# Table of Contents

What Is Amazon Rekognition? ................................................................. 1
Are You a First-Time Amazon Rekognition User? ................................... 2
How It Works .......................................................................................... 4
  Types of Detection and Recognition .................................................. 4
    Labels ................................................................................................. 4
    Faces .................................................................................................. 5
    Face Search ....................................................................................... 5
    Persons ............................................................................................... 5
    Celebrities ......................................................................................... 5
    Unsafe Content .................................................................................. 5

Image and Video Operations .................................................................... 5
  Rekognition Image Operations ............................................................. 6
  Rekognition Video Operations ............................................................... 6
  Non-Storage and Storage Based Operations ......................................... 6
  Using the AWS SDK or HTTP to Call Amazon Rekognition API Operations ............................................ 6

Non-Storage and Storage API Operations ............................................. 7
  Non Storage Operations ...................................................................... 7
  Storage-Based API Operations ............................................................ 8

Model Versioning ................................................................................... 9

Getting Started ...................................................................................... 11
  Step 1: Set Up an Account .................................................................... 11
  Sign up for AWS ................................................................................ 11
  Create an IAM User .......................................................................... 12
  Next Step ........................................................................................... 12
  Step 2: Set up the AWS CLI ................................................................. 12
  Next Step ........................................................................................... 12
  Step 3: Getting Started Using the Console .......................................... 13
    Exercise 1: Detect Objects and Scenes (Console) ............................... 14
    Exercise 2: Analyze Faces (Console) .................................................... 18
    Exercise 3: Compare Faces (Console) ................................................. 21
    Exercise 4: See Aggregated Metrics (Console) .................................. 23
  Step 4: Getting Started Using the API ................................................. 23
  What's Next ......................................................................................... 25

Authentication and Access Control ....................................................... 26
  Authentication ................................................................................. 26
  Access Control ................................................................................ 27
  Overview of Managing Access .......................................................... 27
    Amazon Rekognition Resources and Operations ............................... 28
    Understanding Resource Ownership ................................................. 28
    Managing Access to Resources ....................................................... 28
    Specifying Policy Elements: Actions, Effects, and Principals .............. 30
    Specifying Conditions in a Policy ..................................................... 31

Using Identity-Based Policies (IAM Policies) ........................................ 31
  Permissions Required to Use the Amazon Rekognition Console  ............ 32
  AWS Managed (Predefined) Policies for Amazon Rekognition ............... 32
  Customer Managed Policy Examples ................................................. 32

Amazon Rekognition API Permissions Reference .................................. 33

Working with Images ............................................................................ 37
  Images ............................................................................................... 37
  Correcting Image Orientation ............................................................. 37

Best Practices for Working With Images .............................................. 38
  Rekognition Image Operation Latency ................................................. 38
  Recommendations for Facial Recognition Input Images ......................... 38

Using an Amazon S3 Bucket ................................................................. 38
Detecting and Analyzing Faces ................................................................. 85
Detecting Objects and Scenes ............................................................................. 81
Working with Streaming Videos ............................................................................... 68
Working with Stored Videos .................................................................................. 51
Types of Detection and Recognition ................................................................. 51
Recogniton Video API Overview ........................................................................... 51
Video Formats and Storage ................................................................................... 52
Searching for People ............................................................................................ 52
Calling Recogniton Video Operations .................................................................. 53
Starting Video Analysis ........................................................................................ 53
Getting the Completion Status of a Recogniton Video Analysis Request .......... 54
Getting Recogniton Video Analysis Results ....................................................... 55
Giving Access to Amazon SNS Topics ............................................................... 58
Giving Access to Multiple Amazon SNS Topics ............................................... 58
Giving Access to a Single Amazon SNS Topic .................................................. 59
Analyzing a Video (AWS CLI) ............................................................................ 59
Prerequisites ........................................................................................................ 60
Analyzing a Video (SDK for Java) ...................................................................... 61
Prerequisites ........................................................................................................ 60
Reference: Video Analysis Results Notification ............................................... 66
Working with Streaming Videos ........................................................................... 68
Recognizing Faces in a Streaming Video .......................................................... 68
Prerequisites ........................................................................................................ 69
Giving Access to Kinesis Streams ........................................................................ 69
Giving Access to Your Kinesis Video Streams and Kinesis Data Streams .......... 69
Giving Access to Individual Kinesis Streams ..................................................... 70
Starting Streaming Video Analysis ................................................................. 70
Creating the Recogniton Video Stream Processor ........................................... 72
Starting the Recogniton Video Stream Processor ............................................. 73
Using Stream Processors .................................................................................... 73
Streaming Video into Recogniton Video ........................................................... 75
Reading Analysis Results ................................................................................... 75
Reference: Kinesis Face Recognition Record .................................................... 77
JSON Record ....................................................................................................... 78
InputInformation ................................................................................................. 78
KinesisVideo ....................................................................................................... 79
StreamProcessorInformation ............................................................................ 79
FaceSearchResponse .......................................................................................... 79
DetectedFace ....................................................................................................... 80
MatchedFace ....................................................................................................... 80
Detecting Objects and Scenes ........................................................................... 81
Detecting Labels in an Image .............................................................................. 81
Detecting Labels in a Video ................................................................................ 83
Detecting and Analyzing Faces .......................................................................... 85
Detect Faces in an Image ..................................................................................... 86
Compare Faces in Images .................................................................................... 89
Detecting Faces in a Stored Video ................................................................. 91
Using Recogniton Video Face Operations with Video ..................................... 92
Detecting Faces in an Image (SDK) ................................................................. 95
Comparing Faces in Images (SDK) ................................................................. 97
Detecting Faces in a Stored Video (SDK for Java) ........................................... 100
Prerequisites ...................................................................................................... 100
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Types</td>
<td>313</td>
</tr>
<tr>
<td>AgeRange</td>
<td>315</td>
</tr>
<tr>
<td>Beard</td>
<td>316</td>
</tr>
<tr>
<td>BoundingBox</td>
<td>317</td>
</tr>
<tr>
<td>Celebrity</td>
<td>319</td>
</tr>
<tr>
<td>CelebrityDetail</td>
<td>320</td>
</tr>
<tr>
<td>CelebrityRecognition</td>
<td>322</td>
</tr>
<tr>
<td>ComparedFace</td>
<td>323</td>
</tr>
<tr>
<td>ComparedSourceImageFace</td>
<td>324</td>
</tr>
<tr>
<td>CompareFacesMatch</td>
<td>325</td>
</tr>
<tr>
<td>ContentModerationDetection</td>
<td>326</td>
</tr>
<tr>
<td>Emotion</td>
<td>327</td>
</tr>
<tr>
<td>Eyeglasses</td>
<td>328</td>
</tr>
<tr>
<td>EyeOpen</td>
<td>329</td>
</tr>
<tr>
<td>Face</td>
<td>330</td>
</tr>
<tr>
<td>FaceDetail</td>
<td>332</td>
</tr>
<tr>
<td>FaceDetection</td>
<td>335</td>
</tr>
<tr>
<td>FaceMatch</td>
<td>336</td>
</tr>
<tr>
<td>FaceRecord</td>
<td>337</td>
</tr>
<tr>
<td>FaceSearchSettings</td>
<td>338</td>
</tr>
<tr>
<td>Gender</td>
<td>339</td>
</tr>
<tr>
<td>Geometry</td>
<td>340</td>
</tr>
<tr>
<td>HTTP Headers</td>
<td>170</td>
</tr>
<tr>
<td>Actions</td>
<td>170</td>
</tr>
<tr>
<td>CompareFaces</td>
<td>172</td>
</tr>
<tr>
<td>CreateCollection</td>
<td>180</td>
</tr>
<tr>
<td>CreateStreamProcessor</td>
<td>183</td>
</tr>
<tr>
<td>DeleteCollection</td>
<td>186</td>
</tr>
<tr>
<td>DeleteFaces</td>
<td>189</td>
</tr>
<tr>
<td>DeleteStreamProcessor</td>
<td>192</td>
</tr>
<tr>
<td>DescribeStreamProcessor</td>
<td>194</td>
</tr>
<tr>
<td>DetectFaces</td>
<td>198</td>
</tr>
<tr>
<td>DetectLabels</td>
<td>205</td>
</tr>
<tr>
<td>DetectModerationLabels</td>
<td>212</td>
</tr>
<tr>
<td>DetectText</td>
<td>216</td>
</tr>
<tr>
<td>GetCelebrityInfo</td>
<td>219</td>
</tr>
<tr>
<td>GetCelebrityRecognition</td>
<td>222</td>
</tr>
<tr>
<td>GetContentModeration</td>
<td>227</td>
</tr>
<tr>
<td>GetFaceDetection</td>
<td>231</td>
</tr>
<tr>
<td>GetFaceSearch</td>
<td>236</td>
</tr>
<tr>
<td>GetLabelDetection</td>
<td>241</td>
</tr>
<tr>
<td>GetPersonTracking</td>
<td>245</td>
</tr>
<tr>
<td>IndexFaces</td>
<td>250</td>
</tr>
<tr>
<td>ListCollections</td>
<td>258</td>
</tr>
<tr>
<td>ListFaces</td>
<td>262</td>
</tr>
<tr>
<td>ListStreamProcessors</td>
<td>268</td>
</tr>
<tr>
<td>RecognizeCelebrities</td>
<td>271</td>
</tr>
<tr>
<td>SearchFaces</td>
<td>277</td>
</tr>
<tr>
<td>SearchFacesByImage</td>
<td>282</td>
</tr>
<tr>
<td>StartCelebrityRecognition</td>
<td>288</td>
</tr>
<tr>
<td>StartContentModeration</td>
<td>291</td>
</tr>
<tr>
<td>StartFaceDetection</td>
<td>295</td>
</tr>
<tr>
<td>StartFaceSearch</td>
<td>299</td>
</tr>
<tr>
<td>StartLabelDetection</td>
<td>303</td>
</tr>
<tr>
<td>StartPersonTracking</td>
<td>307</td>
</tr>
<tr>
<td>StartStreamProcessor</td>
<td>310</td>
</tr>
<tr>
<td>StopStreamProcessor</td>
<td>312</td>
</tr>
</tbody>
</table>
What Is Amazon Rekognition?

Amazon Rekognition makes it easy to add image and video analysis to your applications. You just provide an image or video to the Rekognition API, and the service can identify objects, people, text, scenes, and activities. It can detect any inappropriate content as well. Amazon Rekognition also provides highly accurate facial analysis and facial recognition. You can detect, analyze, and compare faces for a wide variety of use cases, including user verification, cataloging, people counting, and public safety.

Amazon Rekognition is based on the same proven, highly scalable, deep learning technology developed by Amazon’s computer vision scientists to analyze billions of images and videos daily—and requires no machine learning expertise to use. Amazon Rekognition includes a simple, easy-to-use API that can quickly analyze any image or video file that’s stored in Amazon S3. Amazon Rekognition is always learning from new data, and we’re continually adding new labels and facial recognition features to the service.

Common use cases for using Amazon Rekognition include the following:

- **Searchable image and video libraries** – Amazon Rekognition makes images and stored videos searchable so you can discover objects and scenes that appear within them.

- **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 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 images, stored videos, and streaming videos for faces that match those stored in a container known as a face collection. 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.

- **Unsafe Content Detection** – Amazon Rekognition can detect explicit and suggestive adult content in images and in videos. 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 and in videos. Rekognition can recognize thousands of celebrities across a number of categories, such as politics, sports, business, entertainment, and media.
• **Text detection** – Amazon Rekognition Text in Image allows you to recognize and extract textual content from images. Text in Image supports most fonts including highly stylized ones. It detects text and numbers in different orientations such as those commonly found in banners and posters. In image sharing and social media applications, you can use it to enable visual search based on an index of images that contain the same keywords. In media and entertainment applications, you can catalogue videos based on relevant text on screen, such as ads, news, sport scores, and captions. Finally, in security and surveillance applications, you can identify vehicles based on license plate numbers from images taken by street cameras.

Some of the benefits of using Amazon Rekognition include:

• **Integrate powerful image and video recognition into your apps** – Amazon Rekognition removes the complexity of building image recognition capabilities into your applications by making powerful and accurate analysis available with a simple API. You don’t need computer vision or deep learning expertise to take advantage of Rekognition’s reliable image and video analysis. With Rekognition’s API, you can easily and quickly build image and video analysis into any web, mobile or connected device application.

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

• **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 pay for the images and videos 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. 4)** – 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. 11)** – In this section you set your account and test the Amazon Rekognition API.
3. **Working with Images (p. 37)** – This section provides information about using Amazon Rekognition with images stored in Amazon S3 buckets and images loaded from a local file system.

4. **Working with Stored Videos (p. 51)** – This section provides information about using Amazon Rekognition with videos stored in an Amazon S3 bucket.

5. **Working with Streaming Videos (p. 68)** – This section provides information about using Amazon Rekognition with streaming videos.
Amazon Rekognition provides two API sets; Rekognition Image for analyzing images and Rekognition Video for analyzing videos.

Both API perform detection and recognition analysis of images and videos to provide insights you can use in your applications. For example, you could use Rekognition Image to enhance the customer experience for a photo management application. When a customer uploads a photo, your application can use Rekognition Image to detect real-world objects or faces in the image. After your application stores the information returned from Rekognition Image, the user could then query their photo collection for photos with a specific object or face. Deeper querying is possible. For example, the user could query for faces that are smiling or query for faces that are a certain age.

For video applications, you could use Rekognition Video to create a surveillance application. Rekognition Video can track where a person is detected throughout a stored video. Alternatively, you can use Rekognition Video to search a streaming video for persons whose facial descriptions match facial descriptions already stored by Amazon Rekognition.

The Amazon Rekognition API makes deep learning image analysis easy to use. For example, RecognizeCelebrities (p. 271) returns information for up to 100 celebrities detected in an image. This includes information about where celebrity faces are detected on the image and where to get further information about the celebrity.

The following information covers the types of analysis that Amazon Rekognition provides and an overview of Rekognition Image and Rekognition Video operations. Also covered is the difference between non-storage and storage operations.

Topics
- Types of Detection and Recognition (p. 4)
- Image and Video Operations (p. 5)
- Non Storage and Storage API Operations (p. 7)
- Model Versioning (p. 9)

Types of Detection and Recognition

The following are the types of detection and recognition that the Rekognition Image API and Rekognition Video API can perform. For information about the APIs, see ??? (p. 5).

Labels

A label refers to any of the following: objects (for example, flower, tree, or table), events (for example, a wedding, graduation, or birthday party), concepts (for example, a landscape, evening, and nature) or activities (for example, getting out of a car). Amazon Rekognition can detect labels in images and videos. However activities are not detected in images. For more information, see Detecting Objects and Scenes (p. 81).

To detect labels in images, use DetectLabels (p. 205). To detect labels in stored videos, use StartLabelDetection (p. 303).
Faces

Amazon Rekognition can detect faces in images and stored videos. With Amazon Rekognition you can get information about where faces are detected in an image or video, facial landmarks such as the position of eyes, and detected emotions such as happy or sad. You can also compare a face in an image with faces detected in another image. Information about faces can also be stored for later retrieval. For more information, see Detecting and Analyzing Faces (p. 85).

To detect faces in images, use DetectFaces (p. 198). To detect faces in stored videos, use StartFaceDetection (p. 295).

Face Search

Amazon Rekognition can search for faces. Facial information is indexed into a container known as a collection. Face information in the collection can then be matched with faces detected in images, stored videos, and streaming video. For more information, Searching Faces in a Collection (p. 102).

To search for known faces in images, use DetectFaces (p. 198). To search for known faces in stored videos, use StartFaceDetection (p. 295). To search for known faces in streaming videos, use CreateStreamProcessor (p. 183).

Persons

Amazon Rekognition can track persons in a stored video. Rekognition Video provides tracking, face details, and in-frame location information for persons detected in a video. Persons cannot be detected in images. For more information, see Tracking People (p. 127).

To detect persons in stored videos, use StartPersonTracking (p. 307).

Celebrities

Amazon Rekognition can recognize thousand of celebrities in images and stored videos. You can get information about where a celebrity's face is located on an image, facial landmarks and the pose of a celebrity's face. You can get tracking information for celebrities as the appear throughout a stored video. You can also get further information about a recognize celebrity. For more information, see Recognizing Celebrities (p. 132).

To recognize celebrities in images, use RecognizeCelebrities (p. 271). To recognize celebrities in stored videos, use StartCelebrityRecognition (p. 288).

Unsafe Content

Amazon Rekognition can analyse images and stored videos for explicit or suggestive adult content. For more information, see Detecting Unsafe Content (p. 148).

To detect unsage images, use DetectModerationLabels (p. 212). To detect unsafe stored videos, use StartContentModeration (p. 291).

Image and Video Operations

Amazon Rekognition provides two API sets. Rekognition Image for analyzing images and Rekognition Video for analyzing stored and streaming videos. The following topic gives a brief overview of each API set.
The Rekognition Image and Rekognition Video API can detect or recognize a variety of entities such as faces or objects. For information about the types of recognition and detection that are supported, see ??? (p. 4).

**Rekognition Image Operations**

Amazon Rekognition image operations are synchronous. The input and response are in JSON format. Amazon Rekognition Image operations analyze an input image that is in .jpg or .png image format. The image passed to a Rekognition Image operation can be stored in an Amazon S3 bucket. If you are not using the AWS CLI, you can also pass byte64 encoded images bytes directly to an Amazon Rekognition operation. For more information, see Working with Images (p. 37).

**Rekognition Video Operations**

Rekognition Video can analyze videos stored in an Amazon S3 bucket and videos streamed through Amazon Kinesis Video Streams.

Rekognition Video video operations are asynchronous. With Rekognition Video storage video operations, you start analysis by calling the start operation for the type of analysis you want. For example, to detect faces in a stored video, call StartFaceDetection (p. 295). Once completed, Amazon Rekognition publishes the completion status to an Amazon SNS topic. To get the results of the analysis operation, you call the get operation for the type of analysis you requested. For example GetFaceDetection (p. 231). For more information, see Working with Stored Videos (p. 51).

With Rekognition Video streaming video operations you can search for faces stored in Rekognition Video collections. Rekognition Video analyzes a Kinesis video stream and outputs the search results to a Kinesis data stream. You manage video analysis by creating and using a Rekognition Video stream processor. For example, you create a stream processor by calling CreateStreamProcessor (p. 183). For more information, see Working with Streaming Videos (p. 68).

**Non-Storage and Storage Based Operations**

Amazon Rekognition operations are grouped into the following categories.

- **Non-storage API operations** – In these operations, Amazon Rekognition does not persist any information. You provide input images and videos, the operation performs the analysis, and returns results, but nothing is saved by Amazon Rekognition. For more information, see ??? (p. 7).

- **Storage-based API operations** – Amazon Rekognition servers can store detected facial information in containers known as collections. Amazon Rekognition provides additional API operations you can use to search the persisted face information for face matches. For more information, see Storage-Based API Operations (p. 8).

**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. 170) 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.
Non Storage and Storage API Operations

Amazon Rekognition provides two types of API operations. Non-storage operations in which no information is stored by Amazon Rekognition and Storage operations in which certain facial information is stored by Amazon Rekognition.

Non Storage Operations

Amazon Rekognition provides the following non-storage API operations for images:

- DetectLabels (p. 205)
- DetectFaces (p. 198)
- CompareFaces (p. 172)
- DetectModerationLabels (p. 212)
- RecognizeCelebrities (p. 271)
- DetectText (p. 216)
- DetectModerationLabels (p. 212)

Amazon Rekognition provides the following non-storage API operations for videos:

- StartLabelDetection (p. 303)
- StartFaceDetection (p. 295)
- StartPersonTracking (p. 307)
- StartCelebrityRecognition (p. 288)
- StartContentModeration (p. 291)

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:

<table>
<thead>
<tr>
<th>Label</th>
<th>ImageID</th>
</tr>
</thead>
<tbody>
<tr>
<td>tree</td>
<td>image-1</td>
</tr>
<tr>
<td>flower</td>
<td>image-1</td>
</tr>
<tr>
<td>mountain</td>
<td>image-1</td>
</tr>
<tr>
<td>tulip</td>
<td>image-2</td>
</tr>
<tr>
<td>flower</td>
<td>image-2</td>
</tr>
<tr>
<td>apple</td>
<td>image-3</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>ImageID</th>
<th>FaceID</th>
<th>FaceMetadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>image-1</td>
<td>face-1</td>
<td>&lt;boundingbox&gt;, etc.</td>
</tr>
<tr>
<td>image-1</td>
<td>face-2</td>
<td>&lt;boundingbox&gt;, etc.</td>
</tr>
<tr>
<td>image-1</td>
<td>face-3</td>
<td>&lt;boundingbox&gt;, etc.</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

**Storage-Based API Operations**

Rekognition Image supports the IndexFaces (p. 250) operation, which you can use to detect faces in an image and persist information about facial features detected in a Amazon Rekognition collection. This is an example of a storage-based API operation because the service persists information on the server.

Rekognition Image provides the following storage API operations:

- IndexFaces (p. 250)
- ListFaces (p. 262)
- SearchFacesByImage (p. 282)
- SearchFaces (p. 277)
- DeleteFaces (p. 189)

Rekognition Video provides the following storage API operations:

- StartFaceSearch (p. 299)
- CreateStreamProcessor (p. 183)

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. For example, you can detect the largest face in an image and search for matching faces in a collection by calling `searchFacesByImage`.

Facial information stored in collections by `IndexFaces` is accessible to Rekognition Video operations. For example you can search a video for persons whose faces match those in an existing collection by calling `StartFaceSearch` (p. 299).

For information about creating and managing collections, see Searching Faces in a Collection (p. 102).

**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.

**Example 1: An application that authenticates access to a building**

You start by creating a face collection to store scanned badge images using the `IndexFaces` operation, which extracts faces and stores them as searchable image vectors. Then, 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.

---

## Model Versioning

Amazon Rekognition uses deep learning models to perform face detection and to search for faces in collections. It continues to improve the accuracy of its models based on customer feedback and advances in deep learning research. These improvements are shipped as model updates. For example, with version 1.0 of the model, `IndexFaces` (p. 250) can index the 15 largest faces in an image. Later versions of the model allow `IndexFaces` to index the 100 largest faces in an image.

When you create a new collection, it's associated with the most recent version of the model. To improve accuracy, the model is occasionally updated.

When a new version of the model is released, the following happens:

- New collections you create are associated with the latest model. Faces that you add to new collections by using `IndexFaces` (p. 250) are detected using the latest model.
- Your existing collections to use the version of the model that they were created with. The face vectors stored in these collections aren't automatically updated to the latest version of the model.
- New faces that are added to an existing collection are detected by using the model that's already associated with the collection.

Different versions of the model aren't compatible with each other. Specifically, if an image is indexed into multiples collections that use different versions of the model, the face identifiers for the same detected faces are different. If an image is indexed into multiple collections that are associated with the same model, the face identifiers are the same.

Your application might face compatibility issues if your collection management doesn't account for updates to the model. You can determine the version of the model a collection uses by using the `FaceModelVersion` field that's returned by collection operation responses. For example, `CreateCollection`.

Existing face vectors in a collection can't be updated to a later version of the model. Because Amazon Rekognition doesn't store source image bytes, it can't automatically reindex images by using a later version of the model.
To use the latest model on faces that are stored in an existing collection, create a new collection (CreateCollection (p. 180)) and reindex the source images into the new collection (IndexFaces). You need to update any face identifiers that are stored by your application because the face identifiers in the new collection are different from the face identifiers in the old collection. If you no longer need the old collection, you can delete it by using DeleteCollection (p. 186).
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. 4).

Topics
- Step 1: Set Up an AWS Account and Create an Administrator User (p. 11)
- Step 2: Set Up the AWS Command Line Interface (AWS CLI) (p. 12)
- Step 3: Getting Started Using the Amazon Rekognition Console (p. 13)
- Step 4: Getting Started Using the API (p. 23)
- What’s Next (p. 25)

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. 11)
2. Create an IAM User (p. 12)

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 a different account, 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. 12)

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. 23), 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
```

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.

```bash
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.

```bash
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.

Next Step

Step 4: Getting Started Using the API (p. 23)

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. 4). 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. 23).

This section also shows you how to see aggregated Amazon CloudWatch metrics for Rekognition by using the Rekognition console.
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": 99.25359344482422,
            "Name": "Skateboard"
        }
    ]
}
```
Exercise 1: Detect Objects and Scenes (Console)
{
  "Confidence":62.04711151113047,
  "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":50.6806793368725586,
  "Name":"Pedestrian"
},
{
  "Confidence":50.59548568725586,
  "Name":"Freeway"
}
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. 4).

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 100 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 100 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 100 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 **Compare Faces in Images** (p. 89).

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 100 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. 168).

<table>
<thead>
<tr>
<th>Graph</th>
<th>Aggregated Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful calls</td>
<td>SuccessfulRequestCount</td>
</tr>
<tr>
<td>Client errors</td>
<td>UserErrorCount</td>
</tr>
<tr>
<td>Server errors</td>
<td>ServerErrorCount</td>
</tr>
<tr>
<td>Throttled</td>
<td>ThrottledCount</td>
</tr>
<tr>
<td>Detected labels</td>
<td>DetectedLabelCount</td>
</tr>
<tr>
<td>Detected faces</td>
<td>DetectedFaceCount</td>
</tr>
</tbody>
</table>

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. 26).

For more information about monitoring Rekognition with CloudWatch see Monitoring (p. 165).

**To see aggregated metrics (console)**

1. Open the Amazon Rekognition console at [https://console.aws.amazon.com/rekognition/](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 exercise you use the DetectLabels (p. 205) 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

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

   • Using the AWS SDK for Java.

   ```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()   
   ```
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).

```python
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']))
```

What's Next

Once you've completed the Getting Started section, go to the following sections to learn about using Amazon Rekognition with images, stored videos, and streaming videos.

• Working with Images (p. 37)
• Working with Stored Videos (p. 51)
• Working with Streaming Videos (p. 68)
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 (p. 26)
- Access Control (p. 27)

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 an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances 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 (p. 27)
- Using Identity-Based Policies (IAM Policies) for Amazon Rekognition (p. 31)
- Amazon Rekognition API Permissions: Actions, Permissions, and Resources Reference (p. 33)

### 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 (p. 28)
- Understanding Resource Ownership (p. 28)
- Managing Access to Resources (p. 28)
- Specifying Policy Elements: Actions, Effects, and Principals (p. 30)
Amazon Rekognition Resources and Operations

In Amazon Rekognition, there are resource types for a collection and a streamprocessor. 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.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>ARN Format</th>
</tr>
</thead>
</table>

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. 33).

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) (p. 29)
- Resource-Based Policies (p. 29)
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.

```json
"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. 31). 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.

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

For more information about using resource-based policies with Amazon Rekognition, see Using Resource-Based Policies (p. 31). For more information about using resources, see Using Resources with Amazon Rekognition (p. 30).
"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. 28) and Amazon Rekognition Amazon Rekognition API Permissions: Actions, Permissions, and Resources Reference (p. 33).

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. 28).
- **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. 33).

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. 27).

**Topics**
- Permissions Required to Use the Amazon Rekognition Console (p. 32)
- AWS Managed (Predefined) Policies for Amazon Rekognition (p. 32)
- Customer Managed Policy Examples (p. 32)

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.
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.
- **AmazonRekognitionReadOnlyAccess** – Grants read-only access to Amazon Rekognition resources.
- **AmazonRekognitionServiceRole** – Allows Amazon Rekognition to call Amazon Kinesis Data Streams and Amazon SNS services on your behalf.

**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. 32).

**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. 32)
- Example 2: Allow a User Full Access to Resources (p. 33)

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

The following example grants read-only access to Amazon Rekognition resources.
Example 2: Allow a User Full Access to Resources

The following example grants full access to Amazon Rekognition resources.

```json
{
    "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. 27) 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: CreateCollection**
- Required Permissions (API Action): `rekognition:CreateCollection`

**API Operation: DeleteCollection**
- Required Permissions (API Action): `rekognition:DeleteCollection`

**API Operation: DeleteFaces**
- Required Permissions (API Action): `rekognition:DeleteFaces`

**API Operation: DetectFaces**
- Required Permissions (API Action): `rekognition:DetectFaces`

**API Operation: IndexFaces**
- Required Permissions (API Action): `rekognition:IndexFaces`

**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`

**API Operation: SearchFaces**
- Required Permissions (API Action): `rekognition:SearchFaces`

**API Operation: SearchFacesByImage**
- Required Permissions (API Action): `rekognition:SearchFacesByImage`

**API Operation: CreateStreamProcessor**
- Required Permissions (API Action): `rekognition:CreateStreamProcessor`
API Operation: **DeleteStreamProcessor**

Required Permissions (API Action): rekognition:DeleteStreamProcessor


API Operation: **ListStreamProcessors**

Required Permissions (API Action): rekognition:ListStreamProcessors


API Operation: **StartStreamProcessor**

Required Permissions (API Action): rekognition:StartStreamProcessor


API Operation: **StopStreamProcessor**

Required Permissions (API Action): rekognition:StopStreamProcessor


API Operation: **CompareFaces**

Required Permissions (API Action): rekognition:CompareFaces

None used.

API Operation: **DetectFaces**

Required Permissions (API Action): rekognition:DetectFaces

None used.

API Operation: **DetectLabels**

Required Permissions (API Action): rekognition:DetectLabels

None used.

API Operation: **DetectModerationLabels**

Required Permissions (API Action): rekognition:DetectModerationLabels

None used.

API Operation: **DetectText**

Required Permissions (API Action): rekognition:DetectText

None used.

API Operation: **RecognizeCelebrities**

Required Permissions (API Action): rekognition:RecognizeCelebrities

None used.

API Operation:

Required Permissions (API Action): rekognition:

None used.
API Operation: GetCelebrityRecognition
   Required Permissions (API Action): rekognition:GetCelebrityRecognition
   None used.
API Operation: GetContentModeration
   Required Permissions (API Action): rekognition:getContentModeration
   None used.
API Operation: GetFaceDetection
   Required Permissions (API Action): rekognition:GetFaceDetection
   None used.
API Operation: GetLabelDetection
   Required Permissions (API Action): rekognition:GetLabelDetection
   None used.
API Operation: GetPersonTracking
   Required Permissions (API Action): rekognition:getPersonTracking
   None used.
API Operation: StartCelebrityRecognition
   Required Permissions (API Action): rekognition:StartCelebrityRecognition
   None used.
API Operation: StartContentModeration
   Required Permissions (API Action): rekognition:StartContentModeration
   None used.
API Operation: StartFaceDetection
   Required Permissions (API Action): rekognition:StartFaceDetection
   None used.
API Operation: StartLabelDetection
   Required Permissions (API Action): rekognition:StartLabelDetection
   None used.
API Operation: StartPersonTracking
   Required Permissions (API Action): rekognition:StartPersonTracking
   None used.
Working with Images

This section covers the types of detection and recognition that Rekognition Image can perform on images. Rekognition Image can perform the following analysis on images:

- Label detection
- Face detection and comparison
- Celebrity recognition
- Image moderation
- Text in image detection

These are performed by non-storage API operations where Rekognition Image doesn't persist any information discovered by the operation. No input image bytes are persisted by non-storage API operations. For more information, see Non Storage and Storage API Operations (p. 7).

Rekognition Image can also store facial metadata in collections for later retrieval. For more information, see Searching Faces in a Collection (p. 102).

In this section you use the Rekognition Image API operations to analyze images stored in an Amazon S3 bucket and images bytes loaded from the local file system. This section also covers getting image orientation information from a .jpg image.

Topics
- Images (p. 37)
- Best Practices for Working With Images (p. 38)
- Analysing Images Stored in an Amazon S3 Bucket (p. 38)
- Analysing an Image Loaded from a Local File System (p. 40)
- Getting Image Orientation and Bounding Box Coordinates (p. 44)

Images

Rekognition Image operations can analyse images in .jpg or .png format.

You pass image bytes to an Rekognition Image operation as part of the call or you reference an existing Amazon S3 object. For an example of analyzing an image stored in an Amazon S3 bucket, see Analysing Images Stored in an Amazon S3 Bucket (p. 38).

If you use HTTP and pass the image bytes as part of an Rekognition Image 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 Analysing an Image Loaded from a Local File System (p. 40).

If you use the AWS CLI to call Rekognition Image 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.

For information about ensuring the lowest possible latency for Rekognition Image operations, see ??? (p. 38).

Correcting Image Orientation

In several Rekognition API operations, the orientation of an analyzed image is returned. For example, the DetectFaces (p. 198) 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 Getting Image Orientation and Bounding Box Coordinates (p. 44).

Best Practices for Working With Images

The following are best practices for working with images.

Rekognition Image Operation Latency

To ensure the lowest possible latency for Rekognition Image operations, consider the following:

• The region for the Amazon S3 bucket containing your images must match the region you use for Rekognition Image API operations.
• Calling an Rekognition Image operation with image bytes is faster than uploading the image to an Amazon S3 bucket and then referencing the uploaded image in an Rekognition Image operation. Consider this approach if you are uploading images to Rekognition Image for near real-time processing. For example, images uploaded from an IP camera or images uploaded through a web portal.
• If the image is already in an Amazon S3 bucket, referencing it in an Rekognition Image operation will probably be faster than passing image bytes to the operation.
• Low resolution images are processed faster than high resolution images. Consider resizing input images to a lower resolution for faster processing.

Recommendations for Facial Recognition Input Images

Whilst Rekognition Image 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. Rekognition Image can detect faces as small as 40x40 pixels in an HD resolution image (1920x1080). Higher resolution images will need a larger minimum face size.
• Be bright and sharp. You can use DetectFaces (p. 198) to determine the brightness and sharpness of a face.
• Avoid occlusions such as head-bands or masks.

Analysing Images Stored in an Amazon S3 Bucket

Rekognition Image can analyze images stored in an Amazon S3 bucket or images supplied as image bytes.

In this topic you use the DetectLabels (p. 205) API operation to detect objects, concepts, and scenes in an image (JPEG or PNG) stored in an Amazon S3 bucket.
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

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

   • Using the AWS SDK for Java.

   ```java
   import com.amazonaws.servicesrekognition.AmazonRekognition;
   import com.amazonaws.servicesrekognition.AmazonRekognitionClientBuilder;
   import com.amazonaws.AmazonClientException;
   import com.amazonaws.auth.AWSCredentials;
   import com.amazonaws.auth.AWSStaticCredentialsProvider;
   import com.amazonaws.regions.Regions;
   import com.amazonaws.servicesrekognition.model.AmazonRekognitionException;
   import com.amazonaws.servicesrekognition.model.DetectLabelsRequest;
   import com.amazonaws.servicesrekognition.model.DetectLabelsResult;
   import com.amazonaws.servicesrekognition.model.Image;
   import com.amazonaws.servicesrekognition.model.Label;
   import com.amazonaws.servicesrekognition.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 files.
               " + "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','us-east-1')

    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']))

Analysing an Image Loaded from a Local File System

Rekognition Image operations can analyze images supplied as images bytes or images stored in an S3 bucket.

This topics provides AWS SDK examples of supplying image bytes to Rekognition Image 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. 341) input parameter. Within Image you specify the Bytes property to pass base64 encoded image bytes or you specify the S3Object (p. 357) 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 Rekognition Image operations, the image bytes must be a base64-encoded string. For more information, see *Working with Images (p. 37).*

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 (p. 41)
• Supplying Images: Using the Local File System and Python (p. 42)
• Supplying Images: Using the Local File System and PHP (p. 43)

## 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.

```java
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.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')

# end of file
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
<?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']
}
```
Getting Image Orientation and Bounding Box Coordinates

Applications that use Rekognition Image commonly need to display the images that are detected by Rekognition Image 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, Rekognition Image 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. Rekognition Image face detection operations return bounding box coordinates for each detected face.

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

- **CompareFaces** (p. 172)
- **DetectFaces** (p. 198)
- **DetectLabels** (p. 205) (returns only information to correct image orientation)
- **IndexFaces** (p. 250)
- **RecognizeCelebrities** (p. 271)

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 Rekognition Image 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, Rekognition Image 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, Rekognition Image 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 Rekognition Image 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. 317).

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, Rekognition Image 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 Rekognition Image 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 Rekognition Image operation providing an input image with at least one face and with no Exif metadata orientation. For an example, see Detecting Faces in an Image (SDK) (p. 95).
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 Rekognition Image operation.
- <face> represents information about the face that is returned by an Rekognition Image operation. For example, the FaceDetail (p. 332) data type that the DetectFaces (p. 198) 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. 317).

**ROTATE_0**

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

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

**ROTATE_90**

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

top = image.width * <face>.BoundingBox.Left
```
Example: Getting Image Orientation and Bounding Box Coordinates For an Image

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, Rekognition Image 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.

```java
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.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());

        ByteArrayInputStream baos = new ByteArrayInputStream(imageBytes.array());

        byte[] bytes = ImageIO.read(imageBytesStream, "jpg");

        int height = image.getHeight();
    }
}
```java
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 * box.getLeft();
    }
```
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())));
Working with Stored Videos

Amazon Rekognition Video is an API that you can use to analyze videos. With Rekognition Video, you can detect labels, faces, people, celebrities, and adult (suggestive and explicit) content in videos that are stored in an Amazon Simple Storage Service (Amazon S3) bucket. You can use Rekognition Video in categories such as media/entertainment and surveillance. Previously, scanning videos for objects or people would have taken many hours of error-prone viewing by a human being. Rekognition Video automates the detection of items and when they occur throughout a video.

This section covers the types of detection and recognition that Rekognition Video can perform, an overview of the API, and examples for using Rekognition Video.

Topics
- Types of Detection and Recognition (p. 51)
- Rekognition Video API Overview (p. 51)
- Calling Rekognition Video Operations (p. 53)
- Giving Rekognition Video Access to Your Amazon SNS Topics (p. 58)
- Analyzing a Video with the AWS Command Line Interface (p. 59)
- Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61)
- Reference: Video Analysis Results Notification (p. 66)

Types of Detection and Recognition

You can use Rekognition Video to analyze videos for the following information:

- Labels (p. 81)
- Faces (p. 85)
- People (p. 127)
- Celebrities (p. 132)
- Suggestive and explicit adult content (p. 148)

For more information, see Amazon Rekognition: How It Works (p. 4).

Rekognition Video API Overview

Rekognition Video processes a video that’s stored in an Amazon S3 bucket. The design pattern is an asynchronous set of operations. You start video analysis by calling a Start operation such as StartLabelDetection (p. 303). The completion status of the request is published to an Amazon Simple Notification Service (Amazon SNS) topic. To get the completion status from the Amazon SNS topic, you can use an Amazon Simple Queue Service (Amazon SQS) queue or an AWS Lambda function. After you have the completion status, you call a Get operation, such as GetLabelDetection (p. 241), to get the results of the request.
The following diagram shows the process for detecting labels in a video that's stored in an Amazon S3 bucket. In the diagram, an Amazon SQS queue gets the completion status from the Amazon SNS topic. Alternatively, you can use an AWS Lambda function.

![Diagram of video processing](image)

The process is the same for detecting faces and people. The following table lists the Start and Get operations for each of the non-storage Amazon Rekognition operations.

<table>
<thead>
<tr>
<th>Detection</th>
<th>Start Operation</th>
<th>Get Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>People</td>
<td>StartPersonTracking (p. 307)</td>
<td>GetPersonTracking (p. 245)</td>
</tr>
<tr>
<td>Faces</td>
<td>StartFaceDetection (p. 295)</td>
<td>GetFaceDetection (p. 231)</td>
</tr>
<tr>
<td>Labels</td>
<td>StartLabelDetection (p. 303)</td>
<td>GetLabelDetection (p. 241)</td>
</tr>
<tr>
<td>Celebrities</td>
<td>StartCelebrityRecognition (p. 288)</td>
<td>GetCelebrityRecognition (p. 222)</td>
</tr>
<tr>
<td>Explicit or suggestive adult content</td>
<td>StartContentModeration (p. 291)</td>
<td>GetContentModeration (p. 227)</td>
</tr>
</tbody>
</table>

For Get operations other than GetCelebrityRecognition, Rekognition Video returns tracking information for when entities are detected throughout the input video.

For more information about using Rekognition Video, see Calling Rekognition Video Operations (p. 53). For an example that does video analysis by using Amazon SQS, see Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61). For AWS CLI examples, see Analyzing a Video with the AWS Command Line Interface (p. 59).

### Video Formats and Storage

Amazon Rekognition operations can analyze videos that are stored in Amazon S3 buckets. The video must be encoded using the H.264 codec. The supported file formats are MPEG-4 and MOV.

A codec is software or hardware that compresses data for faster delivery and decompresses received data into its original form. The H.264 codec is commonly used for recording, compressing, and distributing video content. A video file format can contain one or more codecs. If your MOV or MPEG-4 format video file doesn't work with Rekognition Video, check that the codec used to encode the video is H.264.

The maximum file size for a stored video is 8GB.

### Searching for People

You can use facial metadata that's stored in a collection to search for people in a video. For example, you can search an archived surveillance video for a specific person or for multiple people. You store facial metadata from source images in a collection by using the IndexFaces (p. 250) operation. You can then use StartFaceSearch (p. 299) to start asynchronously searching for faces in the collection. You use GetFaceSearch (p. 236) to get the search results. For more information, see Searching for Faces with Rekognition Video (p. 121). Searching for people is an example of a storage-based Amazon Rekognition operation. For more information, see Storage-Based API Operations (p. 8).
You can also search for people in a streaming video. For more information, see Working with Streaming Videos (p. 68).

Calling Rekognition Video Operations

Rekognition Video is an asynchronous API that you can use to analyze videos that are stored in an Amazon Simple Storage Service (Amazon S3) bucket. You start the analysis of a video by calling a Rekognition Video Start operation, such as StartPersonTracking (p. 307). Rekognition Video publishes the result of the analysis request to an Amazon Simple Notification Service (Amazon SNS) topic. You can use an Amazon Simple Queue Service (Amazon SQS) queue or an AWS Lambda function to get the completion status of the video analysis request from the Amazon SNS topic. Finally, you get the video analysis request results by calling an Amazon Rekognition Get operation, such as GetPersonTracking (p. 245).

The information in the following sections uses label detection operations to show how Rekognition Video detects labels (objects, events, concepts, and activities) in a video that's stored in an Amazon S3 bucket. The same approach works for the other Rekognition Video operations—for example, StartFaceDetection (p. 295) and StartPersonTracking (p. 307). The AWS SDK for Java example Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61) shows how to analyze a video by using an Amazon SQS queue to get the completion status from the Amazon SNS topic. It's also used as a basis for other Rekognition Video examples, such as Tracking People through a Stored Video (SDK for Java) (p. 129). For AWS CLI examples, see Analyzing a Video with the AWS Command Line Interface (p. 59).

Topics

- Starting Video Analysis (p. 53)
- Getting the Completion Status of a Rekognition Video Analysis Request (p. 54)
- Getting Rekognition Video Analysis Results (p. 55)

Starting Video Analysis

You start a Rekognition Video label detection request by calling StartLabelDetection. The following is an example of a JSON request that's passed by StartLabelDetection.

```json
{
    "Video": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "video.mp4"
        }
    },
    "ClientRequestToken": "LabelDetectionToken",
    "MinConfidence": 50,
    "NotificationChannel": {
        "SNSTopicArn": "arn:aws:sns:us-east-1:nnnnnnnnnn:topic",
        "RoleArn": "arn:aws:iam::nnnnnnnnnn:role/roleopic"
    },
    "JobTag": "DetectingLabels"
}
```

Video provides the video file name and the Amazon S3 bucket to retrieve it from. NotificationChannel contains the Amazon Resource Name (ARN) of the Amazon SNS topic that Rekognition Video notifies when the video analysis request finishes. The Amazon SNS topic must be in
the same AWS region as the Rekognition Video endpoint that you're calling. NotificationChannel also contains the ARN for a role that allows Rekognition Video to publish to the Amazon SNS topic. You give Amazon Rekognition publishing permissions to your Amazon SNS topics by creating an IAM service role. For more information, see Giving Rekognition Video Access to Your Amazon SNS Topics (p. 58).

You can create an Amazon SNS topic for each video analysis request, or you can use a single topic for all requests. To create an Amazon SNS topic, use either the Amazon SNS AWS console or create the topic by using code.

The response to the StartLabelDetection operation is a job identifier (JobId). Use JobId to track requests and get the analysis results after Rekognition Video has published the completion status to the Amazon SNS topic. For example:

```json
{"JobId":"270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1784ca90fc180e3"}
```

You can also specify an optional input parameter, JobTag, that allows you to identify the job in the completion status that's published to the Amazon SNS topic.

To prevent accidental duplication of analysis jobs, you can optionally provide an idempotent token, ClientRequestToken. If you supply a value for ClientRequestToken, the start operation returns the same JobId for multiple identical calls to the start operation, such as StartLabelDetection. A ClientRequestToken token has a lifetime of 7 days. After 7 days, you can reuse it. If you reuse the token during the token lifetime, the following happens:

- If you reuse the token with the same Start operation and the same input parameters, the same JobId is returned. The job is not performed again and Rekognition Video does not send a completion status to the registered Amazon SNS topic.
- If you reuse the token with the same Start operation and a minor input parameter change, you get an IdempotentParameterMismatchException exception raised.
- If you reuse the token with a different Start operation, the operation succeeds.

### Getting the Completion Status of a Rekognition Video Analysis Request

Rekognition Video sends an analysis completion notification to the registered Amazon SNS topic. The notification includes the job identifier and the completion status of the operation in a JSON string. A successful video analysis request has a SUCCEEDED status. For example, the following result shows the successful processing of a label detection job.

```json
{
   "JobId": "270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1nnnnnnnnnnn",
   "Status": "SUCCEEDED",
   "API": "StartLabelDetection",
   "JobTag": "DetectingLabels",
   "Timestamp": 1510865364756,
   "Video": {
       "S3ObjectName": "video.mp4",
       "S3Bucket": "bucket"
   }
}
```

For more information, see Reference: Video Analysis Results Notification (p. 66).

To get the status information that's published to the Amazon SNS topic by Rekognition Video, use one of the following options:
Getting Rekognition Video Analysis Results

To get the results of a video analysis request, first ensure that the completion status that's retrieved from the Amazon SNS topic is **SUCCEEDED**. Then call **GetLabelDetection**, which passes the JobId value that's returned from **StartLabelDetection**. The request JSON is similar to the following example:

```json
{
  "JobId": "270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1784ca90fc180e3",
  "MaxResults": 10,
  "SortBy": "TIMESTAMP"
}
```

JobId is the identifier for the video analysis operation. Because video analysis can generate large amounts of data, use **MaxResults** to specify the maximum number of results to return in a single Get operation. If the operation doesn't return the entire set of results, a pagination token for the next page is returned in the operation response. If you have a pagination token from a previous Get request, use it with **NextToken** to get the next page of results. The default page size is 1000.

**Note**
Amazon Rekognition retains the results of a video analysis operation for 7 days. You will not be able to retrieve the analysis results after this time.

The **GetLabelDetectionResponse** operation response JSON is similar to the following:

```json
{
  "JobStatus": "SUCCEEDED",
  "Labels": [
    {
      "Label": {
        "Confidence": 56.49449920654297,
        "Name": "Bowl"
      },
      "Timestamp": 0
    },
    {
      "Label": {
        "Confidence": 77.5353012084961,
        "Name": "Bowl"
      },
      "Timestamp": 1
    }
  ]
}
```
"Name": "Clothing",
"Timestamp": 0,
",
"Label": {
"Confidence": 53.91270065307617,
"Name": "Guitar"
},
"Timestamp": 0,
",
"Label": {
"Confidence": 53.91270065307617,
"Name": "Musical Instrument"
},
"Timestamp": 0,
",
"Label": {
"Confidence": 77.5353012084961,
"Name": "Overcoat"
},
"Timestamp": 0,
",
"Label": {
"Confidence": 50.50969696044922,
"Name": "Person"
},
"Timestamp": 0,
",
"Label": {
"Confidence": 67.22470092773438,
"Name": "Pumpkin"
},
"Timestamp": 0,
",
"Label": {
"Confidence": 99.03849792480469,
"Name": "Speech"
},
"Timestamp": 0,
",
"Label": {
"Confidence": 67.22470092773438,
"Name": "Squash"
},
"Timestamp": 0,
",
"Label": {
"Confidence": 77.5353012084961,
"Name": "Suit"
},
"Timestamp": 0,
",
"NextToken": "WlKfzZED1PzBYAmTpda655kICVnPPAtQ/8V9Mc16097JKqP08uQun6j6fwGEaJFDUQVawYsg==",
"VideoMetadata": {
"Codec": "h264",
"DurationMillis": 67301,
You can sort the results by detection time (milliseconds from the start of the video) or alphabetically by the detected entity (object, face, celebrity, moderation label, or person). To sort by time, set the value of the SortBy input parameter to TIMESTAMP. If SortBy isn't specified, the default behavior is to sort by time. The preceding example is sorted by time. To sort by entity, use the SortBy input parameter with the value that's appropriate for the operation you're performing. For example, to sort by detected label in a call to GetLabelDetection, use the value NAME. The following example shows sorting by label name for the detected labels Apparel and Badge.

```json
{
    "JobStatus": "SUCCEEDED",
    "Labels": [
        {
            "Label": {
                "Confidence": 50.53730010986328,
                "Name": "Apparel"
            },
            "Timestamp": 46813
        },
        {
            "Label": {
                "Confidence": 50.538700103759766,
                "Name": "Apparel"
            },
            "Timestamp": 47013
        },
        {
            "Label": {
                "Confidence": 50.76940155029297,
                "Name": "Apparel"
            },
            "Timestamp": 47213
        },
        {
            "Label": {
                "Confidence": 50.504798889160156,
                "Name": "Apparel"
            },
            "Timestamp": 63229
        },
        {
            "Label": {
                "Confidence": 54.01129913330078,
                "Name": "Badge"
            },
            "Timestamp": 8775
        },
        {
            "Label": {
                "Confidence": 55.24839782714844,
                "Name": "Badge"
            },
            "Timestamp": 8975
        },
        {
            "Label": {
                "Confidence": 60.499000549316406,
                "Name": "Badge"
            },
            "Timestamp": 57
        }
    ]
}
```
Giving Rekognition Video Access to Your Amazon SNS Topics

You use an AWS Identity and Access Management (IAM) service role to give Rekognition Video access to Amazon SNS topics that you create. IAM provides the Rekognition use case for creating a Rekognition Video service role.

## Giving Access to Multiple Amazon SNS Topics

You can give Rekognition Video access to multiple Amazon SNS topics by using the `AmazonRekognitionServiceRole` permissions policy and prepending the topic names with `AmazonRekognition`—for example, `AmazonRekognitionMyTopicName`.

### To give Rekognition Video access to multiple Amazon SNS topics

1. Create an IAM service role. Use the following information to create the IAM service role:
   1. Choose Rekognition for the service name.
   2. Choose Rekognition for the service role use case.
3. Choose the **AmazonRekognitionServiceRole** permissions policy, which gives Rekognition Video access to Amazon SNS topics that are prefixed with *AmazonRekognition*.

2. Note the ARN of the service role. You need it to start video analysis operations.

### Giving Access to a Single Amazon SNS Topic

You can create a permissions policy that allows Rekognition Video access to a single Amazon SNS topic.

**To give Rekognition Video access to an individual Amazon SNS topic**

1. Create a new permissions policy with the IAM JSON policy editor, and use the following policy. Replace `topicarn` with the Amazon Resource Name (ARN) of the desired Amazon SNS topic.

   ```json
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": ["sns:Publish"],
         "Resource": "topicarn"
       }
     ]
   }
   ```

2. Create an IAM service role, or update an existing IAM service role. Use the following information to create the IAM service role:

   1. Choose **Rekognition** for the service name.
   2. Choose **Rekognition** for the service role use case.
   3. Attach the permissions policy you created in step 1.

3. Note the ARN of the service role. You need it to start video analysis operations.

### Analyzing a Video with the AWS Command Line Interface

You can use the AWS Command Line Interface (AWS CLI) to call Rekognition Video operations. The design pattern is the same as using the Rekognition Video API with the AWS SDK for Java or other AWS SDKs. For more information, see ??? (p. 51). The following procedures show how to use the AWS CLI to detect labels in a video.

You start detecting labels in a video by calling `start-label-detection`. When Amazon Rekognition finishes analyzing the video, the completion status is sent to the Amazon SNS topic that's specified in the `--notification-channel` parameter of `start-label-detection`. You can get the completion status by subscribing an Amazon Simple Queue Service (Amazon SQS) queue to the Amazon SNS topic. You then poll `receive-message` to get the completion status from the Amazon SQS queue.

The completion status notification is a JSON structure within the `receive-message` response. You need to extract the JSON from the response. For information about the completion status JSON, see Reference: Video Analysis Results Notification (p. 66). If the value of the `Status` field of the completed status JSON is `SUCCEEDED`, you can get the results of the video analysis request by calling `get-label-detection`.
The following procedures don't include code to poll the Amazon SQS queue. Also, they don't include code to parse the JSON that's returned from the Amazon SQS queue. For an example in Java, see Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61).

**Prerequisites**

To run this procedure, you need to have the AWS CLI installed. For more information, see Getting Started with Amazon Rekognition (p. 11). The AWS account that 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. 33).

**To configure Rekognition Video and upload a video**

1. If you haven't already, create an IAM service role to give Rekognition Video access to multiple Amazon SNS topics. Note the ARN. For more information, see ??? (p. 58).
2. Create an Amazon SNS topic by using the Amazon SNS console. Prepend the topic name with `AmazonRekognition`. Note the topic Amazon Resource Name (ARN).
3. Create an Amazon SQS standard queue by using the Amazon SQS console. Note the queue ARN.
4. Subscribe the queue to the topic you created in step 2.
5. Upload an .mp4, .mov or .avi format video file to your S3 bucket. While developing and testing, we suggest using short videos no longer than 30 seconds in length.

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

**To detect labels in a video**

1. Run the following AWS CLI command to start detecting labels in a video.

   ```bash
   aws rekognition start-label-detection --video
   S3Object={Bucket="bucketname",Name="videofile"} \
   --endpoint-url Endpoint \
   --notification-channel "SNSTopicArn=TopicARN,RoleArn=RoleARN" \
   --region us-east-1 \
   --profile RekognitionUser
   ```

   Update the following values:
   - Change `bucketname` and `videofile` to the Amazon S3 bucket name and file name of the video that you want to detect labels in.
   - Change `Endpoint` and `us-east-1` to the AWS endpoint and region that you're using.
   - Change `TopicARN` to the ARN of the Amazon SNS topic you created in step 2 of the previous procedure.
   - Change `RoleARN` to the ARN of the IAM role you created in step 1 of the previous procedure.
   - Change `RekognitionUser` to an AWS account that has permissions to call Rekognition Video operations.
2. Note the value of `JobId` in the response. The response looks similar to the following JSON example.

   ```json
   {
   "JobId": "547089ce5b9a8a0e7831afa655f42e5d7b5c838553f1a58bf350ennnnnnnnnn"
   }
   ```
3. Write code to poll the Amazon SQS queue for the completion status JSON (by using `receive-message`).

---

60
4. Write code to extract the Status field from the completion status JSON.
5. If the value of Status is SUCCESS, run the following AWS CLI command to show the label detection results.

```bash
aws rekognition get-label-detection --job-id JobId \
--endpoint-url Endpoint \
--region us-east-1 \
--profile RekognitionUser
```

Update the following values:

- Change JobId to match the job identifier that you noted in step 2.
- Change Endpoint and us-east-1 to the AWS endpoint and region that you're using.
- Change RekognitionUser to an AWS account that has permissions to call Rekognition Video operations.

The results look similar to the following example JSON:

```json
{
  "Labels": [
    {
      "Timestamp": 0,
      "Label": {
        "Confidence": 99.03720092773438,
        "Name": "Speech"
      }
    },
    {
      "Timestamp": 0,
      "Label": {
        "Confidence": 71.6698989868164,
        "Name": "Pumpkin"
      }
    },
    {
      "Timestamp": 0,
      "Label": {
        "Confidence": 71.6698989868164,
        "Name": "Squash"
      }
    },
    {
      "Timestamp": 0,
      "Label": {
        "Confidence": 71.6698989868164,
        "Name": "Vegetable"
      }
    }, .......
```

Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java

This procedure shows you how to detect labels in a video by using Rekognition Video label detection operations, a video stored in an Amazon S3 bucket, and an Amazon SNS topic. The procedure also shows how to use an Amazon SQS queue to get the completion status from the Amazon SNS topic. For more information, see ?? (p. 53). You aren't restricted to using an Amazon SQS queue. For example, you
can use an AWS Lambda function to get the completion status. For more information, see Invoking Lambda functions using Amazon SNS notifications.

The procedure shows you how to use the Amazon SNS console to do the following:

- Create the Amazon SNS topic.
- Create the Amazon SQS queue.
- Give Rekognition Video permission to publish the completion status of a video analysis operation to the Amazon SNS topic.
- Subscribe the Amazon SQS queue to the Amazon SNS topic.

**Note**
This procedure uses a single Amazon SQS queue and a single Amazon SNS topic for all video analysis requests.

The example code in the procedure shows you how to do the following:

1. Start the video analysis request by calling StartLabelDetection (p. 303).
2. Get the completion status from the Amazon SQS queue. The sample tracks the job identifier (JobId) that's returned in StartLabelDetection and only gets the results for matching job identifiers that are read from the completion status. This is an important consideration if other applications are using the same queue and topic. Any other jobs are ignored.
3. Get and display the video analysis results by calling GetLabelDetection (p. 241).

**Prerequisites**

To run this procedure, you need to have the AWS SDK for Java installed. For more information, see Getting Started with Amazon Rekognition (p. 11). The AWS account that 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. 33).

**To Detect Labels in a Video**

1. If you haven't already, create an IAM service role to give Rekognition Video access to your Amazon SNS topics. Note the ARN. For more information, see ??? (p. 58).
2. Create an Amazon SNS topic by using the Amazon SNS console. Prepend the topic name with AmazonRekognition. Note the topic Amazon Resource Name (ARN).
3. Create an Amazon SQS standard queue by using the Amazon SQS console. Note the queue ARN.
4. Subscribe the queue to the topic you created in step 2.
5. Upload an .mp4, .mov or .avi format video file to your S3 Bucket. For test purposes, upload a video that's no longer than 30 seconds in length.

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

6. Use the following AWS SDK for Java code to detect labels in a video.

   - Replace TopicArn, RoleArn, and QueueURL with the Amazon SNS topic ARN, IAM role ARN, and Amazon SQS queue URL that you previously noted.
   - Replace Bucket and VideoFile with the bucket and video file name that you uploaded in step 5.
   - Replace Endpoint and Region with the AWS endpoint and region that you're using.
   - Update .withRegion parameters to the region you are using.
   - Change RekognitionUser to an AWS account that has permissions to call Rekognition Video operations.
package com.amazonaws.samples;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.profile.ProfileCredentialsProvider;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.GetCelebrityRecognitionRequest;
import com.amazonaws.services.rekognition.model.GetCelebrityRecognitionResult;
import com.amazonaws.services.rekognition.model.GetContentModerationRequest;
import com.amazonaws.services.rekognition.model.GetContentModerationResult;
import com.amazonaws.services.rekognition.model.GetFaceDetectionRequest;
import com.amazonaws.services.rekognition.model.GetFaceDetectionResult;
import com.amazonaws.services.rekognition.model.GetFaceSearchRequest;
import com.amazonaws.services.rekognition.model.GetFaceSearchResult;
import com.amazonaws.services.rekognition.model.GetLabelDetectionRequest;
import com.amazonaws.services.rekognition.model.GetLabelDetectionResult;
import com.amazonaws.services.rekognition.model.GetPersonTrackingRequest;
import com.amazonaws.services.rekognition.model.GetPersonTrackingResult;
import com.amazonaws.services.rekognition.model.StartCelebrityRecognitionRequest;
import com.amazonaws.services.rekognition.model.StartCelebrityRecognitionResult;
import com.amazonaws.services.rekognition.model.StartContentModerationRequest;
import com.amazonaws.services.rekognition.model.StartContentModerationResult;
import com.amazonaws.services.rekognition.model.StartFaceDetectionRequest;
import com.amazonaws.services.rekognition.model.StartFaceDetectionResult;
import com.amazonaws.services.rekognition.model.StartFaceSearchRequest;
import com.amazonaws.services.rekognition.model.StartFaceSearchResult;
import com.amazonaws.services.rekognition.model.StartLabelDetectionRequest;
import com.amazonaws.services.rekognition.model.StartLabelDetectionResult;
import com.amazonaws.services.rekognition.model.StartPersonTrackingRequest;
import com.amazonaws.services.rekognition.model.StartPersonTrackingResult;
import com.amazonaws.services.sns.AmazonSNS;
import com.amazonaws.services.sns.AmazonSNSClientBuilder;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.Message;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import java.sql.Time;
import java.util.*;

public class VideoDetect {
    private static AmazonSNS sns = null;
}
private static AmazonSQS sqs = null;
private static AmazonRekognition rek = null;
private static NotificationChannel channel= new NotificationChannel()
    .withSNSTopicArn("TopicArn")
    .withRoleArn("RoleArn");
private static String queueUrl =  "QueueURL";
private static String startJobId = null;

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

    AWSCredentials credentials;
    try {
        credentials = new
            ProfileCredentialsProvider("RekognitionUser").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);
    }

    sns = AmazonSNSClientBuilder
        .standard()
        .withRegion(Regions.US_EAST_1)
        .withCredentials(new AWSStaticCredentialsProvider(credentials))
        .build();

    sqs = AmazonSQSClientBuilder
        .standard()
        .withRegion(Regions.US_EAST_1)
        .withCredentials(new AWSStaticCredentialsProvider(credentials))
        .build();

    rek = AmazonRekognitionClientBuilder.standard().withCredentials( new
            ProfileCredentialsProvider("RekognitionUser")
            .withEndpointConfiguration(new EndpointConfiguration("Endpoint",
                "Region").build());

    //=================================================================
    StartLabels("Bucket", "VideoFile");
    //=---------------------------------=
    System.out.println("Waiting for job: " + startJobId);
    //=Poll queue for messages
    List<Message> messages=null;
    int dotLine=0;
    boolean jobFound=false;

    //=loop until the job status is published. Ignore other messages in queue.
    do{
        //=Get messages.
        do{
            messages = sqs.receiveMessage(queueUrl).getMessages();
            if (dotLine++<20){
                System.out.println(".");
            }else{
                System.out.println();
                dotLine=0;
            }
        }while (messages.size()<1);
while(messages.isEmpty());

System.out.println();

// Loop through messages received.
for (Message message: messages) {
    String notification = message.getBody();
    // Get status and job id from notification.
    ObjectMapper mapper = new ObjectMapper();
    JsonNode jsonMessageTree = mapper.readTree(notification);
    JsonNode messageBodyText = jsonMessageTree.get("Message");
    ObjectMapper operationResultMapper = new ObjectMapper();
    JsonNode jsonResultTree = operationResultMapper.readTree(messageBodyText.textValue());
    JsonNode operationJobId = jsonResultTree.get("JobId");
    JsonNode operationStatus = jsonResultTree.get("Status");
    System.out.println("Job found was " + operationJobId);
    // Found job. Get the results and display.
    if (operationJobId.asText().equals(startJobId)){
        jobFound=true;
        System.out.println("Job id: " + operationJobId);
        System.out.println("Status : " + operationStatus.toString());
        if (operationStatus.asText().equals("SUCCEEDED")){
            //============================================
            GetResultsLabels();
            //============================================
        } else{
            System.out.println("Video analysis failed");
        }
        sqs.deleteMessage(queueUrl, message.getReceiptHandle());
    } else{
        System.out.println("Job received was not job " + startJobId);
    }
} while (!jobFound);

System.out.println("Done!");

private static void StartLabels(String bucket, String video) throws Exception{
    StartLabelDetectionRequest req = new StartLabelDetectionRequest()
        .withVideo(new Video()
            .withS3Object(new S3Object()
                .withBucket(bucket)
                .withName(video)))
            .withMinConfidence(50F)
            .withJobTag("DetectingLabels")
            .withNotificationChannel(channel);

    StartLabelDetectionResult startLabelDetectionResult = rek.startLabelDetection(req);
    startJobId=startLabelDetectionResult.getJobId();
}

private static void GetResultsLabels() throws Exception{
int maxResults=10;
String paginationToken=null;
GetLabelDetectionResult labelDetectionResult=null;

do {
    if (labelDetectionResult !=null){
        paginationToken = labelDetectionResult.getNextToken();
    }

    GetLabelDetectionRequest labelDetectionRequest= new
GetLabelDetectionRequest()
    .withJobId(startJobId)
    .withSortBy(LabelDetectionSortBy.TIMESTAMP)
    .withMaxResults(maxResults)
    .withNextToken(paginationToken);

    labelDetectionResult = rek.getLabelDetection(labelDetectionRequest);

    VideoMetadata videoMetaData=labelDetectionResult.getVideoMetadata();

    System.out.println("Format: " + videoMetaData.getFormat());
    System.out.println("Codec: " + videoMetaData.getCodec());
    System.out.println("Duration: " + videoMetaData.getDurationMillis());
    System.out.println("FrameRate: " + videoMetaData.getFrameRate());

    //Show labels, confidence and detection times
    List<LabelDetection> detectedLabels= labelDetectionResult.getLabels();

    for (LabelDetection detectedLabel: detectedLabels) {
        long seconds=detectedLabel.getTimestamp()/1000;
        System.out.print("Sec: " + Long.toString(seconds) + " ");
        System.out.println("t" + detectedLabel.getLabel().getName() + "	" +
        detectedLabel.getLabel().getConfidence().toString());
        System.out.println();
    }
}

7. Build and run the code. The operation might take a while to finish. After it's finished, a list of the
labels detected in the video is displayed.

Reference: Video Analysis Results Notification

Amazon Rekognition publishes the results of a Rekognition Video analysis request, including completion
status, to an Amazon Simple Notification Service (Amazon SNS) topic. To get the notification from an
Amazon SNS topic, use an Amazon Simple Queue Service queue or an AWS Lambda function. For more
information, see the section called “Calling Rekognition Video Operations” (p. 53). For an example,
see Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61).

The payload is in the following JSON format:

```json
{
    "JobId": "String",
    "Status": "String",
    "API": "String",
}
The following is an example of a successful notification that was sent to an Amazon SNS topic.

```
{
    "JobId": "6de014b0-2121-4bf0-9e31-856a18719e22",
    "Status": "SUCCEEDED",
    "JobType": "LABEL_DETECTION",
    "Message": ",
    "Timestamp": 1502230160926,
    "Video": {
        "S3ObjectName": "video.mpg",
        "S3Bucket": "videobucket"
    }
}
```
Working with Streaming Videos

You can use Amazon Rekognition Video to detect and recognize faces in streaming video. A typical use case is surveillance where you want to detect a known face in a video stream. Rekognition Video uses Amazon Kinesis Video Streams to receive and process a video stream. The analysis results are output from Rekognition Video to a Kinesis data stream and then read by your client application. Rekognition Video provides a stream processor (CreateStreamProcessor (p. 183)) that you can use to start and manage the analysis of streaming video.

The following diagram shows how Rekognition Video detects and recognizes faces in a streaming video.

To use Rekognition Video with streaming video, your application needs to implement the following:

- A Kinesis video stream for sending streaming video to Rekognition Video. For more information, see Kinesis video stream.
- A Rekognition Video stream processor to manage the analysis of the streaming video. For more information, see ??? (p. 70).
- A Kinesis data stream consumer to read the analysis results that Rekognition Video sends to the Kinesis data stream. For more information, see Consumers for Amazon Kinesis Streams.

This section contains information about writing an application that creates the Kinesis video stream and the Kinesis data stream, streams video into Rekognition Video, and consumes the analysis results. For more information, see ??? (p. 68).

Topics

- Recognizing Faces in a Streaming Video (p. 68)
- Giving Rekognition Video Access to Your Kinesis Streams (p. 69)
- Starting Streaming Video Analysis (p. 70)
- Reading Streaming Video Analysis Results (p. 75)
- Reference: Kinesis Face Recognition Record (p. 77)

Recognizing Faces in a Streaming Video

Rekognition Video can search faces in a collection that match faces that are detected in a streaming video. For more information about collections, see Searching Faces in a Collection (p. 102). The following procedure describes the steps you take to recognize faces in a streaming video.
Prerequisites

To run this procedure, you need to have the AWS SDK for Java installed. For more information, see Getting Started with Amazon Rekognition (p. 11). 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. 33).

To recognize faces in a video stream (AWS SDK)

1. If you haven't already, create an IAM service role to give Rekognition Video access to your Kinesis video streams and your Kinesis data streams. Note the ARN. For more information, see ??? (p. 69).
2. Create a collection (p. 103) and note the collection identifier you used.
3. Index the faces (p. 112) you want to search for into the collection you created in step 2.
4. Create a Kinesis video stream and note the stream's Amazon Resource Name (ARN).
5. Create a Kinesis data stream. Prepend the stream name with AmazonRekognition and note the stream's ARN.
6. Create the stream processor (p. 72). Pass the following as parameters to the section called “CreateStreamProcessor” (p. 183): a name of your choosing, the Kinesis video stream ARN (step 4), the Kinesis data stream ARN (step 5), and the collection identifier (step 2).
7. Start the stream processor (p. ) using the stream processor name that you chose in step 6.
8. Use the PutMedia operation to stream the source video into the Kinesis video stream that you created in step 4. For more information, see PutMedia API Example.
9. Consume the analysis output from Rekognition Video (p. 75).

Giving Rekognition Video Access to Your Kinesis Streams

You use an AWS Identity and Access Management (IAM) service role to give Rekognition Video read access to Kinesis video streams and write access to Kinesis data streams.

Giving Access to Your Kinesis Video Streams and Kinesis Data Streams

IAM provides the Rekognition service role use case that, when used with the AmazonRekognitionServiceRole permissions policy, can write to multiple Kinesis data streams and read from all your Kinesis video streams. To give Rekognition Video write access to multiple Kinesis data streams, you can prepend the names of the Kinesis data streams with AmazonRekognition—for example, AmazonRekognitionMyDataStreamName.

To give Rekognition Video access to your Kinesis video stream and Kinesis data stream

1. Create an IAM service role. Use the following information to create the IAM service role:
   1. Choose Rekognition for the service name.
   2. Choose Rekognition for the service role use case.
   3. Choose the AmazonRekognitionServiceRole permissions policy, which gives Rekognition Video write access to Kinesis data streams that are prefixed with AmazonRekognition and read access to all your Kinesis video streams.
   2. Note the Amazon Resource Name (ARN) of the service role. You need it to start video analysis operations.
Giving Access to Individual Kinesis Streams

You can create a permissions policy that allows Rekognition Video access to individual Kinesis video streams and Kinesis data streams.

To give Rekognition Video access to an individual Kinesis video stream and Kinesis data stream

1. Create a new permissions policy with the IAM JSON policy editor, and use the following policy. Replace data-arn with the ARN of the desired Kinesis data stream and video-arn with the ARN of the desired Kinesis video stream.

   ```json
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": ["kinesis:PutRecord", "kinesis:PutRecords"],
         "Resource": "data-arn"
       },
       {
         "Effect": "Allow",
         "Action": ["kinesisvideo:GetDataEndpoint", "kinesisvideo:GetMedia"],
         "Resource": "video-arn"
       }
     ]
   }
   ```

2. Create an IAM service role, or update an existing IAM service role. Use the following information to create the IAM service role:

   1. Choose Rekognition for the service name.
   2. Choose Rekognition for the service role use case.
   3. Attach the permissions policy that you created in step 1.

3. Note the ARN of the service role. You need it to start video analysis operations.

Starting Streaming Video Analysis

You start analyzing a streaming video by starting a Rekognition Video stream processor and streaming video into Rekognition Video. A Rekognition Video stream processor allows you to start, stop, and manage stream processors. You create a stream processor by calling CreateStreamProcessor (p. 183). The request parameters include the Amazon Resource Names (ARNs) for the Kinesis video stream, the Kinesis data stream, and the identifier for the collection that’s used to recognize faces in the streaming video. It also includes the name that you specify for the stream processor.

You start processing a video by calling the StartStreamProcessor. You create a stream processor by calling CreateStreamProcessor (p. 183). To tell StartStreamProcessor which stream processor to start, use the value of the Name field specified in the call to CreateStreamProcessor.
Request Syntax

```
{
  "Name": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

Name (p. 310)

- The name of the stream processor to start processing.
- Type: String
- Pattern: `[a-zA-Z0-9_.-]+`
- Required: Yes

Response Elements

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

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

ResourceInUseException

- HTTP Status Code: 400

ResourceNotFoundException

- The collection specified in the request cannot be found.
Creating the Rekognition Video Stream Processor

Before you can analyze a streaming video, you create a Rekognition Video stream processor (CreateStreamProcessor (p. 183)). The stream processor contains information about the Kinesis data stream and the Kinesis video stream. It also contains the identifier for the collection that contains the faces you want to recognize in the input streaming video. You also specify a name for the stream processor. The following is a JSON example for the CreateStreamProcessor request.

```
{
    "Name": "streamProcessorForCam",
    "Input": {
        "KinesisVideoStream": {
            "Arn": "arn:aws:kinesisvideo:us-east-1:nnnnnnnnnnn:stream/inputVideo"
        }
    },
    "Output": {
        "KinesisDataStream": {
            "Arn": "arn:aws:kinesis:us-east-1:nnnnnnnnnnn:stream/outputData"
        }
    }
}
```
"RoleArn": "arn:aws:iam::nnnnnnnnnnn:role/roleWithKinesisPermission",
"Settings": {
  "FaceSearch": {
    "CollectionId": "collection-with-100-faces",
    "FaceMatchThreshold": 85.5
  }
}
}

The following is an example response from CreateStreamProcessor.

{
  "StreamProcessorArn": "arn:aws:rekognition:us-east-1:987654321098:streamprocessor/streamProcessorForCam"
}

---

### Starting the Rekognition Video Stream Processor

You start analyzing streaming video by calling `StartStreamProcessor` with the stream processor name that you specified in CreateStreamProcessor. The following is a JSON example for the `StartStreamProcessor` request.

```
{
  "Name": "streamProcessorForCam"
}
```

If the stream processor successfully starts, an HTTP 200 response is returned, along with an empty JSON body.

---

### Using Stream Processors

The following example code shows how to call various stream processor operations, such as `CreateStreamProcessor` and `StartStreamProcessor`.

```
package com.amazonaws.rekognition.video.streaming;
import com.amazonaws.services.rekognition.AmazonRekognitionClient;
import com.amazonaws.services.rekognition.model.CreateStreamProcessorRequest;
import com.amazonaws.services.rekognition.model.CreateStreamProcessorResult;
import com.amazonaws.services.rekognition.model.DeleteStreamProcessorRequest;
import com.amazonaws.services.rekognition.model.DeleteStreamProcessorResult;
import com.amazonaws.services.rekognition.model.DescribeStreamProcessorRequest;
import com.amazonaws.services.rekognition.model.DescribeStreamProcessorResult;
import com.amazonaws.services.rekognition.model.FaceSearchSettings;
import com.amazonaws.services.rekognition.model.KinesisDataStream;
import com.amazonaws.services.rekognition.model.KinesisVideoStream;
import com.amazonaws.services.rekognition.model.ListStreamProcessorsRequest;
import com.amazonaws.services.rekognition.model.ListStreamProcessorsResult;
import com.amazonaws.services.rekognition.model.StartStreamProcessorRequest;
import com.amazonaws.services.rekognition.model.StartStreamProcessorResult;
import com.amazonaws.services.rekognition.model.StopStreamProcessorRequest;
import com.amazonaws.services.rekognition.model.StopStreamProcessorResult;
import com.amazonaws.services.rekognition.model.StreamProcessor;
import com.amazonaws.services.rekognition.model.StreamProcessorInput;
import com.amazonaws.services.rekognition.model.StreamProcessorOutput;
import com.amazonaws.services.rekognition.model.StreamProcessorSettings;

public class StreamProcessorSample {
```

```java
private String streamProcessorName;
private String kinesisVideoStreamArn;
private String kinesisDataStreamArn;
private String roleArn;
private String collectionId;
private float matchThreshold;
private AmazonRekognitionClient rekognitionClient;

public void createStreamProcessorSample() {
    KinesisVideoStream kinesisVideoStream = new KinesisVideoStream().withArn(kinesisVideoStreamArn);
    StreamProcessorInput streamProcessorInput = new StreamProcessorInput().withKinesisVideoStream(kinesisVideoStream);
    KinesisDataStream kinesisDataStream = new KinesisDataStream().withArn(kinesisDataStreamArn);
    StreamProcessorOutput streamProcessorOutput = new StreamProcessorOutput().withKinesisDataStream(kinesisDataStream);
    FaceSearchSettings faceSearchSettings = new FaceSearchSettings().withCollectionId(collectionId).withFaceMatchThreshold(matchThreshold);
    StreamProcessorSettings streamProcessorSettings = new StreamProcessorSettings().withFaceSearch(faceSearchSettings);
    CreateStreamProcessorResult createStreamProcessorResult = rekognitionClient.createStreamProcessor(
        new CreateStreamProcessorRequest().withInput(streamProcessorInput).withOutput(streamProcessorOutput)
        .withSettings(streamProcessorSettings).withRoleArn(roleArn).withName(streamProcessorName));
    System.out.println("StreamProcessorArn - " + createStreamProcessorResult.getStreamProcessorArn());
}

public void startStreamProcessorSample() {
    StartStreamProcessorResult startStreamProcessorResult = rekognitionClient.startStreamProcessor(new StartStreamProcessorRequest().withName(streamProcessorName));
}

public void stopStreamProcessorSample() {
    StopStreamProcessorResult stopStreamProcessorResult = rekognitionClient.stopStreamProcessor(new StopStreamProcessorRequest().withName(streamProcessorName));
}

public void deleteStreamProcessorSample() {
    DeleteStreamProcessorResult deleteStreamProcessorResult = rekognitionClient.deleteStreamProcessor(new DeleteStreamProcessorRequest().withName(streamProcessorName));
}

public void describeStreamProcessorSample() {
    DescribeStreamProcessorResult describeStreamProcessorResult = rekognitionClient.describeStreamProcessor(
        new DescribeStreamProcessorRequest().withName(streamProcessorName));
    System.out.println("Arn - " + describeStreamProcessorResult.getStreamProcessorArn());
    System.out.println("Input kinesisVideo stream - " + describeStreamProcessorResult.getInput().getKinesisVideoStream().getArn());
    System.out.println("Output kinesisDataStream - " + describeStreamProcessorResult.getOutput().getKinesisDataStream().getArn());
    System.out.println("RoleArn - " + describeStreamProcessorResult.getRoleArn());
    System.out.println("
```
Streaming Video into Rekognition Video

To stream video into Rekognition Video, you use the Amazon Kinesis Video Streams SDK to create and use a Kinesis video stream. The `putMedia` operation writes video data fragments into a Kinesis video stream that Rekognition Video consumes. Each video data fragment is typically 2–10 seconds in length and contains a self-contained sequence of video frames. Rekognition Video supports H.264 encoded videos, which can have three types of frames (I, B, and P). For more information, see Inter Frame. The first frame in the fragment must be an I-frame. An I-frame can be decoded independent of any other frame.

As video data arrives into the Kinesis video stream, Kinesis Video Streams assigns a unique number to the fragment. For an example, see `PutMedia API Example`.

Reading Streaming Video Analysis Results

You can use the Amazon Kinesis Data Streams Client Library to consume analysis results that are sent to the Amazon Kinesis Data Streams output stream. For more information, see `Reading Data from a Kinesis Data Stream`. Rekognition Video places a JSON frame record for each analyzed frame into the Kinesis output stream. Rekognition Video doesn't analyze every frame that's passed to it though the Kinesis video stream.

A frame record that's sent to a Kinesis data stream contains information about which Kinesis video stream fragment the frame is in, where the frame is in the fragment, and faces that are recognized in the frame. It also includes status information for the stream processor. For more information, see `Reference: Kinesis Face Recognition Record (p. 77)`.

Rekognition Video streams Rekognition Video analysis information to the Kinesis data stream. The following is a JSON example for a single record.

```json
{
  "InputInformation": {
    "KinesisVideo": {
      "FragmentNumber": "9134385233289682796718536144575784843717598",
      "ServerTimestamp": 1510552593.455,
    }
  }
}
"ProducerTimestamp": 1510552593.193,
"FrameOffsetInSeconds": 2
},
"StreamProcessorInformation": {
"Status": "RUNNING"
},
"FaceSearchResponse": [
{
"DetectedFace": {
"BoundingBox": {
"Height": 0.075,
"Width": 0.05625,
"Left": 0.428125,
"Top": 0.40833333
},
"Confidence": 99.975174,
"Landmarks": [
{
"X": 0.4452057,
"Y": 0.4395594,
"Type": "eyeLeft"
},
{
"X": 0.46340984,
"Y": 0.43744427,
"Type": "eyeRight"
},
{
"X": 0.45960626,
"Y": 0.4526856,
"Type": "nose"
},
{
"X": 0.44958648,
"Y": 0.4696949,
"Type": "mouthLeft"
},
{
"X": 0.46409217,
"Y": 0.46704912,
"Type": "mouthRight"
}
],
"Pose": {
"Pitch": 2.9691637,
"Roll": -6.8904796,
"Yaw": 23.84388
},
"Quality": {
"Brightness": 40.592964,
"Sharpness": 96.09616
}
},
"MatchedFaces": [
{
"Similarity": 88.863960,
"Face": {
"BoundingBox": {
"Height": 0.557692,
"Width": 0.749838,
"Left": 0.103426,
"Top": 0.206731
},
"FaceId": "ed1b560f-d6af-5158-989a-ff586c931545",
"Confidence": 99.999201,
In the JSON example, note the following:

- **InputInformation** – Information about the Kinesis video stream that's used to stream video into Rekognition Video. For more information, see ??? (p. 78).

- **StreamProcessorInformation** – Status information for the Rekognition Video stream processor. Possible values for the Status field are STARTING, STARTED, IN_PROGRESS, STOPPING, STOPPED, and FAILED. For more information, see ??? (p. 79).

- **FaceSearchResponse** – Contains information about faces in the streaming video that match faces in the input collection. FaceSearchResponse contains a DetectedFaces array, which is an array of faces that were matched in the analyzed video frame. For each matched face, the array MatchedFaces contains information about the matching faces found in the input collection, along with a similarity score. For more information, see ??? (p. 79).

---

Reference: Kinesis Face Recognition Record

Rekognition Video can recognize faces in a streaming video. For each analyzed frame, Rekognition Video outputs a JSON frame record to a Kinesis data stream. Rekognition Video doesn't analyze every frame that's passed to it though the Kinesis video stream.

The JSON frame record contains information about the input and output stream, the status of the stream processor, and information about faces that are recognized in the analyzed frame. This section contains reference information for the JSON frame record.

The following is the JSON syntax for a Kinesis data stream record. For more information, see Working with Streaming Videos (p. 68).

```json
{

  "InputInformation":{
    "KinesisVideo": {
      "StreamArn": "string",
      "FragmentNumber": "string",
      "ProducerTimestamp": long,
      "ServerTimestamp": long,
      "FrameOffsetInMillis": long
    },
  },

  "StreamProcessorInformation":{
    "Status": "STARTING|IN_PROGRESS|STOPPING|STOPPED|FAILED"
  },

  "FaceSearchResponse":{
    "DetectedFace":{
      "BoundingBox":{
        "Width":number,
        "Top":number,
        "Height":number,
        "Left":number
      },
      "Confidence":number,
    }
  }
}
```
The JSON record includes information about a frame that's processed by Rekognition Video. The record includes information about the streaming video, the status for the analyzed frame, and information about faces that are recognized in the frame.

**InputInformation**

Information about the Kinesis video stream that's used to stream video into Rekognition Video.

Type: ??? (p. 78) object

**StreamProcessorInformation**

Information about the Rekognition Video stream processor. This includes status information for the current status of the stream processor.

Type: ??? (p. 79) object

**FaceSearchResponse**

Information about faces in the streaming video frame that match faces in the input collection.

Type: ??? (p. 79) object

**InputInformation**

Information about a source video stream that's used by Rekognition Video. For more information, see Working with Streaming Videos (p. 68).
KinesisVideo

Information about the Kinesis data stream that's used to stream video into Rekognition Video.

Type: ??? (p. 79) object

KinesisVideo

Information about the Kinesis video stream that streams the source video into Rekognition Video. For more information, see Working with Streaming Videos (p. 68).

StreamArn

The Amazon Resource Name (ARN) of the Kinesis video stream.

Type: String

FragmentNumber

The fragment of streaming video that contains the frame that this record represents.

Type: String

ProducerTimestamp

The producer-side time stamp of the fragment. For more information, see GetMedia.

Type: Timestamp

ServerTimestamp

The server-side time stamp of the fragment. For more information, see GetMedia.

Type: Timestamp

FrameOffsetInMillis

The offset of the frame (in milliseconds) inside the fragment.

Type: Number

StreamProcessorInformation

Status information about the stream processor.

Status

The current status of the stream processor. The possible values are STARTING, IN_PROGRESS, STOPPING, STOPPED, and FAILED.

Type: String

FaceSearchResponse

Information about faces in a streaming video frame that match faces in a collection. You specify the collection in a call to CreateStreamProcessor (p. 183). For more information, see Working with Streaming Videos (p. 68).

DetectedFaces
The array of face details for faces that are recognized in an analyzed video frame.

Type: ??? (p. 80) object

**DetectedFace**

Information about a face that's matched in a streaming video frame. Each `DetectedFace` includes information about the detected face and the faces in the input collection that match the detected face.

**BoundingBox (p. 317)**

The bounding box coordinates for a face that's detected within an analyzed video frame. The `BoundingBox` object has the same properties as the `BoundingBox` object that's used for image analysis.

**Confidence**

The confidence level that Rekognition Video has that the detected face is actually a face.

**Landmarks**

An array of facial landmarks. For more information, see `Landmark (p. 347)`.

**Pose (p. 356)**

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

**Quality**

Identifies face image brightness and sharpness. The quality values are the same as the `ImageQuality (p. 342)` object.

**MatchedFaces**

An array of face match information for faces in the input collection that match the face that's recognized in `FaceDetail`.

Type: ??? (p. 80) object

**MatchedFace**

**Face**

Face match information for a face in the input collection that matches the face that's detected in ??? (p. 80).

Type: `Face (p. 330)` object
Detecting Objects and Scenes

This section provides information for detecting labels in images and videos with Rekognition Image and Rekognition Video.

A label or a tag is an object, scene, or concept found in an image or video 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). Rekognition Video can also detect activities such as a person skiing or riding a bike. Rekognition Image does not detect activities in images.

For example, in the following image, Rekognition Image is able to detect the presence of a person, a skateboard, parked cars and other information. Rekognition Video and Rekognition Image also provide a percentage score for how much confidence Amazon Rekognition has in the accuracy of each detected label.

Topics
- Detecting Labels in an Image (p. 81)
- Detecting Labels in a Video (p. 83)

Detecting Labels in an Image

You can use the DetectLabels (p. 205) operation to detect labels in an image. For an example, see Analysing Images Stored in an Amazon S3 Bucket (p. 38).
For each label, Rekognition Image returns a name and a confidence value in the analysis. The following is an example response from `DetectLabels`.

```json
{
  "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 operation 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:

```json
{
  "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 Labels in a Video

Rekognition Video can detect labels, and the time a label is detected, in a video. For an AWS SDK for Java example, see Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61). For an AWS CLI example, see Analyzing a Video with the AWS Command Line Interface (p. 59).

Rekognition Video label detection is an asynchronous operation. To start the detection of labels in a video, call StartLabelDetection (p. 303). Rekognition Video publishes the completion status of the video analysis to an Amazon Simple Notification Service topic. If the video analysis is succesful, call GetLabelDetection (p. 241) to get the detected labels.

For information about calling Rekognition Video operations, see Calling Rekognition Video Operations (p. 53).

GetLabelDetection returns an array (Labels) that contains information about the labels detected in the video. The array can be sorted either by time or by the label detected by specifying the SortBy parameter. The following example is the JSON response of the GetLabelDetection.

```json
{
  "Labels": [{
    "Label": {
      "Confidence": 99.03720092773438,
      "Name": "Speech"
    },
    "Timestamp": 0
  }, {
    "Label": {
      "Confidence": 71.6698989868164,
      "Name": "Pumpkin"
    },
    "Timestamp": 0
  }, {
    "Label": {
      "Confidence": 71.6698989868164,
      "Name": "Squash"
    },
    "Timestamp": 0
  }, {
    "Label": {
      "Confidence": 71.6698989868164,
      "Name": "Vegetable"
    },
    "Timestamp": 0
  }, {
    "Label": {
      "Confidence": 71.44749450683594,
      "Name": "Clothing"
    },
    "Timestamp": 0
  }, {
    "Label": {
      "Confidence": 71.44749450683594,
      "Name": "Overcoat"
    },
    "Timestamp": 0
  }, {
    "Label": {
      "Confidence": 71.44749450683594,
      "Name": "Suit"
    },
    "Timestamp": 0
  }
}, {
```
"Label": {
   "Confidence": 58.84939956665039,
   "Name": "Jar"
},
"Timestamp": 0
}, {
   "Label": {
   "Confidence": 58.84939956665039,
   "Name": "Porcelain"
},
"Timestamp": 0
}, {
   "Label": {
   "Confidence": 58.84939956665039,
   "Name": "Vase"
},
"Timestamp": 0
}

"NextToken": "BgPXS37fNUY+Hrd7jYbBoTk1I5LevLm/MV+dhCwbXxGOGMi0di6YSe6/Dztya/
Pflx9xxxx==",
"VideoMetadata": {
   "Codec": "h264",
   "DurationMillis": 67301,
   "Format": "mov,mp4,m4a,3gp,3g2,mj2",
   "FrameHeight": 1080,
   "FrameRate": 29.970029830932617,
   "FrameWidth": 1920
}

In the response, note the following:

- **Sort order** – The array of labels returned is sorted by time. To sort by label, specify `NAME` in the `SortBy` input parameter for `GetLabelDetection`. If the label appears multiple times in the video, there will be multiples instances of the `(LabelDetection (p. 346))` element.

- **Label information** – The `LabelDetection` array element contains a `(Label (p. 345))` object which contains the label name and the confidence Amazon Rekognition has in the accuracy of the detected label. `Timestamp` is the time, in milliseconds from the start of the video, that the label was detected.

- **Paging information** – The example shows one page of label detection information. You can specify how many `LabelDetection` objects to return in the `MaxResults` input parameter for `GetLabelDetection`. If more results than `MaxResults` exist, `GetLabelDetection` returns a token (`NextToken`) used to get the next page of results. For more information, see ??? (p. 55).

- **Video information** – The response includes information about the video format (`VideoMetadata`) in each page of information returned by `GetLabelDetection`.
Detecting and Analyzing Faces

Amazon Rekognition can detect faces in images and videos. This section covers non-storage operations for analyzing faces. With Amazon Rekognition you can get information about where faces are detected in an image or video, facial landmarks such as the position of eyes, and detected emotions such as happy or sad. You can also compare a face in an image with faces detected in another image. With video analysis, you can track when a face is detected throughout the length of a video.

For example, When you provide an image that contains a face as input to DetectFaces (p. 198), Amazon Rekognition 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.

This section provides an examples for both image and video facial analysis. For more information about using the Amazon Rekognition API, see Working with Images (p. 37) and Working with Stored Videos (p. 51).

You can use storage operations to save facial metadata for faces detected in an image. Later you can search for stored faces in both images and videos. For example, this enables searching for a specific person in a surveillance video. For more information, see Searching Faces in a Collection (p. 102).

Topics
- Detect Faces in an Image (p. 86)
- Compare Faces in Images (p. 89)
- Detecting Faces in a Stored Video (p. 91)
- Detecting Faces in an Image (SDK) (p. 95)
- Comparing Faces in Images (SDK) (p. 97)
- Detecting Faces in a Stored Video (SDK for Java) (p. 100)
Detect Faces in an Image

Rekognition Image provides the DetectFaces operation that looks for key facial features such as eyes, nose, and mouth to detect faces in an input image. Rekognition Image detects the 100 largest faces in an image. The operation 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. For information about ensuring the best possible face detection, see ???.
- **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.

```json
{
  "FaceDetails": [
    {
      "BoundingBox": {
        "Height": 0.18000000715255737,
        "Left": 0.5555555820465088,
        "Top": 0.33666667342185974,
        "Width": 0.23999999463558197
      },
      "Confidence": 100.0,
      "Landmarks": [
        {
          "Type": "eyeLeft",
          "X": 0.6394737362861633,
          "Y": 0.4081962406635284
        },
        {
          "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":99.9998474121094,
"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**.

```json
...  "Landmarks": [
    {
      "Y": 0.41730427742004395,
      "X": 0.36835095286369324,
      "Type": "eyeLeft"
    },
    {
      "Y": 0.4281611740589142,
      "X": 0.59605656404495239,
      "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.
Compare Faces in Images

To compare a face in the source image with each face in the target image, use the CompareFaces (p. 172) 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. 172).

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.

```json
{
    "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": [
  {
    "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"
      }
    ]
  }
]
Detecting Faces in a Stored Video

Rekognition Video can detect faces in videos stored in an Amazon S3 bucket and provide information such as:

- The time(s) faces are detected in a video.
- The location of faces in the video frame at the time they were detected.
Facial landmarks such as the position of the left eye.

Using Rekognition Video Face Operations with Video

Rekognition Video face detection in videos is an asynchronous operation. To start the detection of persons in videos call StartFaceDetection (p. 295). Rekognition Video publishes the completion status of the video analysis to an Amazon Simple Notification Service topic. If the video analysis is successful, you can call GetFaceDetection (p. 231) to get results of the video analysis.

For information about calling the video recognition API operations, see Calling Rekognition Video Operations (p. 53). For an example, see Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61).

GetFaceDetection returns an array (Faces) containing information about the faces detected in the video. An array element, FaceDetection (p. 335), exists for each time a face is detected in the video. The array elements are returned sorted by time, in milliseconds since the start of the video. The following example is a partial JSON response from GetFaceDetection.

```json
{
  "Faces": [
    {
      "Face": {
        "BoundingBox": {
          "Height": 0.23000000417232513,
          "Left": 0.42500001192092896,
          "Top": 0.16333332657814026,
          "Width": 0.12937499582767487
        },
        "Confidence": 99.97504425048828,
        "Landmarks": [
          {
            "Type": "eyeLeft",
            "X": 0.46415066719055176,
            "Y": 0.2572723925113678
          },
          {
            "Type": "eyeRight",
            "X": 0.5068183541297913,
            "Y": 0.23705792427062988
          },
          {
            "Type": "nose",
            "X": 0.4976589977412415,
            "Y": 0.28383663296699524
          },
          {
            "Type": "mouthLeft",
            "X": 0.487221896648407,
            "Y": 0.3452930748462677
          },
          {
            "Type": "mouthRight",
            "X": 0.5142884850502014,
            "Y": 0.33167609572410583
          }
        ],
        "Pose": {
          "Pitch": 15.966927528381348,
          "Roll": -15.547388076782227,
          "Yaw": 11.34195613861084
        }
      }
    }
  ]
}
```
"Quality": {
  "Brightness": 44.80223083496094,
  "Sharpness": 99.95819854736328
},
"Timestamp": 0
},
"Face": {
  "BoundingBox": {
    "Height": 0.20000000298023224,
    "Left": 0.029999999329447746,
    "Top": 0.2199999988079071,
    "Width": 0.11249999701976776
  },
  "Confidence": 99.85971069335938,
  "Landmarks": [
    {
      "Type": "eyeLeft",
      "X": 0.06842322647571564,
      "Y": 0.3010137975215912
    },
    {
      "Type": "eyeRight",
      "X": 0.10543643683195114,
      "Y": 0.29697132110595703
    },
    {
      "Type": "nose",
      "X": 0.09569807350635529,
      "Y": 0.33701086044311523
    },
    {
      "Type": "mouthLeft",
      "X": 0.0732642263174057,
      "Y": 0.3757539987564087
    },
    {
      "Type": "mouthRight",
      "X": 0.10589495301246643,
      "Y": 0.3722417950630188
    }
  ],
  "Pose": {
    "Pitch": -0.5589138865470886,
    "Roll": -5.1093974113464355,
    "Yaw": 18.69594955444336
  },
  "Quality": {
    "Brightness": 43.052337646484375,
    "Sharpness": 99.68138885498047
  }
},
"Timestamp": 0
},
"Face": {
  "BoundingBox": {
    "Height": 0.2177777737379074,
    "Left": 0.7593749761581421,
    "Top": 0.1333334028720856,
    "Width": 0.12250000238418579
  },
  "Confidence": 99.63436889648438,
  "Landmarks": [

In the response, note the following:

- **Face information** – The FaceDetection array element contains information about the detected face (FaceDetail (p. 332) and the time the face was detected in the video (Timestamp).

- **Paging information** – The example shows one page of face detection information. You can specify how many person elements to return in the MaxResults input parameter for GetFaceDetection. If more results than MaxResults exist, GetFaceDetection returns a token (NextToken) used to get the next page of results. For more information, see ?? (p. 55).

- **Video information** – The response includes information about the video format (VideoMetadata) in each page of information returned by GetFaceDetection.
Detecting Faces in an Image (SDK)

In this procedure you use the DetectFaces (p. 198) 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 Detect Faces in an Image (p. 86).

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

     ```bash
     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.

     ```java
     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. ")
             }
             String s3ObjectKey = "s3ObjectKey";
             String bucketname = "Bucketname";
             DetectFacesRequest request = new DetectFacesRequest();
             request.withAttributes(Attribute.ALL);
             request.withBucketName(bucketname);
             request.withImage(Image.builder().s3Object(S3Object.builder().bucket(bucketname).name(s3ObjectKey).build()).build());
             AmazonRekognitionClient client = AmazonRekognitionClientBuilder.standard().withCredentials(credentials).withRegion(Regions.US_EAST_1).build();
             DetectFacesResult result = client.detectFaces(request);
             List<FaceDetail> faceDetails = result.getFaceDetails();
             for (FaceDetail faceDetail : faceDetails) {
                 System.out.println("Face details: " + faceDetail.toJsonString());
             }
             for (Attribute attribute : result.getAttributes()) {
                 System.out.println("Attribute details: " + attribute.toJsonString());
             }
         }
     }
     ```
• 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.

```python
import boto3
import json

if __name__ == '__main__':
    fileName = 'input.jpg'
    bucket = 'bucket'
    client = boto3.client('rekognition', 'us-east-1')
    response = client.detect_faces(Image={'S3Object':
        {'Bucket': bucket, 'Name': fileName}}, Attributes=['ALL'])

    print('Detected faces for ' + fileName)
    for faceDetail in response['FaceDetails']:
        System.out.println(objectMapper.writerWithDefaultPrettyPrinter().writeValueAsString(faceDetail))
```
Comparing Faces in Images (SDK)

In this procedure you use the CompareFaces (p. 172) 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 assumes 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

   ```bash
   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.

   ```java
   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', 'us-east-1')

    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')

99
Detecting Faces in a Stored Video (SDK for Java)

Amazon Rekognition can detect faces in a video stored in an Amazon S3 bucket. This procedure expands on the code in Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61) which uses an Amazon Simple Queue Service queue to get the completion status of a video analysis request.

Prerequisites

To run this procedure, you need to have the AWS SDK for Java installed. For more information, see Getting Started with Amazon Rekognition (p. 11). 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. 33).

To detect faces in a video stored in an Amazon S3 bucket

1. Perform Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61).
2. Add the following code to the project created in step 1.

```java
//Faces
private static void StartFaces(String bucket, String video) throws Exception{
    StartFaceDetectionRequest req = new StartFaceDetectionRequest()
        .withVideo(new Video()
            .withS3Object(new S3Object()
                .withBucket(bucket)
                .withName(video))
            .withNotificationChannel(channel);
    StartFaceDetectionResult startLabelDetectionResult =
        rek.startFaceDetection(req);
    startJobId=startLabelDetectionResult.getJobId();
}

private static void GetResultsFaces() throws Exception{
    int maxResults=10;
    String paginationToken=null;
    GetFaceDetectionResult faceDetectionResult=null;
    do{
        if (faceDetectionResult !=null){
            paginationToken = faceDetectionResult.getNextToken();
        }
        faceDetectionResult = rek.getFaceDetection(new GetFaceDetectionRequest()
            .withJobId(startJobId)
            .withNextToken(paginationToken)
            .withMaxResults(maxResults));
        VideoMetadata videoMetaData=faceDetectionResult.getVideoMetadata();
        System.out.println("Format: " + videoMetaData.getFormat());
        System.out.println("Codec: " + videoMetaData.getCodec());
        System.out.println("Duration: " + videoMetaData.getDurationMillis());
        System.out.println("FrameRate: " + videoMetaData.getFrameRate());
    }
```
//Show faces, confidence and detection times
List<FaceDetection> faces= faceDetectionResult.getFaces();

for (FaceDetection face: faces) {
    long seconds=face.getTimestamp()/1000;
    System.out.print("Sec: " + Long.toString(seconds) + " ");
    System.out.println(face.getFace().toString());
    System.out.println();
} while (faceDetectionResult !=null && faceDetectionResult.getNextToken() !=null);

3. In the function main, replace the line

    StartLabels("bucket","file");

with

    StartFaces("bucket","file");

If you have already run a video example other than Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61), the function name to replace will be different.

4. In the call to StartFaces, replace bucket and file with the Amazon S3 bucket name and file name of the video you want to analyze.

5. In the function main, replace the line

    GetResultsLabels();

with

    GetResultsFaces();

If you have already run a video example other than Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61), the function name to replace will be different.

6. Run the code. Information about the faces detected in the video is shown.
Searching Faces in a Collection

Amazon Rekognition can store information about detected faces in server-side containers known as collections. You can use the facial information stored in a collection to search for known faces in images, stored videos and streaming videos. Amazon Rekognition supports the `IndexFaces` (p. 250) operation, which you can use to detect faces in an image and persist information about facial features detected into a collection. 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 (CreateCollection (p. 180)) a face collection in one of the 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. To search for faces in an image, call `SearchFacesByImage` (p. 282). To search for faces in a stored video, call `StartFaceSearch` (p. 299). To search for faces in a streaming video, call `CreateStreamProcessor` (p. 183).

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 collection. Amazon Rekognition uses these feature vectors when performing face matches.

You can use collections in a variety of scenarios. For example, you might create a face collection to store scanned badge images using the `IndexFaces` operation. When an employee enters the building, an image of the employee's face is captured and sent to the `SearchFacesByImage` operation. If the face match produces a sufficiently high similarity score (say 99%), you can authenticate the employee.

Topics
- Managing Collections (p. 102)
- Storing Faces in a Face Collection (p. 107)
- Searching for Faces with Rekognition Image Collection (p. 116)
- Searching for Faces with Rekognition Video (p. 121)

Managing Collections

This section covers creating and managing collections. A collection is a container for persisting faces detected by the `IndexFaces` (p. 250) operation.

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. 180)
  Amazon Rekognition creates the collection and returns the Amazon Resource Name (ARN) of the newly created collection. An example response is shown following:

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

- ListCollections (p. 258)
Amazon Rekognition returns a list of collections. The following is an example response:

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

- **DeleteCollection (p. 186)**

  Amazon Rekognition deletes the specified collection.

For information about storing faces in a collection, see Storing Faces in a Face Collection (p. 107). For information about searching faces, see Searching for Faces with Rekognition Image Collection (p. 116) and the section called “Searching Faces with Rekognition Video” (p. 121).

### Creating a Collection

You can use the CreateCollection (p. 180) operation to create a collection.

The following procedures show how you can use the operation with the AWS CLI, the AWS SDK for Java. For more information, see Managing Collections (p. 102).

**To create a collection (AWS CLI)**

1. On the command line, type the following command. Replace `collectionname` with the name of the collection you want to create.

   ```
   aws rekognition create-collection \
   --collection-id "collectionname" \
   --region us-west-2 \
   --profile adminuser
   ```

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

**To create a collection (AWS SDK for Java)**

- To create a collection, use the following AWS SDK for Java example code.

```java
package com.amazonaws.samples;
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;

public class CreateCollection {
    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();

String collectionId = "exampleCollection";
    System.out.println("Creating collection: " +
    collectionId);
    CreateCollectionRequest request = new CreateCollectionRequest()
        .withCollectionId(collectionId);
    CreateCollectionResult createCollectionResult =
        amazonRekognition.createCollection(request);
    System.out.println("CollectionArn : " +
        createCollectionResult.getCollectionArn());
    System.out.println("Status code : " +
        createCollectionResult.getStatusCode().toString());
}

Listing Collections

You can use the ListCollections (p. 258) operation to list the available collections.

The following procedures show how you can use the operation with the AWS CLI, the AWS SDK for Java.

For more information, see Managing Collections (p. 102).

To list collections (AWS CLI)

1. On the command line, type the following command. The command returns a list of collections in the
   us-west-2 region.

   ```bash
   aws rekognition list-collections \
   --region us-west-2 \
   --profile adminuser
   ```

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

To list collections (AWS SDK for Java)

- To list collections, use the following AWS SDK for Java example code.
package com.amazonaws.samples;

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.ListCollectionsRequest;
import com.amazonaws.services.rekognition.model.ListCollectionsResult;

public class ListCollections {

    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();

        System.out.println("Listing collections");
        int limit = 1;
        ListCollectionsResult listCollectionsResult = null;
        String paginationToken = null;
        do {
            if (listCollectionsResult != null) {
                paginationToken = listCollectionsResult.getNextToken();
            }
            ListCollectionsRequest listCollectionsRequest = new ListCollectionsRequest()
                    .withMaxResults(limit)
                    .withNextToken(paginationToken);

            listCollectionsResult = amazonRekognition.listCollections(listCollectionsRequest);

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

You can use the DeleteCollection (p. 186) operation to delete a collection.

The following procedures show how you can use the operation with the AWS CLI, the AWS SDK for Java.

For more information, see Managing Collections (p. 102).

To delete a collection (AWS CLI)

1. On the command line, type the following command. Replace collectionname with the name of the collection you want to delete.

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

2. To run the command, choose Enter. The JSON output for the DeleteCollection operation is displayed.

To delete a collection (AWS SDK for Java)

- To delete a collection, use the following AWS SDK for Java example code.

```
package com.amazonaws.samples;

import java.util.List;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.DeleteCollectionRequest;
import com.amazonaws.services.rekognition.model.DeleteCollectionResult;

public class DeleteCollection {
    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();
    }
}
```
```java
String collectionId = "exampleCollection";
System.out.println("Deleting collections");
DeleteCollectionRequest request = new DeleteCollectionRequest()
    .withCollectionId(collectionId);
DeleteCollectionResult deleteCollectionResult =
    amazonRekognition.deleteCollection(request);
System.out.println(collectionId + ": " + deleteCollectionResult.getStatusCode()
    .toString());
}
```
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.

```json
{
  "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,
      }
    }
  ]
}
```
To get all facial information, specify 'ALL' for the `DetectionAttributes` request parameter. For example, 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.
"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": 0.593608562011719,
    "Type": "mouthRight"
  },
  {
    "Y": 0.4163532555103302,
    "X": 0.369786858586548,
    "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": 0.639077826309204,
"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,
Adding Faces to a Collection

You can use the IndexFaces (p. 250) operation to detect faces in an image and add them to a collection. For each face detected, Amazon Rekognition 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.

The following procedures show how you can use the operation with the AWS CLI, the AWS SDK for Java. For more information, see Managing Collections (p. 102).

To add faces to a collection (AWS CLI)

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. 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. Replace `collection-id` with the name of the collection you want the face to be stored in.

   ```
   aws rekognition index-faces 
   --image '{"S3Object":{"Bucket":"bucketname","input.jpg":"file-name"}}' 
   --collection-id "collection-id" 
   --detection-attributes "ALL" 
   --external-image-id "example-image.jpg" 
   --region us-east-1 
   ```
3. To run the command, choose Enter. The JSON output for the `IndexFaces` operation is displayed.

To add faces to a collection (AWS SDK for Java)

- To delete a collection, use the following AWS SDK for Java example code.

```java
package com.amazonaws.samples;
import java.util.List;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
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.S3Object;
public class AddFaces {
    public static final String collectionId = "exampleCollection";
    public static final String bucket = "photo-label-detect";
    public static final String fileName = "filename.jpg";

    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();

        Image image=new Image()
            .withS3Object(new S3Object()
                .withBucket(bucket)
                .withName(fileName));

        IndexFacesRequest indexFacesRequest = new IndexFacesRequest()
            .withImage(image)
            .withCollectionId(collectionId)
            .withExternalImageId(fileName)
            .withDetectionAttributes("ALL");

        IndexFacesResult indexFacesResult=amazonRekognition.indexFaces(indexFacesRequest);
    }
}
```
Listing Faces in a Collection

You can use the IndexFaces (p. 250) operation to list the faces in a collection.

The following procedures show how you can use the operation with the AWS CLI, the AWS SDK for Java.

For more information, see Managing Collections (p. 102).

To list faces in a collection (AWS CLI)

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. On the command line, type the following command. Replace collection-id with the name of the collection you want to list.

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

3. To run the command, choose Enter. The JSON output for the ListFaces operation is displayed.

   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.

To list faces in a collection (AWS SDK for Java)

- To list faces in a collection, use the following AWS SDK for Java example code.

```java
package com.amazonaws.samples;
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.ListFacesRequest;
import com.amazonaws.services.rekognition.model.ListFacesResult;
import com.fasterxml.jackson.databind.ObjectMapper;
```
public class ListFaces {
    public static final String collectionId = "exampleCollection";
    public static final String bucket = "photo-label-detect";

    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();

        ListFacesResult listFacesResult = null;
        System.out.println("Faces in collection " + collectionId);

        String paginationToken = null;
        do {
            if (listFacesResult != null) {
                paginationToken = listFacesResult.getNextToken();
            }

            ListFacesRequest listFacesRequest = new ListFacesRequest()
                .withCollectionId(collectionId)
                .withMaxResults(1)
                .withNextToken(paginationToken);

            listFacesResult = amazonRekognition.listFaces(listFacesRequest);
            List<Face> faces = listFacesResult.getFaces();
            for (Face face: faces) {
                System.out.println(objectMapper.writerWithDefaultPrettyPrinter()
                                  .writeValueAsString(face));
            }
        } while (listFacesResult != null && listFacesResult.getNextToken() !=
            null);
    }
}
Searching for Faces with Rekognition Image 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 Collections (p. 102) and Storing Faces in a Face Collection (p. 107). With Amazon Rekognition, you can do the following:

- **Search a face collection given an image (SearchFacesByImage (p. 282))** – 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. 277))** – 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": [ ...
```
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 Compare Faces in Images (p. 89).
- `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.

**Searching for a Face Using its Face ID**

You can use the `SearchFaces` (p. 277) operation to search for faces in a collection that match a supplied face ID.

The following procedures show how you can use the operation with the AWS CLI, the AWS SDK for Java.

The face ID is returned in the `IndexFaces` (p. 250) operation response when the face is detected and added to a collection. For more information, see Storing Faces in a Face Collection (p. 107).

For more information, see Managing Collections (p. 102).

**To search for a face in a collection using its face ID (AWS CLI)**

1. On the command line, type the following command. Replace `faceid` with the face identifier you want to search for and replace `collection-id` with the collection you want to search in.

   ```bash
   aws rekognition search-faces
   --face-id face-id
   ```
To search for a face in a collection using its face ID (AWS SDK for Java)

- To search for a face in a collection using its face ID, use the following AWS SDK for Java example code.

```java
package com.amazonaws.samples;

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.SearchFacesRequest;
import com.amazonaws.services.rekognition.model.SearchFacesResult;
import com.fasterxml.jackson.databind.ObjectMapper;

public class SearchFaceMatchingId {
    public static final String collectionId = "exampleCollection";
    public static final String bucket = "photo-label-detect";
    public static final String faceId = "3689c89f-08a6-527d-8dc8-8a2d48cebb58";

    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();

        ObjectMapper objectMapper = new ObjectMapper();
        // Search collection for faces matching the face id.

        SearchFacesRequest searchFacesRequest = new SearchFacesRequest()
                .withCollectionId(collectionId)
                .withFaceId(faceId)
                .withFaceMatchThreshold(70F)
                .withMaxFaces(2);

        SearchFacesResult searchFacesByImageResult =
                amazonRekognition.searchFaces(searchFacesRequest);

        System.out.println("Face matching faceId " + faceId);
        List < FaceMatch > faceImageMatches = searchFacesByImageResult.getFaceMatches();
        for (FaceMatch face: faceImageMatches) {
```
Searching for a Face Using an Image

You can use the `SearchFacesByImage` operation to search for faces in a collection that match
the largest face in a supplied image.

The following procedures show how you can use the operation with the AWS CLI, the AWS SDK for Java.

For more information, see Searching for Faces with Rekognition Image Collection (p. 116).

**To search for a face in a collection using an image (AWS CLI)**

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. On the command line, type the following command. Replace `bucket-name` with the S3 bucket you
   are using. Replace `example.jpg` with the image file name that contains the face you want to search
   for. Replace `collection-id` with the collection you want to search in.

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

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

**To search for a face in a collection using an image (AWS SDK for Java)**

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. To search for a face in a collection using an image, use the following AWS SDK for Java example
code.

```java
package com.amazonaws.samples;

import java.util.List;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
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.S3Object;
import com.amazonaws.services.rekognition.model.SearchFacesByImageRequest;
import com.amazonaws.services.rekognition.model.SearchFacesByImageResult;
import com.fasterxml.jackson.databind.ObjectMapper;

public class SearchFacesMatchingImage {

    public static final String collectionId = "exampleCollection";
    public static final String bucket = "photo-label-detected";
    public static final String fileName = "filename.jpg";

    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();

        ObjectMapper objectMapper = new ObjectMapper();

        // Get an image object from S3 bucket.
        Image image = new Image()
                .withS3Object(new S3Object()
                        .withBucket(bucket)
                        .withName(fileName));

        // Search collection for faces similar to the largest face in the image.
        SearchFacesByImageRequest searchFacesByImageRequest = new SearchFacesByImageRequest()
                .withCollectionId(collectionId)
                .withImage(image)
                .withFaceMatchThreshold(70F)
                .withMaxFaces(2);

        SearchFacesByImageResult searchFacesByImageResult =
                amazonRekognition.searchFacesByImage(searchFacesByImageRequest);

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

The 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.
Searching for Faces with Rekognition Video

You can search a collection for faces that match faces of persons detected in a stored video or a streaming video. This section covers searching for faces in a stored video. For information about searching for faces in a streaming video, see Working with Streaming Videos (p. 68).

The faces you search for must first be indexed into a collection by using IndexFaces (p. 250). For more information see Adding Faces to a Collection (p. 112). Rekognition Video face searching follows the same asynchronous workflow as other Rekognition Video operations that analyze videos stored in an Amazon S3 bucket. For more information, see Calling Rekognition Video Operations (p. 53).

To start searching for faces in a stored video, call StartFaceSearch (p. 299). After you get the the completion status from the Amazon SNS topic, you call GetFaceSearch (p. 236) to get the search results. For an example, see Searching Stored Videos for Faces (SDK for Java) (p. 124).

The following is an example JSON request for StartFaceSearch.

```json
{
    "Video": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "video.mp4"
        }
    },
    "CollectionId": "VideoFaces",
    "NotificationChannel": {
        "SNSTopicArn": "arn:aws:sns:us-east-1:nnnnnnnnnnnn:xxxxxxx",
        "RoleArn": "arn:aws:iam::nnnnnnnnnnnn:role/xxxxxxxx"
    }
}
```

The request includes the collection to use for the search, the bucket the video is stored in, and the Amazon SNS topic to which Amazon Rekognition publishes the completion status of the search.

Once you have the completion status from the AWS SMS topic, you get the search results by calling GetFaceSearch. The following is an example JSON response from GetFaceSearch.

```json
{
    "JobStatus": "SUCCEEDED",
    "NextToken": "IJdbzkZfvBRqj8GPV82BPiZKkLOGCqDiwNZG/gQsEE5faTVK9JHOz/xxxxxxxxxxxxxx",
    "Persons": [
        {
            "FaceMatches": [
                {
                    "Face": {
                        "BoundingBox": {
                            "Height": 0.527472972869873,
                            "Left": 0.33530598878860474,
                            "Top": 0.2161169946193695,
                            "Width": 0.35503000020980835
                        },
                        "Confidence": 99.90239715576172,
                        "ExternalImageId": "image.PNG",
                        "FaceId": "a2f2e224-bfba-456c-b360-7c00241e5e2d",
                        "ImageId": "eb57ed44-8d8d-5ec5-90b8-6d190daff4c3"
                    },
                    "Similarity": 98.40909576416016
                }
            ]
        }
    ],
    "Person": {
```

121
"BoundingBox": {
  "Height": 0.8694444298744202,
  "Left": 0.2473958283662796,
  "Top": 0.10092592239379883,
  "Width": 0.49427083134651184
},
"Face": {
  "BoundingBox": {
    "Height": 0.23000000417232513,
    "Left": 0.42500001192092896,
    "Top": 0.16333332657814026,
    "Width": 0.12937499582767487
  },
  "Confidence": 99.97504425048828,
  "Landmarks": [
    {
      "Type": "eyeLeft",
      "X": 0.46415066719055176,
      "Y": 0.2572723925113678
    },
    {
      "Type": "eyeRight",
      "X": 0.5068183541297913,
      "Y": 0.23705792427062988
    },
    {
      "Type": "nose",
      "X": 0.4976589977412415,
      "Y": 0.28383663296699524
    },
    {
      "Type": "mouthLeft",
      "X": 0.487221896648407,
      "Y": 0.3452930748462677
    },
    {
      "Type": "mouthRight",
      "X": 0.5142884850502014,
      "Y": 0.33167609572410583
    }
  ],
  "Pose": {
    "Pitch": 15.966927528381348,
    "Roll": -15.547388076782227,
    "Yaw": 11.34195613861084
  },
  "Quality": {
    "Brightness": 44.8022308396094,
    "Sharpness": 99.95819854736328
  }
},
"Index": 0,
"Timestamp": 0
},
"Person": {
  "BoundingBox": {
    "Height": 0.2177777737379074,
    "Left": 0.7593749761581421,
    "Top": 0.13333334028720856,
    "Width": 0.12250000238418579
  },
  "Face": {
    "BoundingBox": {
      "Height": 0.2177777737379074,
Amazon Rekognition Developer Guide
Searching Faces with Rekognition Video
"Left": 0.7593749761581421,
"Top": 0.13333334028720856,
"Width": 0.12250000238418579

},
"Confidence": 99.63436889648438,
"Landmarks": [
{
"Type": "eyeLeft",
"X": 0.8005779385566711,
"Y": 0.20915353298187256
},
{
"Type": "eyeRight",
"X": 0.8391435146331787,
"Y": 0.21049551665782928
},
{
"Type": "nose",
"X": 0.8191410899162292,
"Y": 0.2523227035999298
},
{
"Type": "mouthLeft",
"X": 0.8093273043632507,
"Y": 0.29053622484207153
},
{
"Type": "mouthRight",
"X": 0.8366993069648743,
"Y": 0.29101791977882385
}
],
"Pose": {
"Pitch": 3.165884017944336,
"Roll": 1.4182015657424927,
"Yaw": -11.151537895202637
},
"Quality": {
"Brightness": 28.910892486572266,
"Sharpness": 97.61507415771484
}

},
"Index": 1

},
{

},
"Timestamp": 0
"Person": {
"BoundingBox": {
"Height": 0.8388888835906982,
"Left": 0,
"Top": 0.15833333134651184,
"Width": 0.2369791716337204
},
"Face": {
"BoundingBox": {
"Height": 0.20000000298023224,
"Left": 0.029999999329447746,
"Top": 0.2199999988079071,
"Width": 0.11249999701976776
},
"Confidence": 99.85971069335938,
"Landmarks": [
{
"Type": "eyeLeft",
"X": 0.06842322647571564,

123


The response includes an array of persons, Persons, detected in the video whose face(s) matches a face in the input collection. An array element, PersonMatch (p. 354), exists for each time the person is matched in the video. Each PersonMatch includes an array of face matches from the input collection, FaceMatch (p. 336), information about the matched person, PersonDetail (p. 352), and the time the person was matched in the video.

Searching Stored Videos for Faces (SDK for Java)

The following procedure shows how to search a collection for faces that match the faces of persons detected in a video. The procedure also shows how to get the tracking data for persons matched in the video. The video must be stored in an Amazon S3 bucket. The procedure expands on the code in Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61) which uses an Amazon Simple Queue Service queue to get the completion status of a video analysis request.
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. 11). 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. 33).

To search a video

1. Create a collection (p. 103).
2. Index a face into the collection (p. 112).
3. Perform Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61).
4. Add the following code to the project created in step 3.

```java
// Face collection search in video
private static void StartFaceSearchCollection(String bucket, String video) throws Exception{
    StartFaceSearchRequest req = new StartFaceSearchRequest()
        .withCollectionId("CollectionId")
        .withVideo(new Video()
            .withS3Object(new S3Object()
                .withBucket(bucket)
                .withName(video)
            )
        )
        .withNotificationChannel(channel);
    StartFaceSearchResult startPersonCollectionSearchResult = rek.startFaceSearch(req);
    startJobId=startPersonCollectionSearchResult.getJobId();
}

private static void GetResultsFaceSearchCollection() throws Exception{
    GetFaceSearchResult faceSearchResult=null;
    int maxResults=10;
    String paginationToken=null;
    do {
        if (faceSearchResult !=null){
            paginationToken = faceSearchResult.getNextToken();
        }
        faceSearchResult  = rek.getFaceSearch(
            new GetFaceSearchRequest()
                .withJobId(startJobId)
                .withMaxResults(maxResults)
                .withNextToken(paginationToken)
                .withSortBy(FaceSearchSortBy.TIMESTAMP)
        );
        VideoMetadata videoMetaData=faceSearchResult.getVideoMetadata();
        System.out.println("Format: " + videoMetaData.getFormat());
        System.out.println("Codec: " + videoMetaData.getCodec());
    }
```
System.out.println("Duration: " + videoMetaData.getDurationMillis());
System.out.println("FrameRate: " + videoMetaData.getFrameRate());
System.out.println();

// Show search results
List<PersonMatch> matches = faceSearchResult.getPersons();
for (PersonMatch match : matches) {
    long seconds = match.getTimestamp() / 1000;
    System.out.print("Sec: " + Long.toString(seconds) + " + ");
    System.out.println("Person number: " + match.getPerson().getIndex());
    List<FaceMatch> faceMatches = match.getFaceMatches();
    System.out.println("Matches in collection...");
    for (FaceMatch faceMatch : faceMatches) {
        Face face = faceMatch.getFace();
        System.out.println("Face Id: " + face.getFaceId());
        System.out.println("Similarity: " + faceMatch.getSimilarity().toString());
    }
    System.out.println();
}
System.out.println();
while (faceSearchResult != null && faceSearchResult.getNextToken() != null);

5. In the function main, replace the line
   StartLabels("bucket","file");
   with
   StartFaceSearchCollection("bucket","file");

   If you have already run a video example other than Analyzing a Video Stored in an Amazon S3
   Bucket with the AWS SDK for Java (p. 61), the function name to replace will be different.

6. In the call to StartFaceSearchCollection, replace bucket and file with the Amazon S3
   bucket name and the file name of the video you want to search.

7. In the function main, replace the line
   GetResultsLabels();
   with
   GetResultsFaceSearchCollection();

   If you have already run a video example other than Analyzing a Video Stored in an Amazon S3
   Bucket with the AWS SDK for Java (p. 61), the function name to replace will be different.

8. In the function StartFaceSearchCollection, change the value of CollectionId to the name
   of the collection you created in step 1.

9. Run the code. A list of persons in the video whose faces match those in the input collection
   is displayed. The tracking data for each matched person is also displayed.
Tracking People

Rekognition Video can track people in videos and provide information such as:

- The location of the person in the video frame at the time they were tracked.
- Facial landmarks such as the position of the left eye, when detected.

With Rekognition Video you can also search a video for people whose facial metadata is stored in a collection. For more information, see Searching for Faces with Rekognition Video (p. 121).

Rekognition Video person tracking in videos is an asynchronous operation. To start the tracking of people in videos call StartPersonTracking (p. 307). Rekognition Video publishes the completion status of the video analysis to an Amazon Simple Notification Service topic. If the video analysis is successful, call GetPersonTracking (p. 245) to get results of the video analysis.

For information about calling the video recognition API operations, see Calling Rekognition Video Operations (p. 53). For an example, see Tracking People through a Stored Video (SDK for Java) (p. 129).

GetPersonTracking returns an array, Persons, of PersonDetection (p. 353) objects which contain details about tracked people and the times they are tracked in the video.

You can sort Persons by using the SortBy input parameter. Specify TIMESTAMP to sort the elements by the time people are detected in the video. Specify INDEX to sort by people tracked in the video. Within each set of results for a person, the elements are sorted by descending confidence in the accuracy of the tracking. By default, Persons is returned sorted by TIMESTAMP. The following example is the JSON response from GetPersonDetection. The results are sorted by the time, in milliseconds since the start of the video, people are tracked in the video.

```json
{
  "JobStatus": "SUCCEEDED",
  "NextToken": "AcDymG0fSSoaI6+BBypka5wVlqttysSFP8VwCuJMDLuj1QpFo/vf+mrQgBk8eUHiFflllR6g==",
  "Persons": [
    {
      "Person": {
        "BoundingBox": {
          "Height": 0.8787037134170532,
          "Left": 0.00572916679084301,
          "Top": 0.12129629403352737,
          "Width": 0.21666666865348816
        },
        "Face": {
          "BoundingBox": {
            "Height": 0.20000000298023224,
            "Left": 0.02999999329447746,
            "Top": 0.2199999988079071,
            "Width": 0.11249999701976776
          },
          "Confidence": 99.85971069335938,
          "Landmarks": [
            {
              "Type": "eyeLeft",
              "X": 0.06842322647571564,
              "Y": 0.3010137975215912
            },
            {
              "Type": "eyeRight",
              "X": 0.10543643683195114,
              "Y": 0.29697132110595703
            }
          ]
        }
      }
    }
  ]
}
```
{},
{
  "Type": "nose",
  "X": 0.09569807350635529,
  "Y": 0.33701086044311523
},
{
  "Type": "mouthLeft",
  "X": 0.0732642263174057,
  "Y": 0.3757539987564087
},
{
  "Type": "mouthRight",
  "X": 0.10589495301246643,
  "Y": 0.3722417950630188
}
],
"Pose": {
  "Pitch": -0.5589138865470886,
  "Roll": -5.1093974113464355,
  "Yaw": 18.69594955444336
},
"Quality": {
  "Brightness": 43.052337646484375,
  "Sharpness": 99.68138885498047
}
},
"Index": 0
},
"Timestamp": 0
},
{
  "Person": {
    "BoundingBox": {
      "Height": 0.9074074029922485,
      "Left": 0.24791666865348816,
      "Top": 0.09259258955717087,
      "Width": 0.375
    },
    "Face": {
      "BoundingBox": {
        "Height": 0.23000000417232513,
        "Left": 0.425000001192092896,
        "Top": 0.16333332657814026,
        "Width": 0.12037499582767487
      },
      "Confidence": 99.97504425048828,
      "Landmarks": [
        {
          "Type": "eyeLeft",
          "X": 0.46415066719055176,
          "Y": 0.2572723925113678
        },
        {
          "Type": "eyeRight",
          "X": 0.5068183541297913,
          "Y": 0.23705792427062988
        },
        {
          "Type": "nose",
          "X": 0.49765899777412415,
          "Y": 0.28383663296699524
        },
        {
          "Type": "mouthLeft",
          "X": 0.487221896648407,
Tracking People through a Stored Video (SDK for Java)

The following procedure shows how to track people through a video stored in an Amazon S3 bucket. The example expands on the code in Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java.
Java (p. 61) which uses an Amazon Simple Queue Service queue to get the completion status of a video analysis request.

**Prerequisites**

To run these procedures, you need to have the AWS SDK for Java installed. For more information, see Getting Started with Amazon Rekognition (p. 11). 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. 33). You also need an image file that contains one or more celebrity faces.

**To detect people in a video stored in an Amazon S3 bucket**

1. Perform Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61).
2. Add the following code to the project created in step one.

```java
//Persons========================================================================
private static void StartPersons(String bucket, String video) throws Exception{
    int maxResults=10;
    String paginationToken=null;
    StartPersonTrackingRequest req = new StartPersonTrackingRequest()
        .withVideo(new Video()
            .withS3Object(new S3Object()
                .withBucket(bucket)
                .withName(video)))
        .withNotificationChannel(channel);
    StartPersonTrackingResult startPersonDetectionResult =
        rek.startPersonTracking(req);
    startJobId=startPersonDetectionResult.getJobId();
}
private static void GetResultsPersons() throws Exception{
    int maxResults=10;
    String paginationToken=null;
    GetPersonTrackingResult personTrackingResult=null;
    do{
        if (personTrackingResult !=null){
            paginationToken = personTrackingResult.getNextToken();
        }
    }
    personTrackingResult = rek.getPersonTracking(new GetPersonTrackingRequest()
        .withJobId(startJobId)
        .withNextToken(paginationToken)
        .withSortBy(PersonTrackingSortBy.TIMESTAMP)
        .withMaxResults(maxResults));
    VideoMetadata videoMetaData=personTrackingResult.getVideoMetadata();
    System.out.println("Format: " + videoMetaData.getFormat());
    System.out.println("Codec: " + videoMetaData.getCodec());
    System.out.println("Duration: " + videoMetaData.getDurationMillis());
    System.out.println("FrameRate: " + videoMetaData.getFrameRate());
```
//Show persons, confidence and detection times
List<PersonDetection> detectedPersons = personTrackingResult.getPersons();
for (PersonDetection detectedPerson : detectedPersons) {
    long seconds = detectedPerson.getTimestamp() / 1000;
    System.out.print("Sec: "+ Long.toString(seconds) + " ");
    System.out.println("Person Identifier: "+
    detectedPerson.getPerson().getIndex());
    System.out.println();
}  
while (personTrackingResult != null && personTrackingResult.getNextToken() != null);

3. In the function main, replace the line
StartLabels("bucket","file");
with
StartPersons("bucket","file");
If you have already run a video example other than Analyzing a Video Stored in an Amazon S3
Bucket with the AWS SDK for Java (p. 61), the function name to replace will be different.

4. In the call to StartPersons, replace bucket and file with the Amazon S3 bucket name and file
name of the video you want to analyze.

5. In the function main, replace the line
GetResultsLabels();
with
GetResultsPersons();
If you have already run a video example other than Analyzing a Video Stored in an Amazon S3
Bucket with the AWS SDK for Java (p. 61), the function name to replace will be different.

6. Run the code. The unique identifiers for tracked people are shown along with the time, in seconds,
the people were tracked.
Recognizing Celebrities

Amazon Rekognition can recognize thousands of celebrities in a wide range of categories, such as entertainment and media, sports, business, and politics. With Amazon Rekognition you can recognize celebrities in images and in stored videos. You can also get additional information for recognized celebrities.

Topics

- Celebrities in Images (p. 132)
- Celebrities in Video (p. 134)
- Recognizing Celebrities in an Image (p. 136)
- Recognizing Celebrities in a Stored Video (SDK for Java) (p. 138)
- Getting Information about a Celebrity (p. 146)

Celebrities in Images

To recognize celebrities within images and get additional information about recognized celebrities, use the RecognizeCelebrities (p. 271) 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 100 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. For an example, see Recognizing Celebrities in an Image (p. 136).

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. 219). 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.

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. 37). 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:

```json
{
    "CelebrityFaces": [{
        "Face": {
            "BoundingBox": {
                "Height": 0.6171230077774353,
                "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": 0.3737893104553227
}, {
  "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": 0.5663404870033264,
    "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": 0.69520205259332312,
    "Y": 0.5600858926773071
  }],}]}
Celebrities in Video

To recognize celebrities in a video stored in an Amazon S3 bucket, you use `StartCelebrityRecognition` (p. 288) and `GetCelebrityRecognition` (p. 222). Analyzing videos stored in an Amazon S3 bucket is an asynchronous workflow. For more information, see `Working with Stored Videos` (p. 51). For an example, see `Recognizing Celebrities in a Stored Video` (SDK for Java) (p. 138).

The response from `GetCelebrityDetection` includes an array, `Celebrities`, of celebrities (`CelebrityRecognition` (p. 322)) recognized in the video and the time(s) they were recognized. The following is an example JSON response.

```json
{
  "Celebrities": [
    {
      "Celebrity": {
        "BoundingBox": {
          "Height": 0.8842592835426331,
          "Left": 0,
          "Top": 0.11574073880910873,
          "Width": 0.24427083134651184
        },
        "Confidence": 0.699999998079071,
        "Face": {
          "BoundingBox": {
            "Height": 0.20555555820465088,
            "Left": 0.029374999925494194,
            "Top": 0.22333332896232605,
            "Width": 0.11562500149011612
          },
          "Confidence": 99.89837646484375,
          "Landmarks": [
            {
              "Type": "eyeLeft",
              "Size": 0.029374999925494194,
              "Location": {
                "X": 0.029374999925494194,
                "Y": 0.22333332896232605
              }
            }
          ]
        }
      }
    }]
}
```
The response includes the following:

- **Recognized celebrities** – Celebrities is an array of celebrities and the time(s) they are recognized in a video. An CelebrityRecognition (p. 322) object exists for each time the celebrity is recognized in the video. Each CelebrityRecognition contains information about a recognized celebrity (CelebrityDetail (p. 320)) and the time (Timestamp) the celebrity was recognized in the video. Timestamp is measured in milliseconds from the start of the video.

- **CelebrityDetail** – Contains information about a recognized celebrity. It includes the celebrity name (Name), identifier (Id), and a list of URLs pointing to related content (Urls). It also includes the
bounding box for the celebrity's body, the confidence Rekognition Video has in the accuracy of the recognition, and details about the celebrity's face, FaceDetail (p. 332). If you need to get the related content later, you can use ID with GetCelebrityInfo (p. 219).

- **VideoMetadata** – Information about the video that was analyzed.

## Recognizing Celebrities in an Image

You can use the RecognizeCelebrities (p. 271) 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 this procedure, you need to have the AWS CLI and AWS SDK for Java installed. For more information, see Getting Started with Amazon Rekognition (p. 11). 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. 33). 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.

   ```bash
   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. 146).

### 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.

   ```java
   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;
   ```

---

136
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));

        RecognizeCelebritiesResult result = amazonRekognition.recognizeCelebrities(request);

        // Display recognized celebrity information
        List<Celebrity> celebs = result.getCelebrityFaces();
        System.out.println("Found " + celebs.size() + " celebrity(a)es.");
        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() + " top: " + boundingBox.getTop().toString());
            System.out.println("Further information (if available): ");
            for (String url : celebrity.getUrls()) {
                System.out.println(url);
            }
        }
    }
}
Recognizing Celebrities in a Stored Video (SDK for Java)

You can recognize celebrities in a video stored in an Amazon S3 bucket. This procedure expands on the code in Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61) which uses an Amazon Simple Queue Service queue to get the completion status of a video analysis request.

Prerequisites

To run this procedure, you need to have the AWS SDK for Java installed. For more information, see Getting Started with Amazon Rekognition (p. 11). 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. 33). You also need an video file that contains one or more celebrity faces.

To detect celebrities in a video stored in an Amazon S3 bucket

1. Perform Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61).
2. Add the following code to the project created in step one.

```java
private static void StartCelebrities(String bucket, String video) throws Exception{
    StartCelebrityRecognitionRequest req = new StartCelebrityRecognitionRequest()
        .withVideo(new Video()
            .withS3Object(new S3Object()
                .withBucket(bucket)
                .withName(video))
            .withNotificationChannel(channel);
    StartCelebrityRecognitionResult startCelebrityRecognitionResult = rek.startCelebrityRecognition(req);
    startJobId=startCelebrityRecognitionResult.getJobId();
}
```

```java
private static void GetResultsCelebrities() throws Exception{
    int maxResults=10;
    String paginationToken=null;
    GetCelebrityRecognitionResult celebrityRecognitionResult=null;
```
do{
    if (celebrityRecognitionResult !=null){
        paginationToken = celebrityRecognitionResult.getNextToken();
    }
    celebrityRecognitionResult = rek.getCelebrityRecognition(new
                          GetCelebrityRecognitionRequest()
                          .withJobId(startJobId)
                          .withNextToken(paginationToken)
                          .withSortBy(CelebrityRecognitionSortBy.TIMESTAMP)
                          .withMaxResults(maxResults));

    System.out.println("File info for page");
    VideoMetadata videoMetaData=celebrityRecognitionResult.getVideoMetadata();
    System.out.println("Format: " + videoMetaData.getFormat());
    System.out.println("Codec: " + videoMetaData.getCodec());
    System.out.println("Duration: " + videoMetaData.getDurationMillis());
    System.out.println("FrameRate: " + videoMetaData.getFrameRate());
    System.out.println("Job");
    System.out.println("Job status: " +
                          celebrityRecognitionResult.getJobStatus());
    System.out.println("Failure reason: " +
                          celebrityRecognitionResult.getFailureReason());

    //Show celebrities
    List<CelebrityRecognition> celebs=
                          celebrityRecognitionResult.getCelebrities();
    for (CelebrityRecognition celeb: celebs) {
        long seconds=celeb.getTimestamp()/1000;
        System.out.print("Sec: " + Long.toString(seconds) + " ");
        CelebrityDetail details=celeb.getCelebrity();
        System.out.println("Name: " + details.getName());
        System.out.println("Id: " + details.getId());
        System.out.println();
    }
} while (celebrityRecognitionResult !=null &&
        celebrityRecognitionResult.getNextToken() != null);

3. In the function main, replace the line
   StartLabels("bucket","file");
   with
   StartCelebrities("bucket","file");
   If you have already run a video example other than Analyzing a Video Stored in an Amazon S3
   Bucket with the AWS SDK for Java (p. 61), the function name to replace will be different.

4. In the call to StartCelebrities, replace bucket and file with the Amazon S3 bucket name and
   file name of the video you want to analyze.

5. In the function main, replace the line
   GetResultsLabels();
   with
Prerequisites

GetResultsCelebrities();

If you have already run a video example other than

This procedure shows you how to detect labels in a video by using Rekognition Video label detection operations, a video stored in an Amazon S3 bucket, and an Amazon SNS topic. The procedure also shows how to use an Amazon SQS queue to get the completion status from the Amazon SNS topic. For more information, see ??? (p. 53). You aren't restricted to using an Amazon SQS queue. For example, you can use an AWS Lambda function to get the completion status. For more information, see Invoking Lambda functions using Amazon SNS notifications.

The procedure shows you how to use the Amazon SNS console to do the following:

• Create the Amazon SNS topic.

• Create the Amazon SQS queue.

• Give Rekognition Video permission to publish the completion status of a video analysis operation to the Amazon SNS topic.

• Subscribe the Amazon SQS queue to the Amazon SNS topic.

Note
This procedure uses a single Amazon SQS queue and a single Amazon SNS topic for all video analysis requests.

The example code in the procedure shows you how to do the following:

1. Start the video analysis request by calling StartLabelDetection (p. 303).

2. Get the completion status from the Amazon SQS queue. The sample tracks the job identifier (JobId) that's returned in StartLabelDetection and only gets the results for matching job identifiers that are read from the completion status. This is an important consideration if other applications are using the same queue and topic. Any other jobs are ignored.

3. Get and display the video analysis results by calling GetLabelDetection (p. 241).

Prerequisites

To run this procedure, you need to have the AWS SDK for Java installed. For more information, see Getting Started with Amazon Rekognition (p. 11). The AWS account that 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. 33).

To Detect Labels in a Video

1. If you haven't already, create an IAM service role to give Rekognition Video access to your Amazon SNS topics. Note the ARN. For more information, see ??? (p. 58).

2. Create an Amazon SNS topic by using the Amazon SNS console. Prepend the topic name with AmazonRekognition. Note the topic Amazon Resource Name (ARN).

3. Create an Amazon SQS standard queue by using the Amazon SQS console. Note the queue ARN.

4. Subscribe the queue to the topic you created in step 2.
5. Upload an .mp4, .mov or .avi format video file to your S3 Bucket. For test purposes, upload a video that's no longer than 30 seconds in length. For instructions, see Uploading Objects into Amazon S3 in the Amazon Simple Storage Service Console User Guide.

6. Use the following AWS SDK for Java code to detect labels in a video.
   - Replace TopicArn, RoleArn, and QueueURL with the Amazon SNS topic ARN, IAM role ARN, and Amazon SQS queue URL that you previously noted.
   - Replace Bucket and VideoFile with the bucket and video file name that you uploaded in step 5.
   - Replace Endpoint and Region with the AWS endpoint and region that you're using.
   - Update .withRegion parameters to the region you are using.
   - Change RekognitionUser to an AWS account that has permissions to call Rekognition Video operations.

```java
package com.amazonaws.samples;
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.regions.Regions;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.CelebrityDetail;
import com.amazonaws.services.rekognition.model.CelebrityRecognition;
import com.amazonaws.services.rekognition.model.CelebrityRecognitionSortBy;
import com.amazonaws.services.rekognition.model.ContentModerationDetection;
import com.amazonaws.services.rekognition.model.ContentModerationSortBy;
import com.amazonaws.services.rekognition.model.Face;
import com.amazonaws.services.rekognition.model.FaceDetection;
import com.amazonaws.services.rekognition.model.FaceMatch;
import com.amazonaws.services.rekognition.model.GetCelebrityRecognitionRequest;
import com.amazonaws.services.rekognition.model.GetCelebrityRecognitionResult;
import com.amazonaws.services.rekognition.model.GetContentModerationRequest;
import com.amazonaws.services.rekognition.model.GetContentModerationResult;
import com.amazonaws.services.rekognition.model.GetFaceDetectionRequest;
import com.amazonaws.services.rekognition.model.GetFaceDetectionResult;
import com.amazonaws.services.rekognition.model.GetFaceSearchRequest;
import com.amazonaws.services.rekognition.model.GetFaceSearchResult;
import com.amazonaws.services.rekognition.model.GetLabelDetectionRequest;
import com.amazonaws.services.rekognition.model.GetLabelDetectionResult;
```

public class VideoDetect {
    private static AmazonSNS sns = null;
    private static AmazonSQS sqs = null;
    private static AmazonRekognition rek = null;
    private static NotificationChannel channel = new NotificationChannel()
        .withSNSTopicArn("TopicArn")
        .withRoleArn("RoleArn");

    private static String queueUrl = "QueueURL";
    private static String startJobId = null;

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

try {
    credentials = new ProfileCredentialsProvider("RekognitionUser").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);
    }

    sns = AmazonSNSClientBuilder
        .standard()
        .withRegion(Regions.US_EAST_1)
        .withCredentials(new AWSStaticCredentialsProvider(credentials))
        .build();

    sqs = AmazonSQSClientBuilder
        .standard()
        .withRegion(Regions.US_EAST_1)
        .withCredentials(new AWSStaticCredentialsProvider(credentials))
        .build();

    rek = AmazonRekognitionClientBuilder.standard().withCredentials( new ProfileCredentialsProvider("RekognitionUser"))
        .withEndpointConfiguration(new EndpointConfiguration("Endpoint", "Region"))
        .build();

    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    StartLabels("Bucket", "VideoFile");
    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    System.out.println("Waiting for job: " + startJobId);
    //Poll queue for messages
    List<Message> messages=null;
    int dotLine=0;
    boolean jobFound=false;

    //Loop until the job status is published. Ignore other messages in queue.
    do{
        //Get messages.
        do{
            messages = sqs.receiveMessage(queueUrl).getMessages();
            if (dotLine++<20){
                System.out.print(".");
            }else{
                System.out.println();
                dotLine=0;
            }
        }while(messages.isEmpty());

        System.out.println();

        //Loop through messages received.
        for (Message message: messages) {
            String notification = message.getBody();
        }
    }
// Get status and job id from notification.
ObjectMapper mapper = new ObjectMapper();
JsonNode jsonMessageTree = mapper.readTree(notification);
JsonNode messageBodyText = jsonMessageTree.get("Message");
ObjectMapper operationResultMapper = new ObjectMapper();
JsonNode jsonResultTree =
    operationResultMapper.readTree(messageBodyText.textValue());
JsonNode operationJobId = jsonResultTree.get("JobId");
JsonNode operationStatus = jsonResultTree.get("Status");
System.out.println("Job found was "+operationJobId);
// Found job. Get the results and display.
if(operationJobId.asText().equals(startJobId)){
    jobFound=true;
    System.out.println("Job id: " + operationJobId);
    System.out.println("Status : " +
    operationStatus.toString());
    if (operationStatus.asText().equals("SUCCEEDED")){
        //============================================
        GetResultsLabels();
        //============================================
    } else{
        System.out.println("Video analysis failed");
    }
}
sqs.deleteMessage(queueUrl,message.getReceiptHandle());
else{
    System.out.println("Job received was not job "+
    startJobId);
}
} while (!jobFound);
System.out.println("Done!");

private static void StartLabels(String bucket, String video)
    throws Exception{
    StartLabelDetectionRequest req = new
    StartLabelDetectionRequest()
        .withVideo(new Video()
            .withS3Object(new S3Object()
                .withBucket(bucket)
                .withName(video))
            .withMinConfidence(50F)
            .withJobTag("DetectingLabels")
            .withNotificationChannel(channel));

    StartLabelDetectionResult startLabelDetectionResult =
        rek.startLabelDetection(req);
    startJobId=startLabelDetectionResult.getJobId();
}

private static void GetResultsLabels() throws Exception{
    int maxResults=10;
String paginationToken=null;

GetLabelDetectionResult labelDetectionResult=null;

do {
    if (labelDetectionResult !=null){
        paginationToken = labelDetectionResult.getNextToken();
    }

    GetLabelDetectionRequest labelDetectionRequest= new GetLabelDetectionRequest()
        .withJobId(startJobId)
        .withSortBy(LabelDetectionSortBy.TIMESTAMP)
        .withMaxResults(maxResults)
        .withNextToken(paginationToken);

    labelDetectionResult = rek.getLabelDetection(labelDetectionRequest);

    VideoMetadata videoMetaData=labelDetectionResult.getVideoMetadata();

    System.out.println("Format: " + videoMetaData.getFormat());
    System.out.println("Codec: " + videoMetaData.getCodec());
    System.out.println("Duration: " + videoMetaData.getDurationMillis());
    System.out.println("FrameRate: " + videoMetaData.getFrameRate());

    //Show labels, confidence and detection times
    List<LabelDetection> detectedLabels=
        labelDetectionResult.getLabels();

    for (LabelDetection detectedLabel: detectedLabels) {
        long seconds=detectedLabel.getTimestamp()/1000;
        System.out.print("Sec: " + Long.toString(seconds) + " ");
        System.out.println("\t" +
            detectedLabel.getLabel().getName() + 
            " \t" +
            detectedLabel.getLabel().getConfidence().toString());
    }
} while (labelDetectionResult !=null &&
    labelDetectionResult.getNextToken() != null);

7. Build and run the code. The operation might take a while to finish. After it's finished, a list of the labels detected in the video is displayed.

6. Run the code. Information about the celebrities recognized in the video is shown.
Getting Information about a Celebrity

In these procedures, you get celebrity information by using the GetCelebrityInfo (p. 219) 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. 11). 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. 33).

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. 136).

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. 136).

   ```bash
   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. 136).

   ```java
   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);
           }
   ```

   "ID" is a placeholder for the actual celebrity ID you need to replace. The code snippet demonstrates how to use the AWS SDK to get celebrity information from Amazon Rekognition.
2. Run the example code. The celebrity name and information about the celebrity, if it is available, is displayed.
Detecting Unsafe Content

Amazon Rekognition can determine if an image or video contains explicit or suggestive adult content. Rekognition Image provides the DetectModerationLabels (p. 212) operation to detect unsafe content in images. Rekognition Video can be detect unsafe content asynchronously by using StartContentModeration (p. 291) and GetContentModeration (p. 227).

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.

<table>
<thead>
<tr>
<th>Top-Level Category</th>
<th>Second-Level Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit Nudity</td>
<td>Nudity</td>
</tr>
<tr>
<td></td>
<td>Graphic Male Nudity</td>
</tr>
<tr>
<td></td>
<td>Graphic Female Nudity</td>
</tr>
<tr>
<td></td>
<td>Sexual Activity</td>
</tr>
<tr>
<td></td>
<td>Partial Nudity</td>
</tr>
<tr>
<td>Suggestive</td>
<td>Female Swimwear Or Underwear</td>
</tr>
<tr>
<td></td>
<td>Male Swimwear Or Underwear</td>
</tr>
<tr>
<td></td>
<td>Revealing Clothes</td>
</tr>
</tbody>
</table>

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. 348) labels array returned by DetectModerationLabels (images) and by GetContentModeration (videos).

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 Unsafe Images (API) (p. 150).

**Note**

Rekognition Unsafe Image Detection API is not an authority on, or in any way purports to be an exhaustive filter of, explicit and suggestive adult content. Furthermore, the Unsafe Image Detection API does not detect whether an image includes illegal content (such as child pornography) or unnatural adult content.

**Topics**

- Unsafe Image Detection (p. 149)
- Unsafe Video Detection (p. 149)
- Detecting Unsafe Images (API) (p. 150)
Unsafe Image Detection

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.

```json
{
    "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:

- **Unsafe Image Detection 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.

Unsafe Video Detection

To detect suggestive and adult content in a video stored in an Amazon S3 bucket, you use StartContentModeration (p. 291) and GetContentModeration (p. 227). Analyzing videos stored in an Amazon S3 bucket is an asynchronous workflow. For more information, see Working with Stored Videos (p. 51). For an example, see Detecting Unsafe Stored Videos (SDK for Java) (p. 152). The following is an example response from GetContentModeration.

```json
{
    "JobStatus": "SUCCEEDED",
    "ModerationLabels": [
        {
            "ModerationLabel": {
                "Confidence": 93.02153015136719,
                "Name": "Male Swimwear Or Underwear",
```
The response from GetContentModeration is an array, ModerationLabels, of ContentModerationDetection (p. 326) objects. The array contains an element for each time a moderation label is detected. Within a ContentModerationDetectionObject object, ModerationLabel (p. 348) contains information for a detected item of suggestive or adult content. Timestamp is the time, in milliseconds from the start of the video, when the label was detected. The labels are organized hierarchically in the same manner as the moderation labels detected by image analysis. For more information, see Detecting Unsafe Content (p. 148).

Detecting Unsafe Images (API)

You can use the DetectModerationLabels (p. 212) 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. 11). 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. 33).

**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.

   ```bash
   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.

   ```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.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
   ```
```java
{ 
    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()
                               + "\nConfidence: " + label.getConfidence().toString() + "%"
                               + "\nParent:" + 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.

Detecting Unsafe Stored Videos (SDK for Java)

Rekognition Video can detect unsafe content in a video stored in an Amazon S3 bucket. The example expands on the code in Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61) which uses an Amazon Simple Queue Service queue to get the completion status of a video analysis request.

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. 11). 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. 33).

To detect unsafe content in a video stored in an Amazon S3 bucket
1. Perform Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61).
2. Add the following code to the project created in step one.

```java
//Content moderation
//==================================================================================================================================
private static void StartModerationLabels(String bucket, String video) throws Exception{

    StartContentModerationRequest req = new StartContentModerationRequest()
        .withVideo(new Video()
            .withS3Object(new S3Object()
                .withBucket(bucket)
                .withName(video))
            .withNotificationChannel(channel));

    StartContentModerationResult startModerationLabelDetectionResult = rek.startContentModeration(req);
    startJobId=startModerationLabelDetectionResult.getJobId();
}

private static void GetResultsModerationLabels() throws Exception{

    int maxResults=10;
    String paginationToken=null;
    GetContentModerationResult moderationLabelDetectionResult =null;

    do{
        if (moderationLabelDetectionResult !=null){
            paginationToken = moderationLabelDetectionResult.getNextToken();
        }
    moderationLabelDetectionResult = rek.getContentModeration(
        new GetContentModerationRequest()
            .withJobId(startJobId)
            .withNextToken(paginationToken)
            .withSortBy(ContentModerationSortBy.TIMESTAMP)
            .withMaxResults(maxResults));

    VideoMetadata videoMetaData=moderationLabelDetectionResult.getVideoMetadata();
    System.out.println("Format: " + videoMetaData.getFormat());
    System.out.println("Codec: " + videoMetaData.getCodec());
    System.out.println("Duration: " + videoMetaData.getDurationMillis());
    System.out.println("FrameRate: " + videoMetaData.getFrameRate());

    //Show moderated content labels, confidence and detection times
    List<ContentModerationDetection> moderationLabelsInFrames= moderationLabelDetectionResult.getModerationLabels();

    for (ContentModerationDetection label: moderationLabelsInFrames) {
        long seconds=label.getTimestamp()/1000;
        System.out.print("Sec: " + Long.toString(seconds));
    }
}
```
3. In the function `main`, replace the line

```
StartLabels("bucket","file");
```

with

```
StartModerationLabels("bucket","file");
```

If you have already run a video example other than Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61), the function name to replace will be different.

4. In the call to `StartModerationLabels`, replace `bucket` and `file` with the Amazon S3 bucket name and file name of the video you want to analyze.

5. In the function `main`, replace the line

```
GetResultsLabels();
```

with

```
GetResultsModerationLabels();
```

If you have already run a video example other than Analyzing a Video Stored in an Amazon S3 Bucket with the AWS SDK for Java (p. 61), the function name to replace will be different.

6. Run the code. A list of unsafe content labels detected in the video is shown.
Detecting Text

Amazon Rekognition Text in Image can detect text in images and convert it into machine-readable text. You can use the machine-readable text to implement solutions such as:

- Visual search. For example, retrieving and displaying images that contain the same text.
- Content insights. For example, providing insights into themes that occur in text recognized in extracted video frames. Your application can search recognized text for relevant content, such as news, sport scores, athlete numbers, and captions.
- Navigation. For example, developing a speech enabled mobile app for visually impaired people that recognizes the names of restaurants, shops, or street signs.
- Public safety and transportation support. For example, detecting car licence plate numbers from traffic camera images.
- Filtering. For example, filtering out personally identifiable information from images.

DetectText (p. 216) detects text in .jpeg or .png format images and supports most fonts including highly stylized ones. After detecting text, DetectText creates a representation of detected words and lines of text and shows the relationship between them. The DetectText API also tells you where the text is on an image.

Consider the following image:

The blue boxes represent information about the detected text and location of the text that the DetectText operation returns. To be detected, text must be within +/- 30 degrees orientation of the horizontal axis. DetectText categorizes recognized text as either a word or a line of text.

A word is one or more ISO basic latin script characters that are not separated by spaces. DetectText can detect up to 50 words in an image.

A line is a string of equally spaced words. A line isn't necessarily a complete sentence. For example, a driver's license number is detected as a line. A line ends when there is no aligned text after it. Also, a line ends when there is a large gap between words, relative to the length of the words. This means, depending on the gap between words, Amazon Rekognition may detect multiple lines in text aligned in the same direction. Periods don't represent the end of a line. If a sentence spans multiple lines, the DetectText operation returns multiple lines.

Amazon Rekognition can also detect numbers and common symbols such as @, /, $, %, -, _, +, *, and #.

Calling the DetectText Operation

In the DetectText operation, you supply an input image either as a byte64-encoded byte array or as an image stored in an Amazon S3 bucket. For an example, see ??? (p. 162).

The DetectText operation analyses the image and returns an array, TextDetections, where each element (TextDetection (p. 364)) represents a line or word detected in the image. For each element, DetectText returns the following information:
• The detected text (DetectedText)
• The relationships between words and lines (Id and ParentId)
• The location of text on the image (Geometry)
• The confidence Amazon Rekognition has in the accuracy of the detected text and bounding box (Confidence)
• The type of the detected text (Type)

**Detected Text**

Each TextDetection element contains recognized text (words or lines) in the DetectedText field. Returned text might include characters that makes a word unrecognizable. For example, C@t instead of Cat. To determine whether a TextDetection element represents a line of text or a word, use the Type field.

Each TextDetection element includes a percentage value that represents the degree of confidence that Amazon Rekognition has in the accuracy of the detected text and of the bounding box surrounding the text.

**Word and Line Relationships**

Each TextDetection element has an identifier field, Id. The Id shows the word's position in a line. If the element is a word, the parent identifier field, ParentId, identifies the line where the word was detected. The ParentId for a line is null. For example, the line “but keep” in the preceding image has the following the Id and ParentId values:

<table>
<thead>
<tr>
<th>Text</th>
<th>ID</th>
<th>Parent ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>but keep</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>but</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>keep</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>

**Text Location on an Image**

To determine where the recognized text is on an image, use the bounding box (Geometry (p. 340)) information returned by DetectText. The Geometry object contains two types of bounding box information for detected lines and words:

• An axis-aligned coarse rectangular outline in a BoundingBox (p. 317) object
• A finer-grained polygon made up of multiple X and Y coordinates in a Point (p. 355) array

The bounding box and polygon coordinates show where the text is located on the source image. The coordinate values are a ratio of the overall image size. For more information, see BoundingBox (p. 317).

**Operation Response**

The following JSON response from the DetectText operation shows the words and lines detected in the preceding image.

```
{
```

156
"TextDetections": [
  {
    "Confidence": 90.54900360107422,
    "DetectedText": "IT'S",
    "Geometry": {
      "BoundingBox": {
        "Height": 0.10317354649305344,
        "Left": 0.6677391529083252,
        "Top": 0.17569075524806976,
        "Width": 0.15113449096679688
      },
      "Polygon": [
        {
          "X": 0.6677391529083252,
          "Y": 0.17569075524806976
        },
        {
          "X": 0.8188736438751221,
          "Y": 0.17574213445186615
        },
        {
          "X": 0.8188582062721252,
          "Y": 0.278915673494339
        },
        {
          "X": 0.6677237153053284,
          "Y": 0.2788642942905426
        }
      ]
    },
    "Id": 0,
    "Type": "LINE"
  },
  {
    "Confidence": 59.411651611328125,
    "DetectedText": "I",
    "Geometry": {
      "BoundingBox": {
        "Height": 0.05955825746059418,
        "Left": 0.2763049304485321,
        "Top": 0.394121915102005,
        "Width": 0.026684552431106567
      },
      "Polygon": [
        {
          "X": 0.2763049304485321,
          "Y": 0.394121915102005
        },
        {
          "X": 0.30298948287963867,
          "Y": 0.3932435214519501
        },
        {
          "X": 0.30385109782218933,
          "Y": 0.45280176401138306
        },
        {
          "X": 0.27716654539108276,
          "Y": 0.453680157661438
        }
      ]
    },
    "Id": 1,
    "Type": "LINE"
  },
  {
    "Confidence": 90.54900360107422,
    "DetectedText": "IT'S",
    "Geometry": {
      "BoundingBox": {
        "Height": 0.10317354649305344,
        "Left": 0.6677391529083252,
        "Top": 0.17569075524806976,
        "Width": 0.15113449096679688
      },
      "Polygon": [
        {
          "X": 0.6677391529083252,
          "Y": 0.17569075524806976
        },
        {
          "X": 0.8188736438751221,
          "Y": 0.17574213445186615
        },
        {
          "X": 0.8188582062721252,
          "Y": 0.278915673494339
        },
        {
          "X": 0.6677237153053284,
          "Y": 0.2788642942905426
        }
      ]
    },
    "Id": 0,
    "Type": "LINE"
  }
]
"Confidence": 92.76634979248047,
"DetectedText": "MONDAY",
"Geometry": {
  "BoundingBox": {
    "Height": 0.11997425556182861,
    "Left": 0.5545867085456848,
    "Top": 0.349201411100883484,
    "Width": 0.39841532707214355
  },
  "Polygon": [
    {
      "X": 0.5545867085456848,
      "Y": 0.349201411100883484
    },
    {
      "X": 0.9530020356178284,
      "Y": 0.3471102714538574
    },
    {
      "X": 0.9532787799835205,
      "Y": 0.46708452701568604
    },
    {
      "X": 0.554863452911377,
      "Y": 0.46917566657066345
    }
  ],
  "Id": 2,
  "Type": "LINE"
},
{
  "Confidence": 96.76364898681646,
  "DetectedText": "but keep",
  "Geometry": {
    "BoundingBox": {
      "Height": 0.0756164938211441,
      "Left": 0.634815514087677,
      "Top": 0.5181083083152771,
      "Width": 0.20877975225448608
    },
    "Polygon": [
      {
        "X": 0.634815514087677,
        "Y": 0.5181083083152771
      },
      {
        "X": 0.8435952663421631,
        "Y": 0.52589350938797
      },
      {
        "X": 0.8423560857772827,
        "Y": 0.6015099883079529
      },
      {
        "X": 0.6335763335227966,
        "Y": 0.59372478723526
      }
    ],
    "Id": 3,
    "Type": "LINE"
  }
},
{
  "Confidence": 99.47185516357422,
  "DetectedText": "Smiling",
  "Geometry": {
    "BoundingBox": {
      "Height": 0.18999742556182861,
      "Left": 0.0545867085456848,
      "Top": 0.349201411100883484,
      "Width": 0.39841532707214355
    },
    "Polygon": [
      {
        "X": 0.5545867085456848,
        "Y": 0.349201411100883484
      },
      {
        "X": 0.9530020356178284,
        "Y": 0.3471102714538574
      },
      }
"Geometry": {
  "BoundingBox": {
    "Height": 0.2814019024372101,
    "Left": 0.48475268483161926,
    "Top": 0.6823741793632507,
    "Width": 0.47539761662483215
  },
  "Polygon": [
    {
      "X": 0.48475268483161926,
      "Y": 0.6823741793632507
    },
    {
      "X": 0.9601503014564514,
      "Y": 0.587857186794281
    },
    {
      "X": 0.9847385287284851,
      "Y": 0.8692590594291687
    },
    {
      "X": 0.5093409419059753,
      "Y": 0.9637760519981384
    }
  ],
  "Id": 4,
  "Type": "LINE"
},

{ "Confidence": 90.54900360107422,
  "DetectedText": "IT'S",
  "Geometry": {
    "BoundingBox": {
      "Height": 0.10387301445007324,
      "Left": 0.6685508489608765,
      "Top": 0.17597118020057678,
      "Width": 0.14985692501068115
    },
    "Polygon": [
      {
        "X": 0.6677391529083252,
        "Y": 0.17569075524806976
      },
      {
        "X": 0.8188736438751221,
        "Y": 0.17574213445186615
      },
      {
        "X": 0.8188582062721252,
        "Y": 0.278915673494339
      },
      {
        "X": 0.6677237153053284,
        "Y": 0.2788642942905426
      }
    ],
    "Id": 5,
    "ParentId": 0,
    "Type": "WORD"
},

{ "Confidence": 92.76634979248047,
  "DetectedText": "MONDAY",
  "Geometry": {

"BoundingBox": {  
  "Height": 0.11929994821548462,  
  "Left": 0.5540683269500732,  
  "Top": 0.34858056902885437,  
  "Width": 0.3998897075653076  
},  
"Polygon": [  
  {  
    "X": 0.5545867085456848,  
    "Y": 0.34920141100883484  
  },  
  {  
    "X": 0.9530020356178284,  
    "Y": 0.3471102714538574  
  },  
  {  
    "X": 0.9532787799835205,  
    "Y": 0.46708452701568604  
  },  
  {  
    "X": 0.554863452911377,  
    "Y": 0.46917566657066345  
  }  
],  
"Id": 7,  
"ParentId": 2,  
"Type": "WORD"  
},  
{  
  "Confidence": 59.411651611328125,  
  "DetectedText": "I",  
  "Geometry": {  
    "BoundingBox": {  
      "Height": 0.05981886386871338,  
      "Left": 0.2779299318790436,  
      "Top": 0.3935416042804718,  
      "Width": 0.0262411236763005  
    },  
    "Polygon": [  
      {  
        "X": 0.2763049304485321,  
        "Y": 0.394121915102005  
      },  
      {  
        "X": 0.30298948287963867,  
        "Y": 0.3932435214519501  
      },  
      {  
        "X": 0.30385109782218933,  
        "Y": 0.45280176401138306  
      },  
      {  
        "X": 0.27716654539108276,  
        "Y": 0.453680157661438  
      }  
    ]  
  },  
  "Id": 6,  
  "ParentId": 1,  
  "Type": "WORD"  
},  
{  
  "Confidence": 95.33189392089844,  
  "DetectedText": "but",  
  "Geometry": {
"BoundingBox": {  
  "Height": 0.06849122047424316,  
  "Left": 0.6350157260894775,  
  "Top": 0.5214487314224243,  
  "Width": 0.08413040637969971  
},  
"Polygon": [  
  {  
    "X": 0.6347596645355225,  
    "Y": 0.5215170383453369  
  },  
  {  
    "X": 0.719483494758606,  
    "Y": 0.5212655067443848  
  },  
  {  
    "X": 0.7195737957954407,  
    "Y": 0.5904868841171265  
  },  
  {  
    "X": 0.6348499655723572,  
    "Y": 0.5907384157180786  
  }  
],  
"Id": 8,  
"ParentId": 3,  
"Type": "WORD"  
},  
{  
  "Confidence": 98.1954116821289,  
  "DetectedText": "keep",  
  "Geometry": {  
    "BoundingBox": {  
      "Height": 0.07207882404327393,  
      "Left": 0.7295929789543152,  
      "Top": 0.5265749096870422,  
      "Width": 0.1119604110717734  
    },  
    "Polygon": [  
      {  
        "X": 0.7290706038475037,  
        "Y": 0.5251666903495789  
      },  
      {  
        "X": 0.842876672744751,  
        "Y": 0.5268880724906921  
      },  
      {  
        "X": 0.8423973917961121,  
        "Y": 0.5989091886711121  
      },  
      {  
        "X": 0.7285913228988647,  
        "Y": 0.5972678065299988  
      }  
    ]  
  },  
  "Id": 9,  
  "ParentId": 3,  
  "Type": "WORD"  
},  
{  
  "Confidence": 99.47185516357422,  
  "DetectedText": "Smiling",  
  "Geometry": {  

Detecting Text in an Image

You can provide an input image as an image byte array (base64-encoded image bytes) or as an Amazon S3 object. In this procedure, you upload a .jpeg or .png image to your S3 bucket and specify the file name.

Prerequisites

To detect text in images:

- You need the AWS Command Line Interface (AWS CLI) or AWS SDK for Java. For more information, see Getting Started with Amazon Rekognition (p. 11).
- 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. 33).

To detect text in an image (AWS CLI)

1. Upload an image that contains text to your S3 bucket.

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

2. At the command line, run the following command. Replace bucketname and input.jpg with the names of the S3 bucket and image from Step 1. Replace region with the name of the region

```
```

```
you are using. For information about Amazon Rekognition regions and endpoints, see Regions and Endpoints.

```
aws rekognition detect-text \
--image "S3Object={Bucket=bucketname,Name=input.jpg}" \
--region region
```

3. Choose Enter. You'll see the JSON output for the DetectText operation.

**To detect text in an image (API)**

1. Upload an image that contains text to your S3 bucket.

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

2. In the AWS SDK for Java, run the following example code. Replace `bucketname` and `input.jpg` with the names of the S3 bucket and image that you used in Step 1. Replace `Regions.US_EAST_1` with the name of the region you are using.

   ```java
   package com.amazonaws.samples;
   
   import java.util.List;
   import com.amazonaws.AmazonClientException;
   import com.amazonaws.auth.AWSStaticCredentialsProvider;
   import com.amazonaws.regions.Regions;
   import com.amazonaws.services.rekognition.AmazonRekognition;
   import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
   import com.amazonaws.services.rekognition.model.DetectTextRequest;
   import com.amazonaws.services.rekognition.model.DetectTextResult;
   import com.amazonaws.services.rekognition.model.TextDetection;
   
   public class TextDetectionInImage {
       public static void main(String[] args) throws Exception {
           String photo = "input.jpg";
           String bucket = "bucketname";

           AWSCredentials credentials;
           try {
               credentials = new AWSStaticCredentialsProvider("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()
DetectTextRequest request = new DetectTextRequest()
    .withImage(new Image()
        .withS3Object(new S3Object()
            .withName(photo)
            .withBucket(bucket)));

try {
    DetectTextResult result = rekognitionClient.detectText(request);
    List<TextDetection> textDetections = result.getTextDetections();
    System.out.println("Detected lines and words for ", photo);
    for (TextDetection text: textDetections) {
        System.out.println("Detected: ", text.getDetectedText());
        System.out.println("Confidence: ", text.getConfidence().toString());
        System.out.println("Id : ", text.getId());
        System.out.println("Parent Id: ", text.getParentId());
        System.out.println("Type: ", text.getType());
        System.out.println();
    }
} catch(AmazonRekognitionException e) {
    e.printStackTrace();
}
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 (p. 165)
• CloudWatch Metrics for Rekognition (p. 168)

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. 23).

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.

<table>
<thead>
<tr>
<th>How Do I?</th>
<th>Relevant Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>How do I track the numbers of faces recognized?</td>
<td>Monitor the Sum statistic of the DetectedFaceCount metric.</td>
</tr>
</tbody>
</table>
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)

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:

  ```bash
  
  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: 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.

```bash
```

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. 23).

### CloudWatch Metrics for Rekognition

The following table summarizes the Rekognition metrics.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuccessfulRequestCount</td>
<td>The number of successful requests. The response code range for a successful request is 200 to 299.</td>
</tr>
<tr>
<td></td>
<td>Unit: Count</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Sum, Average</td>
</tr>
<tr>
<td>ThrottledCount</td>
<td>The number of throttled requests. Rekognition throttles a request when it receives more requests than the limit of transactions per second for your account. If the limit set for your account is frequently exceeded, you can request a limit increase. To request an increase, see <em>AWS Service Limits</em>.</td>
</tr>
<tr>
<td></td>
<td>Unit: Count</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Sum, Average</td>
</tr>
<tr>
<td>ResponseTime</td>
<td>The time in milliseconds for Rekognition to compute the response.</td>
</tr>
<tr>
<td></td>
<td>Units:</td>
</tr>
<tr>
<td></td>
<td>1. Count for <em>Data Samples</em> statistics</td>
</tr>
<tr>
<td></td>
<td>2. Milliseconds for <em>Average</em> statistics</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: <em>Data Samples</em>, <em>Average</em></td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>The <em>ResponseTime</em> metric is not included in the Rekognition metric pane.</td>
</tr>
<tr>
<td>DetectedFaceCount</td>
<td>The number of faces detected with the <em>IndexFaces</em> or <em>DetectFaces</em> operation.</td>
</tr>
<tr>
<td></td>
<td>Unit: Count</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Sum, Average</td>
</tr>
<tr>
<td>DetectedLabelCount</td>
<td>The number of labels detected with the <em>DetectLabels</em> operation.</td>
</tr>
<tr>
<td></td>
<td>Unit: Count</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Sum, Average</td>
</tr>
<tr>
<td>ServerErrorCount</td>
<td>The number of server errors. The response code range for a server error is 500 to 599.</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UserErrorCount</td>
<td>The number of user errors (invalid parameters, invalid image, no permission, etc). The response code range for a user error is 400 to 499.</td>
</tr>
</tbody>
</table>

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.
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:

<table>
<thead>
<tr>
<th>Header</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-Type:</td>
<td>application/x-amz-json-1.1</td>
<td>Specifies that the request content is JSON. Also specifies the JSON version.</td>
</tr>
<tr>
<td>X-Amz-Date:</td>
<td>&lt;Date&gt;</td>
<td>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.</td>
</tr>
<tr>
<td>X-Amz-Target:</td>
<td>RekognitionService.&lt;operation&gt;</td>
<td>The target Amazon Rekognition operation. For example, use RekognitionService.ListCollections to call the ListCollections operation.</td>
</tr>
</tbody>
</table>

Topics

- Actions (p. 170)
- Data Types (p. 313)

Actions

The following actions are supported:

- CompareFaces (p. 172)
- CreateCollection (p. 180)
- CreateStreamProcessor (p. 183)
- DeleteCollection (p. 186)
- DeleteFaces (p. 189)
- DeleteStreamProcessor (p. 192)
• DescribeStreamProcessor (p. 194)
• DetectFaces (p. 198)
• DetectLabels (p. 205)
• DetectModerationLabels (p. 212)
• DetectText (p. 216)
• GetCelebrityInfo (p. 219)
• GetCelebrityRecognition (p. 222)
• GetContentModeration (p. 227)
• GetFaceDetection (p. 231)
• GetFaceSearch (p. 236)
• GetLabelDetection (p. 241)
• GetPersonTracking (p. 245)
• IndexFaces (p. 250)
• ListCollections (p. 258)
• ListFaces (p. 262)
• ListStreamProcessors (p. 268)
• RecognizeCelebrities (p. 271)
• SearchFaces (p. 277)
• SearchFacesByImage (p. 282)
• StartCelebrityRecognition (p. 288)
• StartContentModeration (p. 291)
• StartFaceDetection (p. 295)
• StartFaceSearch (p. 299)
• StartLabelDetection (p. 303)
• StartPersonTracking (p. 307)
• StartStreamProcessor (p. 310)
• StopStreamProcessor (p. 312)
CompareFaces

Compares a face in the source input image with each of the 100 largest faces 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 Comparing Faces in Images (SDK) (p. 97).

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

**Request Syntax**

```json
{
  "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. 172)**

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. 172)**

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. 341) object

Required: Yes

**TargetImage (p. 172)**

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. 341) object

Required: Yes

**Response Syntax**

```json
{
    "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,
 "Taw": 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. 173)

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. 325) objects

SourceImageFace (p. 173)

The face in the source image that was used for comparison.
**SourceImageOrientationCorrection (p. 173)**

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. 173)**

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. 173)**

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

Type: Array of `ComparedFace` (p. 323) 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. 369).

HTTP Status Code: 400

**InternalServerException**

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

**InvalidFormatException**

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

```json
{
  "TargetImage":{
    "S3Object":{
      "Bucket":"example-photos",
      "Name":"family.jpg"
    }
  },
  "SourceImage":{
    "S3Object":{
```
Sample Response

```json
{
    "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": 0.1098901079072952,
        "Height": 0.278298944234048
    }
}]
}```
"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": 0.4856561423480225,
    "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. 250) 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.

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. 180)

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",
   "FaceModelVersion": "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. 180)

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

Type: String
**FaceModelVersion (p. 180)**

Version number of the face detection model associated with the collection you are creating.

Type: String

**StatusCode (p. 180)**

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

**InternalServerException**

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

HTTP Status Code: 500

**InvalidParameterValue**

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=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

{"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
{
  "FaceModelVersion":"2.0",
  "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
CreateStreamProcessor

Creates an Amazon Rekognition stream processor that you can use to detect and recognize faces in a streaming video.

Rekognition Video is a consumer of live video from Amazon Kinesis Video Streams. Rekognition Video sends analysis results to Amazon Kinesis Data Streams.

You provide as input a Kinesis video stream (Input) and a Kinesis data stream (Output) stream. You also specify the face recognition criteria in Settings. For example, the collection containing faces that you want to recognize. Use Name to assign an identifier for the stream processor. You use Name to manage the stream processor. For example, you can start processing the source video by calling StartStreamProcessor (p. 310) with the Name field.

After you have finished analyzing a streaming video, use StopStreamProcessor (p. 312) to stop processing. You can delete the stream processor by calling DeleteStreamProcessor (p. 192).

Request Syntax

```
{
    "Input": {
        "KinesisVideoStream": {
            "Arn": "string"
        }
    },
    "Name": "string",
    "Output": {
        "KinesisDataStream": {
            "Arn": "string"
        }
    },
    "RoleArn": "string",
    "Settings": {
        "FaceSearch": {
            "CollectionId": "string",
            "FaceMatchThreshold": number
        }
    }
}
```

Request Parameters

The request accepts the following data in JSON format.

**Input (p. 183)**

Kinesis video stream stream that provides the source streaming video. If you are using the AWS CLI, the parameter name is StreamProcessorInput.

Type: StreamProcessorInput (p. 360) object

Required: Yes

**Name (p. 183)**

An identifier you assign to the stream processor. You can use Name to manage the stream processor. For example, you can get the current status of the stream processor by calling DescribeStreamProcessor (p. 194). Name is idempotent.

Type: String

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

Required: Yes  

Output (p. 183)

Kinesis data stream stream to which Rekognition Video puts the analysis results. If you are using the AWS CLI, the parameter name is StreamProcessorOutput.

Type: StreamProcessorOutput (p. 361) object  

Required: Yes  

RoleArn (p. 183)

ARN of the IAM role that allows access to the stream processor.

Type: String  

Pattern: arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@-_/]+  

Required: Yes  

Settings (p. 183)

Face recognition input parameters to be used by the stream processor. Includes the collection to use for face recognition and the face attributes to detect.

Type: StreamProcessorSettings (p. 362) object  

Required: Yes  

Response Syntax

```json
{
    "StreamProcessorArn": "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.

StreamProcessorArn (p. 184)

ARN for the newly create stream processor.

Type: String  

Pattern: (^arn:[a-z\d-]+:rekognition:[a-z\d-]+:\d{12}:streamprocessor/\.+$)

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

`LimitExceededException`

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

`ResourceInUseException`

HTTP Status Code: 400

`ThrottlingException`

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

HTTP Status Code: 500

**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 Deleting a Collection (p. 106).

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

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

CollectionId (p. 186)

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

```json
{
   "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. 186)

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**

The collection specified in the request cannot be 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**

```plaintext
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
```
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. 189)**

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. 189)**

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. 189)**

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

InternalServerException

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

The collection specified in the request cannot be 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
```
DeleteFaces

```
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
DeleteStreamProcessor

Deletes the stream processor identified by Name. You assign the value for Name when you create the stream processor with CreateStreamProcessor (p. 183). You might not be able to use the same name for a stream processor for a few seconds after calling DeleteStreamProcessor.

Request Syntax

```json
{
    "Name": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

Name (p. 192)

The name of the stream processor you want to delete.

Type: String


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

Required: Yes

Response Elements

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

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

ProvisionedThroughputExceeded Exception

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

HTTP Status Code: 400
ResourceInUseException

HTTP Status Code: 400

ResourceNotFoundException

The collection specified in the request cannot be found.

HTTP Status Code: 400

ThrottlingException

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

HTTP Status Code: 500

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
DescribeStreamProcessor

Provides information about a stream processor created by CreateStreamProcessor (p. 183). You can get information about the input and output streams, the input parameters for the face recognition being performed, and the current status of the stream processor.

Request Syntax

```
{
  "Name": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

Name (p. 194)

Name of the stream processor for which you want information.

Type: String


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

Required: Yes

Response Syntax

```
{
  "CreationTimestamp": number,
  "Input": {
    "KinesisVideoStream": {
      "Arn": "string"
    }
  },
  "LastUpdateTimestamp": number,
  "Name": "string",
  "Output": {
    "KinesisDataStream": {
      "Arn": "string"
    }
  },
  "RoleArn": "string",
  "Settings": {
    "FaceSearch": {
      "CollectionId": "string",
      "FaceMatchThreshold": number
    }
  },
  "Status": "string",
  "StatusMessage": "string",
  "StreamProcessorArn": "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.

**CreationTimestamp (p. 194)**

Date and time the stream processor was created

Type: Timestamp

**Input (p. 194)**

Kinesis video stream that provides the source streaming video.

Type: StreamProcessorInput (p. 360) object

**LastUpdateTimestamp (p. 194)**

The time, in Unix format, the stream processor was last updated. For example, when the stream processor moves from a running state to a failed state, or when the user starts or stops the stream processor.

Type: Timestamp

**Name (p. 194)**

Name of the stream processor.

Type: String


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

**Output (p. 194)**

Kinesis data stream to which Rekognition Video puts the analysis results.

Type: StreamProcessorOutput (p. 361) object

**RoleArn (p. 194)**

ARN of the IAM role that allows access to the stream processor.

Type: String

Pattern: arn:aws:iam::\d{12}:role/?[a-zA-Z0-9+=,.@\-_]+

**Settings (p. 194)**

Face recognition input parameters that are being used by the stream processor. Includes the collection to use for face recognition and the face attributes to detect.

Type: StreamProcessorSettings (p. 362) object

**Status (p. 194)**

Current status of the stream processor.

Type: String

Valid Values: STOPPED | STARTING | RUNNING | FAILED | STOPPING

**StatusMessage (p. 194)**

Detailed status message about the stream processor.

Type: String
StreamProcessorArn (p. 194)

ARN of the stream processor.

Type: String

Pattern: (^arn:[a-z\d-]+:rekognition:[a-z\d-]+:\d{12}:streamprocessor\/.+$)

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

The collection specified in the request cannot be found.

HTTP Status Code: 400

ThrottlingException

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

HTTP Status Code: 500

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.

DetectFaces detects the 100 largest faces in the image. 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 Detecting Faces in an Image (SDK) (p. 95).

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. 198)

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. 198)**

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. 341) object

Required: Yes

**Response Syntax**

```json
{
   "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. 199)**

Details of each face found in the image.

Type: Array of FaceDetail (p. 332) objects

**OrientationCorrection (p. 199)**

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. 369).
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": 0.18, "Left": 0.56, "Top": 0.34, "Width": 0.24
        },
        "Confidence": 100, "Landmarks": [
          {
            "Type": "eyeLeft", "X": 0.64, "Y": 0.41
          },
          {
            "Type": "eyeRight", "X": 0.73, "Y": 0.41
          },
          {
            "Type": "nose", "X": 0.70, "Y": 0.44
          },
          {
            "Type": "mouthLeft", "X": 0.63, "Y": 0.47
          },
          {
            "Type": "mouthRight", "X": 0.72, "Y": 0.47
          }
        ]
      },
      "Pose": {
        "Pitch": 4.05, "Roll": 0.99, "Yaw": 13.69
      },
      "Quality": {
        "Brightness": 37.6, "Sharpness": 80.0
      }
    ]}
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 entities 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 Analysing Images Stored in an Amazon S3 Bucket (p. 38).

Note
DetectLabels does not support the detection of activities. However, activity detection is supported for label detection in videos. For more information, see StartLabelDetection (p. 303).

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. 198) 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

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

The request accepts the following data in JSON format.

Image (p. 205)

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. 341) object

Required: Yes

MaxLabels (p. 205)

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. 205)

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

```json
{
  "Labels": [
    {
      "Confidence": number,
      "Name": "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. 206)

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

Type: Array of Label (p. 345) objects

OrientationCorrection (p. 206)

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. 369).

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

ProvisionedThroughputExceedededException

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

```plaintext
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"
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 Detecting Unsafe Content (p. 148).

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

```json
{
  "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. 212)**

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. 341) object

Required: Yes

**MinConfidence (p. 212)**

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

```json
{
}
```
"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. 212)

Array of detected Moderation labels and the time, in milliseconds from the start of the video, they were detected.

Type: Array of ModerationLabel (p. 348) 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. 369).

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=XXXXXXXXXXXXXXXXXXXXXXXXX/20170424/us-west-2/rekognition/aws4_request,
SignedHeaders=content-type;host;x-amz-date;x-amz-target,
Signature=XXXXXXXXXXXXXXXXXXXXXXXXX
{
  "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": 79.0318786621094,
      "ParentName": "",
      "Name": "Explicit Nudity"
    },
    {
      "Confidence": 79.0318786621094,
      "ParentName": "Explicit Nudity",
      "Name": "Graphic Male Nudity"
    }
  ]
}
```
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
DetectText

Detects text in the input image and converts it into machine-readable text.

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 AWS CLI to call Amazon Rekognition operations, you must pass it as a reference to an image in an Amazon S3 bucket. For the AWS CLI, passing image bytes is not supported. The image must be either a .png or .jpeg formatted file.

The DetectText operation returns text in an array of TextDetection (p. 364) elements, TextDetections. Each TextDetection element provides information about a single word or line of text that was detected in the image.

A word is one or more ISO basic latin script characters that are not separated by spaces. DetectText can detect up to 50 words in an image.

A line is a string of equally spaced words. A line isn't necessarily a complete sentence. For example, a driver's license number is detected as a line. A line ends when there is no aligned text after it. Also, a line ends when there is a large gap between words, relative to the length of the words. This means, depending on the gap between words, Amazon Rekognition may detect multiple lines in text aligned in the same direction. Periods don't represent the end of a line. If a sentence spans multiple lines, the DetectText operation returns multiple lines.

To determine whether a TextDetection element is a line of text or a word, use the TextDetection object Type field.

To be detected, text must be within +/- 30 degrees orientation of the horizontal axis.

For more information, see Detecting Text (p. 155).

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. 216)**

The input image as base64-encoded bytes or an Amazon S3 object. If you use the AWS CLI to call Amazon Rekognition operations, you can't pass image bytes.

Type: Image (p. 341) object

Required: Yes

Response Syntax

```
{
```
"TextDetections": [  
  {  
    "Confidence": number,  
    "DetectedText": "string",  
    "Geometry": {  
      "BoundingBox": {  
        "Height": number,  
        "Left": number,  
        "Top": number,  
        "Width": number  
      },  
      "Polygon": [  
        {  
          "X": number,  
          "Y": number  
        }  
      ]  
    },  
    "Id": number,  
    "ParentId": number,  
    "Type": "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.

TextDetections (p. 216)

An array of text that was detected in the input image.

Type: Array of TextDetection (p. 364) 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. 369).

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

**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 Getting Information about a Celebrity (p. 146).

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

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

**Id (p. 219)**

The ID for the celebrity. You get the celebrity ID from a call to the RecognizeCelebrities (p. 271) operation, which recognizes celebrities in an image.

Type: String

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

Required: Yes

Response Syntax

```json
{
   "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. 219)**

The name of the celebrity.

Type: String

**Urls (p. 219)**

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
The collection specified in the request cannot be 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=XXXXXXXXXXXXXXXXX

{"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
GetCelebrityRecognition

Gets the celebrity recognition results for a Rekognition Video analysis started by StartCelebrityRecognition (p. 288).

Celebrity recognition in a video is an asynchronous operation. Analysis is started by a call to StartCelebrityRecognition (p. 288) which returns a job identifier (JobId). When the celebrity recognition operation finishes, Rekognition Video publishes a completion status to the Amazon Simple Notification Service topic registered in the initial call to StartCelebrityRecognition. To get the results of the celebrity recognition analysis, first check that the status value published to the Amazon SNS topic is SUCCEEDED. If so, call GetCelebrityDetection and pass the job identifier (JobId) from the initial call to StartCelebrityDetection. For more information, see Working with Stored Videos (p. 51).

GetCelebrityRecognition returns detected celebrities and the time(s) they are detected in an array (Celebrities) of CelebrityRecognition (p. 322) objects. Each CelebrityRecognition contains information about the celebrity in a CelebrityDetail (p. 320) object and the time, Timestamp, the celebrity was detected.

By default, the Celebrities array is sorted by time (milliseconds from the start of the video). You can also sort the array by celebrity by specifying the value ID in the SortBy input parameter.

The CelebrityDetail object includes the celebrity identifier and additional information urls. If you don't store the additional information urls, you can get them later by calling GetCelebrityInfo (p. 219) with the celebrity identifier.

No information is returned for faces not recognized as celebrities.

Use MaxResults parameter to limit the number of labels returned. If there are more results than specified in MaxResults, the value of NextToken in the operation response contains a pagination token for getting the next set of results. To get the next page of results, call GetCelebrityDetection and populate the NextToken request parameter with the token value returned from the previous call to GetCelebrityRecognition.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

**JobId (p. 222)**

Job identifier for the required celebrity recognition analysis. You can get the job identifier from a call to StartCelebrityRecognition.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: ^[a-zA-Z0-9-]+$
Required: Yes

**MaxResults (p. 222)**

Maximum number of celebrities you want Rekognition Video to return in the response. The default is 1000.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

**NextToken (p. 222)**

If the previous response was incomplete (because there is more recognized celebrities to retrieve), Rekognition Video returns a pagination token in the response. You can use this pagination token to retrieve the next set of celebrities.

Type: String

Length Constraints: Maximum length of 255.

Required: No

**SortBy (p. 222)**

Sort to use for celebrities returned in Celebrities field. Specify ID to sort by the celebrity identifier, specify TIMESTAMP to sort by the time the celebrity was recognized.

Type: String

Valid Values: ID | TIMESTAMP

Required: No

**Response Syntax**

```
{
   "Celebrities": [ 
       { 
           "Celebrity": { 
               "BoundingBox": { 
                   "Height": number, 
                   "Left": number, 
                   "Top": number, 
                   "Width": number 
               },
               "Confidence": number,
               "Face": { 
                   "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
  }
},
  "Id": "string",
  "Name": "string",
  "Urls": [ "string" ]
},
  "Timestamp": number
]
},
  "JobStatus": "string",
  "NextToken": "string",
  "StatusMessage": "string",
  "VideoMetadata": {
    "Codec": "string",
    "DurationMillis": 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.

**Celebrities (p. 223)**

Array of celebrities recognized in the video.

Type: Array of CelebrityRecognition (p. 322) objects

**JobStatus (p. 223)**

The current status of the celebrity recognition job.

Type: String

Valid Values: IN_PROGRESS | SUCCEEDED | FAILED

**NextToken (p. 223)**

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

Type: String

Length Constraints: Maximum length of 255.

**StatusMessage (p. 223)**

If the job fails, StatusMessage provides a descriptive error message.

Type: String

**VideoMetadata (p. 223)**

Information about a video that Rekognition Video analyzed. VideoMetadata is returned in every page of paginated responses from a Rekognition Video operation.

Type: VideoMetadata (p. 367) object

Errors

**AccessDeniedException**

You are not authorized to perform the action.

HTTP Status Code: 400

**InternalServerException**

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

The collection specified in the request cannot be found.

HTTP Status Code: 400

ThrottlingException

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

HTTP Status Code: 500

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
GetContentModeration

Gets the content moderation analysis results for a Rekognition Video analysis started by StartContentModeration (p. 291).

Content moderation analysis of a video is an asynchronous operation. You start analysis by calling StartContentModeration (p. 291), which returns a job identifier (JobId). When analysis finishes, Rekognition Video publishes a completion status to the Amazon Simple Notification Service topic registered in the initial call to StartContentModeration. To get the results of the content moderation analysis, first check that the status value published to the Amazon SNS topic is SUCCEEDED. If so, call GetCelebrityDetection and pass the job identifier (JobId) from the initial call to StartCelebrityDetection. For more information, see Working with Stored Videos (p. 51).

GetContentModeration returns detected content moderation labels, and the time they are detected, in an array, ModerationLabels, of ContentModerationDetection (p. 326) objects.

By default, the moderated labels are returned sorted by time, in milliseconds from the start of the video. You can also sort them by moderated label by specifying NAME for the SortBy input parameter.

Since video analysis can return a large number of results, use the MaxResults parameter to limit the number of labels returned in a single call to GetContentModeration. If there are more results than specified in MaxResults, the value of NextToken in the operation response contains a pagination token for getting the next set of results. To get the next page of results, call GetContentModeration and populate the NextToken request parameter with the value of NextToken returned from the previous call to GetContentModeration.

For more information, see Detecting Unsafe Content (p. 148).

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

**JobId (p. 227)**

The identifier for the content moderation job. Use JobId to identify the job in a subsequent call to GetContentModeration.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: ^[a-zA-Z0-9-_.]+$

Required: Yes

**MaxResults (p. 227)**

Maximum number of content moderation labels to return. The default is 1000.

Type: Integer
Valid Range: Minimum value of 1.

Required: No

NextToken (p. 227)

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 content moderation labels.

Type: String

Length Constraints: Maximum length of 255.

Required: No

SortBy (p. 227)

Sort to use for elements in the ModerationLabelDetections array. Use TIMESTAMP to sort array elements by the time labels are detected. Use NAME to alphabetically group elements for a label together. Within each label group, the array element are sorted by detection confidence. The default sort is by TIMESTAMP.

Type: String

Valid Values: NAME | TIMESTAMP

Required: No

Response Syntax

```json
{
   "JobStatus": "string",
   "ModerationLabels": [
      { "ModerationLabel": {
         "Confidence": number,
         "Name": "string",
         "ParentName": "string"
      }, "Timestamp": number
   ],
   "NextToken": "string",
   "StatusMessage": "string",
   "VideoMetadata": {
      "Codec": "string",
      "DurationMillis": number,
      "Format": "string",
      "FrameHeight": number,
      "FrameRate": number,
      "FrameWidth": 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.
**JobStatus (p. 228)**

The current status of the content moderation job.

Type: String

Valid Values: IN_PROGRESS | SUCCEEDED | FAILED

**ModerationLabels (p. 228)**

The detected moderation labels and the time(s) they were detected.

Type: Array of ContentModerationDetection (p. 326) objects

**NextToken (p. 228)**

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

Type: String

Length Constraints: Maximum length of 255.

**StatusMessage (p. 228)**

If the job fails, StatusMessage provides a descriptive error message.

Type: String

**VideoMetadata (p. 228)**

Information about a video that Amazon Rekognition analyzed. VideoMetadata is returned in every page of paginated responses from GetContentModeration.

Type: VideoMetadata (p. 367) object

**Errors**

**AccessDeniedException**

You are not authorized to perform the action.

HTTP Status Code: 400

**InternalServerException**

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

**ProvisionedThroughputExceeded**

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

**ResourceNotFoundException**

The collection specified in the request cannot be found.

HTTP Status Code: 400

**ThrottlingException**

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

HTTP Status Code: 500

**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
GetFaceDetection

Gets face detection results for a Rekognition Video analysis started by StartFaceDetection (p. 295).

Face detection with Rekognition Video is an asynchronous operation. You start face detection by calling StartFaceDetection (p. 295) which returns a job identifier (JobId). When the face detection operation finishes, Rekognition Video publishes a completion status to the Amazon Simple Notification Service topic registered in the initial call to StartFaceDetection. To get the results of the face detection operation, first check that the status value published to the Amazon SNS topic is SUCCEEDED. If so, call GetFaceDetection (p. 231) and pass the job identifier (JobId) from the initial call to StartFaceDetection.

GetFaceDetection returns an array of detected faces (Faces) sorted by the time the faces were detected.

Use MaxResults parameter to limit the number of labels returned. If there are more results than specified in MaxResults, the value of NextToken in the operation response contains a pagination token for getting the next set of results. To get the next page of results, call GetFaceDetection and populate the NextToken request parameter with the token value returned from the previous call to GetFaceDetection.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

**JobId (p. 231)**

Unique identifier for the face detection job. The JobId is returned from StartFaceDetection.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: ^[a-zA-Z0-9-_.]+$

Required: Yes

**MaxResults (p. 231)**

Maximum number of detected faces to return. The default is 1000.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

**NextToken (p. 231)**

If the previous response was incomplete (because there are more faces to retrieve), Rekognition Video 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

```json
{
  "Faces": [
    {
      "Face": {
        "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
        }
      }
    }
  
```
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. 232)

An array of faces detected in the video. Each element contains a detected face's details and the time, in milliseconds from the start of the video, the face was detected.

Type: Array of FaceDetection (p. 335) objects

JobStatus (p. 232)

The current status of the face detection job.

Type: String

Valid Values: IN_PROGRESS | SUCCEEDED | FAILED

NextToken (p. 232)

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

Length Constraints: Maximum length of 255.

StatusMessage (p. 232)

If the job fails, StatusMessage provides a descriptive error message.
Type: String

**VideoMetadata (p. 232)**

Information about a video that Rekognition Video analyzed. VideoMetadata is returned in every page of paginated responses from a Amazon Rekognition video operation.

Type: VideoMetadata (p. 367) object

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**

The collection specified in the request cannot be found.

HTTP Status Code: 400

**ThrottlingException**

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

HTTP Status Code: 500

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
GetFaceSearch

Gets the face search results for Rekognition Video face search started by StartFaceSearch (p. 299). The search returns faces in a collection that match the faces of persons detected in a video. It also includes the time(s) that faces are matched in the video.

Face search in a video is an asynchronous operation. You start face search by calling to StartFaceSearch (p. 299) which returns a job identifier (JobId). When the search operation finishes, Rekognition Video publishes a completion status to the Amazon Simple Notification Service topic registered in the initial call to StartFaceSearch. To get the search results, first check that the status value published to the Amazon SNS topic is SUCCEEDED. If so, call GetFaceSearch and pass the job identifier (JobId) from the initial call to StartFaceSearch. For more information, see Searching Faces in a Collection (p. 102).

The search results are returned in an array, Persons, of PersonMatch (p. 354) objects. Each PersonMatch element contains details about the matching faces in the input collection, person information for the matched person, and the time the person was matched in the video.

By default, the Persons array is sorted by the time, in milliseconds from the start of the video, persons are matched. You can also sort by persons by specifying INDEX for the SORTBY input parameter.

Request Syntax

```json
{
  "JobId": "string",
  "MaxResults": number,
  "NextToken": "string",
  "SortBy": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

**JobId (p. 236)**

The job identifier for the search request. You get the job identifier from an initial call to StartFaceSearch.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: ^[a-zA-Z0-9-\_]+$

Required: Yes

**MaxResults (p. 236)**

Maximum number of search results you want Rekognition Video to return in the response. The default is 1000.

Type: Integer

Valid Range: Minimum value of 1.

Required: No
NextToken (p. 236)

If the previous response was incomplete (because there is more search results to retrieve), Rekognition Video returns a pagination token in the response. You can use this pagination token to retrieve the next set of search results.

Type: String
Length Constraints: Maximum length of 255.
Required: No

SortBy (p. 236)

Sort to use for grouping faces in the response. Use TIMESTAMP to group faces by the time that they are recognized. Use INDEX to sort by recognized faces.

Type: String
Valid Values: INDEX | TIMESTAMP
Required: No

Response Syntax

```
{
  "JobStatus": "string",
  "NextToken": "string",
  "Persons": [
    {
      "FaceMatches": [
        {
          "Face": {
            "BoundingBox": {
              "Height": number,
              "Left": number,
              "Top": number,
              "Width": number
            },
            "Confidence": number,
            "ExternalImageId": "string",
            "FaceId": "string",
            "ImageId": "string"
          },
          "Similarity": number
        }
      ],
      "Person": {
        "BoundingBox": {
          "Height": number,
          "Left": number,
          "Top": number,
          "Width": number
        },
        "Face": {
          "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
}
},
"Index": number
},
"Timestamp": number
]
,"StatusMessage": "string",
"VideoMetadata": {
    "Codec": "string",
    "Duration": number
}
}
"DurationMillis": number,
"Format": "string",
"FrameHeight": number,
"FrameRate": number,
"FrameWidth": 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.

**JobStatus (p. 237)**

- The current status of the face search job.
  
  Type: String

  Valid Values: **IN_PROGRESS** | **SUCCEEDED** | **FAILED**

**NextToken (p. 237)**

- If the response is truncated, Rekognition Video returns this token that you can use in the subsequent request to retrieve the next set of search results.

  Type: String

  Length Constraints: Maximum length of 255.

**Persons (p. 237)**

- An array of persons, **PersonMatch (p. 354)**, in the video whose face(s) match the face(s) in an Amazon Rekognition collection. It also includes time information for when persons are matched in the video. You specify the input collection in an initial call to **StartFaceSearch**. Each **Persons** element includes a time the person was matched, face match details (**FaceMatches**) for matching faces in the collection, and person information (**Person**) for the matched person.

  Type: Array of **PersonMatch (p. 354)** objects

**StatusMessage (p. 237)**

- If the job fails, **StatusMessage** provides a descriptive error message.

  Type: String

**VideoMetadata (p. 237)**

- Information about a video that Amazon Rekognition analyzed. **VideoMetadata** is returned in every page of paginated responses from a Rekognition Video operation.

  Type: **VideoMetadata (p. 367)** object

Errors

**AccessDeniedException**

- You are not authorized to perform the action.

  HTTP Status Code: 400
InternalServerException

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

The collection specified in the request cannot be found.

HTTP Status Code: 400

ThrottlingException

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

HTTP Status Code: 500

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
GetLabelDetection

Gets the label detection results of a Rekognition Video analysis started by StartLabelDetection (p. 303).

The label detection operation is started by a call to StartLabelDetection (p. 303) which returns a job identifier (JobId). When the label detection operation finishes, Amazon Rekognition publishes a completion status to the Amazon Simple Notification Service topic registered in the initial call to StartLabelDetection. To get the results of the label detection operation, first check that the status value published to the Amazon SNS topic is SUCCEEDED. If so, call GetLabelDetection (p. 241) and pass the job identifier (JobId) from the initial call to StartLabelDetection.

GetLabelDetection returns an array of detected labels (Labels) sorted by the time the labels were detected. You can also sort by the label name by specifying NAME for the SortBy input parameter.

The labels returned include the label name, the percentage confidence in the accuracy of the detected label, and the time the label was detected in the video.

Use MaxResults parameter to limit the number of labels returned. If there are more results than specified in MaxResults, the value of NextToken in the operation response contains a pagination token for getting the next set of results. To get the next page of results, call GetLabelDetection and populate the NextToken request parameter with the token value returned from the previous call to GetLabelDetection.

Request Syntax

```json
{
  "JobId": "string",
  "MaxResults": number,
  "NextToken": "string",
  "SortBy": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

JobId (p. 241)

Job identifier for the label detection operation for which you want results returned. You get the job identifier from an initial call to StartLabelDetection.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: ^[a-zA-Z0-9-_.]+$

Required: Yes

MaxResults (p. 241)

Maximum number of labels you want Amazon Rekognition to return in the response. The default is 1000.

Type: Integer

Valid Range: Minimum value of 1.

Required: No
NextToken (p. 241)

If the previous response was incomplete (because there are more labels to retrieve), Rekognition Video returns a pagination token in the response. You can use this pagination token to retrieve the next set of labels.

Type: String

Length Constraints: Maximum length of 255.

Required: No

SortBy (p. 241)

Sort to use for elements in the Labels array. Use TIMESTAMP to sort array elements by the time labels are detected. Use NAME to alphabetically group elements for a label together. Within each label group, the array element are sorted by detection confidence. The default sort is by TIMESTAMP.

Type: String

Valid Values: NAME | TIMESTAMP

Required: No

Response Syntax

```json
{
    "JobStatus": "string",
    "Labels": [
        {
            "Label": {
                "Confidence": number,
                "Name": "string"
            },
            "Timestamp": number
        },
        "NextToken": "string",
        "StatusMessage": "string",
        "VideoMetadata": {
            "Codec": "string",
            "DurationMillis": number,
            "Format": "string",
            "FrameHeight": number,
            "FrameRate": number,
            "FrameWidth": 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.

JobStatus (p. 242)

The current status of the label detection job.

Type: String
Valid Values: IN_PROGRESS | SUCCEEDED | FAILED

**Labels (p. 242)**

An array of labels detected in the video. Each element contains the detected label and the time, in milliseconds from the start of the video, that the label was detected.

Type: Array of LabelDetection (p. 346) objects

**NextToken (p. 242)**

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

Type: String

Length Constraints: Maximum length of 255.

**StatusMessage (p. 242)**

If the job fails, StatusMessage provides a descriptive error message.

Type: String

**VideoMetadata (p. 242)**

Information about a video that Rekognition Video analyzed. VideoMetadata is returned in every page of paginated responses from a Amazon Rekognition video operation.

Type: VideoMetadata (p. 367) object

**Errors**

**AccessDeniedException**

You are not authorized to perform the action.

HTTP Status Code: 400

**InternalServerException**

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

The collection specified in the request cannot be found.

HTTP Status Code: 400

ThrottlingException

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

HTTP Status Code: 500

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
GetPersonTracking

Gets the person tracking results of a Rekognition Video analysis started by StartPersonTracking (p. 307).

The person detection operation is started by a call to StartPersonTracking which returns a job identifier (JobId). When the person detection operation finishes, Rekognition Video publishes a completion status to the Amazon Simple Notification Service topic registered in the initial call to StartPersonTracking.

To get the results of the person tracking operation, first check that the status value published to the Amazon SNS topic is SUCCEEDED. If so, call GetPersonTracking (p. 245) and pass the job identifier (JobId) from the initial call to StartPersonTracking.

GetPersonTracking returns an array, Persons, of tracked persons and the time(s) they were tracked in the video.

By default, the array is sorted by the time(s) a person is tracked in the video. You can sort by tracked persons by specifying INDEX for the SortBy input parameter.

Use the MaxResults parameter to limit the number of items returned. If there are more results than specified in MaxResults, the value of NextToken in the operation response contains a pagination token for getting the next set of results. To get the next page of results, call GetPersonTracking and populate the NextToken request parameter with the token value returned from the previous call to GetPersonTracking.

Request Syntax

```json
{
   "JobId": "string",
   "MaxResults": number,
   "NextToken": "string",
   "SortBy": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

JobId (p. 245)

The identifier for a job that tracks persons in a video. You get the JobId from a call to StartPersonTracking.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: ^[a-zA-Z0-9-_]+$

Required: Yes

MaxResults (p. 245)

Maximum number of tracked persons to return. The default is 1000.

Type: Integer

Valid Range: Minimum value of 1.
**NextToken (p. 245)**

If the previous response was incomplete (because there are more persons to retrieve), Rekognition Video returns a pagination token in the response. You can use this pagination token to retrieve the next set of persons.

Type: String

Length Constraints: Maximum length of 255.

**SortBy (p. 245)**

Sort to use for elements in the Persons array. Use TIMESTAMP to sort array elements by the time persons are detected. Use INDEX to sort by the tracked persons. If you sort by INDEX, the array elements for each person are sorted by detection confidence. The default sort is by TIMESTAMP.

Type: String

Valid Values: INDEX | TIMESTAMP

Required: No

**Response Syntax**

```json
{
    "JobStatus": "string",
    "NextToken": "string",
    "Persons": [
        {
            "Person": {
                "BoundingBox": {
                    "Height": number,
                    "Left": number,
                    "Top": number,
                    "Width": number
                },
                "Face": {
                    "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
                    }
                }
            }
        }
    ]
}
```
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.
**JobStatus (p. 246)**

The current status of the person tracking job.

Type: String

Valid Values: IN_PROGRESS | SUCCEEDED | FAILED

**NextToken (p. 246)**

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

Type: String

Length Constraints: Maximum length of 255.

**Persons (p. 246)**

An array of the persons detected in the video and the times they are tracked throughout the video. An array element will exist for each time the person is tracked.

Type: Array of PersonDetection (p. 353) objects

**StatusMessage (p. 246)**

If the job fails, StatusMessage provides a descriptive error message.

Type: String

**VideoMetadata (p. 246)**

Information about a video that Rekognition Video analyzed. VideoMetadata is returned in every page of paginated responses from a Rekognition Video operation.

Type: VideoMetadata (p. 367) object

## Errors

**AccessDeniedException**

You are not authorized to perform the action.

HTTP Status Code: 400

**InternalServerException**

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

The collection specified in the request cannot be found.

HTTP Status Code: 400

ThrottlingException

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

HTTP Status Code: 500

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. 277) and SearchFacesByImage (p. 282) operations.

If you are using version 1.0 of the face detection model, IndexFaces indexes the 15 largest faces in the input image. Later versions of the face detection model index the 100 largest faces in the input image. To determine which version of the model you are using, check the the value of FaceModelVersion in the response from IndexFaces. For more information, see Model Versioning (p. 9).

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. 262) 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.

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

Request Syntax

```json
{
    "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. 250)**

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. 250)

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. 250)

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. 250)

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. 341) object
Required: Yes

Response Syntax

```json
{
  "FaceModelVersion": "string",
  "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
}
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.

**FaceModelVersion (p. 251)**

Version number of the face detection model associated with the input collection (CollectionId).

Type: String

**FaceRecords (p. 251)**

An array of faces detected and added to the collection. For more information, see Storing Faces in a Face Collection (p. 107).

Type: Array of FaceRecord (p. 337) objects

**OrientationCorrection (p. 251)**

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. 369).

HTTP Status Code: 400

**InternalServerException**

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
The collection specified in the request cannot be found.

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

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

{
  "FaceModelVersion":"2.0",
  "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": 0.529397726058596, "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 Listing Collections (p. 104).

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. 258)**

Maximum number of collection IDs to return.

Type: Integer

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

Required: No

**NextToken (p. 258)**

Pagination token from the previous response.

Type: String

Length Constraints: Maximum length of 255.

Required: No

**Response Syntax**

```
{
  "CollectionIds": [ "string" ],
  "FaceModelVersions": [ "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. 258)**

An array of collection IDs.
Type: Array of strings

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

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

FaceModelVersions (p. 258)

Version numbers of the face detection models associated with the collections in the array CollectionIds. For example, the value of FaceModelVersions[2] is the version number for the face detection model used by the collection in CollectionId[2].

Type: Array of strings

NextToken (p. 258)

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

ProvisionedThroughputExceeded Exception

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

HTTP Status Code: 400

ResourceNotFoundException

The collection specified in the request cannot be 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

```plaintext
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

```plaintext
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"
    ],
    "FaceModelVersions": ["2.0", "2.0"
]}
```

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
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 Listing Faces in a Collection (p. 114).

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. 262)**

ID of the collection from which to list the faces.

- Type: String
- Pattern: \[a-zA-Z0-9_.-]+\]
- Required: Yes

**MaxResults (p. 262)**

Maximum number of faces to return.

- Type: Integer
- Valid Range: Minimum value of 0. Maximum value of 4096.
- Required: No

**NextToken (p. 262)**

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

```
{
```
"FaceModelVersion": "string",
"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.

**FaceModelVersion (p. 262)**

Version number of the face detection model associated with the input collection (CollectionId).

Type: String

**Faces (p. 262)**

An array of Face objects.

Type: Array of Face (p. 330) objects

**NextToken (p. 262)**

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

**InternalServerException**

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

The collection specified in the request cannot be 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-8bcb9b028b4
Content-Length: 3440
Connection: keep-alive

{
```
"FaceModelVersion":"2.0",
"Faces": [
    {
        "BoundingBox": {
            "Height": 0.14222200214862823,
            "Left": 0.06083089858293533,
            "Top": 0.247779984474182,
            "Width": 0.18991099298000336
        },
        "Confidence": 99.9989937379297,
        "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": 0.22106799483299225
        },
        "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": 0.2277800261974335,
            "Width": 0.14243300259113312
        },
        "Confidence": 99.9761962890625,
        "ExternalImageId": "externalimageidONE",
        "FaceId": "11111111-2222-3333-4444-555555555555",
        "ImageId": "11111111-2222-3333-4444-555555555555"
    },
    {
        "BoundingBox": {
            "Height": 0.1088899862766266,
            "Left": 0.8056380152702332,
            "Top": 0.2155599570274353,
            "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,
            "Width": 0.18991099298000336
        },
        "Confidence": 99.9989937379297,
        "ExternalImageId": "externalimageidONE",
        "FaceId": "11111111-2222-3333-4444-555555555555",
        "ImageId": "11111111-2222-3333-4444-555555555555"
    }
]
"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":-0.0688890266418457,
    "Width":0.27448099851608276
},
"Confidence":100.0,
"ExternalImageId":"externalimageidFIVE",
"FaceId":"11111111-2222-3333-4444-555555555555",
"ImageId":"
},

{ "BoundingBox":{
    "Height":0.1622219979763031,
    "Left":0.18942700326442719,
    "Top":0.00888890035450459,
    "Width":0.2129200684547424
},
"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":"
},

{ "BoundingBox":{
    "Height":0.04222220182418823,
    "Left":0.46696001291275024,
    "Top":0.894439888000488,
    "Width":0.05580030009150505
},
"Confidence":90.60900115966797,
"ExternalImageId":"externalimageidSIX",
"FaceId":"11111111-2222-3333-4444-555555555555",
"ImageId":"
},

{ "BoundingBox":{
    "Height":0.3222219944000244,
    "Left":0.24332299828529358,
    "Top":0.22333300113677797,
    "Width":0.43026700615882874
},
"Confidence":99.97339630126953,
"ExternalImageId":"externalimageidFIVE",
"FaceId":"11111111-2222-3333-4444-555555555555",
"ImageId":"
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
ListStreamProcessors

Gets a list of stream processors that you have created with CreateStreamProcessor (p. 183).

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

**MaxResults (p. 268)**

Maximum number of stream processors you want Rekognition Video to return in the response. The default is 1000.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

**NextToken (p. 268)**

If the previous response was incomplete (because there are more stream processors to retrieve), Rekognition Video returns a pagination token in the response. You can use this pagination token to retrieve the next set of stream processors.

Type: String

Length Constraints: Maximum length of 255.

Required: No

Response Syntax

```json
{
    "NextToken": "string",
    "StreamProcessors": [
        {
            "Name": "string",
            "Status": "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.
NextToken (p. 268)

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

Type: String

Length Constraints: Maximum length of 255.

StreamProcessors (p. 268)

List of stream processors that you have created.

Type: Array of StreamProcessor (p. 359) objects

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

ThrottlingException

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

HTTP Status Code: 500

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. 132).

RecognizeCelebrities returns the 100 largest faces in the image. It lists recognized celebrities in the CelebrityFaces array and unrecognized faces in the UnrecognizedFaces array. RecognizeCelebrities doesn't return celebrities whose faces are not amongst the largest 100 faces in the image.

For each celebrity recognized, the RecognizeCelebrities 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. 219) 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.

For an example, see Recognizing Celebrities in an Image (p. 136).

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

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

Image (p. 271)

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. 341) object

Required: Yes

Response Syntax

```json
{
  "CelebrityFaces": [ 
    
  ]
}
```
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. 271)**

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. 319) objects

**OrientationCorrection (p. 271)**

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. 271)**

Details about each unrecognized face in the image.

Type: Array of ComparedFace (p. 323) 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. 369).

HTTP Status Code: 400

**InternalServerException**

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=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
{
  "Image": {
    "S3Object": {
      "Bucket": "photo-collection",
      "Name": "image.jpg"
    }
  }
}
```

Sample Response

```
"CelebrityFaces": [{
  "Face": {
    "BoundingBox": {
      "Height": 0.617123007774353,
      "Left": 0.15641026198863983,
      "Top": 0.10864841192960739,
      "Width": 0.364102572026825
    },
    "Confidence": 99.99589538574219,
    "Landmarks": {
      "Type": "eyeLeft",
      "X": 0.2837241291999817,
      "Y": 0.3637104034423828
    },
    "Type": "eyeRight",
    "X": 0.4091649055480957,
    "Y": 0.3737893104553227
  },
  "Type": "nose",
  "X": 0.35267341136932373,
  "Y": 0.49657556414604187
},
  "Type": "mouthLeft",
  "X": 0.278635382652827,
  "Y": 0.5455248355865479
},
  "Type": "mouthRight",
  "X": 0.39566439390182495,
  "Y": 0.5597742199897766
},
  "Pose": {
    "Pitch": -7.749263763427734,
    "Roll": 2.0045521259930786,
    "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
}
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. 250)` 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 Searching for a Face Using its Face ID (p. 117).

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

**Request Syntax**

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

**Request Parameters**

The request accepts the following data in JSON format.

**CollectionId (p. 277)**

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. 277)**

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. 277)**

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.
SearchFaces

Required: No

MaxFaces (p. 277)

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
        }
    ],
    "FaceModelVersion": "string",
    "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. 278)

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

Type: Array of FaceMatch (p. 336) objects

FaceModelVersion (p. 278)

Version number of the face detection model associated with the input collection (CollectionId).

Type: String

SearchedFaceId (p. 278)

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

The collection specified in the request cannot be 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

```plaintext
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": 0.00888890035450459,
                    "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": 0.1290799753952026
                },
                "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
        }
    ],
    "FaceModelVersion": "2.0",
    "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. 250) operation, and then use the face IDs returned in subsequent calls to the SearchFaces (p. 277) 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 Searching for a Face Using an Image (p. 119).

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

**Request Syntax**

```json
{
  "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. 282)**

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. 282)**

(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%.
SearchFacesByImage

Type: Float

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

Required: No

Image (p. 282)

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. 341) object

Required: Yes

MaxFaces (p. 282)

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

```json
{
   "FaceMatches": [
      {
         "Face": {
            "BoundingBox": {
               "Height": number,
               "Left": number,
               "Top": number,
               "Width": number
            },
            "Confidence": number,
            "ExternalImageId": "string",
            "FaceId": "string",
            "ImageId": "string"
         },
         "Similarity": number
      }
   ],
   "FaceModelVersion": "string",
   "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. 283)**

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

Type: Array of **FaceMatch (p. 336)** objects

**FaceModelVersion (p. 283)**

Version number of the face detection model associated with the input collection (**CollectionId**).

Type: String

**SearchedFaceBoundingBox (p. 283)**

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

Type: **BoundingBox (p. 317)** object

**SearchedFaceConfidence (p. 283)**

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. 369)**.

HTTP Status Code: 400

**InternalServerException**

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

The collection specified in the request cannot be 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.jpg) 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": 0.2122900038957598,
      "Width": 0.1766670408236847
    },
    "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": 99.9990973144531,
    "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": 0.018888900056488136,
      "Width": 0.14985199276326
    },
    "Confidence": 99.99949645996094,
    "ExternalImageId": "noahjade",
    "FaceId": "11111111-2222-3333-4444-555555555555",
    "ImageId": "11111111-2222-3333-4444-555555555555"
  },
  "Similarity": 92.22525787353516
}
]
"FaceModelVersion": "2.0",
"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
StartCelebrityRecognition

Starts asynchronous recognition of celebrities in a stored video.

Rekognition Video can detect celebrities in a video must be stored in an Amazon S3 bucket. Use Video (p. 366) to specify the bucket name and the filename of the video. StartCelebrityRecognition returns a job identifier (JobId) which you use to get the results of the analysis. When celebrity recognition analysis is finished, Rekognition Video publishes a completion status to the Amazon Simple Notification Service topic that you specify in NotificationChannel. To get the results of the celebrity recognition analysis, first check that the status value published to the Amazon SNS topic is SUCCEEDED. If so, call GetCelebrityRecognition (p. 222) and pass the job identifier (JobId) from the initial call to StartCelebrityRecognition. For more information, see Recognizing Celebrities (p. 132).

Request Syntax

```
{
    "ClientRequestToken": "string",
    "JobTag": "string",
    "NotificationChannel": {
        "RoleArn": "string",
        "SNSTopicArn": "string"
    },
    "Video": {
        "S3Object": {
            "Bucket": "string",
            "Name": "string",
            "Version": "string"
        }
    }
}
```

Request Parameters

The request accepts the following data in JSON format.

**ClientRequestToken (p. 288)**

Idempotent token used to identify the start request. If you use the same token with multiple StartCelebrityRecognition requests, the same JobId is returned. Use ClientRequestToken to prevent the same job from being accidently started more than once.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: ^[a-zA-Z0-9-_.]+$

Required: No

**JobTag (p. 288)**

Unique identifier you specify to identify the job in the completion status published to the Amazon Simple Notification Service topic.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: [a-zA-Z0-9_.\-:]+
Required: No

**NotificationChannel (p. 288)**

The Amazon SNS topic ARN that you want Rekognition Video to publish the completion status of the celebrity recognition analysis to.

Type: NotificationChannel (p. 351) object

Required: No

**Video (p. 288)**

The video in which you want to recognize celebrities. The video must be stored in an Amazon S3 bucket.

Type: Video (p. 366) object

Required: Yes

**Response Syntax**

```json
{
   "JobId": "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.

**JobId (p. 289)**

The identifier for the celebrity recognition analysis job. Use JobId to identify the job in a subsequent call to GetCelebrityRecognition.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: ^[a-zA-Z0-9-_]+$

**Errors**

**AccessDeniedException**

You are not authorized to perform the action.

HTTP Status Code: 400

**IdempotentParameterMismatchException**

A ClientRequestToken input parameter was reused with an operation, but at least one of the other input parameters is different from the previous call to the operation.

HTTP Status Code: 400

**InternalServer>Error**

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
InvalidS3ObjectException
Amazon Rekognition is unable to access the S3 object specified in the request.

HTTP Status Code: 400
LimitExceededException

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
VideoTooLargeException
The file size or duration of the supplied media is too large. The maximum file size is 8GB. The maximum duration is 2 hours.

HTTP Status Code: 400

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
StartContentModeration

Starts asynchronous detection of explicit or suggestive adult content in a stored video.

Rekognition Video can moderate content in a video stored in an Amazon S3 bucket. Use Video (p. 366) to specify the bucket name and the filename of the video. StartContentModeration returns a job identifier (JobId) which you use to get the results of the analysis. When content moderation analysis is finished, Rekognition Video publishes a completion status to the Amazon Simple Notification Service topic that you specify in NotificationChannel.

To get the results of the content moderation analysis, first check that the status value published to the Amazon SNS topic is SUCCEEDED. If so, call GetContentModeration (p. 227) and pass the job identifier (JobId) from the initial call to StartContentModeration. For more information, see Detecting Unsafe Content (p. 148).

**Request Syntax**

```json
{
    "ClientRequestToken": "string",
    "JobTag": "string",
    "MinConfidence": number,
    "NotificationChannel": {
        "RoleArn": "string",
        "SNSTopicArn": "string"
    },
    "Video": {
        "S3Object": {
            "Bucket": "string",
            "Name": "string",
            "Version": "string"
        }
    }
}
```

**Request Parameters**

The request accepts the following data in JSON format.

**ClientRequestToken (p. 291)**

Idempotent token used to identify the start request. If you use the same token with multiple StartContentModeration requests, the same JobId is returned. Use ClientRequestToken to prevent the same job from being accidently started more than once.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: ^[a-zA-Z0-9-_.]+$

Required: No

**JobTag (p. 291)**

Unique identifier you specify to identify the job in the completion status published to the Amazon Simple Notification Service topic.

Type: String
Length Constraints: Minimum length of 1. Maximum length of 256.

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

Required: No

**MinConfidence (p. 291)**

Specifies the minimum confidence that Amazon Rekognition must have in order to return a moderated content label. Confidence represents how certain Amazon Rekognition is that the moderated content is correctly identified. 0 is the lowest confidence. 100 is the highest confidence. Amazon Rekognition doesn't return any moderated content labels with a confidence level lower than this specified value.

Type: Float

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

Required: No

**NotificationChannel (p. 291)**

The Amazon SNS topic ARN that you want Rekognition Video to publish the completion status of the content moderation analysis to.

Type: NotificationChannel (p. 351) object

Required: No

**Video (p. 291)**

The video in which you want to moderate content. The video must be stored in an Amazon S3 bucket.

Type: Video (p. 366) object

Required: Yes

**Response Syntax**

```
{
   "JobId": "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.

**JobId (p. 292)**

The identifier for the content moderation analysis job. Use JobId to identify the job in a subsequent call to GetContentModeration.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: ^[a-zA-Z0-9-\_]+$
Errors

AccessDeniedException

You are not authorized to perform the action.

HTTP Status Code: 400

IdempotentParameterMismatchException

A ClientRequestToken input parameter was reused with an operation, but at least one of the other input parameters is different from the previous call to the operation.

HTTP Status Code: 400

InternalServerException

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

InvalidS3ObjectException

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

HTTP Status Code: 400

LimitExceededException

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

VideoTooLargeException

The file size or duration of the supplied media is too large. The maximum file size is 8GB. The maximum duration is 2 hours.

HTTP Status Code: 400

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
StartFaceDetection

Starts asynchronous detection of faces in a stored video.

Rekognition Video can detect faces in a video stored in an Amazon S3 bucket. Use Video (p. 366) to specify the bucket name and the filename of the video. StartFaceDetection returns a job identifier (JobId) that you use to get the results of the operation. When face detection is finished, Rekognition Video publishes a completion status to the Amazon Simple Notification Service topic that you specify in NotificationChannel. To get the results of the label detection operation, first check that the status value published to the Amazon SNS topic is SUCCEEDED. If so, call GetFaceDetection (p. 231) and pass the job identifier (JobId) from the initial call to StartFaceDetection. For more information, see Detecting Faces in a Stored Video (p. 91).

Request Syntax

```
{
    "ClientRequestToken": "string",
    "FaceAttributes": "string",
    "JobTag": "string",
    "NotificationChannel": {
        "RoleArn": "string",
        "SNSTopicArn": "string"
    },
    "Video": {
        "S3Object": {
            "Bucket": "string",
            "Name": "string",
            "Version": "string"
        }
    }
}
```

Request Parameters

The request accepts the following data in JSON format.

**ClientRequestToken (p. 295)**

Idempotent token used to identify the start request. If you use the same token with multiple StartFaceDetection requests, the same JobId is returned. Use ClientRequestToken to prevent the same job from being accidently started more than once.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: ^[a-zA-Z0-9-_.]+$

Required: No

**FaceAttributes (p. 295)**

The face attributes you want returned.

**DEFAULT** - The following subset of facial attributes are returned: BoundingBox, Confidence, Pose, Quality and Landmarks.

**ALL** - All facial attributes are returned.

Type: String
Valid Values: DEFAULT | ALL
Required: No

**JobTag (p. 295)**
Unique identifier you specify to identify the job in the completion status published to the Amazon Simple Notification Service topic.

Type: String
Length Constraints: Minimum length of 1. Maximum length of 256.
Pattern: [a-zA-Z0-9_\-\.\:]+
Required: No

**NotificationChannel (p. 295)**
The ARN of the Amazon SNS topic to which you want Rekognition Video to publish the completion status of the face detection operation.

Type: NotificationChannel (p. 351) object
Required: No

**Video (p. 295)**
The video in which you want to detect faces. The video must be stored in an Amazon S3 bucket.

Type: Video (p. 366) object
Required: Yes

**Response Syntax**

```json
{
   "JobId": "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.

**JobId (p. 296)**
The identifier for the face detection job. Use JobId to identify the job in a subsequent call to GetFaceDetection.

Type: String
Length Constraints: Minimum length of 1. Maximum length of 64.
Pattern: ^[a-zA-Z0-9-\_]+$

**Errors**

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

IdempotentParameterMismatchException

A ClientRequestToken input parameter was reused with an operation, but at least one of the other input parameters is different from the previous call to the operation.

HTTP Status Code: 400

InternalServerException

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

InvalidS3ObjectException

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

HTTP Status Code: 400

LimitExceededException

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

VideoTooLargeException

The file size or duration of the supplied media is too large. The maximum file size is 8GB. The maximum duration is 2 hours.

HTTP Status Code: 400

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
StartFaceSearch

Starts the asynchronous search for faces in a collection that match the faces of persons detected in a stored video.

The video must be stored in an Amazon S3 bucket. Use Video (p. 366) to specify the bucket name and the filename of the video. StartFaceSearch returns a job identifier (JobId) which you use to get the search results once the search has completed. When searching is finished, Rekognition Video publishes a completion status to the Amazon Simple Notification Service topic that you specify in NotificationChannel. To get the search results, first check that the status value published to the Amazon SNS topic is SUCCEEDED. If so, call GetFaceSearch (p. 236) and pass the job identifier (JobId) from the initial call to StartFaceSearch. For more information, see Searching for Faces with Rekognition Video (p. 121).

Request Syntax

```json
{
    "ClientRequestToken": "string",
    "CollectionId": "string",
    "FaceMatchThreshold": number,
    "JobTag": "string",
    "NotificationChannel": {
        "RoleArn": "string",
        "SNSTopicArn": "string"
    },
    "Video": {
        "S3Object": {
            "Bucket": "string",
            "Name": "string",
            "Version": "string"
        }
    }
}
```

Request Parameters

The request accepts the following data in JSON format.

**ClientRequestToken (p. 299)**

Idempotent token used to identify the start request. If you use the same token with multiple StartFaceSearch requests, the same JobId is returned. Use ClientRequestToken to prevent the same job from being accidently started more than once.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: ^[a-zA-Z0-9-_]+$

Required: No

**CollectionId (p. 299)**

ID of the collection that contains the faces you want to search for.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.
Pattern: [a-zA-Z0-9_.\-]+  
Required: Yes

**FaceMatchThreshold (p. 299)**

The minimum confidence in the person 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

**JobTag (p. 299)**

Unique identifier you specify to identify the job in the completion status published to the Amazon Simple Notification Service topic.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

Pattern: [a-zA-Z0-9_.\-:]+  
Required: No

**NotificationChannel (p. 299)**

The ARN of the Amazon SNS topic to which you want Rekognition Video to publish the completion status of the search.

Type: NotificationChannel (p. 351) object  
Required: No

**Video (p. 299)**

The video you want to search. The video must be stored in an Amazon S3 bucket.

Type: Video (p. 366) object  
Required: Yes

**Response Syntax**

```
{
   "JobId": "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.

**JobId (p. 300)**

The identifier for the search job. Use JobId to identify the job in a subsequent call to GetFaceSearch.

Type: String
Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: ^[a-zA-Z0-9-_]+$  

Errors

AccessDeniedException
You are not authorized to perform the action.

HTTP Status Code: 400

IdempotentParameterMismatchException
A ClientRequestToken input parameter was reused with an operation, but at least one of the other input parameters is different from the previous call to the operation.

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

InvalidS3ObjectException
Amazon Rekognition is unable to access the S3 object specified in the request.

HTTP Status Code: 400

LimitExceededException

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
The collection specified in the request cannot be found.

HTTP Status Code: 400

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

HTTP Status Code: 500

VideoTooLargeException
The file size or duration of the supplied media is too large. The maximum file size is 8GB. The maximum duration is 2 hours.
HTTP Status Code: 400

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
StartLabelDetection

Starts asynchronous detection of labels in a stored video.

Rekognition Video can detect labels in a video. Labels are instances of real-world entities. This includes objects like flower, tree, and table; events like wedding, graduation, and birthday party; concepts like landscape, evening, and nature; and activities like a person getting out of a car or a person skiing.

The video must be stored in an Amazon S3 bucket. Use Video (p. 366) to specify the bucket name and the filename of the video. StartLabelDetection returns a job identifier (JobId) which you use to get the results of the operation. When label detection is finished, Rekognition Video publishes a completion status to the Amazon Simple Notification Service topic that you specify in NotificationChannel.

To get the results of the label detection operation, first check that the status value published to the Amazon SNS topic is SUCCEEDED. If so, call GetLabelDetection (p. 241) and pass the job identifier (JobId) from the initial call to StartLabelDetection.

Request Syntax

```json
{
  "ClientRequestToken": "string",
  "JobTag": "string",
  "MinConfidence": number,
  "NotificationChannel": {
    "RoleArn": "string",
    "SNSTopicArn": "string"
  },
  "Video": {
    "S3Object": {
      "Bucket": "string",
      "Name": "string",
      "Version": "string"
    }
  }
}
```

Request Parameters

The request accepts the following data in JSON format.

**ClientRequestToken (p. 303)**

Idempotent token used to identify the start request. If you use the same token with multiple StartLabelDetection requests, the same JobId is returned. Use ClientRequestToken to prevent the same job from being accidently started more than once.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

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

Required: No

**JobTag (p. 303)**

Unique identifier you specify to identify the job in the completion status published to the Amazon Simple Notification Service topic.
StartLabelDetection

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

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

Required: No

MinConfidence (p. 303)

Specifies the minimum confidence that Rekognition Video must have in order to return a detected label. Confidence represents how certain Amazon Rekognition is that a label is correctly identified. 0 is the lowest confidence. 100 is the highest confidence. Rekognition Video 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

NotificationChannel (p. 303)

The Amazon SNS topic ARN you want Rekognition Video to publish the completion status of the label detection operation to.

Type: NotificationChannel (p. 351) object

Required: No

Video (p. 303)

The video in which you want to detect labels. The video must be stored in an Amazon S3 bucket.

Type: Video (p. 366) object

Required: Yes

Response Syntax

```json
{
  "JobId": "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.

JobId (p. 304)

The identifier for the label detection job. Use JobId to identify the job in a subsequent call to GetLabelDetection.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.
Pattern: ^[a-zA-Z0-9-_.]+$  

**Errors**

**AccessDeniedException**

You are not authorized to perform the action.

HTTP Status Code: 400

**IdempotentParameterMismatchException**

A ClientRequestToken input parameter was reused with an operation, but at least one of the other input parameters is different from the previous call to the operation.

HTTP Status Code: 400

**InternalServer Error**

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

**InvalidS3ObjectException**

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

HTTP Status Code: 400

**LimitExceededException**

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

**VideoTooLargeException**

The file size or duration of the supplied media is too large. The maximum file size is 8GB. The maximum duration is 2 hours.

HTTP Status Code: 400

**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
StartPersonTracking

Starts the asynchronous tracking of persons in a stored video.

Rekognition Video can track persons in a video stored in an Amazon S3 bucket. Use Video (p. 366) to specify the bucket name and the filename of the video. StartPersonTracking returns a job identifier (JobId) which you use to get the results of the operation. When label detection is finished, Amazon Rekognition publishes a completion status to the Amazon Simple Notification Service topic that you specify in NotificationChannel.

To get the results of the person detection operation, first check that the status value published to the Amazon SNS topic is SUCCEEDED. If so, call GetPersonTracking (p. 245) and pass the job identifier (JobId) from the initial call to StartPersonTracking.

Request Syntax

```json
{
    "ClientRequestToken": "string",
    "JobTag": "string",
    "NotificationChannel": {
        "RoleArn": "string",
        "SNSTopicArn": "string"
    },
    "Video": {
        "S3Object": {
            "Bucket": "string",
            "Name": "string",
            "Version": "string"
        }
    }
}
```

Request Parameters

The request accepts the following data in JSON format.

**ClientRequestToken (p. 307)**

Idempotent token used to identify the start request. If you use the same token with multiple StartPersonTracking requests, the same JobId is returned. Use ClientRequestToken to prevent the same job from being accidently started more than once.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: ^[a-zA-Z0-9-_.]+$

Required: No

**JobTag (p. 307)**

Unique identifier you specify to identify the job in the completion status published to the Amazon Simple Notification Service topic.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 256.

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

Required: No

**NotificationChannel (p. 307)**

The Amazon SNS topic ARN you want Rekognition Video to publish the completion status of the people detection operation to.

Type: NotificationChannel (p. 351) object

Required: No

**Video (p. 307)**

The video in which you want to detect people. The video must be stored in an Amazon S3 bucket.

Type: Video (p. 366) object

Required: Yes

**Response Syntax**

```json
{
    "JobId": "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.

**JobId (p. 308)**

The identifier for the person detection job. Use JobId to identify the job in a subsequent call to GetPersonTracking.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: ^[a-zA-Z0-9-_]+$

**Errors**

**AccessDeniedException**

You are not authorized to perform the action.

HTTP Status Code: 400

**IdempotentParameterMismatchException**

A ClientRequestToken input parameter was reused with an operation, but at least one of the other input parameters is different from the previous call to the operation.

HTTP Status Code: 400

**InternalServerException**

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
InvalidS3ObjectException
Amazon Rekognition is unable to access the S3 object specified in the request.

HTTP Status Code: 400
LimitExceededException

HTTP Status Code: 400
ProvisionedThroughputExceedededException
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
VideoTooLargeException
The file size or duration of the supplied media is too large. The maximum file size is 8GB. The maximum duration is 2 hours.

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
StartStreamProcessor

Starts processing a stream processor. You create a stream processor by calling CreateStreamProcessor (p. 183). To tell StartStreamProcessor which stream processor to start, use the value of the Name field specified in the call to CreateStreamProcessor.

Request Syntax

```
{
   "Name": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

Name (p. 310)

The name of the stream processor to start processing.

- Type: String
- Pattern: [a-zA-Z0-9_.\-]+
- Required: Yes

Response Elements

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

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

ProvisionedThroughputExceededExeception

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

HTTP Status Code: 400
ResourceInUseException

HTTP Status Code: 400

ResourceNotFoundException

The collection specified in the request cannot be found.

HTTP Status Code: 400

ThrottlingException

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

HTTP Status Code: 500

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
StopStreamProcessor

Stops a running stream processor that was created by CreateStreamProcessor (p. 183).

Request Syntax

```
{
  "Name": "string"
}
```

Request Parameters

The request accepts the following data in JSON format.

Name (p. 312)

The name of a stream processor created by CreateStreamProcessor (p. 183).

Type: String


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

Required: Yes

Response Elements

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

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

ResourceInUseException
HTTP Status Code: 400

**ResourceNotFoundException**

The collection specified in the request cannot be found.

HTTP Status Code: 400

**ThrottlingException**

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

HTTP Status Code: 500

**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:

- AgeRange (p. 315)
- Beard (p. 316)
- BoundingBox (p. 317)
- Celebrity (p. 319)
- CelebrityDetail (p. 320)
- CelebrityRecognition (p. 322)
- ComparedFace (p. 323)
- ComparedSourceImageFace (p. 324)
- CompareFacesMatch (p. 325)
- ContentModerationDetection (p. 326)
- Emotion (p. 327)
- Eyeglasses (p. 328)
- EyeOpen (p. 329)
- Face (p. 330)
- FaceDetail (p. 332)
- FaceDetection (p. 335)
- FaceMatch (p. 336)
- FaceRecord (p. 337)
• FaceSearchSettings (p. 338)
• Gender (p. 339)
• Geometry (p. 340)
• Image (p. 341)
• ImageQuality (p. 342)
• KinesisDataStream (p. 343)
• KinesisVideoStream (p. 344)
• Label (p. 345)
• LabelDetection (p. 346)
• Landmark (p. 347)
• ModerationLabel (p. 348)
• MouthOpen (p. 349)
• Mustache (p. 350)
• NotificationChannel (p. 351)
• PersonDetail (p. 352)
• PersonDetection (p. 353)
• PersonMatch (p. 354)
• Point (p. 355)
• Pose (p. 356)
• S3Object (p. 357)
• Smile (p. 358)
• StreamProcessor (p. 359)
• StreamProcessorInput (p. 360)
• StreamProcessorOutput (p. 361)
• StreamProcessorSettings (p. 362)
• Sunglasses (p. 363)
• TextDetection (p. 364)
• Video (p. 366)
• VideoMetadata (p. 367)
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, face or text. 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. 271) operation.

Contents

Face

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

Type: ComparedFace (p. 323) 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
CelebrityDetail

Information about a recognized celebrity.

Contents

BoundingBox

Bounding box around the body of a celebrity.
Type: BoundingBox (p. 317) object
Required: No

Confidence

The confidence, in percentage, that Amazon Rekognition has that the recognized face is the celebrity.
Type: Float
Valid Range: Minimum value of 0. Maximum value of 100.
Required: No

Face

Face details for the recognized celebrity.
Type: FaceDetail (p. 332) object
Required: No

Id

The unique identifier for the celebrity.
Type: String
Pattern: [0-9A-Za-z]*
Required: No

Name

The name of the celebrity.
Type: String
Required: No

Urls

An array of URLs pointing to additional celebrity information.
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
CelebrityRecognition

Information about a detected celebrity and the time the celebrity was detected in a stored video. For more information, see GetCelebrityRecognition (p. 222).

Contents

Celebrity

Information about a recognized celebrity.

Type: CelebrityDetail (p. 320) object

Required: No

Timestamp

The time, in milliseconds from the start of the video, that the celebrity was recognized.

Type: Long

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. 317) 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. 347) objects

Required: No

Pose

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

Type: Pose (p. 356) object

Required: No

Quality

Identifies face image brightness and sharpness.

Type: ImageQuality (p. 342) 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. 317) 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. 323) 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
ContentModerationDetection

Information about a moderation label detection in a stored video.

Contents

ModerationLabel

The moderation label detected by in the stored video.

Type: ModerationLabel (p. 348) object

Required: No

Timestamp

Time, in milliseconds from the beginning of the video, that the moderation label was detected.

Type: Long

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. 317) 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. 315) object
Required: No

Beard
Indicates whether or not the face has a beard, and the confidence level in the determination.
Type: Beard (p. 316) object
Required: No

BoundingBox
Bounding box of the face.
Type: BoundingBox (p. 317) 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. 327) objects
Required: No

Eyeglasses
Indicates whether or not the face is wearing eye glasses, and the confidence level in the determination.
Type: Eyeglasses (p. 328) 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. 329) object
Gender

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

Type: Gender (p. 339) object

Required: No

Landmarks

Indicates the location of landmarks on the face.

Type: Array of Landmark (p. 347) 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. 349) object

Required: No

Mustache

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

Type: Mustache (p. 350) object

Required: No

Pose

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

Type: Pose (p. 356) object

Required: No

Quality

Identifies image brightness and sharpness.

Type: ImageQuality (p. 342) object

Required: No

Smile

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

Type: Smile (p. 358) object

Required: No

Sunglasses

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

Type: Sunglasses (p. 363) 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
FaceDetection

Information about a face detected in a video analysis request and the time the face was detected in the video.

Contents

Face

The face properties for the detected face.

Type: FaceDetail (p. 332) object

Required: No

Timestamp

Time, in milliseconds from the start of the video, that the face was detected.

Type: Long

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. 330) 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. 330) object

Required: No

FaceDetail

Structure containing attributes of the face that the algorithm detected.

Type: FaceDetail (p. 332) 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
FaceSearchSettings
Input face recognition parameters for an Amazon Rekognition stream processor. `FaceRecognitionSettings` is a request parameter for CreateStreamProcessor (p. 183).

Contents

CollectionId
The ID of a collection that contains faces that you want to search for.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 255.
Pattern: [a-zA-Z0-9_.-]+
Required: No

FaceMatchThreshold
Minimum face match confidence score that must be met to return a result for a recognized face. Default is 70. 0 is the lowest confidence. 100 is the highest confidence.
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
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
Geometry

Information about where text detected by DetectText (p. 216) is located on an image.

Contents

BoundingBox

An axis-aligned coarse representation of the detected text's location on the image.

Type: BoundingBox (p. 317) object

Required: No

Polygon

Within the bounding box, a fine-grained polygon around the detected text.

Type: Array of Point (p. 355) objects

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 Analysing an Image Loaded from a Local File System (p. 40).

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. 29).

Contents

Bytes

Blob of image bytes up to 5 MBs.

Type: Base64-encoded binary data object


Required: No

S3Object

Identifies an S3 object as the image source.

Type: `S3Object` (p. 357) 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
KinesisDataStream

The Kinesis data stream Amazon Rekognition to which the analysis results of a Amazon Rekognition stream processor are streamed. For more information, see CreateStreamProcessor (p. 183).

Contents

Arn

ARN of the output Amazon Kinesis Data Streams stream.

Type: String

Pattern: (^arn:([a-z\d-]+):kinesis([a-z\d-]+):\d{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
KinesisVideoStream

Kinesis video stream stream that provides the source streaming video for a Rekognition Video stream processor. For more information, see CreateStreamProcessor (p. 183).

Contents

Arn

ARN of the Kinesis video stream stream that streams the source video.

Type: String

Pattern: (^arn:([a-z\d-]+):kinesisvideo:([a-z\d-]+):\d{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
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
LabelDetection

Information about a label detected in a video analysis request and the time the label was detected in the video.

Contents

Label

Details about the detected label.

Type: Label (p. 345) object

Required: No

Timestamp

Time, in milliseconds from the start of the video, that the label was detected.

Type: Long

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 or video. Each type of moderated content has a label within a hierarchical taxonomy. For more information, see Detecting Unsafe Content (p. 148).

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
NotificationChannel

The Amazon Simple Notification Service topic to which Amazon Rekognition publishes the completion status of a video analysis operation. For more information, see Calling Rekognition Video Operations (p. 53).

Contents

RoleArn

The ARN of an IAM role that gives Amazon Rekognition publishing permissions to the Amazon SNS topic.

Type: String

Pattern: arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_]+

Required: Yes

SNSTopicArn

The Amazon SNS topic to which Amazon Rekognition to posts the completion status.

Type: String

Pattern: (^arn:aws:sns:.+:\w{12}:.+$)

Required: Yes

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
PersonDetail

Details about a person detected in a video analysis request.

Contents

BoundingBox

Bounding box around the detected person.

Type: BoundingBox (p. 317) object

Required: No

Face

Face details for the detected person.

Type: FaceDetail (p. 332) object

Required: No

Index

Identifier for the person detected person within a video. Use to keep track of the person throughout the video. The identifier is not stored by Amazon Rekognition.

Type: Long

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
PersonDetection

Details and tracking information for a single time a person is tracked in a video. Amazon Rekognition operations that track persons return an array of PersonDetection objects with elements for each time a person is tracked in a video. For more information, see GetPersonTracking (p. 245).

Contents

Person

Details about a person tracked in a video.

Type: PersonDetail (p. 352) object

Required: No

Timestamp

The time, in milliseconds from the start of the video, that the person was tracked.

Type: Long

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
PersonMatch

Information about a person whose face matches a face(s) in a Amazon Rekognition collection. Includes information about the faces in the Amazon Rekognition collection (FaceMatch (p. 336)), information about the person (PersonDetail (p. 352)) and the timestamp for when the person was detected in a video. An array of PersonMatch objects is returned by GetFaceSearch (p. 236).

Contents

FaceMatches

Information about the faces in the input collection that match the face of a person in the video.

Type: Array of FaceMatch (p. 336) objects

Required: No

Person

Information about the matched person.

Type: PersonDetail (p. 352) object

Required: No

Timestamp

The time, in milliseconds from the beginning of the video, that the person was matched in the video.

Type: Long

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
Point

The X and Y coordinates of a point on an image. The X and Y values returned are ratios of the overall image size. For example, if the input image is 700x200 and the operation returns X=0.5 and Y=0.25, then the point is at the (350,50) pixel coordinate on the image.

An array of `Point` objects, `Polygon`, is returned by `DetectText (p. 216)`. Polygon represents a fine-grained polygon around detected text. For more information, see `Geometry (p. 340)`.

Contents

X

The value of the X coordinate for a point on a `Polygon`.
Type: Float
Required: No

Y

The value of the Y coordinate for a point on a `Polygon`.
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
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


Required: No

Roll

Value representing the face rotation on the roll axis.

Type: Float


Required: No

Yaw

Value representing the face rotation on the yaw axis.

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
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. 29).

Contents

Bucket

Name of the S3 bucket.

Type: String


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

Required: No

Name

S3 object key name.

Type: String


Required: No

Version

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

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
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
StreamProcessor

An object that recognizes faces in a streaming video. An Amazon Rekognition stream processor is created by a call to CreateStreamProcessor (p. 183). The request parameters for CreateStreamProcessor describe the Kinesis video stream source for the streaming video, face recognition parameters, and where to stream the analysis results.

Contents

Name

Name of the Amazon Rekognition stream processor.

Type: String


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

Required: No

Status

Current status of the Amazon Rekognition stream processor.

Type: String

Valid Values: STOPPED | STARTING | RUNNING | FAILED | STOPPING

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
StreamProcessorInput

Information about the source streaming video.

Contents

KinesisVideoStream

The Kinesis video stream input stream for the source streaming video.

Type: KinesisVideoStream (p. 344) 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
StreamProcessorOutput

Information about the Amazon Kinesis Data Streams stream to which a Rekognition Video stream processor streams the results of a video analysis. For more information, see CreateStreamProcessor (p. 183).

Contents

KinesisDataStream

The Amazon Kinesis Data Streams stream to which the Amazon Rekognition stream processor streams the analysis results.

Type: KinesisDataStream (p. 343) 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
StreamProcessorSettings

Input parameters used to recognize faces in a streaming video analyzed by a Amazon Rekognition stream processor.

Contents

FaceSearch

Face search settings to use on a streaming video.

Type: FaceSearchSettings (p. 338) 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
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
TextDetection

Information about a word or line of text detected by DetectText (p. 216).

The DetectedText field contains the text that Amazon Rekognition detected in the image.

Every word and line has an identifier (Id). Each word belongs to a line and has a parent identifier (ParentId) that identifies the line of text in which the word appears. The word Id is also an index for the word within a line of words.

For more information, see Detecting Text (p. 155).

Contents

Confidence

The confidence that Amazon Rekognition has in the accuracy of the detected text and the accuracy of the geometry points around the detected text.

Type: Float

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

Required: No

DetectedText

The word or line of text recognized by Amazon Rekognition.

Type: String

Required: No

Geometry

The location of the detected text on the image. Includes an axis aligned coarse bounding box surrounding the text and a finer grain polygon for more accurate spatial information.

Type: Geometry (p. 340) object

Required: No

Id

The identifier for the detected text. The identifier is only unique for a single call to DetectText.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

ParentId

The Parent identifier for the detected text identified by the value of ID. If the type of detected text is LINE, the value of ParentId is Null.

Type: Integer

Valid Range: Minimum value of 0.

Required: No
Type
The type of text that was detected.

Type: String
Valid Values: LINE | WORD
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
Video

Video file stored in an Amazon S3 bucket. Amazon Rekognition video start operations such as StartLabelDetection (p. 303) use video to specify a video for analysis. The supported file formats are .mp4, .mov and .avi.

Contents

S3Object

The Amazon S3 bucket name and file name for the video.

Type: S3Object (p. 357) 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
**VideoMetadata**

Information about a video that Amazon Rekognition analyzed. VideoMetadata is returned in every page of paginated responses from an Amazon Rekognition video operation.

**Contents**

**Codec**
- Type of compression used in the analyzed video.
  - Type: String
  - Required: No

**DurationMillis**
- Length of the video in milliseconds.
  - Type: Long
  - Valid Range: Minimum value of 0.
  - Required: No

**Format**
- Format of the analyzed video. Possible values are MP4, MOV and AVI.
  - Type: String
  - Required: No

**FrameHeight**
- Vertical pixel dimension of the video.
  - Type: Long
  - Valid Range: Minimum value of 0.
  - Required: No

**FrameRate**
- Number of frames per second in the video.
  - Type: Float
  - Required: No

**FrameWidth**
- Horizontal pixel dimension of the video.
  - Type: Long
  - 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
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.
- To be detected, a face must be no smaller that 40x40 pixels in an image with 1920X1080 pixel resolution. Higher resolution images will need a larger minimum face size.
- 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 20 million.
- The maximum matching faces the search API returns is 4096.
- Rekognition Video can analyze stored videos up to 8GB in size.
## Document History for Amazon Rekognition

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

- **API version: 2017-11-29**
- **Latest documentation update:** November 29th, 2017

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyze stored and streaming videos. New table of contents.</td>
<td>For information about analyzing stored videos, see Working with Stored Videos (p. 51). For information about analyzing streaming videos, see Working with Streaming Videos (p. 68). The table of contents for the Amazon Rekognition documentation has been rearranged to accommodate image and video operations.</td>
<td>In this release</td>
</tr>
<tr>
<td>Text in Image and Face Detection Models</td>
<td>Amazon Rekognition can now detect text in images. For more information, see Detecting Text (p. 155). Amazon Rekognition introduces versioning for the face detection deep learning model. For more information, see Model Versioning (p. 9).</td>
<td>November 21st, 2017</td>
</tr>
<tr>
<td>Celebrity Recognition</td>
<td>Amazon Rekognition can now analyze images for celebrities. For more information, see Recognizing Celebrities (p. 132).</td>
<td>June 8th, 2017</td>
</tr>
<tr>
<td>Image Moderation</td>
<td>Amazon Rekognition can now determine if an image contains explicit or suggestive adult content. For more information, see Detecting Unsafe Content (p. 148).</td>
<td>April 19th, 2017</td>
</tr>
<tr>
<td>Age Range for Detected Faces Aggregated Rekognition Metrics Pane</td>
<td>Amazon Rekognition now returns the estimated age range, in years, for faces detected by the Rekognition API. For more information, see AgeRange (p. 315).</td>
<td>February 9th, 2017</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td></td>
<td>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. 23).</td>
<td>November 30, 2016</td>
</tr>
<tr>
<td>New service and guide</td>
<td>This is the initial release of the image analysis service, Amazon Rekognition, and the <em>Amazon Rekognition Developer Guide</em>.</td>
<td>November 30, 2016</td>
</tr>
</tbody>
</table>
AWS Glossary

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