Amazon Rekognition

Developer Guide

Amazon Rekognition: Developer Guide

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

What Is Amazon Rekognition? 1
- Amazon Rekognition and HIPAA Eligibility 2
- Are You a First-Time Amazon Rekognition User? 3

How It Works 4
- Types of Detection and Recognition 4
  - Labels 4
  - Faces 5
  - Face Search 5
  - People Paths 5
  - Celebrities 5
  - Text Detection 5
  - Unsafe Content 5
- Image and Video Operations 6
  - Amazon Rekognition Image Operations 6
  - Amazon 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 and AWS SDKs 12
  - Next Step 14
- Step 3: Getting Started Using the AWS CLI and AWS SDK API 14
  - Formatting the AWS CLI Examples 14
  - Next Step 14
- Step 4: Getting Started Using the Console 15
  - Exercise 1: Detect Objects and Scenes (Console) 15
  - Exercise 2: Analyze Faces (Console) 20
  - Exercise 3: Compare Faces (Console) 22
  - Exercise 4: See Aggregated Metrics (Console) 24

Authentication and Access Control 25
- Authentication 25
- Access Control 26
- Overview of Managing Access 26
  - Amazon Rekognition Resources and Operations 27
  - Understanding Resource Ownership 27
  - Managing Access to Resources 27
  - Specifying Policy Elements: Actions, Effects, and Principals 29
  - Specifying Conditions in a Policy 30
- Using Identity-Based Policies (IAM Policies) 30
  - Permissions Required to Use the Amazon Rekognition Console 31
  - AWS Managed (Predefined) Policies for Amazon Rekognition 31
  - Customer Managed Policy Examples 31
- Amazon Rekognition API Permissions Reference 32

Working with Images 36
- Images 36
  - Correcting Image Orientation 37
- Best Practices for Working With Images 37

Amazon Rekognition Image Operation Latency 37
Recommendations for Facial Recognition Input Images 37
Using an Amazon S3 Bucket 38
Using a Local File System 41
Using JavaScript 45
Getting Image Orientation and Bounding Box Coordinates 49
Finding an Image's Orientation 50
Displaying Bounding Boxes 50
Example: Getting Image Orientation and Bounding Box Coordinates For an Image 53
Working with Stored Videos 59
Types of Detection and Recognition 59
Amazon Rekognition Video API Overview 59
Video Formats and Storage 60
Searching for People 60
Calling Amazon Rekognition Video Operations 61
Starting Video Analysis 61
Getting the Completion Status of an Amazon Rekognition Video Analysis Request 62
Getting Amazon Rekognition Video Analysis Results 63
Configuring Amazon Rekognition Video 66
Giving Access to Multiple Amazon SNS Topics 67
Giving Access to an Existing Amazon SNS Topic 68
Analyzing a Stored Video (SDK) 68
Prerequisites 69
Analyzing a Video (AWS CLI) 75
Prerequisites 69
Tutorial: Creating an Amazon Rekognition Lambda Function 77
Prerequisites 78
Create the SNS Topic 78
Create the Lambda Function 78
Configure the Lambda Function 78
Configure the IAM Lambda Role 79
Create the AWS Toolkit for Eclipse Lambda Project 80
Test the Lambda Function 82
Reference: Video Analysis Results Notification 83
Troubleshooting Amazon Rekognition Video 84
I never receive the completion status that's sent to the Amazon SNS topic 84
Working with Streaming Videos 86
Recognizing Faces in a Streaming Video 86
Prerequisites 87
Giving Access to Kinesis Streams 87
Giving Access to Your Kinesis Video Streams and Kinesis Data Streams 87
Giving Access to Individual Kinesis Streams 88
Starting Streaming Video Analysis 88
Creating the Amazon Rekognition Video Stream Processor 90
Starting the Amazon Rekognition Video Stream Processor 91
Using Stream Processors 91
Streaming Video into Amazon Rekognition Video 95
Reading Analysis Results 95
Mapping the Kinesis Video Stream to the Kinesis Data Stream 97
Reference: Kinesis Face Recognition Record 97
JSON Record 99
InputInformation 99
KinesisVideo 99
StreamProcessorInformation 100
FaceSearchResponse 100
DetectedFace 100
MatchedFace 101

Detecting Objects and Scenes 102
Detecting Labels in an Image 103
DetectLabels Operation Request 107
DetectLabels Response 107
Detecting Labels in a Video 111
GetLabelDetection Operation Response 112
Detecting and Analyzing Faces 114
Detecting Faces in an Image 115
DetectFaces Operation Request 118
DetectFaces Operation Response 118
Comparing Faces in Images 124
CompareFaces Operation Request 128
CompareFaces Operation Response 129
Detecting Faces in a Stored Video 131
GetFaceDetection Operation Response 134
Searching Faces in a Collection 137
Managing Collections 137
Managing Faces in a Collection 137
Searching for Faces Within a Collection 138
Using Similarity Thresholds to Match Faces 138
Using Amazon Rekognition to Help Public Safety 139
Creating a Collection 139
CreateCollection Operation Request 141
CreateCollection Operation Response 142
Listing Collections 142
ListCollections Operation Request 144
ListCollections Operation Response 145
Describing a Collection 145
DescribeCollection Operation Request 148
DescribeCollection Operation Response 148
Deleting a Collection 148
DeleteCollection Operation Request 151
DeleteCollection Operation Response 151
Adding Faces to a Collection 151
Filtering Faces 152
IndexFaces Operation Request 156
IndexFaces Operation Response 156
Listing Faces in a Collection 162
ListFaces Operation Request 164
ListFaces Operation Response 165
Deleting Faces from a Collection 165
DeleteFaces Operation Request 168
DeleteFaces Operation Response 168
Searching for a Face (Face ID) 168
SearchFaces Operation Request 171
SearchFaces Operation Response 171
Searching for a Face (Image) 172
SearchFacesByImage Operation Request 175
SearchFacesByImage Operation Response 176
Searching Stored Videos for Faces 176
GetFaceSearch Operation Response 180
People Pathing 184
GetPersonTracking Operation Response 186
Recognizing Celebrities 190
Recognizing Celebrities in an Image 190
Calling RecognizeCelebrities 190
RecognizeCelebrities Operation Request 194

RecognizeCelebrities Operation Response 194
Recognizing Celebrities in a Stored Video 196
GetCelebrityRecognition Operation Response 199
Getting Celebrity Information 200
Calling GetCelebrityInfo 200
GetCelebrityInfo Operation Request 203
..... 203
Detecting Unsafe Content 204
Detecting Unsafe Images (API) 205
Detecting Unsafe Content in an Image (SDK) 205
..... 205
DetectModerationLabels Operation Request 208
DetectModerationLabels Operation Response 208
Detecting Unsafe Stored Videos 209
GetContentModeration Operation Response 211
Detecting Text 213
Detecting Text in an Image 213
DetectText Operation Request 217
DetectText Operation Response 217
Monitoring 224
Monitoring 224
Using CloudWatch Metrics for Rekognition 224
Access Rekognition Metrics 225
Create an Alarm 226
CloudWatch Metrics for Rekognition 227
CloudWatch Metrics for Rekognition 227
CloudWatch Dimension for Rekognition 228
Logging Amazon Rekognition API Calls with AWS CloudTrail 228
Amazon Rekognition Information in CloudTrail 228
Example: Amazon Rekognition Log File Entries 229
API Reference 234
HTTP Headers 234
Actions 234
CompareFaces 236
CreateCollection 241
CreateStreamProcessor 244
DeleteCollection 247
DeleteFaces 249
DeleteStreamProcessor 252
DescribeCollection 254
DescribeStreamProcessor 257
DetectFaces 261
DetectLabels 265
DetectModerationLabels 269
DetectText 272
GetCelebrityInfo 275
GetCelebrityRecognition 277
GetContentModeration 282
GetFaceDetection 286
GetFaceSearch 291
GetLabelDetection 296
GetPersonTracking 300
IndexFaces 305
ListCollections 313
ListFaces 316
ListStreamProcessors 319
RecognizeCelebrities 322

SearchFaces ... 326
SearchFacesByImage ... 329
StartCelebrityRecognition ... 333
StartContentModeration ... 337
StartFaceDetection ... 341
StartFaceSearch ... 345
StartLabelDetection ... 349
StartPersonTracking ... 353
StartStreamProcessor ... 356
StopStreamProcessor ... 358
Data Types ... 359
AgeRange ... 361
Beard ... 362
BoundingBox ... 363
Celebrity ... 365
CelebrityDetail ... 367
CelebrityRecognition ... 369
ComparedFace ... 370
ComparedSourceImageFace ... 371
CompareFacesMatch ... 372
ContentModerationDetection ... 373
Emotion ... 374
Eyeglasses ... 375
EyeOpen ... 376
Face ... 377
FaceDetail ... 379
FaceDetection ... 382
FaceMatch ... 383
FaceRecord ... 384
FaceSearchSettings ... 385
Gender ... 386
Geometry ... 387
Image ... 388
ImageQuality ... 389
Instance ... 390
KinesisDataStream ... 391
KinesisVideoStream ... 392
Label ... 393
LabelDetection ... 395
Landmark ... 396
ModerationLabel ... 397
MouthOpen ... 398
Mustache ... 399
NotificationChannel ... 400
Parent ... 401
PersonDetail ... 402
PersonDetection ... 403
PersonMatch ... 404
Point ... 405
Pose ... 406
S3Object ... 407
Smile ... 408
StreamProcessor ... 409
StreamProcessorInput ... 410
StreamProcessorOutput ... 411
StreamProcessorSettings ... 412
Sunglasses ... 413

TextDetection 414
UnindexedFace 416
Video 417
VideoMetadata 418
Limits 420
Amazon Rekognition Image 420
Amazon Rekognition Video Stored Video 420
Amazon Rekognition Video Streaming Video 420
Document History 421
AWS Glossary 423

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. For more information, see the Amazon Rekognition FAQs.

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 public safety 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.

Amazon Rekognition and HIPAA Eligibility

This is a HIPAA Eligible Service. For more information about AWS, U.S. Health Insurance Portability and Accountability Act of 1996 (HIPAA), and using AWS services to process, store, and transmit protected health information (PHI), see HIPAA Overview.

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. 36)** – 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. 59)** – This section provides information about using Amazon Rekognition with videos stored in an Amazon S3 bucket.
5. **Working with Streaming Videos (p. 86)** – This section provides information about using Amazon Rekognition with streaming videos.

Amazon Rekognition: How It Works

Amazon Rekognition provides two API sets. They are Amazon Rekognition Image, for analyzing images, and Amazon 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 Amazon Rekognition Image to enhance the customer experience for a photo management application. When a customer uploads a photo, your application can use Amazon Rekognition Image to detect real-world objects or faces in the image. After your application stores the information returned from Amazon 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.

You can use Amazon Rekognition Video to track the path of people in a stored video. Alternatively, you can use Amazon 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. 322) 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 Amazon Rekognition Image and Amazon 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. 6)
- 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 Amazon Rekognition Image API and Amazon Rekognition Video API can perform. For information about the APIs, see Image and Video Operations (p. 6).

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. 102).

To detect labels in images, use DetectLabels (p. 265). To detect labels in stored videos, use StartLabelDetection (p. 349).

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. 114).

To detect faces in images, use DetectFaces (p. 261). To detect faces in stored videos, use StartFaceDetection (p. 341).

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. 137).

To search for known faces in images, use DetectFaces (p. 261). To search for known faces in stored videos, use StartFaceDetection (p. 341). To search for known faces in streaming videos, use CreateStreamProcessor (p. 244).

People Paths

Amazon Rekognition can track the paths of people detected in a stored video. Amazon Rekognition Video provides path tracking, face details, and in-frame location information for people detected in a video. For more information, see People Pathing (p. 184).

To detect people in stored videos, use StartPersonTracking (p. 353).

Celebrities

Amazon Rekognition can recognize thousands 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 they appear throughout a stored video. You can also get further information about a recognized celebrity. For more information, see Recognizing Celebrities (p. 190).

To recognize celebrities in images, use RecognizeCelebrities (p. 322). To recognize celebrities in stored videos, use StartCelebrityRecognition (p. 333).

Text Detection

Amazon Rekognition Text in Image can detect text in images and convert it into machine-readable text. For more information, see Detecting Text (p. 213).

To detect text in images, use DetectText (p. 272).

Unsafe Content

Amazon Rekognition can analyze images and stored videos for explicit or suggestive adult content. For more information, see Detecting Unsafe Content (p. 204).

To detect unsafe images, use DetectModerationLabels (p. 269). To detect unsafe stored videos, use StartContentModeration (p. 337).

Image and Video Operations

Amazon Rekognition provides two API sets. They are Amazon Rekognition Image, for analyzing images, and Amazon Rekognition Video, for analyzing stored and streaming videos. The following topic gives a brief overview of each API set.

The Amazon Rekognition Image and Amazon 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 Types of Detection and Recognition (p. 4).

Amazon 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 an Amazon 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. 36).

Amazon Rekognition Video Operations

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

Amazon Rekognition Video video operations are asynchronous. With Amazon 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. 341). 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. 286). For more information, see Working with Stored Videos (p. 59).

With Amazon Rekognition Video streaming video operations you can search for faces stored in Amazon Rekognition Video collections. Amazon 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 an Amazon Rekognition Video stream processor. For example, you create a stream processor by calling CreateStreamProcessor (p. 244). For more information, see Working with Streaming Videos (p. 86).

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 Non-Storage Operations (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 .NET examples in this section uses the AWS SDK for .NET. You can use the AWS Toolkit for Visual Studio to develop AWS applications using .NET. It includes helpful templates and the AWS Explorer for deploying applications and managing services.

The API Reference (p. 234) in this guide covers calling Amazon Rekognition operations using HTTP. For Java reference information, see AWS SDK for Java.

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

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

Non-Storage and Storage API Operations

Amazon Rekognition provides two types of API operations. They are non-storage operations where no information is stored by Amazon Rekognition, and storage operations where certain facial information is stored by Amazon Rekognition.

Non-Storage Operations

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

- DetectLabels (p. 265)
- DetectFaces (p. 261)
- CompareFaces (p. 236)
- DetectModerationLabels (p. 269)
- RecognizeCelebrities (p. 322)
- DetectText (p. 272)

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

- StartLabelDetection (p. 349)
- StartFaceDetection (p. 341)
- StartPersonTracking (p. 353)
- StartCelebrityRecognition (p. 333)
- StartContentModeration (p. 337)

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

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

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

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

```
Label        ImageID
```

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

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

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

Example 2: An application to display enhanced face images

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

```
ImageID     FaceID     FaceMetaData

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

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

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

Storage-Based API Operations

Amazon Rekognition Image supports the IndexFaces (p. 305) operation, which you can use to detect faces in an image and persist information about facial features detected in an Amazon Rekognition collection. This is an example of a *storage-based* API operation because the service persists information on the server.

Amazon Rekognition Image provides the following storage API operations:

- IndexFaces (p. 305)
- ListFaces (p. 316)
- SearchFacesByImage (p. 329)
- SearchFaces (p. 326)
- DeleteFaces (p. 249)
- DescribeCollection (p. 254)

Amazon Rekognition Video provides the following storage API operations:

- StartFaceSearch (p. 345)
- CreateStreamProcessor (p. 244)

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 Amazon Rekognition Video operations. For example you can search a video for persons whose faces match those in an exisiting collection by calling StartFaceSearch (p. 345).

For information about creating and managing collections, see Searching Faces in a Collection (p. 137).

> **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. 305) 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. 305) are detected using the latest model.
- Your existing collections continue 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`. You can get the model version of an existing collection by calling DescribeCollection (p. 254). For more information, see Describing a Collection (p. 145).

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. 241)) 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. 247).

Getting Started with Amazon Rekognition

This section provides topics to get you started using Amazon Rekognition. If you're 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 IAM User (p. 11)
- Step 2: Set Up the AWS CLI and AWS SDKs (p. 12)
- Step 3: Getting Started Using the AWS CLI and AWS SDK API (p. 14)
- Step 4: Getting Started Using the Amazon Rekognition Console (p. 15)

Step 1: Set Up an AWS Account and Create an IAM 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're charged only for the services that you use.

With Amazon Rekognition, you pay only for the resources you use. If you're 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
   If you previously signed in to the AWS Management Console using AWS account root user credentials, choose **Sign in to a different account**. If you previously signed in to the console using IAM credentials, choose **Sign-in using root account credentials**. 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 verification code 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. This is 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 by using the credentials for your AWS account. Instead, we recommend that you:

- Use AWS Identity and Access Management (IAM) to create an IAM user.
- Add the user to an IAM group with administrative permissions.

You can then access AWS by 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 by using the IAM console. Follow the procedure to create an IAM user in your account.

To create an IAM user and sign in to the console

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

   Note
   An IAM user with administrator permissions has unrestricted access to the AWS services in your account. For information about restricting access to Amazon Rekognition operations, see Using Identity-Based Policies (IAM Policies) for Amazon Rekognition (p. 30). The code examples in this guide assume that you have a user with the `AmazonRekognitionFullAccess` permissions. `AmazonS3ReadOnlyAccess` is required for examples that access images or videos that are stored in an Amazon S3 bucket. The Amazon Rekognition Video stored video code examples also require `AmazonSQSFullAccess` permissions. Depending on your security requirements, you might want to use an IAM group that's limited to these permissions. For more information, see Creating IAM Groups.

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 CLI and AWS SDKs (p. 12)

Step 2: Set Up the AWS CLI and AWS SDKs

The following steps show you how to install the AWS Command Line Interface (AWS CLI) and AWS SDKs that the examples in this documentation use. There are a number of different ways to authenticate AWS SDK calls. The examples in this guide assume that you're using a default credentials profile for calling AWS CLI commands and AWS SDK API operations.

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

Follow the steps to download and configure the AWS SDKs.

To set up the AWS CLI and the AWS SDKs

1. Download and install the AWS CLI and the AWS SDKs that you want to use. This guide provides examples for the AWS CLI, Java, Python, PHP, .NET, and JavaScript. For information about other AWS SDKs, see Tools for Amazon Web Services.

   - AWS CLI
   - AWS SDK for Java
   - AWS SDK for Python (Boto 3)
   - AWS SDK for PHP
   - AWS SDK for .NET
   - AWS SDK for JavaScript

2. Create an access key for the user you created in Create an IAM User (p. 12).

   a. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
   b. In the navigation pane, choose **Users**.
   c. Choose the name of the user you created in Create an IAM User (p. 12).
   d. Choose the **Security credentials** tab.
   e. Choose **Create access key**. Then choose **Download .csv file** to save the access key ID and secret access key to a CSV file on your computer. Store the file in a secure location. You will not have access to the secret access key again after this dialog box closes. After you have downloaded the CSV file, choose **Close**.

3. If you have installed the AWS CLI, you can configure the credentials and region for most AWS SDKs by entering `aws configure` at the command prompt. Otherwise, use the following instructions.
4. On your computer, navigate to your home directory, and create an `.aws` directory. On Unix-based systems, such as Linux or macOS, this is in the following location:

   ```
   ~/.aws
   ```

   On Windows, this is in the following location:

   ```
   %HOMEPATH%\.aws
   ```

5. In the `.aws` directory, create a new file named `credentials`.
6. Open the credentials CSV file that you created in step 2 and copy its contents into the `credentials` file using the following format:

   ```
   [default]
   aws_access_key_id = your_access_key_id
   aws_secret_access_key = your_secret_access_key
   ```

   Substitute your acccess key ID and secret access key for *your_access_key_id* and *your_secret_access_key*.
7. Save the `Credentials` file and delete the CSV file.
8. In the `.aws` directory, create a new file named `config`.
9. Open the `config` file and enter your region in the following format.

   ```
   [default]
   region = your_aws_region
   ```

Substitute your desired AWS Region (for example, "us-west-2") for *your_aws_region*.

Note
If you don't select a region, then us-east-1 will be used by default.

10. Save the `config` file.

Next Step

Step 3: Getting Started Using the AWS CLI and AWS SDK API (p. 14)

Step 3: Getting Started Using the AWS CLI and AWS SDK API

After you've set up the AWS CLI and AWS SDKs that you want to use, you can build applications that use Amazon Rekognition. The following topics show you how to get started with Amazon Rekognition Image and Amazon Rekognition Video.

- Working with Images (p. 36)
- Working with Stored Videos (p. 59)
- Working with Streaming Videos (p. 86)

Formatting the AWS CLI Examples

The AWS CLI examples in this guide are formatted for the Linux operating system. To use the samples with Microsoft Windows, you 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 that's formatted for Microsoft Windows.

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

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

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

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

Next Step

Step 4: Getting Started Using the Amazon Rekognition Console (p. 15)

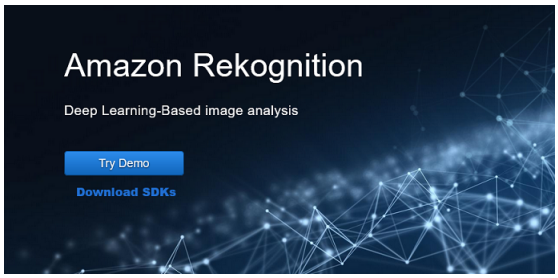
Step 4: Getting Started Using the Amazon Rekognition Console

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 3: Getting Started Using the AWS CLI and AWS SDK API (p. 14).

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

Topics

- Exercise 1: Detect Objects and Scenes in an Image (Console) (p. 15)
- Exercise 2: Analyze Faces in an Image (Console) (p. 20)
- Exercise 3: Compare Faces in Images (Console) (p. 22)
- Exercise 4: See Aggregated Metrics (Console) (p. 24)

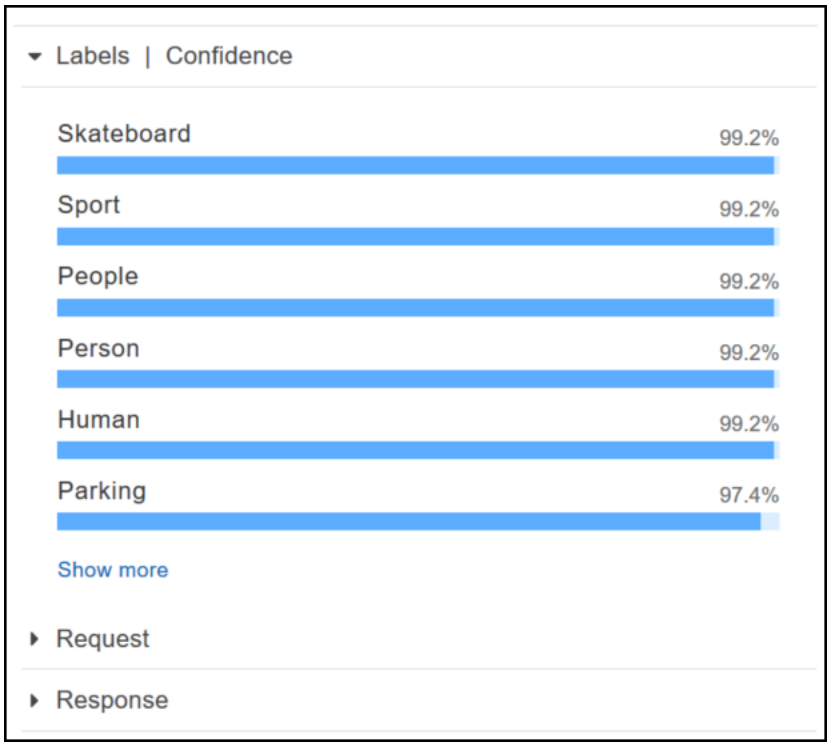


Exercise 1: Detect Objects and Scenes in an Image (Console)

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

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

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



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

Request


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


Response


```
{
   "Labels":[
      {
         "Confidence":99.25359344482422,
         "Name":"Skateboard"
      },
      {
         "Confidence":99.25359344482422,
         "Name":"Sport"
      },
      {
         "Confidence":99.24723052978516,
         "Name":"People"
      },
      {
         "Confidence":99.24723052978516,
         "Name":"Person"
      },
      {
         "Confidence":99.23908233642578,
         "Name":"Human"
      },
      {
         "Confidence":97.42484283447266,
         "Name":"Parking"
      },
      {
         "Confidence":97.42484283447266,
         "Name":"Parking Lot"
      },
      {
         "Confidence":91.53300476074219,
         "Name":"Automobile"
      },
      {
         "Confidence":91.53300476074219,
         "Name":"Car"
      },
      {
         "Confidence":91.53300476074219,
         "Name":"Vehicle"
      },
      {
         "Confidence":76.85114288330078,
         "Name":"Intersection"
      },
```

```
    {
       "Confidence":76.85114288330078,
       "Name":"Road"
    },
    {
       "Confidence":76.21503448486328,
       "Name":"Boardwalk"
    },
    {
       "Confidence":76.21503448486328,
       "Name":"Path"
    },
    {
       "Confidence":76.21503448486328,
       "Name":"Pavement"
    },
    {
       "Confidence":76.21503448486328,
       "Name":"Sidewalk"
    },
    {
       "Confidence":76.21503448486328,
       "Name":"Walkway"
    },
    {
       "Confidence":66.71541595458984,
       "Name":"Building"
    },
    {
       "Confidence":62.04711151123047,
       "Name":"Coupe"
    },
    {
       "Confidence":62.04711151123047,
       "Name":"Sports Car"
    },
    {
       "Confidence":61.98909378051758,
       "Name":"City"
    },
    {
       "Confidence":61.98909378051758,
       "Name":"Downtown"
    },
    {
       "Confidence":61.98909378051758,
       "Name":"Urban"
    },
    {
       "Confidence":60.978023529052734,
       "Name":"Neighborhood"
    },
    {
       "Confidence":60.978023529052734,
       "Name":"Town"
    },
    {
       "Confidence":59.22066116333008,
       "Name":"Sedan"
    },
    {
       "Confidence":56.48063278198242,
       "Name":"Street"
    },
    {
       "Confidence":54.235477447509766,
```

```
            "Name":"Housing"
         },
         {
            "Confidence":53.85226058959961,
            "Name":"Metropolis"
         },
         {
            "Confidence":52.001792907714844,
            "Name":"Office Building"
         },
         {
            "Confidence":51.325313568115234,
            "Name":"Suv"
         },
         {
            "Confidence":51.26075744628906,
            "Name":"Apartment Building"
         },
         {
            "Confidence":51.26075744628906,
            "Name":"High Rise"
         },
         {
            "Confidence":50.68067932128906,
            "Name":"Pedestrian"
         },
         {
            "Confidence":50.59548568725586,
            "Name":"Freeway"
         },
         {
            "Confidence":50.568580627441406,
            "Name":"Bumper"
         }
      ]
}
```


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

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

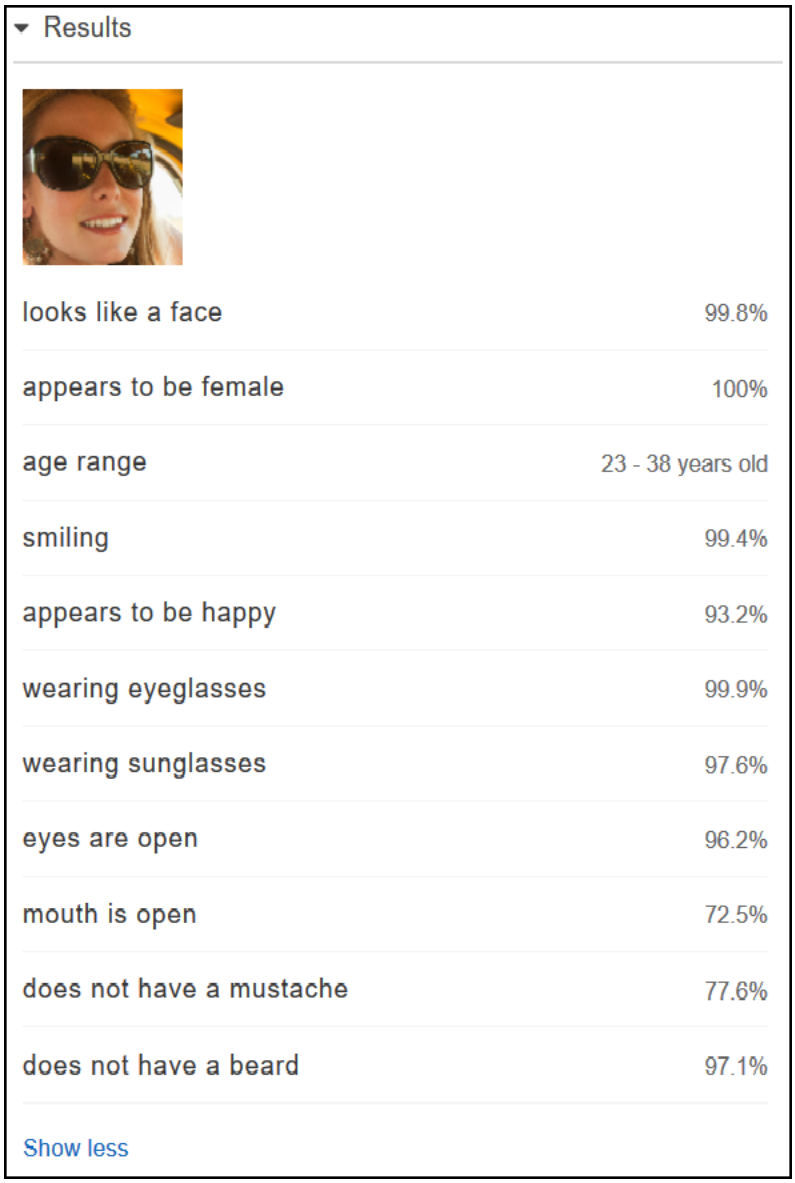
1. Open the Amazon Rekognition console.
2. Choose **Object and scene detection**.
3. Do one of the following:
   - Upload an image – Choose **Upload**, go to the location where you stored your image, and then select the image.
   - Use a URL – Type the URL in the text box, and then choose **Go**.
4.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

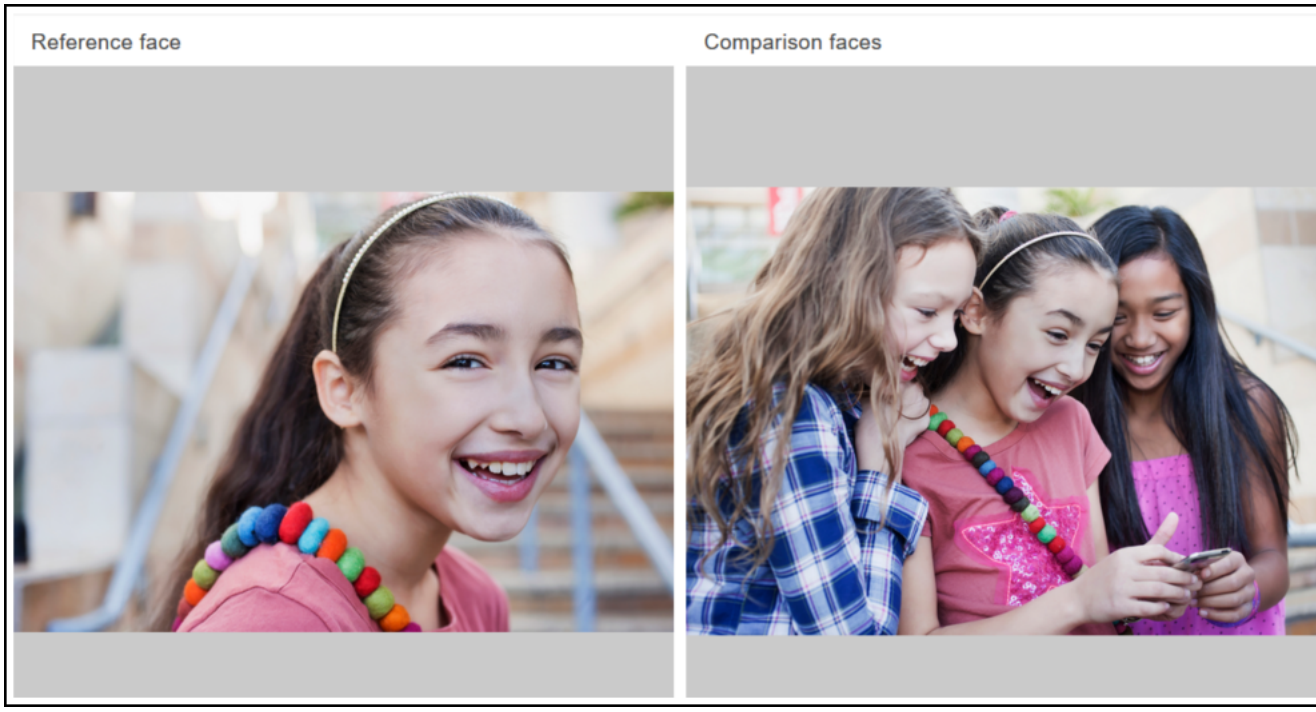
This section shows you how to use the Amazon Rekognition console to compare faces within a set of images with multiple faces in them. When you specify a **Reference face** (source) and a **Comparison faces** (target) image, Rekognition compares the largest face in the source image (that is, the reference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

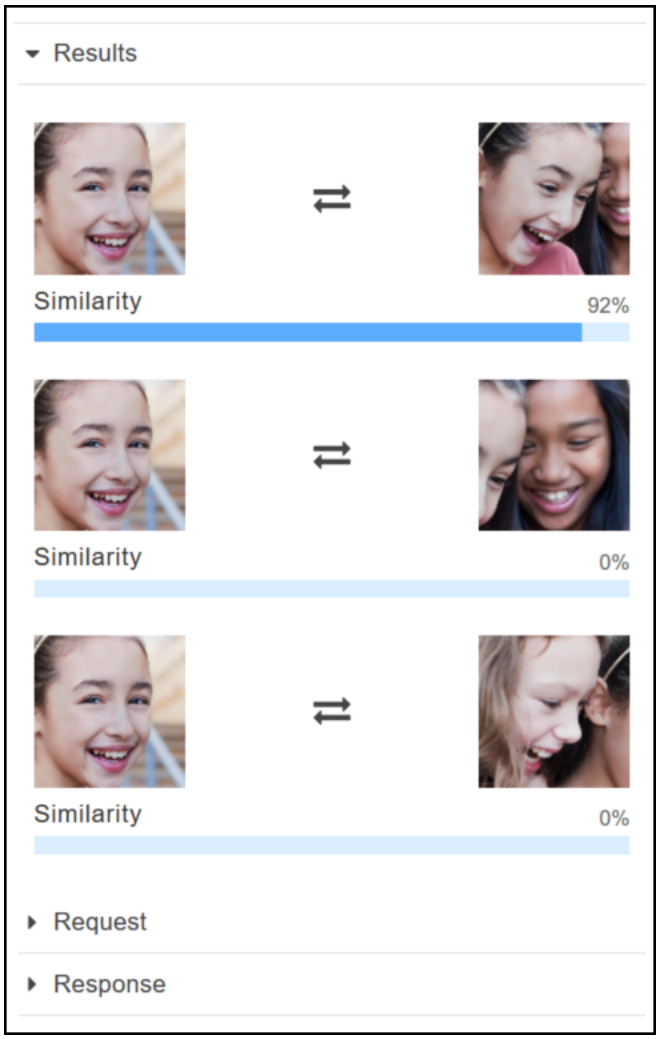
If the source image contains multiple faces, the service detects the largest face in the source image and uses it to compare with each face detected in the target image.

For more information, see Comparing Faces in Images (p. 124).

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. 227).

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

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

To allow users access to the Rekognition metrics pane, ensure that the user has appropriate CloudWatch and Rekognition permissions. For example, a user with `AmazonRekognitionReadOnlyAccess` and `CloudWatchReadOnlyAccess` managed policy permissions can see the metrics pane. If a user does not have the required permissions, when the user opens the metrics pane, no graphs appear. For more information, see Authentication and Access Control for Amazon Rekognition (p. 25).

For more information about monitoring Rekognition with CloudWatch see Monitoring (p. 224).

To see aggregated metrics (console)

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

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. 25)
- Access Control (p. 26)

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. 26)
- Using Identity-Based Policies (IAM Policies) for Amazon Rekognition (p. 30)
- Amazon Rekognition API Permissions: Actions, Permissions, and Resources Reference (p. 32)

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. 27)
- Understanding Resource Ownership (p. 27)
- Managing Access to Resources (p. 27)
- Specifying Policy Elements: Actions, Effects, and Principals (p. 29)

- Specifying Conditions in a Policy (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.

Resource Type	ARN Format
Collection ARN	`arn:aws:rekognition:region:account-id:collection/collection-id`
Stream Processor ARN	`arn:aws:rekognition:region:account-id:streamprocessor/stream-processor-name`

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. 32).

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. 28)
- Resource-Based Policies (p. 28)

Identity-Based Policies (IAM Policies)

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

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

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

The following is an example policy that lists all collections.

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

For more information about using identity-based policies with Amazon Rekognition, see Using Identity-Based Policies (IAM Policies) for Amazon Rekognition (p. 30). For more information about users, groups, roles, and permissions, see Identities (Users, Groups, and Roles) in the *IAM User Guide*.

Resource-Based Policies

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

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

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": [
                "arn:aws:s3:::Tests3bucket"
            ]
        }
    ]
}
```

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

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

Specifying Policy Elements: Actions, Effects, and Principals

For each Amazon Rekognition resource, the service defines a set of API operations. To grant permissions for these API operations, Amazon Rekognition defines a set of actions that you can specify in a policy. Some API operations can require permissions for more than one action in order to perform the API operation. For more information about resources and API operations, see Amazon Rekognition Resources and Operations (p. 27) and Amazon Rekognition Amazon Rekognition API Permissions: Actions, Permissions, and Resources Reference (p. 32).

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. 27).
- **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. 32).

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. 26).

Topics

- Permissions Required to Use the Amazon Rekognition Console (p. 31)
- AWS Managed (Predefined) Policies for Amazon Rekognition (p. 31)
- Customer Managed Policy Examples (p. 31)

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 a user read-only access to resources using a limited set of Amazon Rekognition operations.

For a table showing all of the Amazon Rekognition API operations and the resources that they apply to, see Amazon Rekognition API Permissions: Actions, Permissions, and Resources Reference (p. 32).

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. 31).

Examples

- Example 1: Allow a User Read-Only Access to Resources (p. 31)
- Example 2: Allow a User Full Access to Resources (p. 32)

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

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

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:CompareFaces",
                "rekognition:DetectFaces",
                "rekognition:DetectLabels",
                "rekognition:ListCollections",
                "rekognition:ListFaces",
                "rekognition:SearchFaces",
                "rekognition:SearchFacesByImage",
                "rekognition:DetectText",
                "rekognition:GetCelebrityInfo",
                "rekognition:RecognizeCelebrities",
                "rekognition:DetectModerationLabels",
                "rekognition:GetLabelDetection",
                "rekognition:GetFaceDetection",
                "rekognition:GetContentModeration",
                "rekognition:GetPersonTracking",
                "rekognition:GetCelebrityRecognition",
                "rekognition:GetFaceSearch",
                "rekognition:DescribeStreamProcessor",
                "rekognition:ListStreamProcessors"
            ],
            "Resource": "*"
        }
    ]
}
```

Example 2: Allow a User Full Access to Resources

The following example grants full access to Amazon Rekognition resources.

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

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

When you are setting up Access Control (p. 26) 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

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

API Operation: DeleteCollection

Required Permissions (API Action): rekognition:DeleteCollection

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

API Operation: DeleteFaces

Required Permissions (API Action): rekognition:DeleteFaces

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

API Operation: DetectFaces

Required Permissions (API Action): rekognition:DetectFaces

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

API Operation: IndexFaces

Required Permissions (API Action): rekognition:IndexFaces

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

API Operation: ListCollections

Required Permissions (API Action): rekognition:ListCollections

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

API Operation: ListFaces

Required Permissions (API Action): rekognition:ListFaces

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

API Operation: SearchFaces

Required Permissions (API Action): rekognition:SearchFaces

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

API Operation: SearchFacesByImage

Required Permissions (API Action): rekognition:SearchFacesByImage

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

API Operation: CreateStreamProcessor

Required Permissions (API Action): rekognition:CreateStreamProcessor

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

Resources: arn:aws:rekognition:*region*:*account-id*:*streamprocessor/stream-processor-name*

API Operation: DeleteStreamProcessor

Required Permissions (API Action): rekognition:DeleteStreamProcessor

Resources: arn:aws:rekognition:*region*:*account-id*:*streamprocessor/stream-processor-name*

API Operation: ListStreamProcessors

Required Permissions (API Action): rekognition:ListStreamProcessors

Resources: arn:aws:rekognition:*region*:*account-id*:*streamprocessor/stream-processor-name*

API Operation: StartStreamProcessor

Required Permissions (API Action): rekognition:StartStreamProcessor

Resources: arn:aws:rekognition:*region*:*account-id*:*streamprocessor/stream-processor-name*

API Operation: StopStreamProcessor

Required Permissions (API Action): rekognition:StopStreamProcessor

Resources: arn:aws:rekognition:*region*:*account-id*:*streamprocessor/stream-processor-name*

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 Amazon Rekognition Image can perform 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 Amazon 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).

Amazon Rekognition Image can also store facial metadata in collections for later retrieval. For more information, see Searching Faces in a Collection (p. 137).

In this section, you use the Amazon 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. 36)
- Best Practices for Working With Images (p. 37)
- Analyzing Images Stored in an Amazon S3 Bucket (p. 38)
- Analyzing an Image Loaded from a Local File System (p. 41)
- Getting Image Orientation and Bounding Box Coordinates (p. 49)

Images

Amazon Rekognition Image operations can analyze images in .jpg or .png format.

You pass image bytes to an Amazon 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 Analyzing Images Stored in an Amazon S3 Bucket (p. 38). For an example of passing image bytes to an Amazon Rekognition Image API operation, see Analyzing an Image Loaded from a Local File System (p. 41).

If you use HTTP and pass the image bytes as part of an Amazon 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.

The following common AWS SDKs automatically base64-encode images, and you don't need to encode image bytes before calling an Amazon Rekognition Image API operation.

- Java
- JavaScript
- Python
- PHP

If you're 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.

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

> **Note**
> The image doesn't need to be base64 encoded if you pass an image stored in an `S3Object` instead of image bytes.

For information about ensuring the lowest possible latency for Amazon Rekognition Image operations, see Amazon Rekognition Image Operation Latency (p. 37).

Correcting Image Orientation

In several Rekognition API operations, the orientation of an analyzed image is returned. For example, the DetectFaces (p. 261) 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. 49).

Best Practices for Working With Images

The following are best practices for working with images.

Amazon Rekognition Image Operation Latency

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

- The region for the Amazon S3 bucket containing your images must match the region you use for Amazon Rekognition Image API operations.
- Calling an Amazon 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 Amazon Rekognition Image operation. Consider this approach if you are uploading images to Amazon 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 Amazon 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

The models used for face and celebrity recognition operations are designed to work in as wide a variety of poses, rotations, lighting conditions, and range of sizes as possible. We recommend the following guidelines when choosing reference photos for facial recognition:

- Have a front-facing face.
- Have a face that is not obscured or tightly cropped. The image should contain the full head and shoulders of the person. It should not be cropped to the face bounding box.
- Avoid occlusions such as head-bands or masks.
- Have a face that occupies a large proportion of the image. Proportionally larger faces are matched with greater accuracy.
- Be sufficiently large. Amazon Rekognition Image can detect faces as small as 40x40 pixels in an HD resolution image (1920x1080). Higher-resolution images require a larger minimum face size. Faces larger than the minimum size provide more accurate detection and attribute analysis.
- Use color images.
- 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 bright and sharp. You can use DetectFaces (p. 261) to determine the brightness and sharpness of a face.

Faces recognized by Amazon Rekognition can be used to help identify potential matches in public safety and law enforcement scenarios. For more information, see Using Amazon Rekognition to Help Public Safety (p. 139).

Analyzing Images Stored in an Amazon S3 Bucket

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

In this topic, you use the DetectLabels (p. 265) API operation to detect objects, concepts, and scenes in an image (JPEG or PNG) that's stored in an Amazon S3 bucket. You pass an image to an Amazon Rekognition Image API operation by using the the section called “Image” (p. 388) input parameter. Within `Image`, you specify the S3Object (p. 407) object property to reference an image stored in an S3 bucket. Image bytes for images stored in Amazon S3 buckets don't need to be base64 encoded. For more information, see Images (p. 36).

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

In this example JSON request for `DetectLabels`, the source image (`input.jpg`) is loaded from an Amazon S3 bucket named `MyBucket`.

```
{
    "Image": {
        "S3Object": {
            "Bucket": "MyBucket",
            "Name": "input.jpg"
        }
    },
    "MaxLabels": 10,
    "MinConfidence": 75
}
```

The following examples use various AWS SDKs and the AWS CLI to call `DetectLabels`. For information about the `DetectLabels` operation response, see DetectLabels Response (p. 107).

To detect labels in an image

1. If you haven't already:

a. Create or update an IAM user with `AmazonRekognitionFullAccess` and `AmazonS3ReadOnlyAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).

b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).

2. Upload an image that contains one or more objects—such as trees, houses, and boat—to your S3 bucket. The image must be in *.jpg* or *.png* format.

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

3. Use the following examples to call the `DetectLabels` operation.

   Java

   This example displays a list of labels that were detected in the input image. Replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and image that you used in step 2.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package com.amazonaws.samples;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
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 DetectLabels {

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

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


      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      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();
        }
    }
}
```

AWS CLI

This example displays the JSON output from the `detect-labels` CLI operation. Replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and image that you used in Step 2.

```
aws rekognition detect-labels \
--image '{"S3Object":{"Bucket":"bucket","Name":"file"}}'
```

Python

This example displays the labels that were detected in the input image. Replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and image that you used in Step 2.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

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

    client=boto3.client('rekognition')

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

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

.NET

This example displays a list of labels that were detected in the input image. Replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and image that you used in Step 2.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DetectLabels
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "bucket";
```

```
        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        DetectLabelsRequest detectlabelsRequest = new DetectLabelsRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket
                },
            },
            MaxLabels = 10,
            MinConfidence = 75F
        };

        try
        {
            DetectLabelsResponse detectLabelsResponse =
 rekognitionClient.DetectLabels(detectlabelsRequest);
            Console.WriteLine("Detected labels for " + photo);
            foreach (Label label in detectLabelsResponse.Labels)
                Console.WriteLine("{0}: {1}", label.Name, label.Confidence);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

Analyzing an Image Loaded from a Local File System

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

These topics provide examples of supplying image bytes to Amazon Rekognition Image API operations by using a file loaded from a local file system. You pass image bytes to an Amazon Rekognition API operation by using the Image (p. 388) input parameter. Within `Image`, you specify the `Bytes` property to pass base64-encoded image bytes.

Image bytes passed to an Amazon Rekognition API operation by using the `Bytes` input parameter must be base64 encoded. The AWS SDKs that these examples use automatically base64-encode images. You don't need to encode image bytes before calling an Amazon Rekognition API operation. For more information, see Images (p. 36).

In this example JSON request for `DetectLabels`, the source image bytes are passed in the `Bytes` input parameter.

```
{
    "Image": {
        "Bytes": "/9j/4AAQSk....."
    },
    "MaxLabels": 10,
    "MinConfidence": 77
}
```

The following examples use various AWS SDKs and the AWS CLI to call `DetectLabels`. For information about the `DetectLabels` operation response, see DetectLabels Response (p. 107).

For a client-side JavaScript example, see Using JavaScript (p. 45).

To detect labels in a local image

1. If you haven't already:
   a. Create or update an IAM user with `AmazonRekognitionFullAccess` and `AmazonS3ReadOnlyAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).
2. Use the following examples to call the `DetectLabels` operation.

   Java

   The following Java example shows how to load an image from the local file system and detect labels by using the detectLabels AWS SDK operation. Change the value of `photo` to the path and file name of an image file (.jpg or .png format).

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
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.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 DetectLabelsLocalFile {
    public static void main(String[] args) throws Exception {
     String photo="input.jpg";


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


        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        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();
        }

    }
}
```

Python

The following AWS SDK for Python example shows how to load an image from the local file system and call the detect_labels operation. Change the value of `imageFile` to the path and file name of an image file (.jpg or .png format).

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

if __name__ == "__main__":

    imageFile='input.jpg'
    client=boto3.client('rekognition')

    with open(imageFile, 'rb') as image:
        response = client.detect_labels(Image={'Bytes': image.read()})

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

    print('Done...')
```

.NET

The following example shows how to load an image from the local file system and detect labels by using the `DetectLabels` operation. Change the value of `photo` to the path and file name of an image file (.jpg or .png format).

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using System.IO;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;
```

```
public class DetectLabelsLocalfile
{
    public static void Example()
    {
        String photo = "input.jpg";

        Amazon.Rekognition.Model.Image image = new
 Amazon.Rekognition.Model.Image();
        try
        {
            using (FileStream fs = new FileStream(photo, FileMode.Open,
 FileAccess.Read))
            {
                byte[] data = null;
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                image.Bytes = new MemoryStream(data);
            }
        }
        catch (Exception)
        {
            Console.WriteLine("Failed to load file " + photo);
            return;
        }

        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        DetectLabelsRequest detectlabelsRequest = new DetectLabelsRequest()
        {
            Image = image,
            MaxLabels = 10,
            MinConfidence = 77F
        };

        try
        {
            DetectLabelsResponse detectLabelsResponse =
 rekognitionClient.DetectLabels(detectlabelsRequest);
            Console.WriteLine("Detected labels for " + photo);
            foreach (Label label in detectLabelsResponse.Labels)
                Console.WriteLine("{0}: {1}", label.Name, label.Confidence);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

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. Change the value of `photo` to the path and file name of an image file (.jpg or .png format).

```
<?php
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    require 'vendor/autoload.php';

    use Aws\Rekognition\RekognitionClient;
```

```
    $options = [
       'region'            => 'us-west-2',
        'version'           => 'latest'
    ];

    $rekognition = new RekognitionClient($options);

    // Get local image
    $photo = 'input.jpg';
    $fp_image = fopen($photo, 'r');
    $image = fread($fp_image, filesize($photo));
    fclose($fp_image);


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

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

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

Using JavaScript

The following JavaScript webpage example allows a user to choose an image and view the estimated ages of faces that are detected in the image. The estimated ages are returned by a call to the section called “DetectFaces” (p. 261).

The chosen image is loaded by using the JavaScript `FileReader.readAsDataURL` function, which base64-encodes the image. This is useful for displaying the image on an HTML canvas. But, it means the image bytes have to be unencoded before they're passed to an Amazon Rekognition Image operation. This example shows how to unencode the loaded image bytes. If the encoded image bytes aren't useful to you, use `FileReader.readAsArrayBuffer` instead because the loaded image isn't encoded. This means that Amazon Rekognition Image operations can be called without first unencoding the image bytes. For an example, see Using readAsArrayBuffer (p. 47).

To run the JavaScript example

1. Load the example source code into an editor.
2. Get the Amazon Cognito identity pool identifier. For more information, see Getting the Amazon Cognito Identity Pool Identifier (p. 48).
3. In the `AnonLog` function of the example code, change `IdentityPoolIdToUse` and `RegionToUse` to the values that you noted in step 9 of Getting the Amazon Cognito Identity Pool Identifier (p. 48).

4. In the `DetectFaces` function, change `RegionToUse` to the value you used in the previous step.
5. Save the example source code as an `.html` file.
6. Load the file into your browser.
7. Choose the **Browse...** button, and choose an image that contains one or more faces. A table is shown that contains the estimated ages for each face detected in the image.

JavaScript Example Code

```
<!--
Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)
-->
<!DOCTYPE html>
<html>
<head>
  <script src="aws-cognito-sdk.min.js"></script>
  <script src="amazon-cognito-identity.min.js"></script>
  <script src="https://sdk.amazonaws.com/js/aws-sdk-2.16.0.min.js"></script>
  <script src="./app.js"></script>
  <meta charset="UTF-8">
  <title>Rekognition</title>
</head>

<body>
  <H1>Age Estimator</H1>
  <input type="file" name="fileToUpload" id="fileToUpload" accept="image/*">
  <p id="opResult"></p>
</body>
<script>

  document.getElementById("fileToUpload").addEventListener("change", function (event) {
    ProcessImage();
  }, false);

  //Calls DetectFaces API and shows estimated ages of detected faces
  function DetectFaces(imageData) {
    AWS.region = "RegionToUse";
    var rekognition = new AWS.Rekognition();
    var params = {
      Image: {
        Bytes: imageData
      },
      Attributes: [
        'ALL',
      ]
    };
    rekognition.detectFaces(params, function (err, data) {
      if (err) console.log(err, err.stack); // an error occurred
      else {
       var table = "<table><tr><th>Low</th><th>High</th></tr>";
        // show each face and build out estimated age table
        for (var i = 0; i < data.FaceDetails.length; i++) {
          table += '<tr><td>' + data.FaceDetails[i].AgeRange.Low +
            '</td><td>' + data.FaceDetails[i].AgeRange.High + '</td></tr>';
        }
        table += "</table>";
        document.getElementById("opResult").innerHTML = table;
      }
    });
  }
  //Loads selected image and unencodes image bytes for Rekognition DetectFaces API
```

```
  function ProcessImage() {
    AnonLog();
    var control = document.getElementById("fileToUpload");
    var file = control.files[0];

    // Load base64 encoded image
    var reader = new FileReader();
    reader.onload = (function (theFile) {
      return function (e) {
        var img = document.createElement('img');
        var image = null;
        img.src = e.target.result;
        var jpg = true;
        try {
          image = atob(e.target.result.split("")[1]);

        } catch (e) {
          jpg = false;
        }
        if (jpg == false) {
          try {
            image = atob(e.target.result.split("")[1]);
          } catch (e) {
            alert("Not an image file Rekognition can process");
            return;
          }
        }
        //unencode image bytes for Rekognition DetectFaces API
        var length = image.length;
        imageBytes = new ArrayBuffer(length);
        var ua = new Uint8Array(imageBytes);
        for (var i = 0; i < length; i++) {
          ua[i] = image.charCodeAt(i);
        }
        //Call Rekognition
        DetectFaces(imageBytes);
      };
    })(file);
    reader.readAsDataURL(file);
  }
  //Provides anonymous log on to AWS services
  function AnonLog() {

    // Configure the credentials provider to use your identity pool
    AWS.config.region = 'RegionToUse'; // Region
    AWS.config.credentials = new AWS.CognitoIdentityCredentials({
      IdentityPoolId: 'IdentityPoolIdToUse',
    });
    // Make the call to obtain credentials
    AWS.config.credentials.get(function () {
      // Credentials will be available when this function is called.
      var accessKeyId = AWS.config.credentials.accessKeyId;
      var secretAccessKey = AWS.config.credentials.secretAccessKey;
      var sessionToken = AWS.config.credentials.sessionToken;
    });
  }
</script>
</html>
```

Using readAsArrayBuffer

The following code snippet is an alternative implementation of the `ProcessImage` function in the sample code. It uses `readAsArrayBuffer` to load an image and call `DetectFaces`. Because

`readAsArrayBuffer` doesn't base64-encode the loaded file, it's not necessary to unencode the image bytes before calling an Amazon Rekognition Image operation.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

function ProcessImage() {
    AnonLog();
    var control = document.getElementById("fileToUpload");
    var file = control.files[0];

    // Load base64 encoded image for display
    var reader = new FileReader();
    reader.onload = (function (theFile) {
      return function (e) {
        //Call Rekognition
        AWS.region = "RegionToUse";
        var rekognition = new AWS.Rekognition();
        var params = {
          Image: {
          Bytes: e.target.result
        },
        Attributes: [
        'ALL',
      ]
    };
    rekognition.detectFaces(params, function (err, data) {
      if (err) console.log(err, err.stack); // an error occurred
      else {
       var table = "<table><tr><th>Low</th><th>High</th></tr>";
        // show each face and build out estimated age table
        for (var i = 0; i < data.FaceDetails.length; i++) {
          table += '<tr><td>' + data.FaceDetails[i].AgeRange.Low +
            '</td><td>' + data.FaceDetails[i].AgeRange.High + '</td></tr>';
        }
        table += "</table>";
        document.getElementById("opResult").innerHTML = table;
      }
    });

      };
    })(file);
    reader.readAsArrayBuffer(file);
  }
```

Getting the Amazon Cognito Identity Pool Identifier

For simplicity, the example uses an anonymous Amazon Cognito identity pool to provide unauthenticated access to the Amazon Rekognition Image API. This might be suitable for your needs. For example, you can use unauthenticated access to provide free, or trial, access to your website before users sign up. To provide authenticated access, use an Amazon Cognito user pool. For more information, see Amazon Cognito User Pool.

The following procedure shows how to create an identity pool that enables access to unauthenticated identities, and how to get the identity pool identifier that's needed in the example code.

To get the identity pool identifier

1. Open the Amazon Cognito console.
2. Choose **Create new identity pool**.
3. For **Identity pool name***, type a name for your identity pool.

4. In **Unauthenticated identites**, choose **Enable access to unauthenticated identities**.
5. Choose **Create Pool**.
6. Choose **View Details**, and note the role name for unauthenticated identities.
7. Choose **Allow**.
8. In **Platform**, choose **JavaScript**.
9. In **Get AWS Credentials**, note the values of *Identity pool identifier* and *region* that are shown in the code snippet.
10. Open the IAM console at https://console.aws.amazon.com/iam/.
11. In the navigation pane, choose **Roles**.
12. Choose the role name that you noted in step 6.
13. In **Permissions**, choose **Attach Policy**.
14. Choose **AmazonRekognitionReadOnlyAccess**.
15. Choose **Attach Policy**.

Getting Image Orientation and Bounding Box Coordinates

Applications that use Amazon Rekognition Image commonly need to display the images that are detected by Amazon 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, Amazon Rekognition Image operations return the estimated orientation.

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

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

- CompareFaces (p. 236)
- DetectFaces (p. 261)
- IndexFaces (p. 305)
- RecognizeCelebrities (p. 322)

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

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

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

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

Finding an Image's Orientation

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

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

Some images in .jpg format contain orientation information in Exif metadata. If the value of the `OrientationCorrection` field is `null` in the operation's response, the Exif metadata for the image contains the orientation. In the Exif metadata, you can find the image's orientation in the `orientation` field. Although Amazon Rekognition 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, Amazon 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, Amazon 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 Amazon 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. 363).

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

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

If an image doesn't have Exif metadata, or if the `orientation` field in the Exif metadata is not populated, Amazon Rekognition 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 Amazon 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 Amazon 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 (p. 115).
2. Note the estimated orientation returned in the response's `OrientationCorrection` field.
3. Rotate the image to 0 degrees orientation by using the estimated orientation you noted in step 2 in your code.
4. Translate the top and left bounding box coordinates to 0 degrees orientation and convert them to pixel points on the image in your code. Use the formula in the following list that matches the estimated orientation you noted in step 2.

   Note the following definitions:

   - `ROTATE_(n)` is the estimated image orientation returned by an Amazon Rekognition Image operation.
   - `<face>` represents information about the face that is returned by an Amazon Rekognition Image operation. For example, the FaceDetail (p. 379) data type that the DetectFaces (p. 261) 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. 363).

ROTATE_0

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

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

ROTATE_90

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

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

ROTATE_180

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

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

ROTATE_270

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

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

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

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

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

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

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

Displaying Bounding Boxes When Orientation Information is Present in Exif Metadata

If an image's orientation is included in Exif metadata, Amazon Rekognition 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 the value of `photo` with the name and path of an image that is stored locally in either .png or .jpg format.

Java

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
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.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.util.IOUtils;
import com.amazonaws.services.rekognition.model.AgeRange;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.Attribute;
import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.model.DetectFacesRequest;
import com.amazonaws.services.rekognition.model.DetectFacesResult;
import com.amazonaws.services.rekognition.model.FaceDetail;

public class RotateImage {

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

  String photo = "input.png";

  //Get Rekognition client
 AmazonRekognition amazonRekognition = AmazonRekognitionClientBuilder.defaultClient();


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

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

  }
  catch(Exception e)
  {
```

```
      System.out.println("Failed to load file " + photo);
      System.exit(1);
  }

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

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

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

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

  //Call detect faces and show face age and placement

  try{
    DetectFacesRequest request = new DetectFacesRequest()
            .withImage(new Image()
               .withBytes((imageBytes)))
            .withAttributes(Attribute.ALL);


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

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

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

}


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

  float left = 0;
  float top = 0;

  if(rotation==null){
      System.out.println("No estimated estimated orientation. Check Exif data.");
      return;
  }
  //Calculate face position based on image orientation.
  switch (rotation) {
     case "ROTATE_0":
```

```
        left = imageWidth * box.getLeft();
        top = imageHeight * box.getTop();
        break;
     case "ROTATE_90":
        left = imageHeight * (1 - (box.getTop() + box.getHeight()));
        top = imageWidth * box.getLeft();
        break;
     case "ROTATE_180":
        left = imageWidth - (imageWidth * (box.getLeft() + box.getWidth()));
        top = imageHeight * (1 - (box.getTop() + box.getHeight()));
        break;
     case "ROTATE_270":
        left = imageHeight * box.getTop();
        top = imageWidth * (1 - box.getLeft() - box.getWidth());
        break;
     default:
        System.out.println("No estimated orientation information. Check Exif data.");
        return;
  }

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

  }
}
```

Python

This example uses the PIL/Pillow image library to get the image width and height. For more information, see Pillow. This example preserves exif metadata which you might need elsewhere in your application. If you choose to not save the exif metadata, the estimated orientation is returned from the call to `DetectFaces`.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
import io
from PIL import Image

# Calculate positions from from estimated rotation
def ShowBoundingBoxPositions(imageHeight, imageWidth, box, rotation):
    left = 0
    top = 0

    if rotation == 'ROTATE_0':
        left = imageWidth * box['Left']
        top = imageHeight * box['Top']

    if rotation == 'ROTATE_90':
        left = imageHeight * (1 - (box['Top'] + box['Height']))
        top = imageWidth * box['Left']

    if rotation == 'ROTATE_180':
        left = imageWidth - (imageWidth * (box['Left'] + box['Width']))
        top = imageHeight * (1 - (box['Top'] + box['Height']))
```

```
    if rotation == 'ROTATE_270':
        left = imageHeight * box['Top']
        top = imageWidth * (1- box['Left'] - box['Width'] )

    print('Left: ' + '{0:.0f}'.format(left))
    print('Top: ' + '{0:.0f}'.format(top))
    print('Face Width: ' + "{0:.0f}".format(imageWidth * box['Width']))
    print('Face Height: ' + "{0:.0f}".format(imageHeight * box['Height']))


if __name__ == "__main__":

    photo='input.png'
    client=boto3.client('rekognition')


    #Get image width and height
    image = Image.open(open(photo,'rb'))
    width, height = image.size

    print ('Image information: ')
    print (photo)
    print ('Image Height: ' + str(height))
    print('Image Width: ' + str(width))


    # call detect faces and show face age and placement
    # if found, preserve exif info
    stream = io.BytesIO()
    if 'exif' in image.info:
        exif=image.info['exif']
        image.save(stream,format=image.format, exif=exif)
    else:
        image.save(stream, format=image.format)
    image_binary = stream.getvalue()

    response = client.detect_faces(Image={'Bytes': image_binary}, Attributes=['ALL'])

    print('Detected faces for ' + photo)
    for faceDetail in response['FaceDetails']:
        print ('Face:')
        if 'OrientationCorrection'  in response:
            print('Orientation: ' + response['OrientationCorrection'])
            ShowBoundingBoxPositions(height, width, faceDetail['BoundingBox'],
 response['OrientationCorrection'])

        else:
            print ('No estimated orientation. Check Exif data')

        print('The detected face is estimated to be between ' +
 str(faceDetail['AgeRange']['Low'])
              + ' and ' + str(faceDetail['AgeRange']['High']) + ' years')
        print()
```

.NET

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)
```

```
using System;
using System.Collections.Generic;
using System.IO;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class ImageOrientationAndBoundingBox
{
    public static void Example()
    {
        String photo = "photo.jpg";

        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        Image image = new Image();
        try
        {
            using (FileStream fs = new FileStream(photo, FileMode.Open,
 FileAccess.Read))
            {
                byte[] data = null;
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                image.Bytes = new MemoryStream(data);
            }
        }
        catch (Exception)
        {
            Console.WriteLine("Failed to load file " + photo);
            return;
        }

        int height;
        int width;
        // Used to extract original photo width/height
        using (System.Drawing.Bitmap imageBitmap = new System.Drawing.Bitmap(photo))
        {
            height = imageBitmap.Height;
            width = imageBitmap.Width;
        }

        Console.WriteLine("Image Information:");
        Console.WriteLine(photo);
        Console.WriteLine("Image Height: " + height);
        Console.WriteLine("Image Width: " + width);

        try
        {
            DetectFacesRequest detectFacesRequest = new DetectFacesRequest()
            {
                Image = image,
                Attributes = new List<String>() { "ALL" }
            };

            DetectFacesResponse detectFacesResponse =
 rekognitionClient.DetectFaces(detectFacesRequest);
            foreach (FaceDetail face in detectFacesResponse.FaceDetails)
            {
                Console.WriteLine("Face:");
                ShowBoundingBoxPositions(height, width,
                    face.BoundingBox, detectFacesResponse.OrientationCorrection);
                Console.WriteLine("BoundingBox: top={0} left={1} width={2} height={3}",
 face.BoundingBox.Left,
                    face.BoundingBox.Top, face.BoundingBox.Width,
 face.BoundingBox.Height);
```

```
                Console.WriteLine("The detected face is estimated to be between " +
                    face.AgeRange.Low + " and " + face.AgeRange.High + " years old.");
                Console.WriteLine();
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }

    public static void ShowBoundingBoxPositions(int imageHeight, int imageWidth,
 BoundingBox box, String rotation)
    {
        float left = 0;
        float top = 0;

        if (rotation == null)
        {
            Console.WriteLine("No estimated estimated orientation. Check Exif data.");
            return;
        }
        //Calculate face position based on image orientation.
        switch (rotation)
        {
            case "ROTATE_0":
                left = imageWidth * box.Left;
                top = imageHeight * box.Top;
                break;
            case "ROTATE_90":
                left = imageHeight * (1 - (box.Top + box.Height));
                top = imageWidth * box.Left;
                break;
            case "ROTATE_180":
                left = imageWidth - (imageWidth * (box.Left + box.Width));
                top = imageHeight * (1 - (box.Top + box.Height));
                break;
            case "ROTATE_270":
                left = imageHeight * box.Top;
                top = imageWidth * (1 - box.Left - box.Width);
                break;
            default:
                Console.WriteLine("No estimated orientation information. Check Exif
 data.");
                return;
        }

        //Display face location information.
        Console.WriteLine("Left: " + left);
        Console.WriteLine("Top: " + top);
        Console.WriteLine("Face Width: " + imageWidth * box.Width);
        Console.WriteLine("Face Height: " + imageHeight * box.Height);

    }
}
```

Working with Stored Videos

Amazon Rekognition Video is an API that you can use to analyze videos. With Amazon 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 Amazon Rekognition Video in categories such as media/entertainment and public safety. Previously, scanning videos for objects or people would have taken many hours of error-prone viewing by a human being. Amazon Rekognition Video automates the detection of items and when they occur throughout a video.

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

Topics

- Types of Detection and Recognition (p. 59)
- Amazon Rekognition Video API Overview (p. 59)
- Calling Amazon Rekognition Video Operations (p. 61)
- Configuring Amazon Rekognition Video (p. 66)
- Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68)
- Analyzing a Video with the AWS Command Line Interface (p. 75)
- Tutorial: Creating an Amazon Rekognition Lambda Function (p. 77)
- Reference: Video Analysis Results Notification (p. 83)
- Troubleshooting Amazon Rekognition Video (p. 84)

Types of Detection and Recognition

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

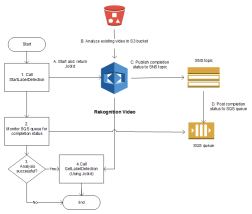
- Labels (p. 102)
- Faces (p. 114)
- People (p. 184)
- Celebrities (p. 190)
- Suggestive and explicit adult content (p. 204)

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

Amazon Rekognition Video API Overview

Amazon 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. 349). 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. 296), 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.



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.

Detection	Start Operation	Get Operation
People	StartPersonTracking (p. 353)	GetPersonTracking (p. 300)
Faces	StartFaceDetection (p. 341)	GetFaceDetection (p. 286)
Labels	StartLabelDetection (p. 349)	GetLabelDetection (p. 296)
Celebrities	StartCelebrityRecognition (p. 333)	GetCelebrityRecognition (p. 277)
Explicit or suggestive adult content	StartContentModeration (p. 337)	GetContentModeration (p. 282)

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

For more information about using Amazon Rekognition Video, see Calling Amazon Rekognition Video Operations (p. 61). For an example that does video analysis by using Amazon SQS, see Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68). For AWS CLI examples, see Analyzing a Video with the AWS Command Line Interface (p. 75).

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 Amazon 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. 305) operation. You can then use StartFaceSearch (p. 345) to start asynchronously searching for faces in the collection. You use GetFaceSearch (p. 291) to get the search results. For more information, see Searching Stored

Videos for Faces (p. 176). 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. 86).

Calling Amazon Rekognition Video Operations

Amazon 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 an Amazon Rekognition Video `Start` operation, such as StartPersonTracking (p. 353). Amazon 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. 300).

The information in the following sections uses label detection operations to show how Amazon 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 Amazon Rekognition Video operations—for example, StartFaceDetection (p. 341) and StartPersonTracking (p. 353). The example Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68) 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 Amazon Rekognition Video examples, such as People Pathing (p. 184). For AWS CLI examples, see Analyzing a Video with the AWS Command Line Interface (p. 75).

Topics

- Starting Video Analysis (p. 61)
- Getting the Completion Status of an Amazon Rekognition Video Analysis Request (p. 62)
- Getting Amazon Rekognition Video Analysis Results (p. 63)

Starting Video Analysis

You start an Amazon Rekognition Video label detection request by calling StartLabelDetection (p. 349). The following is an example of a JSON request that's passed by `StartLabelDetection`.

```
{
    "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"
}
```

The input parameter `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 Amazon Rekognition Video notifies when the video analysis request finishes. The Amazon SNS

topic must be in the same AWS region as the Amazon Rekognition Video endpoint that you're calling. `NotificationChannel` also contains the ARN for a role that allows Amazon 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 Configuring Amazon Rekognition Video (p. 66).

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 same `Start` operation and the same input parameters, the same `JobId` is returned. The job is not performed again and Amazon 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` (HTTP status code: 400) exception raised.
- If you reuse the token with a different `Start` operation, the operation succeeds.

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

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

If you start too many jobs concurrently, calls to `StartLabelDetection` raise a `LimitExceededException` exception (HTTP status code: 400) until the number of concurrently running jobs is below the Amazon Rekognition service limit.

If you find that LimitExceededException exceptions are raised with bursts of activity, consider using an Amazon SQS queue to manage incoming requests. Contact AWS support if you find that your average number of concurrent requests cannot be managed by an Amazon SQS queue and you are still receiving `LimitExceededException` exceptions.

Getting the Completion Status of an Amazon Rekognition Video Analysis Request

Amazon 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.

```
{
    "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. 83).

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

- **AWS Lambda** – You can subscribe an AWS Lambda function that you write to an Amazon SNS topic. The function is called when Amazon Rekognition notifies the Amazon SNS topic that the request has completed. Use a Lambda function if you want server-side code to process the results of a video analysis request. For example, you might want to use server-side code to annotate the video or create a report on the video contents before returning the information to a client application. We also recommend server-side processing for large videos because the Amazon Rekognition API might return large volumes of data.
- **Amazon Simple Queue Service** – You can subscribe an Amazon SQS queue to an Amazon SNS topic. You then poll the Amazon SQS queue to retrieve the completion status that's published by Amazon Rekognition when a video analysis request completes. For more information, see Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68). Use an Amazon SQS queue if you want to call Amazon Rekognition Video operations only from a client application.

Important
We don't recommend getting the request completion status by repeatedly calling the Amazon Rekognition Video `Get` operation. This is because Amazon Rekognition Video throttles the `Get` operation if too many requests are made. If you're processing multiple videos concurrently, it's simpler and more efficient to monitor one SQS queue for the completion notification than to poll Amazon Rekognition Video for the status of each video individually.

Getting Amazon 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:

```
{
    "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. The default value for `MaxResults` is 1000. If you specify a value greater than 1000, a maximum of 1000 results is returned. 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.

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 `GetLabelDetection` operation response JSON is similar to the following:

```
{
    "JobStatus": "SUCCEEDED",
    "Labels": [
        {
            "Label": {
                "Confidence": 56.49449920654297,
                "Name": "Bowl"
```

```
            },
            "Timestamp": 0
        },
        {
            "Label": {
                "Confidence": 77.5353012084961,
                "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":
 "WlKfzZED1PzBLyAmTpda655kICVnPPAtQ/8V9Mci6097JKjqP08uQun6j6fwGEaJFDUQVawYsg==",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 67301,
        "FileExtension": "mp4",
        "Format": "QuickTime / MOV",
        "FrameHeight": 1080,
        "FrameRate": 29.970029830932617,
        "FrameWidth": 1920
    }
}
```

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*.

```
{
    "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": 9175
        },
        {
            "Label": {
                "Confidence": 60.73529815673828,
                "Name": "Badge"
            },
            "Timestamp": 9376
        },
        {
            "Label": {
                "Confidence": 60.914302825927734,
                "Name": "Badge"
            },
            "Timestamp": 9576
        },
        {
            "Label": {
                "Confidence": 50.74089813232422,
                "Name": "Badge"
            },
            "Timestamp": 9776
        }
    ],
    "NextToken": "CmUTsWamUKmnzPvN+FB0REjh+xv6+iY3H0IXrHCn3yMSW1zuOcWyLKob2JjvfPjznalo3m/
MGw==",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 67301,
        "FileExtension": "mp4",
        "Format": "QuickTime / MOV",
        "FrameHeight": 1080,
        "FrameRate": 29.970029830932617,
        "FrameWidth": 1920
    }
}
```


Configuring Amazon Rekognition Video

To use the Amazon Rekognition Video API with stored videos, you have to configure the IAM user and an IAM service role to access your Amazon SNS topics. The following procedures show you how to configure Amazon Rekognition Video for use with the examples in this section. To do this, you:

- Create an IAM user, or give an IAM user access to the Amazon Rekognition Video API. In this example, full access to the entire Amazon Rekognition API is given, but you can restrict access as required.
- Install and configure the AWS SDK that you want to use.
- Create an IAM service role that allows Amazon Rekognition Video to publish the completion status of an analysis request to an Amazon SNS topic.
- Give the example code access to the Amazon SQS queue that the Amazon SNS message is retrieved from. Full access is required because the example code deletes the message from the queue.
- Give the example code read access to the Amazon S3 bucket that contains the stored video file.

Note
The examples in this section create a new Amazon SNS topic by using the instructions that give Amazon Rekognition Video access to multiple topics. If you want to use an existing Amazon SNS topic, use Giving Access to an Existing Amazon SNS Topic (p. 68) for step 3.

To configure Amazon Rekognition Video

1. Set up an AWS account to access Amazon Rekognition Video. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 11).

   Ensure the user has at least the following permissions:

   - AmazonSQSFullAccess
   - AmazonRekognitionFullAccess
   - AmazonS3ReadOnlyAccess
2. Install and configure the required AWS SDK. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).
3. Create an IAM service role to give Amazon Rekognition Video access to your Amazon SNS topics. Note the Amazon Resource Name (ARN) of the service role. For more information, see Giving Access to Multiple Amazon SNS Topics (p. 67).
4. Add the following inline policy to the IAM user that you created in step 1:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "MySid",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:Service role ARN from step 3"
        }
    ]
}
```

   Give the inline policy a name of your choosing.
5. You can now run the examples in Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68) and Analyzing a Video with the AWS Command Line Interface (p. 75).

Giving Access to Multiple Amazon SNS Topics

You use an IAM service role to give Amazon Rekognition Video access to Amazon SNS topics that you create. IAM provides the *Rekognition* use case for creating an Amazon Rekognition Video service role.

You can give Amazon 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 Amazon 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. You should see the **AmazonRekognitionServiceRole** permissions policy listed. **AmazonRekognitionServiceRole**

gives Amazon Rekognition Video access to Amazon SNS topics that are prefixed with *AmazonRekognition*.

3. Give the service role a name of your choosing.

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

Giving Access to an Existing Amazon SNS Topic

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

To give Amazon Rekognition Video access to an existing 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.

```
{
    "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 Stored in an Amazon S3 Bucket with Java or Python (SDK)

This procedure shows you how to detect labels in a video by using Amazon 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 Calling Amazon Rekognition Video Operations (p. 61). 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 Amazon 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. 349).
2. Get the completion status from the Amazon SQS queue. The example 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. For simplicity, the example deletes jobs that don't match. Consider adding them to an Amazon SQS dead-letter queue for further investigation.
3. Get and display the video analysis results by calling GetLabelDetection (p. 296).

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. 32).

To Detect Labels in a Video

1. Configure user access to Amazon Rekognition Video and configure Amazon Rekognition Video access to Amazon SNS. For more information, see Configuring Amazon Rekognition Video (p. 66).
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). Ensure the topic is in the same region as the AWS endpoint that you are using.
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. Give permission to the Amazon SNS topic to send messages to the Amazon SQS queue.
6. 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*.
7. 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 the values of `bucket` and `video` with the bucket and video file name that you specified in step 6.

   Java

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package com.amazonaws.samples;
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.FaceSearchSortBy;
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.LabelDetection;
import com.amazonaws.services.rekognition.model.LabelDetectionSortBy;
import com.amazonaws.services.rekognition.model.NotificationChannel;
import com.amazonaws.services.rekognition.model.PersonDetection;
import com.amazonaws.services.rekognition.model.PersonMatch;
import com.amazonaws.services.rekognition.model.PersonTrackingSortBy;
import com.amazonaws.services.rekognition.model.S3Object;
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.rekognition.model.Video;
import com.amazonaws.services.rekognition.model.VideoMetadata;
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.util.*;

public class VideoDetect {

    private static String bucket = "";
    private static String video = "";
    private static String queueUrl =  "";
    private static String topicArn="";
    private static String roleArn="";
    private static AmazonSQS sqs = null;
    private static AmazonRekognition rek = null;

    private static NotificationChannel channel= new NotificationChannel()
            .withSNSTopicArn(topicArn)
            .withRoleArn(roleArn);


    private static String startJobId = null;
```

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


       sqs = AmazonSQSClientBuilder.defaultClient();
       rek = AmazonRekognitionClientBuilder.defaultClient();

       //===================================================
       StartLabels(bucket, video);
       //===================================================
       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{
           messages = sqs.receiveMessage(queueUrl).getMessages();
           if (dotLine++<20){
               System.out.print(".");
           }else{
               System.out.println();
               dotLine=0;
           }

           if (!messages.isEmpty()) {
               //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);
                       //Delete unknown message. Consider moving message to dead
letter queue
                       sqs.deleteMessage(queueUrl,message.getReceiptHandle());
                   }
```

```
                }
            }
        } 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();
                System.out.print("Millisecond: " + Long.toString(seconds) + " ");
                System.out.println("\t" + detectedLabel.getLabel().getName() +
                        "     \t" +
```

```
                        detectedLabel.getLabel().getConfidence().toString());
                System.out.println();
            }
        } while (labelDetectionResult !=null &&
 labelDetectionResult.getNextToken() != null);

    }
}
```

Python

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
import json
import sys


class VideoDetect:
    jobId = ''
    rek = boto3.client('rekognition')
    queueUrl = ''
    roleArn = ''
    topicArn = ''
    bucket = ''
    video = ''

    def main(self):

        jobFound = False
        sqs = boto3.client('sqs')


        #============================================
        response = self.rek.start_label_detection(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
                                                  NotificationChannel={'RoleArn':
 self.roleArn, 'SNSTopicArn': self.topicArn})
        #============================================
        print('Start Job Id: ' + response['JobId'])
        dotLine=0
        while jobFound == False:
            sqsResponse = sqs.receive_message(QueueUrl=self.queueUrl,
 MessageAttributeNames=['ALL'],
                                              MaxNumberOfMessages=10)

            if sqsResponse:

                if 'Messages' not in sqsResponse:
                    if dotLine<20:
                        print('.', end='')
                        dotLine=dotLine+1
                    else:
                        print()
                        dotLine=0
                    sys.stdout.flush()
                    continue

                for message in sqsResponse['Messages']:
                    notification = json.loads(message['Body'])
                    rekMessage = json.loads(notification['Message'])
                    print(rekMessage['JobId'])
```

```
                    print(rekMessage['Status'])
                    if str(rekMessage['JobId']) == response['JobId']:
                        print('Matching Job Found:' + rekMessage['JobId'])
                        jobFound = True
                        #============================================
                        self.GetResultsLabels(rekMessage['JobId'])
                        #============================================

                        sqs.delete_message(QueueUrl=self.queueUrl,
                                           ReceiptHandle=message['ReceiptHandle'])
                    else:
                        print("Job didn't match:" +
                              str(rekMessage['JobId']) + ' : ' +
str(response['JobId']))
                    # Delete the unknown message. Consider sending to dead letter
queue
                    sqs.delete_message(QueueUrl=self.queueUrl,
                                       ReceiptHandle=message['ReceiptHandle'])

        print('done')


    def GetResultsLabels(self, jobId):
        maxResults = 10
        paginationToken = ''
        finished = False

        while finished == False:
            response = self.rek.get_label_detection(JobId=jobId,
                                                    MaxResults=maxResults,
                                                    NextToken=paginationToken,
                                                    SortBy='TIMESTAMP')

            print(response['VideoMetadata']['Codec'])
            print(str(response['VideoMetadata']['DurationMillis']))
            print(response['VideoMetadata']['Format'])
            print(response['VideoMetadata']['FrameRate'])

            for labelDetection in response['Labels']:
                print(labelDetection['Label']['Name'])
                print(labelDetection['Label']['Confidence'])
                print(str(labelDetection['Timestamp']))

            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True




if __name__ == "__main__":

    analyzer=VideoDetect()
    analyzer.main()
```

8. 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. For more information, see Detecting Labels in a Video (p. 111).

Analyzing a Video with the AWS Command Line Interface

You can use the AWS Command Line Interface (AWS CLI) to call Amazon Rekognition Video operations. The design pattern is the same as using the Amazon Rekognition Video API with the AWS SDK for Java or other AWS SDKs. For more information, see Amazon Rekognition Video API Overview (p. 59). 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. 83). 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 Java or Python (SDK) (p. 68).

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. 32).

To configure Amazon Rekognition Video and upload a video

1. Configure user access to Amazon Rekognition Video and configure Amazon Rekognition Video access to Amazon SNS. For more information, see Configuring Amazon Rekognition Video (p. 66).
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. Give permission to the Amazon SNS topic to send messages to the Amazon SQS queue.
6. 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.

```
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 that you specified in step 6.
- 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 Amazon Rekognition Video operations.

2. Note the value of `JobId` in the response. The response looks similar to the following JSON example.

```
{
    "JobId": "547089ce5b9a8a0e7831afa655f42e5d7b5c838553f1a584bf350ennnnnnnnnn"
}
```

3. Write code to poll the Amazon SQS queue for the completion status JSON (by using receive-message).
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.

```
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 Amazon Rekognition Video operations.

The results look similar to the following example JSON:

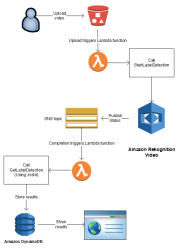
```
{
    "Labels": [
        {
            "Timestamp": 0,
            "Label": {
                "Confidence": 99.03720092773438,
                "Name": "Speech"
            }
        },
        {
            "Timestamp": 0,
            "Label": {
                "Confidence": 71.6698989868164,
 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

```
        },
        {
            "Timestamp": 0,
            "Label": {
                "Confidence": 71.6698989868164,
                "Name": "Squash"
            }
        },
        {
            "Timestamp": 0,
            "Label": {
                "Confidence": 71.6698989868164,
                "Name": "Vegetable"
            }
        }, .......
```

Tutorial: Creating an Amazon Rekognition Lambda Function

This tutorial shows how to get the results of a video analysis operation for label detection by using a Java Lambda function.

You can use Lambda functions with Amazon Rekognition Video operations. For example, the following diagram shows a website that uses a Lambda function to automatically start analysis of a video when it's uploaded to an Amazon S3 bucket. When the Lambda function is triggered, it calls the section called "StartLabelDetection" (p. 349) to start detecting labels in the uploaded video. A second Lambda function is triggered when the analysis completion status is sent to the registered Amazon SNS topic. The second Lambda function calls the section called "GetLabelDetection" (p. 296) to get the analysis results. The results are then stored in a database in preparation for displaying on a webpage.



In this tutorial, the Lambda function is triggered when Amazon Rekognition Video sends the completion status for the video analysis to the registered Amazon SNS topic. It then collects video analysis results by calling the section called "GetLabelDetection" (p. 296). For demonstration purposes, this tutorial writes label detection results to a CloudWatch log. In your application's Lambda function, you should store the analysis results for later use. For example, you can use Amazon DynamoDB to save the analysis results. For more information, see Working with DynamoDB.

The following procedures show you how to:

- Create the Amazon SNS topic and set up permissions.
- Create the Lambda function by using the AWS Management Console and subscribe it to the Amazon SNS topic.
- Configure the Lambda function by using the AWS Management Console.
- Add sample code to an AWS Toolkit for Eclipse project and upload it to the Lambda function.
- Test the Lambda function by using the AWS CLI.

Prerequisites

This tutorial assumes that you're familiar with the AWS Toolkit for Eclipse. For more information, see AWS Toolkit for Eclipse.

Create the SNS Topic

The completion status of an Amazon Rekognition Video video analysis operation is sent to an Amazon SNS topic. This procedure creates the Amazon SNS topic and the IAM service role that gives Amazon Rekognition Video access to your Amazon SNS topics. For more information, see Calling Amazon Rekognition Video Operations (p. 61).

To create an Amazon SNS topic

1. If you haven't already, create an IAM service role to give Amazon Rekognition Video access to your Amazon SNS topics. Note the Amazon Resource Name (ARN). For more information, see Giving Access to Multiple Amazon SNS Topics (p. 67).
2. Create an Amazon SNS topic by using the Amazon SNS console. Prepend the topic name with *AmazonRekognition*. Note the topic ARN.

Create the Lambda Function

You create the Lambda function by using the AWS Management Console. Then you use an AWS Toolkit for Eclipse project to upload the Lambda function package to AWS Lambda. It's also possible to create the Lambda function with the AWS Toolkit for Eclipse. For more information, see Tutorial: How to Create, Upload, and Invoke an AWS Lambda Function.

To create the Lambda function

1. Sign in to the AWS Management Console, and open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose **Create function**.
3. Choose **Author from scratch**.
4. In **Name***, type a name for your function.
5. In **Runtime***, choose **Java 8**.
6. In **Role***, choose **Create a custom role**. A new tab is displayed for the custom role.
7. On the new tab, do the following:
    1. Choose **IAM Role**, and then choose **Create a new IAM Role**.
    2. In **Role Name**, type a name for the new custom role.
    3. Choose **Allow** to create the new role. The custom role tab is closed, and you're returned to the Lambda creation page.
8. In **Role***, choose **Choose an existing role**.
9. In **Existing role***, choose the role that you created in step 7.
10. Choose **Create function**.

Configure the Lambda Function

After you create the Lambda function, you configure it to be triggered by the Amazon SNS topic that you create in Create the SNS Topic (p. 78). You also adjust the memory requirements and timeout period for the Lambda function.

To configure the Lambda function

1. In **Function Code**, type `com.amazonaws.lambda.demo.JobCompletionHandler` for **Handler**.
2. In **Basic settings**, choose **1024** for **Memory**.
3. In **Basic settings**, choose **10** seconds for **Timeout**.
4. In **Designer**, choose **SNS** from **Add Triggers**.
5. In **Configure triggers**, choose the Amazon SNS topic that you created in Create the SNS Topic (p. 78).
6. Choose **Enable trigger**.
7. To add the trigger, choose **Add**.
8. Choose **Save**.

Configure the IAM Lambda Role

To call Amazon Rekognition Video operations, you add the *AmazonRekognitionFullAccess* AWS managed policy to the IAM Lambda role. Start operations, such as the section called "StartLabelDetection" (p. 349), also require pass role permissions for the IAM service role that Amazon Rekognition Video uses to access the Amazon SNS topic.

To configure the role

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. In the list, choose the name of the custom role that you created in Create the Lambda Function (p. 78).
4. Choose the **Permissions** tab.
5. Choose **Attach policy**.
6. Choose *AmazonRekognitionFullAccess* from the list of policies.
7. Choose **Attach policy**.
8. Again, choose the custom role.
9. Scroll to the bottom of the page, and choose **Add inline policy**.
10. Choose the **JSON** tab.
11. Replace the existing policy with the following policy. Replace `servicerole` with the IAM service role that you created in Create the SNS Topic (p. 78).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "mysid",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:servicerole"
        }
    ]
}
```

12. Choose **Review policy**.
13. In **Name***, type a name for the policy.
14. Choose **Create policy**.

Create the AWS Toolkit for Eclipse Lambda Project

When the Lambda function is triggered, the following code gets the completion status from the Amazon SNS topic, and calls the section called "GetLabelDetection" (p. 296) to get the analysis results. A count of labels detected, and a list of labels detected is written to a CloudWatch log. Your Lambda function should store the video analysis results for later use.

To create the AWS Toolkit for Eclipse Lambda project

1. Create an AWS Toolkit for Eclipse AWS Lambda project.
   - For **Project name:**, type a project name of your choosing.
   - For **Input type:**, choose **SNS Event**.
   - Leave the other fields unchanged.
2. In the **Eclipse Project** explorer, open the generated Lambda handler method and replace the contents with the following:

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package com.amazonaws.lambda.demo;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SNSEvent;
import java.util.List;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.GetLabelDetectionRequest;
import com.amazonaws.services.rekognition.model.GetLabelDetectionResult;
import com.amazonaws.services.rekognition.model.LabelDetection;
import com.amazonaws.services.rekognition.model.LabelDetectionSortBy;
import com.amazonaws.services.rekognition.model.VideoMetadata;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;



public class JobCompletionHandler implements RequestHandler<SNSEvent, String> {

   @Override
   public String handleRequest(SNSEvent event, Context context) {

      String message = event.getRecords().get(0).getSNS().getMessage();
      LambdaLogger logger = context.getLogger();

      // Parse SNS event for analysis results. Log results
      try {
         ObjectMapper operationResultMapper = new ObjectMapper();
         JsonNode jsonResultTree = operationResultMapper.readTree(message);
         logger.log("Rekognition Video Operation:========================");
         logger.log("Job id: " + jsonResultTree.get("JobId"));
         logger.log("Status : " + jsonResultTree.get("Status"));
         logger.log("Job tag : " + jsonResultTree.get("JobTag"));
         logger.log("Operation : " + jsonResultTree.get("API"));

         if (jsonResultTree.get("API").asText().equals("StartLabelDetection")) {

            if (jsonResultTree.get("Status").asText().equals("SUCCEEDED")){
```

```
               GetResultsLabels(jsonResultTree.get("JobId").asText(), context);
            }
            else{
               String errorMessage = "Video analysis failed for job "
                     + jsonResultTree.get("JobId")
                     + "State " + jsonResultTree.get("Status");
               throw new Exception(errorMessage);
            }

         } else
            logger.log("Operation not StartLabelDetection");

      } catch (Exception e) {
         logger.log("Error: " + e.getMessage());
         throw new RuntimeException (e);


      }

      return message;
   }

   void GetResultsLabels(String startJobId, Context context) throws Exception {

      LambdaLogger logger = context.getLogger();

      AmazonRekognition rek =
AmazonRekognitionClientBuilder.standard().withRegion(Regions.US_EAST_1).build();

      int maxResults = 1000;
      String paginationToken = null;
      GetLabelDetectionResult labelDetectionResult = null;
      String labels = "";
      Integer labelsCount = 0;
      String label = "";
      String currentLabel = "";

      //Get label detection results and log them.
      do {

         GetLabelDetectionRequest labelDetectionRequest = new
GetLabelDetectionRequest().withJobId(startJobId)

.withSortBy(LabelDetectionSortBy.NAME).withMaxResults(maxResults).withNextToken(paginationToken);

         labelDetectionResult = rek.getLabelDetection(labelDetectionRequest);

         paginationToken = labelDetectionResult.getNextToken();
         VideoMetadata videoMetaData = labelDetectionResult.getVideoMetadata();

         // Add labels to log
         List<LabelDetection> detectedLabels = labelDetectionResult.getLabels();

         for (LabelDetection detectedLabel : detectedLabels) {
            label = detectedLabel.getLabel().getName();
            if (label.equals(currentLabel)) {
               continue;
            }
            labels = labels + label + " / ";
            currentLabel = label;
            labelsCount++;

         }
      } while (labelDetectionResult != null && labelDetectionResult.getNextToken() !=
null);
```

```
        logger.log("Total number of labels : " + labelsCount);
        logger.log("labels : " + labels);

    }


}
```

3. The Rekognition namespaces aren't resolved. To correct this:
   - Pause your mouse over the underlined portion of the line `import com.amazonaws.services.rekognition.AmazonRekognition;`.
   - Choose **Fix project set up...** .
   - Choose the latest version of the Amazon Rekognition archive.
   - Choose **OK** to add the archive to the project.
4. Right-click in your Eclipse code window, choose **AWS Lambda**, and then choose **Upload function to AWS Lambda**.
5. On the **Select Target Lambda Function** page, choose the AWS Region to use.
6. Choose **Choose an existing lambda function**, and select the Lambda function that you created in Create the Lambda Function (p. 78).
7. Choose **Next**.
8. On the **Function Configuration** page, select the IAM role that you created in Create the Lambda Function (p. 78).
9. Choose **Finish**, and the Lambda function is uploaded to AWS.

Test the Lambda Function

Use the following AWS CLI command to test the Lambda function by starting the label detection analysis of a video. After analysis is finished, the Lambda function is triggered. Confirm that the analysis succeeded by checking the CloudWatch Logs logs.

To test the Lambda function

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

```
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
```

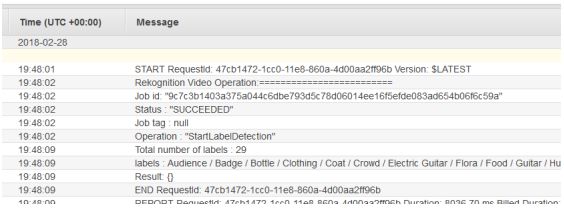
   Update the following values:
   - Change `bucketname` and `videofile` to the Amazon S3 bucket name and file name of the video that you want to detect labels in.
   - Change `Endpoint` and `us-east-1` to the AWS endpoint and region that you're using.

- Change `TopicARN` to the ARN of the Amazon SNS topic that you created in Create the SNS Topic (p. 78).
- Change `RoleARN` to the ARN of the IAM role that you created in Create the SNS Topic (p. 78).
- Change `RekognitionUser` to an AWS account that has permissions to call Amazon Rekognition Video operations.

3. Note the value of `JobId` in the response. The response looks similar to the following JSON example.

```
{
    "JobId": "547089ce5b9a8a0e7831afa655f42e5d7b5c838553f1a584bf350ennnnnnnnnn"
}
```

4. Open the https://console.aws.amazon.com/cloudwatch/ console.
5. When the analysis completes, a log entry for the Lambda function appears in the **Log Group**.
6. Choose the Lambda function to see the log streams.
7. Choose the latest log stream to see the log entries made by the Lambda function. If the operation succeeded, it looks similar to the following:



The value of **Jod id** should match the value of `JobId` that you noted in step 3.

Reference: Video Analysis Results Notification

Amazon Rekognition publishes the results of an Amazon 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 Amazon Rekognition Video Operations" (p. 61). For an example, see Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68).

The payload is in the following JSON format:

```
{
  "JobId": "String",
  "Status": "String",
  "API": "String",
  "JobTag": "String",
  "Timestamp": Number,
  "Video": {
    "S3ObjectName": "String",
    "S3Bucket": "String"
  }
}
```

Name	Description
JobId	The job identifier. Matches a job identifier that's returned from a `Start` operation, such as StartPersonTracking (p. 353).

Name	Description
Status	The status of the job. Valid values are SUCCEEDED, FAILED, or ERROR.
API	The Amazon Rekognition Video operation used to analyze the input video.
JobTag	Identifier for the job. You specify `JobTag` in a call to Start operation, such as StartLabelDetection (p. 349).
Timestamp	The Unix time stamp for when the job finished.
Video	Details about the video that was processed. Includes the file name and the Amazon S3 bucket that the file is stored in.

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"
  }
}
```

Troubleshooting Amazon Rekognition Video

The following covers troubleshooting information for working with Amazon Rekognition Video and stored videos.

I never receive the completion status that's sent to the Amazon SNS topic

Amazon Rekognition Video publishes status information to an Amazon SNS topic when video analysis completes. Typically, you get the completion status message by subscribing to the topic with an Amazon SQS queue or Lambda function. To help your investigation, subscribe to the Amazon SNS topic by email so you receive the messages that are sent to your Amazon SNS topic in your email inbox. For more information, see Subscribe to a Topic.

If you don't receive the message in your application, consider the following:

- Verify that the analysis has completed. Check the `JobStatus` value in the Get operation response (`GetLabelDetection`, for example). If the value is `IN_PROGRESS`, the analysis isn't complete, and the completion status hasn't yet been published to the Amazon SNS topic.
- Verify that you have an IAM service role that gives Amazon Rekognition Video permissions to publish to your Amazon SNS topics. For more information, see Configuring Amazon Rekognition Video (p. 66).

- Confirm that the IAM service role that you're using can publish to the Amazon SNS topic by using role credentials. Use the following steps:
  - Get the user Amazon Resource Name (ARN):

```
aws sts get-caller-identity --profile RekognitionUser
```

  - Add the user ARN to the role trust relationship by using the AWS Management Console. For example:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "rekognition.amazonaws.com",
        "AWS": "arn:User ARN"
      },
      "Action": "sts:AssumeRole",
      "Condition": {}
    }
  ]
}
```

  - Assume the role: `aws sts assume-role --role-arn arn:Role ARN --role-session-name SessionName --profile RekognitionUser`
  - Publish to the Amazon SNS topic: `aws sns publish --topic-arn arn:Topic ARN --message "Hello World!" --region us-east-1 --profile RekognitionUser`

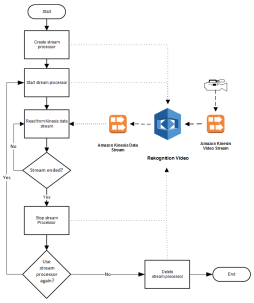
  If the AWS CLI command works, you receive the message (in your email inbox, if you've subscribed to the topic by email). If you don't receive the message:
  - Check that you've configured Amazon Rekognition Video. For more information, see Configuring Amazon Rekognition Video (p. 66).
  - Check the other tips for this troubleshooting question.
- Check that you're using the correct Amazon SNS topic:
  - If you use an IAM service role to give Amazon Rekognition Video access to a single Amazon SNS topic, check that you've given permissions to the correct Amazon SNS topic. For more information, see Giving Access to an Existing Amazon SNS Topic (p. 68).
  - If you use an IAM service role to give Amazon Rekognition Video access to multiple SNS topics, verify that you're using the correct topic and that the topic name is prepended with *AmazonRekognition*. For more information, see Giving Access to Multiple Amazon SNS Topics (p. 67).
  - If you use an AWS Lambda function, confirm that your Lambda function is subscribed to the correct Amazon SNS topic. For more information, see Invoking Lambda Functions Using Amazon SNS Notifications.
- If you subscribe an Amazon SQS queue to your Amazon SNS topic, confirm that your Amazon SNS topic has permissions to send messages to the Amazon SQS queue. For more information, see Give permission to the Amazon SNS topic to send messages to the Amazon SQS queue.

Working with Streaming Videos

You can use Amazon Rekognition Video to detect and recognize faces in streaming video. A typical use case is when you want to detect a known face in a video stream. Amazon Rekognition Video uses Amazon Kinesis Video Streams to receive and process a video stream. The analysis results are output from Amazon Rekognition Video to a Kinesis data stream and then read by your client application. Amazon Rekognition Video provides a stream processor (CreateStreamProcessor (p. 244)) that you can use to start and manage the analysis of streaming video.

> **Note**
> The Amazon Rekognition Video streaming API is not available in either the US East (Ohio) region or the Asia Pacific (Sydney) region.

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



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

- A Kinesis video stream for sending streaming video to Amazon Rekognition Video. For more information, see Kinesis video stream.
- An Amazon Rekognition Video stream processor to manage the analysis of the streaming video. For more information, see Starting Streaming Video Analysis (p. 88).
- A Kinesis data stream consumer to read the analysis results that Amazon 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 Amazon Rekognition Video, and consumes the analysis results. For more information, see Recognizing Faces in a Streaming Video (p. 86).

Topics

- Recognizing Faces in a Streaming Video (p. 86)
- Giving Amazon Rekognition Video Access to Your Kinesis Streams (p. 87)
- Starting Streaming Video Analysis (p. 88)
- Reading Streaming Video Analysis Results (p. 95)
- Reference: Kinesis Face Recognition Record (p. 97)

Recognizing Faces in a Streaming Video

Amazon 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. 137). 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. 32).

To recognize faces in a video stream (AWS SDK)

1. If you haven't already, create an IAM service role to give Amazon Rekognition Video access to your Kinesis video streams and your Kinesis data streams. Note the ARN. For more information, see Giving Access to Your Kinesis Video Streams and Kinesis Data Streams (p. 87).
2. Create a collection (p. 139) and note the collection identifier you used.
3. Index the faces (p. 151) 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. 90). Pass the following as parameters to the section called "CreateStreamProcessor" (p. 244): 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 Amazon Rekognition Video (p. 95).

Giving Amazon Rekognition Video Access to Your Kinesis Streams

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

To give Amazon 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.

```
{
    "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 an Amazon Rekognition Video stream processor and streaming video into Amazon Rekognition Video. An Amazon Rekognition Video stream processor allows you to start, stop, and manage stream processors. You create a stream processor by calling CreateStreamProcessor (p. 244). 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

Starts processing a stream processor. You create a stream processor by calling CreateStreamProcessor (p. 244). 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. 356)

The name of the stream processor to start processing.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

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

(p. 356) operation. To get status information for a stream processor, call DescribeStreamProcessor (p. 257). Other operations you can call are StopStreamProcessor (p. 358) to stop a stream processor, and DeleteStreamProcessor (p. 252) to delete a stream processor. To get a list of stream processors in your account, call ListStreamProcessors (p. 319).

After the stream processor starts running, you stream the video into Amazon Rekognition Video through the Kinesis video stream that you specified in `CreateStreamProcessor`. Use the Kinesis Video Streams SDK PutMedia operation to deliver video into the Kinesis video stream. For an example, see PutMedia API Example.

For information about how your application can consume Amazon Rekognition Video analysis results, see Reading Streaming Video Analysis Results (p. 95).

Creating the Amazon Rekognition Video Stream Processor

Before you can analyze a streaming video, you create an Amazon Rekognition Video stream processor (CreateStreamProcessor (p. 244)). 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:nnnnnnnnnnnn:stream/inputVideo"
		}
	},
	"Output": {
		"KinesisDataStream": {
			"Arn": "arn:aws:kinesis:us-east-1:nnnnnnnnnnnn: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:nnnnnnnnnnnn:streamprocessor/
streamProcessorForCam”
}
```

Starting the Amazon Rekognition Video Stream Processor

You start analyzing streaming video by calling StartStreamProcessor (p. 356) 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 (p. 244) and StartStreamProcessor (p. 356). The example includes a stream processor manager class (StreamManager) that provides methods to call stream processor operations. The starter class (Starter) creates a StreamManager object and calls various operations.

To configure the example:

1. Set the values of the Starter class member fields to your desired values.
2. In the Starter class function `main`, uncomment the desired function call.

Starter Class

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

// Starter class. Use to create a StreamManager class
// and call stream processor operations.
package com.amazonaws.samples;
import com.amazonaws.samples.*;

public class Starter {

 public static void main(String[] args) {


     String streamProcessorName="Stream Processor Name";
     String kinesisVideoStreamArn="Kinesis Video Stream Arn";
     String kinesisDataStreamArn="Kinesis Data Stream Arn";
     String roleArn="Role Arn";
     String collectionId="Collection ID";
     Float matchThreshold=50F;

  try {
   StreamManager sm= new StreamManager(streamProcessorName,
     kinesisVideoStreamArn,
     kinesisDataStreamArn,
     roleArn,
     collectionId,
     matchThreshold);
   //sm.createStreamProcessor();
   //sm.startStreamProcessor();
   //sm.deleteStreamProcessor();
   //sm.deleteStreamProcessor();
   //sm.stopStreamProcessor();
   //sm.listStreamProcessors();
   //sm.describeStreamProcessor();
  }
  catch(Exception e){
   System.out.println(e.getMessage());
  }
 }
}
```

StreamManager Class

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

// Stream manager class. Provides methods for calling
// Stream Processor operations.
package com.amazonaws.samples;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
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 StreamManager {

    private String streamProcessorName;
    private String kinesisVideoStreamArn;
    private String kinesisDataStreamArn;
    private String roleArn;
    private String collectionId;
    private float matchThreshold;

    private AmazonRekognition rekognitionClient;


    public StreamManager(String spName,
      String kvStreamArn,
      String kdStreamArn,
      String iamRoleArn,
      String collId,
      Float threshold){
     streamProcessorName=spName;
     kinesisVideoStreamArn=kvStreamArn;
     kinesisDataStreamArn=kdStreamArn;
     roleArn=iamRoleArn;
     collectionId=collId;
     matchThreshold=threshold;
     rekognitionClient=AmazonRekognitionClientBuilder.defaultClient();

    }

    public void createStreamProcessor() {
     //Setup input parameters
        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);

        //Create the stream processor
        CreateStreamProcessorResult createStreamProcessorResult =
 rekognitionClient.createStreamProcessor(
                new
 CreateStreamProcessorRequest().withInput(streamProcessorInput).withOutput(streamProcessorOutput)

 .withSettings(streamProcessorSettings).withRoleArn(roleArn).withName(streamProcessorName));
```

```
        //Display result
        System.out.println("Stream Processor " + streamProcessorName + " created.");
        System.out.println("StreamProcessorArn - " +
createStreamProcessorResult.getStreamProcessorArn());
    }

    public void startStreamProcessor() {
        StartStreamProcessorResult startStreamProcessorResult =
                rekognitionClient.startStreamProcessor(new
StartStreamProcessorRequest().withName(streamProcessorName));
        System.out.println("Stream Processor " + streamProcessorName + " started.");
    }

    public void stopStreamProcessor() {
        StopStreamProcessorResult stopStreamProcessorResult =
                rekognitionClient.stopStreamProcessor(new
StopStreamProcessorRequest().withName(streamProcessorName));
        System.out.println("Stream Processor " + streamProcessorName + " stopped.");
    }

    public void deleteStreamProcessor() {
        DeleteStreamProcessorResult deleteStreamProcessorResult = rekognitionClient
                .deleteStreamProcessor(new
DeleteStreamProcessorRequest().withName(streamProcessorName));
        System.out.println("Stream Processor " + streamProcessorName + " deleted.");
    }

    public void describeStreamProcessor() {
        DescribeStreamProcessorResult describeStreamProcessorResult = rekognitionClient
                .describeStreamProcessor(new
DescribeStreamProcessorRequest().withName(streamProcessorName));

        //Display various stream processor attributes.
        System.out.println("Arn - " +
describeStreamProcessorResult.getStreamProcessorArn());
        System.out.println("Input kinesisVideo stream - "
                +
describeStreamProcessorResult.getInput().getKinesisVideoStream().getArn());
        System.out.println("Output kinesisData stream - "
                +
describeStreamProcessorResult.getOutput().getKinesisDataStream().getArn());
        System.out.println("RoleArn - " + describeStreamProcessorResult.getRoleArn());
        System.out.println(
                "CollectionId - " +
describeStreamProcessorResult.getSettings().getFaceSearch().getCollectionId());
        System.out.println("Status - " + describeStreamProcessorResult.getStatus());
        System.out.println("Status message - " +
describeStreamProcessorResult.getStatusMessage());
        System.out.println("Creation timestamp - " +
describeStreamProcessorResult.getCreationTimestamp());
        System.out.println("Last update timestamp - " +
describeStreamProcessorResult.getLastUpdateTimestamp());
    }

    public void listStreamProcessors() {
        ListStreamProcessorsResult listStreamProcessorsResult =
                rekognitionClient.listStreamProcessors(new
ListStreamProcessorsRequest().withMaxResults(100));

        //List all stream processors (and state) returned from Rekognition
        for (StreamProcessor streamProcessor :
listStreamProcessorsResult.getStreamProcessors()) {
            System.out.println("StreamProcessor name - " + streamProcessor.getName());
            System.out.println("Status - " + streamProcessor.getStatus());
        }
    }
```

```
}
```

Streaming Video into Amazon Rekognition Video

To stream video into Amazon 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 Amazon Rekognition Video consumes. Each video data fragment is typically 2–10 seconds in length and contains a self-contained sequence of video frames. Amazon 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. Amazon Rekognition Video places a JSON frame record for each analyzed frame into the Kinesis output stream. Amazon Rekognition Video doesn't analyze every frame that's passed to it through 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. 97).

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

```
{
  "InputInformation": {
    "KinesisVideo": {
      "StreamArn": "arn:aws:kinesisvideo:us-west-2:nnnnnnnnnnnn:stream/stream-name",
      "FragmentNumber": "91343852333289682796718532614445757584843717598",
      "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,
            "ImageId": "70e09693-2114-57e1-807c-50b6d61fa4dc",
            "ExternalImageId": "matchedImage.jpeg"
          }
        }
      ]
    }
  ]
}
```


In the JSON example, note the following:

- **InputInformation** – Information about the Kinesis video stream that's used to stream video into Amazon Rekognition Video. For more information, see InputInformation (p. 99).
- **StreamProcessorInformation** – Status information for the Amazon Rekognition Video stream processor. The only possible value for the `Status` field is RUNNING. For more information, see StreamProcessorInformation (p. 100).
- **FaceSearchResponse** – Contains information about faces in the streaming video that match faces in the input collection. FaceSearchResponse (p. 100) contains a DetectedFace (p. 100) object, which is a face that was detected in in the analyzed video frame. For each detected face, the array

`MatchedFaces` contains an array of matching face objects (MatchedFace (p. 101)) found in the input collection, along with a similarity score.

Mapping the Kinesis Video Stream to the Kinesis Data Stream

You might want to map the Kinesis video stream frames to the analyzed frames that are sent to the Kinesis data stream. For example, during the display of a streaming video, you might want to display boxes around the faces of recognized people. The bounding box coordinates are sent as part of the Kinesis Face Recognition Record to the Kinesis data stream. To display the bounding box correctly, you need to map the time information that's sent with the Kinesis Face Recognition Record with the corresponding frames in the source Kinesis video stream.

The technique that you use to map the Kinesis video stream to the Kinesis data stream depends on if you're streaming live media (such as a live streaming video), or if you're streaming archived media (such as a stored video).

Mapping When You're Streaming Live Media

To map a Kinesis video stream frame to a Kinesis data stream frame

1. Set the input parameter `FragmentTimeCodeType` of the PutMedia operation to `RELATIVE`.
2. Call `PutMedia` to deliver live media into the Kinesis video stream.
3. When you receive a Kinesis Face Recognition Record from the Kinesis data stream, store the values of `ProducerTimestamp` and `FrameOffsetInSeconds` from the KinesisVideo (p. 99) field.
4. Calculate the time stamp that corresponds to the Kinesis video stream frame by adding the `ProducerTimestamp` and `FrameOffsetInSeconds` field values together.

Mapping When You're Streaming Archived Media

To map a Kinesis video stream frame to a Kinesis data stream frame

1. Call PutMedia to deliver archived media into the Kinesis video stream.
2. When you receive an `Acknowledgement` object from the `PutMedia` operation response, store the `FragmentNumber` field value from the Payload field. `FragmentNumber` is the fragment number for the MKV cluster.
3. When you receive a Kinesis Face Recognition Record from the Kinesis data stream, store the `FrameOffsetInSeconds` field value from the KinesisVideo (p. 99) field.
4. Calculate the mapping by using the `FrameOffsetInSeconds` and `FragmentNumber` values that you stored in steps 2 and 3. `FrameOffsetInSeconds` is the offset into the fragment with the specific `FragmentNumber` that's sent to the Amazon Kinesis data stream. For more information about getting the video frames for a given fragment number, see Amazon Kinesis Video Streams Archived Media.

Reference: Kinesis Face Recognition Record

Amazon Rekognition Video can recognize faces in a streaming video. For each analyzed frame, Amazon Rekognition Video outputs a JSON frame record to a Kinesis data stream. Amazon Rekognition Video doesn't analyze every frame that's passed to it through 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. 86).

```
{
    "InputInformation": {
        "KinesisVideo": {
            "StreamArn": "string",
            "FragmentNumber": "string",
            "ProducerTimestamp": number,
            "ServerTimestamp": number,
            "FrameOffsetInSeconds": number
        }
    },
    "StreamProcessorInformation": {
        "Status": "RUNNING"
    },
    "FaceSearchResponse": [
        {
            "DetectedFace": {
                "BoundingBox": {
                    "Width": number,
                    "Top": number,
                    "Height": number,
                    "Left": number
                },
                "Confidence": 23,
                "Landmarks": [
                    {
                        "Type": "string",
                        "X": number,
                        "Y": number
                    }
                ],
                "Pose": {
                    "Pitch": number,
                    "Roll": number,
                    "Yaw": number
                },
                "Quality": {
                    "Brightness": number,
                    "Sharpness": number
                }
            },
            "MatchedFaces": [
                {
                    "Similarity": number,
                    "Face": {
                        "BoundingBox": {
                            "Width": number,
                            "Top": number,
                            "Height": number,
                            "Left": number
                        },
                        "Confidence": number,
                        "ExternalImageId": "string",
                        "FaceId": "string",
                        "ImageId": "string"
                    }
                }
            ]
        }
```

```
    ]
}
```

JSON Record

The JSON record includes information about a frame that's processed by Amazon 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 Amazon Rekognition Video.

Type: InputInformation (p. 99) object

StreamProcessorInformation

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

Type: StreamProcessorInformation (p. 100) object

FaceSearchResponse

Information about the faces detected in a streaming video frame and the matching faces found in the input collection.

Type: FaceSearchResponse (p. 100) object array

InputInformation

Information about a source video stream that's used by Amazon Rekognition Video. For more information, see Working with Streaming Videos (p. 86).

KinesisVideo

Type: KinesisVideo (p. 99) object

KinesisVideo

Information about the Kinesis video stream that streams the source video into Amazon Rekognition Video. For more information, see Working with Streaming Videos (p. 86).

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 Unix time stamp of the fragment. For more information, see PutMedia.

Type: Number

ServerTimestamp

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

Type: Number

FrameOffsetInSeconds

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

Type: Number

StreamProcessorInformation

Status information about the stream processor.

Status

The current status of the stream processor. The one possible value is RUNNING.

Type: String

FaceSearchResponse

Information about a face detected in a streaming video frame and the faces in a collection that match the detected face. You specify the collection in a call to CreateStreamProcessor (p. 244). For more information, see Working with Streaming Videos (p. 86).

DetectedFace

Face details for a face detected in an analyzed video frame.

Type: DetectedFace (p. 100) object

MatchedFaces

An array of face details for faces in a collection that matches the face detected in `DetectedFace`.

Type: MatchedFace (p. 101) object array

DetectedFace

Information about a face that's detected in a streaming video frame. Matching faces in the input collection are available in MatchedFace (p. 101) object field.

BoundingBox

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.

Type: BoundingBox (p. 363) object

Confidence

The confidence level (1-100) that Amazon Rekognition Video has that the detected face is actually a face. 1 is the lowest confidence, 100 is the highest.

Type: Number

Landmarks

An array of facial landmarks.

Type: Landmark (p. 396) object array

Pose

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

Type: Pose (p. 406) object

Quality

Identifies face image brightness and sharpness.

Type: ImageQuality (p. 389) object

MatchedFace

Information about a face that matches a face detected in an analyzed video frame.

Face

Face match information for a face in the input collection that matches the face in the DetectedFace (p. 100) object.

Type: Face (p. 377) object

Similarity

The level of confidence (1-100) that the faces match. 1 is the lowest confidence, 100 is the highest.

Type: Number

Detecting Objects and Scenes

This section provides information for detecting labels in images and videos with Amazon Rekognition Image and Amazon 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). Amazon Rekognition Video can also detect activities such as a person skiing or riding a bike. Amazon Rekognition Image does not detect activities in images.

Amazon Rekognition Image can return the bounding box for common object labels such as people, cars, furniture, apparel or pets. You can use bounding boxes to find the exact locations of objects in an image, count instances of detected objects, or to measure an object's size using bounding box dimensions.

Amazon Rekognition Image uses a hierarchical taxonomy of ancestor labels to categorize labels. For example, a person walking across a road might be detected as a *Pedestrian*. The parent label for *Pedestrian* is *Person*. Both of these labels are returned in the response. All ancestor labels are returned and a given label contains a list of its parent and other ancestor labels. For example, grandparent and great grandparent labels, if they exist. You can use parent labels to build groups of related labels and to allow querying of similar labels in one or more images. For example, a query for all *Vehicles* might return a car from one image and a motor bike from another.

Amazon Rekognition Image returns the version of the label detection model used to detect labels in the image.

> **Note**
> Amazon Rekognition Video does not return bounding box, hierarchical taxonomy information or the version of label detection model.

For example, in the following image, Amazon Rekognition Image is able to detect the presence of a person, a skateboard, parked cars and other information. Amazon Rekognition Image also returns the bounding box for a detected person, and other detected objects such as cars and wheels. Amazon Rekognition Video and Amazon 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. 103)
- Detecting Labels in a Video (p. 111)

Detecting Labels in an Image

You can use the DetectLabels (p. 265) operation to detect labels in an image. For an example, see Analyzing Images Stored in an Amazon S3 Bucket (p. 38).

The following examples use various AWS SDKs and the AWS CLI to call `DetectLabels`. For information about the `DetectLabels` operation response, see DetectLabels Response (p. 107).

To detect labels in an image

1. If you haven't already:
   a. Create or update an IAM user with `AmazonRekognitionFullAccess` and `AmazonS3ReadOnlyAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).
2. Upload an image that contains one or more objects—such as trees, houses, and boat—to your S3 bucket. The image must be in *.jpg* or *.png* format.

   For instructions, see Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service Console User Guide*.
3. Use the following examples to call the `DetectLabels` operation.

Java

This example displays a list of labels that were detected in the input image. Replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and image that you used in step 2.

```
package com.amazonaws.samples;
import java.util.List;

import com.amazonaws.services.rekognition.model.BoundingBox;
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.Instance;
import com.amazonaws.services.rekognition.model.Label;
import com.amazonaws.services.rekognition.model.Parent;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;

public class DetectLabels {

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

        String photo = "photo";
        String bucket = "bucket";

        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        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 + "\n");
            for (Label label : labels) {
                System.out.println("Label: " + label.getName());
                System.out.println("Confidence: " +
 label.getConfidence().toString() + "\n");

                List<Instance> instances = label.getInstances();
                System.out.println("Instances of " + label.getName());
                if (instances.isEmpty()) {
                    System.out.println("  " + "None");
                } else {
                    for (Instance instance : instances) {
                        System.out.println("  Confidence: " +
 instance.getConfidence().toString());
                        System.out.println("  Bounding box: " +
 instance.getBoundingBox().toString());
                    }
                }
                System.out.println("Parent labels for " + label.getName() + ":");
                List<Parent> parents = label.getParents();
                if (parents.isEmpty()) {
                    System.out.println("  None");
```

```
                } else {
                    for (Parent parent : parents) {
                        System.out.println("   " + parent.getName());
                    }
                }
                System.out.println("--------------------");
                System.out.println();

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

AWS CLI

This example displays the JSON output from the `detect-labels` CLI operation. Replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and image that you used in Step 2.

```
aws rekognition detect-labels \
--image '{"S3Object":{"Bucket":"bucket","Name":"file"}}'
```

Python

This example displays the labels that were detected in the input image. Replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and image that you used in Step 2.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

if __name__ == "__main__":

    bucket='bucket'
    photo='photo.jpg'



    client=boto3.client('rekognition')


    response = client.detect_labels(Image={'S3Object':
{'Bucket':bucket,'Name':photo}},
        MaxLabels=10)

    print('Detected labels for ' + photo)
    print()
    for label in response['Labels']:
        print ("Label: " + label['Name'])
        print ("Confidence: " + str(label['Confidence']))
        print ("Instances:")
        for instance in label['Instances']:
            print ("  Bounding box")
            print ("    Top: " + str(instance['BoundingBox']['Top']))
            print ("    Left: " + str(instance['BoundingBox']['Left']))
            print ("    Width: " +  str(instance['BoundingBox']['Width']))
            print ("    Height: " +  str(instance['BoundingBox']['Height']))
```

```
            print ("  Confidence: " + str(instance['Confidence']))
            print()

        print ("Parents:")
        for parent in label['Parents']:
            print ("   " + parent['Name'])
        print ("----------")
        print ()


```

.NET

This example displays a list of labels that were detected in the input image. Replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and image that you used in Step 2.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DetectLabels
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "bucket";

        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        DetectLabelsRequest detectlabelsRequest = new DetectLabelsRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket
                },
            },
            MaxLabels = 10,
            MinConfidence = 75F
        };

        try
        {
            DetectLabelsResponse detectLabelsResponse =
 rekognitionClient.DetectLabels(detectlabelsRequest);
            Console.WriteLine("Detected labels for " + photo);
            foreach (Label label in detectLabelsResponse.Labels)
                Console.WriteLine("{0}: {1}", label.Name, label.Confidence);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

DetectLabels Operation Request

The input to `DetectLabel` is an image. In this example JSON input, the source image is loaded from an Amazon S3 Bucket. `MaxLabels` is the maximum number of labels to return in the response. `MinConfidence` is the minimum confidence that Amazon Rekognition Image must have in the accuracy of the detected label for it to be returned in the response.

```
{
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "MaxLabels": 10,
    "MinConfidence": 75
}
```

DetectLabels Response

The reponse from `DetectLabels` is an array of labels detected in the image and the level of confidence by which they were detected.

The following is an example response from `DetectLabels`.

The response shows that the operation detected multiple labels including Person, Pedestrian, Vehicle, and Car. Each label has an associated level of confidence. For example, the detection algorithm is 99.99962% confident that the image contains a person.

The response also includes the ancestor labels for a label in the `Parents` array. For example, the label Pedestrian has a parent label named Person.

The response for common object labels includes bounding box information for the location of the label on the input image. For example, the Person label has an instances array containing two bounding boxes. These are the locations of two people detected in the image.

The field `LabelModelVersion` contains the version number of the detection model used by`DetectLabels`.

```
{

    LabelModelVersion": 2.0,
    Labels: [
        {
            Name: Person,
            Confidence: 99.99962,
            Instances: [
                {
                    BoundingBox: {
                        Width: 0.19360729,
                        Height: 0.27422005,
                        Left: 0.43734854,
                        Top: 0.35072067
                    },
                    Confidence: 99.99962
                },
                {
                    BoundingBox: {
                        Width: 0.038017172,
                        Height: 0.06597328,
```

```
                Left: 0.9155802,
                Top: 0.5010884
            },
            Confidence: 91.415535
        }
    ],
    Parents: [

    ]
},
{
    Name: Pedestrian,
    Confidence: 99.48226,
    Instances: [

    ],
    Parents: [
        {
            Name: Person
        }
    ]
},
{
    Name: Tarmac,
    Confidence: 99.385605,
    Instances: [

    ],
    Parents: [

    ]
},
{
    Name: Path,
    Confidence: 98.70715,
    Instances: [

    ],
    Parents: [

    ]
},
{
    Name: Road,
    Confidence: 98.11571,
    Instances: [

    ],
    Parents: [

    ]
},
{
    Name: Urban,
    Confidence: 96.2798,
    Instances: [

    ],
    Parents: [

    ]
},
{
    Name: City,
    Confidence: 96.2798,
    Instances: [
```

```
        ],
        Parents: [
            {
                Name: Urban
            }
        ]
    },
    {
        Name: Vehicle,
        Confidence: 94.803375,
        Instances: [

        ],
        Parents: [

        ]
    },
    {
        Name: Car,
        Confidence: 94.803375,
        Instances: [
            {
                BoundingBox: {
                    Width: 0.101054385,
                    Height: 0.122262456,
                    Left: 0.5743986,
                    Top: 0.53436863
                },
                Confidence: 58.146957
            },
            {
                BoundingBox: {
                    Width: 0.110863954,
                    Height: 0.10271988,
                    Left: 0.10355594,
                    Top: 0.5354845
                },
                Confidence: 70.77716
            },
            {
                BoundingBox: {
                    Width: 0.056389667,
                    Height: 0.171637,
                    Left: 0.942777,
                    Top: 0.52358043
                },
                Confidence: 57.8401
            },
            {
                BoundingBox: {
                    Width: 0.046953984,
                    Height: 0.18796417,
                    Left: 0.0031801604,
                    Top: 0.50652236
                },
                Confidence: 48.009228
            },
            {
                BoundingBox: {
                    Width: 0.060038604,
                    Height: 0.06737709,
                    Left: 0.22409998,
                    Top: 0.54413414
                },
                Confidence: 56.240288
```

```
        },
        {
            BoundingBox: {
                Width: 0.040678814,
                Height: 0.03428703,
                Left: 0.31641594,
                Top: 0.5566274
            },
            Confidence: 53.59637
        },
        {
            BoundingBox: {
                Width: 0.02848697,
                Height: 0.19150497,
                Left: 0.0,
                Top: 0.5107087
            },
            Confidence: 54.186676
        },
        {
            BoundingBox: {
                Width: 0.06254628,
                Height: 0.053911515,
                Left: 0.4608306,
                Top: 0.5573825
            },
            Confidence: 58.661976
        },
        {
            BoundingBox: {
                Width: 0.04341105,
                Height: 0.08935959,
                Left: 0.18293385,
                Top: 0.539492
            },
            Confidence: 52.312466
        },
        {
            BoundingBox: {
                Width: 0.031183116,
                Height: 0.0398999,
                Left: 0.2853088,
                Top: 0.55793667
            },
            Confidence: 51.869164
        },
        {
            BoundingBox: {
                Width: 0.03111379,
                Height: 0.056484755,
                Left: 0.2580395,
                Top: 0.550482
            },
            Confidence: 45.591652
        },
        {
            BoundingBox: {
                Width: 0.10616336,
                Height: 0.1852818,
                Left: 0.0037978808,
                Top: 0.5039217
            },
            Confidence: 94.34959
        },
        {
            BoundingBox: {
```

```
                        Width: 0.14233612,
                        Height: 0.15528248,
                        Left: 0.6494813,
                        Top: 0.5333096
                    },
                    Confidence: 88.45155
                },
                {
                    BoundingBox: {
                        Width: 0.08586375,
                        Height: 0.0855043,
                        Left: 0.5128013,
                        Top: 0.5438793
                    },
                    Confidence: 45.12628
                },
                {
                    BoundingBox: {
                        Width: 0.24299885,
                        Height: 0.21577217,
                        Left: 0.7309805,
                        Top: 0.52518845
                    },
                    Confidence: 94.146774
                }
            ],
            Parents: [
                {
                    Name: Vehicle
                }
            ]
        },
        {
            Name: Downtown,
            Confidence: 91.13891,
            Instances: [

            ],
            Parents: [
                {
                    Name: Urban
                },
                {
                    Name: City
                }
            ]
        }
    ],

}
```

Detecting Labels in a Video

Amazon Rekognition Video can detect labels, and the time a label is detected, in a video. For an SDK code example, see Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68). For an AWS CLI example, see Analyzing a Video with the AWS Command Line Interface (p. 75).

Amazon Rekognition Video label detection is an asynchronous operation. To start the detection of labels in a video, call StartLabelDetection (p. 349). Amazon 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. 296) to get the detected labels. For information about calling the video recognition API operations, see Calling Amazon Rekognition Video Operations (p. 61).

GetLabelDetection Operation Response

`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.

> **Note**
> `GetLabelDetection` doesn't support the hierarchical taxonomy or bounding box information that the image operation `DetectLabels` supports. GetLabelDetection returns `null` for the `Parents` and `Instances` attributes.

The following example is the JSON response of the `GetLabelDetection`. 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. 395)) element.
- **Label information** – The `LabelDetection` array element contains a (Label (p. 393)) 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 Getting Amazon Rekognition Video Analysis Results (p. 63).
- **Video information** – The response includes information about the video format (`VideoMetadata`) in each page of information returned by `GetLabelDetection`.


```
{
  "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+dhCwbXxoVOgMi0di6xYSeL6/Dztya/
Pflx9xxxx==",
  "VideoMetadata": {
    "Codec": "h264",
    "DurationMillis": 67301,
    "Format": "mov,mp4,m4a,3gp,3g2,mj2",
    "FrameHeight": 1080,
    "FrameRate": 29.970029830932617,
    "FrameWidth": 1920
  }
}
```

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 (for example, appearing 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.

When you provide an image that contains a face, 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 that are detected in the image.

This section provides examples for both image and video facial analysis. For more information about using the Amazon Rekognition API, see Working with Images (p. 36) and Working with Stored Videos (p. 59).

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 video. For more information, see Searching Faces in a Collection (p. 137).

Topics

- Detecting Faces in an Image (p. 115)
- Comparing Faces in Images (p. 124)
- Detecting Faces in a Stored Video (p. 131)

Detecting Faces in an Image

Amazon Rekognition Image provides the DetectFaces (p. 261) operation that looks for key facial features such as eyes, nose, and mouth to detect faces in an input image. Amazon Rekognition Image detects the 100 largest faces in an image.

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

To detect faces in an image

1. If you haven't already:
   a. Create or update an IAM user with `AmazonRekognitionFullAccess` and `AmazonS3ReadOnlyAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).
2. Upload an image (that contains 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*.
3. Use the following examples to call `DetectFaces`.

   Java

   This example displays the estimated age range for detected faces, and lists the JSON for all detected facial attributes. Change the value of `photo` to the image file name. Change the value of `bucket` to the Amazon S3 bucket where the image is stored.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.rekognition.model.AgeRange;
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.fasterxml.jackson.databind.ObjectMapper;
import java.util.List;


public class DetectFaces {


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

      String photo = "input.jpg";
      String bucket = "bucket";
```

```
     AmazonRekognition rekognitionClient =
AmazonRekognitionClientBuilder.defaultClient();


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

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

        for (FaceDetail face: faceDetails) {
           if (request.getAttributes().contains("ALL")) {
              AgeRange ageRange = face.getAgeRange();
              System.out.println("The detected face is estimated to be between "
                 + ageRange.getLow().toString() + " and " +
ageRange.getHigh().toString()
                 + " years old.");
              System.out.println("Here's the complete set of attributes:");
           } else { // non-default attributes have null values.
              System.out.println("Here's the default set of attributes:");
           }

           ObjectMapper objectMapper = new ObjectMapper();

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

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

  }

}
```

AWS CLI

This example displays the JSON output from the `detect-faces` AWS CLI operation. Replace `file` with the name of an image file. Replace `bucket` with the name of the Amazon S3 bucket that contains the image file.

```
aws rekognition detect-faces \
--image '{"S3Object":{"Bucket":"bucket","Name":"file"}}' \
--attributes "ALL"
```

Python

This example displays the estimated age range for detected faces, and lists the JSON for all detected facial attributes. Change the value of `photo` to the image file name. Change the value of `bucket` to the Amazon S3 bucket where the image is stored.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)
```

```
import boto3
import json

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

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

    print('Detected faces for ' + photo)
    for faceDetail in response['FaceDetails']:
        print('The detected face is between ' + str(faceDetail['AgeRange']['Low'])

              + ' and ' + str(faceDetail['AgeRange']['High']) + ' years old')
        print('Here are the other attributes:')
        print(json.dumps(faceDetail, indent=4, sort_keys=True))
```

.NET

This example displays the estimated age range for detected faces, and lists the JSON for all detected facial attributes. Change the value of `photo` to the image file name. Change the value of `bucket` to the Amazon S3 bucket where the image is stored.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using System.Collections.Generic;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DetectFaces
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "bucket";

        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        DetectFacesRequest detectFacesRequest = new DetectFacesRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket
                },
            },
            // Attributes can be "ALL" or "DEFAULT".
            // "DEFAULT": BoundingBox, Confidence, Landmarks, Pose, and Quality.
            // "ALL": See https://docs.aws.amazon.com/sdkfornet/v3/apidocs/items/
Rekognition/TFaceDetail.html
            Attributes = new List<String>() { "ALL" }
        };

        try
        {
```

```
            DetectFacesResponse detectFacesResponse =
rekognitionClient.DetectFaces(detectFacesRequest);
            bool hasAll = detectFacesRequest.Attributes.Contains("ALL");
            foreach(FaceDetail face in detectFacesResponse.FaceDetails)
            {
                Console.WriteLine("BoundingBox: top={0} left={1} width={2}
height={3}", face.BoundingBox.Left,
                    face.BoundingBox.Top, face.BoundingBox.Width,
face.BoundingBox.Height);
                Console.WriteLine("Confidence: {0}\nLandmarks: {1}\nPose: pitch={2}
roll={3} yaw={4}\nQuality: {5}",
                    face.Confidence, face.Landmarks.Count, face.Pose.Pitch,
                    face.Pose.Roll, face.Pose.Yaw, face.Quality);
                if (hasAll)
                    Console.WriteLine("The detected face is estimated to be between
" +
                        face.AgeRange.Low + " and " + face.AgeRange.High + " years
old.");
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

DetectFaces Operation Request

The input to `DetectFaces` is an image. In this example, the image is loaded from an Amazon S3 bucket. The `Attributes` parameter specifies that all facial attributes should be returned. For more information, see Working with Images (p. 36).

```
{
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "Attributes": [
        "ALL"
    ]
}
```

DetectFaces Operation Response

`DetectFaces` returns the following information for each detected face:

- **Bounding box** – The coordinates of the bounding box that surrounds the face. For more information, see Getting Image Orientation and Bounding Box Coordinates (p. 49).
- **Confidence** – The 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 and 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 type (whether a person is wearing sunglasses) or a string (whether the person is male or

female). 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 Recommendations for Facial Recognition Input Images (p. 37).
- **Pose** – Describes the rotation of the face inside the image.
- **Emotions** – A set of emotions with confidence in the analysis.

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


```
{
    "FaceDetails": [
        {
            "AgeRange": {
                "High": 36,
                "Low": 19
            },
            "Beard": {
                "Confidence": 99.99388122558594,
                "Value": false
            },
            "BoundingBox": {
                "Height": 0.35353535413742065,
                "Left": 0.10707070678472519,
                "Top": 0.12777778506278992,
                "Width": 0.47138047218322754
            },
            "Confidence": 99.99736785888672,
            "Emotions": [
                {
                    "Confidence": 89.59288024902344,
                    "Type": "HAPPY"
                },
                {
                    "Confidence": 5.3619384765625,
                    "Type": "CALM"
                },
                {
                    "Confidence": 4.1074934005737305,
                    "Type": "ANGRY"
                }
            ],
            "Eyeglasses": {
                "Confidence": 99.96102142333984,
                "Value": false
            },
            "EyesOpen": {
                "Confidence": 99.97982788085938,
                "Value": true
            },
            "Gender": {
                "Confidence": 100,
                "Value": "Female"
            },
            "Landmarks": [
                {
                    "Type": "eyeLeft",
                    "X": 0.23772846162319183,
                    "Y": 0.2778792679309845
                },
                {
                    "Type": "eyeRight",
                    "X": 0.3944779932498932,
```

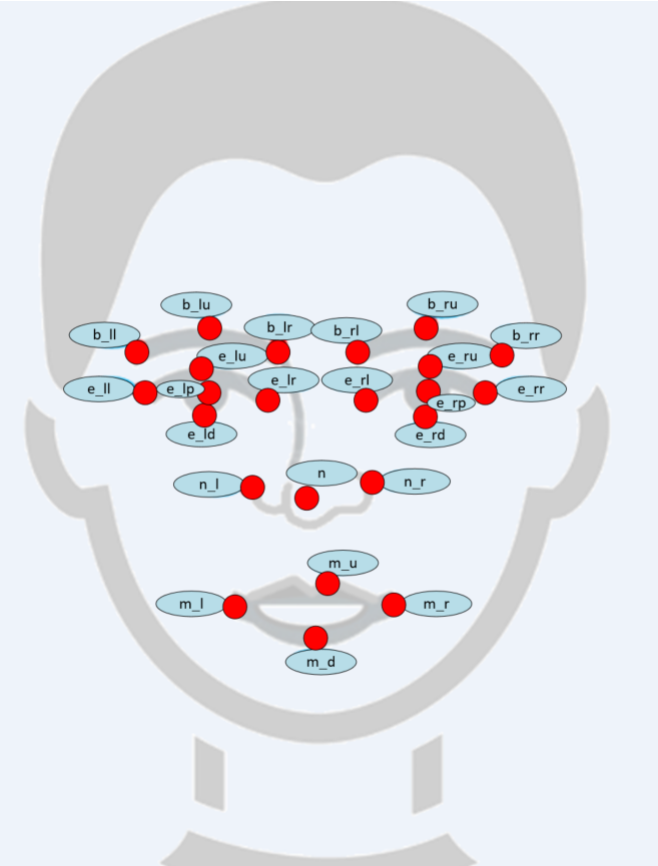
```
            "Y": 0.2527812421321869
        },
        {
            "Type": "nose",
            "X": 0.28617316484451294,
            "Y": 0.3367626965045929
        },
        {
            "Type": "mouthLeft",
            "X": 0.288897842168808,
            "Y": 0.4019121527671814
        },
        {
            "Type": "mouthRight",
            "X": 0.41363197565078735,
            "Y": 0.38766127824783325
        },
        {
            "Type": "leftPupil",
            "X": 0.25244733691215515,
            "Y": 0.27739065885543823
        },
        {
            "Type": "rightPupil",
            "X": 0.4029206931591034,
            "Y": 0.24940147995948792
        },
        {
            "Type": "leftEyeBrowLeft",
            "X": 0.17436790466308594,
            "Y": 0.24362699687480927
        },
        {
            "Type": "leftEyeBrowUp",
            "X": 0.21201953291893005,
            "Y": 0.2338741421699524
        },
        {
            "Type": "leftEyeBrowRight",
            "X": 0.2513192892074585,
            "Y": 0.24069637060165405
        },
        {
            "Type": "rightEyeBrowLeft",
            "X": 0.32193484902381897,
            "Y": 0.21918891370296478
        },
        {
            "Type": "rightEyeBrowUp",
            "X": 0.38548189401626587,
            "Y": 0.19604144990444183
        },
        {
            "Type": "rightEyeBrowRight",
            "X": 0.45430734753608704,
            "Y": 0.2027731090784073
        },
        {
            "Type": "leftEyeLeft",
            "X": 0.20944036543369293,
            "Y": 0.28281378746032715
        },
        {
            "Type": "leftEyeRight",
            "X": 0.2710888683795929,
            "Y": 0.2738289535045624
```

```
            },
            {
                "Type": "leftEyeUp",
                "X": 0.2339935451745987,
                "Y": 0.2687133252620697
            },
            {
                "Type": "leftEyeDown",
                "X": 0.23892717063426971,
                "Y": 0.28660306334495544
            },
            {
                "Type": "rightEyeLeft",
                "X": 0.36334219574928284,
                "Y": 0.2598327100276947
            },
            {
                "Type": "rightEyeRight",
                "X": 0.4293186664581299,
                "Y": 0.249033123254776
            },
            {
                "Type": "rightEyeUp",
                "X": 0.39038628339767456,
                "Y": 0.2431529313325882
            },
            {
                "Type": "rightEyeDown",
                "X": 0.3967171609401703,
                "Y": 0.26075780391693115
            },
            {
                "Type": "noseLeft",
                "X": 0.28841185569763184,
                "Y": 0.3598580062389374
            },
            {
                "Type": "noseRight",
                "X": 0.3451237976551056,
                "Y": 0.3516968786716461
            },
            {
                "Type": "mouthUp",
                "X": 0.3349839448928833,
                "Y": 0.38809144496917725
            },
            {
                "Type": "mouthDown",
                "X": 0.3422594964504242,
                "Y": 0.41868656873703003
            }
        ],
        "MouthOpen": {
            "Confidence": 99.97990417480469,
            "Value": false
        },
        "Mustache": {
            "Confidence": 99.97885131835938,
            "Value": false
        },
        "Pose": {
            "Pitch": 1.0711474418640137,
            "Roll": -10.933034896850586,
            "Yaw": -25.838171005249023
        },
        "Quality": {
```

```
                "Brightness": 43.86729431152344,
                "Sharpness": 99.95819854736328
            },
            "Smile": {
                "Confidence": 96.89310455322266,
                "Value": true
            },
            "Sunglasses": {
                "Confidence": 80.4033432006836,
                "Value": false
            }
        }
    ]
}
```

Note the following:

- The `Pose` data describes the rotation of the face detected. You can use the combination of the `BoundingBox` and `Pose` data to draw the bounding box around faces that your application displays.
- The `Quality` describes the brightness and the sharpness of the face. You might find this useful to compare faces across images and find the best face.
- The `DetectFaces` operation first detects orientation of the input image, before detecting facial features. The `OrientationCorrection` in the response returns the degrees of rotation detected (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 attributes: `BoundingBox`, `Confidence`, `Pose`, `Quality` and `landmarks`. The default landmarks returned are: `eyeLeft`, `eyeRight`, `nose`, `mouthLeft`, and `mouthRight`.
- The following illustration shows the relative location of the facial landmarks (`Landmarks`) on the face that are returned by the `DetectFaces` API operation.



The labels on the image map to the Landmark (p. 396) data type as shown in the following table. For clarity, the left eye and right eye landmark points aren't shown on the image.

Label	Landmark
b_ll	leftEyeBrowLeft
b_lu	leftEyeBrowUp
b_lr	leftEyeBrowRight
e_ll	leftEyeLeft
e_lu	leftEyeUp
e_lr	leftEyeRight
e_ld	leftEyeDown
e_lp	leftPupil
n_l	noseLeft
n	nose
n_r	noseRight

Label	Landmark
m_u	mouthUp
m_d	mouthDown
m_l	mouthLeft
m_r	mouthRight
b_rl	rightEyeBrowLeft
b_ru	rightEyeBrowUp
b_rr	rightEyeBrowRight
e_rl	rightEyeLeft
e_ru	rightEyeUp
e_rr	rightEyeRight
e_rp	rightPupil
e_rd	rightEyeDown

Comparing Faces in Images

To compare a face in the *source* image with each face in the *target* image, use the CompareFaces (p. 236) operation.

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. 236).

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

You can provide the source and target images as an image byte array (base64-encoded image bytes), or specify Amazon S3 objects. In the AWS CLI example, you upload two JPEG images to your Amazon S3 bucket and specify the object key name. In the other examples, you load two files from the local file system and input them as image byte arrays.

To compare faces

1. If you haven't already:
   a. Create or update an IAM user with `AmazonRekognitionFullAccess` and `AmazonS3ReadOnlyAccess` (AWS CLI example only) permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).
2. Use the following example code to call the `CompareFaces` operation.

   Java

   This example displays information about matching faces in source and target images that are loaded from the local file system.

Replace the values of `sourceImage` and `targetImage` with the path and file name of the source and target images.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.model.CompareFacesMatch;
import com.amazonaws.services.rekognition.model.CompareFacesRequest;
import com.amazonaws.services.rekognition.model.CompareFacesResult;
import com.amazonaws.services.rekognition.model.ComparedFace;
import java.util.List;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.ByteBuffer;
import com.amazonaws.util.IOUtils;

public class CompareFaces {

   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;

       AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

       //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 was " + uncompared.size()
           + " face(s) that did not match");
      System.out.println("Source image rotation: " +
 compareFacesResult.getSourceImageOrientationCorrection());
      System.out.println("target image rotation: " +
 compareFacesResult.getTargetImageOrientationCorrection());
   }
}
```

AWS CLI

This example displays the JSON output from the `compare-faces` AWS CLI operation.

Replace `bucket-name` with the name of the Amazon S3 bucket that contains the source and target images. Replace `source.jpg` and `target.jpg` with the file names for the source and target images.

```
aws rekognition compare-faces \
--source-image '{"S3Object":{"Bucket":"bucket-name","Name":"source.jpg"}}' \
--target-image '{"S3Object":{"Bucket":"bucket-name","Name":"target.jpg"}}'
```

Python

This example displays information about matching faces in source and target images that are loaded from the local file system.

Replace the values of `sourceFile` and `targetFile` with the path and file name of the source and target images.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

if __name__ == "__main__":

    sourceFile='source.jpg'
    targetFile='target.jpg'
    client=boto3.client('rekognition')

    imageSource=open(sourceFile,'rb')
    imageTarget=open(targetFile,'rb')
```

```
    response=client.compare_faces(SimilarityThreshold=70,
                                  SourceImage={'Bytes': imageSource.read()},
                                  TargetImage={'Bytes': imageTarget.read()})

    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')

    imageSource.close()
    imageTarget.close()
```

.NET

This example displays information about matching faces in source and target images that are loaded from the local file system.

Replace the values of `sourceImage` and `targetImage` with the path and file name of the source and target images.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using System.IO;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class CompareFaces
{
    public static void Example()
    {
        float similarityThreshold = 70F;
        String sourceImage = "source.jpg";
        String targetImage = "target.jpg";

        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        Amazon.Rekognition.Model.Image imageSource = new
 Amazon.Rekognition.Model.Image();
        try
        {
            using (FileStream fs = new FileStream(sourceImage, FileMode.Open,
 FileAccess.Read))
            {
                byte[] data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                imageSource.Bytes = new MemoryStream(data);
            }
        }
        catch (Exception)
        {
            Console.WriteLine("Failed to load source image: " + sourceImage);
            return;
        }
```

```
        Amazon.Rekognition.Model.Image imageTarget = new
Amazon.Rekognition.Model.Image();
        try
        {
            using (FileStream fs = new FileStream(targetImage, FileMode.Open,
FileAccess.Read))
            {
                byte[] data = new byte[fs.Length];
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
                imageTarget.Bytes = new MemoryStream(data);
            }
        }
        catch (Exception)
        {
            Console.WriteLine("Failed to load target image: " + targetImage);
            return;
        }

        CompareFacesRequest compareFacesRequest = new CompareFacesRequest()
        {
            SourceImage = imageSource,
            TargetImage = imageTarget,
            SimilarityThreshold = similarityThreshold
        };

        // Call operation
        CompareFacesResponse compareFacesResponse =
rekognitionClient.CompareFaces(compareFacesRequest);

        // Display results
        foreach(CompareFacesMatch match in compareFacesResponse.FaceMatches)
        {
            ComparedFace face = match.Face;
            BoundingBox position = face.BoundingBox;
            Console.WriteLine("Face at " + position.Left
                  + " " + position.Top
                  + " matches with " + face.Confidence
                  + "% confidence.");
        }

        Console.WriteLine("There was " + compareFacesResponse.UnmatchedFaces.Count
+ " face(s) that did not match");
        Console.WriteLine("Source image rotation: " +
compareFacesResponse.SourceImageOrientationCorrection);
        Console.WriteLine("Target image rotation: " +
compareFacesResponse.TargetImageOrientationCorrection);
    }
}
```

CompareFaces Operation Request

The input to `CompareFaces` is an image. In this example, the source and target images are loaded from the local file system. The `SimilarityThreshold` input parameter specifies the minimum confidence that compared faces must match to be included in the response. For more information, see Working with Images (p. 36).

```
{
    "SourceImage": {
        "Bytes": "/9j/4AAQSk2Q==..."
    },
    "TargetImage": {
```

```
        "Bytes": "/9j/4O1Q==..."
    },
    "SimilarityThreshold": 70
}
```

CompareFaces Operation Response

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. Face metadata includes information such as the bounding box of the matching face and an array of facial landmarks. The array of unmatched faces includes face metadata.

In the following example response, note the following:

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

  If multiple faces match, the `faceMatches` array includes all of the face matches.
- **Source face information** – The response includes information about the face from the source image that was used for comparison, including the bounding box and confidence value.
- **Image Orientation** – The response includes information about the orientation of the source and target images. Amazon Rekognition needs this to display the images and retrieve the correct location of the matched face in the target image. For more information, see Getting Image Orientation and Bounding Box Coordinates (p. 49).
- **Unmatched face match information** – The example shows one face that Amazon Rekognition found in the target image that didn't match the face that was analyzed in the source image. For that face, it provides a bounding box and a confidence value, which indicates the level of confidence that Amazon Rekognition has that the bounding box contains a face. The face information also includes an array of landmark locations.

  If Amazon Rekognition finds multiple faces that don't match, the `UnmatchedFaces` array includes all of the faces that didn't match.

```
{
    "FaceMatches": [{
        "Face": {
            "BoundingBox": {
                "Width": 0.5521978139877319,
                "Top": 0.1203877404332161,
                "Left": 0.23626373708248138,
                "Height": 0.3126954436302185
            },
            "Confidence": 99.98751068115234,
            "Pose": {
                "Yaw": -82.36799621582031,
                "Roll": -62.13221740722656,
                "Pitch": 0.8652129173278809
            },
            "Quality": {
                "Sharpness": 99.99880981445312,
                "Brightness": 54.49755096435547
            },
            "Landmarks": [{
                    "Y": 0.2996366024017334,
```

```
                    "X": 0.41685718297958374,
                    "Type": "eyeLeft"
                },
                {
                    "Y": 0.2658946216106415,
                    "X": 0.4414493441581726,
                    "Type": "eyeRight"
                },
                {
                    "Y": 0.3465650677680969,
                    "X": 0.48636093735694885,
                    "Type": "nose"
                },
                {
                    "Y": 0.30935320258140564,
                    "X": 0.6251809000968933,
                    "Type": "mouthLeft"
                },
                {
                    "Y": 0.26942989230155945,
                    "X": 0.6454493403434753,
                    "Type": "mouthRight"
                }
            ]
        },
        "Similarity": 100.0
    }],
    "SourceImageOrientationCorrection": "ROTATE_90",
    "TargetImageOrientationCorrection": "ROTATE_90",
    "UnmatchedFaces": [{
        "BoundingBox": {
            "Width": 0.4890109896659851,
            "Top": 0.6566604375839233,
            "Left": 0.10989011079072952,
            "Height": 0.278298944234848
        },
        "Confidence": 99.99992370605469,
        "Pose": {
            "Yaw": 51.51519012451172,
            "Roll": -110.32493591308594,
            "Pitch": -2.322134017944336
        },
        "Quality": {
            "Sharpness": 99.99671173095703,
            "Brightness": 57.23163986206055
        },
        "Landmarks": [{
                "Y": 0.8288310766220093,
                "X": 0.3133862614631653,
                "Type": "eyeLeft"
            },
            {
                "Y": 0.7632885575294495,
                "X": 0.28091415762901306,
                "Type": "eyeRight"
            },
            {
                "Y": 0.7417283654212952,
                "X": 0.3631140887737274,
                "Type": "nose"
            },
            {
                "Y": 0.8081989884376526,
                "X": 0.48565614223480225,
                "Type": "mouthLeft"
            },
```

```
            {
                "Y": 0.7548204660415649,
                "X": 0.46090251207351685,
                "Type": "mouthRight"
            }
        ]
    }],
    "SourceImageFace": {
        "BoundingBox": {
            "Width": 0.5521978139877319,
            "Top": 0.1203877404332161,
            "Left": 0.23626373708248138,
            "Height": 0.3126954436302185
        },
        "Confidence": 99.98751068115234
    }
}
```

Detecting Faces in a Stored Video

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

- The time or times 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.

Amazon Rekognition Video face detection in stored videos is an asynchronous operation. To start the detection of faces in videos, call StartFaceDetection (p. 341). Amazon Rekognition Video publishes the completion status of the video analysis to an Amazon Simple Notification Service (Amazon SNS) topic. If the video analysis is successful, you can call GetFaceDetection (p. 286) to get the results of the video analysis. For more information about starting video analysis and getting the results, see Calling Amazon Rekognition Video Operations (p. 61).

This procedure expands on the code in Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68), which uses an Amazon Simple Queue Service (Amazon SQS) queue to get the completion status of a video analysis request.

To detect faces in a video stored in an Amazon S3 bucket (SDK)

1. Perform Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68).
2. Add the following code to the class `VideoDetect` that you created in step 1.

   Java

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)


   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);


  }
```

2a. In the function `main`, replace the line:

```
StartLabels(bucket,video);
```

with:

```
StartFaces(bucket,video);
```

2b. Replace the line:

```
GetResultsLabels();
```

with:

```
GetResultsFaces();
```

Python

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    def GetResultsFaces(self, jobId):
        maxResults = 10
        paginationToken = ''
        finished = False

        while finished == False:
            response = self.rek.get_face_detection(JobId=jobId,
                                        MaxResults=maxResults,
                                        NextToken=paginationToken)

            print(response['VideoMetadata']['Codec'])
            print(str(response['VideoMetadata']['DurationMillis']))
            print(response['VideoMetadata']['Format'])
            print(response['VideoMetadata']['FrameRate'])

            for faceDetection in response['Faces']:
                print('Face: ' + str(faceDetection['Face']))
                print('Confidence: ' + str(faceDetection['Face']['Confidence']))
                print('Timestamp: ' + str(faceDetection['Timestamp']))
                print()

            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True
```

2a. In the function `main`, replace the line:

```
        response = self.rek.start_label_detection(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
                                            NotificationChannel={'RoleArn':
 self.roleArn, 'SNSTopicArn': self.topicArn})
```

with:

```
        response = self.rek.start_face_detection(Video={'S3Object':
{'Bucket':self.bucket,'Name':self.video}},
            NotificationChannel={'RoleArn':self.roleArn,
 'SNSTopicArn':self.topicArn})
```

2b. Replace the line:

```
                        self.GetResultsLabels(rekMessage['JobId'])
```

with:

```
                        self.GetResultsFaces(rekMessage['JobId'])
```

Note
If you've already run a video example other than Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68), the function name to replace is different.

3. Run the code. Information about the faces that were detected in the video is shown.

GetFaceDetection Operation Response

`GetFaceDetection` returns an array (`Faces`) that contains information about the faces detected in the video. An array element, FaceDetection (p. 382), exists for each time a face is detected in the video. The array elements returned are sorted by time, in milliseconds since the start of the video.

The following example is a partial JSON response from `GetFaceDetection`. In the response, note the following:

- **Face information** – The `FaceDetection` array element contains information about the detected face (FaceDetail (p. 379)) and the time that 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`) that's used to get the next page of results. For more information, see Getting Amazon Rekognition Video Analysis Results (p. 63).
- **Video information** – The response includes information about the video format (`VideoMetadata`) in each page of information that's returned by `GetFaceDetection`.


```
{
    "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": 0.49765899777412415,
                        "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": 0.13333334028720856,
```

```
                    "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
                }
            },
            "Timestamp": 0
        }.......

    ],
    "JobStatus": "SUCCEEDED",
    "NextToken": "i7fj5XPV/
fwviXqz0eag9Ow332Jd5G8ZGWf7hooirD/6V1qFmjKFOQZ6QPWUiqv29HbyuhMNqQ==",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 67301,
        "FileExtension": "mp4",
        "Format": "QuickTime / MOV",
        "FrameHeight": 1080,
        "FrameRate": 29.970029830932617,
        "FrameWidth": 1920
    }
}
```

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 that's stored in a collection to search for known faces in images, stored videos, and streaming videos. Amazon Rekognition supports the IndexFaces (p. 305) operation. You can use this operation to detect faces in an image and persist information about facial features that are 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. 241)) 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. 329). To search for faces in a stored video, call StartFaceSearch (p. 345). To search for faces in a streaming video, call CreateStreamProcessor (p. 244).

Note
The service doesn't persist actual image bytes. Instead, the underlying detection algorithm first detects the faces in the input image, extracts facial features into a feature vector for each face, and then stores it in the collection. Amazon Rekognition uses these feature vectors when performing face matches.

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

Managing Collections

The face collection is the primary Amazon Rekognition resource, and 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. When a collection is created, it's associated with the most recent version of the face detection model. For more information, see Model Versioning (p. 9).

You can perform the following management operations on a collection.

- Create a collection with the section called “CreateCollection” (p. 241). For more information, see Creating a Collection (p. 139).
- List the available collections with the section called “ListCollections” (p. 313). For more information, see Listing Collections (p. 142).
- Describe a collection with the section called “DescribeCollection” (p. 254). For more information, see Describing a Collection (p. 145).
- Delete a collection with the section called “DeleteCollection” (p. 247). For more information, see Deleting a Collection (p. 148).

Managing Faces in a Collection

After you create a face collection, you can store faces in it. Amazon Rekognition provides the following operations for managing faces in a collection.

- The the section called “IndexFaces” (p. 305) operation detects faces in the input image (JPEG or PNG), and adds them to the specified face collection. A unique face ID is returned for each face that's

detected in the image. After you persist faces, you can search the face collection for face matches. For more information, see Adding Faces to a Collection (p. 151).

- The the section called "ListFaces" (p. 316) operation lists the faces in a collection. For more information, see Adding Faces to a Collection (p. 151).
- The the section called "DeleteFaces" (p. 249) operation deletes faces from a collection. For more information, see Deleting Faces from a Collection (p. 165).

Searching for Faces Within a Collection

After you create a face collection and store faces, you can search a face collection for face matches. With Amazon Rekognition, you can search for faces in a collection that match:

- A supplied face ID (the section called "SearchFaces" (p. 326)). For more information, see Searching for a Face Using Its Face ID (p. 168).
- The largest face in a supplied image (the section called "SearchFacesByImage" (p. 329)). For more information, see Searching for a Face Using an Image (p. 172).
- Faces in a stored video. For more information, see Searching Stored Videos for Faces (p. 176).
- Faces in a streaming video. For more information, see Working with Streaming Videos (p. 86).

The `CompareFaces` operation and the 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 that are detected in the target image. For more information, see Comparing Faces in Images (p. 124).
- `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 that are already stored in the face collection, you can search for matching faces multiple times.

Using Similarity Thresholds to Match Faces

We allow you to control the results of all search operations (CompareFaces (p. 236), SearchFaces (p. 326), and SearchFacesByImage (p. 329)) by providing a similarity threshold as an input parameter.

The similarity threshold input attribute for `SearchFaces` and `SearchFacesByImage`, `FaceMatchThreshold`, controls how many results are returned based on the similarity to the face being matched. (This attribute is `SimilarityThreshold` for `CompareFaces`.) Responses with a `Similarity` response attribute value that's lower than the threshold aren't returned. This threshold is important to calibrate for your use case, because it can determine how many false positives are included in your match results. This controls the recall of your search results—the lower the threshold, the higher the recall.

All machine learning systems are probabilistic. You should use your judgment in setting the right similarity threshold, depending on your use case. For example, if you're looking to build a photos app to identify similar-looking family members, you might choose a lower threshold (such as 80%). On the other hand, for many law enforcement use cases, we recommend using a high threshold value of 99% or above to reduce accidental misidentification.

In addition to `FaceMatchThreshold`, you can use the `Similarity` response attribute as a means to reduce accidental misidentification. For instance, you can choose to use a low threshold (like 80%) to return more results. Then you can use the response attribute `Similarity` (percentage of similarity) to narrow the choice and filter for the right responses in your application. Again, using a higher similarity (such as 99% and above) reduces the risk of misidentification.

Using Amazon Rekognition to Help Public Safety

Amazon Rekognition can help in public safety and law enforcement scenarios—such as finding lost children, combating human trafficking, or preventing crimes. In public safety and law enforcement scenarios, consider the following:

- Use Amazon Rekognition as the first step in recognizing an individual. The responses from Amazon Rekognition face operations allow you to quickly get a set of potential matches for further consideration.
- Don't use Amazon Rekognition responses to make autonomous decisions for scenarios that require analysis by a human. An example of this is determining who committed a crime. Instead, have a human review the responses, and use that information to inform further decisions.
- Some scenarios might not need human review of Amazon Rekognition responses. An example of this is two-factor authentication with a badge and a face that's recognized by Amazon Rekognition with a high (99%) similarity.
- Use a 99% similarity threshold for scenarios where highly accurate face similarity matches are necessary. An example of this is authenticating access to a building.
- Use a similarity threshold lower than 99% for scenarios that benefit from a larger set of potential matches. An example of this is finding missing persons. If necessary, you can use the Similarity response attribute to determine how similar potential matches are to the person you want to recognize.
- Have a plan for false-positive face matches that are returned by Amazon Rekognition. For example, improve matching by using multiple images of the same person when you build the index with the IndexFaces (p. 305) operation.

In other use cases (such as social media), there isn't the same need to double check responses. Also, depending on your application's requirements, the similarity threshold can be lower.

Creating a Collection

You can use the CreateCollection (p. 241) operation to create a collection.

For more information, see Managing Collections (p. 137).

To create a collection (SDK)

1. If you haven't already:
   a. Create or update an IAM user with `AmazonRekognitionFullAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).
2. Use the following examples to call the `CreateCollection` operation.

   Java

   The following example creates a collection and displays its Amazon Resource Name (ARN).

   Change the value of `collectionId` to the name of the collection you want to create.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)
```

```
package aws.example.rekognition.image;

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 {


      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();


      String collectionId = "MyCollection";
            System.out.println("Creating collection: " +
         collectionId );

        CreateCollectionRequest request = new CreateCollectionRequest()
                    .withCollectionId(collectionId);

      CreateCollectionResult createCollectionResult =
 rekognitionClient.createCollection(request);
      System.out.println("CollectionArn : " +
         createCollectionResult.getCollectionArn());
      System.out.println("Status code : " +
         createCollectionResult.getStatusCode().toString());

   }

}
```

AWS CLI

This AWS CLI command displays the JSON output for the `create-collection` CLI operation.

Replace the value of `collection-id` with the name of the collection you want to create.

```
aws rekognition create-collection \
    --collection-id "collectionname"
```

Python

The following example creates a collection and displays its Amazon Resource Name (ARN).

Change the value of `collectionId` to the name of collection you want to create.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

if __name__ == "__main__":

    maxResults=2
    collectionId='MyCollection'
```

```
    client=boto3.client('rekognition')

    #Create a collection
    print('Creating collection:' + collectionId)
    response=client.create_collection(CollectionId=collectionId)
    print('Collection ARN: ' + response['CollectionArn'])
    print('Status code: ' + str(response['StatusCode']))
    print('Done...')
```

.NET

The following example creates a collection and displays its Amazon Resource Name (ARN).

Change the value of `collectionId` to the name of collection you want to create.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class CreateCollection
{
    public static void Example()
    {
        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        String collectionId = "MyCollection";
        Console.WriteLine("Creating collection: " + collectionId);

        CreateCollectionRequest createCollectionRequest = new
 CreateCollectionRequest()
        {
            CollectionId = collectionId
        };

        CreateCollectionResponse createCollectionResponse =
 rekognitionClient.CreateCollection(createCollectionRequest);
        Console.WriteLine("CollectionArn : " +
 createCollectionResponse.CollectionArn);
        Console.WriteLine("Status code : " + createCollectionResponse.StatusCode);

    }
}
```

CreateCollection Operation Request

The input to `CreationCollection` is the name of the collection that you want to create.

```
{
    "CollectionId": "MyCollection"
}
```

CreateCollection Operation Response

Amazon Rekognition creates the collection and returns the Amazon Resource Name (ARN) of the newly created collection.

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

Listing Collections

You can use the ListCollections (p. 313) operation to list the collections in the region that you are using.

For more information, see Managing Collections (p. 137).

To list collections (SDK)

1. If you haven't already:
   a. Create or update an IAM user with `AmazonRekognitionFullAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).
2. Use the following examples to call the `ListCollections` operation.

   Java

   The following example lists the collections in the current region.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;

import java.util.List;
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 {


      RotateImageAmazonRekognition amazonRekognition =
 AmazonRekognitionClientBuilder.defaultClient();


      System.out.println("Listing collections");
      int limit = 10;
      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);

   }
}
```

AWS CLI

This AWS CLI command displays the JSON output for the `list-collections` CLI operation.

```
aws rekognition list-collections
```

Python

The following example lists the collections in the current region.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

if __name__ == "__main__":

    maxResults=2

    client=boto3.client('rekognition')

    #Display all the collections
    print('Displaying collections...')
    response=client.list_collections(MaxResults=maxResults)

    while True:
        collections=response['CollectionIds']

        for collection in collections:
            print (collection)
        if 'NextToken' in response:
            nextToken=response['NextToken']

 response=client.list_collections(NextToken=nextToken,MaxResults=maxResults)

        else:
            break

    print('done...')
```

.NET

The following example lists the collections in the current region.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class ListCollections
{
    public static void Example()
    {
        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        Console.WriteLine("Listing collections");
        int limit = 10;

        ListCollectionsResponse listCollectionsResponse = null;
        String paginationToken = null;
        do
        {
            if (listCollectionsResponse != null)
                paginationToken = listCollectionsResponse.NextToken;

            ListCollectionsRequest listCollectionsRequest = new
 ListCollectionsRequest()
            {
                MaxResults = limit,
                NextToken = paginationToken
            };

            listCollectionsResponse =
 rekognitionClient.ListCollections(listCollectionsRequest);

            foreach (String resultId in listCollectionsResponse.CollectionIds)
                Console.WriteLine(resultId);
        } while (listCollectionsResponse != null &&
 listCollectionsResponse.NextToken != null);
    }
}
```

ListCollections Operation Request

The input to `ListCollections` is the maximum number of collections to be returned.

```
{
    "MaxResults": 2
}
```

If the response has more faces than requested by `MaxResults`, a token is returned that you can use to get the next set of results, in a subsequent call to `ListCollections`. For example:

```
{
    "NextToken": "MGYZLAHX1T5a....",
    "MaxResults": 2
```

```
}
```

ListCollections Operation Response

Amazon Rekognition returns an array of collections (`CollectionIds`). A separate array (`FaceModelversions`) provides the version of the face model used to analyze faces in each collection. For example, in the following JSON response, the collection `MyCollection` analyzes faces by using version 2.0 of the face model. The collection `AnotherCollection` uses version 3.0 of the face model. For more information, see Model Versioning (p. 9).

`NextToken` is the token that's used to get the next set of results, in a subsequent call to `ListCollections`.

```
{
    "CollectionIds": [
        "MyCollection",
        "AnotherCollection"
    ],
    "FaceModelVersions": [
        "2.0",
        "3.0"
    ],
    "NextToken": "MGYZLAHX1T5a...."
}
```

Describing a Collection

You can use the DescribeCollection (p. 254) operation to get the following information about a collection:

- The number of faces that are indexed into the collection.
- The version of the model that's used by the collection for face detection. For more information, see the section called “Model Versioning” (p. 9).
- The Amazon Resource Name (ARN) of the collection.
- The creation date and time of the collection.

To describe a collection (SDK)

1. If you haven't already:
   a. Create or update an IAM user with `AmazonRekognitionFullAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).
2. Use the following examples to call the `DescribeCollection` operation.

   Java

   This example describes a collection.

   Change the value `collectionId` to the ID of the desired collection.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)
```

```
package com.amazonaws.samples;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.DescribeCollectionRequest;
import com.amazonaws.services.rekognition.model.DescribeCollectionResult;


public class DescribeCollection {

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

      String collectionId = "CollectionID";

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();


      System.out.println("Describing collection: " +
         collectionId );


        DescribeCollectionRequest request = new DescribeCollectionRequest()
                    .withCollectionId(collectionId);

      DescribeCollectionResult describeCollectionResult =
 rekognitionClient.describeCollection(request);
      System.out.println("Collection Arn : " +
         describeCollectionResult.getCollectionARN());
      System.out.println("Face count : " +
         describeCollectionResult.getFaceCount().toString());
      System.out.println("Face model version : " +
         describeCollectionResult.getFaceModelVersion());
      System.out.println("Created : " +
         describeCollectionResult.getCreationTimestamp().toString());

   }

}
```

AWS CLI

This AWS CLI command displays the JSON output for the `describe-collection` CLI operation. Change the value of `collection-id` to the ID of the desired collection.

```
aws rekognition describe-collection --collection-id collectionname
```

Python

This example describes a collection.

Change the value `collectionId` to the ID of the desired collection.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
from botocore.exceptions import ClientError
```

```
from os import environ

if __name__ == "__main__":

    collectionId='MyCollection'
    print('Attempting to describe collection ' + collectionId)
    client=boto3.client('rekognition')

    try:
        response=client.describe_collection(CollectionId=collectionId)
        print("Collection Arn: "  + response['CollectionARN'])
        print("Face Count: "  + str(response['FaceCount']))
        print("Face Model Version: "  + response['FaceModelVersion'])
        print("Timestamp: "  + str(response['CreationTimestamp']))


    except ClientError as e:
        if e.response['Error']['Code'] == 'ResourceNotFoundException':
            print ('The collection ' + collectionId + ' was not found ')
        else:
            print ('Error other than Not Found occurred: ' + e.response['Error']
['Message'])
    print('Done...')
```

.NET

This example describes a collection.

Change the value `collectionId` to the ID of the desired collection.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DescribeCollection
{
    public static void Example()
    {
        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        String collectionId = "CollectionID";
        Console.WriteLine("Describing collection: " + collectionId);

        DescribeCollectionRequest describeCollectionRequest = new
 DescribeCollectionRequest()
        {
            CollectionId = collectionId
        };

        DescribeCollectionResponse describeCollectionResponse =
 rekognitionClient.DescribeCollection(describeCollectionRequest);
        Console.WriteLine("Collection ARN: " +
 describeCollectionResponse.CollectionARN);
        Console.WriteLine("Face count: " + describeCollectionResponse.FaceCount);
```

```
        Console.WriteLine("Face model version: " +
 describeCollectionResponse.FaceModelVersion);
        Console.WriteLine("Created: " +
 describeCollectionResponse.CreationTimestamp);
    }
}
```

DescribeCollection Operation Request

The input to `DescribeCollection` is the ID of the desired collection, as shown in the following JSON example.

```
{
    "CollectionId": "MyCollection"
}
```

DescribeCollection Operation Response

The response includes:

- The number of faces that are indexed into the collection, `FaceCount`.
- The version of the face model that's used to detect faces, `FaceModelVersion`.
- The collection Amazon Resource Name, `CollectionARN`.
- The creation time and date of the collection, `CreationTimestamp`. The value of `CreationTimestamp` is the number of milliseconds since the Unix epoch time until the creation of the collection. The Unix epoch time is 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970. For more information, see Unix Time.

```
{
    "CollectionARN": "arn:aws:rekognition:us-east-1:nnnnnnnnnnnn:collection/MyCollection",
    "CreationTimestamp": 1.533422155042E9,
    "FaceCount": 200,
    "FaceModelVersion": "1.0"
}
```

Deleting a Collection

You can use the DeleteCollection (p. 247) operation to delete a collection.

For more information, see Managing Collections (p. 137).

To delete a collection (SDK)

1. If you haven't already:
   a. Create or update an IAM user with `AmazonRekognitionFullAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).
2. Use the following examples to call the `DeleteCollection` operation.

Java

This example deletes a collection.

Change the value `collectionId` to the collection that you want to delete.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
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 {

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      String collectionId = "MyCollection";

      System.out.println("Deleting collections");

      DeleteCollectionRequest request = new DeleteCollectionRequest()
         .withCollectionId(collectionId);
      DeleteCollectionResult deleteCollectionResult =
 rekognitionClient.deleteCollection(request);

      System.out.println(collectionId + ": " +
 deleteCollectionResult.getStatusCode()
         .toString());

   }

}
```

AWS CLI

This AWS CLI command displays the JSON output for the `delete-collection` CLI operation. Replace the value of `collection-id` with the name of the collection that you want to delete.

```
aws rekognition delete-collection \
    --collection-id "collectionname"
```

Python

This example deletes a collection.

Change the value `collectionId` to the collection that you want to delete.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
from botocore.exceptions import ClientError
from os import environ

if __name__ == "__main__":

    collectionId='MyCollection'
    print('Attempting to delete collection ' + collectionId)
    client=boto3.client('rekognition')
    statusCode=''
    try:
        response=client.delete_collection(CollectionId=collectionId)
        statusCode=response['StatusCode']

    except ClientError as e:
        if e.response['Error']['Code'] == 'ResourceNotFoundException':
            print ('The collection ' + collectionId + ' was not found ')
        else:
            print ('Error other than Not Found occurred: ' + e.response['Error']
['Message'])
        statusCode=e.response['ResponseMetadata']['HTTPStatusCode']
    print('Operation returned Status Code: ' + str(statusCode))
    print('Done...')
```

.NET

This example deletes a collection.

Change the value `collectionId` to the collection that you want to delete.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DeleteCollection
{
    public static void Example()
    {
        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        String collectionId = "MyCollection";
        Console.WriteLine("Deleting collection: " + collectionId);

        DeleteCollectionRequest deleteCollectionRequest = new
 DeleteCollectionRequest()
        {
            CollectionId = collectionId
        };

        DeleteCollectionResponse deleteCollectionResponse =
 rekognitionClient.DeleteCollection(deleteCollectionRequest);
        Console.WriteLine(collectionId + ": " +
 deleteCollectionResponse.StatusCode);
    }
```

```
}
```

DeleteCollection Operation Request

The input to `DeleteCollection` is the ID of the collection to be deleted, as shown in the following JSON example.

```
{
    "CollectionId": "MyCollection"
}
```

DeleteCollection Operation Response

The `DeleteCollection` response contains an HTTP status code that indicates the success or failure of the operation. `200` is returned if the collection is successfully deleted.

```
{"StatusCode":200}
```

Adding Faces to a Collection

You can use the IndexFaces (p. 305) operation to detect faces in an image and add them to a collection. For each face detected, Amazon Rekognition extracts facial features and stores the feature information in a database. In addition, the command stores metadata for each face that's detected in the specified face collection. Amazon Rekognition doesn't store the actual image bytes.

For information about providing suitable faces for indexing, see Recommendations for Facial Recognition Input Images (p. 37).

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

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

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

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

  - `the section called “ListFaces” (p. 316)`
  - Search faces operations – The responses for the section called “SearchFaces” (p. 326) and the section called “SearchFacesByImage” (p. 329) return the confidence in the match for each matching face, along with this metadata of the matched face.

The number of faces indexed by `IndexFaces` depends on the version of the face detection model that's associated with the input collection. For more information, see Model Versioning (p. 9).

Information about indexed faces is returned in an array of the section called "FaceRecord" (p. 384) objects.

You might want to associate indexed faces with the image they were detected in. For example, you might want to maintain a client-side index of images and faces in the images. To associate faces with an image, specify an image ID in the `ExternalImageId` request parameter. The image ID can be the file name or another ID that you create.

In addition to the preceding information that the API persists in the face collection, the API also returns face details that aren't 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.

Filtering Faces

The IndexFaces operation enables you to filter the faces that are indexed from an image. With `IndexFaces` you can specify a maximum number of faces to index, or you can choose to only index faces detected with a high quality.

You can specify the maximum number of faces that are indexed by `IndexFaces` by using the `MaxFaces` input parameter. This is useful when you want to index the largest faces in an image and don't want to index smaller faces, such as faces of people standing in the background.

By default, `IndexFaces` filters out faces that are detected with low quality. You can explicitly choose to filter out low quality faces by specifying `AUTO` for the value of the `QualityFilter` input parameter. If you don't want to filter faces based on quality, specify `NONE` for `QualityFilter`. `IndexFaces` filters low-quality faces for the following reasons:

- The face is too small compared to the image dimensions.
- The face is too blurry.
- The image is too dark.
- The face has an extreme pose.

Note
To use quality filtering, you need a collection that's associated with version 3 of the face model. To get the version of the face model associated with a collection, call the section called "DescribeCollection" (p. 254).

Information about faces that aren't indexed by `IndexFaces` is returned in an array of the section called "UnindexedFace" (p. 416) objects. The `Reasons` array contains a list of reasons why a face isn't indexed. For example, a value of `EXCEEDS_MAX_FACES` is a face that's not indexed because the number of faces specified by `MaxFaces` has already been detected.

For more information, see Managing Faces in a Collection (p. 137).

To add faces to a collection (SDK)

1. If you haven't already:
   a. Create or update an IAM user with `AmazonRekognitionFullAccess` and `AmazonS3ReadOnlyAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).
2. Upload an image (containing one or more faces) to your Amazon S3 bucket.

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

3. Use the following examples to call the `IndexFaces` operation.

   Java

   This example displays the face identifiers for faces added to the collection.

   Change the value of `collectionId` to the name of the collection that you want to add a face to. Replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and image that you used in step 2. The `.withMaxFaces(1)` parameter restricts the number of indexed faces to 1. Remove or change its value to suit your needs.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;

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;
import com.amazonaws.services.rekognition.model.UnindexedFace;
import java.util.List;

public class AddFacesToCollection {
    public static final String collectionId = "MyCollection";
    public static final String bucket = "bucket";
    public static final String photo = "input.jpg";

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

        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

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

        IndexFacesRequest indexFacesRequest = new IndexFacesRequest()
                .withImage(image)
                .withQualityFilter(QualityFilter.AUTO)
                .withMaxFaces(1)
                .withCollectionId(collectionId)
                .withExternalImageId(photo)
                .withDetectionAttributes("DEFAULT");

        IndexFacesResult indexFacesResult =
 rekognitionClient.indexFaces(indexFacesRequest);

        System.out.println("Results for " + photo);
        System.out.println("Faces indexed:");
        List<FaceRecord> faceRecords = indexFacesResult.getFaceRecords();
        for (FaceRecord faceRecord : faceRecords) {
            System.out.println("  Face ID: " + faceRecord.getFace().getFaceId());
            System.out.println("  Location:" +
 faceRecord.getFaceDetail().getBoundingBox().toString());
        }
```

```
        List<UnindexedFace> unindexedFaces = indexFacesResult.getUnindexedFaces();
        System.out.println("Faces not indexed:");
        for (UnindexedFace unindexedFace : unindexedFaces) {
            System.out.println("  Location:" +
 unindexedFace.getFaceDetail().getBoundingBox().toString());
            System.out.println("  Reasons:");
            for (String reason : unindexedFace.getReasons()) {
                System.out.println("    " + reason);
            }
        }
    }
}
```

AWS CLI

This AWS CLI command displays the JSON output for the `index-faces` CLI operation.

Replace the value of `collection-id` with the name of the collection you want the face to be stored in. Replace the values of `Bucket` and `Name` with the Amazon S3 bucket and image file that you used in step 2. The `max-faces` parameter restricts the number of indexed faces to 1. Remove or change its value to suit your needs.

```
aws rekognition index-faces \
      --image '{"S3Object":{"Bucket":"bucket-name","Name":"file-name"}}' \
      --collection-id "collection-id" \
      --max-faces 1 \
      --quality-filter "AUTO" \
      --detection-attributes "ALL" \
      --external-image-id "example-image.jpg"
```

Python

This example displays the face identifiers for faces added to the collection.

Change the value of `collectionId` to the name of the collection that you want to add a face to. Replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and image that you used in step 2. The `MaxFaces` input parameter restricts the number of indexed faces to 1. Remove or change its value to suit your needs.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

if __name__ == "__main__":

    bucket='bucket'
    collectionId='MyCollection'
    photo='photo'

    client=boto3.client('rekognition')

    response=client.index_faces

    response=client.index_faces(CollectionId=collectionId,
                                Image={'S3Object':{'Bucket':bucket,'Name':photo}},
                                ExternalImageId=photo,
                                MaxFaces=1,
                                QualityFilter="AUTO",
                                DetectionAttributes=['ALL'])
```

```
    print ('Results for ' + photo)
    print('Faces indexed:')
    for faceRecord in response['FaceRecords']:
         print('  Face ID: ' + faceRecord['Face']['FaceId'])
         print('  Location: {}'.format(faceRecord['Face']['BoundingBox']))

    print('Faces not indexed:')
    for unindexedFace in response['UnindexedFaces']:
        print(' Location: {}'.format(unindexedFace['FaceDetail']['BoundingBox']))
        print(' Reasons:')
        for reason in unindexedFace['Reasons']:
            print('   ' + reason)
```

.NET

This example displays the face identifiers for faces added to the collection.

Change the value of `collectionId` to the name of the collection that you want to add a face to. Replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and image that you used in step 2.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using System.Collections.Generic;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class AddFaces
{
    public static void Example()
    {
        String collectionId = "MyCollection";
        String bucket = "bucket";
        String photo = "input.jpg";

        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        Image image = new Image()
        {
            S3Object = new S3Object()
            {
                Bucket = bucket,
                Name = photo
            }
        };

        IndexFacesRequest indexFacesRequest = new IndexFacesRequest()
        {
            Image = image,
            CollectionId = collectionId,
            ExternalImageId = photo,
            DetectionAttributes = new List<String>(){ "ALL" }
        };

        IndexFacesResponse indexFacesResponse =
 rekognitionClient.IndexFaces(indexFacesRequest);

        Console.WriteLine(photo + " added");
        foreach (FaceRecord faceRecord in indexFacesResponse.FaceRecords)
```

```
            Console.WriteLine("Face detected: Faceid is " +
               faceRecord.Face.FaceId);
        }
    }
```

IndexFaces Operation Request

The input to `IndexFaces` is the image to be indexed and the collection to add the face or faces to.

```
{
    "CollectionId": "MyCollection",
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "ExternalImageId": "input.jpg",
    "DetectionAttributes": [
        "DEFAULT"
    ],
    "MaxFaces": 1,
    "QualityFilter": "AUTO"
}
```

IndexFaces Operation Response

`IndexFaces` returns information about the faces that were detected in the image. For example, the following JSON response includes the default detection attributes for faces detected in the input image. The example also shows faces not indexed because the value of the `MaxFaces` input parameter has been exceeded — the `Reasons` array contains *EXCEEDS_MAX_FACES*. If a face is not indexed for quality reasons, `Reasons` contains values such as *LOW_SHARPNESS* or *LOW_BRIGHTNESS*. For more information, see the section called “UnindexedFace” (p. 416).

```
{
    "FaceModelVersion": "3.0",
    "FaceRecords": [
        {
            "Face": {
                "BoundingBox": {
                    "Height": 0.3247932195663452,
                    "Left": 0.5055555701255798,
                    "Top": 0.2743072211742401,
                    "Width": 0.21444444358348846
                },
                "Confidence": 99.99998474121094,
                "ExternalImageId": "input.jpg",
                "FaceId": "b86e2392-9da1-459b-af68-49118dc16f87",
                "ImageId": "09f43d92-02b6-5cea-8fbd-9f187db2050d"
            },
            "FaceDetail": {
                "BoundingBox": {
                    "Height": 0.3247932195663452,
                    "Left": 0.5055555701255798,
                    "Top": 0.2743072211742401,
                    "Width": 0.21444444358348846
                },
                "Confidence": 99.99998474121094,
                "Landmarks": [
                    {
```

```
                        "Type": "eyeLeft",
                        "X": 0.5751981735229492,
                        "Y": 0.4010535478591919
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.6511467099189758,
                        "Y": 0.4017036259174347
                    },
                    {
                        "Type": "nose",
                        "X": 0.6314528584480286,
                        "Y": 0.4710812568664551
                    },
                    {
                        "Type": "mouthLeft",
                        "X": 0.5879443287849426,
                        "Y": 0.5171778798103333
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.6444502472877502,
                        "Y": 0.5164633989334106
                    }
                ],
                "Pose": {
                    "Pitch": -10.313642501831055,
                    "Roll": -1.0316886901855469,
                    "Yaw": 18.079818725585938
                },
                "Quality": {
                    "Brightness": 71.2919921875,
                    "Sharpness": 78.74752044677734
                }
            }
        }
    ],
    "OrientationCorrection": "ROTATE_0",
    "UnindexedFaces": [
        {
            "FaceDetail": {
                "BoundingBox": {
                    "Height": 0.1329464465379715,
                    "Left": 0.5611110925674438,
                    "Top": 0.6832437515258789,
                    "Width": 0.08777777850627899
                },
                "Confidence": 92.37225341796875,
                "Landmarks": [
                    {
                        "Type": "eyeLeft",
                        "X": 0.5796897411346436,
                        "Y": 0.7452847957611084
                    },
                    {
                        "Type": "eyeRight",
                        "X": 0.6078574657440186,
                        "Y": 0.742687463760376
                    },
                    {
                        "Type": "nose",
                        "X": 0.597953200340271,
                        "Y": 0.7620673179626465
                    },
                    {
                        "Type": "mouthLeft",
```

```
                        "X": 0.5884202122688293,
                        "Y": 0.7920381426811218
                    },
                    {
                        "Type": "mouthRight",
                        "X": 0.60627681016922,
                        "Y": 0.7919750809669495
                    }
                ],
                "Pose": {
                    "Pitch": 15.658954620361328,
                    "Roll": -4.583454608917236,
                    "Yaw": 10.558992385864258
                },
                "Quality": {
                    "Brightness": 42.54612350463867,
                    "Sharpness": 86.93206024169922
                }
            },
            "Reasons": [
                "EXCEEDS_MAX_FACES"
            ]
        }
    ]
}
```

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 isn't persisted on the server:

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

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

`FaceModelVersion` is the version of the face model that's associated with the collection. For more information, see Model Versioning (p. 9).

`OrientationCorrection` is the estimated orientation of the image. For more information, see Getting Image Orientation and Bounding Box Coordinates (p. 49).

```
{
    "FaceModelVersion": "3.0",
    "FaceRecords": [
        {
            "Face": {
                "BoundingBox": {
                    "Height": 0.06333333253860474,
                    "Left": 0.17185185849666595,
                    "Top": 0.7366666793823242,
                    "Width": 0.11061728745698929
                },
                "Confidence": 99.99999237060547,
                "ExternalImageId": "input.jpg",
                "FaceId": "578e2e1b-d0b0-493c-aa39-ba476a421a34",
                "ImageId": "9ba38e68-35b6-5509-9d2e-fcffa75d1653"
            },
            "FaceDetail": {
                "AgeRange": {
                    "High": 25,
                    "Low": 15
```

```
        },
        "Beard": {
            "Confidence": 99.98077392578125,
            "Value": false
        },
        "BoundingBox": {
            "Height": 0.06333333253860474,
            "Left": 0.17185185849666595,
            "Top": 0.7366666793823242,
            "Width": 0.11061728745698929
        },
        "Confidence": 99.99999237060547,
        "Emotions": [
            {
                "Confidence": 95.40877532958984,
                "Type": "HAPPY"
            },
            {
                "Confidence": 6.6088080406188965,
                "Type": "CALM"
            },
            {
                "Confidence": 0.7385611534118652,
                "Type": "SAD"
            }
        ],
        "Eyeglasses": {
            "Confidence": 99.96795654296875,
            "Value": false
        },
        "EyesOpen": {
            "Confidence": 64.0671157836914,
            "Value": true
        },
        "Gender": {
            "Confidence": 100,
            "Value": "Female"
        },
        "Landmarks": [
            {
                "Type": "eyeLeft",
                "X": 0.21361233294010162,
                "Y": 0.757106363773346
            },
            {
                "Type": "eyeRight",
                "X": 0.2518567442893982,
                "Y": 0.7599404454231262
            },
            {
                "Type": "nose",
                "X": 0.2262365221977234,
                "Y": 0.7711842060089111
            },
            {
                "Type": "mouthLeft",
                "X": 0.2050037682056427,
                "Y": 0.7801263332366943
            },
            {
                "Type": "mouthRight",
                "X": 0.2430567592382431,
                "Y": 0.7836716771125793
            },
            {
                "Type": "leftPupil",
```

```
            "X": 0.2161938101053238,
            "Y": 0.756662905216217
        },
        {
            "Type": "rightPupil",
            "X": 0.2523181438446045,
            "Y": 0.7603650689125061
        },
        {
            "Type": "leftEyeBrowLeft",
            "X": 0.20066319406032562,
            "Y": 0.7501518130302429
        },
        {
            "Type": "leftEyeBrowUp",
            "X": 0.2130996286869049,
            "Y": 0.7480520606040955
        },
        {
            "Type": "leftEyeBrowRight",
            "X": 0.22584207355976105,
            "Y": 0.7504606246948242
        },
        {
            "Type": "rightEyeBrowLeft",
            "X": 0.24509544670581818,
            "Y": 0.7526801824569702
        },
        {
            "Type": "rightEyeBrowUp",
            "X": 0.2582615911960602,
            "Y": 0.7516844868659973
        },
        {
            "Type": "rightEyeBrowRight",
            "X": 0.26881539821624756,
            "Y": 0.7554477453231812
        },
        {
            "Type": "leftEyeLeft",
            "X": 0.20624476671218872,
            "Y": 0.7568746209144592
        },
        {
            "Type": "leftEyeRight",
            "X": 0.22105035185813904,
            "Y": 0.7582521438598633
        },
        {
            "Type": "leftEyeUp",
            "X": 0.21401576697826385,
            "Y": 0.7553104162216187
        },
        {
            "Type": "leftEyeDown",
            "X": 0.21317370235919952,
            "Y": 0.7584449648857117
        },
        {
            "Type": "rightEyeLeft",
            "X": 0.24393919110298157,
            "Y": 0.7600628137588501
        },
        {
            "Type": "rightEyeRight",
            "X": 0.2598416209220886,
```

```
                        "Y": 0.7605880498886108
                    },
                    {
                        "Type": "rightEyeUp",
                        "X": 0.2519053518772125,
                        "Y": 0.7582084536552429
                    },
                    {
                        "Type": "rightEyeDown",
                        "X": 0.25177454948425293,
                        "Y": 0.7612871527671814
                    },
                    {
                        "Type": "noseLeft",
                        "X": 0.2185886949300766,
                        "Y": 0.774715781211853
                    },
                    {
                        "Type": "noseRight",
                        "X": 0.23328955471515656,
                        "Y": 0.7759330868721008
                    },
                    {
                        "Type": "mouthUp",
                        "X": 0.22446128726005554,
                        "Y": 0.7805567383766174
                    },
                    {
                        "Type": "mouthDown",
                        "X": 0.22087252140045166,
                        "Y": 0.7891407608985901
                    }
                ],
                "MouthOpen": {
                    "Confidence": 95.87068939208984,
                    "Value": false
                },
                "Mustache": {
                    "Confidence": 99.9828109741211,
                    "Value": false
                },
                "Pose": {
                    "Pitch": -0.9409101605415344,
                    "Roll": 7.233824253082275,
                    "Yaw": -2.3602254390716553
                },
                "Quality": {
                    "Brightness": 32.01998519897461,
                    "Sharpness": 93.67259216308594
                },
                "Smile": {
                    "Confidence": 86.7142105102539,
                    "Value": true
                },
                "Sunglasses": {
                    "Confidence": 97.38925170898438,
                    "Value": false
                }
            }
        }
    ],
    "OrientationCorrection": "ROTATE_0"
    "UnindexedFaces": []
}
```

Listing Faces in a Collection

You can use the ListFaces (p. 316) operation to list the faces in a collection.

For more information, see Managing Faces in a Collection (p. 137).

To list faces in a collection (SDK)

1. If you haven't already:
   a. Create or update an IAM user with `AmazonRekognitionFullAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).
2. Use the following examples to call the `ListFaces` operation.

   Java

   This example displays a list of faces in a collection.

   Change the value of `collectionId` to the desired collection.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
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 java.util.List;
import com.fasterxml.jackson.databind.ObjectMapper;



public class ListFacesInCollection {
    public static final String collectionId = "MyCollection";

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

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      ObjectMapper objectMapper = new ObjectMapper();

      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 =  rekognitionClient.listFaces(listFacesRequest);
```

```
         List < Face > faces = listFacesResult.getFaces();
         for (Face face: faces) {
            System.out.println(objectMapper.writerWithDefaultPrettyPrinter()
               .writeValueAsString(face));
         }
      } while (listFacesResult != null && listFacesResult.getNextToken() !=
         null);
   }

}
```

AWS CLI

This AWS CLI command displays the JSON output for the `list-faces` CLI operation. Replace the value of `collection-id` with the name of the collection you want to list.

```
aws rekognition list-faces \
      --collection-id "collection-id"
```

Python

This example displays a list of faces in a collection.

Change the value of `collectionId` to the desired collection.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

if __name__ == "__main__":

    bucket='bucket'
    collectionId='MyCollection'
    maxResults=2
    tokens=True

    client=boto3.client('rekognition')
    response=client.list_faces(CollectionId=collectionId,
                               MaxResults=maxResults)

    print('Faces in collection ' + collectionId)


    while tokens:

        faces=response['Faces']

        for face in faces:
            print (face)
        if 'NextToken' in response:
            nextToken=response['NextToken']
            response=client.list_faces(CollectionId=collectionId,
                                       NextToken=nextToken,MaxResults=maxResults)
        else:
            tokens=False
```

.NET

This example displays a list of faces in a collection.

Change the value of `collectionId` to the desired collection.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class ListFaces
{
    public static void Example()
    {
        String collectionId = "MyCollection";

        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        ListFacesResponse listFacesResponse = null;
        Console.WriteLine("Faces in collection " + collectionId);

        String paginationToken = null;
        do
        {
            if (listFacesResponse != null)
                paginationToken = listFacesResponse.NextToken;

            ListFacesRequest listFacesRequest = new ListFacesRequest()
            {
                CollectionId = collectionId,
                MaxResults = 1,
                NextToken = paginationToken
            };

            listFacesResponse = rekognitionClient.ListFaces(listFacesRequest);
            foreach(Face face in listFacesResponse.Faces)
                Console.WriteLine(face.FaceId);
        } while (listFacesResponse != null && !
String.IsNullOrEmpty(listFacesResponse.NextToken));
    }
}
```

ListFaces Operation Request

The input to `ListFaces` is the ID of the collection that you want to list faces for. `MaxResults` is the maximum number of faces to return.

```
{
    "CollectionId": "MyCollection",
    "MaxResults": 1
}
```

If the response has more faces than are requested by `MaxResults`, a token is returned that you can use to get the next set of results, in a subsequent call to `ListFaces`. For example:

```
{
    "CollectionId": "MyCollection",
    "NextToken": "sm+5ythT3aeEVIR4WA....",
    "MaxResults": 1
```

```
}
```

ListFaces Operation Response

The response from `ListFaces` is information about the face metadata that's stored in the specified collection.

- **FaceModelVersion** – The version of the face model that's associated with the collection. For more information, see Model Versioning (p. 9).
- **Faces** – Information about the faces in the collection. This includes information about the section called "BoundingBox" (p. 363), confidence, image identifiers, and the face ID. For more information, see the section called "Face" (p. 377).
- **NextToken** – The token that's used to get the next set of results.

```
{
    "FaceModelVersion": "3.0",
    "Faces": [
        {
            "BoundingBox": {
                "Height": 0.06333330273628235,
                "Left": 0.1718519926071167,
                "Top": 0.7366669774055481,
                "Width": 0.11061699688434601
            },
            "Confidence": 100,
            "ExternalImageId": "input.jpg",
            "FaceId": "0b683aed-a0f1-48b2-9b5e-139e9cc2a757",
            "ImageId": "9ba38e68-35b6-5509-9d2e-fcffa75d1653"
        }
    ],
    "NextToken": "sm+5ythT3aeEVIR4WA...."
}
```

Deleting Faces from a Collection

You can use the DeleteFaces (p. 249) operation to delete faces from a collection. For more information, see Managing Faces in a Collection (p. 137).

To delete faces from a collection

1. If you haven't already:
   a. Create or update an IAM user with `AmazonRekognitionFullAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).
2. Use the following examples to call the `DeleteCollection` operation.

   Java

   This example deletes a single face from a collection.

   Change the value of `collectionId` to the collection that contains the face that you want to delete. Change the value of `faces` to the ID of the face that you want to delete. To delete multiple faces, add the face IDs to the `faces` array.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.DeleteFacesRequest;
import com.amazonaws.services.rekognition.model.DeleteFacesResult;

import java.util.List;


public class DeleteFacesFromCollection {
   public static final String collectionId = "MyCollection";
   public static final String faces[] = {"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"};

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

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();


      DeleteFacesRequest deleteFacesRequest = new DeleteFacesRequest()
              .withCollectionId(collectionId)
              .withFaceIds(faces);

      DeleteFacesResult
 deleteFacesResult=rekognitionClient.deleteFaces(deleteFacesRequest);


      List < String > faceRecords = deleteFacesResult.getDeletedFaces();
      System.out.println(Integer.toString(faceRecords.size()) + " face(s)
 deleted:");
      for (String face: faceRecords) {
         System.out.println("FaceID: " + face);
      }
   }
}
```

AWS CLI

This AWS CLI command displays the JSON output for the `delete-collection` CLI operation. Replace the value of `collection-id` with the name of the collection that contains the face you want to delete. Replace the value of `face-ids` with an array of face IDs that you want to delete.

```
aws rekognition delete-faces --collection-id "collectionname" --face-ids
 '["faceid"]'
```

Python

This example deletes a single face from a collection.

Change the value of `collectionId` to the collection that contains the face that you want to delete. Change the value of `faces` to the ID of the face that you want to delete. To delete multiple faces, add the face IDs to the `faces` array.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

if __name__ == "__main__":

    collectionId='MyCollection'
    faces=[]
    faces.append("xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")

    client=boto3.client('rekognition')

    response=client.delete_faces(CollectionId=collectionId,
                                 FaceIds=faces)

    print(str(len(response['DeletedFaces'])) + ' faces deleted:')
    for faceId in response['DeletedFaces']:
         print (faceId)
```

.NET

This example deletes a single face from a collection.

Change the value of `collectionId` to the collection that contains the face that you want to delete. Change the value of `faces` to the ID of the face that you want to delete. To delete multiple faces, add the face IDs to the `faces` list.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using System.Collections.Generic;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DeleteFaces
{
    public static void Example()
    {
        String collectionId = "MyCollection";
        List<String> faces = new List<String>() { "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx" };

        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        DeleteFacesRequest deleteFacesRequest = new DeleteFacesRequest()
        {
            CollectionId = collectionId,
            FaceIds = faces
        };

        DeleteFacesResponse deleteFacesResponse =
 rekognitionClient.DeleteFaces(deleteFacesRequest);
        foreach (String face in deleteFacesResponse.DeletedFaces)
            Console.WriteLine("FaceID: " + face);
```

```
    }
}
```

DeleteFaces Operation Request

The input to `DeleteFaces` is the ID of the collection that contains the faces, and an array of face IDs for the faces to be deleted.

```
{
    "CollectionId": "MyCollection",
    "FaceIds": [
        "daf29cac-f910-41e9-851f-6eeb0e08f973"
    ]
}
```

DeleteFaces Operation Response

The `DeleteFaces` response contains an array of face IDs for the faces that were deleted.

```
{
    "DeletedFaces": [
        "daf29cac-f910-41e9-851f-6eeb0e08f973"
    ]
}
```

Searching for a Face Using Its Face ID

You can use the SearchFaces (p. 326) operation to search for faces in a collection that match a supplied face ID.

The face ID is returned in the IndexFaces (p. 305) operation response when the face is detected and added to a collection. For more information, see Managing Faces in a Collection (p. 137).

To search for a face in a collection using its face ID (SDK)

1. If you haven't already:
   a. Create or update an IAM user with `AmazonRekognitionFullAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).
2. Use the following examples to call the `SearchFaces` operation.

   Java

   This example displays information about faces that match a face identified by its ID.

   Change the value of `collectionID` to the collection that contains the required face. Change the value of `faceId` to the identifier of the face you want to find.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
```

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.amazonaws.services.rekognition.model.FaceMatch;
import com.amazonaws.services.rekognition.model.SearchFacesRequest;
import com.amazonaws.services.rekognition.model.SearchFacesResult;
import java.util.List;


  public class SearchFaceMatchingIdCollection {
      public static final String collectionId = "MyCollection";
      public static final String faceId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

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

        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        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 searchFacesByIdResult =
               rekognitionClient.searchFaces(searchFacesRequest);

       System.out.println("Face matching faceId " + faceId);
      List < FaceMatch > faceImageMatches =
 searchFacesByIdResult.getFaceMatches();
      for (FaceMatch face: faceImageMatches) {
         System.out.println(objectMapper.writerWithDefaultPrettyPrinter()
                 .writeValueAsString(face));

         System.out.println();
      }
    }

}
```

Run the example code. Information about matching faces is displayed.

AWS CLI

This AWS CLI command displays the JSON output for the `search-faces` CLI operation. Replace the value of `face-id` with the face identifier that you want to search for, and replace the value of `collection-id` with the collection you want to search in.

```
aws rekognition search-faces \
    --face-id face-id \
    --collection-id "collection-id"
```

Python

This example displays information about faces that match a face identified by its ID.

Change the value of `collectionID` to the collection that contains the required face. Change the value of `faceId` to the identifier of the face you want to find.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

if __name__ == "__main__":

    bucket='bucket'
    collectionId='MyCollection'
    threshold = 50
    maxFaces=2
    faceId='xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'

    client=boto3.client('rekognition')


    response=client.search_faces(CollectionId=collectionId,
                                 FaceId=faceId,
                                 FaceMatchThreshold=threshold,
                                 MaxFaces=maxFaces)


    faceMatches=response['FaceMatches']
    print 'Matching faces'
    for match in faceMatches:
            print 'FaceId:' + match['Face']['FaceId']
            print 'Similarity: ' + "{:.2f}".format(match['Similarity']) + "%"
            print
```

.NET

This example displays information about faces that match a face identified by its ID.

Change the value of `collectionID` to the collection that contains the required face. Change the value of `faceId` to the identifier of the face that you want to find.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class SearchFacesMatchingId
{
    public static void Example()
    {
        String collectionId = "MyCollection";
        String faceId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        // Search collection for faces matching the face id.

        SearchFacesRequest searchFacesRequest = new SearchFacesRequest()
        {
            CollectionId = collectionId,
            FaceId = faceId,
            FaceMatchThreshold = 70F,
```

```
            MaxFaces = 2
        };

        SearchFacesResponse searchFacesResponse =
 rekognitionClient.SearchFaces(searchFacesRequest);

        Console.WriteLine("Face matching faceId " + faceId);

        Console.WriteLine("Matche(s): ");
        foreach (FaceMatch face in searchFacesResponse.FaceMatches)
            Console.WriteLine("FaceId: " + face.Face.FaceId + ", Similarity: " +
 face.Similarity);
    }
}
```

Run the example code. Information about matching faces is displayed.

SearchFaces Operation Request

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. By default, `SearchFaces` returns 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.

```
{
    "CollectionId": "MyCollection",
    "FaceId": "0b683aed-a0f1-48b2-9b5e-139e9cc2a757",
    "MaxFaces": 2,
    "FaceMatchThreshold": 70
}
```

SearchFaces Operation Response

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

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

For each face match that was found, the response includes similarity and face metadata, as shown in the following example response:

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

```
            }
        },
        {
            "Similarity": 84.6859,
            "Face": {
                "BoundingBox": {
                    "Width": 0.2044,
                    "Top": 0.2254,
                    "Left": 0.4622,
                    "Height": 0.3119
                },
                "FaceId": "6fc892c7-5739-50da-a0d7-80cc92c0ba54",
                "Confidence": 99.9981,
                "ImageId": "5d913eaf-cf7f-5e09-8c8f-cb1bdea8e6aa"
            }
        }
    ]
}
```

Searching for a Face Using an Image

You can use the SearchFacesByImage (p. 329) operation to search for faces in a collection that match the largest face in a supplied image.

For more information, see Searching for Faces Within a Collection (p. 138).

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

1. If you haven't already:
   a. Create or update an IAM user with `AmazonRekognitionFullAccess` and `AmazonS3ReadOnlyAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).
2. Upload an image (that contains 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*.
3. Use the following examples to call the `SearchFacesByImage` operation.

   Java

   This example displays information about faces that match the largest face in an image. The code example specifies both the `FaceMatchThreshold` and `MaxFaces` parameters to limit the results that are returned in the response.

   In the following example, change the following: change the value of `collectionId` to the collection you want to search, change the value of `bucket` to the bucket containing the input image, and change the value of `photo` to the input image.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
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 java.util.List;
import com.fasterxml.jackson.databind.ObjectMapper;


public class SearchFaceMatchingImageCollection {
    public static final String collectionId = "MyCollection";
    public static final String bucket = "bucket";
    public static final String photo = "input.jpg";

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

       AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      ObjectMapper objectMapper = new ObjectMapper();

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

      // 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 =
               rekognitionClient.searchFacesByImage(searchFacesByImageRequest);

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

AWS CLI

This AWS CLI command displays the JSON output for the `search-faces-by-image` CLI operation. Replace the value of `Bucket` with the S3 bucket that you used in step 2. Replace the value of `Name` with the image file name that you used in step 2. Replace the value of `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"
```

Python

This example displays information about faces that match the largest face in an image. The code example specifies both the `FaceMatchThreshold` and `MaxFaces` parameters to limit the results that are returned in the response.

In the following example, change the following: change the value of `collectionId` to the collection you want to search, and replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and image that you used in Step 2.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

if __name__ == "__main__":

    bucket='bucket'
    collectionId='MyCollection'
    fileName='input.jpg'
    threshold = 70
    maxFaces=2

    client=boto3.client('rekognition')


    response=client.search_faces_by_image(CollectionId=collectionId,
                                Image={'S3Object':
{'Bucket':bucket,'Name':fileName}},
                                FaceMatchThreshold=threshold,
                                MaxFaces=maxFaces)


    faceMatches=response['FaceMatches']
    print 'Matching faces'
    for match in faceMatches:
            print 'FaceId:' + match['Face']['FaceId']
            print 'Similarity: ' + "{:.2f}".format(match['Similarity']) + "%"
            print
```

.NET

This example displays information about faces that match the largest face in an image. The code example specifies both the `FaceMatchThreshold` and `MaxFaces` parameters to limit the results that are returned in the response.

In the following example, change the following: change the value of `collectionId` to the collection you want to search, and replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and image that you used in step 2.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class SearchFacesMatchingImage
```

```
{
    public static void Example()
    {
        String collectionId = "MyCollection";
        String bucket = "bucket";
        String photo = "input.jpg";

        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        // Get an image object from S3 bucket.
        Image image = new Image()
        {
            S3Object = new S3Object()
            {
                Bucket = bucket,
                Name = photo
            }
        };

        SearchFacesByImageRequest searchFacesByImageRequest = new
 SearchFacesByImageRequest()
        {
            CollectionId = collectionId,
            Image = image,
            FaceMatchThreshold = 70F,
            MaxFaces = 2
        };

        SearchFacesByImageResponse searchFacesByImageResponse =
 rekognitionClient.SearchFacesByImage(searchFacesByImageRequest);

        Console.WriteLine("Faces matching largest face in image from " + photo);
        foreach (FaceMatch face in searchFacesByImageResponse.FaceMatches)
            Console.WriteLine("FaceId: " + face.Face.FaceId + ", Similarity: " +
 face.Similarity);
    }
}
```

SearchFacesByImage Operation Request

The input parameters to `SearchFacesImageByImage` are the collection to search in and the source image location. In this example, the source image is stored in an Amazon S3 bucket (`S3Object`). Also specified are the maximum number of faces to return (`Maxfaces`) and the minimum confidence that must be matched for a face to be returned (`FaceMatchThreshold`).

```
{
    "CollectionId": "MyCollection",
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "MaxFaces": 2,
    "FaceMatchThreshold": 70
}
```

SearchFacesByImage Operation Response

Given an 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 that's detected for searching the face collection.

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

By default, `SearchFacesByImage` returns 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": [
        {
            "Face": {
                "BoundingBox": {
                    "Height": 0.06333330273628235,
                    "Left": 0.1718519926071167,
                    "Top": 0.7366669774055481,
                    "Width": 0.11061699688434601
                },
                "Confidence": 100,
                "ExternalImageId": "input.jpg",
                "FaceId": "578e2e1b-d0b0-493c-aa39-ba476a421a34",
                "ImageId": "9ba38e68-35b6-5509-9d2e-fcffa75d1653"
            },
            "Similarity": 99.9764175415039
        }
    ],
    "FaceModelVersion": "3.0",
    "SearchedFaceBoundingBox": {
        "Height": 0.06333333253860474,
        "Left": 0.17185185849666595,
        "Top": 0.7366666793823242,
        "Width": 0.11061728745698929
    },
    "SearchedFaceConfidence": 99.99999237060547
}
```

Searching Stored Videos for Faces

You can search a collection for faces that match faces of people who are 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. 86).

The faces that you search for must first be indexed into a collection by using IndexFaces (p. 305). For more information, see Adding Faces to a Collection (p. 151).

Amazon Rekognition Video face searching follows the same asynchronous workflow as other Amazon Rekognition Video operations that analyze videos stored in an Amazon S3 bucket. To start searching for faces in a stored video, call StartFaceSearch (p. 345) and provide the ID of the collection that you want to search. Amazon Rekognition Video publishes the completion status of the video analysis to an Amazon Simple Notification Service (Amazon SNS) topic. If the video analysis is successful, call

GetFaceSearch (p. 291) to get the search results. For more information about starting video analysis and getting the results, see Calling Amazon Rekognition Video Operations (p. 61).

The following procedure shows how to search a collection for faces that match the faces of people who are detected in a video. The procedure also shows how to get the tracking data for people who are matched in the video. The procedure expands on the code in Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68), which uses an Amazon Simple Queue Service (Amazon SQS) queue to get the completion status of a video analysis request.

To search a video for matching faces (SDK)

1. Create a collection (p. 139).
2. Index a face into the collection (p. 151).
3. Perform Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68).
4. Add the following code to the class `VideoDetect` that you created in step 3.

   Java

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    //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();

    }

    //Face collection search in video
 =========================================================================
    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 milliSeconds=match.getTimestamp();
                System.out.print("Timestamp: " + Long.toString(milliSeconds));
                System.out.println(" Person number: " +
match.getPerson().getIndex());
                List <FaceMatch> faceMatches = match.getFaceMatches();
                if (faceMatches != null) {
                    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();
            }

            System.out.println();

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

        }
```

4a. In the function `main`, replace the line:

```
StartLabels(bucket,video);
```

with

```
StartFaceSearchCollection(bucket,video);
```

4b. Replace the line:

```
GetResultsLabels();
```

with:

```
GetResultsFaceSearchCollection();
```

Python

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    def GetResultsFaceSearchCollection(self, jobId):
        maxResults = 10
        paginationToken = ''

        finished = False

        while finished == False:
            response = self.rek.get_face_search(JobId=jobId,
                                        MaxResults=maxResults,
                                        NextToken=paginationToken)

            print(response['VideoMetadata']['Codec'])
            print(str(response['VideoMetadata']['DurationMillis']))
            print(response['VideoMetadata']['Format'])
            print(response['VideoMetadata']['FrameRate'])

            for personMatch in response['Persons']:
                print('Person Index: ' + str(personMatch['Person']['Index']))
                print('Timestamp: ' + str(personMatch['Timestamp']))

                if ('FaceMatches' in personMatch):
                    for faceMatch in personMatch['FaceMatches']:
                        print('Face ID: ' + faceMatch['Face']['FaceId'])
                        print('Similarity: ' + str(faceMatch['Similarity']))
                print()
            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True
            print()
```

4a. In the function `main`, replace the line:

```
       response = self.rek.start_label_detection(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
                                          NotificationChannel={'RoleArn':
 self.roleArn, 'SNSTopicArn': self.topicArn})
```

with:

```
       response = self.rek.start_face_search(Video={'S3Object':
{'Bucket':self.bucket,'Name':self.video}},
            CollectionId='CollectionId',
            NotificationChannel={'RoleArn':self.roleArn,
 'SNSTopicArn':self.topicArn})
```

4b. Replace the line:

```
                        self.GetResultsLabels(rekMessage['JobId'])
```

with:

```
                        self.GetResultsFaceSearchCollection(rekMessage['JobId'])
```

If you've already run a video example other than Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68), the function name to replace is different.

5. Change the value of `CollectionId` to the name of the collection you created in step 1.
6. Run the code. A list of people in the video whose faces match those in the input collection is displayed. The tracking data for each matched person is also displayed.

GetFaceSearch Operation Response

The following is an example JSON response from `GetFaceSearch`.

The response includes an array of people (`Persons`) detected in the video whose faces match a face in the input collection. An array element, PersonMatch (p. 404), 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. 383), information about the matched person, PersonDetail (p. 402), and the time the person was matched in the video.


```
{
    "JobStatus": "SUCCEEDED",
    "NextToken": "IJdbzkZfvBRqj8GPV82BPiZKkLOGCqDIsNZG/gQsEE5faTVK9JHOz/xxxxxxxxxxxxxxx",
    "Persons": [
        {
            "FaceMatches": [
                {
                    "Face": {
                        "BoundingBox": {
                            "Height": 0.527472972869873,
                            "Left": 0.33530598878860474,
                            "Top": 0.2161169946193695,
                            "Width": 0.35503000020980835
                        },
                        "Confidence": 99.90239715576172,
                        "ExternalImageId": "image.PNG",
                        "FaceId": "a2f2e224-bfaa-456c-b360-7c00241e5e2d",
                        "ImageId": "eb57ed44-8d8d-5ec5-90b8-6d190daff4c3"
                    },
                    "Similarity": 98.40909576416016
                }
            ],
            "Person": {
                "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": 0.49765899777412415,
                        "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
                }
            },
            "Index": 0
        },
        "Timestamp": 0
    },
    {
        "Person": {
            "BoundingBox": {
                "Height": 0.2177777737379074,
                "Left": 0.7593749761581421,
                "Top": 0.13333334028720856,
                "Width": 0.12250000238418579
            },
            "Face": {
                "BoundingBox": {
                    "Height": 0.2177777737379074,
                    "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,
                        "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": 2
            },
            "Timestamp": 0
        }......

    ],
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 67301,
        "Format": "QuickTime / MOV",
        "FrameHeight": 1080,
        "FrameRate": 29.970029830932617,
        "FrameWidth": 1920
    }
}
```

People Pathing

Amazon Rekognition Video can create a track of the path people take in videos and provide information such as:

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

Amazon Rekognition Video people pathing in stored videos is an asynchronous operation. To start the pathing of people in videos call StartPersonTracking (p. 353). Amazon Rekognition Video publishes the completion status of the video analysis to an Amazon Simple Notification Service topic. If the video analysis is succesful, call GetPersonTracking (p. 300) to get results of the video analysis. For more information about calling Amazon Rekognition Video API operations, see Calling Amazon Rekognition Video Operations (p. 61).

The following procedure shows how to track the path of 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 Java or Python (SDK) (p. 68) which uses an Amazon Simple Queue Service queue to get the completion status of a video analysis request.

To detect people in a video stored in an Amazon S3 bucket (SDK)

1. Perform Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68).
2. Add the following code to the class `VideoDetect` that you created in step 1.

   Java

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    //
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);

  }
```

2a. In the function `main`, replace the line:

```
StartLabels(bucket,video);
```

with:

```
StartPersons(bucket,video);
```

2b. Replace the line:

```
GetResultsLabels();
```

with:

```
GetResultsPersons();
```

Python

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    def GetResultsPersons(self, jobId):
        maxResults = 10
        paginationToken = ''
        finished = False
```

```
        while finished == False:
            response = self.rek.get_person_tracking(JobId=jobId,
                                                    MaxResults=maxResults,
                                                    NextToken=paginationToken)

            print(response['VideoMetadata']['Codec'])
            print(str(response['VideoMetadata']['DurationMillis']))
            print(response['VideoMetadata']['Format'])
            print(response['VideoMetadata']['FrameRate'])

            for personDetection in response['Persons']:
                print('Index: ' + str(personDetection['Person']['Index']))
                print('Timestamp: ' + str(personDetection['Timestamp']))
                print()

            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True
```

2a. In the function `main`, replace the line:

```
        response = self.rek.start_label_detection(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
                                                  NotificationChannel={'RoleArn':
 self.roleArn, 'SNSTopicArn': self.topicArn})
```

with:

```
        response = self.rek.start_person_tracking(Video={'S3Object':
{'Bucket':self.bucket,'Name':self.video}},
            NotificationChannel={'RoleArn':self.roleArn,
 'SNSTopicArn':self.topicArn})
```

2b. Replace the line:

```
                        self.GetResultsLabels(rekMessage['JobId'])
```

with:

```
                        self.GetResultsPersons(rekMessage['JobId'])
```

Note
If you've already run a video example other than Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68), the function name to replace is different.

3. Run the code. The unique identifiers for tracked people are shown along with the time, in seconds, the people's paths were tracked.

GetPersonTracking Operation Response

`GetPersonTracking` returns an array, `Persons`, of PersonDetection (p. 403) objects which contain details about people detected in the video and when their paths are tracked.

You can sort `Persons` by using the `SortBy` input parameter. Specify `TIMESTAMP` to sort the elements by the time people's paths are tracked 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 path 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, that people's paths are tracked in the video. In the response, note the following:

- **Person information** – The `PersonDetection` array element contains information about the detected person. For example, the time the person was detected (`Timestamp`), the position of the person in the video frame at the time they were detected (`BoundingBox`), and how confident Amazon Rekognition Video is that the person has been correctly detected (`Confidence`).

  Facial features are not returned at every timestamp for which the person's path is tracked. Furthermore, in some circumstances a tracked person's body might not be visible, in which case only their face location is returned.
- **Paging information** – The example shows one page of person detection information. You can specify how many person elements to return in the `MaxResults` input parameter for `GetPersonTracking`. If more results than `MaxResults` exist, `GetPersonTracking` returns a token (`NextToken`) used to get the next page of results. For more information, see Getting Amazon Rekognition Video Analysis Results (p. 63).
- **Index** – A unique identifier for identifying the person throughout the video.
- **Video information** – The response includes information about the video format (`VideoMetadata`) in each page of information returned by `GetPersonDetection`.


```
{
    "JobStatus": "SUCCEEDED",
    "NextToken": "AcDymG0fSSoaI6+BBYpka5wVlqttysSPP8VvWcujMDluj1QpFo/vf
+mrMoqBGk8eUEiFlllR6g==",
    "Persons": [
        {
            "Person": {
                "BoundingBox": {
                    "Height": 0.8787037134170532,
                    "Left": 0.00572916679084301,
                    "Top": 0.12129629403352737,
                    "Width": 0.21666666865348816
                },
                "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
                }
            },
            "Index": 0
        },
        "Timestamp": 0
    },
    {
        "Person": {
            "BoundingBox": {
                "Height": 0.9074074029922485,
                "Left": 0.24791666865348816,
                "Top": 0.09259258955717087,
                "Width": 0.375
            },
            "Face": {
                "BoundingBox": {
                    "Height": 0.23000000417232513,
                    "Left": 0.42500001192092896,
                    "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": 0.49765899777412415,
                        "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
                    }
                },
                "Index": 1
            },
            "Timestamp": 0
        }.....

    ],
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 67301,
        "FileExtension": "mp4",
        "Format": "QuickTime / MOV",
        "FrameHeight": 1080,
        "FrameRate": 29.970029830932617,
        "FrameWidth": 1920
    }
}
```

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.

The Amazon Rekognition celebrity recognition API is tuned to detect as many celebrities as possible in different settings, cosmetic makeup, and other conditions. Social, media, and entertainment customers can build apps that use celebrity recognition. For example, an entertainment app that identifies celebrity lookalikes or an app that identifies celebrities as part of automated footage tagging. Amazon Rekognition celebrity recognition is designed to be exclusively used in cases where you expect there may be a known celebrity in an image or a video. For information about recognizing faces that are not celebrities, see Searching Faces in a Collection (p. 137).

Topics

- Recognizing Celebrities in an Image (p. 190)
- Recognizing Celebrities in a Stored Video (p. 196)
- Getting Information About a Celebrity (p. 200)

Recognizing Celebrities in an Image

To recognize celebrities within images and get additional information about recognized celebrities, use the RecognizeCelebrities (p. 322) 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 webpages, if they're available. Amazon Rekognition doesn't remember which image it detected a celebrity in. Your application must store this information.

If you haven't stored the additional information for a celebrity that's returned by `RecognizeCelebrities` and you want to avoid reanalyzing an image to get it, use GetCelebrityInfo (p. 275). To call `GetCelebrityInfo`, you need the unique identifier that Amazon Rekognition assigns to each celebrity. The identifier is returned as part of the `RecognizeCelebrities` response for each celebrity recognized in an image.

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

Calling RecognizeCelebrities

You can provide the input image as an image byte array (base64-encoded image bytes) or as an Amazon S3 object, by using either the AWS Command Line Interface (AWS CLI) or the AWS SDK. 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 that's loaded from your local file system. For information about input image recommendations, see Working with Images (p. 36).

To run this procedure, you need an image file that contains one or more celebrity faces.

To recognize celebrities in an image

1. If you haven't already:
   a. Create or update an IAM user with `AmazonRekognitionFullAccess` and `AmazonS3ReadOnlyAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).
2. Use the following examples to call the `RecognizeCelebrities` operation.

   Java

   This example displays information about the celebrities that are detected in an image.

   Change the value of `photo` to the path and file name of an image file that contains one or more celebrity faces.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.model.Celebrity;
import com.amazonaws.services.rekognition.model.RecognizeCelebritiesRequest;
import com.amazonaws.services.rekognition.model.RecognizeCelebritiesResult;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.ByteBuffer;
import com.amazonaws.util.IOUtils;
import java.util.List;


public class RecognizeCelebrities {

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

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

      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);
      }


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

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

      RecognizeCelebritiesResult
 result=rekognitionClient.recognizeCelebrities(request);

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

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

AWS CLI

This AWS CLI command displays the JSON output for the `recognize-celebrities` CLI operation.

Change `bucketname` to the name of an Amazon S3 bucket that contains an image. Change `input.jpg` to the file name of an image that contains one or more celebrity faces.

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

Python

This example displays information about the celebrities that are detected in an image.

Change the value of `photo` to the path and file name of an image file that contains one or more celebrity faces.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
import json

if __name__ == "__main__":
    photo='moviestars.jpg'

    client=boto3.client('rekognition')

    with open(photo, 'rb') as image:
        response = client.recognize_celebrities(Image={'Bytes': image.read()})

    print('Detected faces for ' + photo)
    for celebrity in response['CelebrityFaces']:
```

```
        print 'Name: ' + celebrity['Name']
        print 'Id: ' + celebrity['Id']
        print 'Position:'
        print '   Left: ' + '{:.2f}'.format(celebrity['Face']['BoundingBox']
['Height'])
        print '   Top: ' + '{:.2f}'.format(celebrity['Face']['BoundingBox']
['Top'])
        print 'Info'
        for url in celebrity['Urls']:
            print '   ' + url
        print
```

.NET

This example displays information about the celebrities that are detected in an image.

Change the value of `photo` to the path and file name of an image file that contains one or more celebrity faces (.jpg or .png format).

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using System.IO;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class CelebritiesInImage
{
    public static void Example()
    {
        String photo = "moviestars.jpg";

        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        RecognizeCelebritiesRequest recognizeCelebritiesRequest = new
 RecognizeCelebritiesRequest();

        Amazon.Rekognition.Model.Image img = new Amazon.Rekognition.Model.Image();
        byte[] data = null;
        try
        {
            using (FileStream fs = new FileStream(photo, FileMode.Open,
 FileAccess.Read))
            {
                data = new byte[fs.Length];
                fs.Read(data, 0, (int)fs.Length);
            }
        }
        catch(Exception)
        {
            Console.WriteLine("Failed to load file " + photo);
            return;
        }

        img.Bytes = new MemoryStream(data);
        recognizeCelebritiesRequest.Image = img;

        Console.WriteLine("Looking for celebrities in image " + photo + "\n");
```

```
        RecognizeCelebritiesResponse recognizeCelebritiesResponse =
rekognitionClient.RecognizeCelebrities(recognizeCelebritiesRequest);

        Console.WriteLine(recognizeCelebritiesResponse.CelebrityFaces.Count + "
celebrity(s) were recognized.\n");
        foreach (Celebrity celebrity in
recognizeCelebritiesResponse.CelebrityFaces)
        {
            Console.WriteLine("Celebrity recognized: " + celebrity.Name);
            Console.WriteLine("Celebrity ID: " + celebrity.Id);
            BoundingBox boundingBox = celebrity.Face.BoundingBox;
            Console.WriteLine("position: " +
               boundingBox.Left + " " + boundingBox.Top);
            Console.WriteLine("Further information (if available):");
            foreach (String url in celebrity.Urls)
                Console.WriteLine(url);
        }
        Console.WriteLine(recognizeCelebritiesResponse.UnrecognizedFaces.Count + "
face(s) were unrecognized.");
    }
}
```

3. Record the value of one of the celebrity IDs that are displayed. You'll need it in Getting Information About a Celebrity (p. 200).

RecognizeCelebrities Operation Request

The input to `RecognizeCelebrities` is an image. In this example, the image is passed as image bytes. For more information, see Working with Images (p. 36).

```
{
    "Image": {
        "Bytes": "/AoSiyvFpm....."
    }
}
```

RecognizeCelebrities Operation Response

The following is example JSON input and output for `RecognizeCelebrities`.

`RecognizeCelebrities` returns an array of recognized celebrities and an array of unrecognized faces. In the example, note the following:

- **Recognized celebrities** – `Celebrities` is an array of recognized celebrities. Each Celebrity (p. 365) object in the array contains the celebrity name and a list of URLs pointing to related content—for example, the celebrity's IMDB link. Amazon Rekognition returns an ComparedFace (p. 370) object that your application can use to determine where the celebrity's face is on the image and a unique identifier for the celebrity. Use the unique identifier to retrieve celebrity information later with the GetCelebrityInfo (p. 275) API operation.
- **Unrecognized faces** – `UnrecognizedFaces` is an array of faces that didn't match any known celebrities. Each ComparedFace (p. 370) object in the array contains a bounding box (as well as other information) that you can use to locate the face in the image.
- **Image orientation** – `OrientationCorrection` is image orientation information that you can use to correctly display the image. For more information, see Getting Image Orientation and Bounding Box Coordinates (p. 49).

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

```
            "Y": 0.4804527163505554
        }, {
            "Type": "mouthLeft",
            "X": 0.5872702598571777,
            "Y": 0.5535582304000854
        }, {
            "Type": "mouthRight",
            "X": 0.6952020525932312,
            "Y": 0.5600858926773071
        }],
        "Pose": {
            "Pitch": -7.386096477508545,
            "Roll": 2.304218292236328,
            "Yaw": -6.175624370574951
        },
        "Quality": {
            "Brightness": 37.16635513305664,
            "Sharpness": 99.9305191040039
        }
    }]
}
```

Recognizing Celebrities in a Stored Video

Amazon Rekognition Video celebrity recognition in stored videos is an asynchronous operation. To recognize celebrities in a stored video, use StartCelebrityRecognition (p. 333) to start video analysis. Amazon Rekognition Video publishes the completion status of the video analysis to an Amazon Simple Notification Service topic. If the video analysis is succesful, call GetCelebrityRecognition (p. 277). to get the analysis results. For more information about starting video analysis and getting the results, see Calling Amazon Rekognition Video Operations (p. 61).

This procedure expands on the code in Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68), which uses an Amazon SQS queue to get the completion status of a video analysis request. To run this procedure, you need a video file that contains one or more celebrity faces.

To detect celebrities in a video stored in an Amazon S3 bucket (SDK)

1. Perform Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68).
2. Add the following code to the class `VideoDetect` that you created in step 1.

   Java

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

//
 Celebrities=========================================================================
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();

}

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());


      //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);

}
```

2a. In the function `main`, replace the line:

```
StartLabels(bucket,video);
```

with:

```
StartCelebrities(bucket,video);
```

2b. Replace the line:

```
GetResultsLabels();
```

with:

```
GetResultsCelebrities();
```

Python

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    def GetResultsCelebrities(self, jobId):
        maxResults = 10
        paginationToken = ''
        finished = False

        while finished == False:
            response = self.rek.get_celebrity_recognition(JobId=jobId,
                                                          MaxResults=maxResults,
                                                          NextToken=paginationToken)

            print(response['VideoMetadata']['Codec'])
            print(str(response['VideoMetadata']['DurationMillis']))
            print(response['VideoMetadata']['Format'])
            print(response['VideoMetadata']['FrameRate'])

            for celebrityRecognition in response['Celebrities']:
                print('Celebrity: ' +
                    str(celebrityRecognition['Celebrity']['Name']))
                print('Timestamp: ' + str(celebrityRecognition['Timestamp']))
                print()

            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True
```

2a. In the function `main`, replace the line:

```
        response = self.rek.start_label_detection(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
                                          NotificationChannel={'RoleArn':
 self.roleArn, 'SNSTopicArn': self.topicArn})
```

with:

```
        response = self.rek.start_celebrity_recognition(Video={'S3Object':
{'Bucket':self.bucket,'Name':self.video}},
            NotificationChannel={'RoleArn':self.roleArn,
 'SNSTopicArn':self.topicArn})
```

2b. Replace the line:

```
                        self.GetResultsLabels(rekMessage['JobId'])
```

with:

```
                    self.GetResultsCelebrities(rekMessage['JobId'])
```

Note
If you've already run a video example other than Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68), the function name to replace is different.

3. Run the code. Information about the celebrities recognized in the video is shown.

GetCelebrityRecognition Operation Response

The following is an example JSON response. The response includes the following:

- **Recognized celebrities** – `Celebrities` is an array of celebrities and the times that they are recognized in a video. A CelebrityRecognition (p. 369) object exists for each time the celebrity is recognized in the video. Each `CelebrityRecognition` contains information about a recognized celebrity (CelebrityDetail (p. 367)) 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 level that Amazon Rekognition Video has in the accuracy of the recognition, and details about the celebrity's face, FaceDetail (p. 379). If you need to get the related content later, you can use `ID` with GetCelebrityInfo (p. 275).
- **VideoMetadata** – Information about the video that was analyzed.


```
{
    "Celebrities": [
        {
            "Celebrity": {
                "BoundingBox": {
                    "Height": 0.8842592835426331,
                    "Left": 0,
                    "Top": 0.11574073880910873,
                    "Width": 0.24427083134651184
                },
                "Confidence": 0.699999988079071,
                "Face": {
                    "BoundingBox": {
                        "Height": 0.20555555820465088,
                        "Left": 0.029374999925494194,
                        "Top": 0.22333332896232605,
                        "Width": 0.11562500149011612
                    },
                    "Confidence": 99.89837646484375,
                    "Landmarks": [
                        {
                            "Type": "eyeLeft",
                            "X": 0.06857934594154358,
                            "Y": 0.30842265486717224
                        },
                        {
                            "Type": "eyeRight",
                            "X": 0.10396526008844376,
                            "Y": 0.300625205039978
                        },
                        {
                            "Type": "nose",
                            "X": 0.0966852456331253,
```

```
                            "Y": 0.34081998467445374
                        },
                        {
                            "Type": "mouthLeft",
                            "X": 0.075217105448246,
                            "Y": 0.3811396062374115
                        },
                        {
                            "Type": "mouthRight",
                            "X": 0.10744428634643555,
                            "Y": 0.37407416105270386
                        }
                    ],
                    "Pose": {
                        "Pitch": -0.9784082174301147,
                        "Roll": -8.808176040649414,
                        "Yaw": 20.28228759765625
                    },
                    "Quality": {
                        "Brightness": 43.312068939208984,
                        "Sharpness": 99.9305191040039
                    }
                },
                "Id": "XXXXXX",
                "Name": "Celeb A",
                "Urls": []
            },
            "Timestamp": 367
        },.....
    ],
    "JobStatus": "SUCCEEDED",
    "NextToken": "XfXnZKiyMOGDhzBzYUhS5puM+g1IgezqFeYpv/H/+5noP/LmM57FitUAwSQ5D6G4AB/
PNwolrw==",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 67301,
        "FileExtension": "mp4",
        "Format": "QuickTime / MOV",
        "FrameHeight": 1080,
        "FrameRate": 29.970029830932617,
        "FrameWidth": 1920
    }
}
```


Getting Information About a Celebrity

In these procedures, you get celebrity information by using the GetCelebrityInfo (p. 275) API operation. The celebrity is identified by using the celebrity ID that's returned from a previous call to the section called “RecognizeCelebrities” (p. 322).

Calling GetCelebrityInfo

These procedures require the celebrity ID for a celebrity that Amazon Rekognition knows. Use the celebrity ID that you note in Recognizing Celebrities in an Image (p. 190).

To get celebrity information (SDK)

1. If you haven't already:

a. Create or update an IAM user with `AmazonRekognitionFullAccess` and `AmazonS3ReadOnlyAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).

b. Install and configure the AWS CLI and AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).

2. Use the following examples to call the `GetCelebrityInfo` operation.

Java

This example displays the name and information about a celebrity.

Replace `id` with one of the celebrity IDs displayed in Recognizing Celebrities in an Image (p. 190).

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
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 = "nnnnnnnn";

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

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

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

      GetCelebrityInfoResult result=rekognitionClient.getCelebrityInfo(request);

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

AWS CLI

This AWS CLI command displays the JSON output for the `get-celebrity-info` CLI operation. Replace `ID` with one of the celebrity IDs displayed in Recognizing Celebrities in an Image (p. 190).

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

Python

This example displays the name and information about a celebrity.

Replace `id` with one of the celebrity IDs displayed in Recognizing Celebrities in an Image (p. 190).

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

if __name__ == "__main__":

    id="nnnnnnnn"

    client=boto3.client('rekognition')

    #Display celebrity info
    print('Getting celebrity info for celebrity: ' + id)
    response=client.get_celebrity_info(Id=id)

    print (response['Name'])
    print ('Further information (if available):')
    for url in response['Urls']:
        print (url)
```

.NET

This example displays the name and information about a celebrity.

Replace `id` with one of the celebrity IDs displayed in Recognizing Celebrities in an Image (p. 190).

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;


public class CelebrityInfo
{
    public static void Example()
    {
        String id = "nnnnnnnn";

        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        GetCelebrityInfoRequest celebrityInfoRequest = new
 GetCelebrityInfoRequest()
        {
            Id = id
        };

        Console.WriteLine("Getting information for celebrity: " + id);
```

```
        GetCelebrityInfoResponse celebrityInfoResponse =
 rekognitionClient.GetCelebrityInfo(celebrityInfoRequest);

        //Display celebrity information
        Console.WriteLine("celebrity name: " + celebrityInfoResponse.Name);
        Console.WriteLine("Further information (if available):");
        foreach (String url in celebrityInfoResponse.Urls)
            Console.WriteLine(url);
    }
}
```

GetCelebrityInfo Operation Request

The following is example JSON input and output for `GetCelebrityInfo`.

The input to `GetCelebrityInfo` is the ID for the required celebrity.

```
{
    "Id": "nnnnnnn"
}
```

`GetCelebrityInfo` returns an array (`Urls`) of links to information about the requested celebrity.

```
{
    "Name": "Celebrity Name",
    "Urls": [
        "www.imdb.com/name/nmnnnnnnn"
    ]
}
```

Detecting Unsafe Content

Amazon Rekognition can determine if an image or video contains explicit or suggestive adult content. Amazon Rekognition Image provides the DetectModerationLabels (p. 269) operation to detect unsafe content in images. Amazon Rekognition Video can detect unsafe content asynchronously by using StartContentModeration (p. 337) and GetContentModeration (p. 282).

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

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

You determine the suitability of an image for your application. For example, images of a suggestive nature might be acceptable, but images containing nudity might not. To filter images, use the ModerationLabel (p. 397) labels array returned by `DetectModerationLabels` (images) and by `GetContentModeration` (videos).

You can set the confidence threshold used by Amazon Rekognition to detect unsafe content by specifying the `MinConfidence` input parameter. Labels for unsafe content detected with a lower confidence than `MinConfidence` are not returned.

Specifying a value for `MinConfidence` that is less than 50% is likely to return a high number of false-positive results. We recommend you use a value that is less than 50% only when detection with a lower precision is acceptable. If you don't specify a value for `MinConfidence`, Amazon Rekognition returns labels for unsafe content detected with at least 50% confidence.

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. 205).

> **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

- Detecting Unsafe Images (API) (p. 205)
- Detecting Unsafe Stored Videos (p. 209)

Detecting Unsafe Images (API)

You can use the DetectModerationLabels (p. 269) operation to determine if an image contains explicit or suggestive adult content.

Detecting Unsafe Content in an Image (SDK)

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 Amazon S3 object. In these procedures, you upload an image (.jpg or .png) to your S3 bucket.

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. 32).

To detect moderation labels in an image (SDK)

1. If you haven't already:
   a. Create or update an IAM user with `AmazonRekognitionFullAccess` and `AmazonS3ReadOnlyAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).
   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).
2. Upload an image to your S3 bucket.

   For instructions, see Uploading Objects into Amazon S3 in the *Amazon Simple Storage Service Console User Guide*.
3. Use the following examples to call the `DetectModerationLabels` operation.

   Java

   This example outputs detected moderation label names, confidence levels, and the parent label for detected moderation labels.

   Replace the values of `bucket` and `photo` with the S3 bucket name and the image file name that you used in step 2.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
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 DetectModerationLabels
{
   public static void main(String[] args) throws Exception
   {
      String photo = "input.jpg";
      String bucket = "bucket";

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

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

AWS CLI

This AWS CLI command displays the JSON output for the `detect-moderation-labels` CLI operation.

Replace `bucket` and `input.jpg` with the S3 bucket name and the image file name that you used in step 2.

```
aws rekognition detect-moderation-labels \

--image '{"S3Object":{"Bucket":"bucket","Name":"input.jpg"}}'
```

Python

This example outputs detected moderation label names, confidence levels, and the parent label for detected moderation labels.

Replace the values of `bucket` and `photo` with the S3 bucket name and the image file name that you used in step 2.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)
```

```
import boto3

if __name__ == "__main__":
    photo='moderate.png'
    bucket='bucket'

    client=boto3.client('rekognition')

    response = client.detect_moderation_labels(Image={'S3Object':
{'Bucket':bucket,'Name':photo}})

    print('Detected labels for ' + photo)
    for label in response['ModerationLabels']:
        print (label['Name'] + ' : ' + str(label['Confidence']))
        print (label['ParentName'])
```

.NET

This example outputs detected moderation label names, confidence levels, and the parent label for detected moderation labels.

Replace the values of `bucket` and `photo` with the S3 bucket name and the image file name that you used in step 2.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DetectModerationLabels
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "bucket";

        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        DetectModerationLabelsRequest detectModerationLabelsRequest = new
 DetectModerationLabelsRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket
                },
            },
            MinConfidence = 60F
        };

        try
        {
            DetectModerationLabelsResponse detectModerationLabelsResponse =
 rekognitionClient.DetectModerationLabels(detectModerationLabelsRequest);
            Console.WriteLine("Detected labels for " + photo);
            foreach (ModerationLabel label in
 detectModerationLabelsResponse.ModerationLabels)
                Console.WriteLine("Label: {0}\n Confidence: {1}\n Parent: {2}",
```

```
                    label.Name, label.Confidence, label.ParentName);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
}
```

DetectModerationLabels Operation Request

The input to `DetectModerationLabels` is an image. In this example JSON input, the source image is loaded from an Amazon S3 bucket. `MinConfidence` is the minimum confidence that Amazon Rekognition Image must have in the accuracy of the detected label for it to be returned in the response.

```
{
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "MinConfidence": 60
}
```

DetectModerationLabels Operation Response

`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`.

In the following example JSON response, note 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 that are 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.

```
{
"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"
    }
]
}
```

Detecting Unsafe Stored Videos

Amazon Rekognition Video unsafe content detection in stored videos is an asynchronous operation. To started detecting unsafe content, call StartContentModeration (p. 337). Amazon Rekognition Video publishes the completion status of the video analysis to an Amazon Simple Notification Service topic. If the video analysis is succesful, call GetContentModeration (p. 282) to get the analysis results. For more information about starting video analysis and getting the results, see Calling Amazon Rekognition Video Operations (p. 61).

This procedure expands on the code in Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68), which uses an Amazon Simple Queue Service queue to get the completion status of a video analysis request.

To detect unsafe content in a video stored in an Amazon S3 bucket (SDK)

1. Perform Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68).
2. Add the following code to the class `VideoDetect` that you created in step 1.

   Java

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    //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));
                System.out.println(label.getModerationLabel().toString());
                System.out.println();
            }
        } while (moderationLabelDetectionResult !=null &&
 moderationLabelDetectionResult.getNextToken() != null);

    }
```

2a. In the function `main`, replace the line:

```
StartLabels(bucket,video);
```

with:

```
StartModerationLabels(bucket,video);
```

2b. Replace the line:

```
GetResultsLabels();
```

with:

```
GetResultsModerationLabels();
```

Python

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

    def GetResultsModerationLabels(self, jobId):
        maxResults = 10
        paginationToken = ''
        finished = False

        while finished == False:
            response = self.rek.get_content_moderation(JobId=jobId,
                                                       MaxResults=maxResults,
                                                       NextToken=paginationToken)
```

```
            print(response['VideoMetadata']['Codec'])
            print(str(response['VideoMetadata']['DurationMillis']))
            print(response['VideoMetadata']['Format'])
            print(response['VideoMetadata']['FrameRate'])

            for contentModerationDetection in response['ModerationLabels']:
                print('Label: ' +
                    str(contentModerationDetection['ModerationLabel']['Name']))
                print('Confidence: ' +
                    str(contentModerationDetection['ModerationLabel']
['Confidence']))
                print('Parent category: ' +
                    str(contentModerationDetection['ModerationLabel']
['ParentName']))
                print('Timestamp: ' + str(contentModerationDetection['Timestamp']))
                print()

            if 'NextToken' in response:
                paginationToken = response['NextToken']
            else:
                finished = True
```

2a. In the function `main`, replace the line:

```
        response = self.rek.start_label_detection(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
                                                  NotificationChannel={'RoleArn':
 self.roleArn, 'SNSTopicArn': self.topicArn})
```

with:

```
        response = self.rek.start_content_moderation(Video={'S3Object':
{'Bucket':self.bucket,'Name':self.video}},
            NotificationChannel={'RoleArn':self.roleArn,
 'SNSTopicArn':self.topicArn})
```

2b. Replace the line:

```
                        self.GetResultsLabels(rekMessage['JobId'])
```

with:

```
                        self.GetResultsModerationLabels(rekMessage['JobId'])
```

Note
If you've already run a video example other than Analyzing a Video Stored in an Amazon S3 Bucket with Java or Python (SDK) (p. 68), the function name to replace is different.

3. Run the code. A list of unsafe content labels detected in the video is shown.

GetContentModeration Operation Response

The response from `GetContentModeration` is an array, `ModerationLabels`, of ContentModerationDetection (p. 373) objects. The array contains an element for each time a moderation label is detected. Within a `ContentModerationDetectionObject` object,

ModerationLabel (p. 397) 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. 204).

The following is an example response from `GetContentModeration`.

```
{
    "JobStatus": "SUCCEEDED",
    "ModerationLabels": [
        {
            "ModerationLabel": {
                "Confidence": 93.02153015136719,
                "Name": "Male Swimwear Or Underwear",
                "ParentName": "Suggestive"
            },
            "Timestamp": 0
        },
        {
            "ModerationLabel": {
                "Confidence": 93.02153015136719,
                "Name": "Suggestive",
                "ParentName": ""
            },
            "Timestamp": 0
        },
        {
            "ModerationLabel": {
                "Confidence": 98.29075622558594,
                "Name": "Male Swimwear Or Underwear",
                "ParentName": "Suggestive"
            },
            "Timestamp": 1000
        },
        {
            "ModerationLabel": {
                "Confidence": 98.29075622558594,
                "Name": "Suggestive",
                "ParentName": ""
            },
            "Timestamp": 1000
        },
        {
            "ModerationLabel": {
                "Confidence": 97.91191101074219,
                "Name": "Male Swimwear Or Underwear",
                "ParentName": "Suggestive"
            },
            "Timestamp": 1999
        }
    ],
    "NextToken": "w5xfYx64+QvCdGTidVVWtczKHe0JAcUFu2tJ1RgDevHRovJ
+1xej2GUDfTMWrTVn1nwSMHi9",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 3533,
        "Format": "QuickTime / MOV",
        "FrameHeight": 1080,
        "FrameRate": 30,
        "FrameWidth": 1920
    }
}
```

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. An example is retrieving and displaying images that contain the same text.
- Content insights. An example is 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. An example is 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. An example is detecting car license plate numbers from traffic camera images.
- Filtering. An example is filtering out personally identifiable information from images.

DetectText (p. 272) 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 +/- 90 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 aren't 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's a large gap between words, relative to the length of the words. This means, depending on the gap between words, Amazon Rekognition might detect multiple lines in text that are 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 #.

For an example, see Detecting Text in an Image (p. 213).

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.

To detect text in an image (API)

1. If you haven't already:

   a. Create or update an IAM user with `AmazonRekognitionFullAccess` and `AmazonS3ReadOnlyAccess` permissions. For more information, see Step 1: Set Up an AWS Account and Create an IAM User (p. 12).

   b. Install and configure the AWS CLI and the AWS SDKs. For more information, see Step 2: Set Up the AWS CLI and AWS SDKs (p. 12).

2. 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*.

3. Use the following examples to call the `DetectText` operation.

   Java

   The following example code displays detected lines and words that were detected in an image.

   Replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and image that you used in step 2.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package aws.example.rekognition.image;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.rekognition.model.DetectTextRequest;
import com.amazonaws.services.rekognition.model.DetectTextResult;
import com.amazonaws.services.rekognition.model.TextDetection;
import java.util.List;



public class DetectText {

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


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

      AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();



      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();
     }
   }
}
```

AWS CLI

This AWS CLI command displays the JSON output for the `detect-text` CLI operation.

Replace the values of `Bucket` and `Name` with the names of the Amazon S3 bucket and image that you used in step 2.

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

Python

The following example code displays detected lines and words detected in an image.

Replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and image that you used in step 2.

```
#Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3

if __name__ == "__main__":

    bucket='bucket'
    photo='inputtext.jpg'

    client=boto3.client('rekognition')


    response=client.detect_text(Image={'S3Object':{'Bucket':bucket,'Name':photo}})


    textDetections=response['TextDetections']
    print response
    print 'Matching faces'
    for text in textDetections:
            print 'Detected text:' + text['DetectedText']
            print 'Confidence: ' + "{:.2f}".format(text['Confidence']) + "%"
            print 'Id: {}'.format(text['Id'])
            if 'ParentId' in text:
                print 'Parent Id: {}'.format(text['ParentId'])
            print 'Type:' + text['Type']
```

```
            print
```

.NET

The following example code displays detected lines and words detected in an image.

Replace the values of `bucket` and `photo` with the names of the Amazon S3 bucket and image that you used in step 2.

```
//Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-developer-guide/blob/master/LICENSE-SAMPLECODE.)

using System;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

public class DetectText
{
    public static void Example()
    {
        String photo = "input.jpg";
        String bucket = "bucket";

        AmazonRekognitionClient rekognitionClient = new AmazonRekognitionClient();

        DetectTextRequest detectTextRequest = new DetectTextRequest()
        {
            Image = new Image()
            {
                S3Object = new S3Object()
                {
                    Name = photo,
                    Bucket = bucket
                }
            }
        };

        try
        {
            DetectTextResponse detectTextResponse =
 rekognitionClient.DetectText(detectTextRequest);
            Console.WriteLine("Detected lines and words for " + photo);
            foreach (TextDetection text in detectTextResponse.TextDetections)
            {
                Console.WriteLine("Detected: " + text.DetectedText);
                Console.WriteLine("Confidence: " + text.Confidence);
                Console.WriteLine("Id : " + text.Id);
                Console.WriteLine("Parent Id: " + text.ParentId);
                Console.WriteLine("Type: " + text.Type);
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

DetectText Operation Request

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. The following example JSON request shows the image loaded from an Amazon S3 bucket.

```
{
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "inputtext.jpg"
        }
    }
}
```

DetectText Operation Response

The `DetectText` operation analyzes the image and returns an array, TextDetections, where each element (`TextDetection (p. 414)`) 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 make 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 that surrounds 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:

Text	ID	Parent ID
but keep	3	
but	8	3
keep	9	3

Text Location on an Image

To determine where the recognized text is on an image, use the bounding box (Geometry (p. 387)) information that's 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. 363) object
- A finer-grained polygon that's made up of multiple X and Y coordinates in a Point (p. 405) 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. 363).

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

```
{
    "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": 92.76634979248047,
        "DetectedText": "MONDAY",
        "Geometry": {
            "BoundingBox": {
                "Height": 0.11997425556182861,
                "Left": 0.5545867085456848,
                "Top": 0.34920141100883484,
                "Width": 0.39841532707214355
            },
            "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": 2,
        "Type": "LINE"
    },
    {
        "Confidence": 96.7636489868164,
        "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": 0.2814019024372101,
                "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": 0.02624112367630005
```

```
            },
            "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": 0.11196041107177734
```

```
                },
                "Polygon": [
                    {
                        "X": 0.7290706038475037,
                        "Y": 0.5251666903495789
                    },
                    {
                        "X": 0.842876672744751,
                        "Y": 0.5268880724906921
                    },
                    {
                        "X": 0.8423973917961121,
                        "Y": 0.5989891886711121
                    },
                    {
                        "X": 0.7285913228988647,
                        "Y": 0.5972678065299988
                    }
                ]
            },
            "Id": 9,
            "ParentId": 3,
            "Type": "WORD"
        },
        {
            "Confidence": 99.47185516357422,
            "DetectedText": "Smiling",
            "Geometry": {
                "BoundingBox": {
                    "Height": 0.3739858865737915,
                    "Left": 0.48920923471450806,
                    "Top": 0.5900818109512329,
                    "Width": 0.5097314119338989
                },
                "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": 10,
            "ParentId": 4,
            "Type": "WORD"
        }
    ]
}
```

Monitoring

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

Topics

- Monitoring Rekognition (p. 224)
- CloudWatch Metrics for Rekognition (p. 227)
- Logging Amazon Rekognition API Calls with AWS CloudTrail (p. 228)

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. 24).

Using CloudWatch Metrics for Rekognition

To use metrics, you must specify the following information:

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

You can get monitoring data for Rekognition using the AWS Management Console, the AWS CLI, or the CloudWatch API. You can also use the CloudWatch API through one of the Amazon AWS Software Development Kits (SDKs) or the CloudWatch API tools. The console displays a series of graphs based on the raw data from the CloudWatch API. Depending on your needs, you might prefer to use either the graphs displayed in the console or retrieved from the API.

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

How Do I?	Relevant Metrics
How do I track the numbers of faces recognized?	Monitor the `Sum` statistic of the `DetectedFaceCount` metric.

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

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

Access Rekognition Metrics

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

To view metrics (console)

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

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

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

- Open the AWS CLI and enter the following command:

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

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

To access metrics (CloudWatch API)

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

Create an Alarm

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

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

To set an alarm (console)

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

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

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

To set an alarm (AWS CLI)

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

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

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

To set an alarm (CloudWatch API)

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

CloudWatch Metrics for Rekognition

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

You can also see an aggregate view of Rekognition metrics from the Rekognition console. For more information, see Exercise 4: See Aggregated Metrics (Console) (p. 24).

CloudWatch Metrics for Rekognition

The following table summarizes the Rekognition metrics.

Metric	Description
SuccessfulRequestCount	The number of successful requests. The response code range for a successful request is 200 to 299. Unit: Count Valid statistics: `Sum,Average`
ThrottledCount	The number of throttled requests. Rekognition throttles a request when it receives more requests than the limit of transactions per second set for your account. If the limit set for your account is frequently exceeded, you can request a limit increase. To request an increase, see AWS Service Limits. Unit: Count Valid statistics: `Sum,Average`
ResponseTime	The time in milliseconds for Rekognition to compute the response. Units: 1. Count for `Data Samples` statistics 2. Milliseconds for `Average` statistics Valid statistics: `Data Samples,Average` **Note** The `ResponseTime` metric is not included in the Rekognition metric pane.
DetectedFaceCount	The number of faces detected with the `IndexFaces` or `DetectFaces` operation. Unit: Count Valid statistics: `Sum,Average`
DetectedLabelCount	The number of labels detected with the `DetectLabels` operation. Unit: Count Valid statistics: `Sum,Average`
ServerErrorCount	The number of server errors. The response code range for a server error is 500 to 599.

Metric	Description
	Unit: Count Valid statistics: `Sum, Average`
UserErrorCount	The number of user errors (invalid parameters, invalid image, no permission, etc). The response code range for a user error is 400 to 499. Unit: Count Valid statistics: `Sum,Average`

CloudWatch Dimension for Rekognition

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

Logging Amazon Rekognition API Calls with AWS CloudTrail

Amazon Rekognition is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon Rekognition. CloudTrail captures a subset of API calls for Amazon Rekognition as events, including calls from the Amazon Rekognition console and from code calls to the Amazon Rekognition APIs. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon Rekognition. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Rekognition, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, including how to configure and enable it, see the AWS CloudTrail User Guide.

Amazon Rekognition Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in Amazon Rekognition, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon Rekognition, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all regions. The trail logs events from all regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail

- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

Amazon Rekognition supports logging the following actions as events in CloudTrail log files:

- the section called "CreateCollection" (p. 241)
- the section called "DeleteCollection" (p. 247)
- the section called "CreateStreamProcessor" (p. 244)
- the section called "DeleteStreamProcessor" (p. 252)
- the section called "DescribeStreamProcessor" (p. 257)
- the section called "ListStreamProcessors" (p. 319)
- the section called "ListCollections" (p. 313)

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Example: Amazon Rekognition Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following example shows a CloudTrail log entry with actions for the following API: `CreateCollection`, `DeleteCollection`, `CreateStreamProcessor`, `DeleteStreamProcessor`, `DescribeStreamProcessor`, `ListStreamProcessors`, and `ListCollections`.

```
{
    "Records": [
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111122223333:user/Alice",
                "accountId": "111122223333",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2018-03-26T21:46:22Z",
            "eventSource": "rekognition.amazonaws.com",
            "eventName": "CreateCollection",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-internal/3",
            "requestParameters": {
                "collectionId": "8fa6aa65-cab4-4d0a-b976-7fd5df63f38a"
            },
            "responseElements": {
```

```
                "collectionArn": "aws:rekognition:us-
east-1:111122223333:collection/8fa6aa65-cab4-4d0a-b976-7fd5df63f38a",
                "faceModelVersion": "2.0",
                "statusCode": 200
            },
            "requestID": "1e77d2d5-313f-11e8-8c0e-75c0272f31a4",
            "eventID": "c6da4992-a9a1-4962-93b6-7d0483d95c30",
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        },
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::111122223333:user/Alice",
                "accountId": "111122223333",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2018-03-26T21:46:25Z",
            "eventSource": "rekognition.amazonaws.com",
            "eventName": "DeleteCollection",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-internal/3",
            "requestParameters": {
                "collectionId": "8fa6aa65-cab4-4d0a-b976-7fd5df63f38a"
            },
            "responseElements": {
                "statusCode": 200
            },
            "requestID": "213d5b78-313f-11e8-8c0e-75c0272f31a4",
            "eventID": "3ed4f4c9-22f8-4de4-a051-0d9d0c2faec9",
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        },
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Alice",
                "accountId": "111122223333",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2018-03-26T21:48:49Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "EX_PRINCIPAL_ID",
                        "arn": "arn:aws:iam::111122223333:role/Admin",
                        "accountId": "111122223333",
                        "userName": "Admin"
                    }
                }
            },
            "eventTime": "2018-03-26T21:53:09Z",
            "eventSource": "rekognition.amazonaws.com",
            "eventName": "ListCollections",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-cli/1.14.63 Python/3.4.7
 Linux/3.2.45-0.6.wd.971.49.326.metal1.x86_64 botocore/1.9.16",
```

```
            "requestParameters": null,
            "responseElements": null,
            "requestID": "116a57f5-3140-11e8-8c0e-75c0272f31a4",
            "eventID": "94bb5ddd-7836-4fb1-a63e-a782eb009824",
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        },
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Alice",
                "accountId": "111122223333",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2018-03-26T21:48:49Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "EX_PRINCIPAL_ID",
                        "arn": "arn:aws:iam::111122223333:role/Admin",
                        "accountId": "111122223333",
                        "userName": "Admin"
                    }
                }
            },
            "eventTime": "2018-03-26T22:06:19Z",
            "eventSource": "rekognition.amazonaws.com",
            "eventName": "CreateStreamProcessor",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-cli/1.14.63 Python/3.4.7
 Linux/3.2.45-0.6.wd.971.49.326.metal1.x86_64 botocore/1.9.16",
            "requestParameters": {
                "roleArn": "arn:aws:iam::111122223333:role/AmazonRekognition-
StreamProcessorRole",
                "settings": {
                    "faceSearch": {
                        "collectionId": "test"
                    }
                },
                "name": "ProcessorName",
                "input": {
                    "kinesisVideoStream": {
                        "arn": "arn:aws:kinesisvideo:us-east-1:111122223333:stream/
VideoStream"
                    }
                },
                "output": {
                    "kinesisDataStream": {
                        "arn": "arn:aws:kinesis:us-east-1:111122223333:stream/
AmazonRekognition-DataStream"
                    }
                }
            },
            "responseElements": {
                "StreamProcessorArn": "arn:aws:rekognition:us-
east-1:111122223333:streamprocessor/ProcessorName"
            },
            "requestID": "e8fb2b3c-3141-11e8-8c0e-75c0272f31a4",
            "eventID": "44ff8f90-fcc2-4740-9e57-0c47610df8e3",
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
```

```
        },
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Alice",
                "accountId": "111122223333",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2018-03-26T21:48:49Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "EX_PRINCIPAL_ID",
                        "arn": "arn:aws:iam::111122223333:role/Admin",
                        "accountId": "111122223333",
                        "userName": "Admin"
                    }
                }
            },
            "eventTime": "2018-03-26T22:09:42Z",
            "eventSource": "rekognition.amazonaws.com",
            "eventName": "DeleteStreamProcessor",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-cli/1.14.63 Python/3.4.7
Linux/3.2.45-0.6.wd.971.49.326.metal1.x86_64 botocore/1.9.16",
            "requestParameters": {
                "name": "ProcessorName"
            },
            "responseElements": null,
            "requestID": "624c4c3e-3142-11e8-8c0e-75c0272f31a4",
            "eventID": "27fd784e-fbf3-4163-9f0b-0006c6eed39f",
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        },
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Alice",
                "accountId": "111122223333",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2018-03-26T21:48:49Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "EX_PRINCIPAL_ID",
                        "arn": "arn:aws:iam::111122223333:role/Admin",
                        "accountId": "111122223333",
                        "userName": "Admin"
                    }
                }
            },
            "eventTime": "2018-03-26T21:56:14Z",
            "eventSource": "rekognition.amazonaws.com",
            "eventName": "ListStreamProcessors",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
```

```
            "userAgent": "aws-cli/1.14.63 Python/3.4.7
 Linux/3.2.45-0.6.wd.971.49.326.metal1.x86_64 botocore/1.9.16",
            "requestParameters": null,
            "responseElements": null,
            "requestID": "811735f9-3140-11e8-8c0e-75c0272f31a4",
            "eventID": "5cfc86a6-758c-4fb9-af13-557b04805c4e",
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        },
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Alice",
                "accountId": "111122223333",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2018-03-26T22:55:22Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "EX_PRINCIPAL_ID",
                        "arn": "arn:aws:iam::111122223333:role/Admin",
                        "accountId": "111122223333",
                        "userName": "Admin"
                    }
                }
            },
            "eventTime": "2018-03-26T22:57:38Z",
            "eventSource": "rekognition.amazonaws.com",
            "eventName": "DescribeStreamProcessor",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-cli/1.14.63 Python/3.4.7
 Linux/3.2.45-0.6.wd.971.49.326.metal1.x86_64 botocore/1.9.16",
            "requestParameters": {
                "name": "ProcessorName"
            },
            "responseElements": null,
            "requestID": "14b2dd34-3149-11e8-8c0e-75c0272f31a4",
            "eventID": "0db3498c-e084-4b30-b5fb-aa0b71ef9b7b",
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

API Reference

This section provides documentation for the Amazon Rekognition API operations.

HTTP Headers

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

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

Topics

- Actions (p. 234)
- Data Types (p. 359)

Actions

The following actions are supported:

- CompareFaces (p. 236)
- CreateCollection (p. 241)
- CreateStreamProcessor (p. 244)
- DeleteCollection (p. 247)
- DeleteFaces (p. 249)
- DeleteStreamProcessor (p. 252)

- DescribeCollection (p. 254)
- DescribeStreamProcessor (p. 257)
- DetectFaces (p. 261)
- DetectLabels (p. 265)
- DetectModerationLabels (p. 269)
- DetectText (p. 272)
- GetCelebrityInfo (p. 275)
- GetCelebrityRecognition (p. 277)
- GetContentModeration (p. 282)
- GetFaceDetection (p. 286)
- GetFaceSearch (p. 291)
- GetLabelDetection (p. 296)
- GetPersonTracking (p. 300)
- IndexFaces (p. 305)
- ListCollections (p. 313)
- ListFaces (p. 316)
- ListStreamProcessors (p. 319)
- RecognizeCelebrities (p. 322)
- SearchFaces (p. 326)
- SearchFacesByImage (p. 329)
- StartCelebrityRecognition (p. 333)
- StartContentModeration (p. 337)
- StartFaceDetection (p. 341)
- StartFaceSearch (p. 345)
- StartLabelDetection (p. 349)
- StartPersonTracking (p. 353)
- StartStreamProcessor (p. 356)
- StopStreamProcessor (p. 358)

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 references to images in an Amazon S3 bucket. If you use the AWS CLI to call Amazon Rekognition operations, passing image bytes isn't supported. The image must be formatted as a PNG or JPEG 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 (p. 124).

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

Request Syntax

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

```
}
```

Request Parameters

The request accepts the following data in JSON format.

SimilarityThreshold (p. 236)

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. 236)

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. 388) object

Required: Yes

TargetImage (p. 236)

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. 388) object

Required: Yes

Response Syntax

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

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

Response Elements

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

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

FaceMatches (p. 237)

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. 372) objects

SourceImageFace (p. 237)

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

Type: ComparedSourceImageFace (p. 371) object

SourceImageOrientationCorrection (p. 237)

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. 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. 237)

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. 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. 237)

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

Type: Array of ComparedFace (p. 370) 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. 420).

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

CreateCollection

Creates a collection in an AWS Region. You can add faces to the collection using the IndexFaces (p. 305) 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. 241)

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. 241)

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

Type: String

FaceModelVersion (p. 241)

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

Type: String

StatusCode (p. 241)

HTTP status code indicating the result of the operation.

Type: Integer

Valid Range: Minimum value of 0.

Errors

AccessDeniedException

You are not authorized to perform the action.

HTTP Status Code: 400

InternalServerError

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

HTTP Status Code: 500

InvalidParameterException

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

HTTP Status Code: 400

ProvisionedThroughputExceededException

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

HTTP Status Code: 400

ResourceAlreadyExistsException

A collection with the specified ID already exists.

HTTP Status Code: 400

ThrottlingException

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

HTTP Status Code: 500

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.

Amazon Rekognition Video is a consumer of live video from Amazon Kinesis Video Streams. Amazon 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. 356) with the `Name` field.

After you have finished analyzing a streaming video, use StopStreamProcessor (p. 358) to stop processing. You can delete the stream processor by calling DeleteStreamProcessor (p. 252).

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. 244)

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. 410) object

Required: Yes

Name (p. 244)

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. 257). `Name` is idempotent.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

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

Required: Yes

Output (p. 244)

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

Type: StreamProcessorOutput (p. 411) object

Required: Yes

RoleArn (p. 244)

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. 244)

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. 412) object

Required: Yes

Response Syntax

```
{
   "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. 245)

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

An Amazon Rekognition service limit was exceeded. For example, if you start too many Amazon Rekognition Video jobs concurrently, calls to start operations (`StartLabelDetection`, for example) will raise a `LimitExceededException` exception (HTTP status code: 400) until the number of concurrently running jobs is below the Amazon Rekognition service limit.

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. 148).

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

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

CollectionId (p. 247)

ID of the collection to delete.

Type: String

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

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

Required: Yes

Response Syntax

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

Response Elements

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

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

StatusCode (p. 247)

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

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. 249)

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. 249)

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. 249)

An array of strings (face IDs) of the faces that were deleted.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 4096 items.

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

Errors

AccessDeniedException

You are not authorized to perform the action.

HTTP Status Code: 400

InternalServerError

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

HTTP Status Code: 500

InvalidParameterException

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

HTTP Status Code: 400

ProvisionedThroughputExceededException

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

HTTP Status Code: 400

ResourceNotFoundException

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

DeleteStreamProcessor

Deletes the stream processor identified by `Name`. You assign the value for `Name` when you create the stream processor with CreateStreamProcessor (p. 244). You might not be able to use the same name for a stream processor for a few seconds after calling `DeleteStreamProcessor`.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

Name (p. 252)

The name of the stream processor you want to delete.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

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

DescribeCollection

Describes the specified collection. You can use `DescribeCollection` to get information, such as the number of faces indexed into a collection and the version of the model used by the collection for face detection.

For more information, see Describing a Collection (p. 145).

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

CollectionId (p. 254)

The ID of the collection to describe.

Type: String

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

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

Required: Yes

Response Syntax

```
{
   "CollectionARN": "string",
   "CreationTimestamp": number,
   "FaceCount": number,
   "FaceModelVersion": "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.

CollectionARN (p. 254)

The Amazon Resource Name (ARN) of the collection.

Type: String

CreationTimestamp (p. 254)

The number of milliseconds since the Unix epoch time until the creation of the collection. The Unix epoch time is 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970.

Type: Timestamp

FaceCount (p. 254)

The number of faces that are indexed into the collection. To index faces into a collection, use IndexFaces (p. 305).

Type: Long

Valid Range: Minimum value of 0.

FaceModelVersion (p. 254)

The version of the face model that's used by the collection for face detection.

For more information, see Model Versioning (p. 9).

Type: String

Errors

AccessDeniedException

You are not authorized to perform the action.

HTTP Status Code: 400

InternalServerError

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

HTTP Status Code: 500

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

DescribeStreamProcessor

Provides information about a stream processor created by CreateStreamProcessor (p. 244). 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. 257)

Name of the stream processor for which you want information.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

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. 257)

Date and time the stream processor was created

Type: Timestamp

Input (p. 257)

Kinesis video stream that provides the source streaming video.

Type: StreamProcessorInput (p. 410) object

LastUpdateTimestamp (p. 257)

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. 257)

Name of the stream processor.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

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

Output (p. 257)

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

Type: StreamProcessorOutput (p. 411) object

RoleArn (p. 257)

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+=,.@\-_/]+`

Settings (p. 257)

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. 412) object

Status (p. 257)

Current status of the stream processor.

Type: String

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

StatusMessage (p. 257)

Detailed status message about the stream processor.

Type: String

StreamProcessorArn (p. 257)

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. These details include 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, and so on.

The face-detection algorithm is most effective on frontal faces. For non-frontal or obscured faces, the algorithm might 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 AWS 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.

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. 261)

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 takes 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. 261)

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. 388) object

Required: Yes

Response Syntax

```
{
   "FaceDetails": [
      {
         "AgeRange": {
            "High": number,
            "Low": number
         },
         "Beard": {
            "Confidence": number,
            "Value": boolean
         },
         "BoundingBox": {
            "Height": number,
            "Left": number,
            "Top": number,
            "Width": number
         },
         "Confidence": number,
         "Emotions": [
            {
               "Confidence": number,
               "Type": "string"
            }
         ],
         "Eyeglasses": {
            "Confidence": number,
            "Value": boolean
         },
         "EyesOpen": {
            "Confidence": number,
            "Value": boolean
         },
         "Gender": {
            "Confidence": number,
            "Value": "string"
         },
         "Landmarks": [
            {
               "Type": "string",
               "X": number,
               "Y": number
            }
         ],
         "MouthOpen": {
            "Confidence": number,
            "Value": boolean
         },
         "Mustache": {
            "Confidence": number,
            "Value": boolean
         },
         "Pose": {
            "Pitch": number,
            "Roll": number,
            "Yaw": number
         },
         "Quality": {
```

```
            "Brightness": number,
            "Sharpness": number
         },
         "Smile": {
            "Confidence": number,
            "Value": boolean
         },
         "Sunglasses": {
            "Confidence": number,
            "Value": boolean
         }
      }
   ],
   "OrientationCorrection": "string"
}
```

Response Elements

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

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

FaceDetails (p. 262)

Details of each face found in the image.

Type: Array of FaceDetail (p. 379) objects

OrientationCorrection (p. 262)

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. 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. 420).

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

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 Analyzing 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. 349).

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 AWS 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 includes 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. 261) operation provides.

`DetectLabels` returns bounding boxes for instances of common object labels in an array of Instance (p. 390) objects. An `Instance` object contains a BoundingBox (p. 363) object, for the location of the label on the image. It also includes the confidence by which the bounding box was detected.

`DetectLabels` also returns a hierarchical taxonomy of detected labels. For example, a detected car might be assigned the label *car*. The label *car* has two parent labels: *Vehicle* (its parent) and *Transportation* (its grandparent). The response returns the entire list of ancestors for a label. Each ancestor is a unique label in the response. In the previous example, *Car*, *Vehicle*, and *Transportation* are returned as unique labels in the response.

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

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

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

Image (p. 266)

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. 388) object

Required: Yes

MaxLabels (p. 266)

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. 266)

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

```
{
   "LabelModelVersion": "string",
   "Labels": [
      {
         "Confidence": number,
         "Instances": [
```

```
            {
               "BoundingBox": {
                  "Height": number,
                  "Left": number,
                  "Top": number,
                  "Width": number
               },
               "Confidence": number
            }
         ],
         "Name": "string",
         "Parents": [
            {
               "Name": "string"
            }
         ]
      }
   ],
   "OrientationCorrection": "string"
}
```

Response Elements

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

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

LabelModelVersion (p. 266)

Version number of the label detection model that was used to detect labels.

Type: String

Labels (p. 266)

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

Type: Array of Label (p. 393) objects

OrientationCorrection (p. 266)

The value of `OrientationCorrection` is always null.

If the input image is in .jpeg format, it might contain exchangeable image (Exif) metadata that includes the image's orientation. Amazon Rekognition uses this orientation information to perform image correction - the bounding box coordinates are translated to represent object locations after the orientation information in the Exif metadata is used to correct the image orientation. Images in .png format don't contain Exif metadata.

Amazon Rekognition doesn't perform image correction for images in .png format and .jpeg images without orientation information in the image Exif metadata. The bounding box coordinates are not translated and represent the object locations before the image is rotated.

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. 420).

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

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. 204).

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 AWS CLI to call Amazon Rekognition operations, passing image bytes is not supported. The image must be either a PNG or JPEG formatted file.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

Image (p. 269)

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. 388) object

Required: Yes

MinConfidence (p. 269)

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

If you don't specify `MinConfidence`, the operation returns labels with confidence values greater than or equal to 50 percent.

Type: Float

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

Required: No

Response Syntax

```
{
```

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

Response Elements

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

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

ModerationLabels (p. 269)

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

Type: Array of ModerationLabel (p. 397) 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. 420).

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

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. 414) 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 +/- 90 degrees orientation of the horizontal axis.

For more information, see Detecting Text (p. 213).

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. 272)

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. 388) 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. 272)

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

Type: Array of TextDetection (p. 414) 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. 420).

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 Amazon 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. 200).

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

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

Id (p. 275)

The ID for the celebrity. You get the celebrity ID from a call to the RecognizeCelebrities (p. 322) operation, which recognizes celebrities in an image.

Type: String

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

Required: Yes

Response Syntax

```
{
   "Name": "string",
   "Urls": [ "string" ]
}
```

Response Elements

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

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

Name (p. 275)

The name of the celebrity.

Type: String

Urls (p. 275)

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

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 Amazon Rekognition Video analysis started by StartCelebrityRecognition (p. 333).

Celebrity recognition in a video is an asynchronous operation. Analysis is started by a call to StartCelebrityRecognition (p. 333) which returns a job identifier (`JobId`). When the celebrity recognition operation finishes, Amazon 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. 59).

`GetCelebrityRecognition` returns detected celebrities and the time(s) they are detected in an array (`Celebrities`) of CelebrityRecognition (p. 369) objects. Each `CelebrityRecognition` contains information about the celebrity in a CelebrityDetail (p. 367) object and the time, `Timestamp`, the celebrity was detected.

> **Note**
> `GetCelebrityRecognition` only returns the default facial attributes (`BoundingBox`, `Confidence`, `Landmarks`, `Pose`, and `Quality`). The other facial attributes listed in the `Face` object of the following response syntax are not returned. For more information, see FaceDetail (p. 379).

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 identifer and additional information urls. If you don't store the additional information urls, you can get them later by calling GetCelebrityInfo (p. 275) with the celebrity identifer.

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. 277)

Job identifier for the required celebrity recognition analysis. You can get the job identifer 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. 277)

Maximum number of results to return per paginated call. The largest value you can specify is 1000. If you specify a value greater than 1000, a maximum of 1000 results is returned. The default value is 1000.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

NextToken (p. 277)

If the previous response was incomplete (because there is more recognized celebrities to retrieve), Amazon 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. 277)

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,
      "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.

Celebrities (p. 278)

Array of celebrities recognized in the video.

Type: Array of CelebrityRecognition (p. 369) objects

JobStatus (p. 278)

The current status of the celebrity recognition job.

Type: String

Valid Values: `IN_PROGRESS | SUCCEEDED | FAILED`

NextToken (p. 278)

If the response is truncated, Amazon 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. 278)

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

Type: String

VideoMetadata (p. 278)

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

Type: VideoMetadata (p. 418) 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

GetContentModeration

Gets the content moderation analysis results for a Amazon Rekognition Video analysis started by StartContentModeration (p. 337).

Content moderation analysis of a video is an asynchronous operation. You start analysis by calling StartContentModeration (p. 337). which returns a job identifier (`JobId`). When analysis finishes, Amazon 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. 59).

`GetContentModeration` returns detected content moderation labels, and the time they are detected, in an array, `ModerationLabels`, of ContentModerationDetection (p. 373) 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. 204).

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

JobId (p. 282)

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. 282)

Maximum number of results to return per paginated call. The largest value you can specify is 1000. If you specify a value greater than 1000, a maximum of 1000 results is returned. The default value is 1000.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

NextToken (p. 282)

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. 282)

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

```
{
   "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. 283)

The current status of the content moderation job.

Type: String

Valid Values: `IN_PROGRESS | SUCCEEDED | FAILED`

ModerationLabels (p. 283)

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

Type: Array of ContentModerationDetection (p. 373) objects

NextToken (p. 283)

If the response is truncated, Amazon 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. 283)

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

Type: String

VideoMetadata (p. 283)

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

Type: VideoMetadata (p. 418) 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

GetFaceDetection

Gets face detection results for a Amazon Rekognition Video analysis started by StartFaceDetection (p. 341).

Face detection with Amazon Rekognition Video is an asynchronous operation. You start face detection by calling StartFaceDetection (p. 341) which returns a job identifier (`JobId`). When the face detection operation finishes, Amazon 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. 286) 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. 286)

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. 286)

Maximum number of results to return per paginated call. The largest value you can specify is 1000. If you specify a value greater than 1000, a maximum of 1000 results is returned. The default value is 1000.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

NextToken (p. 286)

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

```
{
   "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,
                "Yaw": number
            },
            "Quality": {
                "Brightness": number,
                "Sharpness": number
            },
            "Smile": {
                "Confidence": number,
                "Value": boolean
            },
            "Sunglasses": {
                "Confidence": number,
                "Value": boolean
            }
        },
        "Timestamp": number
    }
  ],
  "JobStatus": "string",
  "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.

Faces (p. 287)

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. 382) objects

JobStatus (p. 287)

The current status of the face detection job.

Type: String

Valid Values: `IN_PROGRESS | SUCCEEDED | FAILED`

NextToken (p. 287)

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. 287)

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

Type: String

VideoMetadata (p. 287)

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

Type: VideoMetadata (p. 418) 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 Amazon Rekognition Video face search started by StartFaceSearch (p. 345). 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. 345) which returns a job identifier (`JobId`). When the search operation finishes, Amazon 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. 137).

The search results are retured in an array, `Persons`, of PersonMatch (p. 404) objects. Each`PersonMatch` element contains details about the matching faces in the input collection, person information (facial attributes, bounding boxes, and person identifer) for the matched person, and the time the person was matched in the video.

> **Note**
> `GetFaceSearch` only returns the default facial attributes (`BoundingBox`, `Confidence`, `Landmarks`, `Pose`, and `Quality`). The other facial attributes listed in the `Face` object of the following response syntax are not returned. For more information, see FaceDetail (p. 379).

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

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

Request Parameters

The request accepts the following data in JSON format.

JobId (p. 291)

The job identifer 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. 291)

Maximum number of results to return per paginated call. The largest value you can specify is 1000. If you specify a value greater than 1000, a maximum of 1000 results is returned. The default value is 1000.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

NextToken (p. 291)

If the previous response was incomplete (because there is more search results to retrieve), Amazon 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. 291)

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",
      "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. 292)

The current status of the face search job.

Type: String

Valid Values: `IN_PROGRESS | SUCCEEDED | FAILED`

NextToken (p. 292)

If the response is truncated, Amazon 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. 292)

An array of persons, PersonMatch (p. 404), 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. 404) objects

StatusMessage (p. 292)

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

Type: String

VideoMetadata (p. 292)

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

Type: VideoMetadata (p. 418) 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

GetLabelDetection

Gets the label detection results of a Amazon Rekognition Video analysis started by StartLabelDetection (p. 349).

The label detection operation is started by a call to StartLabelDetection (p. 349) 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. 296) 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`.

> **Note**
> `GetLabelDetection` doesn't return a hierarchical taxonomy, or bounding box information, for detected labels. `GetLabelDetection` returns `null` for the `Parents` and `Instances` attributes of the Label (p. 393) object which is returned in the `Labels` array.

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

JobId (p. 296)

Job identifier for the label detection operation for which you want results returned. You get the job identifer 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. 296)

Maximum number of results to return per paginated call. The largest value you can specify is 1000. If you specify a value greater than 1000, a maximum of 1000 results is returned. The default value is 1000.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

NextToken (p. 296)

If the previous response was incomplete (because there are more labels to retrieve), Amazon 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. 296)

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

```
{
   "JobStatus": "string",
   "Labels": [
      {
         "Label": {
            "Confidence": number,
            "Instances": [
               {
                  "BoundingBox": {
                     "Height": number,
                     "Left": number,
                     "Top": number,
                     "Width": number
                  },
                  "Confidence": number
               }
            ],
            "Name": "string",
            "Parents": [
               {
                  "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. 297)

The current status of the label detection job.

Type: String

Valid Values: `IN_PROGRESS | SUCCEEDED | FAILED`

Labels (p. 297)

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. 395) objects

NextToken (p. 297)

If the response is truncated, Amazon 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. 297)

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

Type: String

VideoMetadata (p. 297)

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

Type: VideoMetadata (p. 418) 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

GetPersonTracking

Gets the path tracking results of a Amazon Rekognition Video analysis started by StartPersonTracking (p. 353).

The person path tracking operation is started by a call to `StartPersonTracking` which returns a job identifier (`JobId`). When the operation finishes, Amazon 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 path tracking operation, first check that the status value published to the Amazon SNS topic is `SUCCEEDED`. If so, call GetPersonTracking (p. 300) and pass the job identifier (`JobId`) from the initial call to `StartPersonTracking`.

`GetPersonTracking` returns an array, `Persons`, of tracked persons and the time(s) their paths were tracked in the video.

> **Note**
> `GetPersonTracking` only returns the default facial attributes (`BoundingBox`, `Confidence`, `Landmarks`, `Pose`, and `Quality`). The other facial attributes listed in the `Face` object of the following response syntax are not returned.
> For more information, see FaceDetail (p. 379).

By default, the array is sorted by the time(s) a person's path 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

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

Request Parameters

The request accepts the following data in JSON format.

JobId (p. 300)

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. 300)

Maximum number of results to return per paginated call. The largest value you can specify is 1000. If you specify a value greater than 1000, a maximum of 1000 results is returned. The default value is 1000.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

NextToken (p. 300)

If the previous response was incomplete (because there are more persons to retrieve), Amazon 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.

Required: No

SortBy (p. 300)

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

```
{
   "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
             },
             "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",
    "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. 301)

The current status of the person tracking job.

Type: String

Valid Values: `IN_PROGRESS | SUCCEEDED | FAILED`

NextToken (p. 301)

If the response is truncated, Amazon 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. 301)

An array of the persons detected in the video and the time(s) their path was tracked throughout the video. An array element will exist for each time a person's path is tracked.

Type: Array of PersonDetection (p. 403) objects

StatusMessage (p. 301)

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

Type: String

VideoMetadata (p. 301)

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

Type: VideoMetadata (p. 418) 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

IndexFaces

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

Amazon Rekognition doesn't save the actual faces that are detected. Instead, the underlying detection algorithm first detects the faces in the input image. For each face, the algorithm extracts facial features into a feature vector, and stores it in the backend database. Amazon Rekognition uses feature vectors when it performs face match and search operations using the SearchFaces (p. 326) and SearchFacesByImage (p. 329) operations.

For more information, see Adding Faces to a Collection (p. 151).

To get the number of faces in a collection, call DescribeCollection (p. 254).

If you're 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're using, call DescribeCollection (p. 254) and supply the collection ID. You can also get the model version from 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. 316) 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.

You can specify the maximum number of faces to index with the `MaxFaces` input parameter. This is useful when you want to index the largest faces in an image and don't want to index smaller faces, such as those belonging to people standing in the background.

The `QualityFilter` input parameter allows you to filter out detected faces that don’t meet the required quality bar chosen by Amazon Rekognition. The quality bar is based on a variety of common use cases. By default, `IndexFaces` filters detected faces. You can also explicitly filter detected faces by specifying `AUTO` for the value of `QualityFilter`. If you do not want to filter detected faces, specify `NONE`.

> **Note**
> To use quality filtering, you need a collection associated with version 3 of the face model. To get the version of the face model associated with a collection, call DescribeCollection (p. 254).

Information about faces detected in an image, but not indexed, is returned in an array of UnindexedFace (p. 416) objects, `UnindexedFaces`. Faces aren't indexed for reasons such as:

- The number of faces detected exceeds the value of the `MaxFaces` request parameter.
- The face is too small compared to the image dimensions.
- The face is too blurry.
- The image is too dark.
- The face has an extreme pose.

In response, the `IndexFaces` operation returns an array of metadata for all detected faces, `FaceRecords`. This includes:

- The bounding box, `BoundingBox`, of the detected face.
- A confidence value, `Confidence`, which indicates the confidence that the bounding box contains a face.

- A face ID, `faceId`, assigned by the service for each face that's detected and stored.
- An image ID, `ImageId`, assigned by the service for the input image.

If you request all facial attributes (by using the `detectionAttributes` parameter), Amazon Rekognition returns detailed facial attributes, such as facial landmarks (for example, location of eye and mouth) and other facial attributes like 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 AWS CLI to call Amazon Rekognition operations, passing image bytes isn't supported. The image must be formatted as a PNG or JPEG file.

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

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

CollectionId (p. 306)

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. 306)

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 takes 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. 306)

The 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. 306)

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 isn't supported.

Type: Image (p. 388) object

Required: Yes

MaxFaces (p. 306)

The maximum number of faces to index. The value of `MaxFaces` must be greater than or equal to 1. `IndexFaces` returns no more than 100 detected faces in an image, even if you specify a larger value for `MaxFaces`.

If `IndexFaces` detects more faces than the value of `MaxFaces`, the faces with the lowest quality are filtered out first. If there are still more faces than the value of `MaxFaces`, the faces with the smallest bounding boxes are filtered out (up to the number that's needed to satisfy the value of `MaxFaces`). Information about the unindexed faces is available in the `UnindexedFaces` array.

The faces that are returned by `IndexFaces` are sorted by the largest face bounding box size to the smallest size, in descending order.

`MaxFaces` can be used with a collection associated with any version of the face model.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

QualityFilter (p. 306)

A filter that specifies how much filtering is done to identify faces that are detected with low quality. Filtered faces aren't indexed. If you specify `AUTO`, filtering prioritizes the identification of faces that don’t meet the required quality bar chosen by Amazon Rekognition. The quality bar is based on a variety of common use cases. Low-quality detections can occur for a number of reasons. Some examples are an object that's misidentified as a face, a face that's too blurry, or a face with a pose that's too extreme to use. If you specify `NONE`, no filtering is performed. The default value is AUTO.

To use quality filtering, the collection you are using must be associated with version 3 of the face model.

Type: String

Valid Values: `NONE | AUTO`

Required: No

Response Syntax

```
{
   "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
         }
      }
   }
],
"OrientationCorrection": "string",
"UnindexedFaces": [
   {
      "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
            }
         },
         "Reasons": [ "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. 308)

The version number of the face detection model that's associated with the input collection (`CollectionId`).

Type: String

FaceRecords (p. 308)

An array of faces detected and added to the collection. For more information, see Managing Faces in a Collection (p. 137).

Type: Array of FaceRecord (p. 384) objects

OrientationCorrection (p. 308)

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. 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`

UnindexedFaces (p. 308)

An array of faces that were detected in the image but weren't indexed. They weren't indexed because the quality filter identified them as low quality, or the `MaxFaces` request parameter filtered them out. To use the quality filter, you specify the `QualityFilter` request parameter.

Type: Array of UnindexedFace (p. 416) 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. 420).

HTTP Status Code: 400

InternalServerError

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

HTTP Status Code: 500

InvalidImageFormatException

The provided image format is not supported.

HTTP Status Code: 400

InvalidParameterException

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

HTTP Status Code: 400

InvalidS3ObjectException

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

HTTP Status Code: 400

ProvisionedThroughputExceededException

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

HTTP Status Code: 400

ResourceNotFoundException

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

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. 142).

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. 313)

Maximum number of collection IDs to return.

Type: Integer

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

Required: No

NextToken (p. 313)

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. 313)

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. 313)

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. 313)

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

Type: String

Length Constraints: Maximum length of 255.

Errors

AccessDeniedException

You are not authorized to perform the action.

HTTP Status Code: 400

InternalServerError

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

HTTP Status Code: 500

InvalidPaginationTokenException

Pagination token in the request is not valid.

HTTP Status Code: 400

InvalidParameterException

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

HTTP Status Code: 400

ProvisionedThroughputExceededException

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

HTTP Status Code: 400

ResourceNotFoundException

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

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. 162).

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. 316)

ID of the collection from which to list the faces.

Type: String

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

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

Required: Yes

MaxResults (p. 316)

Maximum number of faces to return.

Type: Integer

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

Required: No

NextToken (p. 316)

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. 316)

Version number of the face detection model associated with the input collection (`CollectionId`).

Type: String

Faces (p. 316)

An array of `Face` objects.

Type: Array of Face (p. 377) objects

NextToken (p. 316)

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

Type: String

Errors

AccessDeniedException

You are not authorized to perform the action.

HTTP Status Code: 400

InternalServerError

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

HTTP Status Code: 500

InvalidPaginationTokenException

Pagination token in the request is not valid.

HTTP Status Code: 400

InvalidParameterException

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

HTTP Status Code: 400

ProvisionedThroughputExceededException

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

HTTP Status Code: 400

ResourceNotFoundException

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

ListStreamProcessors

Gets a list of stream processors that you have created with CreateStreamProcessor (p. 244).

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

MaxResults (p. 319)

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

Type: Integer

Valid Range: Minimum value of 1.

Required: No

NextToken (p. 319)

If the previous response was incomplete (because there are more stream processors to retrieve), Amazon 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

```
{
   "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. 319)

If the response is truncated, Amazon 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. 319)

List of stream processors that you have created.

Type: Array of StreamProcessor (p. 409) 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. 190).

`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 aren't among the largest 100 faces in the image.

For each celebrity recognized, `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.

Amazon Rekognition doesn't 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. 275) 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 AWS 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. 190).

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

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

Image (p. 322)

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. 388) object

Required: Yes

Response Syntax

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

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

Response Elements

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

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

CelebrityFaces (p. 322)

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. 365) objects

OrientationCorrection (p. 322)

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. 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. 322)

Details about each unrecognized face in the image.

Type: Array of ComparedFace (p. 370) 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. 420).

HTTP Status Code: 400

InternalServerError

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

HTTP Status Code: 500

InvalidImageFormatException

The provided image format is not supported.

HTTP Status Code: 400

InvalidImageFormatException

The provided image format is not supported.

HTTP Status Code: 400

InvalidParameterException

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

HTTP Status Code: 400

InvalidS3ObjectException

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

HTTP Status Code: 400

ProvisionedThroughputExceededException

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

HTTP Status Code: 400

ThrottlingException

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

HTTP Status Code: 500

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. 305) 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. 168).

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

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

CollectionId (p. 326)

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. 326)

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. 326)

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

Type: Float

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

Required: No

MaxFaces (p. 326)

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. 327)

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

Type: Array of FaceMatch (p. 383) objects

FaceModelVersion (p. 327)

Version number of the face detection model associated with the input collection (`CollectionId`).

Type: String

SearchedFaceId (p. 327)

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

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. 305) operation, and then use the face IDs returned in subsequent calls to the SearchFaces (p. 326) 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 AWS 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. 172).

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

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

CollectionId (p. 329)

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. 329)

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

Type: Float

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

Required: No

Image (p. 329)

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. 388) object

Required: Yes

MaxFaces (p. 329)

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",
   "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. 330)

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

Type: Array of FaceMatch (p. 383) objects

FaceModelVersion (p. 330)

Version number of the face detection model associated with the input collection (`CollectionId`).

Type: String

SearchedFaceBoundingBox (p. 330)

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

Type: BoundingBox (p. 363) object

SearchedFaceConfidence (p. 330)

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. 420).

HTTP Status Code: 400

InternalServerError

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

HTTP Status Code: 500

InvalidImageFormatException

The provided image format is not supported.

HTTP Status Code: 400

InvalidParameterException

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

HTTP Status Code: 400

InvalidS3ObjectException

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

HTTP Status Code: 400

ProvisionedThroughputExceededException

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

HTTP Status Code: 400

ResourceNotFoundException

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

StartCelebrityRecognition

Starts asynchronous recognition of celebrities in a stored video.

Amazon Rekognition Video can detect celebrities in a video must be stored in an Amazon S3 bucket. Use Video (p. 417) 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, Amazon 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. 277) and pass the job identifier (`JobId`) from the initial call to `StartCelebrityRecognition`.

For more information, see Recognizing Celebrities (p. 190).

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. 333)

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. 333)

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. 333)

The Amazon SNS topic ARN that you want Amazon Rekognition Video to publish the completion status of the celebrity recognition analysis to.

Type: NotificationChannel (p. 400) object

Required: No

Video (p. 333)

The video in which you want to recognize celebrities. The video must be stored in an Amazon S3 bucket.

Type: Video (p. 417) 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. 334)

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

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

An Amazon Rekognition service limit was exceeded. For example, if you start too many Amazon Rekognition Video jobs concurrently, calls to start operations (`StartLabelDetection`, for example) will raise a `LimitExceededException` exception (HTTP status code: 400) until the number of concurrently running jobs is below the Amazon Rekognition service limit.

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.

Amazon Rekognition Video can moderate content in a video stored in an Amazon S3 bucket. Use Video (p. 417) 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, Amazon 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. 282) and pass the job identifier (`JobId`) from the initial call to `StartContentModeration`.

For more information, see Detecting Unsafe Content (p. 204).

Request Syntax

```
{
   "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. 337)

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. 337)

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. 337)

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. If you don't specify `MinConfidence`, `GetContentModeration` 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. 337)

The Amazon SNS topic ARN that you want Amazon Rekognition Video to publish the completion status of the content moderation analysis to.

Type: NotificationChannel (p. 400) object

Required: No

Video (p. 337)

The video in which you want to moderate content. The video must be stored in an Amazon S3 bucket.

Type: Video (p. 417) 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. 338)

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

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

An Amazon Rekognition service limit was exceeded. For example, if you start too many Amazon Rekognition Video jobs concurrently, calls to start operations (`StartLabelDetection`, for example) will raise a `LimitExceededException` exception (HTTP status code: 400) until the number of concurrently running jobs is below the Amazon Rekognition service limit.

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.

Amazon Rekognition Video can detect faces in a video stored in an Amazon S3 bucket. Use Video (p. 417) 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, Amazon Rekognition Video publishes a completion status to the Amazon Simple Notification Service topic that you specify in `NotificationChannel`. 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. 286) and pass the job identifier (`JobId`) from the initial call to `StartFaceDetection`.

For more information, see Detecting Faces in a Stored Video (p. 131).

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. 341)

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. 341)

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. 341)

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. 341)

The ARN of the Amazon SNS topic to which you want Amazon Rekognition Video to publish the completion status of the face detection operation.

Type: NotificationChannel (p. 400) object

Required: No

Video (p. 341)

The video in which you want to detect faces. The video must be stored in an Amazon S3 bucket.

Type: Video (p. 417) 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. 342)

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

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

An Amazon Rekognition service limit was exceeded. For example, if you start too many Amazon Rekognition Video jobs concurrently, calls to start operations (`StartLabelDetection`, for example) will raise a `LimitExceededException` exception (HTTP status code: 400) until the number of concurrently running jobs is below the Amazon Rekognition service limit.

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. 417) 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, Amazon 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. 291) and pass the job identifier (`JobId`) from the initial call to `StartFaceSearch`. For more information, see Searching Stored Videos for Faces (p. 176).

Request Syntax

```
{
   "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. 345)

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. 345)

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. 345)

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. 345)

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. 345)

The ARN of the Amazon SNS topic to which you want Amazon Rekognition Video to publish the completion status of the search.

Type: NotificationChannel (p. 400) object

Required: No

Video (p. 345)

The video you want to search. The video must be stored in an Amazon S3 bucket.

Type: Video (p. 417) 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. 346)

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

An Amazon Rekognition service limit was exceeded. For example, if you start too many Amazon Rekognition Video jobs concurrently, calls to start operations (`StartLabelDetection`, for example) will raise a `LimitExceededException` exception (HTTP status code: 400) until the number of concurrently running jobs is below the Amazon Rekognition service limit.

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.

Amazon 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. 417) 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, Amazon 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. 296) and pass the job identifier (`JobId`) from the initial call to `StartLabelDetection`.

Request Syntax

```
{
   "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. 349)

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. 349)

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. 349)

Specifies the minimum confidence that Amazon 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. Amazon 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. 349)

The Amazon SNS topic ARN you want Amazon Rekognition Video to publish the completion status of the label detection operation to.

Type: NotificationChannel (p. 400) object

Required: No

Video (p. 349)

The video in which you want to detect labels. The video must be stored in an Amazon S3 bucket.

Type: Video (p. 417) 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. 350)

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

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

An Amazon Rekognition service limit was exceeded. For example, if you start too many Amazon Rekognition Video jobs concurrently, calls to start operations (`StartLabelDetection`, for example) will raise a `LimitExceededException` exception (HTTP status code: 400) until the number of concurrently running jobs is below the Amazon Rekognition service limit.

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 a person's path in a stored video.

Amazon Rekognition Video can track the path of people in a video stored in an Amazon S3 bucket. Use Video (p. 417) 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. 300) and pass the job identifier (`JobId`) from the initial call to `StartPersonTracking`.

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. 353)

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. 353)

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. 353)

The Amazon SNS topic ARN you want Amazon Rekognition Video to publish the completion status of the people detection operation to.

Type: NotificationChannel (p. 400) object

Required: No

Video (p. 353)

The video in which you want to detect people. The video must be stored in an Amazon S3 bucket.

Type: Video (p. 417) 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. 354)

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

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

An Amazon Rekognition service limit was exceeded. For example, if you start too many Amazon Rekognition Video jobs concurrently, calls to start operations (`StartLabelDetection`, for example) will raise a `LimitExceededException` exception (HTTP status code: 400) until the number of concurrently running jobs is below the Amazon Rekognition service limit.

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

StartStreamProcessor

Starts processing a stream processor. You create a stream processor by calling CreateStreamProcessor (p. 244). 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. 356)

The name of the stream processor to start processing.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

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

StopStreamProcessor

Stops a running stream processor that was created by CreateStreamProcessor (p. 244).

Request Syntax

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

Request Parameters

The request accepts the following data in JSON format.

Name (p. 358)

The name of a stream processor created by CreateStreamProcessor (p. 244).

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

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. 361)
- Beard (p. 362)
- BoundingBox (p. 363)
- Celebrity (p. 365)
- CelebrityDetail (p. 367)
- CelebrityRecognition (p. 369)
- ComparedFace (p. 370)
- ComparedSourceImageFace (p. 371)
- CompareFacesMatch (p. 372)
- ContentModerationDetection (p. 373)
- Emotion (p. 374)
- Eyeglasses (p. 375)
- EyeOpen (p. 376)
- Face (p. 377)
- FaceDetail (p. 379)
- FaceDetection (p. 382)
- FaceMatch (p. 383)
- FaceRecord (p. 384)

- FaceSearchSettings (p. 385)
- Gender (p. 386)
- Geometry (p. 387)
- Image (p. 388)
- ImageQuality (p. 389)
- Instance (p. 390)
- KinesisDataStream (p. 391)
- KinesisVideoStream (p. 392)
- Label (p. 393)
- LabelDetection (p. 395)
- Landmark (p. 396)
- ModerationLabel (p. 397)
- MouthOpen (p. 398)
- Mustache (p. 399)
- NotificationChannel (p. 400)
- Parent (p. 401)
- PersonDetail (p. 402)
- PersonDetection (p. 403)
- PersonMatch (p. 404)
- Point (p. 405)
- Pose (p. 406)
- S3Object (p. 407)
- Smile (p. 408)
- StreamProcessor (p. 409)
- StreamProcessorInput (p. 410)
- StreamProcessorOutput (p. 411)
- StreamProcessorSettings (p. 412)
- Sunglasses (p. 413)
- TextDetection (p. 414)
- UnindexedFace (p. 416)
- Video (p. 417)
- VideoMetadata (p. 418)

AgeRange

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

Amazon Rekognition estimates an age range for faces detected in the input image. Estimated age ranges can overlap. A face of a 5-year-old might have an estimated range of 4-6, while the face of a 6-year-old might 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 label, 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. 322) operation.

Contents

Face

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

Type: ComparedFace (p. 370) 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 Amazon 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. 363) 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. 379) 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. 277).

Contents

Celebrity

Information about a recognized celebrity.

Type: CelebrityDetail (p. 367) 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 analyzed by `CompareFaces` and `RecognizeCelebrities`.

Contents

BoundingBox

Bounding box of the face.

Type: BoundingBox (p. 363) 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. 396) objects

Required: No

Pose

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

Type: Pose (p. 406) object

Required: No

Quality

Identifies face image brightness and sharpness.

Type: ImageQuality (p. 389) 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. 363) 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 analyzed 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. 370) 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. 397) 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. 363) 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.

A `FaceDetail` object contains either the default facial attributes or all facial attributes. The default attributes are `BoundingBox`, `Confidence`, `Landmarks`, `Pose`, and `Quality`.

GetFaceDetection (p. 286) is the only Amazon Rekognition Video stored video operation that can return a `FaceDetail` object with all attributes. To specify which attributes to return, use the `FaceAttributes` input parameter for StartFaceDetection (p. 341). The following Amazon Rekognition Video operations return only the default attributes. The corresponding Start operations don't have a `FaceAttributes` input parameter.

- GetCelebrityRecognition
- GetPersonTracking
- GetFaceSearch

The Amazon Rekognition Image DetectFaces (p. 261) and IndexFaces (p. 305) operations can return all facial attributes. To specify which attributes to return, use the `Attributes` input parameter for `DetectFaces`. For `IndexFaces`, use the `DetectAttributes` input parameter.

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. 361) object

Required: No

Beard

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

Type: Beard (p. 362) object

Required: No

BoundingBox

Bounding box of the face. Default attribute.

Type: BoundingBox (p. 363) object

Required: No

Confidence

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

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. 374) objects

Required: No

Eyeglasses

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

Type: Eyeglasses (p. 375) 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. 376) object

Required: No

Gender

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

Type: Gender (p. 386) object

Required: No

Landmarks

Indicates the location of landmarks on the face. Default attribute.

Type: Array of Landmark (p. 396) 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. 398) object

Required: No

Mustache

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

Type: Mustache (p. 399) object

Required: No

Pose

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

Type: Pose (p. 406) object

Required: No

Quality

Identifies image brightness and sharpness. Default attribute.

Type: ImageQuality (p. 389) object

Required: No

Smile

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

Type: Smile (p. 408) object

Required: No

Sunglasses

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

Type: Sunglasses (p. 413) 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. 379) 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. 377) 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 backend 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. 377) object

Required: No

FaceDetail

Structure containing attributes of the face that the algorithm detected.

Type: FaceDetail (p. 379) 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. 244).

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 the text detected by DetectText (p. 272) is located on an image.

Contents

BoundingBox

An axis-aligned coarse representation of the detected text's location on the image.

Type: BoundingBox (p. 363) object

Required: No

Polygon

Within the bounding box, a fine-grained polygon around the detected text.

Type: Array of Point (p. 405) 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 an Amazon 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 Amazon Rekognition API operations.

For more information, see Analyzing an Image Loaded from a Local File System (p. 41).

You pass images stored in an S3 bucket to an Amazon 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 AWS 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. 28).

Contents

Bytes

Blob of image bytes up to 5 MBs.

Type: Base64-encoded binary data object

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

Required: No

S3Object

Identifies an S3 object as the image source.

Type: S3Object (p. 407) 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

Instance

An instance of a label detected by DetectLabels (p. 265).

Contents

BoundingBox

The position of the label instance on the image.

Type: BoundingBox (p. 363) object

Required: No

Confidence

The confidence that Amazon Rekognition Image has in the accuracy of the bounding box.

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

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. 244).

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 Amazon Rekognition Video stream processor. For more information, see CreateStreamProcessor (p. 244).

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 the name, and level of confidence.

The Amazon Rekognition Image operation DetectLabels (p. 265) operation returns a hierarchical taxonomy (`Parents`) for detected labels and also bounding box information (`Instances`) for detected labels. Amazon Rekognition Video doesn't return this information and GetLabelDetection (p. 296) returns `null` for the `Parents` and `Instances` attributes.

Contents

Confidence

Level of confidence.

Type: Float

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

Required: No

Instances

If `Label` represents an object, `Instances` contains the bounding boxes for each instance of the detected object. Bounding boxes are returned for common object labels such as people, cars, furniture, apparel or pets.

Note
Amazon Rekognition Video does not support bounding box information for detected labels. The value of `Instances` is returned as `null` by `GetLabelDetection`.

Type: Array of Instance (p. 390) objects

Required: No

Name

The name (label) of the object or scene.

Type: String

Required: No

Parents

The parent labels for a label. The response includes all ancestor labels.

Note
Amazon Rekognition Video does not support a hierarchical taxonomy of detected labels. The value of `Parents` is returned as `null` by `GetLabelDetection`.

Type: Array of Parent (p. 401) 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

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. 393) 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 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

The x-coordinate from the top left of the landmark expressed as the ratio of the width of the image. For example, if the image is 700 x 200 and the x-coordinate of the landmark is at 350 pixels, this value is 0.5.

Type: Float

Required: No

Y

The y-coordinate from the top left of the landmark expressed as the ratio of the height of the image. For example, if the image is 700 x 200 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. 204).

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 Amazon Rekognition Video Operations (p. 61).

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

Parent

A parent label for a label. A label can have 0, 1, or more parents.

Contents

Name

The name of 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

PersonDetail

Details about a person detected in a video analysis request.

Contents

BoundingBox

Bounding box around the detected person.

Type: BoundingBox (p. 363) object

Required: No

Face

Face details for the detected person.

Type: FaceDetail (p. 379) 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 path tracking information for a single time a person's path is tracked in a video. Amazon Rekognition operations that track people's paths return an array of `PersonDetection` objects with elements for each time a person's path is tracked in a video.

For more information, see GetPersonTracking (p. 300).

Contents

Person

Details about a person whose path was tracked in a video.

Type: PersonDetail (p. 402) object

Required: No

Timestamp

The time, in milliseconds from the start of the video, that the person's path 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 an Amazon Rekognition collection. Includes information about the faces in the Amazon Rekognition collection (FaceMatch (p. 383)), information about the person (PersonDetail (p. 402)), and the time stamp for when the person was detected in a video. An array of `PersonMatch` objects is returned by GetFaceSearch (p. 291).

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. 383) objects

Required: No

Person

Information about the matched person.

Type: PersonDetail (p. 402) 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. 272). `Polygon` represents a fine-grained polygon around detected text. For more information, see Geometry (p. 387).

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

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

Required: No

Roll

Value representing the face rotation on the roll axis.

Type: Float

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

Required: No

Yaw

Value representing the face rotation on the yaw axis.

Type: Float

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

Required: No

See Also

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

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

S3Object

Provides the S3 bucket name and object name.

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

For Amazon Rekognition to process an S3 object, the user must have permission to access the S3 object. For more information, see Resource-Based Policies (p. 28).

Contents

Bucket

Name of the S3 bucket.

Type: String

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

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

Required: No

Name

S3 object key name.

Type: String

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

Required: No

Version

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

Type: String

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

Required: No

See Also

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

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

Smile

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

Contents

Confidence

Level of confidence in the determination.

Type: Float

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

Required: No

Value

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

Type: Boolean

Required: No

See Also

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

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

StreamProcessor

An object that recognizes faces in a streaming video. An Amazon Rekognition stream processor is created by a call to CreateStreamProcessor (p. 244). The request parameters for `CreateStreamProcessor` describe the Kinesis video stream source for the streaming video, face recognition parameters, and where to stream the analysis resullts.

Contents

Name

Name of the Amazon Rekognition stream processor.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

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. 392) 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 Amazon Rekognition Video stream processor streams the results of a video analysis. For more information, see CreateStreamProcessor (p. 244).

Contents

KinesisDataStream

The Amazon Kinesis Data Streams stream to which the Amazon Rekognition stream processor streams the analysis results.

Type: KinesisDataStream (p. 391) 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. 385) 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. 272).

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. 213).

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. 387) 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

UnindexedFace

A face that IndexFaces (p. 305) detected, but didn't index. Use the `Reasons` response attribute to determine why a face wasn't indexed.

Contents

FaceDetail

The structure that contains attributes of a face that `IndexFaces`detected, but didn't index.

Type: FaceDetail (p. 379) object

Required: No

Reasons

An array of reasons that specify why a face wasn't indexed.

- EXTREME_POSE - The face is at a pose that can't be detected. For example, the head is turned too far away from the camera.
- EXCEEDS_MAX_FACES - The number of faces detected is already higher than that specified by the `MaxFaces` input parameter for `IndexFaces`.
- LOW_BRIGHTNESS - The image is too dark.
- LOW_SHARPNESS - The image is too blurry.
- LOW_CONFIDENCE - The face was detected with a low confidence.
- SMALL_BOUNDING_BOX - The bounding box around the face is too small.

Type: Array of strings

Valid Values: `EXCEEDS_MAX_FACES | EXTREME_POSE | LOW_BRIGHTNESS | LOW_SHARPNESS | LOW_CONFIDENCE | SMALL_BOUNDING_BOX`

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. 349) 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. 407) 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 a 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. For information about limits you can change, see AWS Service Limits. To change a limit, see Create Case.

For information about Transactions Per Second (TPS) limits, see AWS Service Limits.

Amazon Rekognition Image

- 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 pixels. Images with dimensions higher than 1920X1080 pixels will need a larger minimum face size proportionally.
- The 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.
- The 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.

For best practice information about images and facial recognition, see Best Practices for Working With Images (p. 37).

Amazon Rekognition Video Stored Video

- Amazon Rekognition Video can analyze stored videos up to 8GB in size.
- Amazon Rekognition Video can analyze stored videos up to 2 hours in length.
- Amazon Rekognition Video supports a maximum of 20 concurrent jobs per account.

Amazon Rekognition Video Streaming Video

- A Kinesis Video input stream can be associated with at most 1 Amazon Rekognition Video stream processor.
- A Kinesis Data output stream can be associated with at most 1 Amazon Rekognition Video stream processor.
- The Kinesis Video input stream and Kinesis Data output stream associated with an Amazon Rekognition Video stream processor cannot be shared by multiple processors.

Document History for Amazon Rekognition

The following table describes important changes in each release of the *Amazon Rekognition Developer Guide*. For notification about updates to this documentation, you can subscribe to an RSS feed.

- **Latest documentation update:** September 18th, 2018

update-history-change	update-history-description	update-history-date
DetectLabels operation updated (p. 421)	Bounding boxes are now returned for certain objects. A hierarchical taxonomy is now available for labels. You can now get the version of the detection model used for detection.	November 1, 2018
IndexFaces operation updated (p. 421)	With IndexFaces you can now use the QualityFilter input parameter to filter out faces detected with low quality. You can also use the MaxFaces input parameter to reduce the number of faces returned based on the quality of the face detection, and the size of the detected face.	September 18, 2018
DescribeCollection operation added (p. 421)	You can now get information about an existing collection by calling the DescribeCollection operation.	August 22, 2018
New Python examples (p. 421)	Python examples have been added to the Amazon Rekognition Video content along with some content reorganization.	June 26, 2018
Updated content layout (p. 421)	The Amazon Rekognition Image content has been reorganized along with new Python and C# examples.	May 29, 2018
Amazon Rekognition supports AWS CloudTrail (p. 421)	Amazon Rekognition is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon Rekognition. For more information, see Logging Amazon Rekognition API Calls with AWS CloudTrail.	April 6, 2018

Analyze stored and streaming videos. New table of contents (p. 421)	For information about analyzing stored videos, see Working with Stored Videos. For information about analyzing streaming videos, see Working with Streaming Videos. The table of contents for the Amazon Rekognition documentation has been rearranged to accomodate image and video operations.	November 29, 2017
Text in Image and Face Detection Models (p. 421)	Amazon Rekognition can now detect text in images. For more information, see Detecting Text. Amazon Rekognition introduces versioning for the face detection deep learning model. For more information, see Model Versioning.	November 21, 2017
Celebrity Recognition (p. 421)	Amazon Rekognition can now analyze images for celebrities. For more information, see Recognizing Celebrities.	June 8, 2017
Image Moderation (p. 421)	Amazon Rekognition can now determine if an image contains explicit or suggestive adult content. For more information, see Detecting Unsafe Content.	April 19, 2017
Age Range for Detected Faces. Aggregated Rekognition Metrics Pane (p. 421)	Amazon Rekognition now returns the estimated age range, in years, for faces detected by the Rekognition API. For more information, see AgeRange. 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).	February 9, 2017
New service and guide (p. 421)	This is the initial release of the image analysis service, Amazon Rekognition, and the *Amazon Rekognition Developer Guide*.	November 30, 2016

AWS Glossary

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