Amazon Rekognition

Developer Guide

Amazon Rekognition: Developer Guide

Copyright © 2017 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

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 enables your applications to confirm user identities by comparing their live image with a reference image.

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

- **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.

- **Image moderation** – Amazon Rekognition can detect explicit and suggestive adult content in images. Developers can use the returned metadata to filter inappropriate content based on their business needs. Beyond flagging an image based on the presence of adult content, the API also returns a hierarchical list of labels with confidence scores. These labels indicate specific categories of adult

content, thus allowing granular filtering and management of large volumes of user generated content (UGC). For example, social and dating sites, photo sharing platforms, blogs and forums, apps for children, e-commerce sites, entertainment and online advertising services.

- **Celebrity recognition** – Amazon Rekognition can recognize celebrities within supplied images. Rekognition can recognize thousands of celebrities across a number of categories, such as politics, sports, business, entertainment, and media.

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.

- **Integrate 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. 17)** – In this section you set your account and test the Amazon Rekognition API.
3. **Additional Amazon Rekognition Examples (p. 64)** – 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.
  - Detect celebrities in images.
  - Analyse images for explicit or suggestive adult content.

- **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. None of the input image bytes are stored.

Topics

Non-Storage API Operations

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.
- `DetectModerationLabels` to detect explicit or suggestive adult content in images.
- `RecognizeCelebrities` to recognize celebrities in images.
- `GetCelebrityInfo` to get information about a celebrity.

These are referred to as *non-storage* API operations because when you make the operation call, Amazon Rekognition does not persist any information discovered about the input image. Like all other Amazon Rekognition API operations, no input image bytes are persisted by non-storage API operations.

The following example scenarios show where you might integrate non-storage API operations in your application. These scenarios assume that 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 precise labels, you might filter and choose only the labels with higher confidence (such as 95% or higher). If your application doesn't 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 includes a bounding box, facial landmarks (for example, the 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. 9)

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. 115) 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. 109) 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 a 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":[
   {
      "BoundingBox":{
         "Height":0.18000000715255737,
         "Left":0.5555555820465088,
         "Top":0.33666667342185974,
         "Width":0.23999999463558197
      },
      "Confidence":100.0,
      "Landmarks":[
         {
            "Type":"eyeLeft",
            "X":0.6394737362861633,
            "Y":0.40819624066352844
         },
         {
            "Type":"eyeRight",
            "X":0.7266660928726196,
            "Y":0.41039225459098816
         },
         {
            "Type":"nose",
            "X":0.6912462115287781,
            "Y":0.44240960478782654
         },
         {
            "Type":"mouthLeft",
            "X":0.6306198239326477,
            "Y":0.46700039505958557
         },
         {
            "Type":"mouthRight",
            "X":0.7215608954429626,
            "Y":0.47114261984825134
         }
      ],
      "Pose":{
         "Pitch":4.050806522369385,
         "Roll":0.9950747489929199,
         "Yaw":13.693790435791016
      },
      "Quality":{
         "Brightness":37.60169982910156,
         "Sharpness":80.0
      }
   },
   {
      "BoundingBox":{
         "Height":0.16555555164813995,
         "Left":0.3096296191215515,
         "Top":0.7066666483879089,
         "Width":0.22074073553085327
      },
      "Confidence":99.99998474121094,
      "Landmarks":[
         {
            "Type":"eyeLeft",
            "X":0.3767718970775604,
            "Y":0.7863991856575012
         },
         {
            "Type":"eyeRight",
            "X":0.4517287313938141,
            "Y":0.7715709209442139
         },
         {
```

```
                "Type":"nose",
                "X":0.42001065611839294,
                "Y":0.8192070126533508
             },
             {
                "Type":"mouthLeft",
                "X":0.3915625810623169,
                "Y":0.8374140858650208
             },
             {
                "Type":"mouthRight",
                "X":0.46825936436653137,
                "Y":0.823401689529419
             }
          ],
          "Pose":{
             "Pitch":-16.320178985595703,
             "Roll":-15.097439765930176,
             "Yaw":-5.771541118621826
          },
          "Quality":{
             "Brightness":31.440860748291016,
             "Sharpness":60.000003814697266
          }
       }
   ],
   "OrientationCorrection":"ROTATE_0"
}
```


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.

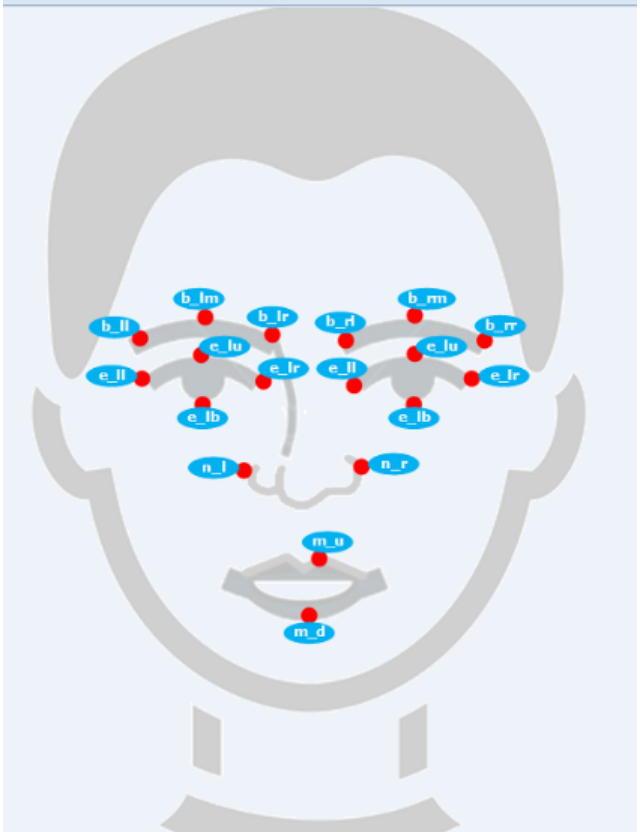
- The `Quality` describes the brightness and the sharpness of the face. You might find this useful to compare faces across images and find the best face.

- 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 (counter-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

To compare a face in the *source* image with each face in the *target* image, use the CompareFaces (p. 92) operation.

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

To specify the minimum level of confidence in the match that you want returned in the response, use `similarityThreshold` in the request. For more information, see CompareFaces (p. 92).

The API returns an array of face matches, source face information, image orientation, and an array of unmatched faces. The following is an example response.


```
{
    "FaceMatches": [{
        "Face": {
            "BoundingBox": {
                "Width": 0.5521978139877319,
                "Top": 0.1203877404332161,
                "Left": 0.23626373708248138,
                "Height": 0.3126954436302185
            },
            "Confidence": 99.98751068115234,
            "Pose": {
                "Yaw": -82.36799621582031,
                "Roll": -62.13221740722656,
                "Pitch": 0.8652129173278809
            },
            "Quality": {
                "Sharpness": 99.99880981445312,
                "Brightness": 54.49755096435547
            },
            "Landmarks": [{
                    "Y": 0.2996366024017334,
                    "X": 0.41685718297958374,
                    "Type": "eyeLeft"
                },
                {
                    "Y": 0.2658946216106415,
                    "X": 0.4414493441581726,
                    "Type": "eyeRight"
                },
                {
                    "Y": 0.3465650677680969,
                    "X": 0.48636093735694885,
                    "Type": "nose"
                },
                {
                    "Y": 0.30935320258140564,
                    "X": 0.6251809000968933,
                    "Type": "mouthLeft"
                },
                {
                    "Y": 0.26942989230155945,
                    "X": 0.6454493403434753,
                    "Type": "mouthRight"
                }
            ]
        },
        "Similarity": 100.0
    }],
    "SourceImageOrientationCorrection": "ROTATE_90",
    "TargetImageOrientationCorrection": "ROTATE_90",
    "UnmatchedFaces": [{
        "BoundingBox": {
            "Width": 0.4890109896659851,
            "Top": 0.6566604375839233,
            "Left": 0.10989011079072952,
            "Height": 0.278298944234848
        },
        "Confidence": 99.99992370605469,
        "Pose": {
            "Yaw": 51.51519012451172,
            "Roll": -110.32493591308594,
            "Pitch": -2.322134017944336
```

```
        },
        "Quality": {
            "Sharpness": 99.99671173095703,
            "Brightness": 57.23163986206055
        },
        "Landmarks": [{
                "Y": 0.8288310766220093,
                "X": 0.3133862614631653,
                "Type": "eyeLeft"
            },
            {
                "Y": 0.7632885575294495,
                "X": 0.28091415762901306,
                "Type": "eyeRight"
            },
            {
                "Y": 0.7417283654212952,
                "X": 0.3631140887737274,
                "Type": "nose"
            },
            {
                "Y": 0.8081989884376526,
                "X": 0.48565614223480225,
                "Type": "mouthLeft"
            },
            {
                "Y": 0.7548204660415649,
                "X": 0.46090251207351685,
                "Type": "mouthRight"
            }
        ]
    }],
    "SourceImageFace": {
        "BoundingBox": {
            "Width": 0.5521978139877319,
            "Top": 0.1203877404332161,
            "Left": 0.23626373708248138,
            "Height": 0.3126954436302185
        },
        "Confidence": 99.98751068115234
    }
}
```


In the response, note the following:

- **Face match information** – The example shows that one face match was found in the target image. For that face match, it provides a bounding box and a confidence value (the level of confidence that Amazon Rekognition has that the bounding box contains a face). The `similarity` score of 99.99 indicates how similar the faces are. The face match information also includes an array of landmark locations.

  If multiple faces match, the `faceMatches` array includes all of the face matches.
- **Source face information** – The response includes information about the face from the source image that was used for comparison, including the bounding box and confidence value.
- **Image Orientation** – The response includes information about the orientation of the source and target images. Amazon Rekognition needs this to display the images and retrieve the correct location of the matched face in the target image.
- **Unmatched face match information** – The example shows one face that Amazon Rekognition found in the target image that didn't match the face analysed in the source image. For that face, it provides a bounding box and a confidence value, indicating the level of confidence that Amazon Rekognition has that the bounding box contains a face. The face information also includes an array of landmark locations.

If Amazon Rekognition finds mutiple faces that do not match, the `UnmatchedFaces` array includes all of the faces that didn't match.

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

Amazon Rekognition supports the IndexFaces (p. 129) 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 might use a 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).

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. 100)
- DeleteCollection (p. 103)
- ListCollections (p. 137)

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. 14).

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. 12). 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**
> `DetectFaces` returns the same information, 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. 12) 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. 157))** – 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. 152))** – Given a face ID (each face stored in the face collection has a face ID), `SearchFaces` searches the specified face collection for 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 operations return faces for which 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"
            }
        }
    ]
}
```


The `CompareFaces` operation and the two search faces 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. 9).
- `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. 17)
2. Create an IAM User (p. 18)

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**.

Note
This might be unavailable in your browser if you previously signed into the AWS Management Console. In that case, choose **Sign In to the Console**, and then choose **Create a new 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:

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

Next Step

Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 18)

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 the API (p. 29), 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 the API (p. 29)

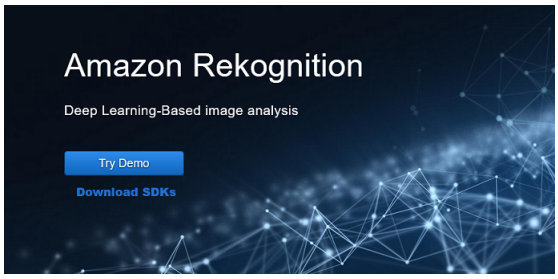
Step 3: Getting Started Using the Amazon Rekognition Console

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 the API (p. 29).

This section also shows you how to see aggregated Amazon CloudWatch metrics for Rekognition by using the Rekognition console.

Topics

- Exercise 1: Detect Objects and Scenes in an Image (Console) (p. 20)
- Exercise 2: Analyze Faces in an Image (Console) (p. 24)
- Exercise 3: Compare Faces in Images (Console) (p. 27)
- Exercise 4: See Aggregated Metrics (Console) (p. 29)

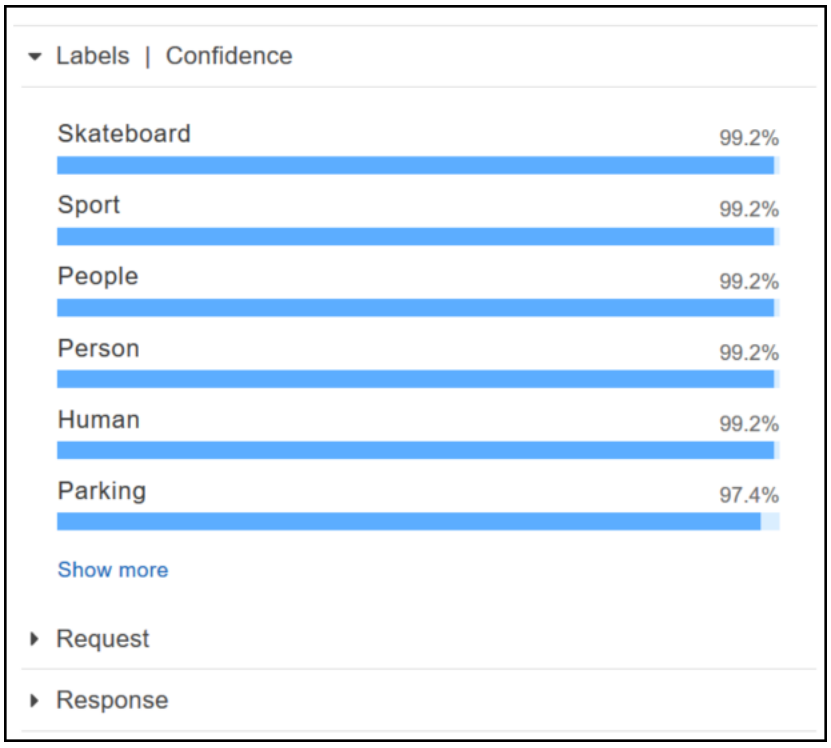


Exercise 1: Detect Objects and Scenes in an Image (Console)

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

```
{
   "contentString":{
      "Attributes":[
         "ALL"
      ],
      "Image":{
         "S3Object":{
            "Bucket":"console-sample-images",
            "Name":"skateboard.jpg"
         }
      }
   }
}
```

Response

```
{
   "Labels":[
      {
         "Confidence":99.25359344482422,
         "Name":"Skateboard"
      },
      {
         "Confidence":99.25359344482422,
         "Name":"Sport"
      },
      {
         "Confidence":99.24723052978516,
         "Name":"People"
```

```
    },
    {
       "Confidence":99.24723052978516,
       "Name":"Person"
    },
    {
       "Confidence":99.23908233642578,
       "Name":"Human"
    },
    {
       "Confidence":97.42484283447266,
       "Name":"Parking"
    },
    {
       "Confidence":97.42484283447266,
       "Name":"Parking Lot"
    },
    {
       "Confidence":91.53300476074219,
       "Name":"Automobile"
    },
    {
       "Confidence":91.53300476074219,
       "Name":"Car"
    },
    {
       "Confidence":91.53300476074219,
       "Name":"Vehicle"
    },
    {
       "Confidence":76.85114288330078,
       "Name":"Intersection"
    },
    {
       "Confidence":76.85114288330078,
       "Name":"Road"
    },
    {
       "Confidence":76.21503448486328,
       "Name":"Boardwalk"
    },
    {
       "Confidence":76.21503448486328,
       "Name":"Path"
    },
    {
       "Confidence":76.21503448486328,
       "Name":"Pavement"
    },
    {
       "Confidence":76.21503448486328,
       "Name":"Sidewalk"
    },
    {
       "Confidence":76.21503448486328,
       "Name":"Walkway"
    },
    {
       "Confidence":66.71541595458984,
       "Name":"Building"
    },
    {
       "Confidence":62.04711151123047,
       "Name":"Coupe"
    },
    {
```

```
      "Confidence":62.04711151123047,
      "Name":"Sports Car"
   },
   {
      "Confidence":61.98909378051758,
      "Name":"City"
   },
   {
      "Confidence":61.98909378051758,
      "Name":"Downtown"
   },
   {
      "Confidence":61.98909378051758,
      "Name":"Urban"
   },
   {
      "Confidence":60.978023529052734,
      "Name":"Neighborhood"
   },
   {
      "Confidence":60.978023529052734,
      "Name":"Town"
   },
   {
      "Confidence":59.22066116333008,
      "Name":"Sedan"
   },
   {
      "Confidence":56.48063278198242,
      "Name":"Street"
   },
   {
      "Confidence":54.235477447509766,
      "Name":"Housing"
   },
   {
      "Confidence":53.85226058959961,
      "Name":"Metropolis"
   },
   {
      "Confidence":52.001792907714844,
      "Name":"Office Building"
   },
   {
      "Confidence":51.325313568115234,
      "Name":"Suv"
   },
   {
      "Confidence":51.26075744628906,
      "Name":"Apartment Building"
   },
   {
      "Confidence":51.26075744628906,
      "Name":"High Rise"
   },
   {
      "Confidence":50.68067932128906,
      "Name":"Pedestrian"
   },
   {
      "Confidence":50.59548568725586,
      "Name":"Freeway"
   },
   {
      "Confidence":50.568580627441406,
      "Name":"Bumper"
```

```
        }
    ]
}
```

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

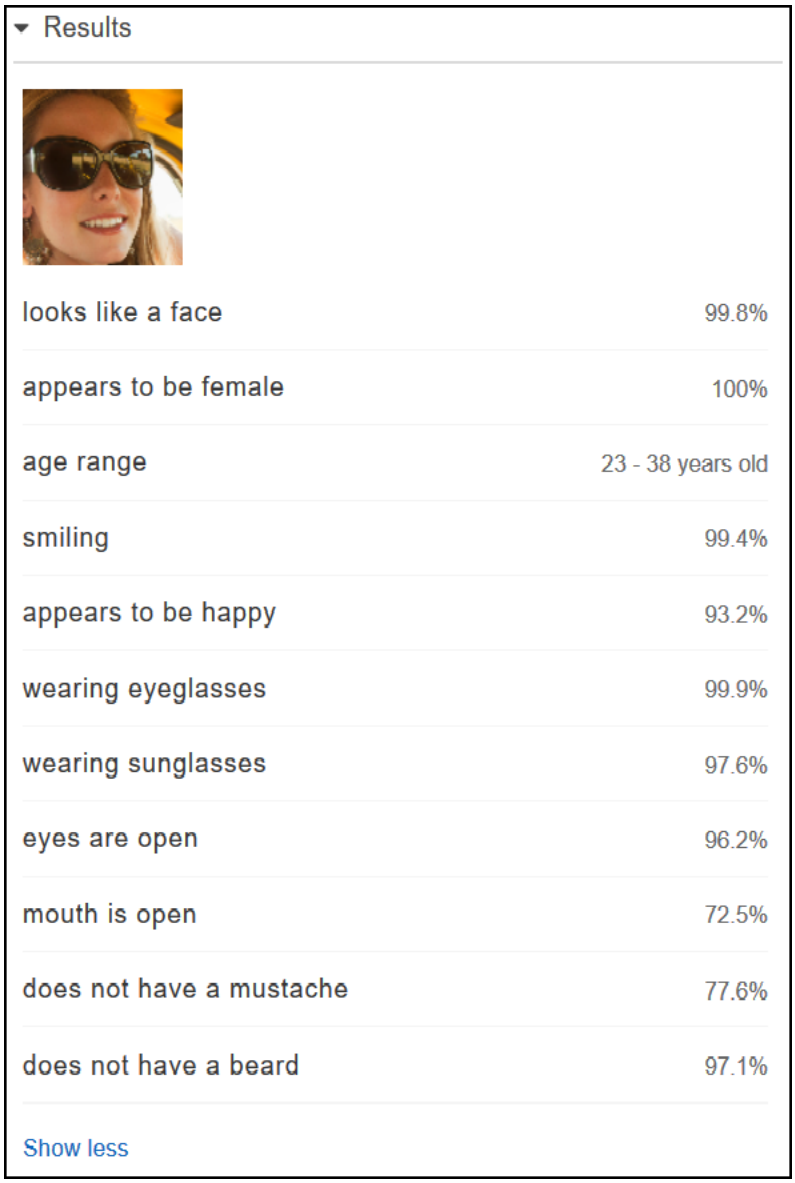
1. Open the Amazon Rekognition console.
2. Choose **Object and scene detection**.
3. Do one of the following:
   - Upload an image – Choose **Upload**, go to the location where you stored your image, and then select the image.
   - Use a URL – Type the URL in the text box, and then choose **Go**.
4. 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)

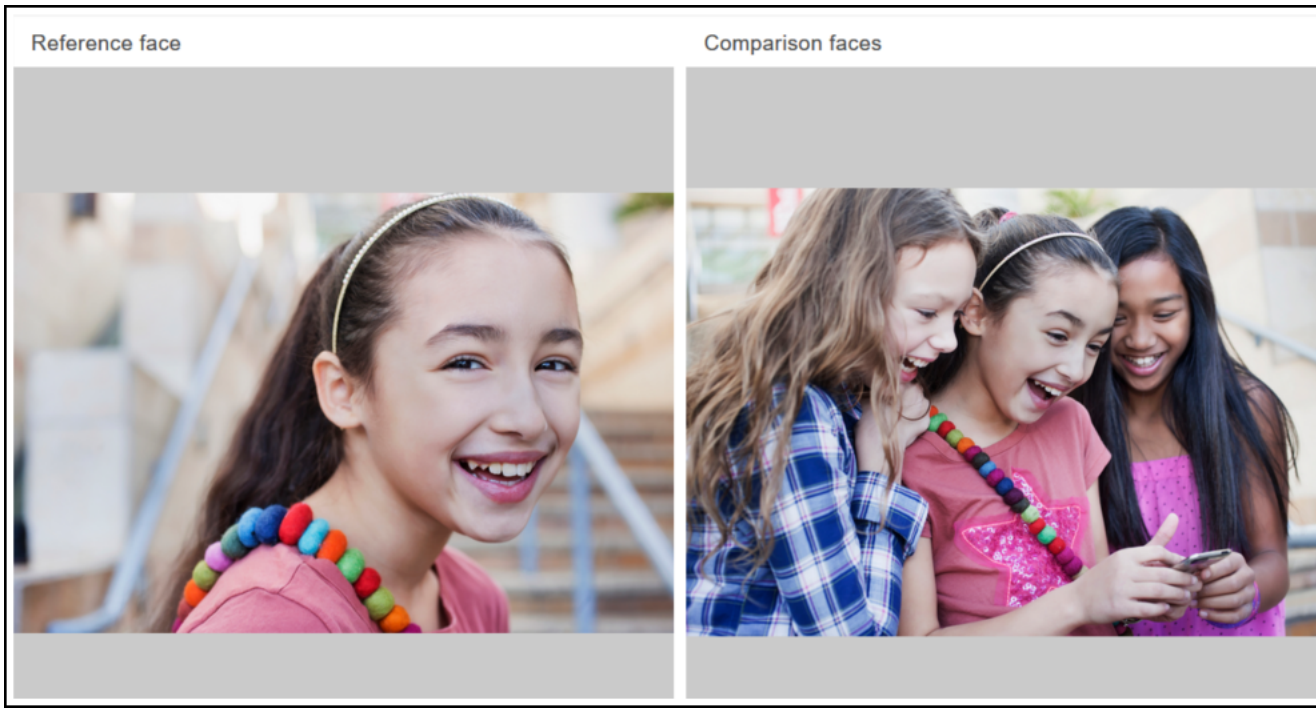
This section shows you how to use the Amazon Rekognition console to compare faces within a set of images with multiple faces in them. When you specify a **Reference face** (source) and a **Comparison faces** (target) image, Rekognition compares the largest face in the source image (that is, the reference 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.

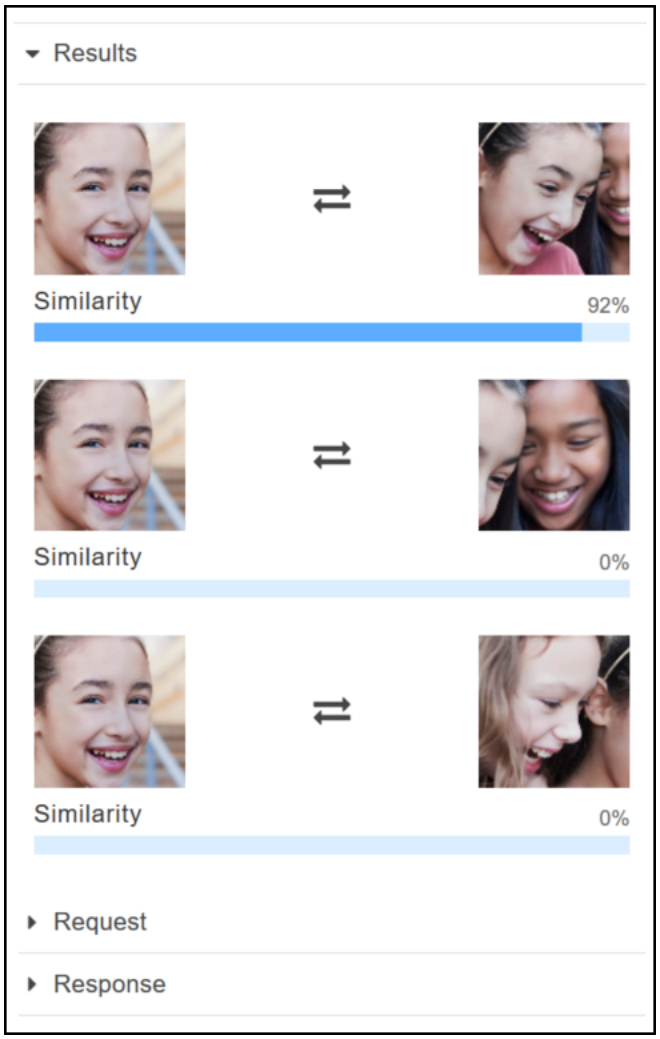
If the source image contains multiple faces, the service detects the largest face in the source image and uses it to compare with each face detected in the target image.

For more information, see Comparing Faces (p. 9).

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.

Exercise 4: See Aggregated Metrics (Console)

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

The following table lists the graphs displayed in the Rekognition metrics pane and the corresponding Rekognition metric. For more information, see CloudWatch Metrics for Rekognition (p. 62).

Graph	Aggregated Metric
Successful calls	SuccessfulRequestCount
Client errors	UserErrorCount
Server errors	ServerErrorCount
Throttled	ThrottledCount
Detected labels	DetectedLabelCount
Detected faces	DetectedFaceCount

Each graph shows aggregated metric data collected over a specified period of time. A total count of aggregated metric data for the time period is also displayed. To see metrics for individual API calls, choose the link beneath each graph.

To allow users access to the Rekognition metrics pane, ensure that the user has appropriate CloudWatch and Rekognition permissions. For example, a user with `AmazonRekognitionReadOnlyAccess` and `CloudWatchReadOnlyAccess` managed policy permissions can see the metrics pane. If a user does not have the required permissions, when the user opens the metrics pane, no graphs appear. For more information, see Authentication and Access Control for Amazon Rekognition (p. 39).

For more information about monitoring Rekognition with CloudWatch see Monitoring (p. 59).

To see aggregated metrics (console)

1. Open the Amazon Rekognition console at https://console.aws.amazon.com/rekognition/.
2. In the navigation pane, choose **Metrics**.
3. In the dropdown, select the period of time you want metrics for.
4. To update the graphs, choose the **Refresh** button.
5. To see detailed CloudWatch metrics for a specific aggregated metric, choose **See details on CloudWatch** beneath the metric graph.

Step 4: Getting Started Using the API

In this section you use the Amazon Rekognition API operations to detect labels and faces in an image. You also explore the CompareFaces (p. 92) API operation. These are the non-storage API operations where Amazon Rekognition doesn't persist any information discovered by the operation. Amazon

Rekognition only detects labels and faces, and it returns information in response. For more information, see Amazon Rekognition: How It Works (p. 3). Like all other Amazon Rekognition API operations, no input image bytes are persisted by non-storage API operations.

Topics

Using the AWS SDK or HTTP to Call Amazon Rekognition API Operations

You can call Amazon Rekognition API operations using either the AWS SDK or directly by using HTTP. Unless you have a good reason not to, you should always use the AWS SDK. The Java examples in this section use the AWS SDK. A Java project file is not provided, but you can use the AWS Toolkit for Eclipse to develop AWS applications using Java.

The API Reference (p. 90) in this guide covers calling Amazon Rekognition operations using HTTP. For Java reference information, see AWS SDK for Java.

The Amazon Rekognition service endpoints you can use are documented at AWS Regions and Endpoints.

When calling Amazon Rekognition with HTTP, use POST HTTP operations.

Formatting the AWS CLI Examples

The AWS CLI examples are formatted for the Linux operating system. To use the samples with Microsoft Windows, you will need to change the JSON formatting of the `--image` parameter and change the line breaks from backslashes (\) to carets(^). For more information about JSON formatting, see Specifying Parameter Values for the AWS Command Line Interface. The following is an example AWS CLI command formatted for Microsoft Windows.

```
aws rekognition detect-labels ^
  --image "{\"S3Object\":{\"Bucket\":\"photo-collection\",\"Name\":\"photo.jpg\"}}" ^
  --region us-west-2
```

You can also provide a shorthand version of the JSON that works on both Microsoft Windows and Linux.

```
aws rekognition detect-labels --image "S3Object={Bucket=photo-collection,Name=photo.jpg}"
 --region us-west-2
```

For more information, see Using Shorthand Syntax with the AWS Command Line Interface.

Working with Images

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

You pass image bytes to an Amazon Rekognition operation as part of the call or you reference an existing S3 object. If you use HTTP and pass the image bytes as part of an Amazon Rekognition

operation, the image bytes must be a base64-encoded string. If you use the AWS SDK and pass image bytes as part of the API operation call, the need to base64-encode the image bytes depends on the language you use. For more information, see Example 4: Supplying Image Bytes to Amazon Rekognition Operations (p. 79).

If you use the AWS CLI to call Amazon Rekognition operations, passing image bytes as part of the call is not supported. You must first upload the image to an Amazon S3 bucket and then call the operation referencing the uploaded image.

To ensure the lowest possible latency, the region for the S3 bucket containing your images must match the region you use for Amazon Rekognition API operations.

Recommendations for Facial Recognition Input Images

Whilst Amazon Rekognition works across a variety of image conditions and sizes, we recommend the following guidelines when choosing reference photos for facial recognition:

- Have a front-facing face.
- Have flat lighting on the face, as opposed to varying shades such as shadows.
- Have sufficient contrast with the background. A high-contrast monochrome background works well.
- Be sufficiently large. Amazon Rekognition can detect faces as small as 40x40 pixels in an HD resolution image (1980x1080). Higher resolution images will need a larger minimum face size.
- Be bright and sharp. You can use DetectFaces (p. 109) to determine the brightness and sharpness of a face.
- Avoid occlusions such as head-bands or masks.

Correcting Image Orientation

In several Rekognition API operations, the orientation of an analyzed image is returned. For example, the DetectFaces (p. 109) API operation returns orientation information about the source image in the `OrientationCorrection` field. Knowing image orientation is important as it allows you to reorient images for display. Rekognition API operations that analyze faces also return bounding boxes for the location of faces within an image. You can use bounding boxes to display a box around a face on an image. The bounding box coordinates returned are affected by image orientation and you may need to translate bounding box coordinates to correctly display a box around a face. For more information, see Example 5: Getting Image Orientation and Bounding Box Coordinates (p. 83).

Exercise 1: Detect Labels in an Image (API)

In this exercise you use the DetectLabels (p. 115) API operation to detect objects, concepts, and scenes 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 a JPEG image to your Amazon S3 bucket and specify the object key name.

The following examples show how you can use the operation with the AWS CLI, the AWS SDK for Java and the AWS SDK for Python (Boto).

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 the following example code to call 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. 18). 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.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
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.Label;
import com.amazonaws.services.rekognition.model.S3Object;
import java.util.List;

public class DetectLabelsExample {

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

      String photo = "photo.jpg";
      String bucket = "S3bucket";

      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);
      }

      AmazonRekognition rekognitionClient = AmazonRekognitionClientBuilder
              .standard()
              .withRegion(Regions.US_WEST_2)
              .withCredentials(new AWSStaticCredentialsProvider(credentials))
              .build();

      DetectLabelsRequest request = new DetectLabelsRequest()
        .withImage(new Image()
        .withS3Object(new S3Object()
        .withName(photo).withBucket(bucket)))
        .withMaxLabels(10)
        .withMinConfidence(75F);

      try {
         DetectLabelsResult result = rekognitionClient.detectLabels(request);
         List <Label> labels = result.getLabels();
```

```
         System.out.println("Detected labels for " + photo);
         for (Label label: labels) {
            System.out.println(label.getName() + ": " +
 label.getConfidence().toString());
         }
      } catch(AmazonRekognitionException e) {
         e.printStackTrace();
      }
   }
}
```

- Using AWS SDK for Python (Boto).

```
import boto3

if __name__ == "__main__":
    fileName='input.jpg'
    bucket='bucketname'

    client=boto3.client('rekognition')

    response = client.detect_labels(Image={'S3Object':
{'Bucket':bucket,'Name':fileName}},MinConfidence=75)

    print('Detected labels for ' + fileName)
    for label in response['Labels']:
        print (label['Name'] + ' : ' + str(label['Confidence']))
```

Next Exercise

Exercise 2: Detect Faces (API) (p. 33)

Exercise 2: Detect Faces (API)

In this exercise you use the DetectFaces (p. 109) 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.

The following examples show how you can use the operation with the AWS CLI, the AWS SDK for Java and the AWS SDK for Python (Boto).

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. Use the following example code to call the `DetectFaces` API operation.

   - Using the 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. This example displays the estimated age range for detected faces and lists the JSON for all detected facial attributes.

```
import java.util.List;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AgeRange;
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.FaceDetail;
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 {

      String photo = "photo.jpg";
      String bucket = "S3bucket";

      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);
      }

      AmazonRekognition rekognitionClient = AmazonRekognitionClientBuilder
         .standard()
         .withRegion(Regions.US_WEST_2)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      DetectFacesRequest request = new DetectFacesRequest()
         .withImage(new Image()
            .withS3Object(new S3Object()
               .withName(photo)
               .withBucket(bucket)))
         .withAttributes(Attribute.ALL);
      // Replace Attribute.ALL with Attribute.DEFAULT to get default values.

      try {
         DetectFacesResult result = rekognitionClient.detectFaces(request);
         List < FaceDetail > faceDetails = result.getFaceDetails();

         for (FaceDetail face: faceDetails) {
            if (request.getAttributes().contains("ALL")) {
               AgeRange ageRange = face.getAgeRange();
               System.out.println("The detected face is estimated to be between "
                  + ageRange.getLow().toString() + " and " +
ageRange.getHigh().toString()
```

```
                + " years old.");
             System.out.println("Here's the complete set of attributes:");
          } else { // non-default attributes have null values.
             System.out.println("Here's the default set of attributes:");
          }

          ObjectMapper objectMapper = new ObjectMapper();

 System.out.println(objectMapper.writerWithDefaultPrettyPrinter().writeValueAsString(face));
          }

       } catch (AmazonRekognitionException e) {
          e.printStackTrace();
       }

    }

}
```

- Using AWS SDK for Python (Boto). This example displays the estimated age range for detected faces and lists the JSON for all detected facial attributes.

```
import boto3
import json

if __name__ == "__main__":
    fileName='input.jpg'
    bucket='bucket'
    client=boto3.client('rekognition')

    response = client.detect_faces(Image={'S3Object':
{'Bucket':bucket,'Name':fileName}},Attributes=['ALL'])

    print('Detected faces for ' + fileName)
    for faceDetail in response['FaceDetails']:
        print('The detected face is between ' + str(faceDetail['AgeRange']['Low'])
              + ' and ' + str(faceDetail['AgeRange']['High']) + ' years old')
        print('Here are the other attributes:')
        print(json.dumps(faceDetail, indent=4, sort_keys=True))
```

Next Exercise

Exercise 3: Compare Faces (API) (p. 35)

Exercise 3: Compare Faces (API)

In this exercise you use the CompareFaces (p. 92) 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, source face information, source and target image orientation, and an array of unmatched faces. For each matching face in the target image, the response provides a similarity score (how similar the face is to the source face) and face metadata such as the bounding box of the matching face and an array of facial landmarks. The array of unmatched faces includes face metadata.

You can provide the source and target images as an image byte array (Base64-encoded image bytes) or specify S3 objects. In the AWS CLI exercise, you upload two JPEG images to your Amazon S3 bucket and specify the object key name. In the Java exercise, you load two files from the local file system and input them as image byte arrays.

The following examples show how you can use the operation with the AWS CLI, the AWS SDK for Java and the AWS SDK for Python (Boto).

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. Use the following example code to call 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. This example compares two images loaded from the local file system.

   ```
   package com.amazonaws.samples;

   import java.io.File;
   import java.io.FileInputStream;
   import java.io.InputStream;
   import java.nio.ByteBuffer;
   import java.util.List;

   import com.amazonaws.AmazonClientException;
   import com.amazonaws.auth.AWSCredentials;
   import com.amazonaws.auth.AWSStaticCredentialsProvider;
   import com.amazonaws.auth.profile.ProfileCredentialsProvider;
   import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
   import com.amazonaws.services.rekognition.AmazonRekognition;
   import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
   import com.amazonaws.services.rekognition.model.Image;
   import com.amazonaws.util.IOUtils;
   import com.amazonaws.services.rekognition.model.BoundingBox;
   import com.amazonaws.services.rekognition.model.CompareFacesMatch;
   import com.amazonaws.services.rekognition.model.CompareFacesRequest;
   import com.amazonaws.services.rekognition.model.CompareFacesResult;
   import com.amazonaws.services.rekognition.model.ComparedFace;


   public class CompareFacesExample {

      public static void main(String[] args) throws Exception{
          Float similarityThreshold = 70F;
          String sourceImage = "source.jpg";
          String targetImage = "target.jpg";
          ByteBuffer sourceImageBytes=null;
          ByteBuffer targetImageBytes=null;


          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);
      }

      EndpointConfiguration endpoint=new EndpointConfiguration("endpoint",
             "us-east-1");

       AmazonRekognition rekognitionClient = AmazonRekognitionClientBuilder
              .standard()
              .withEndpointConfiguration(endpoint)
              .withCredentials(new AWSStaticCredentialsProvider(credentials))
              .build();


      //Load source and target images and create input parameters
      try (InputStream inputStream = new FileInputStream(new File(sourceImage))) {
         sourceImageBytes = ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
      }
      catch(Exception e)
      {
          System.out.println("Failed to load source image " + sourceImage);
          System.exit(1);
      }
      try (InputStream inputStream = new FileInputStream(new File(targetImage))) {
          targetImageBytes = ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
      }
      catch(Exception e)
      {
          System.out.println("Failed to load target images: " + targetImage);
          System.exit(1);
      }

      Image source=new Image()
        .withBytes(sourceImageBytes);
      Image target=new Image()
        .withBytes(targetImageBytes);

      CompareFacesRequest request = new CompareFacesRequest()
              .withSourceImage(source)
              .withTargetImage(target)
              .withSimilarityThreshold(similarityThreshold);

      // Call operation
      CompareFacesResult compareFacesResult=rekognitionClient.compareFaces(request);


      // Display results
      List <CompareFacesMatch> faceDetails = compareFacesResult.getFaceMatches();
      for (CompareFacesMatch match: faceDetails){
       ComparedFace face= match.getFace();
       BoundingBox position = face.getBoundingBox();
       System.out.println("Face at " + position.getLeft().toString()
          + " " + position.getTop()
          + " matches with " + face.getConfidence().toString()
          + "% confidence.");

      }
      List<ComparedFace> uncompared = compareFacesResult.getUnmatchedFaces();

      System.out.println("There were " + uncompared.size()
        + " that did not match");
```

```
        System.out.println("Source image rotation: " +
 compareFacesResult.getSourceImageOrientationCorrection());
        System.out.println("target image rotation: " +
 compareFacesResult.getTargetImageOrientationCorrection());
    }
}
```

- Using AWS SDK for Python (Boto).

```
import boto3

if __name__ == "__main__":

    bucket='bucket-name'
    sourceFile='source.jpg'
    targetFile='target.jpg'

    client=boto3.client('rekognition')

    response=client.compare_faces(SimilarityThreshold=70,
                                  SourceImage={'S3Object':
{'Bucket':bucket,'Name':sourceFile}},
                                  TargetImage={'S3Object':
{'Bucket':bucket,'Name':targetFile}})

    for faceMatch in response['FaceMatches']:
        position = faceMatch['Face']['BoundingBox']
        confidence = str(faceMatch['Face']['Confidence'])
        print('The face at ' +
               str(position['Left']) + ' ' +
               str(position['Top']) +
               ' matches with ' + confidence + '% confidence')
```

What's Next?

You can explore additional Additional Amazon Rekognition Examples (p. 64) 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.

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 first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.
- **IAM user** – An IAM user is 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 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 an IAM identity that 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 existing 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*.

  - **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 from that 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** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are 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. 47).

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. 44). 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"
            ]
        }
    ]
}
```

To use an S3 bucket with versioning enabled, add the `s3:GetObjectVersion` action, as shown in the following example.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "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. 41) and Amazon Rekognition Amazon Rekognition API Permissions: Actions, Permissions, and Resources Reference (p. 47).

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. 41).

- **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. 47).

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. 40).

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. 47).

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.
- **AmazonRekognitionReadWriteAccess** – 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. 45).

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 (p. 46)
- Example 2: Allow a User Full Access to Resources (p. 46)

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. 40) 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*

Recognizing Celebrities

Amazon Rekognition can recognize thousands of celebrities in a wide range of categories, such as entertainment and media, sports, business, and politics.

To recognize celebrities within images and get additional information about recognized celebrities, use the RecognizeCelebrities (p. 146) non-storage API operation. For example, in social media or news and entertainment industries where information gathering can be time critical, you can use the `RecognizeCelebrities` operation to identify as many as 15 celebrities in an image and return links to celebrity web pages, if they are available. Amazon Rekognition doesn't remember which image it detected a celebrity in. Your application must store this information.

If you haven't stored the additional information for a celebrity returned by `RecognizeCelebrities` and you want to avoid reanalyzing an image to get it, use GetCelebrityInfo (p. 126). To call `GetCelebrityInfo`, you need the unique identifier that Rekognition assigns to each celebrity. The identifier is returned as part of the `RecognizeCelebrities` response for each celebrity recognized in an image.

If you have a large collection of images to process for celebrity recognition, consider using AWS Batch to process calls to `RecognizeCelebrities` in batches in the background. When you add a new image to your collection, you can use an AWS Lambda function to recognize celebrities by calling `RecognizeCelebrities` as the image is uploaded into an S3 bucket.

Calling RecognizeCelebrities

You provide an input image to `RecognizeCelebrities` either as image bytes or by referencing an image stored in an S3 bucket. The image must be either a .png or .jpg formatted file. For information about input image recommendations, see Working with Images (p. 30). The input image quality (brightness and sharpness) is returned by `RecognizeCelebrities`.

`RecognizeCelebrities` returns an array of recognized celebrities and an array of unrecognized faces, as shown in the following example:

```
{
    "CelebrityFaces": [{
        "Face": {
            "BoundingBox": {
                "Height": 0.617123007774353,
                "Left": 0.15641026198863983,
                "Top": 0.10864841192960739,
                "Width": 0.3641025722026825
```

```
            },
            "Confidence": 99.99589538574219,
            "Landmarks": [{
                "Type": "eyeLeft",
                "X": 0.2837241291999817,
                "Y": 0.3637104034423828
            }, {
                "Type": "eyeRight",
                "X": 0.4091649055480957,
                "Y": 0.37378931045532227
            }, {
                "Type": "nose",
                "X": 0.35267341136932373,
                "Y": 0.49657556414604187
            }, {
                "Type": "mouthLeft",
                "X": 0.2786353826522827,
                "Y": 0.5455248355865479
            }, {
                "Type": "mouthRight",
                "X": 0.39566439390182495,
                "Y": 0.5597742199897766
            }],
            "Pose": {
                "Pitch": -7.749263763427734,
                "Roll": 2.004552125930786,
                "Yaw": 9.012002944946289
            },
            "Quality": {
                "Brightness": 32.69192123413086,
                "Sharpness": 99.9305191040039
            }
        },
        "Id": "3Ir0du6",
        "MatchConfidence": 98.0,
        "Name": "Jeff Bezos",
        "Urls": ["www.imdb.com/name/nm1757263"]
    }],
    "OrientationCorrection": "ROTATE_0",
    "UnrecognizedFaces": [{
        "BoundingBox": {
            "Height": 0.5345501899719238,
            "Left": 0.48461538553237915,
            "Top": 0.16949152946472168,
            "Width": 0.3153846263885498
        },
        "Confidence": 99.92860412597656,
        "Landmarks": [{
            "Type": "eyeLeft",
            "X": 0.5863404870033264,
            "Y": 0.36940744519233704
        }, {
            "Type": "eyeRight",
            "X": 0.6999204754829407,
            "Y": 0.3769848346710205
        }, {
            "Type": "nose",
            "X": 0.6349524259567261,
            "Y": 0.4804527163505554
        }, {
            "Type": "mouthLeft",
            "X": 0.5872702598571777,
            "Y": 0.5535582304000854
        }, {
            "Type": "mouthRight",
            "X": 0.6952020525932312,
```

```
            "Y": 0.5600858926773071
        }],
        "Pose": {
            "Pitch": -7.386096477508545,
            "Roll": 2.304218292236328,
            "Yaw": -6.175624370574951
        },
        "Quality": {
            "Brightness": 37.16635513305664,
            "Sharpness": 99.884521484375
        }
    }]
}
```

The response includes the following:

- **Recognized celebrities** – `CelebrityFaces` is an array of detected celebrities. Each Celebrity (p. 168) object in the array contains the celebrity name and a list of URLs pointing to related content; for example, the celebrity's IMDB link. Amazon Rekognition returns an ComparedFace (p. 169) object that your application can use to determine where the celebrity's face is on the image and a unique identifier for the celebrity. Use the unique identifier to retrieve celebrity information later with the GetCelebrityInfo (p. 126) API operation.
- **Unrecognized faces** – `UnrecognizedFaces` is an array of faces that didn't match any known celebrities. Each ComparedFace (p. 169) object in the array contains a bounding box (as well as other information) that you can use to locate the face in the image.
- **Image orientation** – Image orientation information is provided to allow you to correctly display of the image.

Recognizing Celebrities in an Image

In these procedures, you use the RecognizeCelebrities (p. 146) operation to recognize celebrities in an image that you supply. You can provide the input image as an image byte array (base64-encoded image bytes) or as an S3 object, using either the AWS command line interface (AWS CLI) or the AWS SDK for Java. In the AWS CLI procedure, you upload an image in .jpg or .png format to an S3 bucket. In the AWS SDK for Java procedure, you use an image loaded from your local file system.

Prerequisites

To run these procedures, you need to have the AWS CLI and AWS SDK for Java installed. For more information, see Getting Started with Amazon Rekognition (p. 17). The AWS account you use must have access permissions to the Amazon Rekognition API. For more information, see Amazon Rekognition API Permissions: Actions, Permissions, and Resources Reference (p. 47). You also need an image file that contains one or more celebrity faces.

To recognize celebrities in an image (AWS CLI)

1. Upload an image that contains one or more celebrity faces to your S3 bucket.

   For instructions, see Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service Console User Guide*.
2. On the command line, type the following command. Replace `bucketname` and `input.jpg` with the S3 bucket name and image name that you used in step 1.

```
aws rekognition recognize-celebrities \
  --image "S3Object={Bucket=bucketname,Name=input.jpg}"
```

3. To run the command, choose **Enter**. The JSON output for the `RecognizeCelebrities` API operation is displayed.
4. Record the value of one of the celebrity IDs that are displayed. You'll need it in Getting Information about a Celebrity (p. 53).

To recognize celebrities in an image (API)

1. To recognize celebrities in an image, use the following AWS SDK for Java example code. Replace `input.jpg` with the name and location of a locally stored .jpg image file that contains one or more celebrity faces.

```
  import java.util.List;
  import com.amazonaws.AmazonClientException;
  import com.amazonaws.auth.AWSCredentials;
  import com.amazonaws.auth.AWSStaticCredentialsProvider;
  import com.amazonaws.auth.profile.ProfileCredentialsProvider;
  import com.amazonaws.regions.Regions;
  import com.amazonaws.services.rekognition.AmazonRekognition;
  import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
  import com.amazonaws.services.rekognition.model.Image;
  import com.amazonaws.services.rekognition.model.BoundingBox;
  import com.amazonaws.services.rekognition.model.Celebrity;
  import com.amazonaws.services.rekognition.model.RecognizeCelebritiesRequest;
  import com.amazonaws.services.rekognition.model.RecognizeCelebritiesResult;
  import java.io.File;
  import java.io.FileInputStream;
  import java.io.InputStream;
  import java.nio.ByteBuffer;
  import com.amazonaws.util.IOUtils;

  public class Celebs {

     public static void main(String[] args) {
        String photo = "input.jpg";
        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);
        }

        ByteBuffer imageBytes=null;
        try (InputStream inputStream = new FileInputStream(new File(photo))) {
           imageBytes = ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
        }
        catch(Exception e)
        {
            System.out.println("Failed to load file " + photo);
            System.exit(1);
        }

        AmazonRekognition amazonRekognition = AmazonRekognitionClientBuilder
           .standard()
           .withRegion(Regions.US_WEST_2)
           .withCredentials(new AWSStaticCredentialsProvider(credentials))
           .build();
```

```
        RecognizeCelebritiesRequest request = new RecognizeCelebritiesRequest()
           .withImage(new Image()
           .withBytes(imageBytes));

        System.out.println("Looking for celebrities in image " + photo + "\n");

        RecognizeCelebritiesResult
result=amazonRekognition.recognizeCelebrities(request);

        //Display recognized celebrity information
        List<Celebrity> celebs=result.getCelebrityFaces();
        System.out.println(celebs.size() + " celebrity(s) were recognized.\n");

        for (Celebrity celebrity: celebs) {
            System.out.println("Celebrity recognized: " + celebrity.getName());
            System.out.println("Celebrity ID: " + celebrity.getId());
            BoundingBox boundingBox=celebrity.getFace().getBoundingBox();
            System.out.println("position: " +
               boundingBox.getLeft().toString() + " " +
               boundingBox.getTop().toString());
            System.out.println("Further information (if available):");
            for (String url: celebrity.getUrls()){
               System.out.println(url);
            }
            System.out.println();
         }
         System.out.println(result.getUnrecognizedFaces().size() + " face(s) were
unrecognized.");
     }
  }
```

2. Run the sample code. The output lists the names of celebrities that were recognized, the celebrity IDs, the location of the celebrities' faces on the image, and links to further information. The output also tells how many faces weren't recognized.
3. Record one of the celebrity IDs. You'll need it in Getting Information about a Celebrity (p. 53).

Getting Information about a Celebrity

In these procedures, you get celebrity information by using the GetCelebrityInfo (p. 126) API operation. The celebrity is identified by using the celebrity ID returned from a previous call to `RecognizeCelebrities`.

Prerequisites

To run these procedures, you need to have the AWS CLI and AWS SDK for Java installed. For more information, see Getting Started with Amazon Rekognition (p. 17). The AWS account you use must have access permissions to the Amazon Rekognition API. For more information, see Amazon Rekognition API Permissions: Actions, Permissions, and Resources Reference (p. 47).

These procedures also require the celebrity ID for a celebrity that Rekognition knows. Use the celebrity ID you note in Recognizing Celebrities in an Image (p. 51).

To get celebrity information (AWS CLI)

1. On the command line, type the following command. Replace `ID` with one of the celebrity IDs displayed in Recognizing Celebrities in an Image (p. 51).

```
aws rekognition get-celebrity-info --id ID
```

2. To run the command, choose **Enter**. The JSON output for the `GetCelebrityInfo` API operation is displayed.

To get celebrity information (SDK)

1. Use the following AWS SDK for Java example code to get information about a celebrity that Amazon Rekognition recognized in an image. Replace `ID` with one of the celebrity IDs displayed in Recognizing Celebrities in an Image (p. 51).

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.GetCelebrityInfoRequest;
import com.amazonaws.services.rekognition.model.GetCelebrityInfoResult;

public class CelebrityInfo {

   public static void main(String[] args) {
      String id = "ID";

      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);
      }

      AmazonRekognition amazonRekognition = AmazonRekognitionClientBuilder
              .standard()
              .withRegion(Regions.US_WEST_2)
              .withCredentials(new AWSStaticCredentialsProvider(credentials))
              .build();


      GetCelebrityInfoRequest request = new GetCelebrityInfoRequest()
         .withId(id);

      System.out.println("Getting information for celebrity: " + id);

      GetCelebrityInfoResult result=amazonRekognition.getCelebrityInfo(request);

      //Display celebrity information
      System.out.println("celebrity name: " + result.getName());
      System.out.println("Further information (if available):");
      for (String url: result.getUrls()){
         System.out.println(url);
      }
   }
}
```

2. Run the example code. The celebrity name and information about the celebrity, if it is available, is displayed.

Moderating Images

To determine if an image contains explicit or suggestive adult content, Amazon Rekognition provides the DetectModerationLabels (p. 122) API operation.

Amazon Rekognition uses a hierarchical taxonomy to label categories of explicit and suggestive content. The two top-level categories are *Explicit Nudity*, and *Suggestive*. Each top-level category has a number of second-level categories.

Top-Level Category	Second-Level Category
Explicit Nudity	Nudity
	Graphic Male Nudity
	Graphic Female Nudity
	Sexual Activity
	Partial Nudity
Suggestive	Female Swimwear Or Underwear
	Male Swimwear Or Underwear
	Revealing Clothes

You determine the suitability of an image for your application. For example, images of a suggestive nature might be acceptable, but images containing nudity might not. To filter images, use the ModerationLabel (p. 187) labels array returned by `DetectModerationLabels`.

The `ModerationLabel` array contains labels in the preceding categories and an estimated confidence in the accuracy of the recognized content. A top-level label is returned along with any second-level labels that were identified. For example, Rekognition may return “Explicit Nudity” with a high confidence score as a top-level label. That may be enough for your filtering needs, but if necessary, you can use the confidence score of a second-level label, such as "Partial Nudity", to obtain more granular filtering. For an example, see Detecting Moderation Labels (p. 56).

Note
Rekognition Image Moderation API is not an authority on, or in any way purports to be an exhaustive filter of, explicit and suggestive adult content. Furthermore, the Image Moderation

API does not detect whether an image includes illegal content (such as child pornography) or unnatural adult content.

`DetectModerationLabels` can retrieve input images from an S3 bucket or you can provide them as image bytes.

The following example is the response from a call to `DetectModerationLabels`.

```
{
"ModerationLabels": [
    {
        "Confidence": 99.24723052978516,
        "ParentName": "",
        "Name": "Explicit Nudity"
    },
    {
        "Confidence": 99.24723052978516,
        "ParentName": "Explicit Nudity",
        "Name": "Graphic Male Nudity"
    },
    {
        "Confidence": 88.25341796875,
        "ParentName": "Explicit Nudity",
        "Name": "Sexual Activity"
    }
]
}
```

The response includes the following:

- **Image moderation information** – The example shows a list of moderation labels for explicit or suggestive content found in the image. The list includes the top-level label and each second-level label detected in the image.

  Label – Each label has a name, an estimation of the confidence that Amazon Rekognition has that the label is accurate, and the name of its parent label. The parent name for a top-level label is `""`.

  Label confidence – Each label has a confidence value between 0 and 100 that indicates the percentage confidence that Amazon Rekognition has that the label is correct. You specify the required confidence level for a label to be returned in the response in the API operation request.

Topics

Detecting Moderation Labels

In these procedures you use the DetectModerationLabels (p. 122) operation to determine if an image contains explicit or suggestive adult content. The image must be in either a .jpg or a .png format. You can provide the input image as an image byte array (base64-encoded image bytes) or specify an S3 object. In these procedures you upload an image (.jpg or .png) to your S3 bucket.

Prerequisites

To run these procedures, you need to have the AWS CLI and AWS SDK for Java installed. For more information, see Getting Started with Amazon Rekognition (p. 17). The AWS account you use must have access permissions to the Amazon Rekognition API. For more information, see Amazon Rekognition API Permissions: Actions, Permissions, and Resources Reference (p. 47).

To detect moderation labels in an image (AWS CLI)

1. Upload an image to your S3 bucket.

   For instructions, see Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service Console User Guide*.
2. On the command line, type the following command. Replace `bucketname` and `input.jpg` with the S3 bucket name and image name that you used in step 1.

```
aws rekognition detect-moderation-labels \

--image '{"S3Object":{"Bucket":"bucketname","Name":"input.jpg"}}' \
--region us-east-1 \
--profile adminuser
```

3. To run the command, choose **Enter**. The JSON output for the `DetectModerationLabels` API operation is displayed.

To detect moderation labels in an image (API)

1. Upload an image to your S3 bucket.

   For instructions, see Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service Console User Guide*.
2. To detect moderation labels in an image, use the following AWS SDK for Java example code. Replace `bucketname` and `input.jpg` with the S3 bucket name and the image file name that you used in step 1.

```
  import com.amazonaws.services.rekognition.AmazonRekognition;
  import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
  import com.amazonaws.AmazonClientException;
  import com.amazonaws.auth.AWSCredentials;
  import com.amazonaws.auth.AWSStaticCredentialsProvider;
  import com.amazonaws.auth.profile.ProfileCredentialsProvider;
  import com.amazonaws.regions.Regions;
  import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
  import com.amazonaws.services.rekognition.model.DetectModerationLabelsRequest;
  import com.amazonaws.services.rekognition.model.DetectModerationLabelsResult;
  import com.amazonaws.services.rekognition.model.Image;
  import com.amazonaws.services.rekognition.model.ModerationLabel;
  import com.amazonaws.services.rekognition.model.S3Object;

  import java.util.List;

  public class DetectModerationLabelsExample
  {
     public static void main(String[] args) throws Exception
     {
        String image = "input.jpg";
        String bucket = "bucketname";
        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);
       }
       AmazonRekognition rekognitionClient = AmazonRekognitionClientBuilder.standard()
             .withRegion(Regions.US_WEST_2)
             .withCredentials(new AWSStaticCredentialsProvider(credentials)).build();

       DetectModerationLabelsRequest request = new DetectModerationLabelsRequest()
         .withImage(new Image().withS3Object(new
S3Object().withName(image).withBucket(bucket)))
         .withMinConfidence(60F);
       try
       {
            DetectModerationLabelsResult result =
rekognitionClient.detectModerationLabels(request);
            List<ModerationLabel> labels = result.getModerationLabels();
            System.out.println("Detected labels for " + image);
            for (ModerationLabel label : labels)
            {
               System.out.println("Label: " + label.getName()
                + "\n Confidence: " + label.getConfidence().toString() + "%"
                + "\n Parent:" + label.getParentName());
           }
        }
        catch (AmazonRekognitionException e)
        {
          e.printStackTrace();
        }
     }
 }
```

3. Run the sample code. The output lists the label name, confidence and parent label for each detected label.

Monitoring

To monitor Amazon Rekognition, use Amazon CloudWatch. This section provides information on how to set up monitoring for Rekognition, and reference content for Rekognition metrics.

Topics

Monitoring Rekognition

With CloudWatch, you can get metrics for individual Rekognition operations or global Rekognition metrics for your account, You can use metrics to track the health of your Rekognition-based solution and set up alarms to notify you when one or more metrics fall outside a defined threshold. For example, you can see metrics for the number of server errors that have occurred, or metrics for the number of faces that have been detected. You can also see metrics for the number of times a specific Rekognition operation has succeeded. To see metrics, you can use Amazon CloudWatch, Amazon AWS Command Line Interface, or the CloudWatch API.

You can also see aggregated metrics, for a chosen period of time, by using the Rekognition console. For more information, see Exercise 4: See Aggregated Metrics (Console) (p. 29).

Using CloudWatch Metrics for Rekognition

To use metrics, you must specify the following information:

- The metric dimension, or no dimension. A *dimension* is a name-value pair that helps you to uniquely identify a metric. Rekognition has one dimension, named *Operation*. It provides metrics for a specific operation. If you do not specify a dimension, the metric is scoped to all Rekognition operations within your account.
- The metric name, such as `UserErrorCount`.

You can get monitoring data for Rekognition using the AWS Management Console, the AWS CLI, or the CloudWatch API. You can also use the CloudWatch API through one of the Amazon AWS Software

Development Kits (SDKs) or the CloudWatch API tools. The console displays a series of graphs based on the raw data from the CloudWatch API. Depending on your needs, you might prefer to use either the graphs displayed in the console or retrieved from the API.

The following list shows some common uses for the metrics. These are suggestions to get you started, not a comprehensive list.

How Do I?	Relevant Metrics
How do I track the numbers of faces recognized?	Monitor the `Sum` statistic of the `DetectedFaceCount` metric.
How do I know if my application has reached the maximum number of requests per second?	Monitor the `Sum` statistic of the `ThrottledCount` metric.
How can I monitor the request errors?	Use the `Sum` statistic of the `UserErrorCount` metric.
How can I find the total number of requests?	Use the `ResponseTime` and `Data Samples` statistic of the `ResponseTime` metric. This includes any request that results in an error. If you want to see only successful operation calls, use the `SuccessfulRequestCount` metric.
How can I monitor the latency of `Rekognition` operation calls?	Use the `ResponseTime` metric.
How can I monitor how many times `IndexFaces` successfully added faces to Rekognition collections?	Monitor the `Sum` statistic with the `SuccessfulRequestCount` metric and `IndexFaces` operation. Use the `Operation` dimension to select the operation and metric.

You must have the appropriate CloudWatch permissions to monitor Rekognition with CloudWatch. For more information, see Authentication and Access Control for Amazon CloudWatch.

Access Rekognition Metrics

The following examples show how to access Rekognition metrics using the CloudWatch console, the AWS CLI, and the CloudWatch API.

To view metrics (console)

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Metrics**, choose the **All Metrics** tab, and then choose **Rekognition**.
3. Choose **Metrics with no dimensions**, and then choose a metric.

   For example, choose the **DetectedFace** metric to measure how many faces have been detected.
4. Choose a value for the date range. The metric count displayed in the graph.

To view metrics successful `DetectFaces` operation calls have been made over a period of time (CLI).

- Open the AWS CLI and enter the following command:

```
aws cloudwatch get-metric-statistics --metric-name SuccessfulRequestCount --start-
time 2017-1-1T19:46:20 --end-time 2017-1-6T19:46:57 --period 3600 --namespace AWS/
Rekognition --statistics Sum --dimensions Name=Operation,Value=DetectFaces --region us-
west-2
```

This example shows the successful `DetectFaces` operation calls made over a period of time. For more information, see get-metric-statistics.

To access metrics (CloudWatch API)

- Call `GetMetricStatistics`. For more information, see the Amazon CloudWatch API Reference.

Create an Alarm

You can create a CloudWatch alarm that sends an Amazon Simple Notification Service (Amazon SNS) message when the alarm changes state. An alarm watches a single metric over a time period you specify, and performs one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon SNS topic or an Auto Scaling policy.

Alarms invoke actions for sustained state changes only. CloudWatch alarms do not invoke actions simply because they are in a particular state. The state must have changed and been maintained for a specified number of time periods.

To set an alarm (console)

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Create Alarm**. This launches the **Create Alarm Wizard**.
3. In the **Metrics with no dimensions** metric list, choose **Rekognition Metrics**, and then choose a metric.

   For example, choose **DetectedFaceCount** to set an alarm for a maximum number of detected faces.
4. In the **Time Range** area, select a date range value that includes face detection operations that you have called. Choose **Next**
5. Fill in the **Name** and **Description**. For **Whenever**, choose **>=**, and enter a maximum value of your choice.
6. If you want CloudWatch to send you email when the alarm state is reached, for **Whenever this alarm:**, choose **State is ALARM**. To send alarms to an existing Amazon SNS topic, for **Send notification to:**, choose an existing SNS topic. To set the name and email addresses for a new email subscription list, choose **Create topic** CloudWatch saves the list and displays it in the field so you can use it to set future alarms.

   Note
   If you use **Create topic** to create a new Amazon SNS topic, the email addresses must be verified before the intended recipients receive notifications. Amazon SNS sends email only when the alarm enters an alarm state. If this alarm state change happens before the email addresses are verified, intended recipients do not receive a notification.
7. Preview the alarm in the **Alarm Preview** section. Choose **Create Alarm**.

To set an alarm (AWS CLI)

- Open the AWS CLI and enter the following command. Change value of the `alarm-actions` parameter to reference an Amazon SNS topic that you previously created.

```
aws cloudwatch put-metric-alarm --alarm-name UserErrors --alarm-description "Alarm
when more than 10 user errors occur" --metric-name UserErrorCount --namespace
AWS/Rekognition --statistic Average --period 300 --threshold 10 --comparison-
operator GreaterThanThreshold --evaluation-periods 2 --alarm-actions arn:aws:sns:us-
west-2:111111111111:UserError --unit Count
```

This example shows how to create an alarm for when more than 10 user errors occur within 5 minutes. For more information, see put-metric-alarm.

To set an alarm (CloudWatch API)

- Call `PutMetricAlarm`. For more information, see *Amazon CloudWatch API Reference*.

CloudWatch Metrics for Rekognition

This section contains information about the Amazon CloudWatch metrics and the *Operation* dimension available for Amazon Rekognition.

You can also see an aggregate view of Rekognition metrics from the Rekognition console. For more information, see Exercise 4: See Aggregated Metrics (Console) (p. 29).

CloudWatch Metrics for Rekognition

The following table summarizes the Rekognition metrics.

Metric	Description
SuccessfulRequestCount	The number of successful requests. The response code range for a successful request is 200 to 299. Unit: Count Valid statistics: `Sum,Average`
ThrottledCount	The number of throttled requests. Rekognition throttles a request when it receives more requests than the limit of transactions per second set for your account. If the limit set for your account is frequently exceeded, you can request a limit increase. To request an increase, see AWS Service Limits. Unit: Count Valid statistics: `Sum,Average`
ResponseTime	The time in milliseconds for Rekognition to compute the response. Units: 1. Count for `Data Samples` statistics 2. Milliseconds for `Average` statistics Valid statistics: `Data Samples,Average` **Note** The `ResponseTime` metric is not included in the Rekognition metric pane.

Metric	Description
DetectedFaceCount	The number of faces detected with the `IndexFaces` or `DetectFaces` operation. Unit: Count Valid statistics: `Sum,Average`
DetectedLabelCount	The number of labels detected with the `DetectLabels` operation. Unit: Count Valid statistics: `Sum,Average`
ServerErrorCount	The number of server errors. The response code range for a server error is 500 to 599. Unit: Count Valid statistics: `Sum, Average`
UserErrorCount	The number of user errors (invalid parameters, invalid image, no permission, etc). The response code range for a user error is 400 to 499. Unit: Count Valid statistics: `Sum,Average`

CloudWatch Dimension for Rekognition

To retrieve operation-specific metrics, use the `Rekognition` namespace and provide an operation dimension. For more information about dimensions, see Dimensions in the *Amazon CloudWatch User Guide*.

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. 17)

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. 18). 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 two collections. The code example also lists the available collections.

```
import java.util.List;
import com.amazonaws.AmazonClientException;
```

```
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
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;

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);
      }


      AmazonRekognition amazonRekognition = AmazonRekognitionClientBuilder
         .standard()
         .withRegion(Regions.US_WEST_2)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // 1. CreateCollection 1
      String collectionId = "exampleCollection";
      String collectionId2 = "exampleCollection2";
      System.out.println("Creating collections: " +
         collectionId +
         " & " + collectionId2);
      CreateCollectionResult createCollectionResult = callCreateCollection(
         collectionId, amazonRekognition);
      System.out.println("CollectionArn : " +
         createCollectionResult.getCollectionArn());
      System.out.println("Status code : " +
         createCollectionResult.getStatusCode().toString());

      //CreateCollection 2
      createCollectionResult = callCreateCollection(collectionId2,
         amazonRekognition);
      System.out.println("CollectionArn : " +
         createCollectionResult.getCollectionArn());
      System.out.println("Status code : " +
         createCollectionResult.getStatusCode().toString());


      // 3. Page through collections with ListCollections
      System.out.println("Listing collections");
      int limit = 1;
      ListCollectionsResult listCollectionsResult = null;
      String paginationToken = null;
      do {
         if (listCollectionsResult != null) {
            paginationToken = listCollectionsResult.getNextToken();
         }
```

```
            listCollectionsResult = callListCollections(paginationToken, limit,
               amazonRekognition);

            List < String > collectionIds = listCollectionsResult.getCollectionIds();
            for (String resultId: collectionIds) {
               System.out.println(resultId);
            }
         } while (listCollectionsResult != null && listCollectionsResult.getNextToken() !=
            null);

         // 4. Clean up collections with DeleteCollection

         System.out.println("Deleting collections");
         DeleteCollectionResult deleteCollectionResult = callDeleteCollection(
            collectionId, amazonRekognition);
         System.out.println(collectionId + ": " + deleteCollectionResult.getStatusCode()
            .toString());

         DeleteCollectionResult deleteCollectionResult2 = callDeleteCollection(
            collectionId2, amazonRekognition);
         System.out.println(collectionId2 + ": " + deleteCollectionResult2.getStatusCode()
            .toString());

      }

      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. 140) 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. 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 com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.Face;
import com.amazonaws.services.rekognition.model.FaceRecord;
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 = "collectionid";
   public static final String S3_BUCKET = "S3Bucket";

   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);
      }

      ObjectMapper objectMapper = new ObjectMapper();


      AmazonRekognition amazonRekognition = AmazonRekognitionClientBuilder
         .standard()
         .withRegion(Regions.US_WEST_2)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();
```

```
    // 1. Index face 1
    Image image = getImageUtil(S3_BUCKET, "image1.jpg");
    String externalImageId = "image1.jpg";
    IndexFacesResult indexFacesResult = callIndexFaces(COLLECTION_ID,
       externalImageId, "ALL", image, amazonRekognition);
    System.out.println(externalImageId + " added");
    List < FaceRecord > faceRecords = indexFacesResult.getFaceRecords();
    for (FaceRecord faceRecord: faceRecords) {
       System.out.println("Face detected: Faceid is " +
          faceRecord.getFace().getFaceId());
    }

    // 2. Index face 2
    indexFacesResult = null;
    faceRecords = null;
    Image image2 = getImageUtil(S3_BUCKET, "image2.jpg");
    String externalImageId2 = "image2.jpg";
    System.out.println(externalImageId2 + " added");
    indexFacesResult = callIndexFaces(COLLECTION_ID, externalImageId2,
       "ALL", image2, amazonRekognition);
    faceRecords = indexFacesResult.getFaceRecords();
    for (FaceRecord faceRecord: faceRecords) {
       System.out.println("Face detected. Faceid is " +
          faceRecord.getFace().getFaceId());
    }


    // 3. Page through the faces with ListFaces
    ListFacesResult listFacesResult = null;
    System.out.println("Faces in collection " + COLLECTION_ID);

    String paginationToken = null;
    do {
       if (listFacesResult != null) {
          paginationToken = listFacesResult.getNextToken();
       }
       listFacesResult = callListFaces(COLLECTION_ID, 1, paginationToken,
          amazonRekognition);
       List < Face > faces = listFacesResult.getFaces();
       for (Face face: faces) {
          System.out.println(objectMapper.writerWithDefaultPrettyPrinter()
             .writeValueAsString(face));
       }
    } 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. 152)) and `search-faces-by-image` (see SearchFacesByImage (p. 157)) 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. 14).

  ```
  {
      "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. 14).

```
{
```

```
    "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. 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 java.util.List;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.FaceMatch;
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;


public class SearchFacesExample {

   public static final String COLLECTION_ID = "collectionid";
   public static final String S3_BUCKET = "S3Bucket";


   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);
        }


        AmazonRekognition amazonRekognition = AmazonRekognitionClientBuilder
           .standard()
           .withRegion(Regions.US_WEST_2)
           .withCredentials(new AWSStaticCredentialsProvider(credentials))
           .build();

        IndexFacesResult indexFacesResult = callIndexFaces(COLLECTION_ID,
           amazonRekognition, "image1.jpg");

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


        callIndexFaces(COLLECTION_ID, amazonRekognition, "image2.jpg");
        callIndexFaces(COLLECTION_ID, amazonRekognition, "image3.jpg");

        Float threshold = 70F;
        int maxFaces = 2;

        //3. Search similar faces for a give face (identified by face ID).
        System.out.println("Faces matching FaceId: " + faceId);
        SearchFacesResult searchFacesResult = callSearchFaces(COLLECTION_ID,
           faceId, threshold, maxFaces, amazonRekognition);
        List < FaceMatch > faceMatches = searchFacesResult.getFaceMatches();
        for (FaceMatch face: faceMatches) {
           System.out.println(face.getFace().toString());
           System.out.println();
        }

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

        //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("Faces matching largest face in image  " + fileName);
        List < FaceMatch > faceImageMatches = searchFacesByImageResult.getFaceMatches();
        for (FaceMatch face: faceImageMatches) {
           System.out.println(face.getFace().toString());
           System.out.println();
        }

     }

     private static IndexFacesResult callIndexFaces(
        String collectionId, AmazonRekognition amazonRekognition, String name) {
        IndexFacesRequest req = new IndexFacesRequest()
           .withImage(getImageUtil(S3_BUCKET, name))
           .withCollectionId(collectionId)
```

```
            .withExternalImageId(name);

        return amazonRekognition.indexFaces(req);
    }

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

}
```

Example 4: Supplying Image Bytes to Amazon Rekognition Operations

This section provides AWS SDK examples of supplying image bytes to Amazon Rekognition API operations by using a file loaded from a local file system. A Rekognition API operation can analyse an image provided as base64 encoded image bytes or it can analyze an image retrieved from an Amazon S3 bucket. You pass an image to a Rekognition API operation by using the Image (p. 183) input parameter. Within `Image` you specify the `Bytes` property to pass base64 encoded image bytes or you specify the S3Object (p. 191) object property to reference an image stored in an S3 bucket.

Image bytes passed to a Rekognition API operation using the `Bytes` input parameter must be base64 encoded. The following common AWS SDKs automatically base64 encode images and you do not need to encode image bytes prior to calling a Rekognition API operation.

- Java
- JavaScript
- Python
- PHP

If you are using another AWS SDK and get an image format error when calling a Rekognition API operation, try base64 encoding the image bytes before passing them to a Rekognition API operation.

Note
The image does not need to be base64 encoded if you pass an image stored in an `S3Object` instead of image bytes.

If you use HTTP to call Amazon Rekognition operations, the image bytes must be a base64-encoded string. For more information, see Working with Images (p. 30).

The following examples show how to load images from the local file system and supply the image bytes to a Rekognition operation.

Topics

Supplying Images: Using the Local File System and Java

The following Java example shows how to load an image from the local file system and detect labels using the `detectLabels` AWS SDK operation.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.util.List;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
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.Label;
import com.amazonaws.util.IOUtils;

public class DetectLabelsExampleImageBytes {
    public static void main(String[] args) throws Exception {
     String photo="/path/inputimage.jpg";

        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 (/Usersuserid.aws/credentials), and is in a valid format.",
 e);
        }
        ByteBuffer imageBytes;
        try (InputStream inputStream = new FileInputStream(new File(photo))) {
            imageBytes = ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
        }
```

```
        AmazonRekognition rekognitionClient = AmazonRekognitionClientBuilder
            .standard()
            .withRegion(Regions.US_WEST_2)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

        DetectLabelsRequest request = new DetectLabelsRequest()
                .withImage(new Image()
                        .withBytes(imageBytes))
                .withMaxLabels(10)
                .withMinConfidence(77F);

        try {

            DetectLabelsResult result = rekognitionClient.detectLabels(request);
            List <Label> labels = result.getLabels();

            System.out.println("Detected labels for " + photo);
            for (Label label: labels) {
               System.out.println(label.getName() + ": " +
 label.getConfidence().toString());
            }

        } catch (AmazonRekognitionException e) {
            e.printStackTrace();
        }

    }
}
```

Supplying Images: Using the Local File System and Python

The following AWS SDK for Python example shows how to load an image from the local file system and add them to a collection using the IndexFaces operation.

```
#!/usr/bin/env python

from argparse import ArgumentParser
import boto3
from os import environ

def get_client(endpoint):
    key_id = environ.get('AWS_ACCESS_KEY_ID')
    secret_key = environ.get('AWS_SECRET_ACCESS_KEY')
    token = environ.get('AWS_SESSION_TOKEN')
    if not key_id or not secret_key or not token:
        raise Exception('Missing AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, or
 AWS_SESSION_TOKEN')
    client = boto3.client('rekognition', region_name='us-east-1', endpoint_url=endpoint,
 verify=False,
                          aws_access_key_id=key_id, aws_secret_access_key=secret_key,
 aws_session_token=token)
    return client

def get_args():
    parser = ArgumentParser(description='Call index faces')
    parser.add_argument('-e', '--endpoint')
    parser.add_argument('-i', '--image')
    parser.add_argument('-c', '--collection')
    return parser.parse_args()
```

```
if __name__ == '__main__':
    args = get_args()
    client = get_client(args.endpoint)
    with open(args.image, 'rb') as image:
        response = client.index_faces(Image={'Bytes': image.read()},
 CollectionId=args.collection)
        print response
    print "help"
```

Supplying Images: Using the Local File System and PHP

The following AWS SDK for PHP example shows how to load an image from the local file system and call the DetectFaces API operation.

```
<?php
    require 'path/vendor/autoload.php';

    use Aws\Rekognition\RekognitionClient;

    $options = [
        'region'            => 'us-west-2',
        'version'           => '2016-06-27',
    ];

    $rekognition = new RekognitionClient($options);

    #Get local image
    $fp_image = fopen('test.png', 'r');
    $image = fread($fp_image, filesize('test.png'));
    fclose($fp_image);


    # Call DetectFaces
    $result = $rekognition->DetectFaces(array(
       'Image' => array(
          'Bytes' => $image,
       ),
       'Attributes' => array('ALL')
       )
    );

    # Display info for each detected person
    print 'People: Image position and estimated age' . PHP_EOL;
    for ($n=0;$n<sizeof($result['FaceDetails']); $n++){

      print 'Position: ' . $result['FaceDetails'][$n]['BoundingBox']['Left'] . " "
      . $result['FaceDetails'][$n]['BoundingBox']['Top']
      . PHP_EOL
      . 'Age (low): '.$result['FaceDetails'][$n]['AgeRange']['Low']
      .  PHP_EOL
      . 'Age (high): ' . $result['FaceDetails'][$n]['AgeRange']['High']
      .  PHP_EOL . PHP_EOL;
    }
?>
```

Example 5: Getting Image Orientation and Bounding Box Coordinates

Applications that use Amazon Rekognition commonly need to display the images that are detected by Amazon Rekognition operations and the boxes around detected faces. To display an image correctly in your application, you need to know the image's orientation and possibly correct it. For some .jpg files, the image's orientation is contained in the image's Exchangeable image file format (Exif) metadata. For other .jpg files and all .png files, Amazon Rekognition operations return the estimated orientation.

To display a box around a face, you need the coordinates for the face's bounding box and, if the box isn't oriented correctly, you might need to adjust those coordinates. Amazon Rekognition face detection operations return bounding box coordinates for each detected face.

The following Amazon Rekognition operations return information for correcting an image's orientation and bounding box coordinates:

- CompareFaces (p. 92)
- DetectFaces (p. 109)
- DetectLabels (p. 115) (returns only information to correct image orientation)
- IndexFaces (p. 129)
- RecognizeCelebrities (p. 146)

This example shows how to get the following information for your code:

- The estimated orientation of an image (if there is no orientation information in Exif metadata)
- The bounding box coordinates for the faces detected in an image
- Translated bounding box coordinates for bounding boxes that are affected by estimated image orientation

Use the information in this example to ensure that your images are oriented correctly and that bounding boxes are displayed in the correct location in your application.

Because the code used to rotate and display images and bounding boxes depends on the language and environment that you use, we don't explain how to display images and bounding boxes in your code or how to get orientation information from Exif metadata.

Finding an Image's Orientation

To display an image correctly in your application, you might need to rotate it. The following image is oriented to 0 degrees and is displayed correctly.

However, the following image is rotated 90 degrees counterclockwise. To display it correctly, you need to find the orientation of the image and use that information in your code to rotate the image to 0 degrees.

Some images in .jpg format contain orientation information in Exif metadata. If the value of the `OrientationCorrection` field is `null` in the operation's response, the Exif metadata for the image contains the orientation. In the Exif metadata, you can find the image's orientation in the `orientation` field. Although Amazon Rekognition identifies the presence of image orientation information in Exif metadata, it does not provide access to it. To access the Exif metadata in an image, use a third-party library or write your own code. For more information, see Exif Version 2.31.

Images in .png format do not have Exif metadata. For .jpg images that don't have Exif metadata and for all .png images, Amazon Rekognition operations return an estimated orientation for the image in the `OrientationCorrection` field. Estimated orientation is measured counterclockwise and in increments of 90 degrees. For example, Amazon Rekognition returns ROTATE_0 for an image that is oriented to 0 degrees and ROTATE_90 for an image that is rotated 90 degrees counterclockwise.

> **Note**
> The `CompareFaces` operation returns the source image orientation in the `SourceImageOrientationCorrection` field and the target image orientation in the `TargetImageOrientationCorrection` field.

When you know an image's orientation, you can write code to rotate and correctly display it.

Displaying Bounding Boxes

The Amazon Rekognition operations that analyze faces in an image also return the coordinates of the bounding boxes that surround the faces. For more information, see BoundingBox (p. 166).

To display a bounding box around a face similar to the box shown in the following image in your application, use the bounding box coordinates in your code. The bounding box coordinates returned by an operation reflect the image's orientation. If you have to rotate the image to display it correctly, you might need to translate the bounding box coordinates.

Displaying Bounding Boxes When Orientation Information Is Not Present in Exif Metadata

If an image doesn't have Exif metadata, or if the `orientation` field in the Exif metadata is not populated, Amazon Rekognition operations return the following:

- An estimated orientation for the image
- The bounding box coordinates oriented to the estimated orientation

If you need to rotate the image to display it correctly, you also need to rotate the bounding box.

For example, the following image is oriented at 90 degrees counterclockwise and shows a bounding box around the face. The bounding box is displayed using the coordinates for the estimated orientation returned from an Amazon Rekognition operation.

When you rotate the image to 0 degrees orientation, you also need to rotate the bounding box by translating the bounding box coordinates. For example, the following image has been rotated to 0 degrees from 90 degrees counterclockwise. The bounding box coordinates have not yet been translated, so the bounding box is displayed in the wrong position.

To rotate and display bounding boxes when orientation isn't present in Exif metadata

1. Call an Amazon Rekognition operation providing an input image with at least one face and with no Exif metadata orientation. For an example, see Exercise 2: Detect Faces (API) (p. 33).
2. Note the estimated orientation returned in the response's `OrientationCorrection` field.
3. Rotate the image to 0 degrees orientation by using the estimated orientation you noted in step 2 in your code.
4. Translate the top and left bounding box coordinates to 0 degrees orientation and convert them to pixel points on the image in your code. Use the formula in the following list that matches the estimated orientation you noted in step 2.

   Note the following definitions:

   - `ROTATE_(n)` is the estimated image orientation returned by an Amazon Rekognition operation.
   - `<face>` represents information about the face that is returned by an Amazon Rekognition operation. For example, the FaceDetail (p. 177) data type that the DetectFaces (p. 109) operation returns contains bounding box information for faces detected in the source image.
   - `image.width` and `image.height` are pixel values for the width and height of the source image.
   - The bounding box coordinates are a value between 0 and 1 relative to the image size. For example, for an image with 0 degree orientation, a `BoundingBox.left` value of 0.9 puts the left coordinate close to the right side of the image. To display the box, translate the bounding box coordinate values to pixel points on the image and rotate them to 0 degrees, as shown in each of the following formulas. For more information, see BoundingBox (p. 166).

   ROTATE_0

   ```
   left = image.width*BoundingBox.Left

   top = image.height*BoundingBox.Top
   ```

   ROTATE_90

   ```
   left = image.height * (1 - (<face>.BoundingBox.Top + <face>.BoundingBox.Height)

   top = image.width * <face>.BoundingBox.Left
   ```

   ROTATE_180

   ```
   left = image.width - (image.width*(<face>.BoundingBox.Left
   +<face>.BoundingBox.Width))

   top = image.height * (1 - (<face>.BoundingBox.Top + <face>.BoundingBox.Height))
   ```

   ROTATE_270

   ```
   left = image.height * BoundingBox.top
   ```

```
top = image.width * (1 - BoundingBox.Left - BoundingBox.Width)
```

5. Using the following formulas, calculate the bounding box's width and height as pixel ranges on the image in your code.

   The width and height of a bounding box is returned in the `BoundingBox.Width` and `BoundingBox.Height` fields. The width and height values range between 0 and 1 relative to the image size. `image.width` and `image.height` are pixel values for the width and height of the source image.

   ```
   box width = image.width * (<face>.BoundingBox.Width)

   box height = image.height * (<face>.BoundingBox.Height)
   ```

6. Display the bounding box on the rotated image by using the values calculated in steps 4 and 5.

Displaying Bounding Boxes When Orientation Information is Present in Exif Metadata

If an image's orientation is included in Exif metadata, Amazon Rekognition operations do the following:

- Return null in the orientation correction field in the operation's response. To rotate the image, use the orientation provided in the Exif metadata in your code.
- Return bounding box coordinates already oriented to 0 degrees. To show the bounding box in the correct position, use the coordinates that were returned. You do not need to translate them.

Example: Getting Image Orientation and Bounding Box Coordinates For an Image

The following example shows how to use the AWS SDK for Java to get the estimated orientation of an image and to translate bounding box coordinates for faces detected by the `DetectFaces` operation.

The example loads an image from the local file system, calls the `DetectFaces` operation, determines the height and width of the image, and calculates the bounding box coordinates of the face for the rotated image. The example does not show how to process orientation information that is stored in Exif metadata.

To use this code, replace `input.jpg` with the name and path of an image that is stored locally in either .png or .jpg format.

```
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.util.List;
import javax.imageio.ImageIO;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.rekognition.AmazonRekognition;
```

```
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.util.IOUtils;
import com.amazonaws.services.rekognition.model.AgeRange;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.Attribute;
import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.model.DetectFacesRequest;
import com.amazonaws.services.rekognition.model.DetectFacesResult;
import com.amazonaws.services.rekognition.model.FaceDetail;

public class RotateImage {

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

  String photo = "input.jpg";

  //Get Rekognition client
  AWSCredentials credentials = null;
  try {
     credentials = new ProfileCredentialsProvider("AdminUser").getCredentials();
  } catch (Exception e) {
     throw new AmazonClientException("Cannot load the credentials: ", e);
  }

  AmazonRekognition amazonRekognition = AmazonRekognitionClientBuilder
     .standard()
     .withRegion(Regions.US_WEST_2)
     .withCredentials(new AWSStaticCredentialsProvider(credentials))
     .build();


  // Load image
  ByteBuffer imageBytes=null;
  BufferedImage image = null;

  try (InputStream inputStream = new FileInputStream(new File(photo))) {
     imageBytes = ByteBuffer.wrap(IOUtils.toByteArray(inputStream));

  }
  catch(Exception e)
  {
      System.out.println("Failed to load file " + photo);
      System.exit(1);
  }

  //Get image width and height
  InputStream imageBytesStream;
  imageBytesStream = new ByteArrayInputStream(imageBytes.array());

  ByteArrayOutputStream baos = new ByteArrayOutputStream();
  image=ImageIO.read(imageBytesStream);
  ImageIO.write(image, "jpg", baos);

  int height = image.getHeight();
  int width = image.getWidth();

  System.out.println("Image Information:");
  System.out.println(photo);
  System.out.println("Image Height: " + Integer.toString(height));
  System.out.println("Image Width: " + Integer.toString(width));

  //Call detect faces and show face age and placement

  try{
    DetectFacesRequest request = new DetectFacesRequest()
```

```
            .withImage(new Image()
               .withBytes((imageBytes)))
            .withAttributes(Attribute.ALL);


      DetectFacesResult result = amazonRekognition.detectFaces(request);
      System.out.println("Orientation: " + result.getOrientationCorrection() + "\n");
      List <FaceDetail> faceDetails = result.getFaceDetails();

      for (FaceDetail face: faceDetails) {
        System.out.println("Face:");
          ShowBoundingBoxPositions(height,
                  width,
                  face.getBoundingBox(),
                  result.getOrientationCorrection());
          AgeRange ageRange = face.getAgeRange();
          System.out.println("The detected face is estimated to be between "
               + ageRange.getLow().toString() + " and " + ageRange.getHigh().toString()
               + " years old.");
            System.out.println();
       }

   } catch (AmazonRekognitionException e) {
      e.printStackTrace();
   }

}


public static void ShowBoundingBoxPositions(int imageHeight, int imageWidth, BoundingBox
 box, String rotation) {

  float left = 0;
  float top = 0;

  if(rotation==null){
      System.out.println("No estimated estimated orientation. Check Exif data.");
      return;
  }
  //Calculate face position based on image orientation.
  switch (rotation) {
     case "ROTATE_0":
        left = imageWidth * box.getLeft();
        top = imageHeight * box.getTop();
        break;
     case "ROTATE_90":
        left = imageHeight * (1 - (box.getTop() + box.getHeight()));
        top = imageWidth * box.getLeft();
        break;
     case "ROTATE_180":
        left = imageWidth - (imageWidth * (box.getLeft() + box.getWidth()));
        top = imageHeight * (1 - (box.getTop() + box.getHeight()));
        break;
     case "ROTATE_270":
        left = imageHeight * box.getTop();
        top = imageWidth * (1 - box.getLeft() - box.getWidth());
        break;
     default:
        System.out.println("No estimated orientation information. Check Exif data.");
        return;
  }

  //Display face location information.
  System.out.println("Left: " + String.valueOf((int) left));
  System.out.println("Top: " + String.valueOf((int) top));
  System.out.println("Face Width: " + String.valueOf((int)(imageWidth * box.getWidth())));
```

```
  System.out.println("Face Height: " + String.valueOf((int)(imageHeight *
 box.getHeight())));

  }
}
```

API Reference

This section provides documentation for the Amazon Rekognition API operations.

HTTP Headers

Beyond the usual HTTP headers, Amazon Rekognition HTTP operations have the following required headers:

Header	Value	Description
Content-Type:	application/x-amz-json-1.1	Specifies that the request content is JSON. Also specifies the JSON version.
X-Amz-Date:	<Date>	The date used to create the signature in the Authorization header. The format must be ISO 8601 basic in the YYYYMMDD'T'HHMMSS'Z' format. For example, the following date/time 20141123T120000Z is a valid x-amz-date for use with Amazon Rekognition.
X-Amz-Target:	RekognitionService.<operation>	The target Amazon Rekognition operation. For example, use `RekognitionService.ListCollections` to call the `ListCollections` operation.

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 compares it with each face detected in the target image.

You pass the input and target images either as base64-encoded image bytes or as a references to images in an Amazon S3 bucket. If you use the Amazon CLI to call Amazon Rekognition operations, passing image bytes is not supported. The image must be either a PNG or JPEG formatted file.

In response, the operation returns an array of face matches ordered by similarity score in descending order. For each face match, the response provides a bounding box of the face, facial landmarks, pose details (pitch, role, and yaw), quality (brightness and sharpness), and confidence value (indicating the level of confidence that the bounding box contains a face). The response also provides a similarity score, which indicates how closely the faces match.

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

`CompareFaces` also returns an array of faces that don't match the source image. For each face, it returns a bounding box, confidence value, landmarks, pose details, and quality. The response also returns information about the face in the source image, including the bounding box of the face and confidence value.

If the image doesn't contain Exif metadata, `CompareFaces` returns orientation information for the source and target images. Use these values to display the images with the correct image orientation.

If no faces are detected in the source or target images, `CompareFaces` returns an `InvalidParameterException` error.

> **Note**
> This is a stateless API operation. That is, data returned by this operation doesn't persist.

For an example, see Exercise 3: Compare Faces (API) (p. 35).

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. 92)

The minimum level of confidence in the face matches that a match must meet to be included in the `FaceMatches` array.

Type: Float

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

Required: No

SourceImage (p. 92)

The input image as base64-encoded bytes or an S3 object. If you use the AWS CLI to call Amazon Rekognition operations, passing base64-encoded image bytes is not supported.

Type: Image (p. 183) object

Required: Yes

TargetImage (p. 92)

The target image as base64-encoded bytes or an S3 object. If you use the AWS CLI to call Amazon Rekognition operations, passing base64-encoded image bytes is not supported.

Type: Image (p. 183) object

Required: Yes

Response Syntax

```
{
   "FaceMatches": [
      {
         "Face": {
            "BoundingBox": {
               "Height": number,
               "Left": number,
               "Top": number,
               "Width": number
            },
            "Confidence": number,
            "Landmarks": [
               {
                  "Type": "string",
                  "X": number,
                  "Y": number
               }
            ],
            "Pose": {
               "Pitch": number,
               "Roll": number,
               "Yaw": number
            },
            "Quality": {
```

```
                "Brightness": number,
                "Sharpness": number
            }
         },
         "Similarity": number
      }
   ],
   "SourceImageFace": {
      "BoundingBox": {
         "Height": number,
         "Left": number,
         "Top": number,
         "Width": number
      },
      "Confidence": number
   },
   "SourceImageOrientationCorrection": "string",
   "TargetImageOrientationCorrection": "string",
   "UnmatchedFaces": [
      {
         "BoundingBox": {
            "Height": number,
            "Left": number,
            "Top": number,
            "Width": number
         },
         "Confidence": number,
         "Landmarks": [
            {
               "Type": "string",
               "X": number,
               "Y": number
            }
         ],
         "Pose": {
            "Pitch": number,
            "Roll": number,
            "Yaw": number
         },
         "Quality": {
            "Brightness": number,
            "Sharpness": 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. 93)

An array of faces in the target image that match the source image face. Each `CompareFacesMatch` object provides the bounding box, the confidence level that the bounding box contains a face, and the similarity score for the face in the bounding box and the face in the source image.

Type: Array of CompareFacesMatch (p. 171) objects

SourceImageFace (p. 93)

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

Type: ComparedSourceImageFace (p. 170) object

SourceImageOrientationCorrection (p. 93)

The orientation of the source image (counterclockwise direction). If your application displays the source image, you can use this value to correct image orientation. The bounding box coordinates returned in `SourceImageFace` represent the location of the face before the image orientation is corrected.

Note
If the source image is in .jpeg format, it might contain exchangeable image (Exif) metadata that includes the image's orientation. If the Exif metadata for the source image populates the orientation field, the value of `OrientationCorrection` is null and the `SourceImageFace` bounding box coordinates represent the location of the face after Exif metadata is used to correct the orientation. Images in .png format don't contain Exif metadata.

Type: String

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

TargetImageOrientationCorrection (p. 93)

The orientation of the target image (in counterclockwise direction). If your application displays the target image, you can use this value to correct the orientation of the image. The bounding box coordinates returned in `FaceMatches` and `UnmatchedFaces` represent face locations before the image orientation is corrected.

Note
If the target image is in .jpg format, it might contain Exif metadata that includes the orientation of the image. If the Exif metadata for the target image populates the orientation field, the value of `OrientationCorrection` is null and the bounding box coordinates in `FaceMatches` and `UnmatchedFaces` represent the location of the face after Exif metadata is used to correct the orientation. Images in .png format don't contain Exif metadata.

Type: String

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

UnmatchedFaces (p. 93)

An array of faces in the target image that did not match the source image face.

Type: Array of ComparedFace (p. 169) objects

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. 194).

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

Example

Example Request

The following example shows a request that compares a source image (people.img) with a target image (family.jpg).

Sample Request

```
POST https://rekognition.us-west-2.amazonaws.com/ HTTP/1.1
Host: rekognition.us-west-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 170
X-Amz-Target: RekognitionService.CompareFaces
X-Amz-Date: 20170105T165437Z
User-Agent: aws-cli/1.11.25 Python/2.7.9 Windows/8 botocore/1.4.82
Content-Type: application/x-amz-json-1.1
Authorization: AWS4-HMAC-SHA256 Credential=XXXXXXXX/20170105/us-west-2/rekognition/
aws4_request,
 SignedHeaders=content-type;host;x-amz-date;x-amz-target, Signature=XXXXXXXX
{
   "TargetImage":{
      "S3Object":{
         "Bucket":"example-photos",
         "Name":"family.jpg"
      }
   },
   "SourceImage":{
      "S3Object":{
```

```
         "Bucket":"example-photos",
         "Name":"people.jpg"
      }
   }
```

Sample Response

```
{
    "FaceMatches": [{
        "Face": {
            "BoundingBox": {
                "Width": 0.5521978139877319,
                "Top": 0.1203877404332161,
                "Left": 0.23626373708248138,
                "Height": 0.3126954436302185
            },
            "Confidence": 99.98751068115234,
            "Pose": {
                "Yaw": -82.36799621582031,
                "Roll": -62.13221740722656,
                "Pitch": 0.8652129173278809
            },
            "Quality": {
                "Sharpness": 99.99880981445312,
                "Brightness": 54.49755096435547
            },
            "Landmarks": [{
                    "Y": 0.2996366024017334,
                    "X": 0.41685718297958374,
                    "Type": "eyeLeft"
                },
                {
                    "Y": 0.2658946216106415,
                    "X": 0.4414493441581726,
                    "Type": "eyeRight"
                },
                {
                    "Y": 0.3465650677680969,
                    "X": 0.48636093735694885,
                    "Type": "nose"
                },
                {
                    "Y": 0.30935320258140564,
                    "X": 0.6251809000968933,
                    "Type": "mouthLeft"
                },
                {
                    "Y": 0.26942989230155945,
                    "X": 0.6454493403434753,
                    "Type": "mouthRight"
                }
            ]
        },
        "Similarity": 100.0
    }],
    "SourceImageOrientationCorrection": "ROTATE_90",
    "TargetImageOrientationCorrection": "ROTATE_90",
    "UnmatchedFaces": [{
        "BoundingBox": {
            "Width": 0.4890109896659851,
            "Top": 0.6566604375839233,
            "Left": 0.10989011079072952,
            "Height": 0.278298944234848
        },
```

```
        "Confidence": 99.99992370605469,
        "Pose": {
            "Yaw": 51.51519012451172,
            "Roll": -110.32493591308594,
            "Pitch": -2.322134017944336
        },
        "Quality": {
            "Sharpness": 99.99671173095703,
            "Brightness": 57.23163986206055
        },
        "Landmarks": [{
                "Y": 0.8288310766220093,
                "X": 0.3133862614631653,
                "Type": "eyeLeft"
            },
            {
                "Y": 0.7632885575294495,
                "X": 0.28091415762901306,
                "Type": "eyeRight"
            },
            {
                "Y": 0.7417283654212952,
                "X": 0.3631140887737274,
                "Type": "nose"
            },
            {
                "Y": 0.8081989884376526,
                "X": 0.48565614223480225,
                "Type": "mouthLeft"
            },
            {
                "Y": 0.7548204660415649,
                "X": 0.46090251207351685,
                "Type": "mouthRight"
            }
        ]
    }],
    "SourceImageFace": {
        "BoundingBox": {
            "Width": 0.5521978139877319,
            "Top": 0.1203877404332161,
            "Left": 0.23626373708248138,
            "Height": 0.3126954436302185
        },
        "Confidence": 99.98751068115234
    }
}
```


See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python

- AWS SDK for Ruby V2

CreateCollection

Creates a collection in an AWS Region. You can add faces to the collection using the IndexFaces (p. 129) 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.

Note
Collection names are case-sensitive.

For an example, see Example 1: Managing Collections (p. 64).

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. 100)

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. 100)

Amazon Resource Name (ARN) of the collection. You can use this to manage permissions on your resources.

Type: String

StatusCode (p. 100)

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

Example

Example Request

The following example shows a request that creates a collection named mycollection.

Sample Request

```
POST https://rekognition.us-west-2.amazonaws.com/ HTTP/1.1
Host: rekognition.us-west-2.amazonaws.com
Accept-Encoding: identity
```

```
Content-Length: 32
X-Amz-Target: RekognitionService.CreateCollection
X-Amz-Date: 20170105T155520Z
User-Agent: aws-cli/1.11.25 Python/2.7.9 Windows/8 botocore/1.4.82
Content-Type: application/x-amz-json-1.1
Authorization: AWS4-HMAC-SHA256 Credential=XXXXXXXXXXXXXXXXXX/20170105/us-west-2/
rekognition/aws4_request,
  SignedHeaders=content-type;host;x-amz-date;x-amz-target,
 Signature=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

{"CollectionId": "mycollection"}
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/x-amz-json-1.1
Date: Thu, 05 Jan 2017 15:55:22 GMT
x-amzn-RequestId: 5d4c8b73-d35f-11e6-96d5-039839f35287
Content-Length: 99
Connection: keep-alive
{
   "CollectionArn":"aws:rekognition:us-west-2:11111111111:collection/mycollection",
   "StatusCode":200
}
```

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DeleteCollection

Deletes the specified collection. Note that this operation removes all faces in the collection. For an example, see Example 1: Managing Collections (p. 64).

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. 103)

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. 103)

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

Example

Example Request

The following example shows a request that deletes a collection named mycollection.

Sample Request

```
POST https://rekognition.us-west-2.amazonaws.com/ HTTP/1.1
Host: rekognition.us-west-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 32
X-Amz-Target: RekognitionService.DeleteCollection
X-Amz-Date: 20170105T170937Z
User-Agent: aws-cli/1.11.25 Python/2.7.9 Windows/8 botocore/1.4.82
Content-Type: application/x-amz-json-1.1
Authorization: AWS4-HMAC-SHA256 Credential=XXXXX/us-west-2/rekognition/aws4_request,
 SignedHeaders=content-type;host;x-amz-date;x-amz-target,
   Signature=XXXXXXXXXXXXXXXX

{"CollectionId": "mycollection"}
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/x-amz-json-1.1
```

```
Date: Thu, 05 Jan 2017 17:09:39 GMT
x-amzn-RequestId: bde4e432-d369-11e6-9921-8744f72327ab
Content-Length: 18
Connection: keep-alive

{"StatusCode":200}
```

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

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. 106)

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. 106)

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. 106)

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

Example

Example Request

The following example shows a request that deletes a face from a collection named examplemyphotos.

Sample Request

```
POST https://rekognition.us-west-2.amazonaws.com/ HTTP/1.1
Host: rekognition.us-west-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 81
X-Amz-Target: RekognitionService.DeleteFaces
X-Amz-Date: 20170105T170305Z
User-Agent: aws-cli/1.11.25 Python/2.7.9 Windows/8 botocore/1.4.82
```

```
Content-Type: application/x-amz-json-1.1
Authorization: AWS4-HMAC-SHA256 Credential=XXXXXXXX/20170105/us-west-2/rekognition/
aws4_request,
  SignedHeaders=content-type;host;x-amz-date;x-amz-target, Signature=XXXXXXXX

{
  "FaceIds":[
     "11111111-2222-3333-4444-555555555555"
  ],
  "CollectionId":"examplemyphotos"
}
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/x-amz-json-1.1
Date: Thu, 05 Jan 2017 17:03:06 GMT
x-amzn-RequestId: d3c6f630-d368-11e6-96d5-039839f35287
Content-Length: 57
Connection: keep-alive

{
   "DeletedFaces":[
      "11111111-2222-3333-4444-555555555555"
   ]
}
```

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DetectFaces

Detects faces within an image 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.

You pass the input image either as base64-encoded image bytes or as a reference to an image in an Amazon S3 bucket. If you use the Amazon CLI to call Amazon Rekognition operations, passing image bytes is not supported. The image must be either a PNG or JPEG formatted file.

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. 33).

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. 109)

An array of facial attributes you want to be returned. This can be the default list of attributes or all attributes. If you don't specify a value for `Attributes` or if you specify `["DEFAULT"]`, the API returns the following subset of facial attributes: `BoundingBox`, `Confidence`, `Pose`, `Quality` and `Landmarks`. If you provide `["ALL"]`, all facial attributes are returned but the operation will take longer to complete.

If you provide both, `["ALL", "DEFAULT"]`, the service uses a logical AND operator to determine which attributes to return (in this case, all attributes).

Type: Array of strings

Valid Values: `DEFAULT | ALL`

Required: No

Image (p. 109)

The input image as base64-encoded bytes or an S3 object. If you use the AWS CLI to call Amazon Rekognition operations, passing base64-encoded image bytes is not supported.

Type: Image (p. 183) object

Required: Yes

Response Syntax

```
{
   "FaceDetails": [
      {
         "AgeRange": {
            "High": number,
            "Low": number
         },
         "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. 110)

Details of each face found in the image.

Type: Array of FaceDetail (p. 177) objects

OrientationCorrection (p. 110)

The orientation of the input image (counter-clockwise direction). If your application displays the image, you can use this value to correct image orientation. The bounding box coordinates returned in `FaceDetails` represent face locations before the image orientation is corrected.

> **Note**
> If the input image is in .jpeg format, it might contain exchangeable image (Exif) metadata that includes the image's orientation. If so, and the Exif metadata for the input image populates the orientation field, the value of `OrientationCorrection` is null and the `FaceDetails` bounding box coordinates represent face locations after Exif metadata is used to correct the image orientation. Images in .png format don't contain Exif metadata.

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. 194).

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

Example

Example Request

The following example shows a request that detects faces in an image (crowd.jpg) stored in an Amazon S3 bucket (example-photos).

Sample Request

```
POST https://rekognition.us-west-2.amazonaws.com/ HTTP/1.1
Host: rekognition.us-west-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 77
X-Amz-Target: RekognitionService.DetectFaces
X-Amz-Date: 20170104T233701Z
User-Agent: aws-cli/1.11.25 Python/2.7.9 Windows/8 botocore/1.4.82
Content-Type: application/x-amz-json-1.1
Authorization: AWS4-HMAC-SHA256 Credential=XXXXXXXX/20170104/us-west-2/rekognition/
aws4_request,
  SignedHeaders=content-type;host;x-amz-date;x-amz-target, Signature=XXXXXXXX
{
   "Image":{
      "S3Object":{
         "Bucket":"example-photos",
         "Name":"crowd.jpg"
      }
   }
}
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/x-amz-json-1.1
Date: Wed, 04 Jan 2017 23:37:03 GMT
x-amzn-RequestId: b1827570-d2d6-11e6-a51e-73b99a9bb0b9
Content-Length: 1355
Connection: keep-alive

{
   "FaceDetails":[
      {
         "BoundingBox":{
            "Height":0.18000000715255737,
            "Left":0.5555555820465088,
            "Top":0.33666667342185974,
            "Width":0.23999999463558197
         },
         "Confidence":100.0,
         "Landmarks":[
            {
               "Type":"eyeLeft",
               "X":0.6394737362861633,
               "Y":0.40819624066352844
            },
            {
               "Type":"eyeRight",
               "X":0.7266660928726196,
               "Y":0.41039225459098816
            },
            {
               "Type":"nose",
               "X":0.6912462115287781,
               "Y":0.44240960478782654
            },
            {
               "Type":"mouthLeft",
               "X":0.6306198239326477,
               "Y":0.46700039505958557
            },
            {
               "Type":"mouthRight",
               "X":0.7215608954429626,
               "Y":0.47114261984825134
            }
         ],
         "Pose":{
            "Pitch":4.050806522369385,
            "Roll":0.9950747489929199,
            "Yaw":13.693790435791016
         },
         "Quality":{
            "Brightness":37.60169982910156,
            "Sharpness":80.0
         }
      },
      {
         "BoundingBox":{
            "Height":0.16555555164813995,
            "Left":0.3096296191215515,
            "Top":0.7066666483879089,
            "Width":0.22074073553085327
         },
         "Confidence":99.99998474121094,
```

```
         "Landmarks":[
            {
               "Type":"eyeLeft",
               "X":0.3767718970775604,
               "Y":0.7863991856575012
            },
            {
               "Type":"eyeRight",
               "X":0.4517287313938141,
               "Y":0.7715709209442139
            },
            {
               "Type":"nose",
               "X":0.4200106561183929 4,
               "Y":0.8192070126533508
            },
            {
               "Type":"mouthLeft",
               "X":0.3915625810623169,
               "Y":0.8374140858650208
            },
            {
               "Type":"mouthRight",
               "X":0.46825936436653137,
               "Y":0.823401689529419
            }
         ],
         "Pose":{
            "Pitch":-16.320178985595703,
            "Roll":-15.097439765930176,
            "Yaw":-5.771541118621826
         },
         "Quality":{
            "Brightness":31.440860748291016,
            "Sharpness":60.000003814697266
         }
      }
   ],
   "OrientationCorrection":"ROTATE_0"
}
```


See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

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. 31).

You pass the input image as base64-encoded image bytes or as a reference to an image in an Amazon S3 bucket. If you use the Amazon CLI to call Amazon Rekognition operations, passing image bytes is not supported. The image must be either a PNG or JPEG formatted file.

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.

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. 109) 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. 115)

The input image as base64-encoded bytes or an S3 object. If you use the AWS CLI to call Amazon Rekognition operations, passing base64-encoded image bytes is not supported.

Type: Image (p. 183) object

Required: Yes

MaxLabels (p. 115)

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. 115)

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. 116)

An array of labels for the real-world objects detected.

Type: Array of Label (p. 185) objects

OrientationCorrection (p. 116)

The orientation of the input image (counter-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.

Note
If the input image Exif metadata populates the orientation field, Amazon Rekognition does not perform orientation correction and the value of OrientationCorrection will be null.

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. 194).

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

Example

Example Request

The following example shows a request that detects labels in an image (skateboard.jpg) stored in an Amazon S3 bucket (example-photos).

Sample Request

```
POST https://rekognition.us-west-2.amazonaws.com/ HTTP/1.1
Host: rekognition.us-west-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 91
X-Amz-Target: RekognitionService.DetectLabels
X-Amz-Date: 20170104T233405Z
User-Agent: aws-cli/1.11.25 Python/2.7.9 Windows/8 botocore/1.4.82
Content-Type: application/x-amz-json-1.1
Authorization: AWS4-HMAC-SHA256 Credential=XXXXXXXX/20170104/us-west-2/rekognition/
aws4_request,
  SignedHeaders=content-type;host;x-amz-date;x-amz-target, Signature=XXXXXXXX

{
   "Image":{
      "S3Object":{
         "Bucket":"example-photos",
         "Name":"skateboard.jpg"
      }
   }
}
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/x-amz-json-1.1
Date: Wed, 04 Jan 2017 23:34:28 GMT
x-amzn-RequestId: 54cd6508-d2d6-11e6-bbda-cfd624bc06b2
Content-Length: 1882
Connection: keep-alive

{
   "Labels":[
      {
         "Confidence":99.25072479248047,
         "Name":"People"
      },
      {
         "Confidence":99.25074005126953,
         "Name":"Person"
      },
      {
         "Confidence":99.2429428100586,
         "Name":"Human"
      },
      {
         "Confidence":99.23795318603516,
         "Name":"Skateboard"
      },
```

```
    {
       "Confidence":99.23795318603516,
       "Name":"Sport"
    },
    {
       "Confidence":97.44398498535156,
       "Name":"Parking"
    },
    {
       "Confidence":97.44398498535156,
       "Name":"Parking Lot"
    },
    {
       "Confidence":87.81458282470703,
       "Name":"Automobile"
    },
    {
       "Confidence":87.81458282470703,
       "Name":"Car"
    },
    {
       "Confidence":87.81458282470703,
       "Name":"Vehicle"
    },
    {
       "Confidence":82.21033477783203,
       "Name":"Sedan"
    },
    {
       "Confidence":78.62909698486328,
       "Name":"Boardwalk"
    },
    {
       "Confidence":78.62909698486328,
       "Name":"Path"
    },
    {
       "Confidence":78.62909698486328,
       "Name":"Pavement"
    },
    {
       "Confidence":78.62909698486328,
       "Name":"Sidewalk"
    },
    {
       "Confidence":78.62909698486328,
       "Name":"Walkway"
    },
    {
       "Confidence":76.63581085205078,
       "Name":"Intersection"
    },
    {
       "Confidence":76.63581085205078,
       "Name":"Road"
    },
    {
       "Confidence":71.48307800292969,
       "Name":"Coupe"
    },
    {
       "Confidence":71.48307800292969,
       "Name":"Sports Car"
    },
    {
       "Confidence":67.8428726196289,
```

```
      "Name":"Building"
   },
   {
      "Confidence":62.91515350341797,
      "Name":"City"
   },
   {
      "Confidence":62.91515350341797,
      "Name":"Downtown"
   },
   {
      "Confidence":62.91515350341797,
      "Name":"Urban"
   },
   {
      "Confidence":62.04115676879883,
      "Name":"Neighborhood"
   },
   {
      "Confidence":62.04115676879883,
      "Name":"Town"
   },
   {
      "Confidence":61.2546272277832,
      "Name":"Suv"
   },
   {
      "Confidence":56.249610900878906,
      "Name":"Street"
   },
   {
      "Confidence":53.987510681152344,
      "Name":"Metropolis"
   },
   {
      "Confidence":52.98323059082031,
      "Name":"Housing"
   },
   {
      "Confidence":52.358848571777344,
      "Name":"Office Building"
   },
   {
      "Confidence":51.10673904418945,
      "Name":"Engine"
   },
   {
      "Confidence":51.10673904418945,
      "Name":"Machine"
   },
   {
      "Confidence":51.10673904418945,
      "Name":"Motor"
   },
   {
      "Confidence":51.06093215942383,
      "Name":"Apartment Building"
   },
   {
      "Confidence":51.06093215942383,
      "Name":"High Rise"
   },
   {
      "Confidence":50.64869689941406,
      "Name":"Pedestrian"
   }
```

```
    ]
}
```

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

DetectModerationLabels

Detects explicit or suggestive adult content in a specified JPEG or PNG format image. Use `DetectModerationLabels` to moderate images depending on your requirements. For example, you might want to filter images that contain nudity, but not images containing suggestive content.

To filter images, use the labels returned by `DetectModerationLabels` to determine which types of content are appropriate. For information about moderation labels, see Moderating Images (p. 55).

You pass the input image either as base64-encoded image bytes or as a reference to an image in an Amazon S3 bucket. If you use the Amazon CLI to call Amazon Rekognition operations, passing image bytes is not supported. The image must be either a PNG or JPEG formatted file.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

Image (p. 122)

The input image as base64-encoded bytes or an S3 object. If you use the AWS CLI to call Amazon Rekognition operations, passing base64-encoded image bytes is not supported.

Type: Image (p. 183) object

Required: Yes

MinConfidence (p. 122)

Specifies the minimum confidence level for the labels to return. Amazon Rekognition doesn't return any labels with a confidence level lower than this specified value.

If you don't specify `MinConfidence`, the operation returns labels with 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

```
{
```

```
   "ModerationLabels": [
      {
         "Confidence": number,
         "Name": "string",
         "ParentName": "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.

ModerationLabels (p. 122)

An array of labels for explicit or suggestive adult content found in the image. The list includes the top-level label and each second-level label detected in the image. This is useful for filtering specific categories of content.

Type: Array of ModerationLabel (p. 187) objects

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. 194).

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

Examples

Example Request

The following example shows the request for a `DetectModerationLabels` API operation.

Sample Request

```
POST / HTTP/1.1
Host: rekognition.us-west-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 77
X-Amz-Target: RekognitionService.DetectModerationLabels
X-Amz-Date: 20170424T195840Z
User-Agent: aws-cli/1.11.44 Python/2.7.6 Linux/4.2.0-42-generic botocore/1.5.7
Content-Type: application/x-amz-json-1.1
Authorization: AWS4-HMAC-SHA256 Credential=XXXXXXXXXXXXXXXXXXXXXXXX/20170424/us-west-2/
rekognition/aws4_request,
SignedHeaders=content-type;host;x-amz-date;x-amz-target,
 Signature=XXXXXXXXXXXXXXXXXXXXXXXX
{
   "Image":{
      "S3Object":{
         "Bucket":"example-photos",
         "Name":"input.jpg"
      }
   }
}
```

Example Response

The following example shows the response of a call to `DetectmoderationLabels`.

Sample Response

```
{
    "ModerationLabels": [
        {
            "Confidence": 79.03318786621094,
            "ParentName": "",
            "Name": "Explicit Nudity"
        },
        {
            "Confidence": 79.03318786621094,
            "ParentName": "Explicit Nudity",
            "Name": "Graphic Male Nudity"
```

```
        },
        {
            "Confidence": 68.99967956542969,
            "ParentName": "Explicit Nudity",
            "Name": "Sexual Activity"
        }
    ]
}
```

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

GetCelebrityInfo

Gets the name and additional information about a celebrity based on his or her Rekognition ID. The additional information is returned as an array of URLs. If there is no additional information about the celebrity, this list is empty. For more information, see Recognizing Celebrities (p. 49).

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

Request Syntax

```
{
   "Id": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

Id (p. 126)

The ID for the celebrity. You get the celebrity ID from a call to the RecognizeCelebrities (p. 146) operation, which recognizes celebrities in an image.

Type: String

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

Required: Yes

Response Syntax

```
{
   "Name": "string",
   "Urls": [ "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.

Name (p. 126)

The name of the celebrity.

Type: String

Urls (p. 126)

An array of URLs pointing to additional celebrity information.

Type: Array of strings

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

Example

Example Request

The following example shows a request that gets information about a celebrity.

Sample Request

```
POST https://rekognition.us-west-2.amazonaws.com/ HTTP/1.1
Host: rekognition.us-west-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 18
X-Amz-Target: RekognitionService.GetCelebrityInfo
X-Amz-Date: 20170414T184757Z
User-Agent: aws-cli/1.11.47 Python/2.7.9 Windows/8 botocore/1.5.10
Content-Type: application/x-amz-json-1.1
Authorization: AWS4-HMAC-SHA256 Credential=XXXXX/us-west-2/rekognition/aws4_request,
 SignedHeaders=content-type;host;x-amz-date;x-amz-target,
   Signature=XXXXXXXXXXXXXXXXXXX

{"Id": "3Ir0du6"}
```

Sample Response

```
{
   "Name": "Jeff Bezos",
   "Urls": ["www.imdb.com/name/nm1757263"]
}
```

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

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. 152) and SearchFacesByImage (p. 157) 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. 140) 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.

The input image is passed either as base64-encoded image bytes or as a reference to an image in an Amazon S3 bucket. If you use the Amazon CLI to call Amazon Rekognition operations, passing image bytes is not supported. The image must be either a PNG or JPEG formatted file.

For an example, see Example 2: Storing Faces (p. 67).

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. 129)

The 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. 129)

An array of facial attributes that you want to be returned. This can be the default list of attributes or all attributes. If you don't specify a value for `Attributes` or if you specify `["DEFAULT"]`, the API returns the following subset of facial attributes: `BoundingBox`, `Confidence`, `Pose`, `Quality` and `Landmarks`. If you provide `["ALL"]`, all facial attributes are returned but the operation will take longer to complete.

If you provide both, `["ALL", "DEFAULT"]`, the service uses a logical AND operator to determine which attributes to return (in this case, all attributes).

Type: Array of strings

Valid Values: `DEFAULT | ALL`

Required: No

ExternalImageId (p. 129)

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. 129)

The input image as base64-encoded bytes or an S3 object. If you use the AWS CLI to call Amazon Rekognition operations, passing base64-encoded image bytes is not supported.

Type: Image (p. 183) 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": {
            "AgeRange": {
               "High": number,
               "Low": number
```

```
            },
            "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. 130)

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. 181) objects

OrientationCorrection (p. 130)

The orientation of the input image (counterclockwise direction). If your application displays the image, you can use this value to correct image orientation. The bounding box coordinates returned in `FaceRecords` represent face locations before the image orientation is corrected.

> **Note**
> If the input image is in jpeg format, it might contain exchangeable image (Exif) metadata. If so, and the Exif metadata populates the orientation field, the value of `OrientationCorrection` is null and the bounding box coordinates in `FaceRecords` represent face locations after Exif metadata is used to correct the image orientation. Images in .png format don't contain Exif metadata.

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. 194).

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

Example

Example Request

The following example shows a request that adds an image (people.jpg) stored in an Amazon S3 bucket (examplephotos) to a collection (examplemyphotos).

Sample Request

```
POST https://rekognition.us-west-2.amazonaws.com/ HTTP/1.1
Host: rekognition.us-west-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 107
X-Amz-Target: RekognitionService.IndexFaces
X-Amz-Date: 20170105T162002Z
User-Agent: aws-cli/1.11.25 Python/2.7.9 Windows/8 botocore/1.4.82
Content-Type: application/x-amz-json-1.1
Authorization: AWS4-HMAC-SHA256 Credential=XXXXXXXX/20170105/us-west-2/rekognition/
aws4_request,
  SignedHeaders=content-type;host;x-amz-date;x-amz-target, Signature=XXXXXXXX

{
   "Image":{
      "S3Object":{
         "Bucket":"example-photos",
         "Name":"people.jpg"
      }
   },
   "CollectionId":"examplemyphotos"
}
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/x-amz-json-1.1
Date: Thu, 05 Jan 2017 16:20:04 GMT
```

```
x-amzn-RequestId: cfe5d2f3-d362-11e6-988e-1194c13fd971
Content-Length: 1889
Connection: keep-alive

{
   "FaceRecords":[
      {
         "Face":{
            "BoundingBox":{
               "Height":0.22206704318523407,
               "Left":0.503333330154419,
               "Top":0.21229049563407898,
               "Width":0.17666666209697723
            },
            "Confidence":99.9996566772461,
            "FaceId":"11111111-2222-3333-4444-555555555555",
            "ImageId":"11111111-2222-3333-4444-555555555555"
         },
         "FaceDetail":{
            "BoundingBox":{
               "Height":0.22206704318523407,
               "Left":0.503333330154419,
               "Top":0.21229049563407898,
               "Width":0.17666666209697723
            },
            "Confidence":99.9996566772461,
            "Landmarks":[
               {
                  "Type":"eyeLeft",
                  "X":0.5582929253578186,
                  "Y":0.327402263879776
               },
               {
                  "Type":"eyeRight",
                  "X":0.6097898483276367,
                  "Y":0.28622597455978394
               },
               {
                  "Type":"nose",
                  "X":0.6182368993759155,
                  "Y":0.34145522117614746
               },
               {
                  "Type":"mouthLeft",
                  "X":0.5820220708847046,
                  "Y":0.40035346150398254
               },
               {
                  "Type":"mouthRight",
                  "X":0.6310185194015503,
                  "Y":0.35822394490242004
               }
            ],
            "Pose":{
               "Pitch":-8.25561237335205,
               "Roll":-34.76542663574219,
               "Yaw":30.61958122253418
            },
            "Quality":{
               "Brightness":45.9112663269043,
               "Sharpness":50.0
            }
         }
      },
      {
         "Face":{
```

```
            "BoundingBox":{
               "Height":0.22067038714885712,
               "Left":0.402222216129303,
               "Top":0.3393854796886444,
               "Width":0.17555555701255798
            },
            "Confidence":99.9998779296875,
            "FaceId":"11111111-2222-3333-4444-555555555555",
            "ImageId":"11111111-2222-3333-4444-555555555555"
         },
         "FaceDetail":{
            "BoundingBox":{
               "Height":0.22067038714885712,
               "Left":0.402222216129303,
               "Top":0.3393854796886444,
               "Width":0.17555555701255798
            },
            "Confidence":99.9998779296875,
            "Landmarks":[
               {
                  "Type":"eyeLeft",
                  "X":0.46082764863967896,
                  "Y":0.4488079249858856
               },
               {
                  "Type":"eyeRight",
                  "X":0.509974479675293,
                  "Y":0.4110442101955414
               },
               {
                  "Type":"nose",
                  "X":0.5182068943977356,
                  "Y":0.4580079913139343
               },
               {
                  "Type":"mouthLeft",
                  "X":0.49137336015701294,
                  "Y":0.5153146386146545
               },
               {
                  "Type":"mouthRight",
                  "X":0.52939772605896,
                  "Y":0.4874058663845062
               }
            ],
            "Pose":{
               "Pitch":-12.197214126586914,
               "Roll":-33.81959533691406,
               "Yaw":32.57762908935547
            },
            "Quality":{
               "Brightness":32.43154525756836,
               "Sharpness":40.0
            }
         }
      }
   ],
   "OrientationCorrection":"ROTATE_0"
}
```


See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

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. 64).

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. 137)

Maximum number of collection IDs to return.

Type: Integer

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

Required: No

NextToken (p. 137)

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. 137)

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. 137)

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

Example

Example Request

The following example shows a request that lists the available collections.

Sample Request

```
POST https://rekognition.us-west-2.amazonaws.com/ HTTP/1.1
Host: rekognition.us-west-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 2
X-Amz-Target: RekognitionService.ListCollections
X-Amz-Date: 20170105T155800Z
User-Agent: aws-cli/1.11.25 Python/2.7.9 Windows/8 botocore/1.4.82
Content-Type: application/x-amz-json-1.1
Authorization: AWS4-HMAC-SHA256 Credential=XXXXXXXX/20170105/us-west-2/rekognition/
aws4_request,
  SignedHeaders=content-type;host;x-amz-date;x-amz-target, Signature=XXXXXXXX

{}
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/x-amz-json-1.1
Date: Thu, 05 Jan 2017 15:58:07 GMT
x-amzn-RequestId: bfe63e6c-d35f-11e6-840b-e97493937970
Content-Length: 45
Connection: keep-alive

{
   "CollectionIds":[
      "mycollection",
      "examplemyphotos"
   ]
}
```

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

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. 75).

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. 140)

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. 140)

Maximum number of faces to return.

Type: Integer

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

Required: No

NextToken (p. 140)

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. 140)

An array of `Face` objects.

Type: Array of Face (p. 175) objects

NextToken (p. 140)

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

Example

Example Request

The following example shows a request that lists the faces in a collection (examplemyphotos).

Sample Request

```
POST https://rekognition.us-west-2.amazonaws.com/ HTTP/1.1
Host: rekognition.us-west-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 28
X-Amz-Target: RekognitionService.ListFaces
X-Amz-Date: 20170104T232032Z
User-Agent: aws-cli/1.11.25 Python/2.7.9 Windows/8 botocore/1.4.82
Content-Type: application/x-amz-json-1.1
Authorization: AWS4-HMAC-SHA256 Credential=XXXXXXXX/20170104/us-west-2/rekognition/
aws4_request,
  SignedHeaders=content-type;host;x-amz-date;x-amz-target, Signature=XXXXXXXX

{"CollectionId": "examplemyphotos"}
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/x-amz-json-1.1
Date: Wed, 04 Jan 2017 23:20:33 GMT
x-amzn-RequestId: 63d76916-d2d4-11e6-aa8b-8bcb9b9028b4
Content-Length: 3440
Connection: keep-alive

{
   "Faces":[
      {
         "BoundingBox":{
            "Height":0.14222200214862823,
            "Left":-0.06083089858293533,
            "Top":0.2477779984474182,
            "Width":0.18991099298000336
         },
```

```
      "Confidence":99.99889373779297,
      "ExternalImageId":"externalimageidONE",
      "FaceId":"11111111-2222-3333-4444-555555555555",
      "ImageId":"11111111-2222-3333-4444-555555555555"
   },
   {
      "BoundingBox":{
         "Height":0.1644439995288849,
         "Left":0.4376850128173828,
         "Top":0.22555600106716156,
         "Width":0.22106799483299255
      },
      "Confidence":99.2842025756836,
      "ExternalImageId":"externalimageidTWO",
      "FaceId":"11111111-2222-3333-4444-555555555555",
      "ImageId":"11111111-2222-3333-4444-555555555555"
   },
   {
      "BoundingBox":{
         "Height":0.1599999964237213,
         "Left":0.643172025680542,
         "Top":0.11999999731779099,
         "Width":0.20998500287532806
      },
      "Confidence":99.99870300292969,
      "ExternalImageId":"externalimageidTHREE",
      "FaceId":"11111111-2222-3333-4444-555555555555",
      "ImageId":"11111111-2222-3333-4444-555555555555"
   },
   {
      "BoundingBox":{
         "Height":0.10555599629878998,
         "Left":0.5014839768409729,
         "Top":0.22777800261974335,
         "Width":0.14243300259113312
      },
      "Confidence":99.9761962890625,
      "ExternalImageId":"externalimageidONE",
      "FaceId":"11111111-2222-3333-4444-555555555555",
      "ImageId":"11111111-2222-3333-4444-555555555555"
   },
   {
      "BoundingBox":{
         "Height":0.10888899862766266,
         "Left":0.8056380152702332,
         "Top":0.21555599570274353,
         "Width":0.1454010009765625
      },
      "Confidence":99.94869995117188,
      "ExternalImageId":"externalimageidONE",
      "FaceId":"11111111-2222-3333-4444-555555555555",
      "ImageId":"11111111-2222-3333-4444-555555555555"
   },
   {
      "BoundingBox":{
         "Height":0.12999999523162842,
         "Left":0.5430560111999512,
         "Top":0.2544440031051636,
         "Width":0.16249999403953552
      },
      "Confidence":99.97720336914062,
      "ExternalImageId":"externalimageidFOUR",
      "FaceId":"11111111-2222-3333-4444-555555555555",
      "ImageId":"11111111-2222-3333-4444-555555555555"
   },
   {
```

```
      "BoundingBox":{
         "Height":0.2055560052394867,
         "Left":0.12166199833154678,
         "Top":-0.06888890266418457,
         "Width":0.27448099851608276
      },
      "Confidence":100.0,
      "ExternalImageId":"externalimageidFIVE",
      "FaceId":"11111111-2222-3333-4444-555555555555",
      "ImageId":"11111111-2222-3333-4444-555555555555"
   },
   {
      "BoundingBox":{
         "Height":0.1622219979763031,
         "Left":0.18942700326442719,
         "Top":0.008888890035450459,
         "Width":0.21292200684547424
      },
      "Confidence":99.96389770507812,
      "ExternalImageId":"externalimageidSIX",
      "FaceId":"11111111-2222-3333-4444-555555555555",
      "ImageId":"11111111-2222-3333-4444-555555555555"
   },
   {
      "BoundingBox":{
         "Height":0.11222200095653534,
         "Left":0.04154299944639206,
         "Top":0.01888890005648136,
         "Width":0.149851992726326
      },
      "Confidence":99.99949645996094,
      "ExternalImageId":"externalimageidSEVEN",
      "FaceId":"11111111-2222-3333-4444-555555555555",
      "ImageId":"11111111-2222-3333-4444-555555555555"
   },
   {
      "BoundingBox":{
         "Height":0.04222220182418823,
         "Left":0.46696001291275024,
         "Top":0.8944439888000488,
         "Width":0.05580030009150505
      },
      "Confidence":90.60900115966797,
      "ExternalImageId":"externalimageidSIX",
      "FaceId":"11111111-2222-3333-4444-555555555555",
      "ImageId":"11111111-2222-3333-4444-555555555555"
   },
   {
      "BoundingBox":{
         "Height":0.3222219944000244,
         "Left":0.24332299828529358,
         "Top":0.22333300113677979,
         "Width":0.43026700615882874
      },
      "Confidence":99.97339630126953,
      "ExternalImageId":"externalimageidFIVE",
      "FaceId":"11111111-2222-3333-4444-555555555555",
      "ImageId":"11111111-2222-3333-4444-555555555555"
   },
   {
      "BoundingBox":{
         "Height":0.09666670113801956,
         "Left":0.04154299944639206,
         "Top":0.05444439873099327,
         "Width":0.12907999753952026
      },
```

```
            "Confidence":99.99909973144531,
            "ExternalImageId":"externalimageidONE",
            "FaceId":"11111111-2222-3333-4444-555555555555",
            "ImageId":"11111111-2222-3333-4444-555555555555"
        }
    ]
}
```

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

RecognizeCelebrities

Returns an array of celebrities recognized in the input image. For more information, see Recognizing Celebrities (p. 49).

`RecognizeCelebrities` returns the 15 largest faces in the image. It lists recognized celebrities in the `CelebrityFaces` list and unrecognized faces in the `UnrecognizedFaces` list. The operation doesn't return celebrities whose face sizes are smaller than the largest 15 faces in the image.

For each celebrity recognized, the API returns a `Celebrity` object. The `Celebrity` object contains the celebrity name, ID, URL links to additional information, match confidence, and a `ComparedFace` object that you can use to locate the celebrity's face on the image.

Rekognition does not retain information about which images a celebrity has been recognized in. Your application must store this information and use the `Celebrity` ID property as a unique identifier for the celebrity. If you don't store the celebrity name or additional information URLs returned by `RecognizeCelebrities`, you will need the ID to identify the celebrity in a call to the GetCelebrityInfo (p. 126) operation.

You pass the imput image either as base64-encoded image bytes or as a reference to an image in an Amazon S3 bucket. If you use the Amazon CLI to call Amazon Rekognition operations, passing image bytes is not supported. The image must be either a PNG or JPEG formatted file.

For an example, see Recognizing Celebrities in an Image (p. 51).

This operation requires permissions to perform the `rekognition:RecognizeCelebrities` operation.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

Image (p. 146)

The input image as base64-encoded bytes or an S3 object. If you use the AWS CLI to call Amazon Rekognition operations, passing base64-encoded image bytes is not supported.

Type: Image (p. 183) object

Required: Yes

Response Syntax

```
{
   "CelebrityFaces": [
      {
```

```
         "Face": {
            "BoundingBox": {
               "Height": number,
               "Left": number,
               "Top": number,
               "Width": number
            },
            "Confidence": number,
            "Landmarks": [
               {
                  "Type": "string",
                  "X": number,
                  "Y": number
               }
            ],
            "Pose": {
               "Pitch": number,
               "Roll": number,
               "Yaw": number
            },
            "Quality": {
               "Brightness": number,
               "Sharpness": number
            }
         },
         "Id": "string",
         "MatchConfidence": number,
         "Name": "string",
         "Urls": [ "string" ]
      }
   ],
   "OrientationCorrection": "string",
   "UnrecognizedFaces": [
      {
         "BoundingBox": {
            "Height": number,
            "Left": number,
            "Top": number,
            "Width": number
         },
         "Confidence": number,
         "Landmarks": [
            {
               "Type": "string",
               "X": number,
               "Y": number
            }
         ],
         "Pose": {
            "Pitch": number,
            "Roll": number,
            "Yaw": number
         },
         "Quality": {
            "Brightness": number,
            "Sharpness": 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.

CelebrityFaces (p. 146)

Details about each celebrity found in the image. Amazon Rekognition can detect a maximum of 15 celebrities in an image.

Type: Array of Celebrity (p. 168) objects

OrientationCorrection (p. 146)

The orientation of the input image (counterclockwise direction). If your application displays the image, you can use this value to correct the orientation. The bounding box coordinates returned in `CelebrityFaces` and `UnrecognizedFaces` represent face locations before the image orientation is corrected.

Note
If the input image is in .jpeg format, it might contain exchangeable image (Exif) metadata that includes the image's orientation. If so, and the Exif metadata for the input image populates the orientation field, the value of `OrientationCorrection` is null and the `CelebrityFaces` and `UnrecognizedFaces` bounding box coordinates represent face locations after Exif metadata is used to correct the image orientation. Images in .png format don't contain Exif metadata.

Type: String

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

UnrecognizedFaces (p. 146)

Details about each unrecognized face in the image.

Type: Array of ComparedFace (p. 169) objects

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. 194).

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

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

Example

Example Request

The following example recognizes celebrities in an image (image.jpg) stored in an Amazon S3 bucket (photo-collection).

Sample Request

```
 POST https://rekognition.us-west-2.amazonaws.com/ HTTP/1.1
 Host: rekognition.us-west-2.amazonaws.com
 Accept-Encoding: identity
 Content-Length: 83
 X-Amz-Target: RekognitionService.RecognizeCelebrities
 X-Amz-Date: 20170414T195420Z
 User-Agent: aws-cli/1.11.47 Python/2.7.9 Windows/8 botocore/1.5.10
 Content-Type: application/x-amz-json-1.1
 Authorization: AWS4-HMAC-SHA256 Credential=XXXX/20170414/us-west-2/rekognition/
aws4_request, SignedHeaders=content-type;host;x-amz-date;x-amz-target,
    Signature=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
{
    "Image": {
        "S3Object": {
            "Bucket": "photo-collection",
            "Name": "image.jpg"
        }
    }
}
```

Sample Response

```
{
```

```
"CelebrityFaces": [{
    "Face": {
        "BoundingBox": {
            "Height": 0.617123007774353,
            "Left": 0.15641026198863983,
            "Top": 0.10864841192960739,
            "Width": 0.3641025722026825
        },
        "Confidence": 99.99589538574219,
        "Landmarks": [{
            "Type": "eyeLeft",
            "X": 0.2837241291999817,
            "Y": 0.3637104034423828
        }, {
            "Type": "eyeRight",
            "X": 0.4091649055480957,
            "Y": 0.37378931045532227
        }, {
            "Type": "nose",
            "X": 0.35267341136932373,
            "Y": 0.49657556414604187
        }, {
            "Type": "mouthLeft",
            "X": 0.2786353826522827,
            "Y": 0.5455248355865479
        }, {
            "Type": "mouthRight",
            "X": 0.39566439390182495,
            "Y": 0.5597742199897766
        }],
        "Pose": {
            "Pitch": -7.749263763427734,
            "Roll": 2.004552125930786,
            "Yaw": 9.012002944946289
        },
        "Quality": {
            "Brightness": 32.69192123413086,
            "Sharpness": 99.9305191040039
        }
    },
    "Id": "3Ir0du6",
    "MatchConfidence": 98.0,
    "Name": "Jeff Bezos",
    "Urls": ["www.imdb.com/name/nm1757263"]
}],
"OrientationCorrection": "ROTATE_0",
"UnrecognizedFaces": [{
    "BoundingBox": {
        "Height": 0.5345501899719238,
        "Left": 0.48461538553237915,
        "Top": 0.16949152946472168,
        "Width": 0.3153846263885498
    },
    "Confidence": 99.92860412597656,
    "Landmarks": [{
        "Type": "eyeLeft",
        "X": 0.5863404870033264,
        "Y": 0.36940744519233704
    }, {
        "Type": "eyeRight",
        "X": 0.6999204754829407,
        "Y": 0.3769848346710205
    }, {
        "Type": "nose",
        "X": 0.6349524259567261,
        "Y": 0.4804527163505554
```

```
        }, {
            "Type": "mouthLeft",
            "X": 0.5872702598571777,
            "Y": 0.5535582304000854
        }, {
            "Type": "mouthRight",
            "X": 0.6952020525932312,
            "Y": 0.5600858926773071
        }],
        "Pose": {
            "Pitch": -7.386096477508545,
            "Roll": 2.304218292236328,
            "Yaw": -6.175624370574951
        },
        "Quality": {
            "Brightness": 37.16635513305664,
            "Sharpness": 99.884521484375
        }
    }]
}
```


See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

SearchFaces

For a given input face ID, searches for matching faces in the collection the face belongs to. You get a face ID when you add a face to the collection using the IndexFaces (p. 129) 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. 75).

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. 152)

ID of the collection the face belongs to.

Type: String

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

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

Required: Yes

FaceId (p. 152)

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. 152)

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. 152)

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. 153)

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

Type: Array of FaceMatch (p. 180) objects

SearchedFaceId (p. 153)

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

Example

Example Request

The following example shows a request that searches for occurrences of a face within the collection (examplemyphotos) the face belongs to.

Sample Request

```
POST https://rekognition.us-west-2.amazonaws.com/ HTTP/1.1
Host: rekognition.us-west-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 78
X-Amz-Target: RekognitionService.SearchFaces
X-Amz-Date: 20170105T163206Z
User-Agent: aws-cli/1.11.25 Python/2.7.9 Windows/8 botocore/1.4.82
Content-Type: application/x-amz-json-1.1
Authorization: AWS4-HMAC-SHA256 Credential=XXXXXXXX/20170105/us-west-2/rekognition/
aws4_request,
  SignedHeaders=content-type;host;x-amz-date;x-amz-target, Signature=XXXXXXXX

{
   "FaceId":"11111111-2222-3333-4444-555555555555",
   "CollectionId":"examplemyphotos"
}
```

Sample Response

```
HTTP/1.1 200 OK
```

```
Content-Type: application/x-amz-json-1.1
Date: Thu, 05 Jan 2017 16:32:08 GMT
x-amzn-RequestId: 80269055-d364-11e6-9e9d-9b236fb2a3ad
Content-Length: 1057
Connection: keep-alive

{
   "FaceMatches":[
      {
         "Face":{
            "BoundingBox":{
               "Height":0.1622219979763031,
               "Left":0.18942700326442719,
               "Top":0.008888890035450459,
               "Width":0.21292200684547424
            },
            "Confidence":99.96389770507812,
            "ExternalImageId":"externalimageidONE",
            "FaceId":"11111111-2222-3333-4444-555555555555",
            "ImageId":"11111111-2222-3333-4444-555555555555"
         },
         "Similarity":95.46002197265625
      },
      {
         "Face":{
            "BoundingBox":{
               "Height":0.09666670113801956,
               "Left":0.04154299944639206,
               "Top":0.05444439873099327,
               "Width":0.12907999753952026
            },
            "Confidence":99.99909973144531,
            "ExternalImageId":"externalimageidTWO",
            "FaceId":"22222222-2222-3333-4444-555555555555",
            "ImageId":"22222222-2222-3333-4444-555555555555"
         },
         "Similarity":92.89849090576172
      },
      {
         "Face":{
            "BoundingBox":{
               "Height":0.11222200095653534,
               "Left":0.04154299944639206,
               "Top":0.01888890005648136,
               "Width":0.149851992726326
            },
            "Confidence":99.99949645996094,
            "ExternalImageId":"externalimageidTHREE",
            "FaceId":"11111111-2222-3333-4444-555555555555",
            "ImageId":"11111111-2222-3333-4444-555555555555"
         },
         "Similarity":92.22525024414062
      }
   ],
   "SearchedFaceId":"11111111-2222-3333-4444-555555555555"
}
```

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

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. 129) operation, and then use the face IDs returned in subsequent calls to the SearchFaces (p. 152) 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.

You pass the input image either as base64-encoded image bytes or as a reference to an image in an Amazon S3 bucket. If you use the Amazon CLI to call Amazon Rekognition operations, passing image bytes is not supported. The image must be either a PNG or JPEG formatted file.

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. 75).

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. 157)

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. 157)

(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. 157)

The input image as base64-encoded bytes or an S3 object. If you use the AWS CLI to call Amazon Rekognition operations, passing base64-encoded image bytes is not supported.

Type: Image (p. 183) object

Required: Yes

MaxFaces (p. 157)

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. 158)

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

Type: Array of FaceMatch (p. 180) objects

SearchedFaceBoundingBox (p. 158)

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

Type: BoundingBox (p. 166) object

SearchedFaceConfidence (p. 158)

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. 194).

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

Example

Example Request

The following example shows a request that determines the largest face in the supplied image (people.img) and scans a collection (examplemyphotos) for matching faces.

Sample Request

```
POST https://rekognition.us-west-2.amazonaws.com/ HTTP/1.1
Host: rekognition.us-west-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 107
X-Amz-Target: RekognitionService.SearchFacesByImage
X-Amz-Date: 20170105T162552Z
User-Agent: aws-cli/1.11.25 Python/2.7.9 Windows/8 botocore/1.4.82
Content-Type: application/x-amz-json-1.1
Authorization: AWS4-HMAC-SHA256 Credential=XXXXXXXX/us-west-2/rekognition/aws4_request,
  SignedHeaders=content-type;host;x-amz-date;x-amz-target, Signature=XXXXXXXX

{
   "Image":{
      "S3Object":{
         "Bucket":"example-photos",
         "Name":"people.jpg"
      }
   },
   "CollectionId":"examplemyphotos"
}
```

Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/x-amz-json-1.1
Date: Thu, 05 Jan 2017 16:25:54 GMT
x-amzn-RequestId: a0c13bb0-d363-11e6-86be-1d11e90c3f85
Content-Length: 1460
Connection: keep-alive

{
   "FaceMatches":[
```

```
    {
       "Face":{
          "BoundingBox":{
             "Height":0.2220669984817505,
             "Left":0.5033329725265503,
             "Top":0.21229000389575958,
             "Width":0.1766670048236847
          },
          "Confidence":99.99970245361328,
          "FaceId":"11111111-2222-3333-4444-555555555555",
          "ImageId":"11111111-2222-3333-4444-555555555555"
       },
       "Similarity":100.0
    },
    {
       "Face":{
          "BoundingBox":{
             "Height":0.1622219979763031,
             "Left":0.18942700326442719,
             "Top":0.008888890035450459,
             "Width":0.21292200684547424
          },
          "Confidence":99.96389770507812,
          "ExternalImageId":"jadenoah2",
          "FaceId":"11111111-2222-3333-4444-555555555555",
          "ImageId":"11111111-2222-3333-4444-555555555555"
       },
       "Similarity":95.46002960205078
    },
    {
       "Face":{
          "BoundingBox":{
             "Height":0.09666670113801956,
             "Left":0.04154299944639206,
             "Top":0.05444439873099327,
             "Width":0.12907999753952026
          },
          "Confidence":99.99909973144531,
          "ExternalImageId":"jadegarrettelinoah",
          "FaceId":"11111111-2222-3333-4444-555555555555",
          "ImageId":"11111111-2222-3333-4444-555555555555"
       },
       "Similarity":92.89849090576172
    },
    {
       "Face":{
          "BoundingBox":{
             "Height":0.11222200095653534,
             "Left":0.04154299944639206,
             "Top":0.01888890005648136,
             "Width":0.149851992726326
          },
          "Confidence":99.99949645996094,
          "ExternalImageId":"noahjade",
          "FaceId":"11111111-2222-3333-4444-555555555555",
          "ImageId":"11111111-2222-3333-4444-555555555555"
       },
       "Similarity":92.22525787353516
    }
 ],
 "SearchedFaceBoundingBox":{
    "Height":0.22206704318523407,
    "Left":0.503333330154419,
    "Top":0.21229049563407898,
    "Width":0.17666666209697723
 },
```

```
   "SearchedFaceConfidence":99.9996566772461
}
```

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

Data Types

The following data types are supported:

- Sunglasses (p. 193)

AgeRange

Structure containing the estimated age range, in years, for a face.

Rekognition estimates an age-range for faces detected in the input image. Estimated age ranges can overlap; a face of a 5 year old may have an estimated range of 4-6 whilst the face of a 6 year old may have an estimated range of 4-8.

Contents

High

The highest estimated age.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

Low

The lowest estimated age.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go

- AWS SDK for Java
- AWS SDK for Ruby V2

Celebrity

Provides information about a celebrity recognized by the RecognizeCelebrities (p. 146) operation.

Contents

Face

Provides information about the celebrity's face, such as its location on the image.

Type: ComparedFace (p. 169) object

Required: No

Id

A unique identifier for the celebrity.

Type: String

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

Required: No

MatchConfidence

The confidence, in percentage, that Rekognition has that the recognized face is the celebrity.

Type: Float

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

Required: No

Name

The name of the celebrity.

Type: String

Required: No

Urls

An array of URLs pointing to additional information about the celebrity. If there is no additional information about the celebrity, this list is empty.

Type: Array of strings

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

ComparedFace

Provides face metadata for target image faces that are analysed by `CompareFaces` and `RecognizeCelebrities`.

Contents

BoundingBox

Bounding box of the face.

Type: BoundingBox (p. 166) 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

Landmarks

An array of facial landmarks.

Type: Array of Landmark (p. 186) objects

Required: No

Pose

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

Type: Pose (p. 190) object

Required: No

Quality

Identifies face image brightness and sharpness.

Type: ImageQuality (p. 184) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

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

Bounding box of the face.

Type: BoundingBox (p. 166) 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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

CompareFacesMatch

Provides information about a face in a target image that matches the source image face analysed by `CompareFaces`. The `Face` property contains the bounding box of the face in the target image. The `Similarity` property is the confidence that the source image face matches the face in the bounding box.

Contents

Face

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

Type: ComparedFace (p. 169) 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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

Face

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

Contents

BoundingBox

Bounding box of the face.

Type: BoundingBox (p. 166) 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 input 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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++

- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

FaceDetail

Structure containing attributes of the face that the algorithm detected.

Contents

AgeRange

The estimated age range, in years, for the face. Low represents the lowest estimated age and High represents the highest estimated age.

Type: AgeRange (p. 164) object

Required: No

Beard

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

Type: Beard (p. 165) object

Required: No

BoundingBox

Bounding box of the face.

Type: BoundingBox (p. 166) 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. 172) objects

Required: No

Eyeglasses

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

Type: Eyeglasses (p. 173) 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. 174) object

Required: No

Gender

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

Type: Gender (p. 182) object

Required: No

Landmarks

Indicates the location of landmarks on the face.

Type: Array of Landmark (p. 186) 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. 188) object

Required: No

Mustache

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

Type: Mustache (p. 189) object

Required: No

Pose

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

Type: Pose (p. 190) object

Required: No

Quality

Identifies image brightness and sharpness.

Type: ImageQuality (p. 184) object

Required: No

Smile

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

Type: Smile (p. 192) object

Required: No

Sunglasses

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

Type: Sunglasses (p. 193) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

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. 175) 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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

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 input image, and external image ID that you assigned.

Type: Face (p. 175) object

Required: No

FaceDetail

Structure containing attributes of the face that the algorithm detected.

Type: FaceDetail (p. 177) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

Image

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

You pass image bytes to a Rekognition API operation by using the `Bytes` property. For example, you would use the `Bytes` property to pass an image loaded from a local file system. Image bytes passed by using the `Bytes` property must be base64-encoded. Your code may not need to encode image bytes if you are using an AWS SDK to call Rekognition API operations. For more information, see Example 4: Supplying Image Bytes to Amazon Rekognition Operations (p. 79).

You pass images stored in an S3 bucket to a Rekognition API operation by using the `S3Object` property. Images stored in an S3 bucket do not need to be base64-encoded.

The region for the S3 bucket containing the S3 object must match the region you use for Amazon Rekognition operations.

If you use the Amazon CLI to call Amazon Rekognition operations, passing image bytes using the Bytes property is not supported. You must first upload the image to an Amazon S3 bucket and then call the operation using the S3Object property.

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. 43).

Contents

Bytes

Blob of image bytes up to 5 MBs.

Type: Base64-encoded binary data object

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

Required: No

S3Object

Identifies an S3 object as the image source.

Type: S3Object (p. 191) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

ImageQuality

Identifies face image brightness and sharpness.

Contents

Brightness

Value representing brightness of the face. The service returns a value between 0 and 100 (inclusive). A higher value indicates a brighter face image.

Type: Float

Required: No

Sharpness

Value representing sharpness of the face. The service returns a value between 0 and 100 (inclusive). A higher value indicates a sharper face image.

Type: Float

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

Label

Structure containing details about the detected label, including 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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

Landmark

Indicates the location of the landmark on the face.

Contents

Type

Type of the landmark.

Type: String

Valid Values: `eyeLeft | eyeRight | nose | mouthLeft | mouthRight | leftEyeBrowLeft | leftEyeBrowRight | leftEyeBrowUp | rightEyeBrowLeft | rightEyeBrowRight | rightEyeBrowUp | leftEyeLeft | leftEyeRight | leftEyeUp | leftEyeDown | rightEyeLeft | rightEyeRight | rightEyeUp | rightEyeDown | noseLeft | noseRight | mouthUp | mouthDown | leftPupil | rightPupil`

Required: No

X

x-coordinate from the top left of the landmark expressed as the ratio 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 ratio 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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

ModerationLabel

Provides information about a single type of moderated content found in an image. Each type of moderated content has a label within a hierarchical taxonomy. For more information, see Moderating Images (p. 55).

Contents

Confidence

Specifies the confidence that Amazon Rekognition has that the label has been correctly identified.

If you don't specify the `MinConfidence` parameter in the call to `DetectModerationLabels`, the operation returns labels with a confidence value greater than or equal to 50 percent.

Type: Float

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

Required: No

Name

The label name for the type of content detected in the image.

Type: String

Required: No

ParentName

The name for the parent label. Labels at the top-level of the hierarchy have the parent label `""`.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

Pose

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

S3Object

Provides the S3 bucket name and object name.

The region for the S3 bucket containing the S3 object must match the region you use for Amazon Rekognition operations.

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. 43).

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

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

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

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.

Document History for Amazon Rekognition

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

- **API version: 2017-02-09**
- **Latest documentation update:** April 19th, 2017

Change	Description	Date
Celebrity Recognition	Amazon Rekognition can now analyze images for celebrities. For more information, see Recognizing Celebrities (p. 49).	In this release
Image Moderation	Amazon Rekognition can now determine if an image contains explicit or suggestive adult content. For more information, see ??? (p. 55).	April 19th, 2017
Age Range for Detected Faces Aggregated Rekognition Metrics Pane	Amazon Rekognition now returns the estimated age range, in years, for faces detected by the Rekognition API. For more information, see AgeRange (p. 164). The Rekognition console now has a metrics pane showing activity graphs for an aggregate of Amazon CloudWatch metrics for Rekognition over a specified period of time. For more information, see Exercise 4: See Aggregated Metrics (Console) (p. 29).	February 9th, 2017

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*.