Amazon Rekognition

Developer Guide

Amazon Rekognition: Developer Guide

Table of Contents

What Is Amazon Rekognition?

Amazon Rekognition is a service that enables you to add image analysis to your applications. With Rekognition, you can detect objects, scenes, and faces in images. You can also search and compare faces. The Rekognition API enables you to quickly add sophisticated deep learning-based visual search and image classification to your applications. Rekognition is built to analyze images at scale and integrates seamlessly with Amazon S3, AWS Lambda, and other AWS services.

Common use cases for using Amazon Rekognition include the following:

- **Searchable image library** – Amazon Rekognition makes images searchable so you can discover objects and scenes that appear within them. You can create an AWS Lambda function that automatically adds newly detected image labels directly into an Amazon Elasticsearch Service search index when a new image is uploaded into Amazon S3.

- **Face-based user verification** – Amazon Rekognition can enable your applications to confirm user identities by comparing their live image with a reference image.

- **Sentiment and demographic analysis** – Amazon Rekognition detect emotions such as happy, sad, or surprise, and demographic information such as age and gender from facial images. Rekognition can analyze live images, and send the emotion and demographic attributes to Amazon Redshift for periodic reporting on trends to location such store locations.

- **Facial recognition** – With Amazon Rekognition, you can search your image collection for similar faces by storing faces, using the `IndexFaces` API operation. You can then use the `SearchFaces` operation to return high-confidence matches. A face collection is an index of faces that you own and manage. Identifying people based on their faces requires two major steps in Amazon Rekognition:

  1. Index the faces.
  2. Search the faces.

Some of the benefits of using Amazon Rekognition include:

- **Integrate powerful image recognition into your apps** – Amazon Rekognition removes the complexity of building image recognition capabilities into your applications by making powerful and accurate image analysis available with a simple API. You don't need computer vision or deep learning expertise to take advantage of Rekognition's reliable image analysis. With Rekognition's

API, you can easily and quickly build image analysis into any web, mobile or connected device application.

- **Deep learning-based image analysis** – Rekognition uses deep learning technology to accurately analyze images, find and compare faces, and detect objects and scenes within your images.

- **Scalable image analysis** – Amazon Rekognition enables you to analyze millions of images so you can curate and organize massive amounts of visual data.

- **Integrated with other AWS services** – Amazon Rekognition is designed to work seamlessly with other AWS services like Amazon S3 and AWS Lambda. Rekognition's API can be called directly from Lambda in response to Amazon S3 events. Since Amazon S3 and Lambda scale automatically in response to your application's demand, you can build scalable, affordable, and reliable image analysis applications. For example, each time a person arrives at your residence, your door camera can upload a photo of the visitor to Amazon S3, triggering a Lambda function that uses Rekognition API operations to identify your guest. You can run analysis directly on images stored in Amazon S3 without having to load or move the data. Support for AWS Identity and Access Management (IAM) makes it easy to securely control access to Rekognition API operations. Using IAM, you can create and manage AWS users and groups to grant the appropriate access to your developers and end users.

- **Low cost** – With Amazon Rekognition, you only pay for the number of images you analyze and the face metadata that you store. There are no minimum fees or upfront commitments. Get started for free, and save more as you grow with Rekognition's tiered pricing model.

Are You a First-Time Amazon Rekognition User?

If you are a first-time user of Amazon Rekognition, we recommend that you read the following sections in order:

1. **Amazon Rekognition: How It Works (p. 3)** – This section introduces various Amazon Rekognition components that you work with to create an end-to-end experience.
2. **Getting Started with Amazon Rekognition (p. 18)** – In this section you set your account and test the Amazon Rekognition API.
3. **Additional Amazon Rekognition Examples (p. 34)** – This section provides additional examples that you can use to explore Amazon Rekognition.

Amazon Rekognition: How It Works

The computer vision API operations that Amazon Rekognition provides can be grouped in the following categories:

- **Non-storage API operations** – The API operations in this group do not persist any information on the server. You provide input images, the API performs the analysis, and returns results, but nothing is saved on the server. The API can be used for operations such as the following:

  - Detect labels or faces in an image. A *label* refers to any of the following: objects (for example, flower, tree, or table), events (for example, a wedding, graduation, or birthday party), or concepts (for example, a landscape, evening, and nature). The input image you provide to these API operations can be in JPEG or PNG image format.
  - Compare faces in two images and return faces in the target image that match a face in the source image.

- **Storage-based API operations** – Amazon Rekognition provides an API operation that detects faces in the input image and persists facial feature vectors in a database on the server. Amazon Rekognition provides additional API operations you can use to search the persisted face vectors for face matches.

Topics

Non-Storage API Operations: Detecting Faces and Labels, and Comparing Faces

Amazon Rekognition provides the following non-storage API operations:

- `DetectLabels` to detect labels. This includes objects (for example, a flower, tree, or table), events (for example, a wedding, graduation, or debate), and concepts (for example, a landscape, adventure, or musical).
- `DetectFaces` to detect faces.
- `CompareFaces` to compare faces in images.

These are referred to as *non-storage* API operations because when you make the API call, Amazon Rekognition does not persist the input image or any image data.

The following example scenarios show where you might integrate non-storage API operations in your application. These scenarios assume you have a local repository of images.

Example 1: An application that finds images in your local repository that contain specific labels

First, you detect labels using the Amazon Rekognition `DetectLabels` operation in each of the images in your repository and build a client-side index, as shown following:

```
Label        ImageID

tree          image-1
flower        image-1
mountain      image-1
tulip         image-2
flower        image-2
apple         image-3
```

Then, your application can search this index to find images in your local repository that contain a specific label. For example, display images that contain a tree.

Each label that Amazon Rekognition detects has a confidence value associated. It indicates the level of confidence that the input image contains that label. You can use this confidence value to optionally perform additional client-side filtering on labels depending on your application requirements about the level of confidence in the detection. For example, if you require extremely precise labels, you might filter and choose only the labels with higher confidence (such as 95% or higher). If you application does not require higher confidence value, you might choose to filter labels with lower confidence value (closer to 50%).

Example 2: An application to display enhanced face images

First, you can detect faces in each of the images in your local repository using the Amazon Rekognition `DetectFaces` operation and build a client-side index. For each face, the operation returns metadata that include a bounding box, facial landmarks (for example, position of mouth and ear), and facial attributes (for example, gender). You can store this metadata in a client-side local index, as shown following:

```
ImageID     FaceID     FaceMetaData

image-1     face-1     <boundingbox>, etc.
image-1     face-2     <boundingbox>, etc.
image-1     face-3     <boundingbox>, etc.
...
```

In this index, the primary key is a combination of both the `ImageID` and `FaceID`.

Then, you can use the information in the index to enhance the images when your application displays them from your local repository. For example you might add a bounding box around the face or highlight facial features.

Related Topics

- Detecting Labels and Faces (p. 5)

- Comparing Faces (p. 11)

Detecting Labels and Faces

Amazon Rekognition provides non-storage API operations for detecting labels and faces in an image. A label or a tag is an object, scene or concept found in an image based on its contents. For example, a photo of people on a tropical beach may contain labels such as Person, Water, Sand, Palm Tree, and Swimwear (objects), Beach (scene) and Outdoors (concept).

These are referred to as the *non-storage* API operations because when you make the API call, Amazon Rekognition does not persist the input image or any image data. The API operations do the necessary analysis and return the results. The sections in this topic describe these operations.

Topics

- Detecting Labels (p. 5)
- Detecting Faces (p. 6)

Detecting Labels

You can use the DetectLabels (p. 65) API operation to detect labels in an image. For each label, Amazon Rekognition returns a name and a confidence value in the analysis. The following is an example response of the `DetectLabels` API call.

```
{
    "Labels": [
        {
            "Confidence": 98.4629,
            "Name": "beacon"
        },
        {
            "Confidence": 98.4629,
            "Name": "building"
        },
        {
            "Confidence": 98.4629,
            "Name": "lighthouse"
        },
        {
            "Confidence": 87.7924,
            "Name": "rock"
        },
        {
            "Confidence": 68.1049,
            "Name": "sea"
        }
    ]
}
```

The response shows that the API detected five labels (that is, beacon, building, lighthouse, rock, and sea). Each label has an associated level of confidence. For example, the detection algorithm is 98.4629% confident that the image contains a building.

If the input image you provide contains a person, the `DetectLabels` operation detects labels such as person, clothing, suit, and selfie, as shown in the following example response:

```
{
    "Labels": [
        {
            "Confidence": 99.2786,
            "Name": "person"
        },
        {
            "Confidence": 90.6659,
            "Name": "clothing"
        },
        {
            "Confidence": 90.6659,
            "Name": "suit"
        },
        {
            "Confidence": 70.0364,
            "Name": "selfie"
        }
    ]
}
```

Note
If you want facial features describing the faces in an image, use the `DetectFaces` operation instead.

Detecting Faces

Amazon Rekognition provides the DetectFaces (p. 61) operation that looks for key facial features such as eyes, nose, and mouth to detect faces in an input image. The response returns the following information for each detected face:

- **Bounding box** – Coordinates of the bounding box surrounding the face.
- **Confidence** – Level of confidence that the bounding box contains a face.
- **Facial landmarks** – An array of facial landmarks. For each landmark, such as the left eye, right eye, and mouth, the response provides the x, y coordinates.
- **Facial attributes** – A set of facial attributes, including gender, or whether the face has a beard. For each such attribute, the response provides a value. The value can be of different types such as a Boolean (whether or not person is wearing sunglasses), a string (whether the person is male or female), etc. In addition, for most attributes the response also provides a confidence in the detected value for the attribute.
- **Quality** – Describes the brightness and the sharpness of the face.
- **Pose** – Describes the rotation of the face inside the image.
- **Emotions** – A set of emotions with confidence in the analysis.

The following is an example response of a `DetectFaces` API call.

```
{
    "FaceDetails": [
        {
            "Confidence": 99.99968719482422,
            "Eyeglasses": {
                "Confidence": 99.94019317626953,
                "Value": false
            },
```

```
            "Sunglasses": {
                "Confidence": 99.62261199951172,
                "Value": false
            },
            "Gender": {
                "Confidence": 99.92701721191406,
                "Value": "Male"
            },
            "Pose": {
                "Yaw": 1.8526556491851807,
                "Roll": 3.623055934906006,
                "Pitch": -10.605680465698242
            },
            "Emotions": [
                {
                    "Confidence": 99.38518524169922,
                    "Type": "HAPPY"
                },
                {
                    "Confidence": 1.1799871921539307,
                    "Type": "ANGRY"
                },
                {
                    "Confidence": 1.0325908660888672,
                    "Type": "CONFUSED"
                }
            ],
            "EyesOpen": {
                "Confidence": 54.15227508544922,
                "Value": false
            },
            "Quality": {
                "Sharpness": 130.0,
                "Brightness": 49.129302978515625
            },
            "BoundingBox": {
                "Width": 0.6154,
                "Top": 0.2442,
                "Left": 0.1765,
                "Height": 0.4692
            },
            "Smile": {
                "Confidence": 99.8236083984375,
                "Value": true
            },
            "MouthOpen": {
                "Confidence": 88.39942169189453,
                "Value": true
            },
            "Landmarks": [
                {
                    "Y": 0.41730427742004395,
                    "X": 0.36835095286369324,
                    "Type": "eyeLeft"
                },
                {
                    "Y": 0.4281611740589142,
                    "X": 0.5960656404495239,
                    "Type": "eyeRight"
```

```
        },
        {
            "Y": 0.5349795818328857,
            "X": 0.47817257046699524,
            "Type": "nose"
        },
        {
            "Y": 0.5721957683563232,
            "X": 0.352621465921402,
            "Type": "mouthLeft"
        },
        {
            "Y": 0.5792245864868164,
            "X": 0.5936088562011719,
            "Type": "mouthRight"
        },
        {
            "Y": 0.4163532555103302,
            "X": 0.3697868585586548,
            "Type": "leftPupil"
        },
        {
            "Y": 0.42626339197158813,
            "X": 0.6037314534187317,
            "Type": "rightPupil"
        },
        {
            "Y": 0.38954615592956543,
            "X": 0.27343833446502686,
            "Type": "leftEyeBrowLeft"
        },
        {
            "Y": 0.3775958716869354,
            "X": 0.35098740458488464,
            "Type": "leftEyeBrowRight"
        },
        {
            "Y": 0.39108505845069885,
            "X": 0.433648943901062,
            "Type": "leftEyeBrowUp"
        },
        {
            "Y": 0.3952394127845764,
            "X": 0.5416828989982605,
            "Type": "rightEyeBrowLeft"
        },
        {
            "Y": 0.38667190074920654,
            "X": 0.6171167492866516,
            "Type": "rightEyeBrowRight"
        },
        {
            "Y": 0.40419116616249084,
            "X": 0.6827319264411926,
            "Type": "rightEyeBrowUp"
        },
        {
            "Y": 0.41925403475761414,
            "X": 0.32195475697517395,
```

```
                "Type": "leftEyeLeft"
            },
            {
                "Y": 0.4225293695926666,
                "X": 0.41227561235427856,
                "Type": "leftEyeRight"
            },
            {
                "Y": 0.4096950888633728,
                "X": 0.3705553412437439,
                "Type": "leftEyeUp"
            },
            {
                "Y": 0.4213259816169739,
                "X": 0.36738231778144836,
                "Type": "leftEyeDown"
            },
            {
                "Y": 0.4294262230396271,
                "X": 0.5498995184898376,
                "Type": "rightEyeLeft"
            },
            {
                "Y": 0.4327501356601715,
                "X": 0.6390777826309204,
                "Type": "rightEyeRight"
            },
            {
                "Y": 0.42076829075813293,
                "X": 0.5977370738983154,
                "Type": "rightEyeUp"
            },
            {
                "Y": 0.4326271116733551,
                "X": 0.5959710478782654,
                "Type": "rightEyeDown"
            },
            {
                "Y": 0.5411174893379211,
                "X": 0.4253743588924408,
                "Type": "noseLeft"
            },
            {
                "Y": 0.5450678467750549,
                "X": 0.5309309959411621,
                "Type": "noseRight"
            },
            {
                "Y": 0.5795656442642212,
                "X": 0.47389525175094604,
                "Type": "mouthUp"
            },
            {
                "Y": 0.6466911435127258,
                "X": 0.47393468022346497,
                "Type": "mouthDown"
            }
        ],
        "Mustache": {
```

```
                "Confidence": 99.75302124023438,
                "Value": false
            },
            "Beard": {
                "Confidence": 89.82911682128906,
                "Value": false
            }
        }
    ]
}
```

Note the following:

- The `Pose` data describes the rotation of the face detected. You can use the combination of the `BoundingBox` and `Pose` data to draw the bounding box around faces that your application displays.

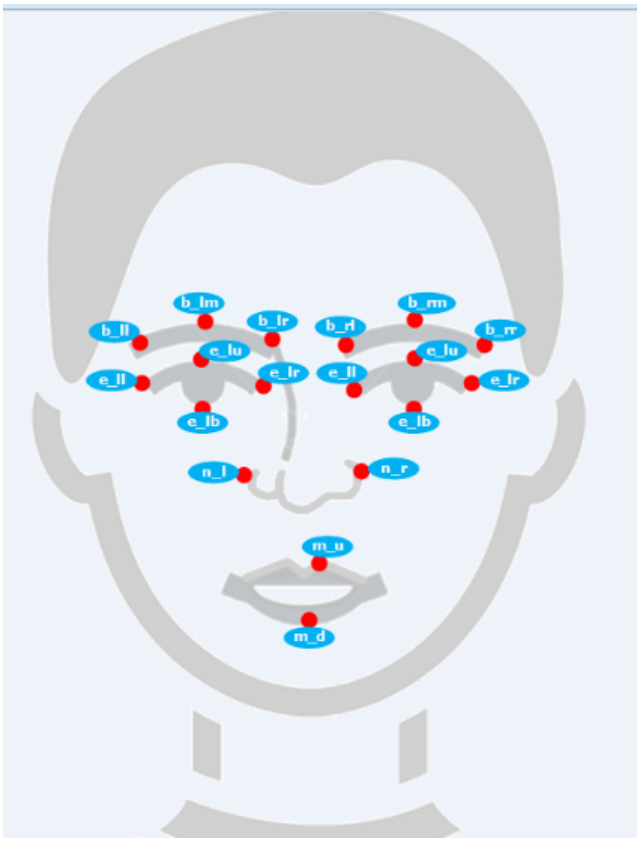
- The `Quality` describes the brightness and the sharpness of the face. You might find this useful to compare faces across images and find the best face.

- The `DetectFaces` operation first detects orientation of the input image, before detecting facial features. The `OrientationCorrection` in the response returns the degrees of rotation detected (clockwise direction). Your application can use this value to correct the image orientation when displaying the image.
- The preceding response shows all facial `landmarks` the service can detect, all facial attributes and emotions. To get all of these in the response, you must specify the `attributes` parameter with value `ALL`. By default, the `DetectFaces` API returns only the following five facial `landmarks`, `Pose`, and `Quality`.

```
...
   "Landmarks": [
        {
            "Y": 0.41730427742004395,
            "X": 0.36835095286369324,
            "Type": "eyeLeft"
        },
        {
            "Y": 0.4281611740589142,
            "X": 0.5960656404495239,
            "Type": "eyeRight"
        },
        {
            "Y": 0.5349795818328857,
            "X": 0.47817257046699524,
            "Type": "nose"
        },
        {
            "Y": 0.5721957683563232,
            "X": 0.352621465921402,
            "Type": "mouthLeft"
        },
        {
            "Y": 0.5792245864868164,
            "X": 0.5936088562011719,
            "Type": "mouthRight"
        }
```

```
    ]
...
```

- The following illustration shows the relative location of the facial landmarks on the face returned by the `DetectFaces` API operation.



Comparing Faces

Amazon Rekognition provides the CompareFaces (p. 52) operation to compare a face in the *source* image with each face in the *target* image.

> **Note**
> If the source image contains more than one face, the service detects the largest face and uses it for comparison.

The API returns an array of face matches as shown in the following example response.

```
{
    "FaceMatches": [
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.495,
                    "Top": 0.2211,
                    "Left": 0.3069,
```

```
                    "Height": 0.3333
                },
                "Confidence": 99.99949645996094
            },
            "Similarity": 92.0
        }
    ],
    "SourceImageFace": {
        "BoundingBox": {
            "Width": 0.6154,
            "Top": 0.2442,
            "Left": 0.1765,
            "Height": 0.4692
        },
        "Confidence": 99.99968719482422
    }
}
```

In the response, note the following:

- **Face match information** – The example shows one face match found in the target image. For that face match, it provides a bounding box and a confidence value (level of confidence that the bounding box contains a face). The `similarity` score of 94.0 indicates how similar the faces are.

  The `similarityThreshold` in the request, determines the minimum level of confidence in the match that you want returned in the response. For more information, see CompareFaces (p. 52).

  If multiple face matches are found, the `faceMatches` (an array) returns all of the face matches.
- **Source face information** – The response also includes information (the bounding box and the confidence) about the face from the source that was used for comparison.

Storage-Based API Operations: Storing Faces and Searching Face Matches

Amazon Rekognition supports the IndexFaces (p. 68) operation, which you can use to detect faces in an image and persist information about facial features detected in a database on the server. This is an example of a *storage-based* API operation because the service persists information on the server.

To store facial information you must first create a face collection in one of AWS Regions in your account. You specify this face collection when you call the `IndexFaces` operation. After you create a face collection and store facial feature information for all faces, you can search the collection for face matches.

> **Note**
> The service does not 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 database. Amazon Rekognition uses these feature vectors when performing face matches.

For example, you might create a face collection to store scanned badge images using the `IndexFaces` operation, which extracts faces and stores them as searchable image vectors. 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.

Managing Face Collections

A collection is a container for persisting faces detected by the `IndexFaces` API. You might choose to create one container to store all faces or create multiple containers to store faces in groups as you choose. Consider the following examples:

- You might create a collection to store scanned badge images using the `IndexFaces` operation, which extracts faces and stores them as searchable image vectors. When an employee enters the building, an image of their face is captured and sent to the `SearchFacesByImage` operation. If the face match produces a sufficiently high `similarity` score, the employee is immediately verified.

  As a developer of identity verification system, you can use a sufficiently high (99%) similarity score
- You might create multiple collections, one per application user so that their uploaded faces are grouped independently. In this scenario, when a user performs a search, the search is scoped to the user's face collection (the search faces operations require a collection ID as input).

You might also choose to create one face collection for each of your application users so that their uploaded faces are grouped independently. In this scenario, when a user performs a search, the search is scoped to the user's face collection (the search faces operations require a collection ID as input).

The face collection is the primary Amazon Rekognition resource, each face collection you create has a unique Amazon Resource Name (ARN). You create each face collection in a specific AWS Region in your account.

Amazon Rekognition provides the following operations for you to manage collections:

- CreateCollection (p. 55)
- DeleteCollection (p. 57)
- ListCollections (p. 72)

For information about storing faces, see Storing Faces In a Face Collection: The IndexFaces Operation (p. 13). For information about searching faces, see Searching Faces In a Face Collection (p. 15).

Storing Faces In a Face Collection: The IndexFaces Operation

After you create a face collection, you can store faces in it. Amazon Rekognition provides the `IndexFaces` operation that can detect faces in the input image (JPEG or PNG) and adds them to the specified face collection. For more information about collections, see Managing Face Collections (p. 13). After you persist faces, you can search the face collection for face matches.

Important
Amazon Rekognition does not save the actual faces detected. Instead, the underlying detection algorithm first detects the faces in the input image, extracts facial features for each

face, and then stores the feature information in a database. Then, Amazon Rekognition uses this information in subsequent operations such as searching a face collection for matching faces.

For each face, the `IndexFaces` operation persists the following information:

- **Multidimensional facial features** – `IndexFaces` uses facial analysis to extract multidimensional information about the facial features and stores the information in the face collection. You cannot access this information directly. However, Amazon Rekognition uses this information when searching a face collection for face matches.

- **Metadata** – The metadata for each face includes a bounding box, confidence level (that the bounding box contains a face), IDs assigned by Amazon Rekognition (face ID and image ID), and an external image ID (if you provided it) in the request. This information is returned to you in response to the `IndexFaces` API call. For an example, see the `face` element in the following example response.

  The service returns this metadata in response to the following API calls:

  - `ListFaces`
  - Search faces operations – The responses for `SearchFaces` and `SearchFacesByImage` return the confidence in the match for each matching face, along with this metadata of the matched face.

In addition to the preceding information that the API persists in the face collection, the API also returns face details that are not persisted in the collection (see the `faceDetail` element in the following example response).

Note
This is the same information that `DetectFaces` returns, so you don't need to call both `DetectFaces` and `IndexFaces` for the same image.


```
{
    "FaceRecords": [
        {
            "FaceDetail": {
                "BoundingBox": {
                    "Width": 0.6154,
                    "Top": 0.2442,
                    "Left": 0.1765,
                    "Height": 0.4692
                },
                "Landmarks": [
                    {
                        "Y": 0.41730427742004395,
                        "X": 0.36835095286369324,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.4281611740589142,
                        "X": 0.5960656404495239,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.5349795818328857,
                        "X": 0.47817257046699524,
                        "Type": "nose"
```

```
                    },
                    {
                        "Y": 0.5721957683563232,
                        "X": 0.352621465921402,
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.5792245864868164,
                        "X": 0.5936088562011719,
                        "Type": "mouthRight"
                    }
                ],
                "Pose": {
                    "Yaw": 1.8526556491851807,
                    "Roll": 3.623055934906006,
                    "Pitch": -10.605680465698242
                },
                "Quality": {
                    "Sharpness": 130.0,
                    "Brightness": 49.129302978515625
                },
                "Confidence": 99.99968719482422
            },
            "Face": {
                "BoundingBox": {
                    "Width": 0.6154,
                    "Top": 0.2442,
                    "Left": 0.1765,
                    "Height": 0.4692
                },
                "FaceId": "84de1c86-5059-53f2-a432-34ebb704615d",
                "Confidence": 99.9997,
                "ImageId": "d38ebf91-1a11-58fc-ba42-f978b3f32f60"
            }
        }
    ],
    "OrientationCorrection": "ROTATE_0"
}
```


Searching Faces In a Face Collection

After you create a face collection and store faces, you can search a face collection for face matches. For more information about storing faces in a face collection, see Managing Face Collections (p. 13) and Storing Faces In a Face Collection: The IndexFaces Operation (p. 13). With Amazon Rekognition, you can do the following:

- **Search a face collection given an image (SearchFacesByImage (p. 79))** – For a given input image (JPEG or PNG), the operation first detects the face in the input image, and then searches the specified face collection for similar faces.

  Note
  If the service detects multiple faces in the input image, it uses the largest face detected for searching the face collection.

  The operation returns an array of face matches found, and information about the input face (such as the bounding box, along with the confidence value that indicates the level of confidence that the bounding box contains a face).

```
{
    "SearchedFaceBoundingBox": {
        "Width": 0.6154,
        "Top": 0.2442,
        "Left": 0.1765,
        "Height": 0.4692
    },
    "SearchedFaceConfidence": 99.9997,
    "FaceMatches": [ list of face matches found ]
}
```

- **Search a face collection given a face ID (SearchFaces (p. 76))** – Given a face ID (each face stored in the face collection has a face ID), `SearchFaces` searches the specified face collection for the similar faces. The response doesn't include the face you are searching for, it includes only similar faces.

  The operation returns an array of face matches found and the face ID you provided as input.

```
{
    "SearchedFaceId": "7ecf8c19-5274-5917-9c91-1db9ae0449e2",
    "FaceMatches": [ list of face matches found ]
}
```

  For example, the `SearchFacesByImage` API performs a search using the largest face in the input image. If you want to search for other faces in the input image, you might first index all faces using the `IndexFaces` API. You get a face ID in response. You can then use `SearchFaces` API to search for faces using the face IDs.

By default, both of these API operations return faces where the algorithm detects similarity of greater than 80%. The similarity indicates how closely the face matches with the input face. Optionally, you can use `FaceMatchThreshold` to specify a different value. For each face match found, the response includes similarity and face metadata as shown in the following example response:

```
{
   ...
    "FaceMatches": [
        {
            "Similarity": 100.0,
            "Face": {
                "BoundingBox": {
                    "Width": 0.6154,
                    "Top": 0.2442,
                    "Left": 0.1765,
                    "Height": 0.4692
                },
                "FaceId": "84de1c86-5059-53f2-a432-34ebb704615d",
                "Confidence": 99.9997,
                "ImageId": "d38ebf91-1a11-58fc-ba42-f978b3f32f60"
            }
        },
        {
            "Similarity": 84.6859,
            "Face": {
```

```
                "BoundingBox": {
                    "Width": 0.2044,
                    "Top": 0.2254,
                    "Left": 0.4622,
                    "Height": 0.3119
                },
                "FaceId": "6fc892c7-5739-50da-a0d7-80cc92c0ba54",
                "Confidence": 99.9981,
                "ImageId": "5d913eaf-cf7f-5e09-8c8f-cb1bdea8e6aa"
            }
        }
    ]
}
```

Note that the `CompareFaces` operation and the two search faces API operations differ as follows:

- The `CompareFaces` operation compares a face in a source image with faces in the target image. The scope of this comparison is limited to the faces detected in the target image. For more information, see Comparing Faces (p. 11).
- `SearchFaces` and `SearchFacesByImage` compare a face (identified either by a `FaceId` or an input image) with all faces in a given face collection. Therefore, the scope of this search is much larger. Also, because the facial feature information is persisted for faces already stored in the face collection, you can search for matching faces multiple times.

Getting Started with Amazon Rekognition

This section provides topics to get you started using Amazon Rekognition. If you are new to Amazon Rekognition, we recommend that you first review the concepts and terminology presented in Amazon Rekognition: How It Works (p. 3).

Topics

Step 1: Set Up an AWS Account and Create an Administrator User

Before you use Amazon Rekognition for the first time, complete the following tasks:

1. Sign up for AWS (p. 18)
2. Create an IAM User (p. 19)

Sign up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all services in AWS, including Amazon Rekognition. You are charged only for the services that you use.

With Amazon Rekognition, you pay only for the resources you use. If you are a new AWS customer, you can get started with Amazon Rekognition for free. For more information, see AWS Free Usage Tier.

If you already have an AWS account, skip to the next task. If you don't have an AWS account, perform the steps in the following procedure to create one.

To create an AWS account

1. Open https://aws.amazon.com/, and then choose **Create an AWS Account**.

2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

Note your AWS account ID because you'll need it for the next task.

Create an IAM User

Services in AWS, such as Amazon Rekognition, require that you provide credentials when you access them so that the service can determine whether you have permissions to access the resources owned by that service. The console requires your password. You can create access keys for your AWS account to access the AWS CLI or API. However, we don't recommend that you access AWS using the credentials for your AWS account. Instead, we recommend that you use AWS Identity and Access Management (IAM). Create an IAM user, add the user to an IAM group with administrative permissions, and then grant administrative permissions to the IAM user that you created. You can then access AWS using a special URL and that IAM user's credentials.

If you signed up for AWS, but you haven't created an IAM user for yourself, you can create one using the IAM console.

The Getting Started exercises in this guide assume that you have a user (`adminuser`) with administrator privileges. Follow the procedure to create `adminuser` in your account.

To create an administrator user and sign in to the console

1. Create an administrator user called `adminuser` in your AWS account. For instructions, see Creating Your First IAM User and Administrators Group in the *IAM User Guide*.
2. A user can sign in to the AWS Management Console using a special URL. For more information, How Users Sign In to Your Account in the *IAM User Guide*.

For more information about IAM, see the following:

- Identity and Access Management (IAM)
- Getting Started
- IAM User Guide

Next Step

Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 19)

Step 2: Set Up the AWS Command Line Interface (AWS CLI)

Follow the steps to download and configure the AWS Command Line Interface (AWS CLI).

Important
You don't need the AWS CLI to perform the steps in the Getting Started exercise. However, some of the exercises in this guide use the AWS CLI. You can skip this step and go to Step 4: Getting Started Using API (p. 26), and then set up the AWS CLI later when you need it.

To set up the AWS CLI

1. Download and configure the AWS CLI. For instructions, see the following topics in the *AWS Command Line Interface User Guide*:

   - Getting Set Up with the AWS Command Line Interface
   - Configuring the AWS Command Line Interface

2. Add a named profile for the administrator user in the AWS CLI config file. You use this profile when executing the AWS CLI commands. For more information about named profiles, see Named Profiles in the *AWS Command Line Interface User Guide*.

   ```
   [profile adminuser]
   aws_access_key_id = adminuser access key ID
   aws_secret_access_key = adminuser secret access key
   region = aws-region
   ```

   For a list of available AWS regions, see Regions and Endpoints in the *Amazon Web Services General Reference*.

3. Verify the setup by entering the following help command at the command prompt:

   ```
   aws help
   ```

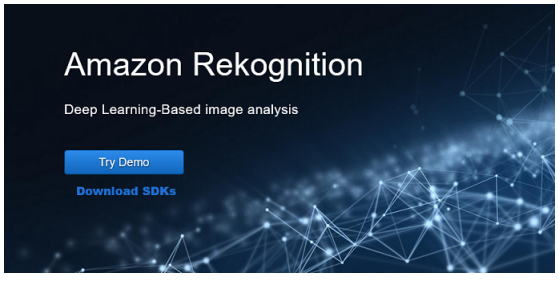
Next Step

Step 4: Getting Started Using API (p. 26)

Getting Started Using the Amazon Rekognition Console

This section shows you how to use a subset of Amazon Rekognition's capabilities such as object and scene detection, facial analysis, and face comparison in a set of images. For more information, see Amazon Rekognition: How It Works (p. 3). You can also use the Amazon Rekognition API or AWS CLI to detect objects and scenes, faces, and compare and search faces. For more information, see Step 4: Getting Started Using API (p. 26).

Topics



Exercise 1: Detect Objects and Scenes in an Image (Console)

This section shows how, at a very high level, Amazon Rekognition's objects and scenes detection capability works. When you specify an image as input, the service detects the objects and scenes in the image and returns them along with a percent confidence score for each object and scene.

For example, Amazon Rekognition detects the following objects and scenes in the sample image: skateboard, sport, person, auto, car and vehicle. To see all the confidence scores shown in this response, choose **Show more** in the **Labels | Confidence** pane.

Amazon Rekognition also returns a confidence score for each object detected in the sample image, as shown in the following sample response.

You can also look at the request to the API and the response from the API as a reference.

Request

```
{
   "method": "POST",
   "path": "/",
   "region": "us-east-1",
   "headers": {
      "Content-Type": "application/x-amz-json-1.1",
      "X-Amz-Date": "Fri, 18 Nov 2016 21:14:23 GMT",
```

```
      "X-Amz-Target":
 "com.amazonaws.rekognitionservice.RekognitionService.DetectLabels"
   },
   "contentString": {
      "Attributes": [
         "ALL"
      ],
      "Image": {
         "S3Object": {
            "Bucket": "console-assets",
            "Name": "images/skateboard.jpg"
         }
      }
   }
}
```

Response

```
{
   "Labels": [
      {
         "Confidence": 99.25342,
         "Name": "skateboard"
      },
      {
         "Confidence": 99.25342,
         "Name": "sport"
      },
      {
         "Confidence": 99.24723,
         "Name": "person"
      },
      {
         "Confidence": 91.53313,
         "Name": "auto"
      },
      {
         "Confidence": 91.53313,
         "Name": "car"
      },
      {
         "Confidence": 91.53313,
         "Name": "vehicle"
      },
      {
         "Confidence": 76.85095,
         "Name": "intersection"
      },
      {
         "Confidence": 76.85095,
         "Name": "road"
      },
      {
         "Confidence": 76.21494,
         "Name": "path"
      },
      {
         "Confidence": 66.715416,
```

```
            "Name": "building"
        },
        {
            "Confidence": 62.04722,
            "Name": "sports_car"
        },
        {
            "Confidence": 61.988914,
            "Name": "city"
        },
        {
            "Confidence": 61.988914,
            "Name": "urban"
        },
        {
            "Confidence": 60.978107,
            "Name": "town"
        },
        {
            "Confidence": 57.33275,
            "Name": "district"
        },
        {
            "Confidence": 56.48067,
            "Name": "street"
        },
        {
            "Confidence": 54.235493,
            "Name": "housing"
        },
        {
            "Confidence": 51.260765,
            "Name": "high_rise"
        },
        {
            "Confidence": 50.595474,
            "Name": "freeway"
        }
    ]
}
```


For more information, see Amazon Rekognition: How It Works (p. 3).

Detect Objects and Scenes in an Image You Provide

You can upload an image that you own or provide the URL to an image as input in the Amazon Rekognition console. Amazon Rekognition returns the object and scenes, confidence scores for each object, and scene it detects in the image you provide.

> **Note**
> The image must be less than 5MB in size and must be of JPEG or PNG format.

To detect objects and scenes in an image you provide

1. Open the Amazon Rekognition console.
2. Choose **Object and scene detection**.
3. Do one of the following:

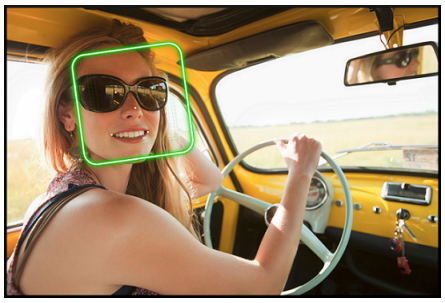
- Upload an image – Choose **Upload**, go to the location where you stored your image, and then select the image.
- Use a URL – Type the URL in the text box, and then choose **Go**.

4. View the confidence score of each label detected in the **Labels | Confidence** pane.

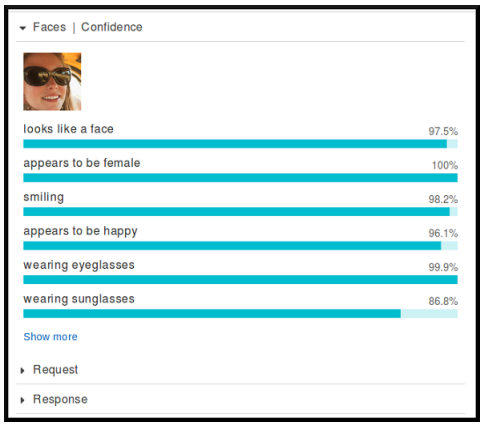
Exercise 2: Analyze Faces in an Image (Console)

This section shows you how to use the Amazon Rekognition console to detect faces and analyze facial attributes in an image. When you provide an image that contains a face as input, the service detects the face in the image, analyzes the facial attributes of the face, and then returns a percent confidence score for the face and the facial attributes detected in the image. For more information, see Amazon Rekognition: How It Works (p. 3).

For example, if you choose the following sample image as input, Amazon Rekognition detects it as a face and returns confidence scores for the face and the facial attributes detected.



The following shows the sample response.



If there are multiple faces in the input image, Rekognition detects up to 15 faces in the image. Each face detected is marked with a square. When you click the area marked with a square on a face, Rekognition displays the confidence score of that face and its attributes detected in the **Faces | Confidence** pane.

Analyze Faces in an Image You Provide

You can upload your own image or provide the URL to the image in the Amazon Rekognition console.

> **Note**
> The image must be less than 5MB in size and must be of JPEG or PNG format.

To analyze a face in an image you provide

1. Open to the Amazon Rekognition console.
2. Choose **Facial analysis**.
3. Do one of the following:
   - Upload an image – Choose **Upload**, go to the location where you stored your image, and then select the image.
   - Use a URL – Type the URL in the text box, and then choose **Go**.
4. View the confidence score of one the faces detected and its facial attributes in the **Faces | Confidence** pane.
5. If there are multiple faces in the image, choose one of the other faces to see its attributes and scores.

Exercise 3: Compare Faces in Images (Console)

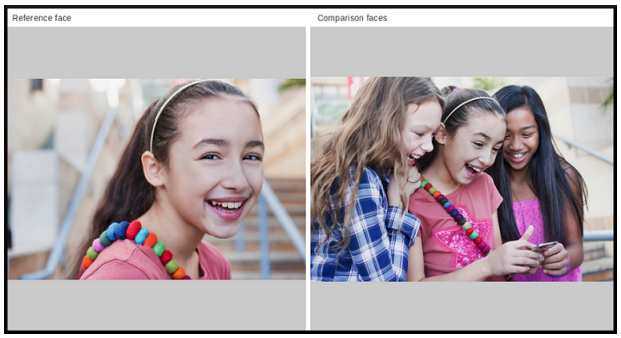
This section shows you how to use the Amazon Rekognition console to compare faces within a set of images with multiple faces in them. When you specify a **Reference face** (source) and a **Comparison faces** (target) image, Rekognition compares the largest face in the source image (that is, the reference face) with up to 15 faces detected in the target image (that is, the comparison faces), and then finds how closely the face in the source matches the faces in the target image. The similarity score for each comparison is displayed in the **Results** pane.

If the target image contains multiple faces, Rekognition matches the face in the source image with up to 15 faces detected in target image, and then assigns a similarity score to each match.

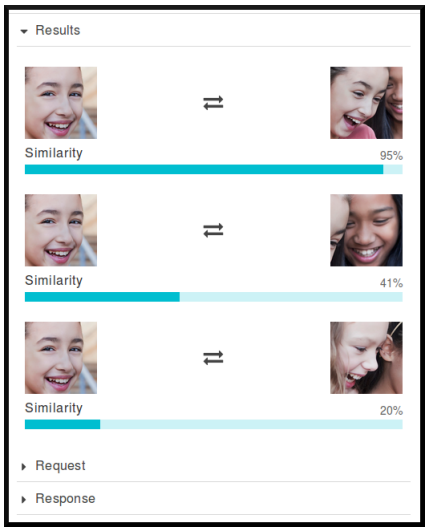
If the source image contains multiple faces, the service detects the largest face in the source image and uses it to compare with each face detected in the target image.

For more information, see Comparing Faces (p. 11).

For example, with the sample image shown on the left as a source image and the sample image on the right as a target image, Rekognition compares the face in the source image, matches it with each face in the target image, displays a similarity score for each face it detects.



The following shows the faces detected in the target image and the similarity score for each face.



Compare Faces in an Image You Provide

You can upload your own source and target images for Rekognition to compare the faces in the images or you can specify a URL for the location of the images.

> **Note**
> The image must be less than 5MB in size and must be of JPEG or PNG format.

To compare faces in your images

1. Open to the Amazon Rekognition console.
2. Choose **Face comparison**.
3. For your source image, do one of the following:
   - Upload an image – Choose **Upload** on the left, go to the location where you stored your source image, and then select the image.
   - Use a URL – Type the URL of your source image in the text box, and then choose **Go**.
4. For your target image, do one of the following:
   - Upload an image – Choose **Upload** on the right, go to the location where you stored your source image, and then select the image.
   - Use a URL – Type the URL of your source image in the text box, and then choose **Go**.
5. Rekognition matches the largest face in your source image with up to 15 faces in the target image and then displays the similarity score for each pair in the **Results** pane.

Step 4: Getting Started Using API

In this section you use the Amazon Rekognition API operations to detect labels and faces in an image. You also explore the compare faces API. These are the non-storage API operations where Amazon Rekognition doesn't persist the input images or any image data. The service only detects labels and faces and returns information in response. For more information, see Amazon Rekognition: How It Works (p. 3).

You need sample images (JPEG or PNG) that you can provide as input to these operations.

> **Note**
> These examples use the `us-east-1` endpoint.

Topics

Exercise 1: Detect Labels in an Image (API)

In this exercise you use the DetectLabels (p. 65) API to detect objects, concepts, and scenes in an image (JPEG or PNG) that you provide as input. You can provide the input image as a image byte array (Base64-encoded image bytes) or specify an S3 object. In this exercise you upload a JPEG image to your Amazon S3 bucket and specify the object key name.

You can test the operation using the AWS CLI or programmatically using the AWS SDK for Java.

1. Upload an image (containing one or more objects, such as trees, houses, and boat etc.) to your S3 bucket. The exercise assumes a .jpg image. If you use .png, update the code accordingly.

   For instructions, see Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service Console User Guide*.
2. Use either the Java example code or the AWS CLI to test the `DetectLabels` operation.

   - Using the AWS CLI

     Note
     The command specifies the `adminuser` profile that you set up in Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 19). Then, the AWS CLI command uses the credentials associated with the adminuser profile to sign and authenticate the request. If you don't provide this profile, the default profile is assumed.

```
aws rekognition detect-labels \
--image '{"S3Object":{"Bucket":"bucketname","Name":"object.jpg"}}' \
--region us-east-1 \
--profile adminuser
```

   - Using the AWS SDK for Java.

```
import com.amazonaws.services.rekognition.AmazonRekognitionClient;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import
 com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.DetectLabelsRequest;
import com.amazonaws.services.rekognition.model.DetectLabelsResult;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.S3Object;
import com.fasterxml.jackson.databind.ObjectMapper;

public class DetectLabelsExample {
    public static void main(String[] args) throws Exception {

        AWSCredentials credentials;
        try {
```

```
            credentials = new
 ProfileCredentialsProvider("adminuser").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load the credentials
 from the credential profiles file. "
                    + "Please make sure that your credentials file is at
 the correct "
                    + "location (/Users/<userid>/.aws/credentials), and
 is in a valid format.", e);
        }

        DetectLabelsRequest request = new DetectLabelsRequest()
                .withImage(new Image()
                        .withS3Object(new S3Object()
                                .withName("s3objectkey")
                                .withBucket("bucket-name")))
                .withMaxLabels(10)
                .withMinConfidence(77F);

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient(credentials)
                .withEndpoint("service endpoint");
        rekognitionClient.setSignerRegionOverride("us-east-1");
        try {
            DetectLabelsResult result =
 rekognitionClient.detectLabels(request);
            ObjectMapper objectMapper = new ObjectMapper();
            System.out.println("Result = " +
 objectMapper.writeValueAsString(result));
        } catch (AmazonRekognitionException e) {
            e.printStackTrace();
        }

    }
}
```

You should get up to 10 labels with at least 75F confidence.

Next Exercise

Exercise 2: Detect Faces (API) (p. 28)

Exercise 2: Detect Faces (API)

In this step, you use the DetectFaces (p. 61) operation to detect faces in an image (JPEG or PNG) that you provide as input. You can provide the input image as an image byte array (Base64-encoded image bytes) or specify an S3 object. In this exercise, you upload an image (JPEG or PNG) to your S3 bucket and specify the object key name.

You can test the operation using the AWS CLI or programmatically using the AWS SDK for Java.

For more information, see Detecting Faces (p. 6).

1. Upload an image (containing one or more faces) to your S3 bucket.

For instructions, see Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service Console User Guide*.

2. Either use the Java example code or the AWS CLI to test the `DetectFaces` operation.

   - Using he AWS CLI

```
aws rekognition detect-faces \
--image '{"S3Object":{"Bucket":"Bucketname","Name":"s3ObjectKey"}}' \
--attributes "ALL" \
--region us-east-1 \
--profile adminuser
```

   - Using the AWS SDK for Java

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.rekognition.AmazonRekognitionClient;
import
 com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.Attribute;
import com.amazonaws.services.rekognition.model.DetectFacesRequest;
import com.amazonaws.services.rekognition.model.DetectFacesResult;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.S3Object;
import com.fasterxml.jackson.databind.ObjectMapper;

public class DetectFacesExample {
    public static void main(String[] args) throws Exception {

        AWSCredentials credentials;
        try {
            credentials = new
 ProfileCredentialsProvider("adminuser").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load the credentials
 from the credential profiles file. "
                    + "Please make sure that your credentials file is at
 the correct "
                    + "location (/Users/<userid>/.aws/credentials), and
 is in a valid format.", e);
        }

        DetectFacesRequest request = new DetectFacesRequest()
                .withImage(new Image()
                        .withS3Object(new S3Object()
                                .withName("s3ObjectKey")
                                .withBucket("bucketname")))
                .withAttributes(Attribute.ALL);

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient(credentials)
                .withEndpoint("service endpoint");
        rekognitionClient.setSignerRegionOverride("us-east-1");
        try {
            DetectFacesResult result =
 rekognitionClient.detectFaces(request);
```

```
            ObjectMapper objectMapper = new ObjectMapper();
            System.out.println("Result = " +
 objectMapper.writeValueAsString(result));
        } catch (AmazonRekognitionException e) {
            e.printStackTrace();
        }
    }
}
```

Next Exercise

Exercise 3: Compare Faces (API) (p. 30)

Exercise 3: Compare Faces (API)

In this step, you use the CompareFaces (p. 52) operation to compare a face in the *source* image with each face detected in the *target* image.

If you provide a source image containing multiple faces, the service detects the largest face and uses it to compare with each face detected in the target image.

In the response you get an array of face matches. For each matching face in the target image, the response provides information including a bounding box of the matching face and similarity score (how similar the face is to the source face).

You can provide the source and target images as a image byte array (Base64-encoded image bytes) or specify S3 objects. In this exercise, you upload two JPEG images to your Amazon S3 bucket and specify the object key name.

You can test the operation using the AWS CLI or programmatically using the AWS SDK for Java.

1. Upload two images (source.jpg and target.jpg) containing faces to your S3 bucket. The exercise assume a .jpg image. If you use .png, update the AWS CLI command accordingly.

   For instructions, see Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service Console User Guide*.
2. Either use the Java example code or the AWS CLI to test the `CompareFaces` operation.

   - Using AWS CLI

     ```
     aws rekognition compare-faces \
     --source-image '{"S3Object":{"Bucket":"bucket-
     name","Name":"source.jpg"}}' \
     --target-image '{"S3Object":{"Bucket":"bucket-
     name","Name":"target.jpg"}}' \
     --region us-east-1 \
     --profile adminuser
     ```

   - Using the AWS SDK for Java

     You need to update the code by providing your bucket name.

     ```
     import com.amazonaws.AmazonClientException;
     import com.amazonaws.auth.AWSCredentials;
     import com.amazonaws.auth.profile.ProfileCredentialsProvider;
     import com.amazonaws.services.rekognition.AmazonRekognition;
     ```

```
import com.amazonaws.services.rekognition.AmazonRekognitionClient;
import
 com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.CompareFacesRequest;
import com.amazonaws.services.rekognition.model.CompareFacesResult;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.S3Object;
import com.fasterxml.jackson.databind.ObjectMapper;

public class CompareFacesExample {
    public static final String S3_BUCKET = "bucket-name";
    public static void main(String[] args) throws Exception {

        AWSCredentials credentials;
        try {
            credentials = new
 ProfileCredentialsProvider("adminuser").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load the credentials
 from the credential profiles file. "
                    + "Please make sure that your credentials file is at
 the correct "
                    + "location (/Users/<userid>/.aws/credentials), and
 is in valid format.", e);
        }

        AmazonRekognitionClient rekognitionClient = new
 AmazonRekognitionClient(credentials)
                .withEndpoint("service-endpoint");
        rekognitionClient.setSignerRegionOverride("us-east-1");
        ObjectMapper objectMapper = new ObjectMapper();

        Image source = getImageUtil(S3_BUCKET, "source-Image.jpg");
        Image target = getImageUtil(S3_BUCKET, "target-Image.jpg");
        Float similarityThreshold = 70F;
        CompareFacesResult compareFacesResult = callCompareFaces(source,
 target, similarityThreshold, rekognitionClient);


 System.out.println(objectMapper.writeValueAsString(compareFacesResult));
    }

    private static CompareFacesResult callCompareFaces(Image
 sourceImage, Image targetImage,
                                                       Float
 similarityThreshold, AmazonRekognition amazonRekognition) {
        CompareFacesRequest compareFacesRequest = new
 CompareFacesRequest()
                .withSourceImage(sourceImage)
                .withTargetImage(targetImage)
                .withSimilarityThreshold(similarityThreshold);
        return amazonRekognition.compareFaces(compareFacesRequest);
    }

    private static Image getImageUtil(String bucket, String key) {
        return new Image()
                .withS3Object(new S3Object()
                        .withBucket(bucket)
                        .withName(key));
```

```
    }
}
```

What's Next?

You can explore additional Additional Amazon Rekognition Examples (p. 34) of how to use other Amazon Rekognition API operations (storage-based API operations) that describe how to create a face collection, add faces to the collection, and search the collection for face matches.

Limits in Amazon Rekognition

The following is a list of limits in Amazon Rekognition:

- Maximum image size stored as an Amazon S3 object is limited to 15 MB. The minimum pixel resolution for height and width is 80 pixels.
- Maximum images size as raw bytes passed in as parameter to an API is 5 MB.
- Amazon Rekognition supports the PNG and JPEG image formats. That is, the images you provide as input to various API operations, such as `DetectLabels` and `IndexFaces` must be in one of the supported formats.
- Maximum number of faces you can store in a single face collection is 1 million.
- The maximum matching faces the search API returns is 4096.

Additional Amazon Rekognition Examples

This section provides additional examples of working with Amazon Rekognition. Examples using AWS SDK for Java and the AWS CLI are provided. We recommend that you first review the following topics:

- Amazon Rekognition: How It Works (p. 3)
- Getting Started with Amazon Rekognition (p. 18)

Topics

Example 1: Managing Collections

This section provides working examples of creating, listing, and deleting collections. Examples using both the AWS CLI and the AWS SDK for Java are provided.

For information about managing collections and related API operations, see Storage-Based API Operations: Storing Faces and Searching Face Matches (p. 12).

Topics

Creating, Listing, and Deleting Collections: Using the AWS CLI

The following are example AWS CLI commands that you can use to create and delete collections. An example AWS CLI command that lists collections is also provided.

- **Create a face collection** – The following `create-collection` AWS CLI command creates a face collection (`examplecollection`) in the us-east-1 region.

  Note
  The command specifies the `adminuser` profile that you set up in Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 19). The AWS CLI command uses the credentials associated with the `adminuser` profile to sign and authenticate the request. If you don't provide this profile, the default profile is assumed.

```
aws rekognition create-collection \
--collection-id "examplecollection" \
--region us-east-1 \
--profile adminuser
```

  Amazon Rekognition creates the collection in the specified region, and returns the Amazon Resource Name (ARN) of the newly created collection. An example response is shown following:

```
{
    "CollectionArn": "aws:rekognition:us-east-1:acct-id:collection/
examplecollection",
    "StatusCode": 200
}
```

- **List Collections** – The following `list-collections` AWS CLI command returns a list of collections in the us-east-1 region.

```
aws rekognition list-collections \
--region us-east-1 \
--profile adminuser
```

  The following is an example response:

```
{
    "CollectionIds": [
        "examplecollection1",
        "examplecollection2",
        "examplecollection3"
    ]
}
```

- **Delete a face collection** – The following `delete-collection` AWS CLI command deletes a face collection.

```
aws rekognition delete-collection \
--collection-id "examplecollection" \
--region us-east-1 \
--profile adminuser
```

Creating, Listing, and Deleting Face Collections: Using the AWS SDK for Java

The following Java example code uses the AWS SDK for Java to create and delete a collection (`examplecollection`) in the `us-east-1` region. The code example also lists collections in the region.

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClient;
import com.amazonaws.services.rekognition.model.CreateCollectionRequest;
import com.amazonaws.services.rekognition.model.CreateCollectionResult;
import com.amazonaws.services.rekognition.model.DeleteCollectionRequest;
import com.amazonaws.services.rekognition.model.DeleteCollectionResult;
import com.amazonaws.services.rekognition.model.ListCollectionsRequest;
import com.amazonaws.services.rekognition.model.ListCollectionsResult;
import com.fasterxml.jackson.databind.ObjectMapper;

public class CollectionExample {
    public static void main(String[] args) throws Exception {
        AWSCredentials credentials;
        try {
            credentials = new
 ProfileCredentialsProvider("adminuser").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load the credentials from
 the credential profiles file. "
                    + "Please make sure that your credentials file is at the
 correct "
                    + "location (/Users/<userid>/.aws/credentials), and is in
 valid format.", e);
        }

        AmazonRekognitionClient amazonRekognition = new
 AmazonRekognitionClient(credentials)
                .withEndpoint("service-endpoint");
        amazonRekognition.setSignerRegionOverride("us-east-1");
        ObjectMapper objectMapper = new ObjectMapper();

        // 1. CreateCollection 1
        String collectionId = "exampleCollection";
        CreateCollectionResult createCollectionResult =
 callCreateCollection(collectionId, amazonRekognition);

 System.out.println(objectMapper.writeValueAsString(createCollectionResult));

        // 2. CreateCollection 2
        callCreateCollection("exampleCollection2", amazonRekognition);

        // 3. Page through collections with ListCollections
        int limit = 1;
        ListCollectionsResult listCollectionsResult = null;
```

```
        String paginationToken = null;
        do {
            if (listCollectionsResult != null) {
                paginationToken = listCollectionsResult.getNextToken();
            }
            listCollectionsResult = callListCollections(paginationToken,
 limit, amazonRekognition);

 System.out.println(objectMapper.writeValueAsString(listCollectionsResult));
        } while (listCollectionsResult != null &&
 listCollectionsResult.getNextToken() != null);

        // 4. Clean up collection 1 with DeleteCollection
        DeleteCollectionResult deleteCollectionResult =
 callDeleteCollection(collectionId, amazonRekognition);

 System.out.println(objectMapper.writeValueAsString(deleteCollectionResult));
    }

    private static CreateCollectionResult callCreateCollection(String
 collectionId, AmazonRekognition amazonRekognition) {
        CreateCollectionRequest request = new CreateCollectionRequest()
                .withCollectionId(collectionId);
        return amazonRekognition.createCollection(request);
    }

    private static DeleteCollectionResult callDeleteCollection(String
 collectionId, AmazonRekognition amazonRekognition) {
        DeleteCollectionRequest request = new DeleteCollectionRequest()
                .withCollectionId(collectionId);
        return amazonRekognition.deleteCollection(request);
    }

    private static ListCollectionsResult callListCollections(String
 paginationToken, int limit, AmazonRekognition amazonRekognition) {
        ListCollectionsRequest listCollectionsRequest = new
 ListCollectionsRequest()
                .withMaxResults(limit)
                .withNextToken(paginationToken);
        return amazonRekognition.listCollections(listCollectionsRequest);
    }
}
```

Example 2: Storing Faces

This section provides working examples of storing faces in a collection. Examples using both the AWS CLI and the AWS SDK for Java are provided.

For information about the collections and storing faces API operations, see Storage-Based API Operations: Storing Faces and Searching Face Matches (p. 12).

Topics

Storing Faces: Using the AWS CLI

The following `index-faces` AWS CLI command detects faces in the input images, and for each face extracts facial features and store the feature information in a database. In addition, the command stores metadata for each face detected in the specified face collection.

```
aws rekognition index-faces \
--image '{"S3Object":{"Bucket":"bucket","Name":"S3ObjectKey"}}' \
--collection-id "collection-id" \
--region us-east-1 \
--profile adminuser
```

For, more information, see Storing Faces In a Face Collection: The IndexFaces Operation (p. 13)

In the following example response, note the following:

- Information in the `faceDetail` element is not persisted on the server. It is only returned as part of this response. The `faceDetail` includes five facial landmarks (see `landmark` element), `pose`, and `quality`.
- Information in the `face` element is the face metadata that is persisted on the server. This is the same information the ListFaces (p. 74) API returns in response.


```
{
    "FaceRecords": [
        {
            "FaceDetail": {
                "BoundingBox": {
                    "Width": 0.6154,
                    "Top": 0.2442,
                    "Left": 0.1765,
                    "Height": 0.4692
                },
                "Landmarks": [
                    {
                        "Y": 0.41730427742004395,
                        "X": 0.36835095286369324,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.4281611740589142,
                        "X": 0.5960656404495239,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.5349795818328857,
                        "X": 0.47817257046699524,
                        "Type": "nose"
                    },
                    {
                        "Y": 0.5721957683563232,
                        "X": 0.352621465921402,
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.5792245864868164,
                        "X": 0.5936088562011719,
```

```
                        "Type": "mouthRight"
                    }
                ],
                "Pose": {
                    "Yaw": 1.8526556491851807,
                    "Roll": 3.623055934906006,
                    "Pitch": -10.605680465698242
                },
                "Quality": {
                    "Sharpness": 130.0,
                    "Brightness": 49.129302978515625
                },
                "Confidence": 99.99968719482422
            },
            "Face": {
                "BoundingBox": {
                    "Width": 0.6154,
                    "Top": 0.2442,
                    "Left": 0.1765,
                    "Height": 0.4692
                },
                "FaceId": "84de1c86-5059-53f2-a432-34ebb704615d",
                "Confidence": 99.9997,
                "ImageId": "d38ebf91-1a11-58fc-ba42-f978b3f32f60"
            }
        }
    ],
    "OrientationCorrection": "ROTATE_0"
}
```

The following `index-faces` command specifies two optional parameters:

- `--detection-attribute` parameter to request all facial attributes in the response.
- `--external-image-id` parameter to specify an ID to be associated with all faces in this image. You might use this information on the client side, for example, you might maintain a client-side index of images and faces in the images.

```
aws rekognition index-faces \
--image '{"S3Object":{"Bucket":"bucketname","Name":"object-key"}}' \
--collection-id "collection-id" \
--detection-attributes "ALL" \
--external-image-id "example-image.jpg" \
--region us-east-1 \
--profile adminuser
```

In the following example response, note the additional information in the `faceDetail` element, which is not persisted on the server:

- 25 facial landmarks (compared to only five in the preceding example)
- Nine facial attributes (eyeglasses, beard, etc)
- Emotions (see the `emotion` element)

The `face` element provides metadata that is persisted on the server.

```
{
```

```
    "FaceRecords": [
        {
            "FaceDetail": {
                "Confidence": 99.99968719482422,
                "Eyeglasses": {
                    "Confidence": 99.94019317626953,
                    "Value": false
                },
                "Sunglasses": {
                    "Confidence": 99.62261199951172,
                    "Value": false
                },
                "Gender": {
                    "Confidence": 99.92701721191406,
                    "Value": "Male"
                },
                "Pose": {
                    "Yaw": 1.8526556491851807,
                    "Roll": 3.623055934906006,
                    "Pitch": -10.605680465698242
                },
                "Emotions": [
                    {
                        "Confidence": 99.38518524169922,
                        "Type": "HAPPY"
                    },
                    {
                        "Confidence": 1.1799871921539307,
                        "Type": "ANGRY"
                    },
                    {
                        "Confidence": 1.0325908660888672,
                        "Type": "CONFUSED"
                    }
                ],
                "EyesOpen": {
                    "Confidence": 54.15227508544922,
                    "Value": false
                },
                "Quality": {
                    "Sharpness": 130.0,
                    "Brightness": 49.129302978515625
                },
                "BoundingBox": {
                    "Width": 0.6153846383094788,
                    "Top": 0.24423076212406158,
                    "Left": 0.17654477059841156,
                    "Height": 0.4692307710647583
                },
                "Smile": {
                    "Confidence": 99.8236083984375,
                    "Value": true
                },
                "MouthOpen": {
                    "Confidence": 88.39942169189453,
                    "Value": true
                },
                "Landmarks": [
                    {
```

```
            "Y": 0.41730427742004395,
            "X": 0.36835095286369324,
            "Type": "eyeLeft"
        },
        {
            "Y": 0.4281611740589142,
            "X": 0.5960656404495239,
            "Type": "eyeRight"
        },
        {
            "Y": 0.5349795818328857,
            "X": 0.47817257046699524,
            "Type": "nose"
        },
        {
            "Y": 0.5721957683563232,
            "X": 0.352621465921402,
            "Type": "mouthLeft"
        },
        {
            "Y": 0.5792245864868164,
            "X": 0.5936088562011719,
            "Type": "mouthRight"
        },
        {
            "Y": 0.4163532555103302,
            "X": 0.3697868585586548,
            "Type": "leftPupil"
        },
        {
            "Y": 0.42626339197158813,
            "X": 0.6037314534187317,
            "Type": "rightPupil"
        },
        {
            "Y": 0.38954615592956543,
            "X": 0.27343833446502686,
            "Type": "leftEyeBrowLeft"
        },
        {
            "Y": 0.3775958716869354,
            "X": 0.35098740458488464,
            "Type": "leftEyeBrowRight"
        },
        {
            "Y": 0.39108505845069885,
            "X": 0.433648943901062,
            "Type": "leftEyeBrowUp"
        },
        {
            "Y": 0.3952394127845764,
            "X": 0.5416828989982605,
            "Type": "rightEyeBrowLeft"
        },
        {
            "Y": 0.38667190074920654,
            "X": 0.6171167492866516,
            "Type": "rightEyeBrowRight"
        },
```

```
            {
                "Y": 0.40419116616249084,
                "X": 0.6827319264411926,
                "Type": "rightEyeBrowUp"
            },
            {
                "Y": 0.41925403475761414,
                "X": 0.32195475697517395,
                "Type": "leftEyeLeft"
            },
            {
                "Y": 0.4225293695926666,
                "X": 0.41227561235427856,
                "Type": "leftEyeRight"
            },
            {
                "Y": 0.4096950888633728,
                "X": 0.3705553412437439,
                "Type": "leftEyeUp"
            },
            {
                "Y": 0.4213259816169739,
                "X": 0.36738231778144836,
                "Type": "leftEyeDown"
            },
            {
                "Y": 0.4294262230396271,
                "X": 0.5498995184898376,
                "Type": "rightEyeLeft"
            },
            {
                "Y": 0.4327501356601715,
                "X": 0.6390777826309204,
                "Type": "rightEyeRight"
            },
            {
                "Y": 0.42076829075813293,
                "X": 0.5977370738983154,
                "Type": "rightEyeUp"
            },
            {
                "Y": 0.4326271116733551,
                "X": 0.5959710478782654,
                "Type": "rightEyeDown"
            },
            {
                "Y": 0.5411174893379211,
                "X": 0.4253743588924408,
                "Type": "noseLeft"
            },
            {
                "Y": 0.5450678467750549,
                "X": 0.5309309959411621,
                "Type": "noseRight"
            },
            {
                "Y": 0.5795656442642212,
                "X": 0.47389525175094604,
                "Type": "mouthUp"
```

```
                    },
                    {
                        "Y": 0.6466911435127258,
                        "X": 0.47393468022346497,
                        "Type": "mouthDown"
                    }
                ],
                "Mustache": {
                    "Confidence": 99.75302124023438,
                    "Value": false
                },
                "Beard": {
                    "Confidence": 89.82911682128906,
                    "Value": false
                }
            },
            "Face": {
                "BoundingBox": {
                    "Width": 0.6153846383094788,
                    "Top": 0.24423076212406158,
                    "Left": 0.17654477059841156,
                    "Height": 0.4692307710647583
                },
                "FaceId": "407b95a5-f8f7-50c7-bf86-27c9ba5c6931",
                "ExternalImageId": "example-image.jpg",
                "Confidence": 99.99968719482422,
                "ImageId": "af554b0d-fcb2-56e8-9658-69aec6c901be"
            }
        }
    ],
    "OrientationCorrection": "ROTATE_0"
}
```

You can use the list-faces command to get a list of faces in a collection:

```
aws rekognition list-faces \
--collection-id "collection-id" \
--region us-east-1
--profile adminuser
```

The command returns faces in the collection along with a `NextToken` in the response. You can use this in your subsequent request (by adding the `--next-token` parameter in the AWS CLI command) to fetch next set of faces.

Storing Faces: Using the AWS SDK for Java

The following example AWS SDK for Java code stores two faces to a collection in the us-east-1 region. You need to update the code by providing an S3 bucket name, two object keys (.jpg objects), and an Amazon Rekognition face collection name.

```
import java.util.List;
import java.util.stream.Collectors;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.rekognition.AmazonRekognition;
```

```
import com.amazonaws.services.rekognition.AmazonRekognitionClient;
import com.amazonaws.services.rekognition.model.DeleteFacesRequest;
import com.amazonaws.services.rekognition.model.DeleteFacesResult;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.IndexFacesRequest;
import com.amazonaws.services.rekognition.model.IndexFacesResult;
import com.amazonaws.services.rekognition.model.ListFacesRequest;
import com.amazonaws.services.rekognition.model.ListFacesResult;
import com.amazonaws.services.rekognition.model.S3Object;
import com.fasterxml.jackson.databind.ObjectMapper;

public class IndexAndListFacesExample {
    public static final String COLLECTION_ID = "collection-id";
    public static final String S3_BUCKET = "bucket";

    public static void main(String[] args) throws Exception {
        AWSCredentials credentials;
        try {
            credentials = new
 ProfileCredentialsProvider("adminuser").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load the credentials from
 the credential profiles file. "
                    + "Please make sure that your credentials file is at the
 correct "
                    + "location (/Users/<userid>/.aws/credentials), and is in
 valid format.", e);
        }

        AmazonRekognitionClient amazonRekognition = new
 AmazonRekognitionClient(credentials)
                .withEndpoint("service-endpoint");
        amazonRekognition.setSignerRegionOverride("us-east-1");
        ObjectMapper objectMapper = new ObjectMapper();

        // 1. Index face 1
        Image image = getImageUtil(S3_BUCKET, "image1.jpeg");
        String externalImageId = "external.jpg";
        IndexFacesResult indexFacesResult = callIndexFaces(COLLECTION_ID,
 externalImageId, "ALL", image, amazonRekognition);

 System.out.println(objectMapper.writeValueAsString(indexFacesResult));

        // 2. Index face 2
        Image image2 = getImageUtil(S3_BUCKET, "image2.jpeg");
        String externalImageId2 = "external2.jpg";
        callIndexFaces(COLLECTION_ID, externalImageId2, "ALL", image2,
 amazonRekognition);

        // 3. Page through the faces with ListFaces
        ListFacesResult listFacesResult = null;
        String paginationToken = null;
        do {
            if (listFacesResult != null) {
                paginationToken = listFacesResult.getNextToken();
            }
            listFacesResult = callListFaces(COLLECTION_ID, 1,
 paginationToken, amazonRekognition);
```

```
System.out.println(objectMapper.writeValueAsString(listFacesResult));

       } while(listFacesResult != null && listFacesResult.getNextToken() !=
null);
   }

   private static IndexFacesResult callIndexFaces(String collectionId,
String externalImageId, String attributes, Image image, AmazonRekognition
amazonRekognition) {
       IndexFacesRequest indexFacesRequest = new IndexFacesRequest()
               .withImage(image)
               .withCollectionId(collectionId)
               .withExternalImageId(externalImageId)
               .withDetectionAttributes(attributes);
       return amazonRekognition.indexFaces(indexFacesRequest);

   }

   private static ListFacesResult callListFaces(String collectionId, int
limit, String paginationToken, AmazonRekognition amazonRekognition) {
       ListFacesRequest listFacesRequest = new ListFacesRequest()
               .withCollectionId(collectionId)
               .withMaxResults(limit)
               .withNextToken(paginationToken);
       return amazonRekognition.listFaces(listFacesRequest);
   }

   private static Image getImageUtil(String bucket, String key) {
       return new Image()
               .withS3Object(new S3Object()
                       .withBucket(bucket)
                       .withName(key));
   }
}
```

Example 3: Searching Faces

This section provides working examples of API operations that you can use to search a face collection for face matches. Examples using both AWS CLI and AWS SDK for Java are provided.

For information about collections and search faces API operations, see Storage-Based API Operations: Storing Faces and Searching Face Matches (p. 12).

Topics

Searching Faces: Using the AWS CLI

You can search a face collection for face matches using the `search-faces` (see SearchFaces (p. 76)) and `search-faces-by-image` (see SearchFacesByImage (p. 79)) commands:

- **Search faces by face ID** – You can use the `search-faces` command to search a face collection for face matches by providing a face ID (that is, one of the face IDs that exists in the face collection). Then, the command searches the collection for similar faces.

  For this exercise, if you don't know a face ID value, you can use the `list-faces` command:

```
aws rekognition list-faces \
--collection-id "collection-id" \
--region us-east-1 \
--profile adminuser
```

  Specify the `search-faces` command, as shown following:

```
aws rekognition search-faces \
--face-id face-id \
--collection-id "collection-id" \
--region us-east-1 \
--profile adminuser
```

  The following is the example response that includes the search face ID you provided as input and three face matches. For more information about the response, see Searching Faces In a Face Collection (p. 15).

```
{
    "SearchedFaceId": "e0182208-f475-55b4-8d88-cf162509718d",
    "FaceMatches": [
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.49505001306533813,
                    "Top": 0.221110999584198,
                    "Left": 0.3069309890270233,
                    "Height": 0.33333298563957214
                },
                "FaceId": "9b01ac35-61be-55b0-bc95-54b6421e4950",
                "ExternalImageId": "example-image.jpg",
                "Confidence": 99.99949645996094,
                "ImageId": "fba488d7-9c3a-537f-a30a-b8a1ee326b6c"
            },
            "Similarity": 0.9172449111938477
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.2044440060853958,
                    "Top": 0.22542400658130646,
                    "Left": 0.46222200989723206,
                    "Height": 0.3118639886379242
                },
                "FaceId": "98fd3f10-a078-5b35-83c5-5d5c8423a8fc",
                "ExternalImageId": "example-image.jpg",
                "Confidence": 99.99810028076172,
                "ImageId": "b5d3f633-1b8c-560a-adfb-08891b6536a0"
            },
            "Similarity": 0.9123537540435791
        },
        {
```

```
            "Face": {
                "BoundingBox": {
                    "Width": 0.6153849959373474,
                    "Top": 0.24423100054264069,
                    "Left": 0.17654499411582947,
                    "Height": 0.4692310094833374
                },
                "FaceId": "407b95a5-f8f7-50c7-bf86-27c9ba5c6931",
                "ExternalImageId": "example-image.jpg",
                "Confidence": 99.99970245361328,
                "ImageId": "af554b0d-fcb2-56e8-9658-69aec6c901be"
            },
            "Similarity": 0.6758826971054077
        }
    ]
}
```

- **Search faces by providing an image as input** – In this case, Amazon Rekognition first detects the face in the input image, and then searches the collection for matching faces. The following `search-faces-by-image` command specifies an S3 object as input image.

```
aws rekognition search-faces-by-image \
--image '{"S3Object":{"Bucket":"bucket-name","Name":"Example.jpg"}}' \
--collection-id "collection-id" \
--region us-east-1 \
--profile adminuser
```

The following is an example response that includes the bounding box of the face in the input image, and a list of face matches. For more information about the response, see Searching Faces In a Face Collection (p. 15).

```
{
    "SearchedFaceBoundingBox": {
        "Width": 0.10111111402511597,
        "Top": 0.32203391194343567,
        "Left": 0.23999999463558197,
        "Height": 0.1542372852563858
    },
    "SearchedFaceConfidence": 98.51010131835938,
    "FaceMatches": [
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.10111100226640701,
                    "Top": 0.32203400135040283,
                    "Left": 0.23999999463558197,
                    "Height": 0.15423700213432312
                },
                "FaceId": "e0182208-f475-55b4-8d88-cf162509718d",
                "ExternalImageId": "example-image.jpg",
                "Confidence": 98.51010131835938,
                "ImageId": "b5d3f633-1b8c-560a-adfb-08891b6536a0"
            },
            "Similarity": 99.9808578491211
        }
    ]
```

```
}
```

Searching Faces: Using the AWS SDK for Java

The following AWS SDK for Java code example stores three faces to an Amazon Rekognition face collection in the us-east-1 region. Then, it searches the face collection for face matches. It shows usage of both `SearchFaces` and `SearchFacesByImage` API operations. The code example specifies both the `FaceMatchThreshold` and `MaxFaces` parameters to limit the results returned in the response.

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClient;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.IndexFacesRequest;
import com.amazonaws.services.rekognition.model.IndexFacesResult;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.rekognition.model.SearchFacesByImageRequest;
import com.amazonaws.services.rekognition.model.SearchFacesByImageResult;
import com.amazonaws.services.rekognition.model.SearchFacesRequest;
import com.amazonaws.services.rekognition.model.SearchFacesResult;
import com.fasterxml.jackson.databind.ObjectMapper;

public class SearchFacesExample {

    public static final String COLLECTION_ID = "collection-id";
    public static final String S3_BUCKET = "bucket-name";

    public static void main(String[] args) throws Exception {
        AWSCredentials credentials;
        try {
            credentials = new
 ProfileCredentialsProvider("adminuser").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load the credentials from
 the credential profiles file. "
                    + "Please make sure that your credentials file is at the
 correct "
                    + "location (/Users/<userid>/.aws/credentials), and is in
 valid format.", e);
        }

        AmazonRekognitionClient amazonRekognition = new
 AmazonRekognitionClient(credentials)
                .withEndpoint("service-endpoint");
        amazonRekognition.setSignerRegionOverride("us-east-1");
        ObjectMapper objectMapper = new ObjectMapper();

        //1. Add three faces to the collection.
        IndexFacesResult indexFacesResult = callIndexFaces(COLLECTION_ID,
 amazonRekognition, "image1.jpg");
        callIndexFaces(COLLECTION_ID, amazonRekognition, "image2.jpg");
        callIndexFaces(COLLECTION_ID, amazonRekognition, "image3.jpg");
```

```
        Float threshold = 70F;
        int maxFaces = 2;

        //2. Retrieve face ID of the 1st face added.
        String faceId = indexFacesResult.getFaceRecords().stream()
                .map(f -> f.getFace().getFaceId())
                .findAny().orElseThrow(() -> new IllegalArgumentException("No
face found"));

        //3. Search similar faces for a give face (identified by face ID).
        SearchFacesResult searchFacesResult = callSearchFaces(COLLECTION_ID,
faceId, threshold, maxFaces, amazonRekognition);

System.out.println(objectMapper.writeValueAsString(searchFacesResult));

        //4. Get an image object in S3 bucket.
        Image image = getImageUtil(S3_BUCKET, "imagex.jpg");

        //5. Search collection for faces similar to the largest face in the
image.
        SearchFacesByImageResult searchFacesByImageResult =
callSearchFacesByImage(COLLECTION_ID, image, threshold, maxFaces,
amazonRekognition);

System.out.println(objectMapper.writeValueAsString(searchFacesByImageResult));
    }

    private static IndexFacesResult callIndexFaces(
            String collectionId, AmazonRekognitionClient amazonRekognition,
String name) {
        IndexFacesRequest req = new IndexFacesRequest()
                .withImage(getImageUtil(S3_BUCKET, name))
                .withCollectionId(collectionId)
                .withExternalImageId("externalId");

        return amazonRekognition.indexFaces(req);
    }

    private static SearchFacesResult callSearchFaces(String collectionId,
String faceId, Float threshold, int maxFaces, AmazonRekognition
amazonRekognition) {
        SearchFacesRequest searchFacesRequest = new SearchFacesRequest()
                .withCollectionId(collectionId)
                .withFaceId(faceId)
                .withFaceMatchThreshold(threshold)
                .withMaxFaces(maxFaces);
        return amazonRekognition.searchFaces(searchFacesRequest);
    }

    private static SearchFacesByImageResult callSearchFacesByImage(String
collectionId, Image image, Float threshold, int maxFaces, AmazonRekognition
amazonRekognition) {
        SearchFacesByImageRequest searchFacesByImageRequest = new
SearchFacesByImageRequest()
                .withCollectionId(collectionId)
                .withImage(image)
                .withFaceMatchThreshold(threshold)
                .withMaxFaces(maxFaces);
```

```
        return
 amazonRekognition.searchFacesByImage(searchFacesByImageRequest);
    }

    private static Image getImageUtil(String bucket, String key) {
        return new Image()
                .withS3Object(new S3Object()
                        .withBucket(bucket)
                        .withName(key));
    }
}
```

API Reference

This section provides documentation for the Amazon Rekognition API operations.

Topics

Actions

The following actions are supported:

CompareFaces

Compares a face in the *source* input image with each face detected in the *target* input image.

> **Note**
> If the source image contains multiple faces, the service detects the largest face and uses it to compare with each face detected in the target image.

In response, the operation returns an array of face matches ordered by similarity score with the highest similarity scores first. For each face match, the response provides a bounding box of the face 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.

> **Note**
> By default, only faces with the similarity score of greater than or equal to 80% are returned in the response. You can change this value.

In addition to the face matches, the response returns information about the face in the source image, including the bounding box of the face and confidence value.

> **Note**
> This is a stateless API operation. That is, the operation does not persist any data.

For an example, see Exercise 3: Compare Faces (API) (p. 30)

This operation requires permissions to perform the `rekognition:CompareFaces` action.

Request Syntax

```
{
   "SimilarityThreshold": number,
   "SourceImage": {
      "Bytes": blob,
      "S3Object": {
         "Bucket": "string",
         "Name": "string",
         "Version": "string"
      }
   },
   "TargetImage": {
      "Bytes": blob,
      "S3Object": {
         "Bucket": "string",
         "Name": "string",
         "Version": "string"
      }
   }
}
```

Request Parameters

The request accepts the following data in JSON format.

SimilarityThreshold (p. 52)
The minimum level of confidence in the match you want included in the result.
Type: Float
Valid Range: Minimum value of 0. Maximum value of 100.
Required: No

SourceImage (p. 52)
Source image either as bytes or an S3 object

Type: Image (p. 97) object

Required: Yes

TargetImage (p. 52)

Target image either as bytes or an S3 object

Type: Image (p. 97) object

Required: Yes

Response Syntax

```
{
   "FaceMatches": [
      {
         "Face": {
            "BoundingBox": {
               "Height": number,
               "Left": number,
               "Top": number,
               "Width": number
            },
            "Confidence": number
         },
         "Similarity": number
      }
   ],
   "SourceImageFace": {
      "BoundingBox": {
         "Height": number,
         "Left": number,
         "Top": number,
         "Width": number
      },
      "Confidence": number
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

FaceMatches (p. 53)

Provides an array of `CompareFacesMatch` objects. Each object provides the bounding box, confidence that the bounding box contains a face, and the similarity between the face in the bounding box and the face in the source image.

Type: array of CompareFacesMatch (p. 87) objects

SourceImageFace (p. 53)

The face from the source image that was used for comparison.

Type: ComparedSourceImageFace (p. 86) object

Errors

AccessDeniedException

You are not authorized to perform the action.

HTTP Status Code: 400

ImageTooLargeException

The input image size exceeds the allowed limit. For more information, see Limits in Amazon Rekognition (p. 33).

HTTP Status Code: 400

InternalServerError

Amazon Rekognition experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidImageFormatException

The provided image format is not supported.

HTTP Status Code: 400

InvalidParameterException

Input parameter violated a constraint. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

InvalidS3ObjectException

Amazon Rekognition is unable to access the S3 object specified in the request.

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Rekognition.

HTTP Status Code: 400

ThrottlingException

Amazon Rekognition is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

CreateCollection

Creates a collection in an AWS Region. You can add faces to the collection using the IndexFaces (p. 68) operation.

For example, you might create collections, one for each of your application users. A user can then index faces using the `IndexFaces` operation and persist results in a specific collection. Then, a user can search the collection for faces in the user-specific container.

For an example, see Example 1: Managing Collections (p. 34).

This operation requires permissions to perform the `rekognition:CreateCollection` action.

Request Syntax

```
{
   "CollectionId": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

CollectionId (p. 55)
ID for the collection that you are creating.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 255.
Pattern: `[a-zA-Z0-9_.\-]+`
Required: Yes

Response Syntax

```
{
   "CollectionArn": "string",
   "StatusCode": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CollectionArn (p. 55)
Amazon Resource Name (ARN) of the collection. You can use this to manage permissions on your resources.
Type: String

StatusCode (p. 55)
HTTP status code indicating the result of the operation.
Type: Integer
Valid Range: Minimum value of 0.

Errors

AccessDeniedException
You are not authorized to perform the action.

HTTP Status Code: 400

InternalServerError

Amazon Rekognition experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidParameterException

Input parameter violated a constraint. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Rekognition.

HTTP Status Code: 400

ResourceAlreadyExistsException

A collection with the specified ID already exists.

HTTP Status Code: 400

ThrottlingException

Amazon Rekognition is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

DeleteCollection

Deletes the specified collection. Note that this operation removes all faces in the collection. For an example, see Example 1: Managing Collections (p. 34).

This operation requires permissions to perform the `rekognition:DeleteCollection` action.

Request Syntax

```
{
   "CollectionId": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

CollectionId (p. 57)
ID of the collection to delete.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 255.
Pattern: `[a-zA-Z0-9_.\-]+`
Required: Yes

Response Syntax

```
{
   "StatusCode": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

StatusCode (p. 57)
HTTP status code that indicates the result of the operation.
Type: Integer
Valid Range: Minimum value of 0.

Errors

AccessDeniedException
You are not authorized to perform the action.
HTTP Status Code: 400

InternalServerError
Amazon Rekognition experienced a service issue. Try your call again.
HTTP Status Code: 500

InvalidParameterException
Input parameter violated a constraint. Validate your parameter before calling the API operation again.
HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Rekognition.

HTTP Status Code: 400

ResourceNotFoundException

Collection specified in the request is not found.

HTTP Status Code: 400

ThrottlingException

Amazon Rekognition is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

DeleteFaces

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

This operation requires permissions to perform the `rekognition:DeleteFaces` action.

Request Syntax

```
{
   "CollectionId": "string",
   "FaceIds": [ "string" ]
}
```

Request Parameters

The request accepts the following data in JSON format.

CollectionId (p. 59)
Collection from which to remove the specific faces.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 255.
Pattern: `[a-zA-Z0-9_.\-]+`
Required: Yes

FaceIds (p. 59)
An array of face IDs to delete.
Type: array of Strings
Array Members: Minimum number of 1 item. Maximum number of 4096 items.
Pattern: `[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}`
Required: Yes

Response Syntax

```
{
   "DeletedFaces": [ "string" ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

DeletedFaces (p. 59)
An array of strings (face IDs) of the faces that were deleted.
Type: array of Strings
Array Members: Minimum number of 1 item. Maximum number of 4096 items.
Pattern: `[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}`

Errors

AccessDeniedException
You are not authorized to perform the action.

HTTP Status Code: 400

InternalServerError

Amazon Rekognition experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidParameterException

Input parameter violated a constraint. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Rekognition.

HTTP Status Code: 400

ResourceNotFoundException

Collection specified in the request is not found.

HTTP Status Code: 400

ThrottlingException

Amazon Rekognition is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

DetectFaces

Detects faces within an image (JPEG or PNG) that is provided as input.

For each face detected, the operation returns face details including a bounding box of the face, a confidence value (that the bounding box contains a face), and a fixed set of attributes such as facial landmarks (for example, coordinates of eye and mouth), gender, presence of beard, sunglasses, etc.

The face-detection algorithm is most effective on frontal faces. For non-frontal or obscured faces, the algorithm may not detect the faces or might detect faces with lower confidence.

Note

This is a stateless API operation. That is, the operation does not persist any data.

For an example, see Exercise 2: Detect Faces (API) (p. 28).

This operation requires permissions to perform the `rekognition:DetectFaces` action.

Request Syntax

```
{
   "Attributes": [ "string" ],
   "Image": {
      "Bytes": blob,
      "S3Object": {
         "Bucket": "string",
         "Name": "string",
         "Version": "string"
      }
   }
}
```

Request Parameters

The request accepts the following data in JSON format.

Attributes (p. 61)

A list of facial attributes you would like to be returned. By default, the API returns subset of facial attributes.

For example, you can specify the value as, ["ALL"] or ["DEFAULT"]. If you provide both, ["ALL", "DEFAULT"], the service uses a logical AND operator to determine which attributes to return (in this case, it is all attributes). If you specify all attributes, Amazon Rekognition performs additional detection.

Type: array of Strings

Valid Values: `DEFAULT | ALL`

Required: No

Image (p. 61)

The image in which you want to detect faces. You can specify a blob or an S3 object.

Type: Image (p. 97) object

Required: Yes

Response Syntax

```
{
   "FaceDetails": [
      {
         "Beard": {
            "Confidence": number,
```

```
            "Value": boolean
        },
        "BoundingBox": {
            "Height": number,
            "Left": number,
            "Top": number,
            "Width": number
        },
        "Confidence": number,
        "Emotions": [
            {
                "Confidence": number,
                "Type": "string"
            }
        ],
        "Eyeglasses": {
            "Confidence": number,
            "Value": boolean
        },
        "EyesOpen": {
            "Confidence": number,
            "Value": boolean
        },
        "Gender": {
            "Confidence": number,
            "Value": "string"
        },
        "Landmarks": [
            {
                "Type": "string",
                "X": number,
                "Y": number
            }
        ],
        "MouthOpen": {
            "Confidence": number,
            "Value": boolean
        },
        "Mustache": {
            "Confidence": number,
            "Value": boolean
        },
        "Pose": {
            "Pitch": number,
            "Roll": number,
            "Yaw": number
        },
        "Quality": {
            "Brightness": number,
            "Sharpness": number
        },
        "Smile": {
            "Confidence": number,
            "Value": boolean
        },
        "Sunglasses": {
            "Confidence": number,
            "Value": boolean
        }
```

```
      }
   ],
   "OrientationCorrection": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

FaceDetails (p. 61)

Details of each face found in the image.

Type: array of FaceDetail (p. 92) objects

OrientationCorrection (p. 61)

The algorithm detects the image orientation. If it detects that the image was rotated, it returns the degrees of rotation. If your application is displaying the image, you can use this value to adjust the orientation.

For example, if the service detects that the input image was rotated by 90 degrees, it corrects orientation, performs face detection, and then returns the faces. That is, the bounding box coordinates in the response are based on the corrected orientation.

Type: String

Valid Values: `ROTATE_0 | ROTATE_90 | ROTATE_180 | ROTATE_270`

Errors

AccessDeniedException

You are not authorized to perform the action.

HTTP Status Code: 400

ImageTooLargeException

The input image size exceeds the allowed limit. For more information, see Limits in Amazon Rekognition (p. 33).

HTTP Status Code: 400

InternalServerError

Amazon Rekognition experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidImageFormatException

The provided image format is not supported.

HTTP Status Code: 400

InvalidParameterException

Input parameter violated a constraint. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

InvalidS3ObjectException

Amazon Rekognition is unable to access the S3 object specified in the request.

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Rekognition.

HTTP Status Code: 400

ThrottlingException

Amazon Rekognition is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

DetectLabels

Detects instances of real-world labels within an image (JPEG or PNG) provided as input. This includes objects like flower, tree, and table; events like wedding, graduation, and birthday party; and concepts like landscape, evening, and nature. For an example, see Exercise 1: Detect Labels in an Image (API) (p. 27).

For each object, scene, and concept the API returns one or more labels. Each label provides the object name, and the level of confidence that the image contains the object. For example, suppose the input image has a lighthouse, the sea, and a rock. The response will include all three labels, one for each object.

```
{Name: lighthouse, Confidence: 98.4629}
{Name: rock,Confidence: 79.2097}
{Name: sea,Confidence: 75.061}
```

In the preceding example, the operation returns one label for each of the three objects. The operation can also return multiple labels for the same object in the image. For example, if the input image shows a flower (for example, a tulip), the operation might return the following three labels.

```
{Name: flower,Confidence: 99.0562}
{Name: plant,Confidence: 99.0562}
{Name: tulip,Confidence: 99.0562}
```

In this example, the detection algorithm more precisely identifies the flower as a tulip.

You can provide the input image as an S3 object or as base64-encoded bytes. In response, the API returns an array of labels. In addition, the response also includes the orientation correction. Optionally, you can specify `MinConfidence` to control the confidence threshold for the labels returned. The default is 50%. You can also add the `MaxLabels` parameter to limit the number of labels returned.

> **Note**
> If the object detected is a person, the operation doesn't provide the same facial details that the DetectFaces (p. 61) operation provides.

This is a stateless API operation. That is, the operation does not persist any data.

This operation requires permissions to perform the `rekognition:DetectLabels` action.

Request Syntax

```
{
   "Image": {
      "Bytes": blob,
      "S3Object": {
         "Bucket": "string",
         "Name": "string",
         "Version": "string"
      }
   },
   "MaxLabels": number,
   "MinConfidence": number
}
```

Request Parameters

The request accepts the following data in JSON format.

Image (p. 65)

The input image. You can provide a blob of image bytes or an S3 object.

Type: Image (p. 97) object

Required: Yes

MaxLabels (p. 65)
Maximum number of labels you want the service to return in the response. The service returns the specified number of highest confidence labels.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

MinConfidence (p. 65)
Specifies the minimum confidence level for the labels to return. Amazon Rekognition doesn't return any labels with confidence lower than this specified value.

If `minConfidence` is not specified, the operation returns labels with a confidence values greater than or equal to 50 percent.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

Response Syntax

```
{
   "Labels": [
      {
         "Confidence": number,
         "Name": "string"
      }
   ],
   "OrientationCorrection": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Labels (p. 66)
An array of labels for the real-world objects detected.

Type: array of Label (p. 99) objects

OrientationCorrection (p. 66)
Amazon Rekognition returns the orientation of the input image that was detected (clockwise direction). If your application displays the image, you can use this value to correct the orientation. If Amazon Rekognition detects that the input image was rotated (for example, by 90 degrees), it first corrects the orientation before detecting the labels.

Type: String

Valid Values: `ROTATE_0 | ROTATE_90 | ROTATE_180 | ROTATE_270`

Errors

AccessDeniedException
You are not authorized to perform the action.

HTTP Status Code: 400

ImageTooLargeException
The input image size exceeds the allowed limit. For more information, see Limits in Amazon Rekognition (p. 33).

HTTP Status Code: 400

InternalServerError

Amazon Rekognition experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidImageFormatException

The provided image format is not supported.

HTTP Status Code: 400

InvalidParameterException

Input parameter violated a constraint. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

InvalidS3ObjectException

Amazon Rekognition is unable to access the S3 object specified in the request.

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Rekognition.

HTTP Status Code: 400

ThrottlingException

Amazon Rekognition is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

IndexFaces

Detects faces in the input image and adds them to the specified collection.

Amazon Rekognition does not save the actual faces detected. Instead, the underlying detection algorithm first detects the faces in the input image, and for each face extracts facial features into a feature vector, and stores it in the back-end database. Amazon Rekognition uses feature vectors when performing face match and search operations using the SearchFaces (p. 76) and SearchFacesByImage (p. 79) operations.

If you provide the optional `externalImageID` for the input image you provided, Amazon Rekognition associates this ID with all faces that it detects. When you call the ListFaces (p. 74) operation, the response returns the external ID. You can use this external image ID to create a client-side index to associate the faces with each image. You can then use the index to find all faces in an image.

In response, the operation returns an array of metadata for all detected faces. This includes, the bounding box of the detected face, confidence value (indicating the bounding box contains a face), a face ID assigned by the service for each face that is detected and stored, and an image ID assigned by the service for the input image If you request all facial attributes (using the `detectionAttributes` parameter, Amazon Rekognition returns detailed facial attributes such as facial landmarks (for example, location of eye and mount) and other facial attributes such gender. If you provide the same image, specify the same collection, and use the same external ID in the `IndexFaces` operation, Amazon Rekognition doesn't save duplicate face metadata.

For an example, see Example 2: Storing Faces (p. 37).

This operation requires permissions to perform the `rekognition:IndexFaces` action.

Request Syntax

```
{
   "CollectionId": "string",
   "DetectionAttributes": [ "string" ],
   "ExternalImageId": "string",
   "Image": {
      "Bytes": blob,
      "S3Object": {
         "Bucket": "string",
         "Name": "string",
         "Version": "string"
      }
   }
}
```

Request Parameters

The request accepts the following data in JSON format.

CollectionId (p. 68)

ID of an existing collection to which you want to add the faces that are detected in the input images.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.\-]+`

Required: Yes

DetectionAttributes (p. 68)

(Optional) Returns detailed attributes of indexed faces. By default, the operation returns a subset of the facial attributes.

For example, you can specify the value as, ["ALL"] or ["DEFAULT"]. If you provide both, ["ALL", "DEFAULT"], Amazon Rekognition uses the logical AND operator to determine which attributes to

return (in this case, it is all attributes). If you specify all attributes, the service performs additional detection, in addition to the default.

Type: array of Strings

Valid Values: `DEFAULT | ALL`

Required: No

ExternalImageId (p. 68)

ID you want to assign to all the faces detected in the image.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.\-:]+`

Required: No

Image (p. 68)

Provides the source image either as bytes or an S3 object.

For Amazon Rekognition to process an S3 object, the user must have permission to access the S3 object. For more information, see Resource-Based Policies (p. 111).

Type: Image (p. 97) object

Required: Yes

Response Syntax

```
{
   "FaceRecords": [
      {
         "Face": {
            "BoundingBox": {
               "Height": number,
               "Left": number,
               "Top": number,
               "Width": number
            },
            "Confidence": number,
            "ExternalImageId": "string",
            "FaceId": "string",
            "ImageId": "string"
         },
         "FaceDetail": {
            "Beard": {
               "Confidence": number,
               "Value": boolean
            },
            "BoundingBox": {
               "Height": number,
               "Left": number,
               "Top": number,
               "Width": number
            },
            "Confidence": number,
            "Emotions": [
               {
                  "Confidence": number,
                  "Type": "string"
               }
            ],
            "Eyeglasses": {
```

```
                "Confidence": number,
                "Value": boolean
            },
            "EyesOpen": {
                "Confidence": number,
                "Value": boolean
            },
            "Gender": {
                "Confidence": number,
                "Value": "string"
            },
            "Landmarks": [
                {
                    "Type": "string",
                    "X": number,
                    "Y": number
                }
            ],
            "MouthOpen": {
                "Confidence": number,
                "Value": boolean
            },
            "Mustache": {
                "Confidence": number,
                "Value": boolean
            },
            "Pose": {
                "Pitch": number,
                "Roll": number,
                "Yaw": number
            },
            "Quality": {
                "Brightness": number,
                "Sharpness": number
            },
            "Smile": {
                "Confidence": number,
                "Value": boolean
            },
            "Sunglasses": {
                "Confidence": number,
                "Value": boolean
            }
        }
    }
],
"OrientationCorrection": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

FaceRecords (p. 69)

An array of faces detected and added to the collection. For more information, see Storing Faces In a Face Collection: The IndexFaces Operation (p. 13).

Type: array of FaceRecord (p. 95) objects

OrientationCorrection (p. 69)

The algorithm detects the image orientation. If it detects that the image was rotated, it returns the degree of rotation. You can use this value to correct the orientation and also appropriately analyze the bounding box coordinates that are returned.

Type: String

Valid Values: `ROTATE_0 | ROTATE_90 | ROTATE_180 | ROTATE_270`

Errors

AccessDeniedException

You are not authorized to perform the action.

HTTP Status Code: 400

ImageTooLargeException

The input image size exceeds the allowed limit. For more information, see Limits in Amazon Rekognition (p. 33).

HTTP Status Code: 400

InternalServerError

Amazon Rekognition experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidImageFormatException

The provided image format is not supported.

HTTP Status Code: 400

InvalidParameterException

Input parameter violated a constraint. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

InvalidS3ObjectException

Amazon Rekognition is unable to access the S3 object specified in the request.

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Rekognition.

HTTP Status Code: 400

ResourceNotFoundException

Collection specified in the request is not found.

HTTP Status Code: 400

ThrottlingException

Amazon Rekognition is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

ListCollections

Returns list of collection IDs in your account. If the result is truncated, the response also provides a `NextToken` that you can use in the subsequent request to fetch the next set of collection IDs.

For an example, see Example 1: Managing Collections (p. 34).

This operation requires permissions to perform the `rekognition:ListCollections` action.

Request Syntax

```
{
   "MaxResults": number,
   "NextToken": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

MaxResults (p. 72)
Maximum number of collection IDs to return.
Type: Integer
Valid Range: Minimum value of 0. Maximum value of 4096.
Required: No

NextToken (p. 72)
Pagination token from the previous response.
Type: String
Length Constraints: Maximum length of 255.
Required: No

Response Syntax

```
{
   "CollectionIds": [ "string" ],
   "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CollectionIds (p. 72)
An array of collection IDs.
Type: array of Strings
Length Constraints: Minimum length of 1. Maximum length of 255.
Pattern: `[a-zA-Z0-9_.\-]+`

NextToken (p. 72)
If the result is truncated, the response provides a `NextToken` that you can use in the subsequent request to fetch the next set of collection IDs.
Type: String
Length Constraints: Maximum length of 255.

Errors

AccessDeniedException
You are not authorized to perform the action.
HTTP Status Code: 400

InternalServerError
Amazon Rekognition experienced a service issue. Try your call again.
HTTP Status Code: 500

InvalidPaginationTokenException
Pagination token in the request is not valid.
HTTP Status Code: 400

InvalidParameterException
Input parameter violated a constraint. Validate your parameter before calling the API operation again.
HTTP Status Code: 400

ProvisionedThroughputExceededException
The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Rekognition.
HTTP Status Code: 400

ResourceNotFoundException
Collection specified in the request is not found.
HTTP Status Code: 400

ThrottlingException
Amazon Rekognition is temporarily unable to process the request. Try your call again.
HTTP Status Code: 500

ListFaces

Returns metadata for faces in the specified collection. This metadata includes information such as the bounding box coordinates, the confidence (that the bounding box contains a face), and face ID. For an example, see Example 3: Searching Faces (p. 45).

This operation requires permissions to perform the `rekognition:ListFaces` action.

Request Syntax

```
{
   "CollectionId": "string",
   "MaxResults": number,
   "NextToken": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

CollectionId (p. 74)
ID of the collection from which to list the faces.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.\-]+`

Required: Yes

MaxResults (p. 74)
Maximum number of faces to return.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 4096.

Required: No

NextToken (p. 74)
If the previous response was incomplete (because there is more data to retrieve), Amazon Rekognition returns a pagination token in the response. You can use this pagination token to retrieve the next set of faces.

Type: String

Length Constraints: Maximum length of 255.

Required: No

Response Syntax

```
{
   "Faces": [
      {
         "BoundingBox": {
            "Height": number,
            "Left": number,
            "Top": number,
            "Width": number
         },
         "Confidence": number,
         "ExternalImageId": "string",
         "FaceId": "string",
```

```
            "ImageId": "string"
        }
    ],
    "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Faces (p. 74)

An array of `Face` objects.

Type: array of Face (p. 91) objects

NextToken (p. 74)

If the response is truncated, Amazon Rekognition returns this token that you can use in the subsequent request to retrieve the next set of faces.

Type: String

Errors

AccessDeniedException

You are not authorized to perform the action.

HTTP Status Code: 400

InternalServerError

Amazon Rekognition experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidPaginationTokenException

Pagination token in the request is not valid.

HTTP Status Code: 400

InvalidParameterException

Input parameter violated a constraint. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Rekognition.

HTTP Status Code: 400

ResourceNotFoundException

Collection specified in the request is not found.

HTTP Status Code: 400

ThrottlingException

Amazon Rekognition is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

SearchFaces

For a given input face ID, searches the specified collection for matching faces. You get a face ID when you add a face to the collection using the IndexFaces (p. 68) operation. The operation compares the features of the input face with faces in the specified collection.

> **Note**
> You can also search faces without indexing faces by using the `SearchFacesByImage` operation.

The operation response returns an array of faces that match, ordered by similarity score with the highest similarity first. More specifically, it is an array of metadata for each face match that is found. Along with the metadata, the response also includes a `confidence` value for each face match, indicating the confidence that the specific face matches the input face.

For an example, see Example 3: Searching Faces (p. 45).

This operation requires permissions to perform the `rekognition:SearchFaces` action.

Request Syntax

```
{
   "CollectionId": "string",
   "FaceId": "string",
   "FaceMatchThreshold": number,
   "MaxFaces": number
}
```

Request Parameters

The request accepts the following data in JSON format.

CollectionId (p. 76)
ID of the collection to search.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 255.
Pattern: `[a-zA-Z0-9_.\-]+`
Required: Yes

FaceId (p. 76)
ID of a face to find matches for in the collection.
Type: String
Pattern: `[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}`
Required: Yes

FaceMatchThreshold (p. 76)
Optional value specifying the minimum confidence in the face match to return. For example, don't return any matches where confidence in matches is less than 70%.
Type: Float
Valid Range: Minimum value of 0. Maximum value of 100.
Required: No

MaxFaces (p. 76)
Maximum number of faces to return. The operation returns the maximum number of faces with the highest confidence in the match.
Type: Integer
Valid Range: Minimum value of 1. Maximum value of 4096.
Required: No

Response Syntax

```
{
   "FaceMatches": [
      {
         "Face": {
            "BoundingBox": {
               "Height": number,
               "Left": number,
               "Top": number,
               "Width": number
            },
            "Confidence": number,
            "ExternalImageId": "string",
            "FaceId": "string",
            "ImageId": "string"
         },
         "Similarity": number
      }
   ],
   "SearchedFaceId": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

FaceMatches (p. 77)

An array of faces that matched the input face, along with the confidence in the match.

Type: array of FaceMatch (p. 94) objects

SearchedFaceId (p. 77)

ID of the face that was searched for matches in a collection.

Type: String

Pattern: `[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}`

Errors

AccessDeniedException

You are not authorized to perform the action.

HTTP Status Code: 400

InternalServerError

Amazon Rekognition experienced a service issue. Try your call again.

HTTP Status Code: 500

InvalidParameterException

Input parameter violated a constraint. Validate your parameter before calling the API operation again.

HTTP Status Code: 400

ProvisionedThroughputExceededException

The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Rekognition.

HTTP Status Code: 400

ResourceNotFoundException

Collection specified in the request is not found.

HTTP Status Code: 400

ThrottlingException

Amazon Rekognition is temporarily unable to process the request. Try your call again.

HTTP Status Code: 500

SearchFacesByImage

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 in the specified collection.

Note

To search for all faces in an input image, you might first call the IndexFaces (p. 68) operation, and then use the face IDs returned in subsequent calls to the SearchFaces (p. 76) operation. You can also call the `DetectFaces` operation and use the bounding boxes in the response to make face crops, which then you can pass in to the `SearchFacesByImage` operation.

The response returns an array of faces that match, ordered by similarity score with the highest similarity first. More specifically, it is an array of metadata for each face match found. Along with the metadata, the response also includes a `similarity` indicating how similar the face is to the input face. In the response, the operation also returns the bounding box (and a confidence level that the bounding box contains a face) of the face that Amazon Rekognition used for the input image.

For an example, see Example 3: Searching Faces (p. 45).

This operation requires permissions to perform the `rekognition:SearchFacesByImage` action.

Request Syntax

```
{
   "CollectionId": "string",
   "FaceMatchThreshold": number,
   "Image": {
      "Bytes": blob,
      "S3Object": {
         "Bucket": "string",
         "Name": "string",
         "Version": "string"
      }
   },
   "MaxFaces": number
}
```

Request Parameters

The request accepts the following data in JSON format.

CollectionId (p. 79)

ID of the collection to search.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.\-]+`

Required: Yes

FaceMatchThreshold (p. 79)

(Optional) Specifies the minimum confidence in the face match to return. For example, don't return any matches where confidence in matches is less than 70%.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

Image (p. 79)

Provides the source image either as bytes or an S3 object.

For Amazon Rekognition to process an S3 object, the user must have permission to access the S3 object. For more information, see Resource-Based Policies (p. 111).

Type: Image (p. 97) object

Required: Yes

MaxFaces (p. 79)

Maximum number of faces to return. The operation returns the maximum number of faces with the highest confidence in the match.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 4096.

Required: No

Response Syntax

```
{
   "FaceMatches": [
      {
         "Face": {
            "BoundingBox": {
               "Height": number,
               "Left": number,
               "Top": number,
               "Width": number
            },
            "Confidence": number,
            "ExternalImageId": "string",
            "FaceId": "string",
            "ImageId": "string"
         },
         "Similarity": number
      }
   ],
   "SearchedFaceBoundingBox": {
      "Height": number,
      "Left": number,
      "Top": number,
      "Width": number
   },
   "SearchedFaceConfidence": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

FaceMatches (p. 80)

An array of faces that match the input face, along with the confidence in the match.

Type: array of FaceMatch (p. 94) objects

SearchedFaceBoundingBox (p. 80)

The bounding box around the face in the input image that Amazon Rekognition used for the search.

Type: BoundingBox (p. 84) object

SearchedFaceConfidence (p. 80)

The level of confidence that the `searchedFaceBoundingBox`, contains a face.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Errors

AccessDeniedException
You are not authorized to perform the action.
HTTP Status Code: 400

ImageTooLargeException
The input image size exceeds the allowed limit. For more information, see Limits in Amazon Rekognition (p. 33).
HTTP Status Code: 400

InternalServerError
Amazon Rekognition experienced a service issue. Try your call again.
HTTP Status Code: 500

InvalidImageFormatException
The provided image format is not supported.
HTTP Status Code: 400

InvalidParameterException
Input parameter violated a constraint. Validate your parameter before calling the API operation again.
HTTP Status Code: 400

InvalidS3ObjectException
Amazon Rekognition is unable to access the S3 object specified in the request.
HTTP Status Code: 400

ProvisionedThroughputExceededException
The number of requests exceeded your throughput limit. If you want to increase this limit, contact Amazon Rekognition.
HTTP Status Code: 400

ResourceNotFoundException
Collection specified in the request is not found.
HTTP Status Code: 400

ThrottlingException
Amazon Rekognition is temporarily unable to process the request. Try your call again.
HTTP Status Code: 500

Data Types

The following data types are supported:

Beard

Indicates whether or not the face has a beard, and the confidence level in the determination.

Contents

Confidence

Level of confidence in the determination.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

Value

Boolean value that indicates whether the face has beard or not.

Type: Boolean

Required: No

BoundingBox

Identifies the bounding box around the object or face. The `left` (x-coordinate) and `top` (y-coordinate) are coordinates representing the top and left sides of the bounding box. Note that the upper-left corner of the image is the origin (0,0).

The `top` and `left` values returned are ratios of the overall image size. For example, if the input image is 700x200 pixels, and the top-left coordinate of the bounding box is 350x50 pixels, the API returns a `left` value of 0.5 (350/700) and a `top` value of 0.25 (50/200).

The `width` and `height` values represent the dimensions of the bounding box as a ratio of the overall image dimension. For example, if the input image is 700x200 pixels, and the bounding box width is 70 pixels, the width returned is 0.1.

> **Note**
> The bounding box coordinates can have negative values. For example, if Amazon Rekognition is able to detect a face that is at the image edge and is only partially visible, the service can return coordinates that are outside the image bounds and, depending on the image edge, you might get negative values or values greater than 1 for the `left` or `top` values.

Contents

Height

Height of the bounding box as a ratio of the overall image height.

Type: Float

Required: No

Left

Left coordinate of the bounding box as a ratio of overall image width.

Type: Float

Required: No

Top

Top coordinate of the bounding box as a ratio of overall image height.

Type: Float

Required: No

Width

Width of the bounding box as a ratio of the overall image width.

Type: Float

Required: No

ComparedFace

Provides face metadata (bounding box and confidence that the bounding box actually contains a face).

Contents

BoundingBox

Identifies the bounding box around the object or face. The `left` (x-coordinate) and `top` (y-coordinate) are coordinates representing the top and left sides of the bounding box. Note that the upper-left corner of the image is the origin (0,0).

The `top` and `left` values returned are ratios of the overall image size. For example, if the input image is 700x200 pixels, and the top-left coordinate of the bounding box is 350x50 pixels, the API returns a `left` value of 0.5 (350/700) and a `top` value of 0.25 (50/200).

The `width` and `height` values represent the dimensions of the bounding box as a ratio of the overall image dimension. For example, if the input image is 700x200 pixels, and the bounding box width is 70 pixels, the width returned is 0.1.

> **Note**
> The bounding box coordinates can have negative values. For example, if Amazon Rekognition is able to detect a face that is at the image edge and is only partially visible, the service can return coordinates that are outside the image bounds and, depending on the image edge, you might get negative values or values greater than 1 for the `left` or `top` values.

Type: BoundingBox (p. 84) object

Required: No

Confidence

Level of confidence that what the bounding box contains is a face.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

ComparedSourceImageFace

Type that describes the face Amazon Rekognition chose to compare with the faces in the target. This contains a bounding box for the selected face and confidence level that the bounding box contains a face. Note that Amazon Rekognition selects the largest face in the source image for this comparison.

Contents

BoundingBox

Identifies the bounding box around the object or face. The `left` (x-coordinate) and `top` (y-coordinate) are coordinates representing the top and left sides of the bounding box. Note that the upper-left corner of the image is the origin (0,0).

The `top` and `left` values returned are ratios of the overall image size. For example, if the input image is 700x200 pixels, and the top-left coordinate of the bounding box is 350x50 pixels, the API returns a `left` value of 0.5 (350/700) and a `top` value of 0.25 (50/200).

The `width` and `height` values represent the dimensions of the bounding box as a ratio of the overall image dimension. For example, if the input image is 700x200 pixels, and the bounding box width is 70 pixels, the width returned is 0.1.

> **Note**
> The bounding box coordinates can have negative values. For example, if Amazon Rekognition is able to detect a face that is at the image edge and is only partially visible, the service can return coordinates that are outside the image bounds and, depending on the image edge, you might get negative values or values greater than 1 for the `left` or `top` values.

Type: BoundingBox (p. 84) object

Required: No

Confidence

Confidence level that the selected bounding box contains a face.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

CompareFacesMatch

For the provided the bounding box, confidence level that the bounding box actually contains a face, and the similarity between the face in the bounding box and the face in the source image.

Contents

Face

Provides face metadata (bounding box and confidence that the bounding box actually contains a face).

Type: ComparedFace (p. 85) object

Required: No

Similarity

Level of confidence that the faces match.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

Emotion

The emotions detected on the face, and the confidence level in the determination. For example, HAPPY, SAD, and ANGRY.

Contents

Confidence

Level of confidence in the determination.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

Type

Type of emotion detected.

Type: String

Valid Values: `HAPPY | SAD | ANGRY | CONFUSED | DISGUSTED | SURPRISED | CALM | UNKNOWN`

Required: No

Eyeglasses

Indicates whether or not the face is wearing eye glasses, and the confidence level in the determination.

Contents

Confidence

Level of confidence in the determination.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

Value

Boolean value that indicates whether the face is wearing eye glasses or not.

Type: Boolean

Required: No

EyeOpen

Indicates whether or not the eyes on the face are open, and the confidence level in the determination.

Contents

Confidence

Level of confidence in the determination.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

Value

Boolean value that indicates whether the eyes on the face are open.

Type: Boolean

Required: No

Face

Describes the face properties such as the bounding box, face ID, image ID of the source image, and external image ID that you assigned.

Contents

BoundingBox

Identifies the bounding box around the object or face. The `left` (x-coordinate) and `top` (y-coordinate) are coordinates representing the top and left sides of the bounding box. Note that the upper-left corner of the image is the origin (0,0).

The `top` and `left` values returned are ratios of the overall image size. For example, if the input image is 700x200 pixels, and the top-left coordinate of the bounding box is 350x50 pixels, the API returns a `left` value of 0.5 (350/700) and a `top` value of 0.25 (50/200).

The `width` and `height` values represent the dimensions of the bounding box as a ratio of the overall image dimension. For example, if the input image is 700x200 pixels, and the bounding box width is 70 pixels, the width returned is 0.1.

> **Note**
> The bounding box coordinates can have negative values. For example, if Amazon Rekognition is able to detect a face that is at the image edge and is only partially visible, the service can return coordinates that are outside the image bounds and, depending on the image edge, you might get negative values or values greater than 1 for the `left` or `top` values.

Type: BoundingBox (p. 84) object

Required: No

Confidence

Confidence level that the bounding box contains a face (and not a different object such as a tree).

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

ExternalImageId

Identifier that you assign to all the faces in the input image.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Pattern: `[a-zA-Z0-9_.\-:]+`

Required: No

FaceId

Unique identifier that Amazon Rekognition assigns to the face.

Type: String

Pattern: `[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}`

Required: No

ImageId

Unique identifier that Amazon Rekognition assigns to the source image.

Type: String

Pattern: `[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}`

Required: No

FaceDetail

Structure containing attributes of the face that the algorithm detected.

Contents

Beard

Indicates whether or not the face has a beard, and the confidence level in the determination.

Type: Beard (p. 83) object

Required: No

BoundingBox

Bounding box of the face.

Type: BoundingBox (p. 84) object

Required: No

Confidence

Confidence level that the bounding box contains a face (and not a different object such as a tree).

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

Emotions

The emotions detected on the face, and the confidence level in the determination. For example, HAPPY, SAD, and ANGRY.

Type: array of Emotion (p. 88) objects

Required: No

Eyeglasses

Indicates whether or not the face is wearing eye glasses, and the confidence level in the determination.

Type: Eyeglasses (p. 89) object

Required: No

EyesOpen

Indicates whether or not the eyes on the face are open, and the confidence level in the determination.

Type: EyeOpen (p. 90) object

Required: No

Gender

Gender of the face and the confidence level in the determination.

Type: Gender (p. 96) object

Required: No

Landmarks

Indicates the location of the landmark on the face.

Type: array of Landmark (p. 100) objects

Required: No

MouthOpen

Indicates whether or not the mouth on the face is open, and the confidence level in the determination.

Type: MouthOpen (p. 101) object

Required: No

Mustache

Indicates whether or not the face has a mustache, and the confidence level in the determination.

Type: Mustache (p. 102) object

Required: No

Pose

Indicates the pose of the face as determined by pitch, roll, and the yaw.

Type: Pose (p. 103) object

Required: No

Quality

Identifies image brightness and sharpness.

Type: ImageQuality (p. 98) object

Required: No

Smile

Indicates whether or not the face is smiling, and the confidence level in the determination.

Type: Smile (p. 105) object

Required: No

Sunglasses

Indicates whether or not the face is wearing sunglasses, and the confidence level in the determination.

Type: Sunglasses (p. 106) object

Required: No

FaceMatch

Provides face metadata. In addition, it also provides the confidence in the match of this face with the input face.

Contents

Face

Describes the face properties such as the bounding box, face ID, image ID of the source image, and external image ID that you assigned.

Type: Face (p. 91) object

Required: No

Similarity

Confidence in the match of this face with the input face.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

FaceRecord

Object containing both the face metadata (stored in the back-end database) and facial attributes that are detected but aren't stored in the database.

Contents

Face

Describes the face properties such as the bounding box, face ID, image ID of the source image, and external image ID that you assigned.

Type: Face (p. 91) object

Required: No

FaceDetail

Structure containing attributes of the face that the algorithm detected.

Type: FaceDetail (p. 92) object

Required: No

Gender

Gender of the face and the confidence level in the determination.

Contents

Confidence

Level of confidence in the determination.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

Value

Gender of the face.

Type: String

Valid Values: `MALE | FEMALE`

Required: No

Image

Provides the source image either as bytes or an S3 object.

For Amazon Rekognition to process an S3 object, the user must have permission to access the S3 object. For more information, see Resource-Based Policies (p. 111).

Contents

Bytes

Blob of image bytes up to 5 MBs.

Type: Base64-encoded binary data

Length Constraints: Minimum length of 1. Maximum length of 5242880.

Required: No

S3Object

Identifies an S3 object as the image source.

Type: S3Object (p. 104) object

Required: No

ImageQuality

Identifies image brightness and sharpness.

Contents

Brightness

Value representing brightness of the face. The service returns a value between 0 and 1 (inclusive).

Type: Float

Required: No

Sharpness

Value representing sharpness of the face.

Type: Float

Required: No

Label

Structure containing details about the detected label, including bounding box, name, and level of confidence.

Contents

Confidence

Level of confidence.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

Name

The name (label) of the object.

Type: String

Required: No

Landmark

Indicates the location of the landmark on the face.

Contents

Type

Type of the landmark.

Type: String

Valid Values: `EYE_LEFT | EYE_RIGHT | NOSE | MOUTH_LEFT | MOUTH_RIGHT | LEFT_EYEBROW_LEFT | LEFT_EYEBROW_RIGHT | LEFT_EYEBROW_UP | RIGHT_EYEBROW_LEFT | RIGHT_EYEBROW_RIGHT | RIGHT_EYEBROW_UP | LEFT_EYE_LEFT | LEFT_EYE_RIGHT | LEFT_EYE_UP | LEFT_EYE_DOWN | RIGHT_EYE_LEFT | RIGHT_EYE_RIGHT | RIGHT_EYE_UP | RIGHT_EYE_DOWN | NOSE_LEFT | NOSE_RIGHT | MOUTH_UP | MOUTH_DOWN | LEFT_PUPIL | RIGHT_PUPIL`

Required: No

X

x-coordinate from the top left of the landmark expressed as the ration of the width of the image. For example, if the images is 700x200 and the x-coordinate of the landmark is at 350 pixels, this value is 0.5.

Type: Float

Required: No

Y

y-coordinate from the top left of the landmark expressed as the ration of the height of the image. For example, if the images is 700x200 and the y-coordinate of the landmark is at 100 pixels, this value is 0.5.

Type: Float

Required: No

MouthOpen

Indicates whether or not the mouth on the face is open, and the confidence level in the determination.

Contents

Confidence

Level of confidence in the determination.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

Value

Boolean value that indicates whether the mouth on the face is open or not.

Type: Boolean

Required: No

Mustache

Indicates whether or not the face has a mustache, and the confidence level in the determination.

Contents

Confidence

Level of confidence in the determination.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

Value

Boolean value that indicates whether the face has mustache or not.

Type: Boolean

Required: No

Pose

Indicates the pose of the face as determined by pitch, roll, and the yaw.

Contents

Pitch

Value representing the face rotation on the pitch axis.

Type: Float

Valid Range: Minimum value of -180. Maximum value of 180.

Required: No

Roll

Value representing the face rotation on the roll axis.

Type: Float

Valid Range: Minimum value of -180. Maximum value of 180.

Required: No

Yaw

Value representing the face rotation on the yaw axis.

Type: Float

Valid Range: Minimum value of -180. Maximum value of 180.

Required: No

S3Object

Provides the S3 bucket name and object name.

For Amazon Rekognition to process an S3 object, the user must have permission to access the S3 object. For more information, see Resource-Based Policies (p. 111).

Contents

Bucket

Name of the S3 bucket.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 255.

Pattern: `[0-9A-Za-z\.\-_]*`

Required: No

Name

S3 object key name.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

Version

If the bucket is versioning enabled, you can specify the object version.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

Smile

Indicates whether or not the face is smiling, and the confidence level in the determination.

Contents

Confidence

Level of confidence in the determination.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

Value

Boolean value that indicates whether the face is smiling or not.

Type: Boolean

Required: No

Sunglasses

Indicates whether or not the face is wearing sunglasses, and the confidence level in the determination.

Contents

Confidence

Level of confidence in the determination.

Type: Float

Valid Range: Minimum value of 0. Maximum value of 100.

Required: No

Value

Boolean value that indicates whether the face is wearing sunglasses or not.

Type: Boolean

Required: No

Authentication and Access Control for Amazon Rekognition

Access to Amazon Rekognition requires credentials. Those credentials must have permissions to access AWS resources, such as an Amazon Rekognition collection. The following sections provide details on how you can use AWS Identity and Access Management (IAM) and Amazon Rekognition to help secure access to your resources.

Authentication

You can access AWS as any of the following types of identities:

- **AWS account root user** – When you sign up for AWS, you provide an email address and password that is associated with your AWS account. These are your *root credentials* and they provide complete access to all of your AWS resources.

  Important
  For security reasons, we recommend that you use the root credentials only to create an *administrator user*, which is an *IAM user* with full permissions to your AWS account. Then, you can use this administrator user to create other IAM users and roles with limited permissions. For more information, see IAM Best Practices and Creating an Admin User and Group in the *IAM User Guide*.
- **IAM user** – An IAM user is simply an identity within your AWS account that has specific custom permissions (for example, permissions to create a collection in Amazon Rekognition). You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

  In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically, either through one of the

several SDKs or by using the AWS Command Line Interface (CLI). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don't use the AWS tools, you must sign the request yourself. Amazon Rekognition supports *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the *AWS General Reference*.

- **IAM role** – An IAM role is another IAM identity you can create in your account that has specific permissions. It is similar to an *IAM user*, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:

  - **Federated user access** – Instead of creating an IAM user, you can use preexisting user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the *IAM User Guide*.

  - **Cross-account access** – You can use an IAM role in your account to grant another AWS account permissions to access your account's resources. For an example, see Tutorial: Delegate Access Across AWS Accounts Using IAM Roles in the *IAM User Guide*.

  - **AWS service access** – You can use an IAM role in your account to grant an AWS service permissions to access your account's resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data stored in the bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

  - **Applications running on Amazon EC2** – Instead of storing access keys within the EC2 instance for use by applications running on the instance and making AWS API requests, you can use an IAM role to manage temporary credentials for these applications. To assign an AWS role to an EC2 instance and make it available to all of its applications, you can create an instance profile that is attached to the instance. An instance profile contains the role and enables programs running on the EC2 instance to get temporary credentials. For more information, see Using Roles for Applications on Amazon EC2 in the *IAM User Guide*.

Access Control

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot create or access Amazon Rekognition resources. For example, you must have permissions to create an Amazon Rekognition collection.

The following sections describe how to manage permissions for Amazon Rekognition. We recommend that you read the overview first.

Overview of Managing Access Permissions to Your Amazon Rekognition Resources

Every AWS resource is owned by an AWS account, and permissions to create or access a resource are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles), and some services (such as AWS Lambda) also support attaching permissions policies to resources.

> **Note**
> An *account administrator* (or administrator user) is a user with administrator privileges. For more information, see IAM Best Practices in the *IAM User Guide*.

When granting permissions, you decide who is getting the permissions, the resources they get permissions for, and the specific actions that you want to allow on those resources.

Topics

Amazon Rekognition Resources and Operations

In Amazon Rekognition, the primary resource is *a collection*. In a policy, you use an Amazon Resource Name (ARN) to identify the resource that the policy applies to.

These resources have unique Amazon Resource Names (ARNs) associated with them, as shown in the following table.

Resource Type	ARN Format
Collection ARN	`arn:aws:rekognition:`*`region`*`:`*`account-id`*`:collection/`*`collection-id`*

Amazon Rekognition provides a set of operations to work with Amazon Rekognition resources. For a list of available operations, see Amazon Rekognition Amazon Rekognition API Permissions: Actions, Permissions, and Resources Reference (p. 115).

Understanding Resource Ownership

The AWS account owns the resources that are created in the account, regardless of who created the resources. Specifically, the resource owner is the AWS account of the principal entity (that is, the root account or an IAM user) that authenticates the resource creation request. The following examples illustrate how this works:

- If you use the root account credentials of your AWS account to create a collection, your AWS account is the owner of the resource (in Amazon Rekognition, the resource is a collection).
- If you create an IAM user in your AWS account and grant permissions to create a collection to that user, the user can create a collection. However, your AWS account, to which the user belongs, owns the collection resource.

Managing Access to Resources

A *permissions policy* describes who has access to what. The following section explains the available options for creating permissions policies.

> **Note**
> This section discusses using IAM in the context of Amazon Rekognition. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see What Is IAM? in the *IAM User Guide*. For information about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

Policies attached to an IAM identity are referred to as *identity-based* policies (IAM polices) and policies attached to a resource are referred to as *resource-based* policies. Amazon Rekognition supports identity-based policies.

Topics

Identity-Based Policies (IAM Policies)

You can attach policies to IAM identities. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – To grant a user permissions to create an Amazon Rekognition resource, such as a collection, you can attach a permissions policy to a user or group that the user belongs to.
- **Attach a permissions policy to a role (grant cross-account permissions)** – You can attach an identity-based permissions policy to an IAM role to grant cross-account permissions. For example, the administrator in account A can create a role to grant cross-account permissions to another AWS account (for example, account B) or an AWS service as follows:
  1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in account A.
  2. Account A administrator attaches a trust policy to the role identifying account B as the principal who can assume the role.
  3. Account B administrator can then delegate permissions to assume the role to any users in account B. Doing this allows users in account B to create or access resources in account A. The principal in the trust policy can also be an AWS service principal if you want to grant an AWS service permissions to assume the role.

  For more information about using IAM to delegate permissions, see Access Management in the *IAM User Guide*.

The following is an example policy that lists all collections.

```
"Version": "2012-10-17",
             "Statement": [
       {
           "Sid": "AllowsListCollectionAction",
           "Effect": "Allow",
           "Action": [
               "rekognition:ListCollections"
           ],
           "Resource": "*"
       }
   ]
```

```
}
```

For more information about using identity-based policies with Amazon Rekognition, see Using Identity-Based Policies (IAM Policies) for Amazon Rekognition (p. 112). For more information about users, groups, roles, and permissions, see Identities (Users, Groups, and Roles) in the *IAM User Guide*.

Resource-Based Policies

Other services, such as Amazon S3, also support resource-based permissions policies. For example, you can attach a policy to an S3 bucket to manage access permissions to that bucket. Amazon Rekognition doesn't support resource-based policies.

To access images stored in an Amazon S3 bucket, you must have permission to access object in the S3 bucket. With this permission, Amazon Rekognition can download images from the S3 bucket. The following example policy allows the user to perform the `s3:GetObject` action on the S3 bucket named Tests3bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": [
                "arn:aws:s3:::Tests3bucket"
            ]
        }
    ]
}
```

Specifying Policy Elements: Actions, Effects, and Principals

For each Amazon Rekognition resource, the service defines a set of API operations. To grant permissions for these API operations, Amazon Rekognition defines a set of actions that you can specify in a policy. Some API operations can require permissions for more than one action in order to perform the API operation. For more information about resources and API operations, see Amazon Rekognition Resources and Operations (p. 109) and Amazon Rekognition Amazon Rekognition API Permissions: Actions, Permissions, and Resources Reference (p. 115).

The following are the most basic policy elements:

- **Resource** – You use an Amazon Resource Name (ARN) to identify the resource that the policy applies to. For more information, see Amazon Rekognition Resources and Operations (p. 109).
- **Action** – You use action keywords to identify resource operations that you want to allow or deny. For example, you can use `ListCollections` to list collections.
- **Effect** – You specify the effect, either allow or deny, when the user requests the specific action. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource, which you might do to make sure that a user cannot access it, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal. For resource-based policies, you specify the user, account, service, or other entity

that you want to receive permissions (applies to resource-based policies only). Amazon Rekognition doesn't support resource-based policies.

To learn more about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

For a list showing all of the Amazon Rekognition API operations and the resources that they apply to, see Amazon Rekognition API Permissions: Actions, Permissions, and Resources Reference (p. 115).

Specifying Conditions in a Policy

When you grant permissions, you can use the access policy language to specify the conditions when a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see Condition in the *IAM User Guide*.

To express conditions, you use predefined condition keys. There are no condition keys specific to Amazon Rekognition. However, there are AWS-wide condition keys that you can use as appropriate. For a complete list of AWS-wide keys, see Available Keys for Conditions in the *IAM User Guide*.

Using Identity-Based Policies (IAM Policies) for Amazon Rekognition

This topic provides examples of identity-based policies that demonstrate how an account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles) and thereby grant permissions to perform operations on Amazon Rekognition resources.

Important
We recommend that you first review the introductory topics that explain the basic concepts and options available to manage access to your Amazon Rekognition resources. For more information, see Overview of Managing Access Permissions to Your Amazon Rekognition Resources (p. 109).

Topics

The following shows an example of a permissions policy.

```
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:CompareFaces",
                "rekognition:DetectFaces",
                "rekognition:DetectLabels",
                "rekognition:ListCollections",
                "rekognition:ListFaces",
                "rekognition:SearchFaces",
```

```
                "rekognition:SearchFacesByImage"
            ],
            "Resource": "*"
        }
    ]
```

This policy example grants read-only access to a user. That is, the user can't list perform write actions in your account.

For a table showing all of the Amazon Rekognition API operations and the resources that they apply to, see Amazon Rekognition API Permissions: Actions, Permissions, and Resources Reference (p. 115).

Permissions Required to Use the Amazon Rekognition Console

Amazon Rekognition does not require any additional permissions when working with the Amazon Rekognition console.

AWS Managed (Predefined) Policies for Amazon Rekognition

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant necessary permissions for common use cases so that you can avoid having to investigate what permissions are needed. For more information, see AWS Managed Policies in the *IAM User Guide*.

The following AWS managed policies, which you can attach to users in your account, are specific to Amazon Rekognition:

- **AmazonRekognitionFullAccess** – Grants full access to Amazon Rekognition resources including creating and deleting collections.
- **AmazonRekognitionReadWriteAcces** – Grants read and write access to Amazon Rekognition resources except creating and deleting collections.
- **AmazonRekognitionReadOnlyAccess** – Grants read-only access to Amazon Rekognition resources.

> **Note**
> You can review these permissions policies by signing in to the IAM console and searching for specific policies there.
> These policies work when you are using AWS SDKs or the AWS CLI.

You can also create your own custom IAM policies to allow permissions for Amazon Rekognition actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions.

Customer Managed Policy Examples

In this section, you can find example user policies that grant permissions for various Amazon Rekognition actions. These policies work when you are using AWS SDKs or the AWS CLI. When you are using the console, you need to grant additional permissions specific to the console, which is discussed in Permissions Required to Use the Amazon Rekognition Console (p. 113).

Note
All examples use the us-west-2 region and contain fictitious account IDs.

Examples

Example 1: Allow a User Read-Only Access to Resources

The following example grants read-only access to Amazon Rekognition resources.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:CompareFaces",
                "rekognition:DetectFaces",
                "rekognition:DetectLabels",
                "rekognition:ListCollections",
                "rekognition:ListFaces",
                "rekognition:SearchFaces",
                "rekognition:SearchFacesByImage"
            ],
            "Resource": "*"
        }
    ]
}
```

Example 2: Allow a User Full Access to Resources

The following example grants full access to Amazon Rekognition resources.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:CompareFaces",
                "rekognition:CreateCollection,
                "rekognition:DeleteCollection",
                "rekognition:DeleteFaces",
                "rekognition:DetectFaces",
                "rekognition:DetectLabels",
                "rekognition:IndexFaces",
                "rekognition:ListCollections",
                "rekognition:ListFaces",
                "rekognition:SearchFaces",
                "rekognition:SearchFacesByImage"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

Amazon Rekognition API Permissions: Actions, Permissions, and Resources Reference

When you are setting up Access Control (p. 108) and writing a permissions policy that you can attach to an IAM identity (identity-based policies), you can use the following list as a reference. The list includes each Amazon Rekognition API operation, the corresponding actions for which you can grant permissions to perform the action, and the AWS resource for which you can grant the permissions. You specify the actions in the policy's `Action` field, and you specify the resource value in the policy's `Resource` field.

You can use AWS-wide condition keys in your Amazon Rekognition policies to express conditions. For a complete list of AWS-wide keys, see Available Keys in the *IAM User Guide*.

> **Note**
> To specify an action, use the `rekognition` prefix followed by the API operation name (for example, `rekognition:DeleteCollection`).

Amazon Rekognition API and Required Permissions for Actions

API Operation: CompareFaces
Required Permissions (API Action): `rekognition:CompareFaces`

Resources: `arn:aws:rekognition:`*`region`*`:`*`account-id`*`:`*`collection/collection-id`*

API Operation: CreateCollection
Required Permissions (API Action): rekognition:CreateCollection

Resources: arn:aws:rekognition:*region*:*account-id*:*collection/collection-id*

API Operation: DeleteCollection
Required Permissions (API Action): rekognition:DeleteCollection

Resources: arn:aws:rekognition:*region*:*account-id*:*collection/collection-id*

API Operation: DeleteFaces
Required Permissions (API Action): rekognition:DeleteFaces

Resources: arn:aws:rekognition:*region*:*account-id*:*collection/collection-id*

API Operation: DetectFaces
Required Permissions (API Action): rekognition:DetectFaces

Resources: arn:aws:rekognition:*region*:*account-id*:*collection/collection-id*

API Operation: IndexFaces
Required Permissions (API Action): rekognition:IndexFaces

Resources: arn:aws:rekognition:*region*:*account-id*:*collection/collection-id*

API Operation: ListCollections
Required Permissions (API Action): rekognition:ListCollections

Resources: arn:aws:rekognition:*region*:*account-id:**

API Operation: ListFaces
Required Permissions (API Action): rekognition:ListFaces

Resources: arn:aws:rekognition:*region*:*account-id*:*collection/collection-id*

API Operation: SearchFaces

Required Permissions (API Action): rekognition:SearchFaces

Resources: arn:aws:rekognition:*region*:*account-id*:*collection/collection-id*

API Operation: SearchFacesByImage

Required Permissions (API Action): rekognition:SearchFacesByImage

Resources: arn:aws:rekognition:*region*:*account-id*:*collection/collection-id*

Document History for Amazon Rekognition

The following table describes the documentation for this release of Amazon Rekognition.

- **API version: 2016-06-27**
- **Latest documentation update:** November 30, 2016

Change	Description	Date
New service and guide	This is the initial release of the image analysis service, Amazon Rekognition, and the *Amazon Rekognition Developer Guide*.	November 30, 2016

AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the *AWS General Reference*.