follows:

1. **Run quality checks in images**
   Ensure that the source and target images pass quality checks using the **DetectFaces** API described in section 3.3.

2. **Compare images**
   Use the **CompareFaces** API to predict a similarity score between the faces in the source and target images

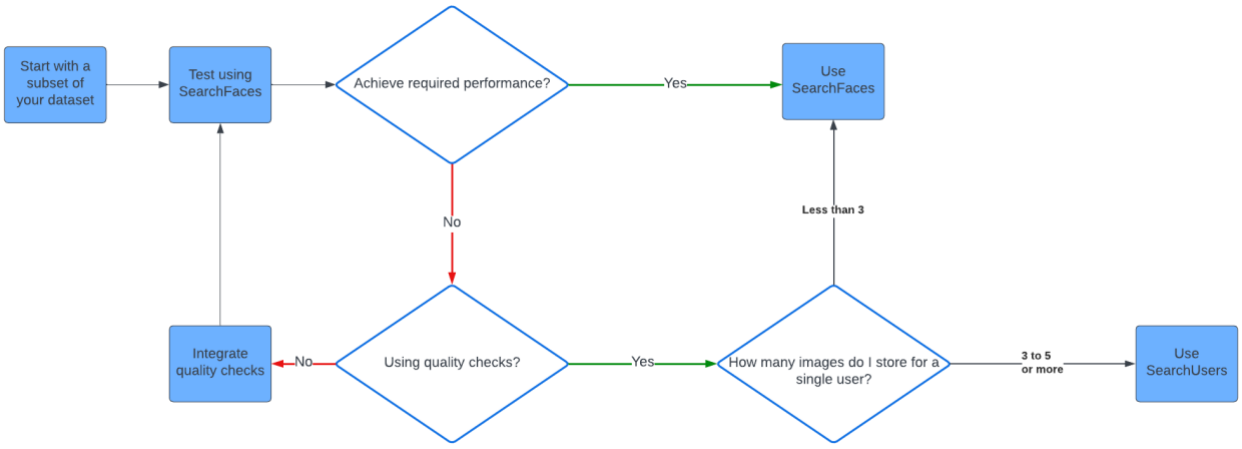
3. **Apply similarity threshold**
   Apply a threshold on the similarity score to give a match / no match decision

3.5. 1:N Search

When face data are stored in collections, they are stored as face vectors. A face vectors is a mathematical representation of the face - it is not the actual image of the face.

Multiple face vectors can then be aggregated to create and store user vectors. User vectors are more robust representations, as they contain multiple face vectors with varying degrees of lighting, sharpness, poses, appearance differences, etc.

You should decide whether your application requires searching using SearchFaces (face vectors) or SearchUsers (user vectors) as early as possible in the development of your application. Refer to the flow chart below to help with this decision:



3.5.1. 1:N Face Search

1:N face search is implemented using the following APIs:

CreateCollection	Creates a collection in the region from which the API request is made
IndexFaces	Detects the 100 largest faces in a single input image, converts them to face vectors, and stores them along with other user provided metadata to a single collection.
SearchFacesByImage	Detect the largest face in an image and then searches a collection for matching faces. It provides a similarity score for up to 4096 faces.
DeleteFaces	Deletes face vectors from a collection. You can delete up to 4096 faces from a collection in a single API request.

A 1:N face search feature would be implemented as follows:

1) **Create and populate a collection:**

    a. Use the CreateCollection API to create a new collection.

    b. Use the IndexFaces API to convert images of faces to face vectors and add them to the collection. If it is possible to maintain the required quality, stitch multiple images together to reduce the number of IndexFaces API calls made. For example, if you have 2 images of user A, one from a selfie and one from an ID document, you can stitch them as a single image and call the IndexFaces API once, instead of calling it for each image individually.

    c. Run quality checks in indexed faces as described in section 3.5 and delete any low quality images using the DeleteFaces API.

2) **Search a new image against the collection**

    Given a new face image, search for similar faces that exist in a collection using the SearchFacesByImage API.

3) **Apply threshold**

For the similar faces returned, select the most similar face or faces by applying a threshold on the similarity score

For further guidance on the responsible design and use of the service, see the AI Service Cards - Amazon Rekognition Face Matching.

3.5.2.1:N User Search

1:N user search is implemented using the following APIs:

CreateCollection	Creates a collection in the region from which the API request is made
IndexFaces	Detects the 100 largest faces in a single input image, converts them to face vectors, and stores them along with other user provided metadata to a single collection.
CreateUser	Creates a new User within a collection specified by CollectionId. Takes UserId as a parameter, which is a user provided ID which should be unique within the collection. The provided UserId will alias the system generated UUID to make the UserId more user friendly.
AssociateFaces	Associates one or more (up to 100) faces with an existing UserID.
SearchUsersByImage	Detect the largest face in an image and then searches a collection for matching Users. It provides a similarity scores for up to 500 UserIDs.
DeleteUser	Deletes face vectors from a collection. You can delete up to 4096 faces from a collection in a single API request.

A 1:N user search feature would be implemented as follows:

1) **Create and populate a collection:**

    a. Use the CreateCollection API to create a new collection.

    b. Use the IndexFaces API to convert images of faces to face vectors and add them to the collection. If it is possible to maintain the required quality, stitch multiple images together to reduce the number of IndexFaces API calls made. For example, if you have 2 images of user A, one from a selfie and one from an ID document, you can stitch them as a single image and call the IndexFaces API once, instead of calling it for each image individually.

    c. Run quality checks in indexed faces as described in section 3.5 and delete any low-quality images using the DeleteFaces API.

2) **Associate faces to users**

    a. Create new users using the CreateUser API

    b. Call the AssociateFaces API to define which faces are attributed to which users.

3) **Search a new image against the collection**

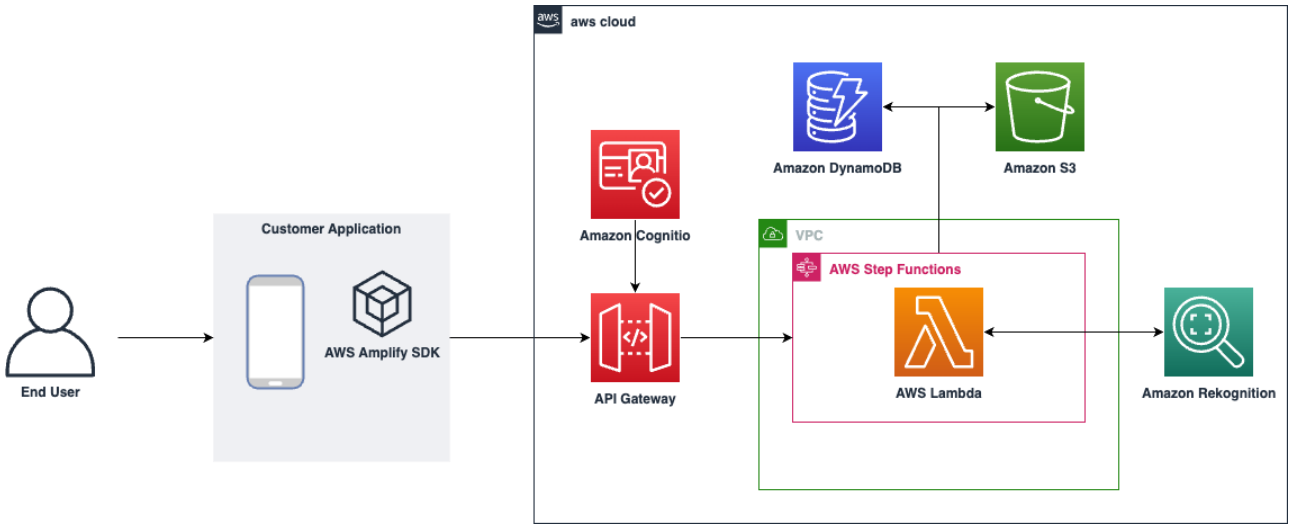
    Given a new face image, search for similar users that exist in a collection using the SearchFacesByUsers API.

4) **Apply threshold**

    For the similar users returned, select the most similar user or users by applying a threshold on the similarity score.

4. Reference Architecture

The following reference architecture shows how Amazon Rekognition, along with other AWS services, can be used to implement face verification.



Reference Architecture for Face Verification

The architecture follows a serverless model and uses the following services:

- AWS Lambda: Allows you to run code without the need to provision or manage server. It is used to run code that makes API calls and contains your business logic.
- Amazon S3: It is an object storage solution to store the photos captured by the end user on their mobile or web camera app.
- Amazon Rekognition: It is a fully managed machine learning service that provides the core API's to enable face based authentication.
- Amazon API Gateway: It is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. It is used as a "door" between the backend (Lambda Functions) and the end user mobile or webcam app.
- AWS Step Functions**:** It is a fully managed service that makes it easy to coordinate the various faces API using visual workflows.
- Amazon DynamoDB**:** It is a key-value and document database that delivers single-digit millisecond performance at any scale. Is used to store the metadata produced (face-id returned from the IndexFaces API, social security number (SSN), S3 URL).
- Amazon Cognito : It is a service that enables developers to create secure user authentication and access control for their web and mobile applications.

5. Collection Management

5.1. Overview

When designing identity verification solutions with Amazon Rekognition, it is necessary to strategically plan for collection design and management requirements for expanding the volume of facial recognition searches and the size of the collections, while ensuring low latency in searches. Additionally, such a management plan should cover continuous updating of facial recognition models and establish proper access management protocols.

5.2. Calculating number of collections

While there is no upper limit to the number of collections that can be created in an AWS account, each individual collection can store up to 20 million face vectors. You must therefore calculate the number of collections you require for your use case.

For instance, when onboarding a new *company A* with 10 million faces, you would want to create a dedicated collection for them (*collection A*) rather than sharing the same collection with another *company B* with 8 million faces (*collection B*). This approach helps in maintaining organizational separation but also importantly limits the search space, hence reducing the number of 1:N Face Search operations required in large scale applications.

Depending on the total number of face vectors you store however, sharding collections across multiple collections may be necessary. Consider the scenario described in the table below:

	Quantity	Volume	Comments
A	Number of images	80 million	An image may have multiple faces
B	Number of faces	80 million	A single collection can store up to 20 million face vectors
C	Number of users	40 million	Each user may be associated with multiple images and faces

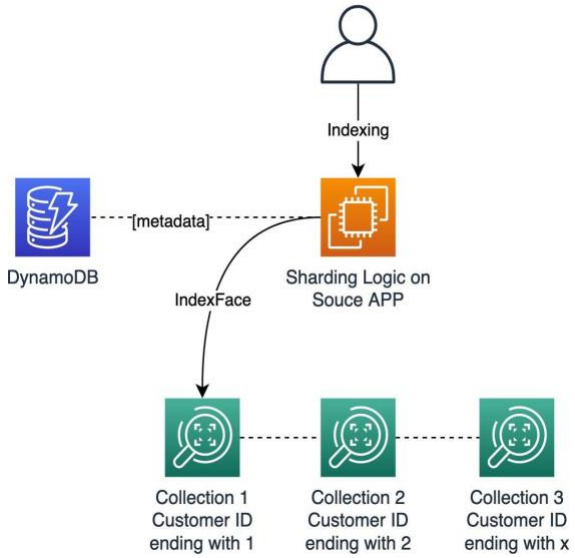
If you are going to search using **faces**, then you will need at least B/Collection-limit = 80M/20M = 4 collections to store all the faces. You can then either selectively choose which collection to search based on your use case, or you can search across all collections by running 5 parallel SearchFaces API calls. If you are going to search using **users,** then considering an average of B/C = 80M/40M = 2 images per user, you will be able to store C/2 = 20M/2 = 10M users per collection.

5.3. Sharding Considerations

Sharding is a technique of distributing faces to multiple collections according to some metadata (name of the customer, first digit of the user's unique id number, state, country).

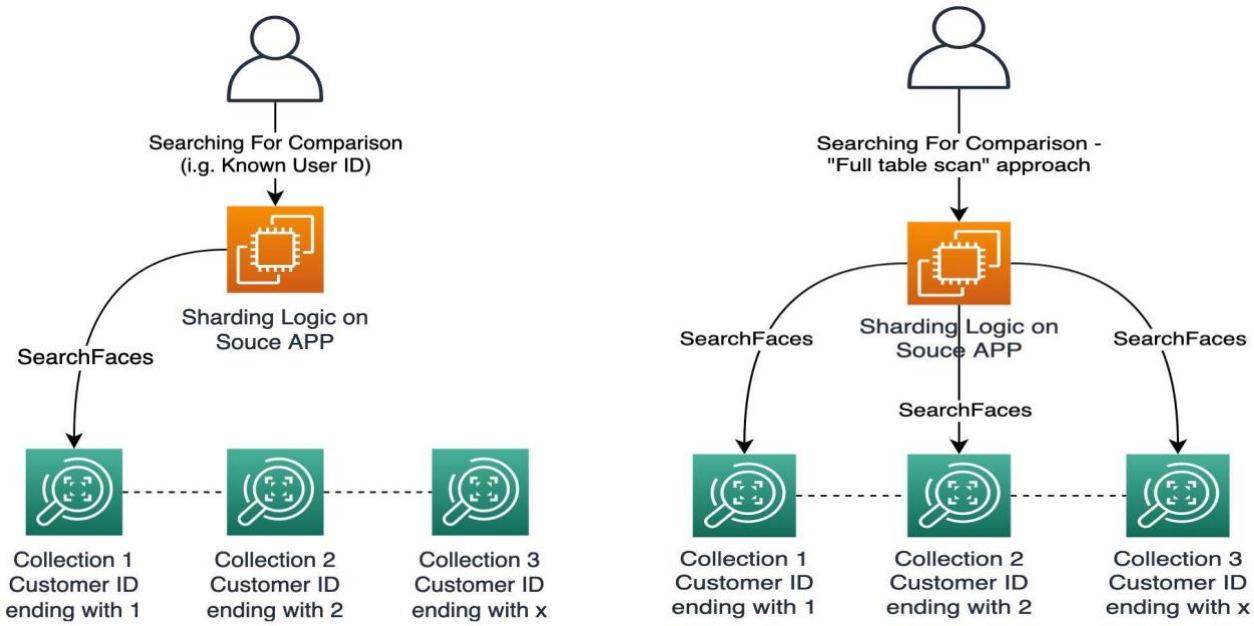
In the following diagram, collections have been sharded based on the unique ID (last. digit) of the user/customer. Faces are indexed into the collection that matches the last digit of the unique ID. DynamoDB is used as a metadata store. Depending on the use case other hashing strategies can be applied to shard the collections.

Using the example below, the face vectors have been indexed across 3 collections. If your use case allows you to predetermine which collection to search (e.g. customer ID ending with 1, or company ID) then when a new input image is provided, you can use a single SearchFaces API call on the identified collection.



Example: Index Face on sharded collections

However, if you do not have such information, and require to scan across all of the collections, you will make 3 SearchFaces API calls, and aggregate the results. These two approaches are depicted below:



Example: On the left, searching a prespecified collection. On the right, searching across all collections

5.4. Secure and isolated data management

For use cases where you require granular control over the data stored in collections such as for multi-tenant isolation, or to implement access controls for your organization, you should consider the following:

- Create multiple collections to partition your data.
- Use Amazon IAM policies to define who can perform actions on collections. You can achieve this by specifying the Amazon Resource Name (ARN) or collection ID of the collection in a policy. When defining actions in a policy, you can create a custom policy or leverage one of the AWS managed policies.
- Assign tags to new and existing collections. Tags are key value pairs which enable you to ensure the right level of access control and also to categorize and track AWS costs.

Note that IAM policies and tags are applied at the resource level, not at the data (face vector/user vector) level.

5.5. Designing for low latency

Increasing the number of faces indexed to a collection does not significantly affect the search API latency. In fact, a collection that is populated with 20M faces can still be queried at sub second latencies. To optimize your application for low latency, consider the following:

- Utilizing high-resolution images, such as those in 4K, can affect network response times. It is recommended not to use very large resolution images of faces (4K etc.). If you have 4K images, down sample to Standard Definition before indexing.
- For batch indexing a new collection, store images in an Amazon S3 bucket in the same region as the collection. As it is required for the Amazon S3 bucket to match the Region used for calling the IndexFaces API, a mismatch would only allow the option of passing bytes across regions which would increase response times and cost.
- For a single search against a collection, call the SearchFaces/SearchUsers API with image bytes. Store the image in the relevant Amazon S3 bucket based on the search results (e.g pass or fail S3 bucket)

- When indexing or searching images, use images with only a single face. For images containing multiple faces, use the DetectFaces API to crop and isolate individual faces and only index the required cropped faces. This reduces computational load and latency, as the system processes fewer faces at a time.

5.6. Designing for Resiliency

Amazon Rekogntion is designed with high resiliency best practices leveraging AWS global infrastructure. For all the APIs included in this document, all server deployments are done over multiple AZs in a region.

If your application requires a multi-Region Active/Active strategy that requires data replication across regions, you should provision collections in each selected region and build the application logic for indexing a face in both collections. With this approach you will duplicate the number of Rekognition API calls and ensure that collections are synchronized between regions.

5.7. Indexing New Collections and Collection Migrations

Whether you are indexing an empty collection with a large number of face images, or migrating an older populated collection to a new collection running the latest face model, you will need to create a batch processing job to call the IndexFaces API on all of the images in the shortest amount of time.

Refer to the following best practices when designing a batch indexing process.

1. Ensure the image source and Amazon Rekognition are in the same region to avoid data transfer costs.
2. If your data privacy and tenancy policies require image data to be hosted in a region where Amazon Rekognition is not supported, then you could read the image data as raw bytes. For example, the image data may need to reside in a S3 bucket in a region where Amazon Rekognition is not supported. In this case, the image data from the S3 bucket can be read as raw bytes and passed to Amazon Rekognition API in a region where Rekognition is supported.
3. Create a mapping between the faces that have been indexed to collections and the actual face images – see section 5.8 for details

4. Verify the account current TPS limit for the Indexfaces API through the Service Quotas page in the AWS console. Next, examine your historical usage by checking Rekognition metrics in CloudWatch.

   If you're nearing your usage limit or experiencing throttling with the IndexFaces API, you can request a temporary TPS limit increase to speed up indexing..

5. The TPS limit can be maximized by making parallel API requests. You can use multiple concurrent lambdas in conjunction with AWS Step Function Parallel States or S3 Batch to achieve this. Try to use scaling best practices, including error handling and retry and exponential backoff in AWS SDK, to make the index process smoother. An Amazon SQS Queue can be used if you run into throttling issues during the indexing process.

6. Take into consideration the source of images currently used (EBS/EFS/other AWS storage and on premises) and analyze a possible migration to the appropriate storage solution that integrates with Amazon Rekognition, such as a S3 bucket.

If you have a mapping table created, you can also quickly deploy this solution in your AWS account which reduces the time taken to index a collection with millions of faces from days to hours or minutes.

5.8. Data Mapping

It is recommended to maintain an up-to-date mapping between the faces that have been indexed to collections and the actual face images. This is required for manual verifications, faceId matching, audits, as well as for migrating collections, where the original images will be required for generating new face vectors.

The following is an example of a **DynamoDB** table that can be leveraged for cases where face vectors are used. Note this schema allows for:

- Mapping multiple faces per image
- Mapping faces across collections and users

You can modify the Primary Key or add additional attributes based on the required access patterns and business requirements.

Mapping Table			
Primary key		Attributes	
PK	SK		
user_id	collection_id#face_id	s3_image_url (ExternalImageId)	bbox
24234234	42342321#12312123	s3:/bucket/image.jpeg	[...]
24234234	42342321#42312199	s3:/bucket/image.jpeg	[...]
24234234	42342321#77712197	s3:/bucket/image2.jpeg	[...]

6. Best practices for Rekognition APIs

6.1. Selecting thresholds

The overall performance of an identity verification system should be measure by calculating the False Rejection Rate (FRR) and the False Acceptance Rate (FAR) value. The FRR measures the probability of a legitimate user being rejected, while the FAR determines the probability of an impostor being accepted. These metrics are generally inversely related. For example, as FAR goes up, FRR goes down and vice versa. Adjusting the score threshold will affect the FAR and FRR.

Amazon Rekognition face matching APIs analyze a pair of facial images features to compute a quantitative match score between 0-100. A score of 0 represents dissimilar faces; with high confidence they are different people. Conversely, a score nearing 100 signifies high facial similarity, suggesting the faces belong to the same person.

Face Liveness is evaluated using a probabilistic calculation, which generates a confidence score ranging from 0 to 100. The higher the score, the more confident we can be that the person doing the check is alive. In addition, Face Liveness produces a frame referred to as a reference image, which can be utilized for face comparison and search.

The score threshold is a business-defined setting that determines if a verification attempt is considered a match or not for that specific workload. Balancing the need to minimize unauthorized users and preserve the user experience is a complex task that requires constant monitoring.

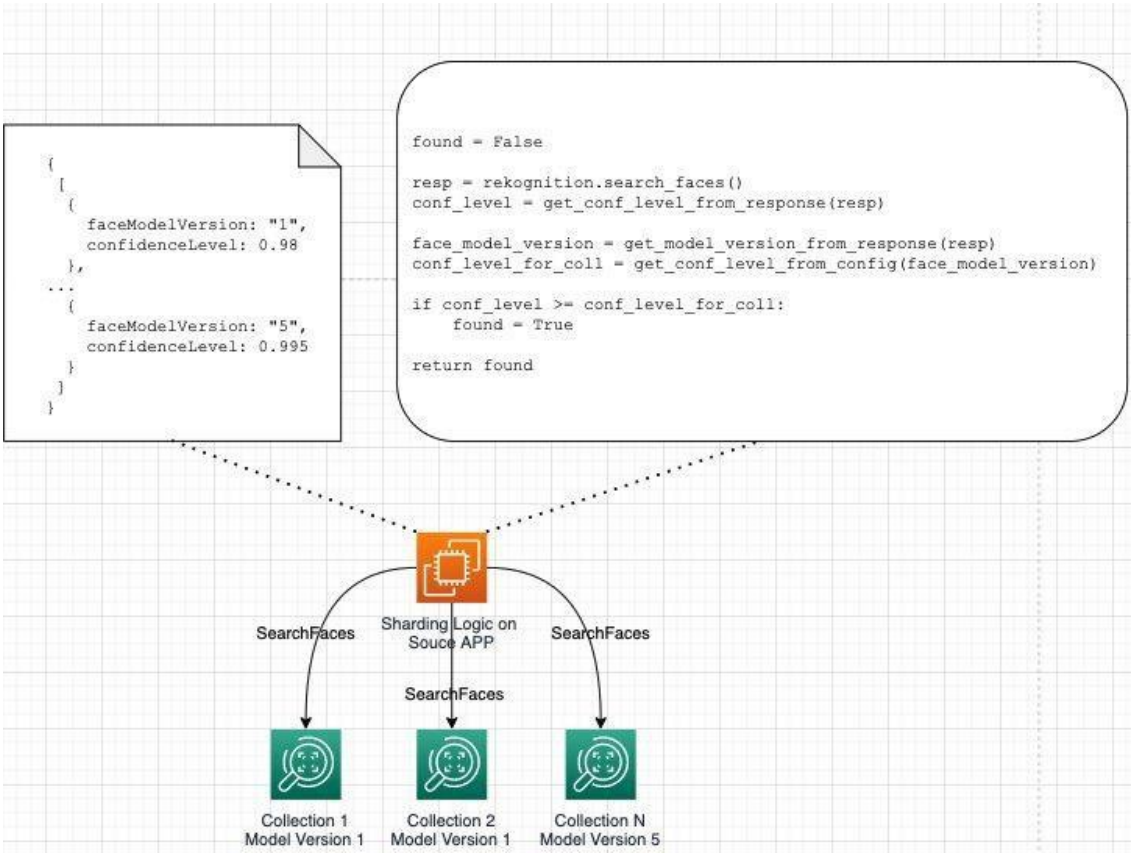
You should select a similarity threshold that is appropriate for your data to achieve the desired performance.

Here is a set of recommended thresholds to get you started:

API	Confidence attribute	Feature	Recommended threshold to begin testing
CompareFaces, SearchFaces, SearchFacesByImage	FaceMatchThreshold	Face Search APIs	95 – for regular use cases 99 – for sensitive use cases
SearchUsers, SearchUsersByImage, AssociateFaces	UserMatchThreshold	User Search APIs	95– for regular use cases 99 – for sensitive use cases
GetFaceLivenessSessionResults	Confidence	Face Liveness	80

For sensitive cases where a threshold or 99 or higher is required, it is recommended to first crop the face using the DetectFaces API before indexing to a collection.

In instances where multiple collections are used, you might encounter inconsistencies in the model versions across collections over a longer period of time. This discrepancy can lead to variations in similarity score distributions for the different face models – in which case you should select a similarity threshold for each version. This can be achieved using configuration files where confidence level thresholds are linked to each model version. In general, applying the same confidence score standards across different model versions can yield unpredictable outcomes, even though newer model versions provide more precise predictions.



Searching for comparison of an unknown user - Sharding Logic

6.2. Rekognition Updates

When a face model version is released, the stateless APIs (DetectFaces, CompareFaces) will automatically begin using the latest model version. In this case, no action is required by a customer.

When creating a new collection, it is established using the most current version of the face model at that time (currently Faces V7). In addition, a collection is permanently associated with the version of the model used during its creation. When a new version of the face model is released, you should migrate existing collections to new collections to incorporate the model's performance advancements in accuracy. Refer to Section 5.7 for collection migration guidelines.

Regularly update AWS SDKs as well as AWS Amplify SDKs to ensure that you are accessing the latest released features and security updates.

7. TPS Limits and scale

Amazon Rekognition provides service limits and quotas by API. Use the Service Quota page in the AWS console to check service limits, and for historical usage, refer to the Rekognition metrics pane. When designing the application, consider service quotas for scalability as volume grows and familiarize yourself with these scalability best practices.

- By utilizing raw bytes rather than an S3 Object, you can distribute traffic across multiple regions for non-storage APIs, without worrying about regional service quotas.
- Account for cold start situations as you ramp up your traffic.
- To create a consistent traffic pattern, use a queueing mechanism to avoid sudden spikes in traffic.
- Configure error handling, retries and exponential backoff in your application. See Error retries and exponential backoff in AWS.

In the event that your application still requires a service limit increase following the implementation of the above suggestions, you can request a quota increase using creating a support case.

8. Monitoring best practices

Organizations implementing an identity verification solution should consider measuring application and system health as part of ongoing application monitoring. There are two aspects to categorize application health.

First, collecting telemetry on system calls to Amazon Rekognition APIs such as DetectFaces, CompareFaces, and SearchFacesByImage. Capturing metrics from these APIs will help tune the performance of the identity verification solution. Second aspect will be to capture counts of successful, incomplete and failed registration or authentication attempts in your application. This will give you an idea of the user interaction trends with your application overtime.

8.1 System calls telemetry

Start monitoring Rekognition with CloudWatch. CloudWatch provides you with data and actionable insights to monitor your applications, respond to system-wide performance changes, and optimize resource utilization. CloudWatch collects monitoring and operational data in the form of logs, metrics, and events.

Use metrics to track the health of your Rekognition-based solution and set up alarms to notify you when one or more metrics fall outside a defined threshold. As an example, you can monitor metrics for server errors, metrics for detected faces, and the number of successful executions for a particular Rekognition operation. You can also monitor API calls latency or throughput and create alerts if there is an increase in latency with throughput degradation.

Within the Amazon Rekognition console, access the metrics pane to display activity graphs that aggregate individual Rekognition metrics for a specific time period. Consider the SuccessfulRequestCount aggregated metric as an example. It provides a summary of successful requests made to all Rekognition API operations in the past seven days.

8.2 Application metrics

As you develop your application, we recommend capturing keep performance indicators (KPIs) around registration and authentication process steps, successes, incomplete or abandoned attempts, and failures. These KPIs will help identity key business and application trends. We recommend capturing the following application metrics:

1. Number of approved, incomplete and failed registrations
    - Count of incomplete, abandoned and failed registrations, including where in the application workflow and reasons the registration was incomplete, abandoned, or failed.
    - Number of duplicate registration attempts. For each registration attempt, compare the selfie and identification document to registered users stored in the registered user collection. Record information on the attempt and existing registered user's details.

2. Number of approved, incomplete and failed authentications (reoccurring users)
    - Count of abandoned authentications. Record the user, reason (if available), and details around the abandoned authentication attempt. This is often an indication of attempted account takeover.
    - Count of face detections and searches, for each authentication attempt record details of the searches, detections, and authentication status as well as similarity scores and reasons why an authentication attempt was approved, incomplete or failed.
    - Count of authentication selfie images that fail the quality checks. Record the image information, quality measures, pose and reasoning why an image failed the quality check.

It's also important to measure metrics for Rekognition API calls made for face detection, comparison, and search. By using this metrics, you can measure your application's performance in detection, comparison, and search. Looking into fine-tuning similarity thresholds and investigating the links between image quality, face comparison, and face search, you can further optimize your API calls. We recommend capturing the following metrics and details:

1. Keep track of face comparisons, detections, and total faces found. Analyze obfuscation like sunglasses or masks, measure bounding boxes, assess quality, identify landmarks, and determine similarity.
2. The number of face detections recorded along with image quality and potential obfuscation.
3. The Number of face searches, recording the number of faces returned and their similarity.
4. Count of faces that were indexed, deleted, and updated.

9. Privacy and Security

As you build the identity verification solution, you must follow all relevant laws and best practices to ensure data privacy. Depending on your requirements, you may also want to opt out of having your image inputs used to improve or develop the quality of Amazon Rekognition and other Amazon machine-learning/artificial-intelligence technologies by using an AWS Organizations opt-out policy. For information about how to opt out, see Managing AI services opt-out policy. The section on **data privacy** and **access control** in the FAQ discusses the topics of content ownership, opt-out and taking audience consent when using Amazon Rekognition services.

9.1. Data protection

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

1. Use multi-factor authentication (MFA) with each account.
2. Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.

3. Set up API and user activity logging with AWS CloudTrail.
4. Use AWS encryption solutions, along with all default security controls within AWS services.
5. Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
6. If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a **Name** field. This includes when you work with Rekognition or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

10. Conclusion

This whitepaper offered best practices and comprehensive guidelines for leveraging Amazon Rekognition's liveness and face APIs. We covered key design considerations for building well architected solutions with a focus on aspects like data security, reliability, as well as scale through cost-efficient implementations. As we continue to witness rapid advancements in face and liveness verification technology, this document will serve as a vital resource, guiding you towards developing face verification workflows that align with evolving best practices.

11. Contributors

Zuhayr Raghib, AI Services SA
Marcel Pividal, Sr AI Services SA

12. Reviewers

David ROBO, Principal WW GTM Specialist
Sean Simmons, Principal Product Manager

13. Further Reading

For additional information, see:

1. Amazon Rekognition Documentation
2. Amazon Rekognition APIs