Identity Verification with Amazon Rekognition

First published June 30, 2022

Last updated Nov 01, 2022

Notices

Contents

Abstract

Today, many companies are turning to Amazon Rekognition to develop their identity verification solutions. This whitepaper outlines the design patterns and best practices for developing an Identity Verification solution using Amazon Rekognition and supporting AWS services. We discuss common identity verification workflows such as new user registration and authentication as well as present a scalable reference architecture to implement these workflows. We share the best practices around Amazon Rekognition Face APIs and offer guidance on various business metrics you can implement to track the health of your identity verification solution.

Are you Well-Architected?

The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems in the cloud. The six pillars of the Framework allow you to learn architectural best practices for designing and operating reliable, secure, efficient, cost-effective, and sustainable systems. Using the AWS Well-Architected Tool, available at no charge in the AWS Management Console, you can review your workloads against these best practices by answering a set of questions for each pillar.

For more expert guidance and best practices for your cloud architecture—reference architecture deployments, diagrams, and whitepapers—refer to the AWS Architecture Center

Responsible Use

Artificial intelligence (AI) applied through machine learning (ML) will be one of the most transformational technologies of our generation, tackling some of humanity's most challenging problems, augmenting human performance, and maximizing productivity. Responsible use of these technologies is key to fostering continued innovation. AWS is committed to developing fair and accurate AI and ML services and providing you with the tools and guidance needed to build AI and ML applications responsibly.

As you adopt and increase your use of AI and ML, AWS offers several resources based on our experience to assist you in the responsible development and use of AI
and ML. For more information, refer to the following resources:

- Use cases that involve public safety
- AWS Service Terms

- Resources and Tools provided by AWS on responsible use of AI and ML

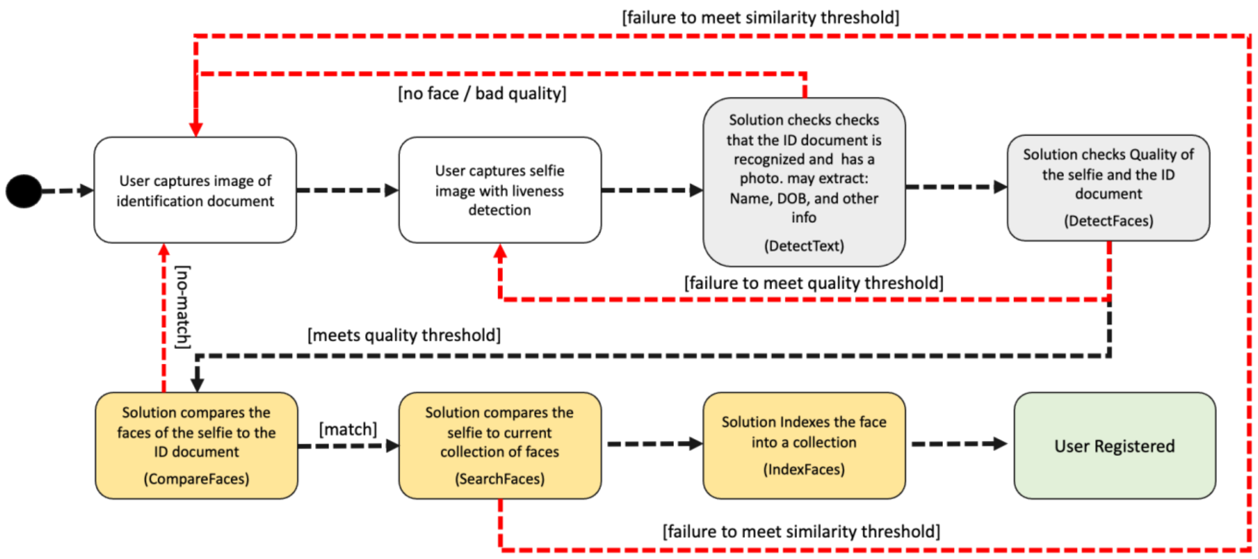
Introduction

The digital shift has accelerated the adoption of identity verification (IDV) as a service simplifying user experiences, making services more accessible, combating fraud, preventing identity theft, and addressing regulatory requirements across industries. People have become accustomed to near-frictionless user experiences as they increasingly rely on digital services for banking, shopping, and other day-to-day services. With this digital shift comes an increased need to serve, protect and verify user identity.

Identity verification is the process of verifying a person's identity by matching the user or customer to some known reference; for example, matching a person's face to a trusted identification document like a driver's license, student ID, or a passport. In this whitepaper, we present commonly used identity verification workflows such as new user registration (onboarding), and existing user login (authentication). We also present a scalable reference architecture to implement these workflows. We share best practices around Amazon Rekognition 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 solution.

Reference Workflow

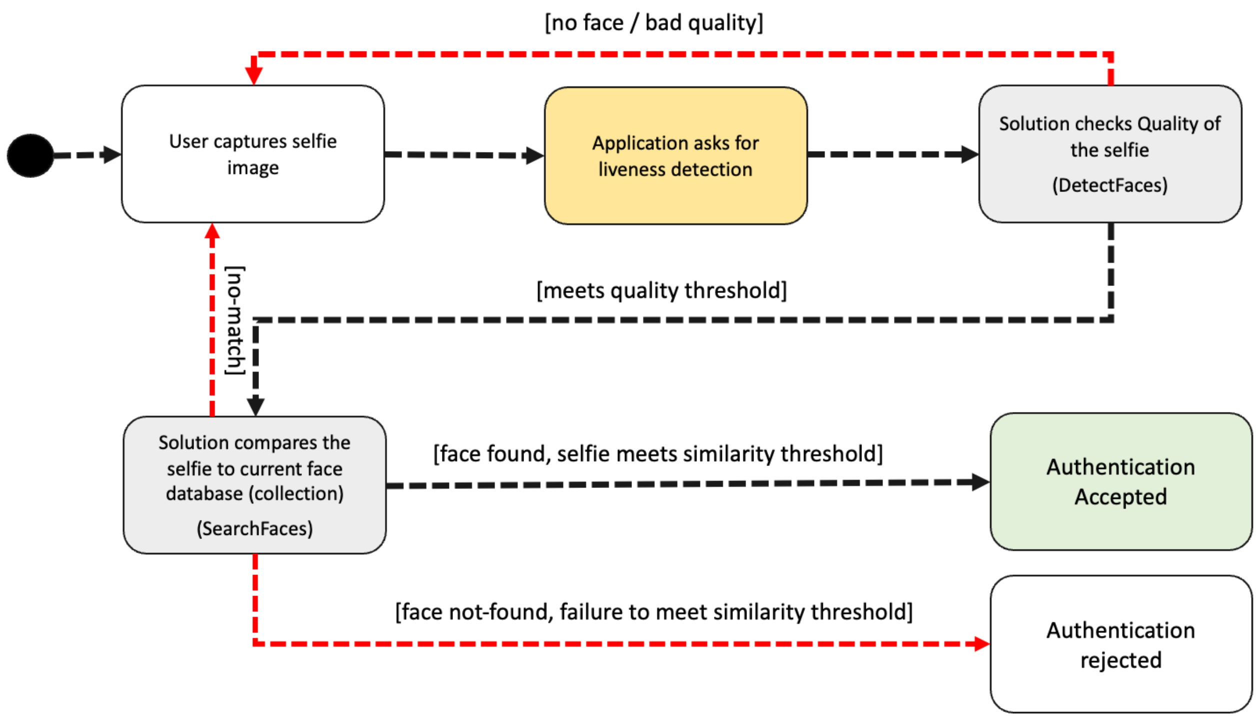
The following figure shows a sample workflow of a new user registration. Typical steps in this process are: Image capture of a user's face (selfie), liveness detection during selfie capture, image capture of a government issued identification document, quality check of the selfie image, comparison of the selfie with the identification document's face image, and a check of the selfie against a database of existing user faces.


[failure to meet similarity threshold]
[no face / bad quality]
User captures image of identification document
User captures selfie image with liveness detection
Solution checks checks that the ID document is recognized and has a photo. may extract: Name, DOB, and other info (DetectText)
Solution checks Quality of the selfie and the ID document (DetectFaces)
[failure to meet quality threshold]
[no-match]
[meets quality threshold]
Solution compares the faces of the selfie to the ID document (CompareFaces)
[match]
Solution compares the selfie to current collection of faces (SearchFaces)
Solution Indexes the face into a collection (IndexFaces)
User Registered
[failure to meet similarity threshold]

New user registration

You can customize the flow according to the business process. Often, it will contain some or all the steps presented above. You can choose to execute all the steps synchronously, i.e., wait for the step to complete before moving on to the next step. Alternately, you can execute some of the steps highlighted in orange asynchronously (not wait for that step to complete) to speed up the user registration process and improving the customer experience. However, care must be taken to roll back the user registration in case the steps are not successful.

In addition to the new user registration, another common flow is the existing or returning user login for authentication. In this flow a check of the user face image (selfie) is performed against a previously registered face. Typical steps in this process are: User face capture, liveness detection during selfie capture, quality check of the selfie image, and search/compare of the selfie against the faces database. The following diagram shows a possible flow.

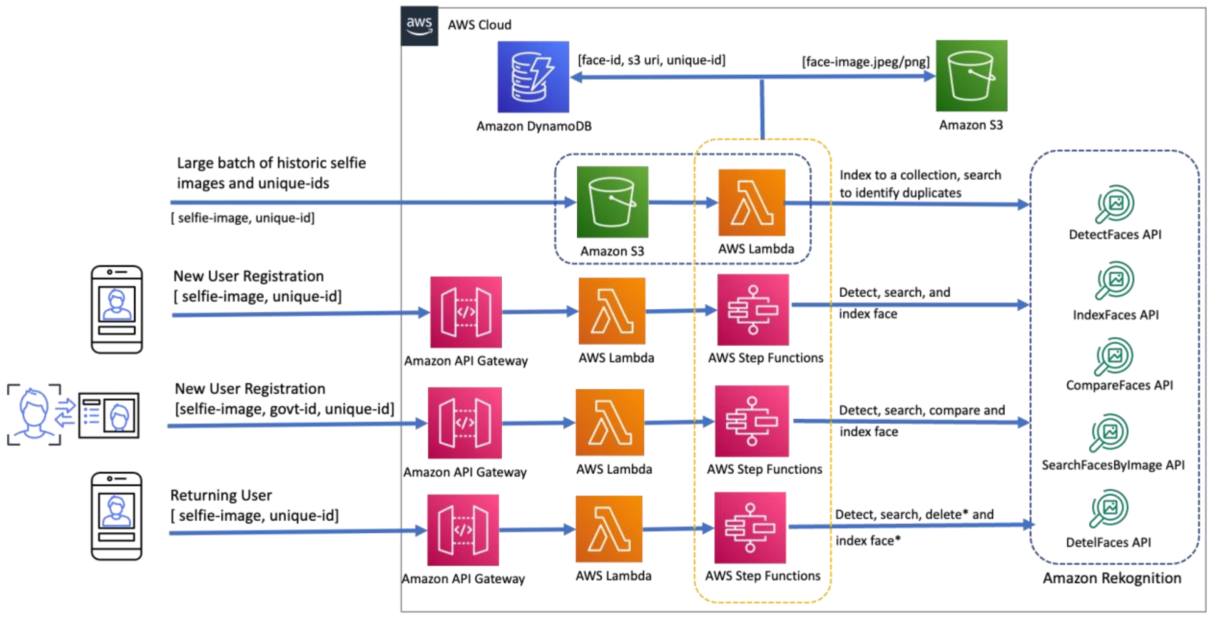

[no face / bad quality]
User captures selfie image
Application asks for liveness detection
Solution checks Quality of the selfie
(DetectFaces)
[meets quality threshold]
[no-match]
Solution compares the selfie to current face database (collection)
(SearchFaces)
[face found, selfie meets similarity threshold]
Authentication Accepted
[face not-found, failure to meet similarity threshold]
Authentication rejected

Existing or returning user login

You can customize the steps of the process according to the business, and choose to include/exclude the liveness detection (refer to the section on **Liveness Detection** for a deeper explanation).

Reference Architecture

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



Reference Architecture for Identity 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 Rekognition Key Concepts

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

Collections: Amazon Rekognition can store information about detected faces in server-side containers known as collections. The service doesn't persist actual image bytes. Instead, the underlying detection algorithm first detects the faces in the input image, extracts facial features into a **feature vector** for each face, and then stores it in the collection. To be clear, the service does not store the image in the collection instead the **feature vector** is stored. **Feature vectors** are simply a secure internal numeric representation of the detected face, they cannot be reverse engineered, and are only used by the service when performing comparisons. Amazon Rekognition uses these **feature vectors** when performing face matches and searches. You can use these feature vectors that's stored in a collection to search for known faces in images, stored videos, and streaming videos. You can use collections in a variety of scenarios. For example, you might create a face collection to store scanned badge images by using the IndexFaces operation. When an employee enters the building, an image of the employee's face is captured and sent to the SearchFacesByImage operation. If the face match produces a sufficiently high similarity score (say 99%), you can authenticate the employee.

Stateful/Stateless: A stateful API refers to any operation where information is persisted from a Collection. In a stateless API, no information is persisted in the system.

DetectFaces API: It detects faces within a provided image and returns information about faces detected. In a user registration workflow this operation helps ensure the quality of an image before moving to the next step. For example: check if a photo contains a face, that the sharpness/brightness levels of the image are good, confirm the person is not wearing sunglasses, and confirms the person's eyes are open. This is a stateless API.

- **Required Input**
    - Image (either as base64-encoded image bytes or as a reference to an image in a S3 bucket)
- **Outputs**
    - FaceDetail objects - returns an array of FaceDetail objects containing information such as: bounding box of the face, position of the face in the image, face angle, and various facial landmarks.

IndexFaces API: Detects faces in the input image and adds the metadata to the collection. This operation is used to add a screened image to a Collection for future query.

- **Required Input**
  - Image (either as base64-encoded image bytes or as a reference to an image in an Amazon S3 bucket) and target Collection (collectionId)
- **Outputs**
  - FaceRecords - An array of faces detected and corresponding face embeddings are added to the collection
- **IndexFaces** is a stateful API, and face embeddings along with an optional ExternalImageID are persisted in a Collection. The service doesn't persist actual image bytes. Instead, the underlying detection algorithm first detects the faces in the input image, extracts facial features into a feature vector for each face, and then stores it in the collection. Amazon Rekognition uses these feature vectors when performing face matches.
-

SearchFacesByImage API: For a given input image, first detects the largest face in the image, and then searches the specified collection for matching faces. The operation compares the features of the input face with faces features in the specified Collection. This is a stateful API.

- **Required Input**
  - Image (either as base64-encoded image bytes or as a reference to an image in an Amazon S3 bucket) and target Collection (CollectionId)
- **Outputs**
  - FaceMatches - An array of faces that matched the input face, along with the confidence in the match

CompareFaces API: Compares a face in the source input image with each of the 100 largest faces detected in the target input image. If the source image contains multiple faces, the service detects the largest face and compares it with each face detected in the target image. For our use case, it is expected both the source and target image contain a single face. This is a stateless API.

- Required Input
  - SourceImage (image as base64-encoded bytes or an S3 object for both parameters)
  - TargetImage (image as base64-encoded bytes or an S3 object for both parameters)
- Outputs - Faces in the target image that match the source image face

DeleteFaces API: Deletes faces from a collection. You specify a collection ID and an array of face IDs to remove from the collection.

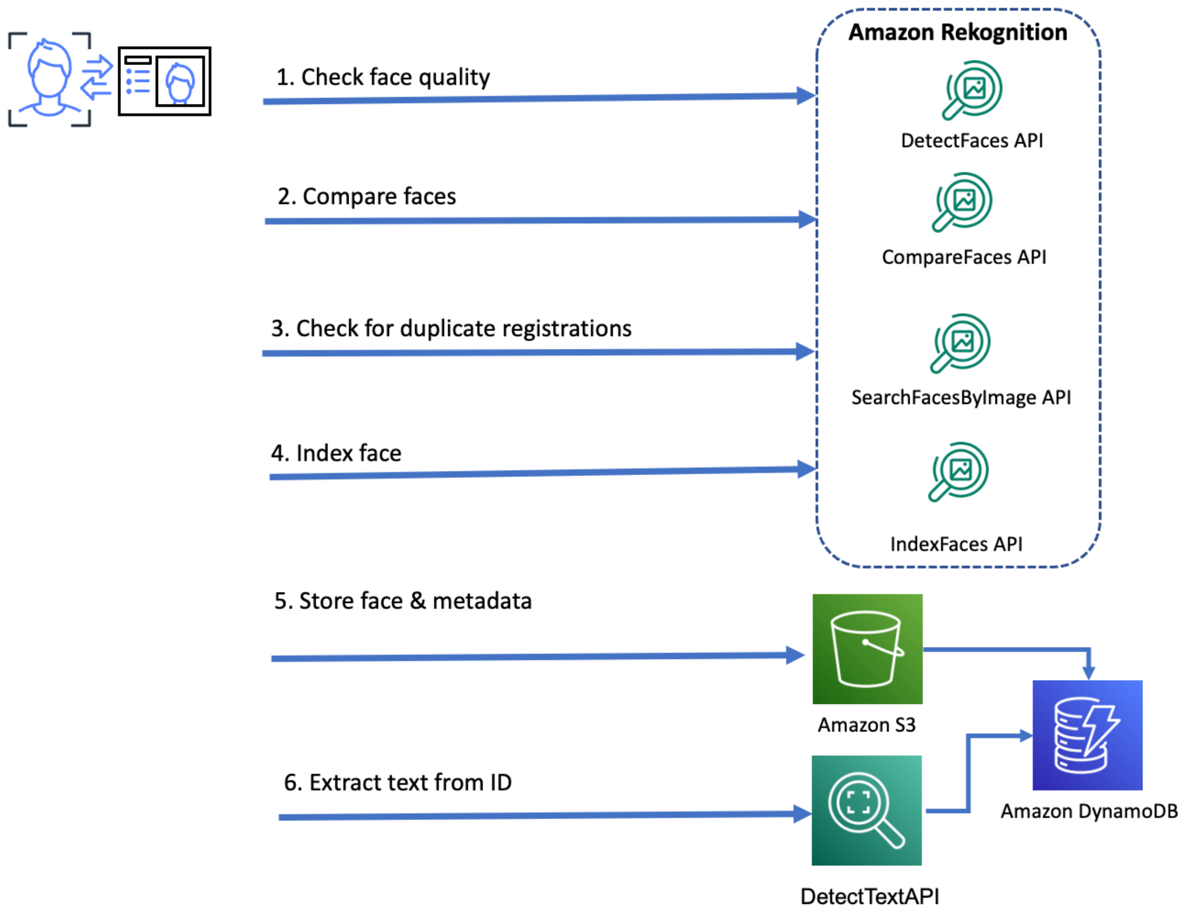
- Required Input
    - Collection ID
- Face IDs
    - Outputs - An array of face IDs that were deleted

Note: This is a stateful API, so face embeddings and associated data (ExternalImageID) are deleted from a collection.

Identity Verification Process Flows

New user registration (onboarding)

Whether you are onboarding a new customer, user, or employee; identity verification starts at the registration process. Historically, organizations have introduced some form of friction like CAPTCHAs or Multi-Factor-Authentication (MFA) to prevent spam and bad actors from registering. Face-based identity verification is as simple as match a selfie to a trusted reference image, like a driver's license, passport or other trusted source. Here we not only capture the images but we compare the similarity of the faces in each image



New user registration (onboarding)

In this workflow, users are asked to present both a trusted identification document like some form of government issued identification and a selfie or other photo containing their face. This can be performed using a web or mobile application. A recommended best practice flow is:

1. First check the face image quality via DetectFaces API.

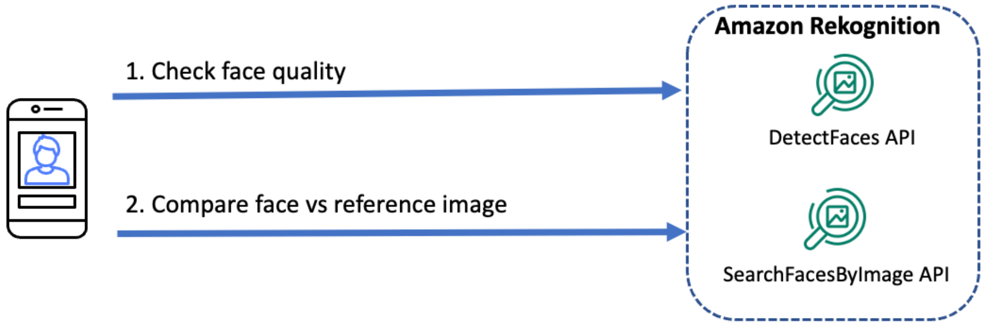
   Note: we recommend the following quality checks:

   - Yaw between -45 to 45 degrees
   - Pitch between -30 to 45 degrees
   - Sharpness > 15
   - Brightness > 15
   - EyesOpen = True
   - Sunglasses = False
2. Use CompareFaces API to compare the face image with ID face image for a match.
3. Use SearchFacesByImage API against the collection(s) to check for any duplicate registration.

4. Index the face image using IndexFaces API and use the ExternalImageID (a username or unique ID that maps to the user) parameter to associate the face embeddings with the ExternalImageID.
5. Depending on your use case, you can optionally store the face image in the S3 bucket along with the user metadata (face-id returned from the IndexFaces API, user identifiers (email, username, phone, etc), demographic data, and the S3 URI) in DynamoDB.
6. Consider the forms of “identification documents" you accept. Simply having a user upload an image of their government issued identification document can simplify and speed the registration steps. Amazon Rekognition’s DetectText API or Amazon Textract’s AnalyzeID API can extract the text from the image of the identification document avoiding the friction from users having to type information like name, address, document type, date of birth, issue date, expiration dates and more.

Existing user login (authentication)

Facial based authentication has become common place



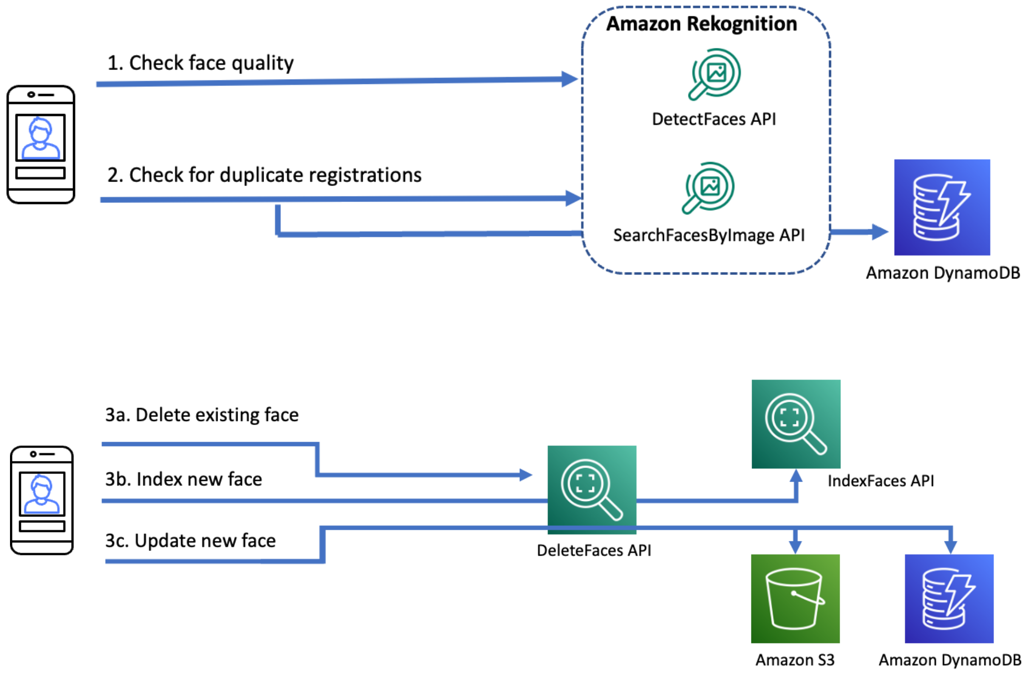
Existing user login (authentication)

In this workflow the user logs in based on face authentication. The user face is captured over a webcam or mobile app and passed to Rekognition’s SearchByFaceImage API which will compare the face against faces in the collection. A recommended best practice flow is:

1. Check the face image quality via DetectFaces API.
2. Search the face against the Collection with SearchFacesbyImage API. If a face match is found a successful response can be returned. If the use case requires the user to provide another factor such as the unique ID during the login process, you can optionally add a comparison of the user unique ID with the ExternalImageID to do an additional check.

Existing user login with a request for photo update

A common need is to update a reference image with the latest image or most recent selfie. For example, suppose you have a need to capture driver's licenses as the reference image in your application. When a license expires, you could simply prompt your user to submit an updated reference image. The following flow details how you might structure your application to update images in the collection.



Updating reference images in a collection

In this workflow the user logs in based on face authentication and provides a new profile photo. The user face is captured over the webcam/mobile app. A recommended best practice flow is:

1. First check the face image quality of both the user face using DetectFaces API.
2. Search the face against the Collection with SearchFacesbyImage API. If the use case requires the user to provide another factor such as the unique ID during the login process, you can optionally add a comparison of the user unique ID with the ExternalImageID to do an additional check.
3. If the faces match and updating the face is a requirement:
    a. Delete the existing face embedding from the collection using DeleteFaces API.
    b. Index the new face using IndexFaces API and use the ExternalImageID (Social Security number or a similar unique ID) parameter to associate the face embeddings with the ExternalImageID.

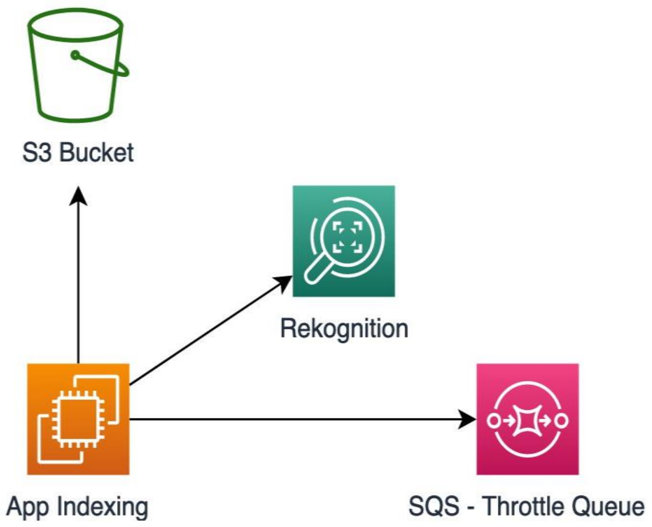
c. Depending on your use case, update the face image in the S3 bucket along with the user metadata (face-id from IndexFaces, Social Security number or unique ID, S3 URI) in DynamoDB.

Refer to the section on the **best practices** for various faces API and the section on liveness detection.

One time indexing of a large collection of images (batch index)

Often customers moving to Amazon Rekognition have a large collection of images. In this scenario, a one-time indexing is needed and its execution may take a period of days.

1. A key best practice is to have the image source and Amazon Rekognition 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 one could read the image data as raw bytes. For example, the image data may need to reside in a S3 bucket in the Latin American 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 (for example: Virginia region) where Rekognition is supported.
3. Optionally, as you are indexing the faces, the image metadata and the Amazon S3 image URL can be stored in Amazon DynamoDB for a later lookup.
4. You can request a temporary Transactions Per Second (TPS) limit increase to speed up the indexing
5. Limit the collection(s) size is a key best practice (Refer to the section on **Best practices - Collections**);
   - Multi-threading can be used to leverage the increased TPS limit. You can use multiple EC2 instances, concurrent lambdas in conjunction with S3 Batch, and multiple containers using AWS Fargate service.
   - A good practice is to calculate how much TPS will be needed to index the historical faces in a certain amount of time, according to the business requirement to launch the workload.
6. In parallel with the temporary Transactions Per Second (TPS) limit increase, try to use scaling best practices including error handling and retry and exponential backoff in AWS SDK to make the index bath process smoother. An Amazon SQS Queue can be used if you run into throttling issues during the indexing process.



Indexing Faces using SQS to handle "throttling and retries"

- 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.

It is also important to keep in mind the possibility of duplicated faces in the historical image database. A best practice is to do a one-time indexing. Once the indexing is complete a SearchFacesByImage API on each photo can be executed to search for a duplicate. If a duplicate is found, a flag can be added in the image metadata storage (e.g., Amazon DynamoDB) and the necessary actions can be taken according to the business logic.

Best practices - Faces APIs

It is important to verify the face quality in the photos, over the webcam or mobile app. Having a high-quality face capture minimizes false positives and negatives during user onboarding and user login. To ensure a high-quality face capture, use DetectFaces API to detect faces and their attributes in an image. Attributes such as: range of face angles, face size (> 50x50 pixels) and facial attributes (check for any occlusions, e.g., sunglasses) can be used to validate the input image. The following example notebook executes the best practices to capture faces. It is recommended to run DetectFaces API prior to running CompareFaces, IndexFaces, SearchFacesByImage for any new user registration, and existing user registration. This will ensure the collections will have high quality face embedding and improve the overall accuracy of the face match.

When implementing a web or mobile app, use a face "frame" to drive a desired pose by the user, this can help ensure high quality face captures.

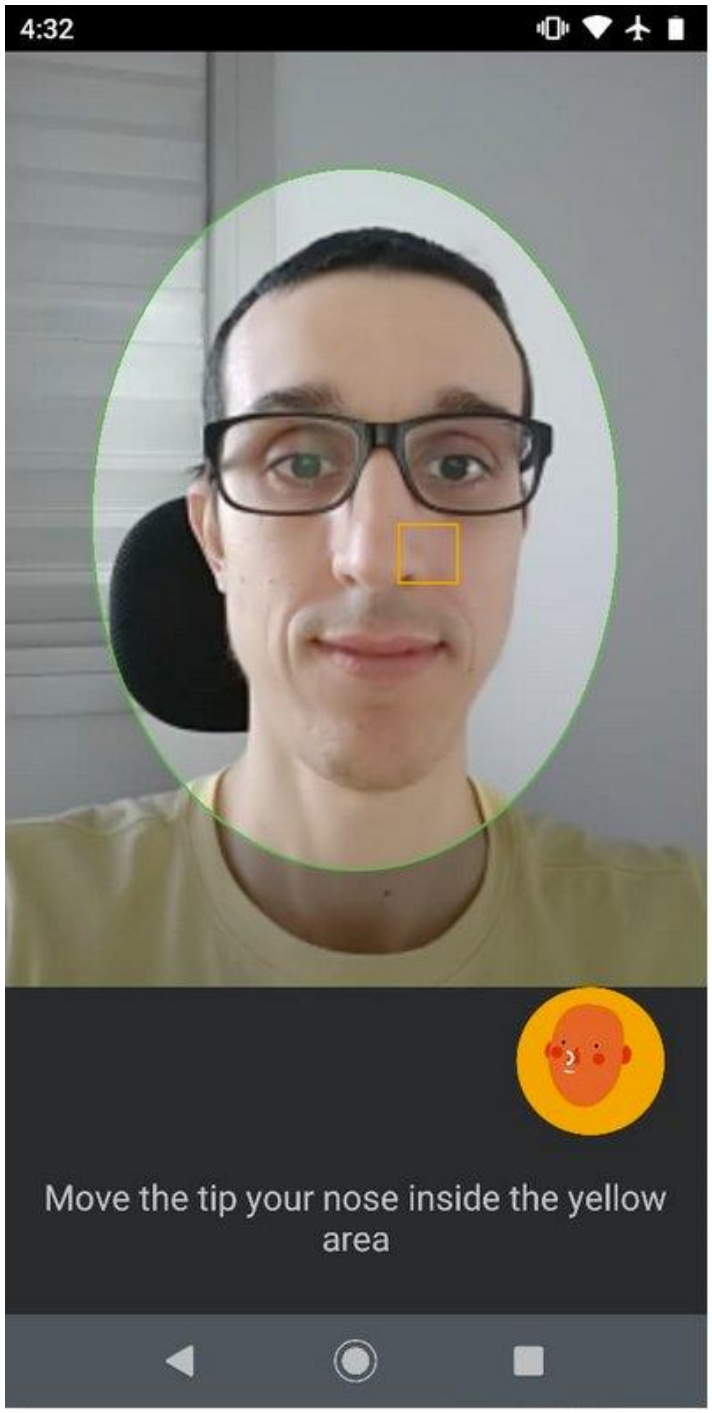
Face "frame" example

Some of the suggestions that can be included in capturing the user face are:

1. Color Indications:
    - Red for wrong position
    - Green for correct size
2. Clear guidance:
    - Take off accessories (cap, hat, glasses, id or something that partially or completely covers the face)
    - Move the hair behind the ears
    - Move the bangs on the back to keep your face neutral
    - Keep your eyes open and directed to the camera

- Keep their face as neutral as possible
- Position the camera so that you can see both ears
- The image cannot contain more than one face.
  - Only the face of the beneficiary to be registered should appear in the photo.

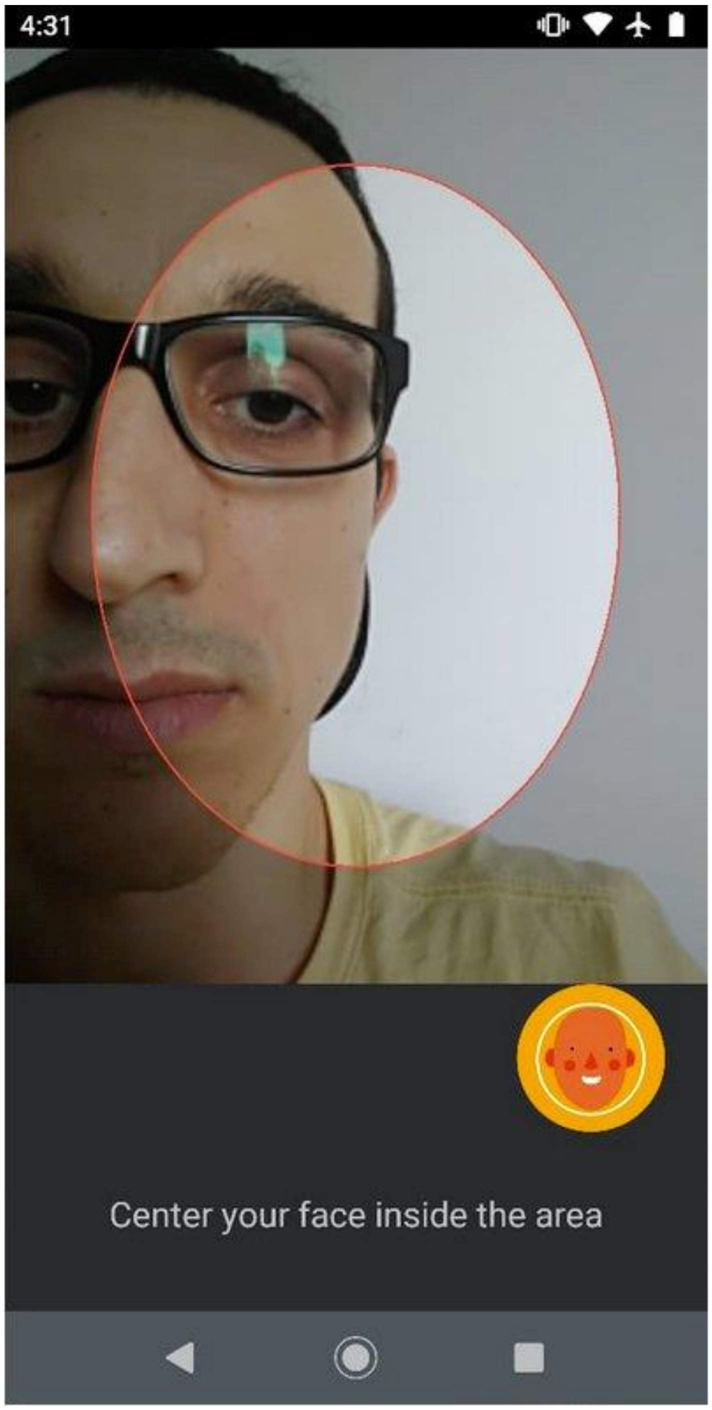
3. If a photo of a document is requested, include a frame for it as well



Move the tip your nose inside the yellow area

Green for correct size

Please refer to the best practices for use cases involving [public safety]

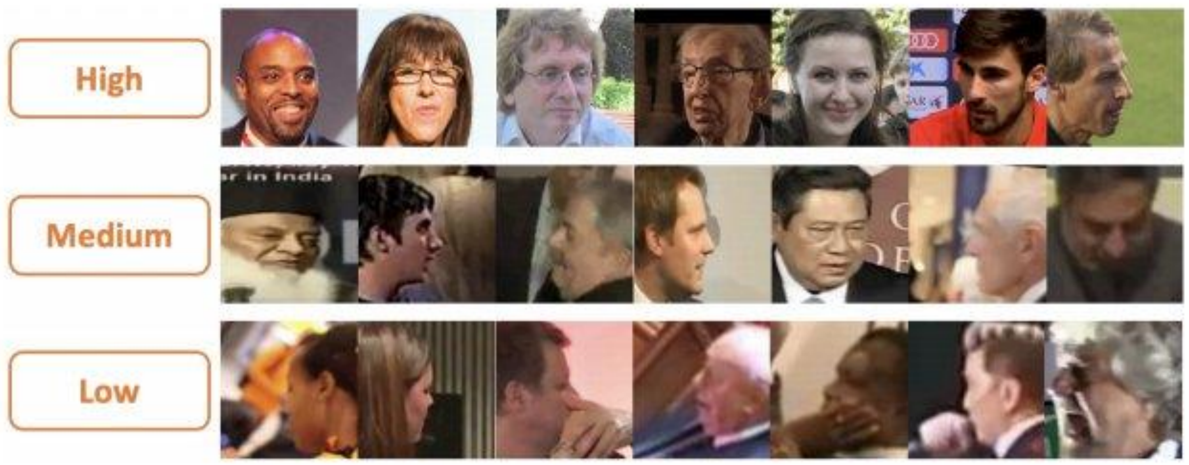


Red for wrong position

Please refer to the best practices for use cases involving public safety

Parameters

When using CompareFaces, IndexFaces or SearchFacesByImage API's, take advantage of the QualityFilter. The QualityFilter is a parameter that specifies a quality bar for how much filtering is done to identify faces. Set the QualityFilter to "HIGH". This is to ensure good source facial images will be considered during compare, indexing and search operations. In the following picture, you can see some example images for various settings of QualityFilter:



The QualityFilter parameter can also be helpful when dealing with photos that may have features that resemble faces, e.g., T-shirts with a movie character face. For identity verification use cases, it is recommended to try various settings in the range of 90 to 99% for the SimilarityThreshold (for CompareFaces API) and FaceMatchThreshold (for SearchFacesByImage API) parameters. A value closer to 90 will reduce the chances of a false reject (non-match). A value closer to 99 will reduce the chances of a false accept (match). Depending on your use case you can pick a value that results in an optimum business outcome. Please note, if there are multiple images of a single person in the collections, the search request is expected return multiple results. If the SearchFacesByImage API returns multiple matches above the default threshold, it is recommended to set a higher threshold or take the top ranked result with the highest confidence.

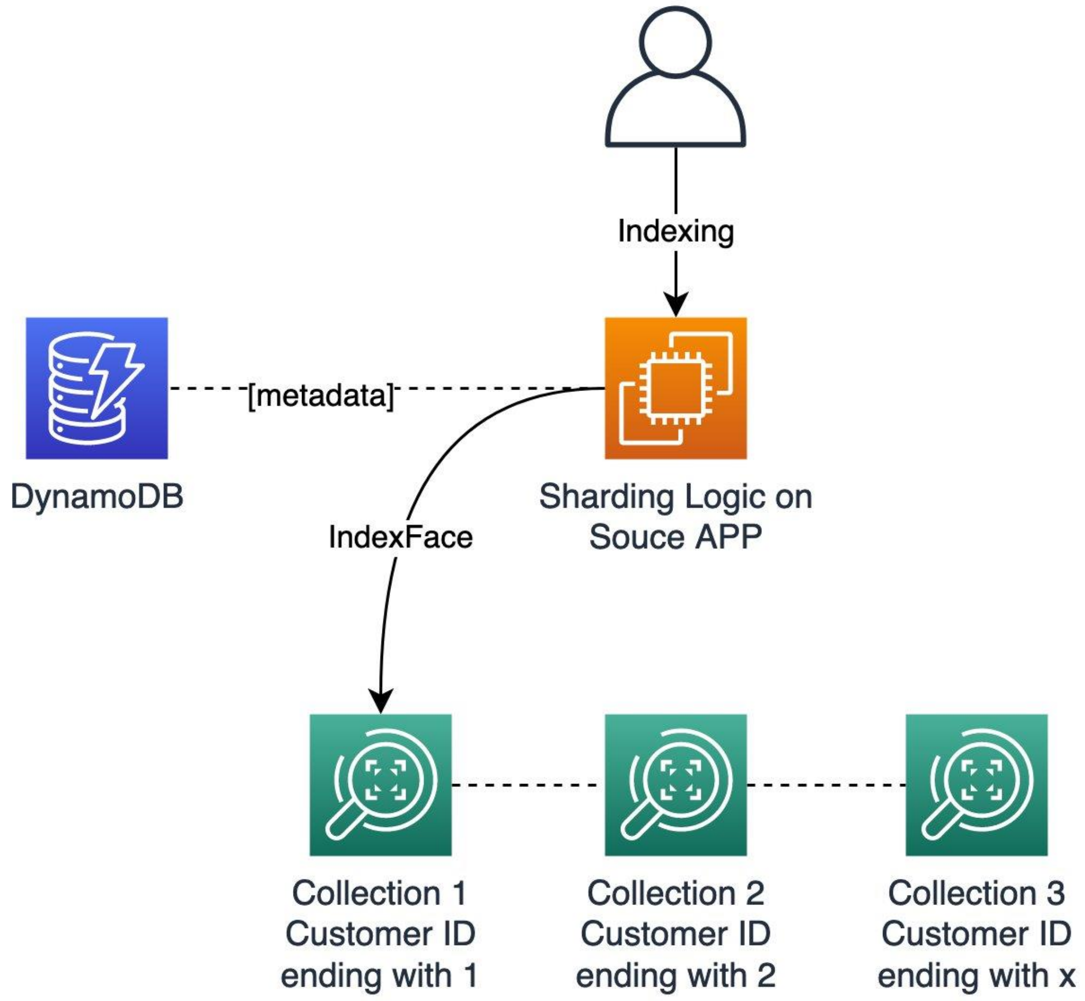
Collections

Depending on the total amount of faces you expect for your use case, sharding collections may be necessary. Shard is a technique of distributing faces to specific collections according to some metadata (name of the customer, first digit of the user unique id number, state, country). In addition, if the use case allows search for a specific collection, you can go straight to the right one, instead of searching across all collections to find a face match.

Each collection can have up to 20 million faces. If you have 100 million faces, you would need to shard across 5 collections. Here are some common sharding strategies that can be applied:

Indexing faces to a sharded collection

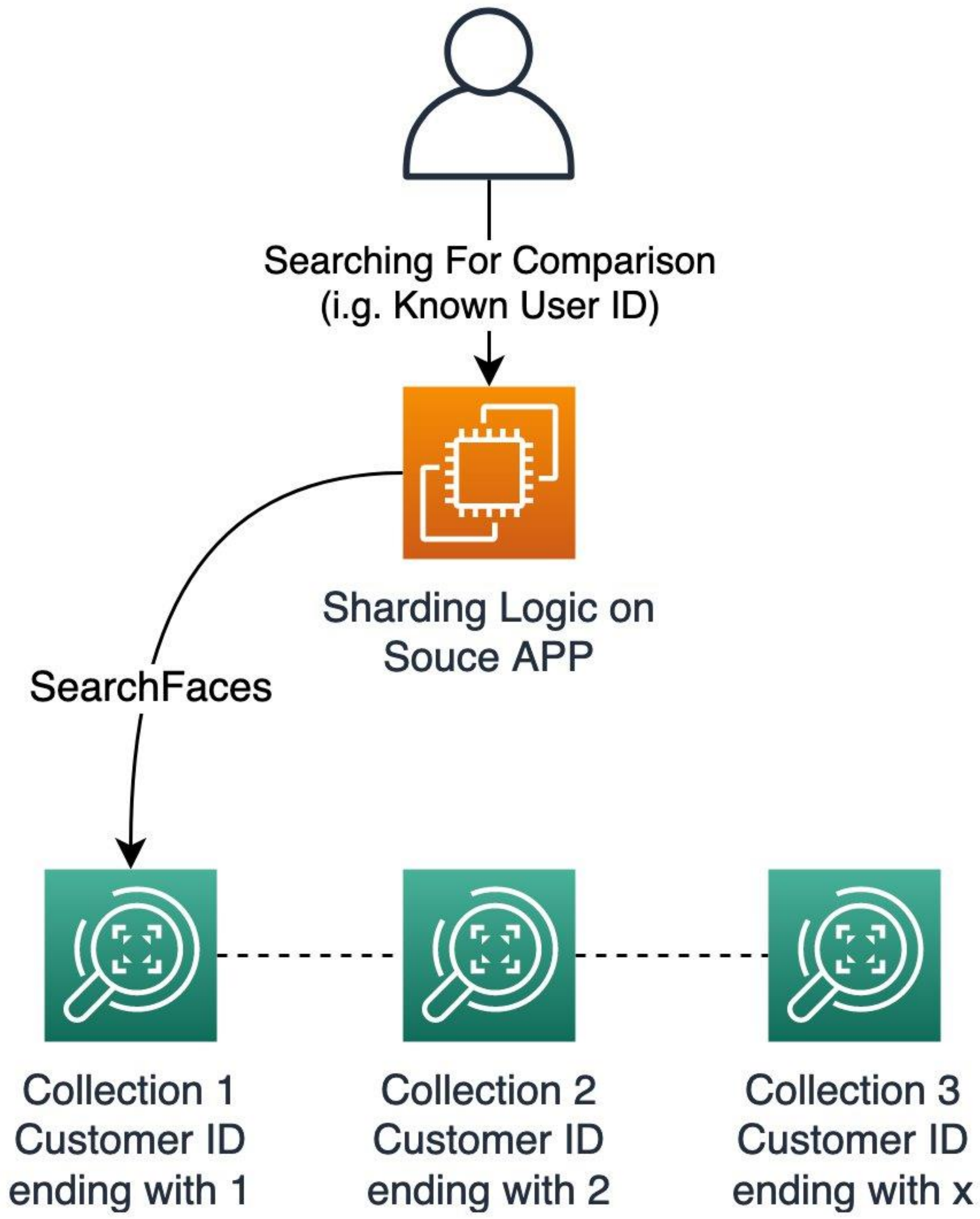
In the following diagram, collections have been sharded based on the unique ID (last. digit) of the user/customer. Faces are indexed into the collection that matches the last digit of the unique ID. DynamoDB is used as a metadata store. Depending on the use case other hashing strategies can be applied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



Example: Index Face on sharded collections

Searching for faces in a sharded collection

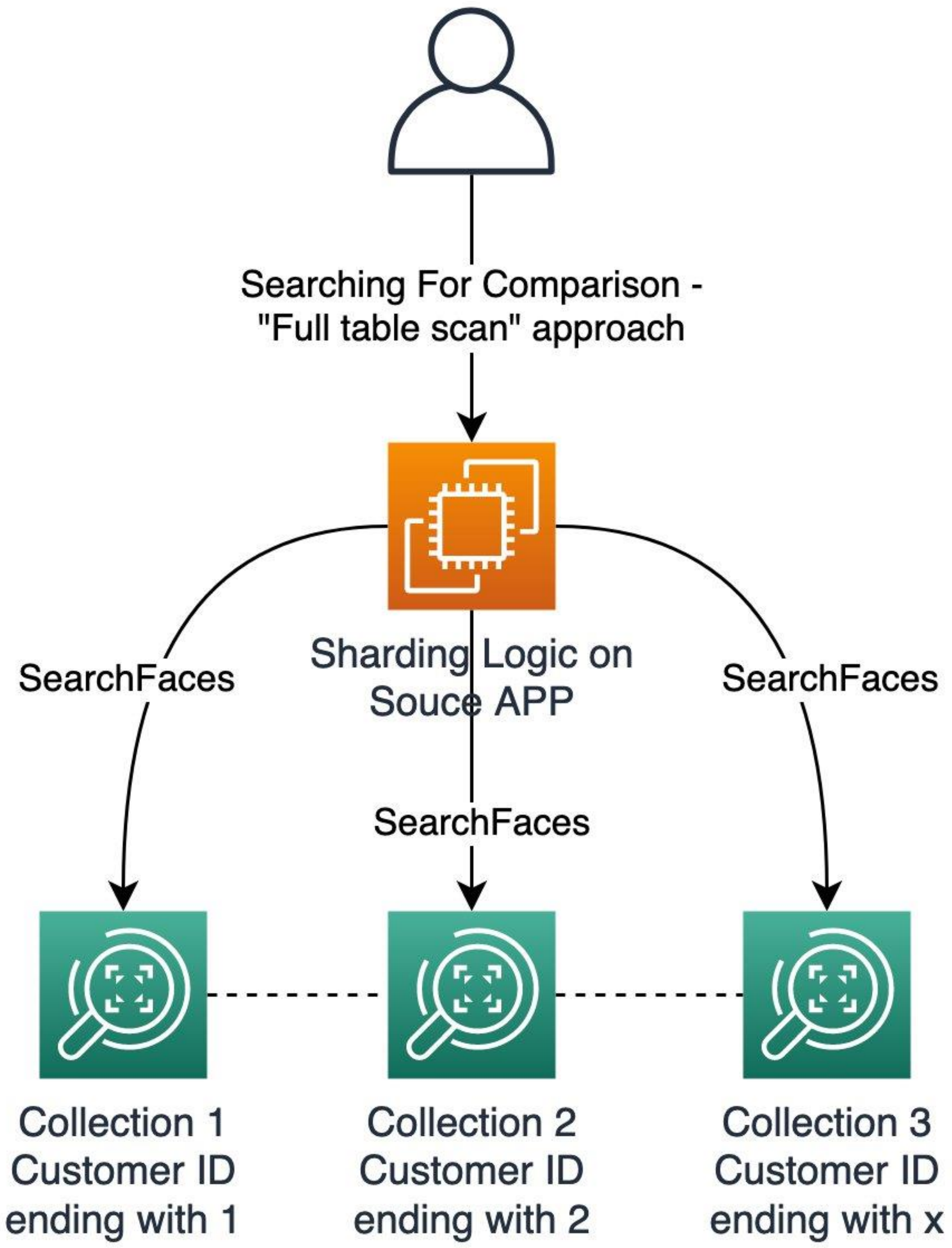
As it's a known user (i.e., they provided a text-based login already), the app only does one SearchFacesByImage on the shard.



Example: Search Face on sharded collections of a known Face

Searching for faces in a sharded collection - unknown user

Given this is an unknown user, the app does a SearchFacesByImage on every collection in parallel, similar to a “Full table scan” approach.



Example: Search Face on sharded collections of a unknown Face

Model Versioning

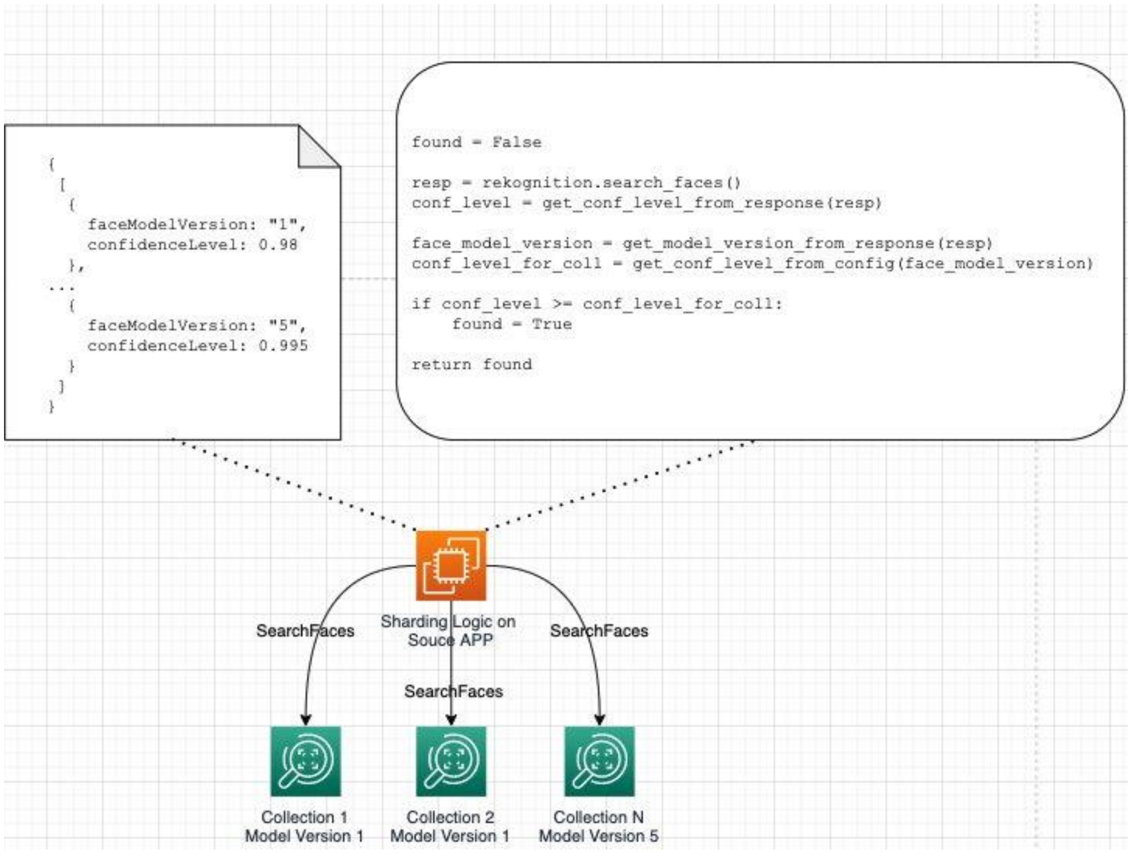
When creating a collection, the latest version of the model is to create the creation. This means that a collection will be forever tied to a specific model version. When a new model becomes available, we recommend re-indexing the collection with the new model to take advantage of any new features and to keep collections and confidence values consistent.

When using a sharded approach for managing collections (i.e., you have multiple collections) you can get into a situation where the collections have different model versions. You may find that the confidence levels between model versions vary. This is caused by Rekognition using different ML models to evaluate faces which gives you different confidence levels, although you should expect that newer models have more accurate predictions overall. Using the same confidence on different model versions may drive unexpected results and impact business process.

The following example highlights this. In face model version 1, we got a confidence level of 0.98 while in version 5 we get a confidence level of 0.995.

```
{
 [
  {
   faceModelVersion: "1",
   confidenceLevel: 0.98
  },
...
  {
   faceModelVersion: "5",
   confidenceLevel: 0.995
  }
 ]
}
```

Taking the above considerations into account, you should prepare to have different confidence levels for different model versions, prepare and test it before the deployment. This can be achieved using simple a configuration files where confidence level thresholds are linked to each model version.



```
{
  [
    {
      faceModelVersion: "1",
      confidenceLevel: 0.98
    },
  ...
    {
      faceModelVersion: "5",
      confidenceLevel: 0.995
    }
  ]
}
```

```
found = False

resp = rekognition.search_faces()
conf_level = get_conf_level_from_response(resp)

face_model_version = get_model_version_from_response(resp)
conf_level_for_coll = get_conf_level_from_config(face_model_version)

if conf_level >= conf_level_for_coll:
    found = True

return found
```

Sharding Logic on Souce APP
SearchFaces
SearchFaces
SearchFaces
Collection 1 Model Version 1
Collection 2 Model Version 1
Collection N Model Version 5

Searching for comparison of an unknown user - Sharding Logic

Limits

Amazon Rekognition provides service limits and quotas by API. You can request service limit increases if your application requires additional throughput. We recommend that you familiarize yourself with the default service quotas by region and refer to the section on best practices for TPS quotas to smooth traffic spikes, configure retries, and exponential backoff and jitter.

Liveness Detection

Comparing a photo with an official ID makes the registration process more secure, but also create new vectors of attacks. This includes impersonation, or the use of masks, photos, or videos. To address this, a liveness detection solution can be used.

Liveness detection is a technique used to detect a spoof attempt by determining whether the source of a biometric sample is a live human being or a fake representation. The overall benefit using a liveness detection solution is to improve the level of security while maintaining a relatively frictionless user registration experience.

Monitoring and Error Handling

You can start monitoring Rekognition with CloudWatch. You can get metrics for individual Rekognition operations or global Rekognition metrics for your account. You can 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. For example, you can see metrics for the number of server errors that have occurred, or metrics for the number of faces that have been detected. You can also see metrics for the number of times a specific Rekognition operation has succeeded.

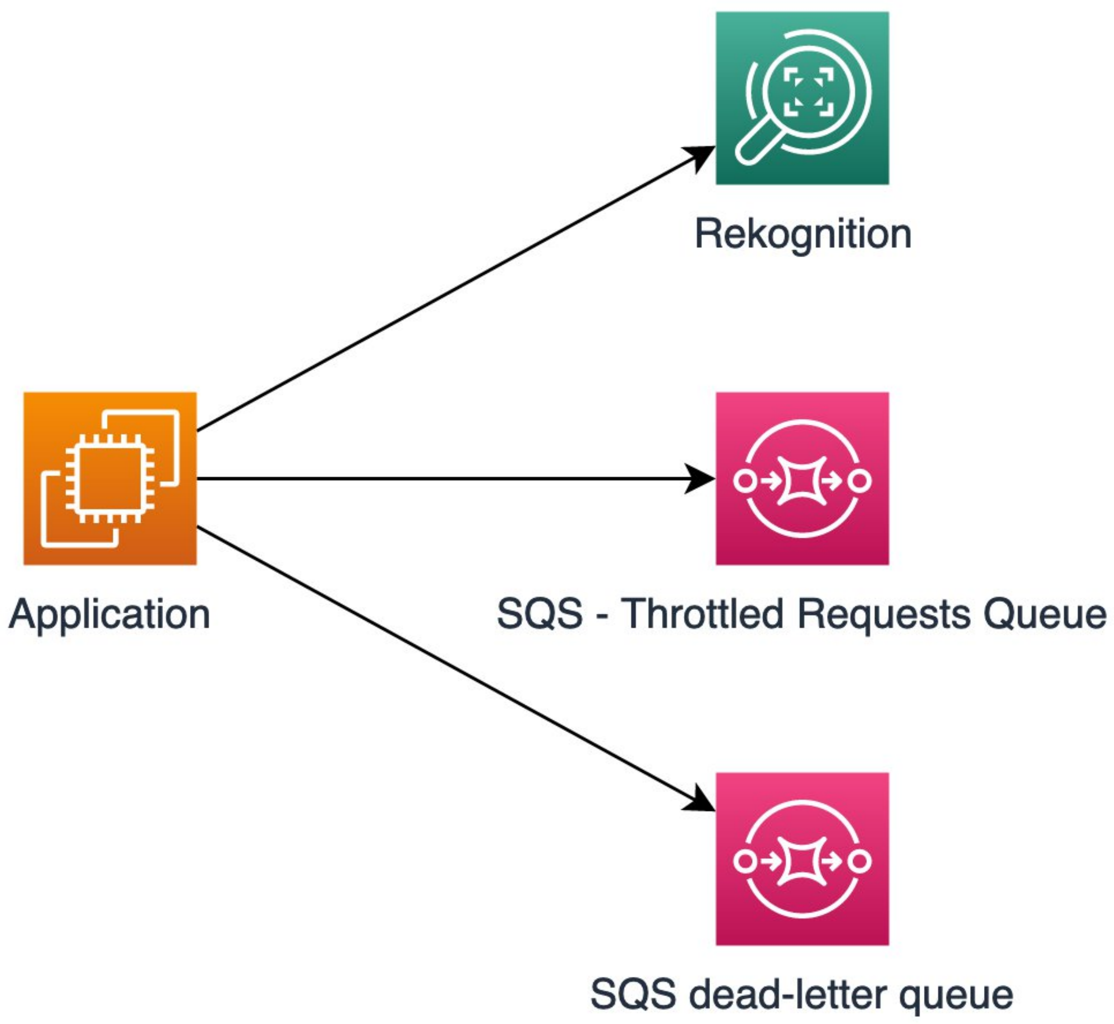
The Amazon Rekognition metrics pane shows activity graphs for an aggregate of individual Rekognition metrics over a specified period of time. For example, the `SuccessfulRequestCount` aggregated metric shows the total number of successful requests to all Rekognition API operations over the last seven days. Some of the things that should be tracked are:

Latency

You should monitor the latency for the operations in use CompareFaces, IndexFaces and SearchFaces. Increased latency in the operations may indicate an issue that should be investigated. You can set a fixed threshold or use CloudWatch anomaly detection to trigger alarms;

Errors

An increased number of errors, may indicate an issue. Errors may be caused due to different reasons and should be investigated. Use error handling best practices and retry and exponential backoff in AWS SDK. Read more about error retries and exponential backoff and timeouts and retries. Also, an Amazon SQS Queue can be used to handle throttling issues.



Application using Amazon SQS to handle Throttle. A [SQS dead-letter queue]() can be included for debugging

Business KPI's

Organizations implementing an identity verification solution should consider measuring application and system health as part of ongoing application monitoring. Application health can be divided into two aspects. First, we want to capture counts of successful, incomplete and failed registration and authentication attempts. This will give us an idea of the registration and authentication trends of our application overtime. Second, we want to capture results and metrics on calls to the key Amazon Rekognition APIs like DetectFaces, CompareFaces, and SearchFacesByImage. Capturing metrics from these APIs will help tune the performance of the identity verification solution. Finally, standard system performance monitoring of failures, latency, and throughput can help pinpoint potential pain-points in your application. The following are some metrics to consider instrumenting your application with:

Application KPIs:

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 over time. We recommend capturing the following application metrics:

1. Number of approved, incomplete and failed registrations
   - Count of incomplete, abandoned and failed registrations, recording the process step and reasons why a registration was incomplete, abandoned, or failed.
   - Count 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 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 a 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.
   - Count of abandoned authentications. Record the user, reason (if available), and details around the abandoned authentication attempt. This can often be an indication of attempted account takeover.

Comparison and Search KPIs

As part of your application, we recommend capturing metrics on Rekognition API calls for face detection, comparison, and search. Using these metrics you can monitor your application's detection, comparison and search performance, fine tune your API calls (ex. increasing similarity thresholds), and analyze the relationships between image quality, face comparison, and face search. We recommend capturing the following metrics and details:

1. Number of face comparisons and detections (ex. selfie to identification document), record number of faces found, potential obfuscation (ex. sunglasses, or PPE masks), bounding boxes, quality, facial landmarks and similarity.

2. Number of face detections, recording the number of faces by image and image quality, potential obfuscation (ex. hats, sunglasses, or PPE masks) and facial landmarks.
3. Number of face searches, recording the number of faces returned and their similarity.
4. Number of faces indexed, deleted, and updated.

Many of these metrics can be added to Amazon 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.

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.

Data protection in Amazon Rekognition

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.

Conclusion

Amazon Rekognition makes it easy to add image analysis to your identity verification applications using proven, highly scalable, deep learning technology that requires no machine learning expertise to use. Amazon Rekognition provides facial analysis and facial search capabilities. With a combination of DetectFaces, CompareFaces, IndexFaces and SearchFacesByImage APIs, you can implement the common flows around new user registration and existing user login's.

Amazon Rekognition Collections provides a method to store information about detected faces in server-side containers. You can then use the facial information stored in a collection to search for known faces in images. When using collections, you do not need to store original photos after you have indexed faces in the collections. Amazon Rekognition collections do not persist actual images. Instead, the underlying detection algorithm first detects the faces in the input image, extracts facial features into a feature vector for each face, and then stores it in the collection.

Contributors

Amit Gupta, Senior AI Services Solutions Architect

David Reis, Principal Solutions Architect

Felipe Brito, Solutions Architect

Karthi Thyagarajan, Principal Solution Architect

Mike Ames, AI Services Solutions Architect

Reviewers

Pradeep Parmar, Senior Product Manager

Tushar Agarwal, Principal Product Manager

Oliver Myers, Principal GTMS

Further Reading

For additional information, see:

1. Amazon Rekognition Documentation
2. Amazon Rekognition APIs