Amazon Rekognition

Developer Guide

Amazon Rekognition: Developer Guide

Table of Contents

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 Amazon 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, face comparison, and face search capabilities. 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. It 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 comparison 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.

- **Detection of Personal Protective Equipment**

  Amazon Rekognition detects Personal Protective Equipment (PPE) such as face covers, head covers, and hand covers on persons in images. You can use PPE detection where safety is the highest priority. For example, industries such as construction, manufacturing, healthcare, food processing, logistics, and retail. With PPE detection, you can automatically detect if a person is wearing a specific type of PPE. You can use the detection results to send a notification or to identify places where safety warnings or training practices can be improved.

- **Sentiment and demographic analysis** – Amazon Rekognition interprets emotional expressions such as happy, sad, or surprise, and demographic information such as gender from facial images. Amazon 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. Note that a prediction of an emotional expression is based on the physical appearance of a person's face only. It is not indicative of a person's internal emotional state, and Rekognition should not be used to make such a determination.

- **Facial Search** – 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. Searching for 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 adult and violent content in images and in stored 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 unsafe content, the API also returns a hierarchical list of labels with confidence scores. These labels indicate specific categories of unsafe content, which enables granular filtering and management of large volumes of user-generated content (UGC). Examples include social and dating sites, photo sharing platforms, blogs and forums, apps for children, ecommerce sites, entertainment, and online advertising services.

- **Celebrity recognition** – Amazon Rekognition can recognize celebrities within supplied images and in videos. Amazon 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 enables 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 catalog 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.

- **Custom labels**– With Amazon Rekognition Custom Labels, you can identify the objects and scenes in images that are specific to your business needs. For example, you can find your logo in social media posts, identify your products on store shelves, classify machine parts in an assembly line, distinguish healthy and infected plants, or detect animated characters in videos. For more information, see What is Amazon Rekognition Custom Labels? in the *Amazon Rekognition Custom Labels Developer Guide*.

Some of the benefits of using Amazon Rekognition include:

- **Integrating powerful image and video analysis into your apps** – You don't need computer vision or deep learning expertise to take advantage of the reliable image and video analysis in Amazon Rekognition. With the 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** – Amazon 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.

- **Integration with other AWS services** – Amazon Rekognition is designed to work seamlessly with other AWS services like Amazon S3 and AWS Lambda. You can call the Amazon Rekognition API directly from Lambda in response to Amazon S3 events. Because 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. This triggers a Lambda function that uses Amazon Rekognition API operations to identify your guest. You can run analysis directly on images that are 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 Amazon 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 that you analyze, and the face metadata that you store. There are no minimum fees or upfront commitments. You can get started for free, and save more as you grow with the Amazon Rekognition 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're a first-time user of Amazon Rekognition, we recommend that you read the following sections in order:

1. **How Amazon Rekognition 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 up your account and test the Amazon Rekognition API.
3. **Working with images (p. 26)** – 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 video analysis (p. 60)** – This section provides information about using Amazon Rekognition with videos stored in an Amazon S3 bucket.
5. **Working with streaming video events (p. 90)** – This section provides information about using Amazon Rekognition with streaming videos.

How Amazon Rekognition works

Amazon Rekognition provides two API sets. You use Amazon Rekognition Image for analyzing images, and Amazon Rekognition Video for analyzing videos.

Both APIs analyze 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 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 analysis

The following are the types of analysis 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 labels (p. 104).

Rekognition can detect a large list of labels in image and stored video. Rekognition can also detect a small number of labels in streaming video.

To detect labels in images, use DetectLabels. To detect labels in stored videos, use StartLabelDetection. To detect labels in streaming video, use CreateStreamProcessor.

Custom labels

Amazon Rekognition Custom Labels can identify the objects and scenes in images that are specific to your business needs by training a machine learning model. For example, you can train a model to detect logos or detect engineering machine parts on an assembly line.

> **Note**
> For information about Amazon Rekognition Custom Labels, see the Amazon Rekognition Custom Labels Developer Guide.

Amazon Rekognition provides a console that you use to create, train, evaluate, and run a machine learning model. For more information, see Getting Started with Amazon Rekognition Custom Labels in the *Amazon Rekognition Custom Labels Develope Guide*. You can also use the Amazon Rekognition Custom Labels API to train and run a model. For more information, see Getting Started with the Amazon Rekognition Custom Labels SDK in the *Amazon Rekognition CustomLabels Developer Guide*.

To analyze images using a trained model, use DetectCustomLabels.

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. 126).

To detect faces in images, use DetectFaces. To detect faces in stored videos, use StartFaceDetection.

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. 157).

To search for known faces in images, use DetectFaces. To search for known faces in stored videos, use StartFaceDetection. To search for known faces in streaming videos, use CreateStreamProcessor.

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. 240).

To detect people in stored videos, use StartPersonTracking.

Personal Protective Equipment

Amazon Rekognition can detect Personal Protective Equipment (PPE) worn by persons detected in an image. Amazon Rekognition detects face covers, hand covers, and head covers. Amazon Rekognition predicts if an item of PPE covers the appropriate body part. You can also get bounding boxes for detected persons and PPE items. For more information, see Detecting personal protective equipment (p. 248).

To detect PPE in images, use DetectProtectiveEquipment.

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, like the emotion expressed, and presentation of gender. For more information, see Recognizing celebrities (p. 270).

To recognize celebrities in images, use RecognizeCelebrities. To recognize celebrities in stored videos, use StartCelebrityRecognition.

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. 308).

To detect text in images, use DetectText.

Inappropriate or offensive content

Amazon Rekognition can analyze images and stored videos for adult and violent content. For more information, see Moderating content (p. 292).

To detect unsafe images, use DetectModerationLabels. To detect unsafe stored videos, use StartContentModeration.

Image and video operations

Amazon Rekognition provides two API sets. You use 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 a variety of entities such as faces or objects. For information about the types of comparison and detection that are supported, see Types of analysis (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 Base64 encoded images bytes directly to an Amazon Rekognition operation. For more information, see Working with images (p. 26).

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. 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. For more information, see Working with stored video analysis (p. 60).

With Amazon Rekognition Video streaming video operations, you can search for faces stored in Amazon Rekognition Video collections or detect labels. Amazon Rekognition Video analyzes a Kinesis video stream. Rekognition sends label detection results from streaming video events as Amazon SNS and Amazon S3 notifications. Rekognition outputs face search results to a Kinesis data stream. You manage streaming video analysis by creating and using an Amazon Rekognition Video stream processor. For example, you create a stream processor by calling CreateStreamProcessor. For more information, see Working with streaming video events (p. 90).

Non-storage and storage-based operations

Amazon Rekognition operations are grouped into the following categories.

- **Non-storage API operations** – In these operations, Amazon Rekognition doesn't 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. 9).

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 use 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 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
- DetectFaces
- CompareFaces
- DetectModerationLabels
- DetectProtectiveEquipment
- RecognizeCelebrities
- DetectText
- GetCelebrityInfo

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

- StartLabelDetection
- StartFaceDetection
- StartPersonTracking
- StartCelebrityRecognition
- StartContentModeration

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 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
- ListFaces
- SearchFacesByImage
- SearchFaces
- DeleteFaces
- DescribeCollection
- DeleteCollection
- ListCollections
- CreateCollection

Amazon Rekognition Video provides the following storage API operations:

- StartFaceSearch
- CreateStreamProcessor

To store facial information, you must first create 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. 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 existing collection by calling StartFaceSearch.

For information about creating and managing collections, see Searching faces in a collection (p. 157).

> **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 can index the 15 largest faces in an image. Later versions of the model enable `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 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 in the response of a collection operation (for example, `CreateCollection`). You can get the model version of an existing collection by calling DescribeCollection. For more information, see Describing a collection (p. 171).

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) 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.

Stateless operations, such as DetectFaces, use the latest version of the model.

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 How Amazon Rekognition works (p. 4).

Before you can use Rekognition, you'll need to create an AWS account and obtain an AWS account ID. You will also want to create an IAM user, which enables the Amazon Rekognition system to determine if you have the permissions needed to access its resources.

After creating your accounts, you'll want to install and configure the AWS CLI and AWS SDKs. The AWS CLI lets you interact with Amazon Rekognition and other services through the command line, while the AWS SDKs let you use programming languages like Java and Python to interact with Amazon Rekognition.

Once you have set up the AWS CLI and AWS SDKs, you can look at some examples of how to use both of them. You can also view some examples of how to interact with Amazon Rekognition using the console.

Topics

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://portal.aws.amazon.com/billing/signup.

2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on 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 root user. 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, see 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 Amazon Rekognition identity-based policies (p. 481). 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. 13)

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, Ruby, Node.js, PHP, .NET, and JavaScript. For information about installing AWS SDKs, see Tools for Amazon Web Services.
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 access 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. 15)

Using Rekognition with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples
AWS SDK for JavaScript	AWS SDK for JavaScript code examples
AWS SDK for Kotlin	AWS SDK for Kotlin code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples
AWS SDK for Rust	AWS SDK for Rust code examples
AWS SDK for Swift	AWS SDK for Swift code examples

For examples specific to Rekognition, see Code examples for Amazon Rekognition using AWS SDKs (p. 380).

Example availability
Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

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. 26)
- Working with stored video analysis (p. 60)
- Working with streaming video events (p. 90)

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

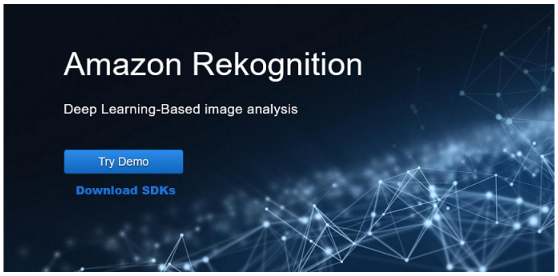
Step 4: Getting started using the Amazon Rekognition console (p. 15)

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 How Amazon Rekognition works (p. 4). You can also use the Amazon Rekognition API or AWS CLI to detect objects and scenes, detect faces, and compare and search faces. For more information, see Step 3: Getting started using the AWS CLI and AWS SDK API (p. 15).

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

Topics

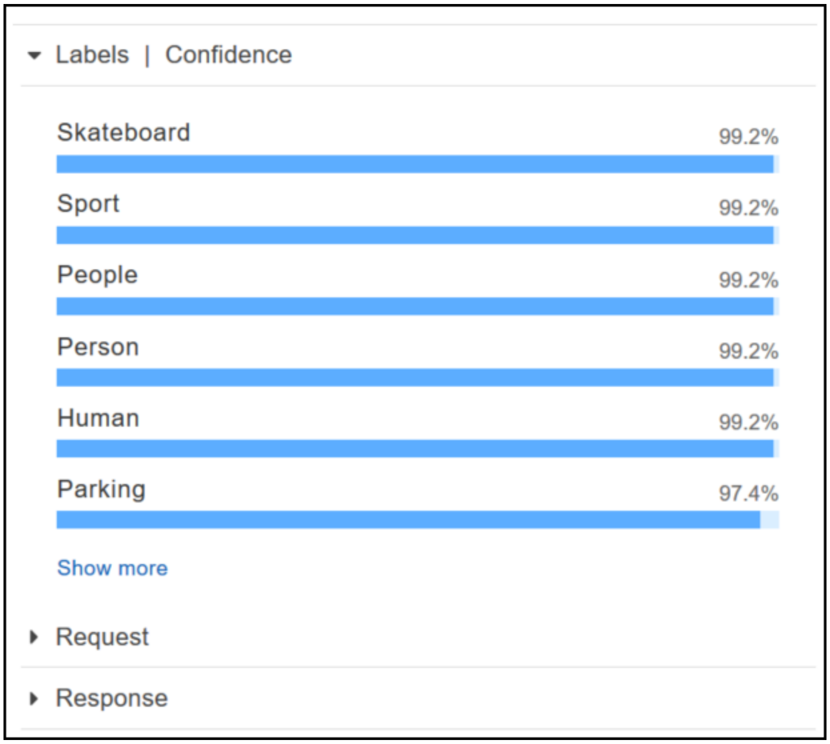


Exercise 1: Detect objects and scenes (Console)

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.

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



To see all the confidence scores shown in the response, choose **Show more** in the **Labels | Confidence** pane.

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 How Amazon Rekognition 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 at https://console.aws.amazon.com/rekognition/.
2. Choose **Label 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.

For more image analysis options, see the section called “Working with images” (p. 26).

Detect objects and people in a video you provide

You can upload a video that you provide as input in the Amazon Rekognition console. Amazon Rekognition returns the people, objects, and labels detected in the video.

> **Note**
> The demo video must not be more than a minute long or larger than 30 MB. It must be in MP4 file format and encoded using the H.264 codec.

To detect objects and people in a video you provide

1. Open the Amazon Rekognition console at https://console.aws.amazon.com/rekognition/.
2. Choose **Video analysis**.
3. Under **Choose a sample or upload your own**, select **Your own video**.
4. Drag and drop your video or select your video from the location where you've stored it.

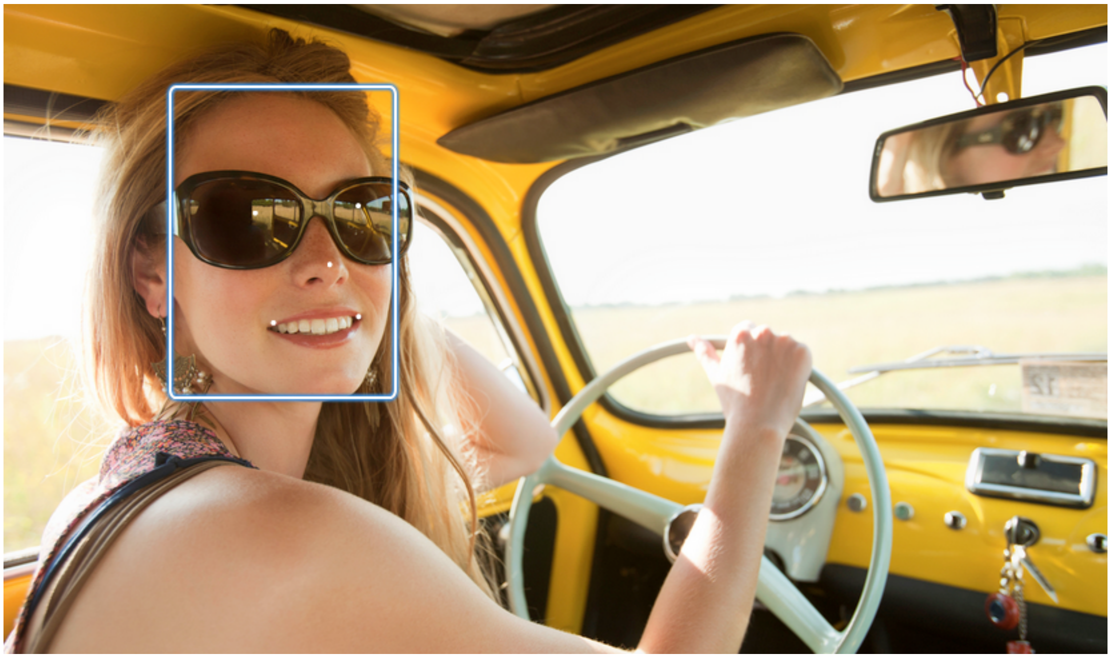
For more video analysis options, see the section called “Working with stored video analysis” (p. 60) or the section called “Working with streaming video events” (p. 90).

Exercise 2: Analyze faces in an image (console)

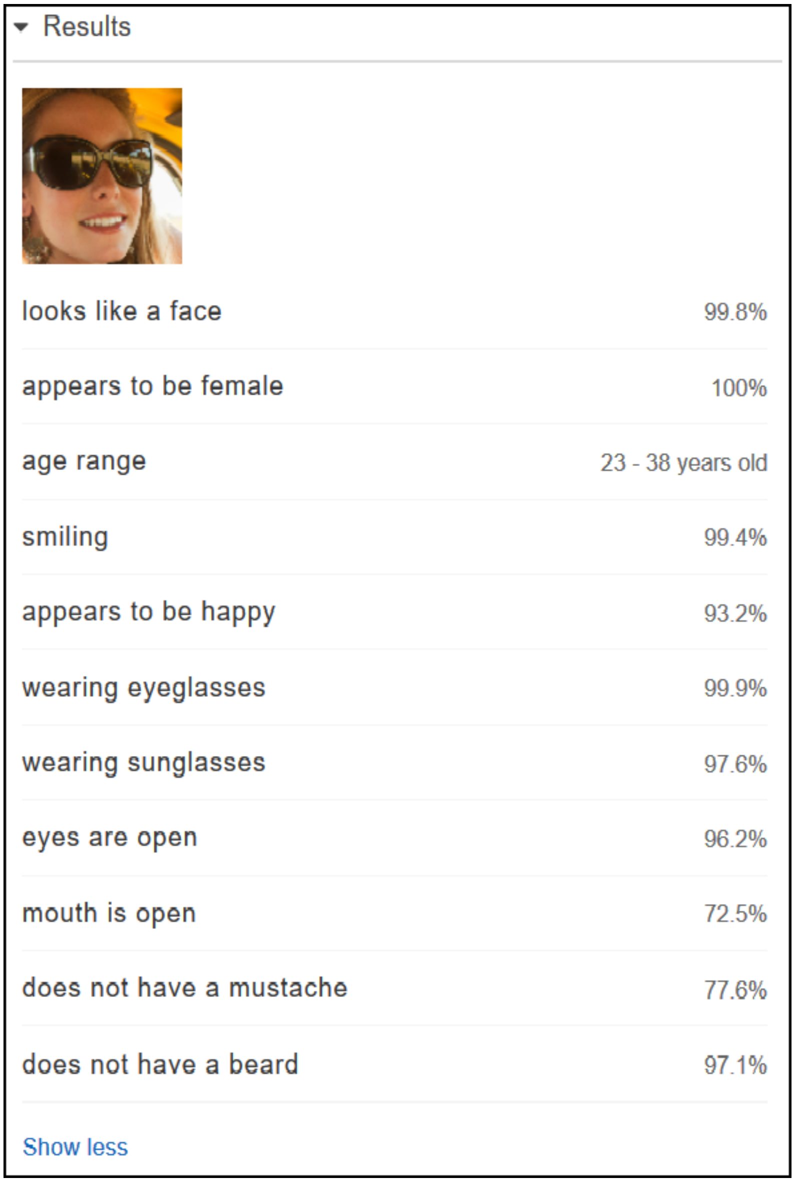
This section shows you how to use the Amazon Rekognition console to detect faces and analyze facial attributes in an image. When you provide an image that contains a face as input, the service detects the

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 How Amazon Rekognition 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 the Amazon Rekognition console at https://console.aws.amazon.com/rekognition/.
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)

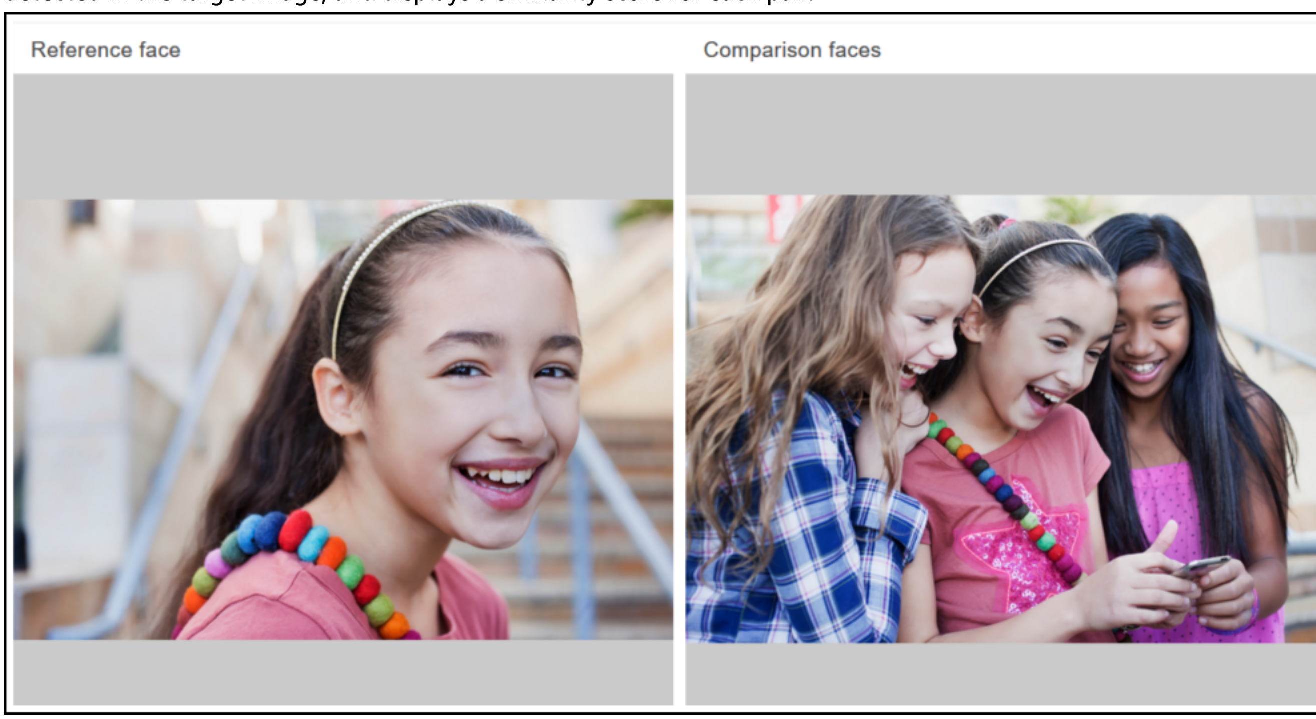
This section shows you how to use the Amazon Rekognition console to compare faces within a set of images with multiple faces in them. When you specify a **Reference face** (source) and a **Comparison faces** (target) image, Rekognition compares the largest face in the source image (that is, the reference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

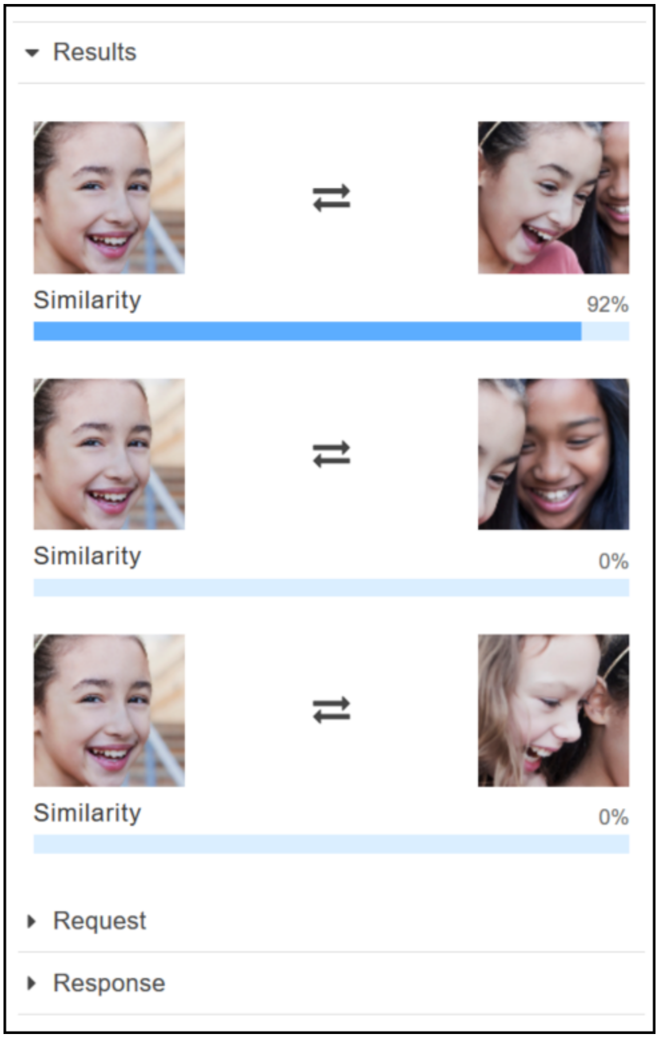
If the source image contains multiple faces, the service detects the largest face in the source image and uses it to compare with each face detected in the target image.

For more information, see Comparing faces in images (p. 139).

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 detects the face in the source image, compares it with each face detected in the target image, and displays a similarity score for each pair.



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 the Amazon Rekognition console at https://console.aws.amazon.com/rekognition/.
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. 498).

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 Identity and access management for Amazon Rekognition (p. 476).

For more information about monitoring Rekognition with CloudWatch see Monitoring Rekognition (p. 495).

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.

Working with images and videos

You can use Amazon Rekognition API operations with images, stored videos, and streaming videos. This section provides general information about writing code that accesses Amazon Rekognition. Other sections in this guide provide information about specific types of image and video analysis, such as face detection.

Topics

Working with images

This section covers the types of analysis that Amazon Rekognition Image can perform on images.

- Object and scene detection (p. 104)
- Face detection and comparison (p. 126)
- Searching faces in a collection (p. 157)
- Celebrity recognition (p. 270)
- Image moderation (p. 292)
- Text in image detection (p. 308)

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. 157).

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

Topics

Image specifications

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. 27). 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. 37).

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. 100).

Correcting image orientation

In several Rekognition API operations, the orientation of an analyzed image is returned. 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. 53).

Image resizing

During analysis, Amazon Rekognition internally resizes images using a set of predefined ranges that best suit a particular model or algorithm. Because of this, Amazon Rekognition might detect a different number of objects, or provide different results, depending on the resolution of the input image. For example, suppose you have two images. The first image has a resolution of 1024x768 pixels. The second image, a resized version of the first image, has a resolution of 640x480 pixels. If you submit the images to DetectLabels, the responses from the two calls to `DetectLabels` might differ slightly.

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 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 Image input parameter. Within `Image`, you specify the S3Object 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 Image specifications (p. 26).

Example request

In this example JSON request for `DetectLabels`, the source image (`input.jpg`) is loaded from an Amazon S3 bucket named `MyBucket`. Note that the region for the S3 bucket containing the S3 object must match the region you use for Amazon Rekognition Image operations.

```
{
    "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. 112).

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. 13).

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 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"}}'
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void getLabelsfromImage(RekognitionClient rekClient, String
 bucket, String image) {

        try {
            S3Object s3Object = S3Object.builder()
                .bucket(bucket)
                .name(image)
                .build() ;

            Image myImage = Image.builder()
                .s3Object(s3Object)
                .build();

            DetectLabelsRequest detectLabelsRequest = DetectLabelsRequest.builder()
                .image(myImage)
```

```
                .maxLabels(10)
                .build();

            DetectLabelsResponse labelsResponse =
 rekClient.detectLabels(detectLabelsRequest);
            List<Label> labels = labelsResponse.labels();
            System.out.println("Detected labels for the given photo");
            for (Label label: labels) {
                System.out.println(label.name() + ": " +
 label.confidence().toString());
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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

def detect_labels(photo, bucket):

    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 ()
    return len(response['Labels'])


def main():
    photo=''
    bucket=''
```

```
    label_count=detect_labels(photo, bucket)
    print("Labels detected: " + str(label_count))


if __name__ == "__main__":
    main()
```

Node.Js

This example displays information about labels 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. Change the value of `bucket` to the name of the S3 bucket containing the provided image file. Change the value of `REGION` to the name of the region associated with your account.

```
// Import required AWS SDK clients and commands for Node.js
import { DetectLabelsCommand } from  "@aws-sdk/client-rekognition";
import  { RekognitionClient } from "@aws-sdk/client-rekognition";

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Create SNS service object.
const rekogClient = new RekognitionClient({ region: REGION });

const bucket = 'bucket-name'
const photo = 'photo-name'

// Set params
const params = {
    Image: {
      S3Object: {
        Bucket: bucket,
        Name: photo
      },
    },
  }

const detect_labels = async () => {
    try {
        const response = await rekogClient.send(new DetectLabelsCommand(params));
        console.log(response.Labels)
        response.Labels.forEach(label =>{
            console.log(`Confidence: ${label.Confidence}`)
            console.log(`Name: ${label.Name}`)
            console.log('Instances:')
            label.Instances.forEach(instance => {
                console.log(instance)
            })
            console.log('Parents:')
            label.Parents.forEach(name => {
                console.log(name)
            })
            console.log("-------")
        })
        return response; // For unit tests.
      } catch (err) {
        console.log("Error", err);
      }
};
```

```
detect_labels();
```

.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);
        }
    }
}
```

Ruby

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 2020 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.)

   # Add to your Gemfile
   # gem 'aws-sdk-rekognition'
   require 'aws-sdk-rekognition'
   credentials = Aws::Credentials.new(
      ENV['AWS_ACCESS_KEY_ID'],
      ENV['AWS_SECRET_ACCESS_KEY']
   )
   bucket = 'bucket' # the bucket name without s3://
   photo  = 'photo' # the name of file
   client   = Aws::Rekognition::Client.new credentials: credentials
   attrs = {
     image: {
       s3_object: {
         bucket: bucket,
         name: photo
       },
     },
     max_labels: 10
   }
  response = client.detect_labels attrs
  puts "Detected labels for: #{photo}"
  response.labels.each do |label|
    puts "Label:      #{label.name}"
    puts "Confidence: #{label.confidence}"
    puts "Instances:"
    label['instances'].each do |instance|
      box = instance['bounding_box']
      puts "  Bounding box:"
      puts "    Top:        #{box.top}"
      puts "    Left:       #{box.left}"
      puts "    Width:      #{box.width}"
      puts "    Height:     #{box.height}"
      puts "  Confidence: #{instance.confidence}"
    end
    puts "Parents:"
    label.parents.each do |parent|
      puts "  #{parent.name}"
    end
    puts "------------"
    puts ""
  end
```

Example response

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

When you perform the `DetectLabels` operation on an image, Amazon Rekognition returns output similar to the following example response.

The response shows that the operation detected multiple labels including Person, Vehicle, and Car. Each label has an associated level of confidence. For example, the detection algorithm is 98.991432% 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 Automobile has two parent labels named Vehicle and Transportation.

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`.

For more information about using the `DetectLabels` operation, see *Detecting labels* (p. 104).


```
{

    {
    "Labels": [
        {
            "Name": "Vehicle",
            "Confidence": 99.15271759033203,
            "Instances": [],
            "Parents": [
                {
                    "Name": "Transportation"
                }
            ]
        },
        {
            "Name": "Transportation",
            "Confidence": 99.15271759033203,
            "Instances": [],
            "Parents": []
        },
        {
            "Name": "Automobile",
            "Confidence": 99.15271759033203,
            "Instances": [],
            "Parents": [
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ]
        },
        {
            "Name": "Car",
            "Confidence": 99.15271759033203,
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.10616336017847061,
                        "Height": 0.18528179824352264,
                        "Left": 0.0037978808395564556,
                        "Top": 0.5039216876029968
                    },
                    "Confidence": 99.15271759033203
                },
                {
                    "BoundingBox": {
                        "Width": 0.2429988533258438,
                        "Height": 0.21577216684818268,
                        "Left": 0.7309805154800415,
                        "Top": 0.5251884460449219
                    },
                    "Confidence": 99.1286392211914
                },
                {
```

```
                "BoundingBox": {
                    "Width": 0.14233611524105072,
                    "Height": 0.15528248250484467,
                    "Left": 0.6494812965393066,
                    "Top": 0.5333095788955688
                },
                "Confidence": 98.48368072509766
            },
            {
                "BoundingBox": {
                    "Width": 0.11086395382881165,
                    "Height": 0.10271988064050674,
                    "Left": 0.10355594009160995,
                    "Top": 0.5354844927787781
                },
                "Confidence": 96.45606231689453
            },
            {
                "BoundingBox": {
                    "Width": 0.06254628300666809,
                    "Height": 0.053911514580249786,
                    "Left": 0.46083059906959534,
                    "Top": 0.5573825240135193
                },
                "Confidence": 93.65448760986328
            },
            {
                "BoundingBox": {
                    "Width": 0.10105438530445099,
                    "Height": 0.12226245552301407,
                    "Left": 0.5743985772132874,
                    "Top": 0.534368634223938
                },
                "Confidence": 93.06217193603516
            },
            {
                "BoundingBox": {
                    "Width": 0.056389667093753815,
                    "Height": 0.17163699865341187,
                    "Left": 0.9427769780158997,
                    "Top": 0.5235804319381714
                },
                "Confidence": 92.6864013671875
            },
            {
                "BoundingBox": {
                    "Width": 0.06003860384225845,
                    "Height": 0.06737709045410156,
                    "Left": 0.22409997880458832,
                    "Top": 0.5441341400146484
                },
                "Confidence": 90.4227066040039
            },
            {
                "BoundingBox": {
                    "Width": 0.02848697081208229,
                    "Height": 0.19150497019290924,
                    "Left": 0.0,
                    "Top": 0.5107086896896362
                },
                "Confidence": 86.65286254882812
            },
            {
                "BoundingBox": {
                    "Width": 0.04067881405353546,
                    "Height": 0.03428703173995018,
```

```
                    "Left": 0.316415935754776,
                    "Top": 0.5566273927688599
                },
                "Confidence": 85.36471557617188
            },
            {
                "BoundingBox": {
                    "Width": 0.043411049991846085,
                    "Height": 0.0893595889210701,
                    "Left": 0.18293385207653046,
                    "Top": 0.5394920110702515
                },
                "Confidence": 82.21705627441406
            },
            {
                "BoundingBox": {
                    "Width": 0.031183116137981415,
                    "Height": 0.03989990055561066,
                    "Left": 0.2853088080883026,
                    "Top": 0.5579366683959961
                },
                "Confidence": 81.0157470703125
            },
            {
                "BoundingBox": {
                    "Width": 0.031113790348172188,
                    "Height": 0.056484755128622055,
                    "Left": 0.2580395042896271,
                    "Top": 0.5504819750785828
                },
                "Confidence": 56.13441467285156
            },
            {
                "BoundingBox": {
                    "Width": 0.08586374670267105,
                    "Height": 0.08550430089235306,
                    "Left": 0.5128012895584106,
                    "Top": 0.5438792705535889
                },
                "Confidence": 52.37760925292969
            }
        ],
        "Parents": [
            {
                "Name": "Vehicle"
            },
            {
                "Name": "Transportation"
            }
        ]
    },
    {
        "Name": "Human",
        "Confidence": 98.9914321899414,
        "Instances": [],
        "Parents": []
    },
    {
        "Name": "Person",
        "Confidence": 98.9914321899414,
        "Instances": [
            {
                "BoundingBox": {
                    "Width": 0.19360728561878204,
                    "Height": 0.2742200493812561,
                    "Left": 0.43734854459762573,
```

```
                        "Top": 0.35072067379951477
                    },
                    "Confidence": 98.9914321899414
                },
                {
                    "BoundingBox": {
                        "Width": 0.03801717236638069,
                        "Height": 0.06597328186035156,
                        "Left": 0.9155802130699158,
                        "Top": 0.5010883808135986
                    },
                    "Confidence": 85.02790832519531
                }
            ],
            "Parents": []
        }
    ],
    "LabelModelVersion": "2.0"
}


}
```


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 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 Image specifications (p. 26).

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. 112).

For a client-side JavaScript example, see Using JavaScript (p. 42).

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. 13).

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 `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.)

import boto3

def detect_labels_local_file(photo):


    client=boto3.client('rekognition')

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

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

    return len(response['Labels'])

def main():
    photo='photo'

    label_count=detect_labels_local_file(photo)
    print("Labels detected: " + str(label_count))


if __name__ == "__main__":
    main()
```

.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;
    }
?>
```

Ruby

This example displays a list of labels that were detected in the input image. 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.)

    # gem 'aws-sdk-rekognition'
    require 'aws-sdk-rekognition'
    credentials = Aws::Credentials.new(
       ENV['AWS_ACCESS_KEY_ID'],
       ENV['AWS_SECRET_ACCESS_KEY']
    )
    client   = Aws::Rekognition::Client.new credentials: credentials
    photo = 'photo.jpg'
    path = File.expand_path(photo) # expand path relative to the current directory
    file = File.read(path)
    attrs = {
      image: {
        bytes: file
      },
      max_labels: 10
    }
    response = client.detect_labels attrs
    puts "Detected labels for: #{photo}"
    response.labels.each do |label|
```

```
      puts "Label:      #{label.name}"
      puts "Confidence: #{label.confidence}"
      puts "Instances:"
      label['instances'].each do |instance|
        box = instance['bounding_box']
        puts "  Bounding box:"
        puts "    Top:        #{box.top}"
        puts "    Left:       #{box.left}"
        puts "    Width:      #{box.width}"
        puts "    Height:     #{box.height}"
        puts "  Confidence: #{instance.confidence}"
      end
      puts "Parents:"
      label.parents.each do |parent|
        puts "  #{parent.name}"
      end
      puts "------------"
      puts ""
    end
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void detectImageLabels(RekognitionClient rekClient, String
 sourceImage) {

        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);

            // Create an Image object for the source image.
            Image souImage = Image.builder()
                .bytes(sourceBytes)
                .build();

            DetectLabelsRequest detectLabelsRequest = DetectLabelsRequest.builder()
                .image(souImage)
                .maxLabels(10)
                .build();

            DetectLabelsResponse labelsResponse =
 rekClient.detectLabels(detectLabelsRequest);
            List<Label> labels = labelsResponse.labels();
            System.out.println("Detected labels for the given photo");
            for (Label label: labels) {
                System.out.println(label.name() + ": " +
 label.confidence().toString());
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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 DetectFaces.

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. 45).

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. 46).
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. 46).
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.

Note
The following code example uses two scripts that are no longer part of Amazon Cognito. To get these files, follow the links for aws-cognito-sdk.min.js and amazon-cognito-identity.min.js, then save the text from each as seperate `.js` files.

JavaScript example code

The following code example uses JavaScript V2. For an example in JavaScript V3, see the example in the AWS Documentation SDK examples GitHub repository.

```
<!--
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>
  <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("data:image/jpeg;base64,")[1]);

      } catch (e) {
        jpg = false;
      }
      if (jpg == false) {
        try {
          image = atob(e.target.result.split("data:image/png;base64,")[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(ua);
    };
  })(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, using JavaScript V2. 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 identities**, 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 `AWS.config.region` and `IdentityPoolId` 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 the **Permissions** tab, choose **Attach Policies**.
14. Choose **AmazonRekognitionReadOnlyAccess**.
15. Choose **Attach Policy**.

Displaying bounding boxes

Amazon Rekognition Image operations can return bounding boxes coordinates for items that are detected in images. For example, the DetectFaces operation returns a bounding box (BoundingBox) for each face detected in an image. You can use the bounding box coordinates to display a box around detected items. For example, the following image shows a bounding box surrounding a face.

A `BoundingBox` has the following properties:

- Height – The height of the bounding box as a ratio of the overall image height.
- Left – The left coordinate of the bounding box as a ratio of overall image width.
- Top – The top coordinate of the bounding box as a ratio of overall image height.
- Width – The width of the bounding box as a ratio of the overall image width.

Each BoundingBox property has a value between 0 and 1. Each property value is a ratio of the overall image width (`Left` and `Width`) or height (`Height` and `Top`). For example, if the input image is 700 x 200 pixels, and the top-left coordinate of the bounding box is 350 x 50 pixels, the API returns a `Left` value of 0.5 (350/700) and a `Top` value of 0.25 (50/200).

The following diagram shows the range of an image that each bounding box property covers.

To display the bounding box with the correct location and size, you have to multiply the BoundingBox values by the image width or height (depending on the value you want) to get the pixel values. You use the pixel values to display the bounding box. For example, the pixel dimensions of the previous image are 608 width x 588 height. The bounding box values for the face are:

```
BoundingBox.Left: 0.3922065
BoundingBox.Top: 0.15567766
BoundingBox.Width: 0.284666
BoundingBox.Height: 0.2930403
```

The location of the face bounding box in pixels is calculated as follows:

```
Left coordinate = BoundingBox.Left (0.3922065) * image width (608) = 238

Top coordinate = BoundingBox.Top (0.15567766) * image height (588) = 91

Face width = BoundingBox.Width (0.284666) * image width (608) = 173

Face height = BoundingBox.Height (0.2930403) * image height (588) = 172
```

You use these values to display a bounding box around the face.

> **Note**
> An image can be orientated in various ways. Your application might need to rotate the image to display it with the correction orientation. Bounding box coordinates are affected by the orientation of the image. You might need to translate the coordinates before you can display a bounding box at the right location. For more information, see Getting image orientation and bounding box coordinates (p. 53).

The following examples show how to display a bounding box around faces that are detected by calling DetectFaces. The examples assume that the images are oriented to 0 degrees. The examples also show how to download the image from an Amazon S3 bucket.

To display a bounding box

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. 13).

2. Use the following examples to call the `DetectFaces` operation.

   Java

   Change the value of `bucket` to the Amazon S3 bucket that contains the image file. Change the value of `photo` to the file name of an image file (.jpg or .png format).

```
//Loads images, detects faces and draws bounding boxes.Determines exif orientation,
 if necessary.
package com.amazonaws.samples;

//Import the basic graphics classes.
import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.*;

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

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;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.S3ObjectInputStream;

// Calls DetectFaces and displays a bounding box around each detected image.
```

```
public class DisplayFaces extends JPanel {

    private static final long serialVersionUID = 1L;

    BufferedImage image;
    static int scale;
    DetectFacesResult result;

    public DisplayFaces(DetectFacesResult facesResult, BufferedImage bufImage)
 throws Exception {
        super();
        scale = 1; // increase to shrink image size.

        result = facesResult;
        image = bufImage;


    }
    // Draws the bounding box around the detected faces.
    public void paintComponent(Graphics g) {
        float left = 0;
        float top = 0;
        int height = image.getHeight(this);
        int width = image.getWidth(this);

        Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.

        // Draw the image.
        g2d.drawImage(image, 0, 0, width / scale, height / scale, this);
        g2d.setColor(new Color(0, 212, 0));

        // Iterate through faces and display bounding boxes.
        List<FaceDetail> faceDetails = result.getFaceDetails();
        for (FaceDetail face : faceDetails) {

            BoundingBox box = face.getBoundingBox();
            left = width * box.getLeft();
            top = height * box.getTop();
            g2d.drawRect(Math.round(left / scale), Math.round(top / scale),
                    Math.round((width * box.getWidth()) / scale),
 Math.round((height * box.getHeight())) / scale);

        }
    }


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

        String photo = "photo.png";
        String bucket = "bucket";
        int height = 0;
        int width = 0;

        // Get the image from an S3 Bucket
        AmazonS3 s3client = AmazonS3ClientBuilder.defaultClient();

        com.amazonaws.services.s3.model.S3Object s3object =
 s3client.getObject(bucket, photo);
        S3ObjectInputStream inputStream = s3object.getObjectContent();
        BufferedImage image = ImageIO.read(inputStream);
        DetectFacesRequest request = new DetectFacesRequest()
                .withImage(new Image().withS3Object(new
 S3Object().withName(photo).withBucket(bucket)));

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

```
        // Call DetectFaces
        AmazonRekognition amazonRekognition =
 AmazonRekognitionClientBuilder.defaultClient();
        DetectFacesResult result = amazonRekognition.detectFaces(request);

        //Show the bounding box info for each face.
        List<FaceDetail> faceDetails = result.getFaceDetails();
        for (FaceDetail face : faceDetails) {

            BoundingBox box = face.getBoundingBox();
            float left = width * box.getLeft();
            float top = height * box.getTop();
            System.out.println("Face:");

            System.out.println("Left: " + String.valueOf((int) left));
            System.out.println("Top: " + String.valueOf((int) top));
            System.out.println("Face Width: " + String.valueOf((int) (width *
 box.getWidth())));
            System.out.println("Face Height: " + String.valueOf((int) (height *
 box.getHeight())));
            System.out.println();

        }

        // Create frame and panel.
        JFrame frame = new JFrame("RotateImage");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        DisplayFaces panel = new DisplayFaces(result, image);
        panel.setPreferredSize(new Dimension(image.getWidth() / scale,
 image.getHeight() / scale));
        frame.setContentPane(panel);
        frame.pack();
        frame.setVisible(true);

    }
}
```

Python

Change the value of `bucket` to the Amazon S3 bucket that contains the image file. Change the value of `photo` to the file name of an image file (.jpg or .png format).

```
import boto3
import io
from PIL import Image, ImageDraw, ExifTags, ImageColor

def show_faces(photo,bucket):


    client=boto3.client('rekognition')

    # Load image from S3 bucket
    s3_connection = boto3.resource('s3')
    s3_object = s3_connection.Object(bucket,photo)
    s3_response = s3_object.get()

    stream = io.BytesIO(s3_response['Body'].read())
    image=Image.open(stream)

    #Call DetectFaces
    response = client.detect_faces(Image={'S3Object': {'Bucket': bucket, 'Name':
 photo}},
```

```
        Attributes=['ALL'])

    imgWidth, imgHeight = image.size
    draw = ImageDraw.Draw(image)


    # calculate and display bounding boxes for each detected face
    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')

        box = faceDetail['BoundingBox']
        left = imgWidth * box['Left']
        top = imgHeight * box['Top']
        width = imgWidth * box['Width']
        height = imgHeight * box['Height']


        print('Left: ' + '{0:.0f}'.format(left))
        print('Top: ' + '{0:.0f}'.format(top))
        print('Face Width: ' + "{0:.0f}".format(width))
        print('Face Height: ' + "{0:.0f}".format(height))

        points = (
            (left,top),
            (left + width, top),
            (left + width, top + height),
            (left , top + height),
            (left, top)

        )
        draw.line(points, fill='#00d400', width=2)

        # Alternatively can draw rectangle. However you can't set line width.
        #draw.rectangle([left,top, left + width, top + height], outline='#00d400')

    image.show()

    return len(response['FaceDetails'])

def main():
    bucket="bucket"
    photo="photo"

    faces_count=show_faces(photo,bucket)
    print("faces detected: " + str(faces_count))


if __name__ == "__main__":
    main()
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

Note that `s3` refers to the AWS SDK Amazon Amazon S3 client and `rekClient` refers to the AWS SDK Amazon Rekognition client.

```
    public static void displayAllFaces(S3Client s3,
                                       RekognitionClient rekClient,
                                       String sourceImage,
                                       String bucketName) {
```

```
        int height;
        int width;
        byte[] data = getObjectBytes (s3, bucketName, sourceImage);
        InputStream is = new ByteArrayInputStream(data);

        try {
            SdkBytes sourceBytes = SdkBytes.fromInputStream(is);
            image = ImageIO.read(sourceBytes.asInputStream());
            width = image.getWidth();
            height = image.getHeight();

            // Create an Image object for the source image
            software.amazon.awssdk.services.rekognition.model.Image souImage =
Image.builder()
                .bytes(sourceBytes)
                .build();

            DetectFacesRequest facesRequest = DetectFacesRequest.builder()
                .attributes(Attribute.ALL)
                .image(souImage)
                .build();

            result = rekClient.detectFaces(facesRequest);

            // Show the bounding box info for each face.
            List<FaceDetail> faceDetails = result.faceDetails();
            for (FaceDetail face : faceDetails) {
                BoundingBox box = face.boundingBox();
                float left = width * box.left();
                float top = height * box.top();
                System.out.println("Face:");

                System.out.println("Left: " + (int) left);
                System.out.println("Top: " + (int) top);
                System.out.println("Face Width: " + (int) (width * box.width()));
                System.out.println("Face Height: " + (int) (height *
box.height()));
                System.out.println();
            }

            // Create the frame and panel.
            JFrame frame = new JFrame("RotateImage");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            DisplayFacesFrame panel = new DisplayFacesFrame(image);
            panel.setPreferredSize(new Dimension(image.getWidth() / scale,
image.getHeight() / scale));
            frame.setContentPane(panel);
            frame.pack();
            frame.setVisible(true);

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        } catch (IOException e) {
           e.printStackTrace();
        }
    }

    public static byte[] getObjectBytes (S3Client s3, String bucketName, String
keyName) {

        try {
            GetObjectRequest objectRequest = GetObjectRequest
                .builder()
                .key(keyName)
                .bucket(bucketName)
```

```
                .build();

            ResponseBytes<GetObjectResponse> objectBytes =
s3.getObjectAsBytes(objectRequest);
            return objectBytes.asByteArray();

        } catch (S3Exception e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    public DisplayFacesFrame(BufferedImage bufImage) {
        super();
        scale = 1; // increase to shrink image size.
        image = bufImage;
    }

    // Draws the bounding box around the detected faces.
    public void paintComponent(Graphics g) {
        float left;
        float top;
        int height = image.getHeight(this);
        int width = image.getWidth(this);
        Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.

        // Draw the image
        g2d.drawImage(image, 0, 0, width / scale, height / scale, this);
        g2d.setColor(new Color(0, 212, 0));

        // Iterate through the faces and display bounding boxes.
        List<FaceDetail> faceDetails = result.faceDetails();
        for (FaceDetail face : faceDetails) {
            BoundingBox box = face.boundingBox();
            left = width * box.left();
            top = height * box.top();
            g2d.drawRect(Math.round(left / scale), Math.round(top / scale),
                    Math.round((width * box.width()) / scale), Math.round((height *
box.height())) / scale);
        }
    }
```

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. You might need to correct this orientation. For some .jpg files, the image's orientation is contained in the image's Exchangeable image file format (Exif) metadata.

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, but it doesn't estimate coordinates for .jpg files without Exif metadata.

The following examples show how to get the bounding box coordinates for the faces detected in an image.

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.

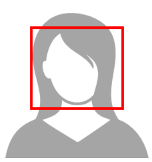
Some images in .jpg format contain orientation information in Exif metadata. If available, the Exif metadata for the image contains the orientation. In the Exif metadata, you can find the image's orientation in the `orientation` field. Although Amazon Rekognition Image identifies the presence of image 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 2.32.

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.

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 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 examples show how to use the AWS SDK to get the Exif image orientation data and the bounding box coordinates for celebrities detected by the `RecognizeCelebrities` operation.

> **Note**
> Support for estimating image orientation using the the `OrientationCorrection` field has ceased as of August 2021. Any returned values for this field included in an API response will always be NULL.

Java

This example loads an image from the local file system, calls the `RecognizeCelebrities` 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.

In the function `main`, replace the value of `photo` with the name and path of an image that is stored locally in either .png or .jpg 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 com.amazonaws.samples;
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.services.rekognition.model.RecognizeCelebritiesRequest;
import com.amazonaws.services.rekognition.model.RecognizeCelebritiesResult;
import com.amazonaws.util.IOUtils;
import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.model.Celebrity;
import com.amazonaws.services.rekognition.model.ComparedFace;

public class RotateImage {

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

  String photo = "photo.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 GetCelebrities

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


      RecognizeCelebritiesResult result =
 amazonRekognition.recognizeCelebrities(request);
      // The returned value of OrientationCorrection will always be null
      System.out.println("Orientation: " + result.getOrientationCorrection() + "\n");
      List <Celebrity> celebs = result.getCelebrityFaces();

      for (Celebrity celebrity: celebs) {
          System.out.println("Celebrity recognized: " + celebrity.getName());
          System.out.println("Celebrity ID: " + celebrity.getId());
          ComparedFace  face = celebrity.getFace()
;             ShowBoundingBoxPositions(height,
                 width,
                 face.getBoundingBox(),
                 result.getOrientationCorrection());

            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.

In the function `main`, replace the value of `photo` with the name and path of an image that is stored locally in either .png or .jpg 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
import io
from PIL import Image


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

```
    top = 0

    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']))


def celebrity_image_information(photo):
    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.recognize_celebrities(Image={'Bytes': image_binary})

    print()
    print('Detected celebrities for ' + photo)

    for celebrity in response['CelebrityFaces']:
        print('Name: ' + celebrity['Name'])
        print('Id: ' + celebrity['Id'])

        # Value of "orientation correction" will always be null
        if 'OrientationCorrection' in response:
            show_bounding_box_positions(height, width, celebrity['Face']
['BoundingBox'])

        print()
    return len(response['CelebrityFaces'])


def main():
    photo = 'photo'

    celebrity_count = celebrity_image_information(photo)
    print("celebrities detected: " + str(celebrity_count))


if __name__ == "__main__":
    main()
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void recognizeAllCelebrities(RekognitionClient rekClient, String
 sourceImage) {
```

```
        try {
            BufferedImage image;
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);

            image = ImageIO.read(sourceBytes.asInputStream());
            int height = image.getHeight();
            int width = image.getWidth();

            Image souImage = Image.builder()
                .bytes(sourceBytes)
                .build();

            RecognizeCelebritiesRequest request = RecognizeCelebritiesRequest.builder()
                .image(souImage)
                .build();

            RecognizeCelebritiesResponse result =
rekClient.recognizeCelebrities(request) ;
            List<Celebrity> celebs=result.celebrityFaces();
            System.out.println(celebs.size() + " celebrity(s) were recognized.\n");
            for (Celebrity celebrity: celebs) {
                System.out.println("Celebrity recognized: " + celebrity.name());
                System.out.println("Celebrity ID: " + celebrity.id());
                ComparedFace  face = celebrity.face();
                ShowBoundingBoxPositions(height,
                        width,
                        face.boundingBox(),
                        result.orientationCorrectionAsString());
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

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

        float left;
        float top;
        if (rotation==null){
            System.out.println("No estimated estimated orientation.");
            return;
        }

        // Calculate face position based on the 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:
                System.out.println("No estimated orientation information. Check Exif
data.");
                return;
        }

        System.out.println("Left: " + (int) left);
        System.out.println("Top: " + (int) top);
        System.out.println("Face Width: " + (int) (imageWidth * box.width()));
        System.out.println("Face Height: " + (int) (imageHeight * box.height()));
    }
```

Working with stored video analysis

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 analysis that Amazon Rekognition Video can perform, an overview of the API, and examples for using Amazon Rekognition Video.

Topics

Types of analysis

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

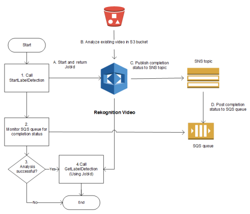
- Video Segments (p. 330)
- Labels (p. 104)
- Suggestive and explicit adult content (p. 292)
- Text (p. 308)
- Celebrities (p. 270)
- Faces (p. 126)
- People (p. 240)

For more information, see How Amazon Rekognition 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. 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, 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 other Amazon Rekognition Video operations. The following table lists the `Start` and `Get` operations for each of the non-storage Amazon Rekognition operations.

Detection	Start Operation	Get Operation
Video Segments	StartSegmentDetection	GetSegmentDetection
Labels	StartLabelDetection	GetLabelDetection
Explicit or suggestive adult content	StartContentModeration	GetContentModeration
Text	StartTextDetection	GetTextDetection
Celebrities	StartCelebrityRecognition	GetCelebrityRecognition
Faces	StartFaceDetection	GetFaceDetection
People	StartPersonTracking	GetPersonTracking

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. 62). 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. 85).

Video formats and storage

Amazon Rekognition operations can analyze videos that are stored in Amazon S3 buckets. For a list of all limits on video analysis operation, see *Guidelines and quotas* (p. 508).

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.

Any Amazon Rekognition Video API that analyzes audio data only supports AAC audio codecs.

The maximum file size for a stored video is 10GB.

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 video for a specific person or for multiple people. You store facial metadata from source images in a collection by using the IndexFaces operation. You can then use StartFaceSearch to start asynchronously searching for faces in the collection. You use GetFaceSearch to get the search results. For more information, see Searching stored videos for faces (p. 208). Searching for people is an example of a storage-based Amazon Rekognition operation. For more information, see Storage-based API operations (p. 9).

You can also search for people in a streaming video. For more information, see Working with streaming video events (p. 90).

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. 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.

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 and StartPersonTracking. 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. 240). For AWS CLI examples, see Analyzing a video with the AWS Command Line Interface (p. 85).

Topics

Starting video analysis

You start an Amazon Rekognition Video label detection request by calling StartLabelDetection. 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 the 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.
- You shoudn't reuse a token with different `Start` operations as you'll get unpredictable results from Amazon Rekognition.

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` (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": "270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1nnnnnnnnnnnn",
    "Status": "SUCCEEDED",
    "API": "StartLabelDetection",
    "JobTag": "DetectingLabels",
    "Timestamp": 1510865364756,
    "Video": {
        "S3ObjectName": "video.mp4",
        "S3Bucket": "bucket"
    }
}
```

For more information, see Reference: Video analysis results notification (p. 87).

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:


```
{
    "Labels": [
        {
            "Timestamp": 0,
            "Label": {
                "Instances": [],
                "Confidence": 60.51791763305664,
                "Parents": [],
                "Name": "Electronics"
            }
        },
        {
            "Timestamp": 0,
            "Label": {
                "Instances": [],
                "Confidence": 99.53411102294922,
                "Parents": [],
                "Name": "Human"
            }
        },
        {
            "Timestamp": 0,
            "Label": {
                "Instances": [
                    {
                        "BoundingBox": {
                            "Width": 0.11109819263219833,
                            "Top": 0.08098889887332916,
                            "Left": 0.8881205320358276,
                            "Height": 0.9073750972747803
                        },
                        "Confidence": 99.5831298828125
                    },
                    {
                        "BoundingBox": {
                            "Width": 0.1268676072359085,
                            "Top": 0.14018426835536957,
                            "Left": 0.0003282368124928324,
                            "Height": 0.7993982434272766
                        },
                        "Confidence": 99.46029663085938
                    }
                ],
                "Confidence": 99.53411102294922,
                "Parents": [],
                "Name": "Person"
            }
        },
        .
        .
        .

        {
            "Timestamp": 166,
            "Label": {
                "Instances": [],
                "Confidence": 73.6471176147461,
                "Parents": [
                    {
```

```
                        "Name": "Clothing"
                    }
                ],
                "Name": "Sleeve"
            }
        }

    ],
    "LabelModelVersion": "2.0",
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
        "Format": "QuickTime / MOV",
        "FrameRate": 23.976024627685547,
        "Codec": "h264",
        "DurationMillis": 5005,
        "FrameHeight": 674,
        "FrameWidth": 1280
    }
}
```

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`.

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. You also have to subscribe an Amazon SQS queue to your Amazon SNS topics.

> **Note**
> If you're using these instructions to set up the Analyzing a video stored in an Amazon S3 bucket with Java or Python (SDK) (p. 68) example, you don't need to do steps 3, 4, 5, and 6. The example includes code to create and configure the Amazon SNS topic and Amazon SQS queue.

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. 67) 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
   - AmazonS3FullAccess
   - AmazonSNSFullAccess
2. Install and configure the required AWS SDK. For more information, see Step 2: Set up the AWS CLI and AWS SDKs (p. 13).
3. 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.

4. Create an Amazon SQS standard queue by using the Amazon SQS console. Note the queue ARN.
5. Subscribe the queue to the topic you created in step 3.
6. Give permission to the Amazon SNS topic to send messages to the Amazon SQS queue.
7. 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).
8. To ensure your account is secure, you will want to limit the scope of Rekognition's access to just the resources you are using. This can be done by attaching a Trust policy to your IAM service role. For information on how to do this, see Cross-service confused deputy prevention (p. 505).
9. 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 7"
        }
    ]
}
```

   Give the inline policy a name of your choosing.
10. 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. 85).

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. 62). 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 example code in this procedure shows you how to do the following:

1. Create the Amazon SNS topic.
2. Create the Amazon SQS queue.
3. Give Amazon Rekognition Video permission to publish the completion status of a video analysis operation to the Amazon SNS topic.
4. Subscribe the Amazon SQS queue to the Amazon SNS topic.
5. Start the video analysis request by calling StartLabelDetection.
6. 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.
7. Get and display the video analysis results by calling GetLabelDetection.

Prerequisites

The example code for this procedure is provided in Java and Python. You need to have the appropriate AWS SDK 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 Actions Defined by Amazon Rekognition.

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). You don't need to do steps 3, 4, 5, and 6 because the example code creates and configures the Amazon SNS topic and Amazon SQS queue.
2. Upload an MOV or MPEG-4 format video file to an Amazon 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 User Guide*.
3. Use the following code examples to detect labels in a video.

   Java

   In the function `main`:

   - Replace `roleArn` with the ARN of the IAM service role that you created in step 7 of To configure Amazon Rekognition Video (p. 66).
   - Replace the values of `bucket` and `video` with the bucket and video file name that you specified 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.auth.policy.Policy;
import com.amazonaws.auth.policy.Condition;
import com.amazonaws.auth.policy.Principal;
import com.amazonaws.auth.policy.Resource;
import com.amazonaws.auth.policy.Statement;
import com.amazonaws.auth.policy.Statement.Effect;
import com.amazonaws.auth.policy.actions.SQSActions;
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.Instance;
import com.amazonaws.services.rekognition.model.Label;
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.Parent;
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.sns.AmazonSNS;
import com.amazonaws.services.sns.AmazonSNSClientBuilder;
import com.amazonaws.services.sns.model.CreateTopicRequest;
import com.amazonaws.services.sns.model.CreateTopicResult;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.Message;
import com.amazonaws.services.sqs.model.QueueAttributeName;
import com.amazonaws.services.sqs.model.SetQueueAttributesRequest;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import java.util.*;

public class VideoDetect {


    private static String sqsQueueName=null;
    private static String snsTopicName=null;
    private static String snsTopicArn = null;
    private static String roleArn= null;
    private static String sqsQueueUrl = null;
    private static String sqsQueueArn = null;
    private static String startJobId = null;
    private static String bucket = null;
    private static String video = null;
    private static AmazonSQS sqs=null;
    private static AmazonSNS sns=null;
    private static AmazonRekognition rek = null;

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


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

```
        video = "";
        bucket = "";
        roleArn= "";

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

        CreateTopicandQueue();

        //=================================================

        StartLabelDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetLabelDetectionResults();

       //=================================================


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

    }


    static boolean GetSQSMessageSuccess() throws Exception
    {
     boolean success=false;


        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(sqsQueueUrl).getMessages();
            if (dotLine++<40){
                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")){
                        success=true;
                       }
                       else{
                           System.out.println("Video analysis failed");
                       }

                       sqs.deleteMessage(sqsQueueUrl,message.getReceiptHandle());
                   }

                   else{
                       System.out.println("Job received was not job " +
startJobId);
                       //Delete unknown message. Consider moving message to dead
letter queue
                       sqs.deleteMessage(sqsQueueUrl,message.getReceiptHandle());
                   }
               }
           }
           else {
               Thread.sleep(5000);
           }
       } while (!jobFound);

       System.out.println("Finished processing video");
       return success;
   }


   private static void StartLabelDetection(String bucket, String video) throws
Exception{

       NotificationChannel channel= new NotificationChannel()
               .withSNSTopicArn(snsTopicArn)
               .withRoleArn(roleArn);


       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 GetLabelDetectionResults() 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();
                Label label=detectedLabel.getLabel();
                System.out.println("Millisecond: " + Long.toString(seconds) + " ");

                System.out.println("   Label:" + label.getName());
                System.out.println("   Confidence:" +
detectedLabel.getLabel().getConfidence().toString());

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

    }

    // Creates an SNS topic and SQS queue. The queue is subscribed to the topic.
    static void CreateTopicandQueue()
    {
        //create a new SNS topic
        snsTopicName="AmazonRekognitionTopic" +
Long.toString(System.currentTimeMillis());
```

```
        CreateTopicRequest createTopicRequest = new
 CreateTopicRequest(snsTopicName);
        CreateTopicResult createTopicResult = sns.createTopic(createTopicRequest);
        snsTopicArn=createTopicResult.getTopicArn();

        //Create a new SQS Queue
        sqsQueueName="AmazonRekognitionQueue" +
 Long.toString(System.currentTimeMillis());
        final CreateQueueRequest createQueueRequest = new
 CreateQueueRequest(sqsQueueName);
        sqsQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();
        sqsQueueArn = sqs.getQueueAttributes(sqsQueueUrl,
 Arrays.asList("QueueArn")).getAttributes().get("QueueArn");

        //Subscribe SQS queue to SNS topic
        String sqsSubscriptionArn = sns.subscribe(snsTopicArn, "sqs",
 sqsQueueArn).getSubscriptionArn();

        // Authorize queue
          Policy policy = new Policy().withStatements(
                  new Statement(Effect.Allow)
                  .withPrincipals(Principal.AllUsers)
                  .withActions(SQSActions.SendMessage)
                  .withResources(new Resource(sqsQueueArn))
                  .withConditions(new
 Condition().withType("ArnEquals").withConditionKey("aws:SourceArn").withValues(snsTopicArn))
                  );


          Map queueAttributes = new HashMap();
          queueAttributes.put(QueueAttributeName.Policy.toString(),
 policy.toJson());
          sqs.setQueueAttributes(new SetQueueAttributesRequest(sqsQueueUrl,
 queueAttributes));


         System.out.println("Topic arn: " + snsTopicArn);
         System.out.println("Queue arn: " + sqsQueueArn);
         System.out.println("Queue url: " + sqsQueueUrl);
         System.out.println("Queue sub arn: " + sqsSubscriptionArn );
     }
    static void DeleteTopicandQueue()
    {
        if (sqs !=null) {
            sqs.deleteQueue(sqsQueueUrl);
            System.out.println("SQS queue deleted");
        }

        if (sns!=null) {
            sns.deleteTopic(snsTopicArn);
            System.out.println("SNS topic deleted");
        }
    }
}
```

Python

In the function `main`:

- Replace `roleArn` with the ARN of the IAM service role that you created in step 7 of To configure Amazon Rekognition Video (p. 66).
- Replace the values of `bucket` and `video` with the bucket and video file name that you specified 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
import json
import sys
import time



class VideoDetect:
    jobId = ''
    rek = boto3.client('rekognition')
    sqs = boto3.client('sqs')
    sns = boto3.client('sns')

    roleArn = ''
    bucket = ''
    video = ''
    startJobId = ''

    sqsQueueUrl = ''
    snsTopicArn = ''
    processType = ''

    def __init__(self, role, bucket, video):
        self.roleArn = role
        self.bucket = bucket
        self.video = video

    def GetSQSMessageSuccess(self):

        jobFound = False
        succeeded = False

        dotLine=0
        while jobFound == False:
            sqsResponse = self.sqs.receive_message(QueueUrl=self.sqsQueueUrl,
 MessageAttributeNames=['ALL'],
                                          MaxNumberOfMessages=10)

            if sqsResponse:

                if 'Messages' not in sqsResponse:
                    if dotLine<40:
                        print('.', end='')
                        dotLine=dotLine+1
                    else:
                        print()
                        dotLine=0
                    sys.stdout.flush()
                    time.sleep(5)
                    continue

                for message in sqsResponse['Messages']:
                    notification = json.loads(message['Body'])
                    rekMessage = json.loads(notification['Message'])
                    print(rekMessage['JobId'])
                    print(rekMessage['Status'])
                    if rekMessage['JobId'] == self.startJobId:
                        print('Matching Job Found:' + rekMessage['JobId'])
                        jobFound = True
```

```
                        if (rekMessage['Status']=='SUCCEEDED'):
                            succeeded=True

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


        return succeeded

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

        self.startJobId=response['JobId']
        print('Start Job Id: ' + self.startJobId)


    def GetLabelDetectionResults(self):
        maxResults = 10
        paginationToken = ''
        finished = False

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

            print('Codec: ' + response['VideoMetadata']['Codec'])
            print('Duration: ' + str(response['VideoMetadata']['DurationMillis']))
            print('Format: ' + response['VideoMetadata']['Format'])
            print('Frame rate: ' + str(response['VideoMetadata']['FrameRate']))
            print()

            for labelDetection in response['Labels']:
                label=labelDetection['Label']

                print("Timestamp: " + str(labelDetection['Timestamp']))
                print("   Label: " + label['Name'])
                print("   Confidence: " +  str(label['Confidence']))
                print("   Instances:")
                for instance in label['Instances']:
                    print ("      Confidence: " + str(instance['Confidence']))
                    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()
                print()
                print ("   Parents:")
                for parent in label['Parents']:
                    print ("      " + parent['Name'])
                print ()
```

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


    def CreateTopicandQueue(self):

        millis = str(int(round(time.time() * 1000)))

        #Create SNS topic

        snsTopicName="AmazonRekognitionExample" + millis

        topicResponse=self.sns.create_topic(Name=snsTopicName)
        self.snsTopicArn = topicResponse['TopicArn']

        #create SQS queue
        sqsQueueName="AmazonRekognitionQueue" + millis
        self.sqs.create_queue(QueueName=sqsQueueName)
        self.sqsQueueUrl = self.sqs.get_queue_url(QueueName=sqsQueueName)
['QueueUrl']

        attribs = self.sqs.get_queue_attributes(QueueUrl=self.sqsQueueUrl,
                                                    AttributeNames=['QueueArn'])
['Attributes']

        sqsQueueArn = attribs['QueueArn']

        # Subscribe SQS queue to SNS topic
        self.sns.subscribe(
            TopicArn=self.snsTopicArn,
            Protocol='sqs',
            Endpoint=sqsQueueArn)

        #Authorize SNS to write SQS queue
        policy = """{{
  "Version":"2012-10-17",
  "Statement":[
    {{
      "Sid":"MyPolicy",
      "Effect":"Allow",
      "Principal" : {{"AWS" : "*"}},
      "Action":"SQS:SendMessage",
      "Resource": "{}",
      "Condition":{{
        "ArnEquals":{{
          "aws:SourceArn": "{}"
        }}
      }}
    }}
  ]
}}""".format(sqsQueueArn, self.snsTopicArn)

        response = self.sqs.set_queue_attributes(
            QueueUrl = self.sqsQueueUrl,
            Attributes = {
                'Policy' : policy
            })

    def DeleteTopicandQueue(self):
        self.sqs.delete_queue(QueueUrl=self.sqsQueueUrl)
        self.sns.delete_topic(TopicArn=self.snsTopicArn)


def main():
```

```
    roleArn = ''
    bucket = ''
    video = ''

    analyzer=VideoDetect(roleArn, bucket,video)
    analyzer.CreateTopicandQueue()

    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()

    analyzer.DeleteTopicandQueue()


if __name__ == "__main__":
    main()
```

Node.Js

In the following sample code:

- Replace the value of `REGION` with the name of your account's operating region.
- Replace the value of `bucket` with the name of the Amazon S3 bucket containing your video file.
- Replace the value of `videoName` with the name of the video file in your Amazon S3 bucket.
- Replace `roleArn` with the ARN of the IAM service role that you created in step 7 of To configure Amazon Rekognition Video (p. 66).

```
// snippet-start:[sqs.JavaScript.queues.createQueueV3]
// Import required AWS SDK clients and commands for Node.js
import { CreateQueueCommand, GetQueueAttributesCommand, GetQueueUrlCommand,
  SetQueueAttributesCommand, DeleteQueueCommand, ReceiveMessageCommand,
 DeleteMessageCommand } from  "@aws-sdk/client-sqs";
import {CreateTopicCommand, SubscribeCommand, DeleteTopicCommand } from "@aws-sdk/
client-sns";
import  { SQSClient } from "@aws-sdk/client-sqs";
import  { SNSClient } from "@aws-sdk/client-sns";
import  { RekognitionClient, StartLabelDetectionCommand, GetLabelDetectionCommand }
 from "@aws-sdk/client-rekognition";
import { stdout } from "process";

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Create SNS service object.
const sqsClient = new SQSClient({ region: REGION });
const snsClient = new SNSClient({ region: REGION });
const rekClient = new RekognitionClient({ region: REGION });

// Set bucket and video variables
const bucket = "bucket-name";
const videoName = "video-name";
const roleArn = "role-arn"
var startJobId = ""

var ts = Date.now();
const snsTopicName = "AmazonRekognitionExample" + ts;
const snsTopicParams = {Name: snsTopicName}
const sqsQueueName = "AmazonRekognitionQueue-" + ts;

// Set the parameters
const sqsParams = {
```

```
  QueueName: sqsQueueName, //SQS_QUEUE_URL
  Attributes: {
    DelaySeconds: "60", // Number of seconds delay.
    MessageRetentionPeriod: "86400", // Number of seconds delay.
  },
};

const createTopicandQueue = async () => {
  try {
    // Create SNS topic
    const topicResponse = await snsClient.send(new
 CreateTopicCommand(snsTopicParams));
    const topicArn = topicResponse.TopicArn
    console.log("Success", topicResponse);
    // Create SQS Queue
    const sqsResponse = await sqsClient.send(new CreateQueueCommand(sqsParams));
    console.log("Success", sqsResponse);
    const sqsQueueCommand = await sqsClient.send(new GetQueueUrlCommand({QueueName:
 sqsQueueName}))
    const sqsQueueUrl = sqsQueueCommand.QueueUrl
    const attribsResponse = await sqsClient.send(new
 GetQueueAttributesCommand({QueueUrl: sqsQueueUrl, AttributeNames: ['QueueArn']}))
    const attribs = attribsResponse.Attributes
    console.log(attribs)
    const queueArn = attribs.QueueArn
    // subscribe SQS queue to SNS topic
    const subscribed = await snsClient.send(new SubscribeCommand({TopicArn:
 topicArn, Protocol:'sqs', Endpoint: queueArn}))
    const policy = {
      Version: "2012-10-17",
      Statement: [
        {
          Sid: "MyPolicy",
          Effect: "Allow",
          Principal: {AWS: "*"},
          Action: "SQS:SendMessage",
          Resource: queueArn,
          Condition: {
            ArnEquals: {
              'aws:SourceArn': topicArn
            }
          }
        }
      ]
    };

    const response = sqsClient.send(new SetQueueAttributesCommand({QueueUrl:
 sqsQueueUrl, Attributes: {Policy: JSON.stringify(policy)}}))
    console.log(response)
    console.log(sqsQueueUrl, topicArn)
    return [sqsQueueUrl, topicArn]

  } catch (err) {
    console.log("Error", err);
  }
};

const startLabelDetection = async (roleArn, snsTopicArn) => {
  try {
    //Initiate label detection and update value of startJobId with returned Job ID
   const labelDetectionResponse = await rekClient.send(new
 StartLabelDetectionCommand({Video:{S3Object:{Bucket:bucket, Name:videoName}},
      NotificationChannel:{RoleArn: roleArn, SNSTopicArn: snsTopicArn}}));
      startJobId = labelDetectionResponse.JobId
      console.log(`JobID: ${startJobId}`)
      return startJobId
```

```
  } catch (err) {
    console.log("Error", err);
  }
};

const getLabelDetectionResults = async(startJobId) => {
  console.log("Retrieving Label Detection results")
  // Set max results, paginationToken and finished will be updated depending on
 response values
  var maxResults = 10
  var paginationToken = ''
  var finished = false

  // Begin retrieving label detection results
  while (finished == false){
    var response = await rekClient.send(new GetLabelDetectionCommand({JobId:
 startJobId, MaxResults: maxResults,
      NextToken: paginationToken, SortBy:'TIMESTAMP'}))
      // Log metadata
      console.log(`Codec: ${response.VideoMetadata.Codec}`)
      console.log(`Duration: ${response.VideoMetadata.DurationMillis}`)
      console.log(`Format: ${response.VideoMetadata.Format}`)
      console.log(`Frame Rate: ${response.VideoMetadata.FrameRate}`)
      console.log()
      // For every detected label, log label, confidence, bounding box, and
 timestamp
      response.Labels.forEach(labelDetection => {
        var label = labelDetection.Label
        console.log(`Timestamp: ${labelDetection.Timestamp}`)
        console.log(`Label: ${label.Name}`)
        console.log(`Confidence: ${label.Confidence}`)
        console.log("Instances:")
        label.Instances.forEach(instance =>{
          console.log(`Confidence: ${instance.Confidence}`)
          console.log("Bounding Box:")
          console.log(`Top: ${instance.Confidence}`)
          console.log(`Left: ${instance.Confidence}`)
          console.log(`Width: ${instance.Confidence}`)
          console.log(`Height: ${instance.Confidence}`)
          console.log()
        })
      console.log()
      // Log parent if found
      console.log("   Parents:")
      label.Parents.forEach(parent =>{
        console.log(`    ${parent.Name}`)
      })
      console.log()
      // Searh for pagination token, if found, set variable to next token
      if (String(response).includes("NextToken")){
        paginationToken = response.NextToken

      }else{
        finished = true
      }

      })
  }
}

// Checks for status of job completion
const getSQSMessageSuccess = async(sqsQueueUrl, startJobId) => {
  try {
    // Set job found and success status to false initially
    var jobFound = false
    var succeeded = false
```

```
    var dotLine = 0
    // while not found, continue to poll for response
    while (jobFound == false){
      var sqsReceivedResponse = await sqsClient.send(new
 ReceiveMessageCommand({QueueUrl:sqsQueueUrl,
        MaxNumberOfMessages:'ALL', MaxNumberOfMessages:10}));
      if (sqsReceivedResponse){
        var responseString = JSON.stringify(sqsReceivedResponse)
        if (!responseString.includes('Body')){
          if (dotLine < 40) {
            console.log('.')
            dotLine = dotLine + 1
          }else {
            console.log('')
            dotLine = 0
          };
          stdout.write('', () => {
            console.log('');
          });
          await new Promise(resolve => setTimeout(resolve, 5000));
          continue
        }
      }

      // Once job found, log Job ID and return true if status is succeeded
      for (var message of sqsReceivedResponse.Messages){
        console.log("Retrieved messages:")
        var notification = JSON.parse(message.Body)
        var rekMessage = JSON.parse(notification.Message)
        var messageJobId = rekMessage.JobId
        if (String(rekMessage.JobId).includes(String(startJobId))){
          console.log('Matching job found:')
          console.log(rekMessage.JobId)
          jobFound = true
          console.log(rekMessage.Status)
          if (String(rekMessage.Status).includes(String("SUCCEEDED"))){
            succeeded = true
            console.log("Job processing succeeded.")
            var sqsDeleteMessage = await sqsClient.send(new
 DeleteMessageCommand({QueueUrl:sqsQueueUrl,
 ReceiptHandle:message.ReceiptHandle}));
          }
        }else{
          console.log("Provided Job ID did not match returned ID.")
          var sqsDeleteMessage = await sqsClient.send(new
 DeleteMessageCommand({QueueUrl:sqsQueueUrl,
 ReceiptHandle:message.ReceiptHandle}));
        }
      }
    }
  return succeeded
  } catch(err) {
    console.log("Error", err);
  }
};

// Start label detection job, sent status notification, check for success status
// Retrieve results if status is "SUCEEDED", delete notification queue and topic
const runLabelDetectionAndGetResults = async () => {
  try {
    const sqsAndTopic = await createTopicandQueue();
    const startLabelDetectionRes = await startLabelDetection(roleArn,
 sqsAndTopic[1]);
    const getSQSMessageStatus = await getSQSMessageSuccess(sqsAndTopic[0],
 startLabelDetectionRes)
    console.log(getSQSMessageSuccess)
```

```
    if (getSQSMessageSuccess){
      console.log("Retrieving results:")
      const results = await getLabelDetectionResults(startLabelDetectionRes)
    }
    const deleteQueue = await sqsClient.send(new DeleteQueueCommand({QueueUrl:
 sqsAndTopic[0]}));
    const deleteTopic = await snsClient.send(new DeleteTopicCommand({TopicArn:
 sqsAndTopic[1]}));
    console.log("Successfully deleted.")
  } catch (err) {
    console.log("Error", err);
  }
};

runLabelDetectionAndGetResults()
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void startLabels(RekognitionClient rekClient,
                                   NotificationChannel channel,
                                   String bucket,
                                   String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                .bucket(bucket)
                .name(video)
                .build();

            Video vidOb = Video.builder()
                .s3Object(s3Obj)
                .build();

            StartLabelDetectionRequest labelDetectionRequest =
 StartLabelDetectionRequest.builder()
                .jobTag("DetectingLabels")
                .notificationChannel(channel)
                .video(vidOb)
                .minConfidence(50F)
                .build();

            StartLabelDetectionResponse labelDetectionResponse =
 rekClient.startLabelDetection(labelDetectionRequest);
            startJobId = labelDetectionResponse.jobId();

            boolean ans = true;
            String status = "";
            int yy = 0;
            while (ans) {

                GetLabelDetectionRequest detectionRequest =
 GetLabelDetectionRequest.builder()
                    .jobId(startJobId)
                    .maxResults(10)
                    .build();

                GetLabelDetectionResponse result =
 rekClient.getLabelDetection(detectionRequest);
                status = result.jobStatusAsString();

                if (status.compareTo("SUCCEEDED") == 0)
                    ans = false;
```

```
                else
                    System.out.println(yy +" status is: "+status);

                Thread.sleep(1000);
                yy++;
            }

            System.out.println(startJobId +" status is: "+status);

        } catch(RekognitionException | InterruptedException e) {
            e.getMessage();
            System.exit(1);
        }
    }

    public static void getLabelJob(RekognitionClient rekClient, SqsClient sqs,
String queueUrl) {

        List<Message> messages;
        ReceiveMessageRequest messageRequest = ReceiveMessageRequest.builder()
            .queueUrl(queueUrl)
            .build();

        try {
            messages = sqs.receiveMessage(messageRequest).messages();

            if (!messages.isEmpty()) {
                for (Message message: messages) {
                    String notification = message.body();

                    // Get the status and job id from the 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 in JSON is " + operationJobId);

                    DeleteMessageRequest deleteMessageRequest =
DeleteMessageRequest.builder()
                        .queueUrl(queueUrl)
                        .build();

                    String jobId = operationJobId.textValue();
                    if (startJobId.compareTo(jobId)==0) {
                        System.out.println("Job id: " + operationJobId );
                        System.out.println("Status : " +
operationStatus.toString());

                        if (operationStatus.asText().equals("SUCCEEDED"))
                            GetResultsLabels(rekClient);
                        else
                            System.out.println("Video analysis failed");

                        sqs.deleteMessage(deleteMessageRequest);
                    }

                    else{
                        System.out.println("Job received was not job " +
startJobId);
                        sqs.deleteMessage(deleteMessageRequest);
                    }
                }
```

```
            }

        } catch(RekognitionException e) {
            e.getMessage();
            System.exit(1);
        } catch (JsonMappingException e) {
            e.printStackTrace();
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
    }

    // Gets the job results by calling GetLabelDetection
    private static void GetResultsLabels(RekognitionClient rekClient) {

        int maxResults=10;
        String paginationToken=null;
        GetLabelDetectionResponse labelDetectionResult=null;

        try {
            do {
                if (labelDetectionResult !=null)
                    paginationToken = labelDetectionResult.nextToken();


                GetLabelDetectionRequest labelDetectionRequest=
GetLabelDetectionRequest.builder()
                    .jobId(startJobId)
                    .sortBy(LabelDetectionSortBy.TIMESTAMP)
                    .maxResults(maxResults)
                    .nextToken(paginationToken)
                    .build();

                labelDetectionResult =
rekClient.getLabelDetection(labelDetectionRequest);
                VideoMetadata videoMetaData=labelDetectionResult.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " + videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());

                List<LabelDetection> detectedLabels= labelDetectionResult.labels();
                for (LabelDetection detectedLabel: detectedLabels) {
                    long seconds=detectedLabel.timestamp();
                    Label label=detectedLabel.label();
                    System.out.println("Millisecond: " + seconds + " ");

                    System.out.println("   Label:" + label.name());
                    System.out.println("   Confidence:" +
detectedLabel.label().confidence().toString());

                    List<Instance> instances = label.instances();
                    System.out.println("   Instances of " + label.name());

                    if (instances.isEmpty()) {
                        System.out.println("        " + "None");
                    } else {
                        for (Instance instance : instances) {
                            System.out.println("        Confidence: " +
instance.confidence().toString());
                            System.out.println("        Bounding box: " +
instance.boundingBox().toString());
                        }
                    }
                    System.out.println("   Parent labels for " + label.name() +
":");
```

```
                    List<Parent> parents = label.parents();

                    if (parents.isEmpty()) {
                        System.out.println("        None");
                    } else {
                        for (Parent parent : parents) {
                            System.out.println("    " + parent.name());
                        }
                    }
                    System.out.println();
                }
            } while (labelDetectionResult !=null &&
labelDetectionResult.nextToken() != null);

        } catch(RekognitionException e) {
            e.getMessage();
            System.exit(1);
        }
    }
```

4. 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. 115).

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. 61). 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. 87). 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, Actions Defined by Amazon Rekognition.

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. Upload an MOV or MPEG-4 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 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}" \
--notification-channel "SNSTopicArn=TopicARN,RoleArn=RoleARN" \
--endpoint-url Endpoint \
--region us-east-1
```

   Update the following values:

   - Change `bucketname` and `videofile` to the Amazon S3 bucket name and file name that you specified in step 2.
   - Change `us-east-1` to the AWS region that you're using.
   - Change `TopicARN` to the ARN of the Amazon SNS topic you created in step 3 of Configuring Amazon Rekognition Video (p. 66).
   - Change `RoleARN` to the ARN of the IAM service role you created in step 7 of Configuring Amazon Rekognition Video (p. 66).
   - If required, you can specify the `endpoint-url`. The AWS CLI should automatically determine the proper endpoint URL based on the provided region. However, if you are using an endpoint from your private VPC, you may need to specify the `endpoint-url`. The AWS Service Endpoints resource lists the syntax for specifying endpoint urls and the names and codes for each region.
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 `SUCCEEDED`, run the following AWS CLI command to show the label detection results.

```
aws rekognition get-label-detection  --job-id JobId \
--region us-east-1
```

   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.

   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"
            }
        }, .......
```

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. 62). 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.

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.
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",
  "API": "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 and that your service role's permissions are securely scoped to the resouces you are using. Carry out 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. The following example trust policy specifies the user's role credentials and restricts the service role's permissions to just the resources you are using (for more information on securely limiting the scope of a service role's permissions, see Cross-service confused deputy prevention (p. 505)):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "rekognition.amazonaws.com",
        "AWS": "arn:User ARN"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
      "StringEquals": {
                    "aws:SourceAccount": "Account ID"
                },
       "StringLike": {
                    "aws:SourceArn":
 "arn:aws:rekognition:region:111122223333:streamprocessor/*"
                }
      }
    }
  ]
}
```

- 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. 67).
  - 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.

I need additional help troubleshooting the Amazon SNS topic

You can use AWS X-Ray with Amazon SNS to trace and analyze the messages that travel through your application. For more information, see Amazon SNS and AWS X-Ray.

For additional help, you can post your question to the Amazon Rekognition forum or consider signing up for AWS technical support.

Working with streaming video events

You can use Amazon Rekognition Video to detect and recognize faces or detect objects in streaming video. Amazon Rekognition Video uses Amazon Kinesis Video Streams to receive and process a video stream. You create a stream processor with parameters that show what you want the stream processor to detect from the video stream. Rekognition sends label detection results from streaming video events as Amazon SNS and Amazon S3 notifications. Rekognition outputs face search results to a Kinesis data stream.

Face search stream processors use `FaceSearchSettings` to search for faces from a collection. For more information about how to implement face search stream processors to analyze faces in streaming video, see the section called “Searching faces in a collection in streaming video” (p. 216).

Label detection stream processors use `ConnectedHomeSettings` to search for people, packages, and pets in streaming video events. For more information about how to implement label detection stream processors, see the section called “Detecting labels in streaming video events” (p. 117).

Overview of Amazon Rekognition Video stream processor operations

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. The request parameters for creating a face search stream processor 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. The request parameters for creating a security monitoring stream processor include the Amazon Resource Names (ARNs) for the Kinesis video stream and the Amazon SNS topic, the types of objects you want to detect in the video stream, and information for an Amazon S3 bucket for the output results. You also include a name that you specify for the stream processor.

You start processing a video by calling the StartStreamProcessor operation. To get status information for a stream processor, call DescribeStreamProcessor. Other operations you can call are TagResource to tag a stream processor and DeleteStreamProcessor to delete a stream processor. If you are using a face search stream processor, you can also use StopStreamProcessor to stop a stream processor. To get a list of stream processors in your account, call ListStreamProcessors.

After the stream processor starts running, you stream the video into Amazon Rekognition Video through the Kinesis video stream that you specified in `CreateStreamProcessor`. You can 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 from a face search stream processor, see Reading streaming video analysis results (p. 226).

Tagging the Amazon Rekognition Video stream processor

You can identify, organize, search for, and filter Amazon Rekognition stream processors by using tags. Each tag is a label consisting of a user-defined key and value.

Topics

Add tags to a new stream processor

You can add tags to a stream processor as you create it using the `CreateStreamProcessor` operation. Specify one or more tags in the `Tags` array input parameter. The following is a JSON example for the `CreateStreamProcessor` request with tags.

```
{
       "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
              },
              "Tags": {
      "Dept": "Engineering",
        "Name": "Ana Silva Carolina",
        "Role": "Developer"
       }
}
```

Add tags to an existing stream processor

To add one or more tags to an existing stream processor, use the `TagResource` operation. Specify the stream processor's Amazon Resource Name (ARN) (`ResourceArn`) and the tags (`Tags`) that you want to add. The following example shows how to add two tags.

```
aws rekognition tag-resource --resource-arn resource-arn \
                --tags '{"key1":"value1","key2":"value2"}'
```

> **Note**
> If you do not know the stream processor's Amazon Resource Name, you can use the `DescribeStreamProcessor` operation.

List tags in a stream processor

To list the tags attached to a stream processor, use the `ListTagsForResource` operation and specify the ARN of the stream processor (`ResourceArn`). The response is a map of tag keys and values that are attached to the specified stream processor.

```
aws rekognition list-tags-for-resource --resource-arn resource-arn
```

The output displays a list of tags attached to the stream processor:

```
                {
    "Tags": {
        "Dept": "Engineering",
        "Name": "Ana Silva Carolina",
        "Role": "Developer"
    }
}
```

Delete tags from a stream processor

To remove one or more tags from a stream processor, use the `UntagResource` operation. Specify the ARN of the model (`ResourceArn`) and the tag keys (`Tag-Keys`) that you want to remove.

```
aws rekognition untag-resource --resource-arn resource-arn \
                --tag-keys '["key1","key2"]'
```

Alternatively, you can specify tag-keys in this format:

```
--tag-keys key1,key2
```

Error handling

This section describes runtime errors and how to handle them. It also describes error messages and codes that are specific to Amazon Rekognition.

Topics

Error components

When your program sends a request, Amazon Rekognition attempts to process it. If the request is successful, Amazon Rekognition returns an HTTP success status code (`200 OK`), along with the results from the requested operation.

If the request is unsuccessful, Amazon Rekognition returns an error. Each error has three components:

- An HTTP status code (such as `400`).
- An exception name (such as `InvalidS3ObjectException`).

- An error message (such as `Unable to get object metadata from S3. Check object key, region and/or access permissions.`).

The AWS SDKs take care of propagating errors to your application, so that you can take appropriate action. For example, in a Java program, you can write `try-catch` logic to handle a `ResourceNotFoundException`.

If you're not using an AWS SDK, you need to parse the content of the low-level response from Amazon Rekognition. The following is an example of such a response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/x-amz-json-1.1
Date: Sat, 25 May 2019 00:28:25 GMT
x-amzn-RequestId: 03507c9b-7e84-11e9-9ad1-854a4567eb71
Content-Length: 222
Connection: keep-alive

{"__type":"InvalidS3ObjectException","Code":"InvalidS3ObjectException","Logref":"5022229e-7e48-11e9-9ad
 to get object metadata from S3. Check object key, region and/or access permissions."}
```

Error messages and codes

The following is a list of exceptions that Amazon Rekognition returns, grouped by HTTP status code. If *OK to retry?* is *Yes*, you can submit the same request again. If *OK to retry?* is *No*, you need to fix the problem on the client side before you submit a new request.

HTTP status code 400

An HTTP `400` status code indicates a problem with your request. Some examples of problems are authentication failure, required parameters that are missing, or exceeding an operation's provisioned throughput. You have to fix the issue in your application before submitting the request again.

AccessDeniedException

Message: *An error occurred (AccessDeniedException) when calling the <Operation> operation: User: <User ARN> is not authorized to perform: <Operation> on resource: <Resource ARN>.*

You aren't authorized to perform the action. Use the Amazon Resource Name (ARN) of an authorized user or IAM role to perform the operation.

OK to retry? No

GroupFacesInProgressException

Message: *Failed to schedule GroupFaces job. There is an existing group faces job for this collection.*

Retry the operation after the existing job finishes.

OK to retry? No

IdempotentParameterMismatchException

Message: *The ClientRequestToken: <Token> you have supplied is already in use.*

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.

OK to retry? No

ImageTooLargeException

Message: *Image size is too large.*

The input image size exceeds the allowed limit. If you are calling DetectProtectiveEquipment, the image size or resolution exceeds the allowed limit. For more information, see Guidelines and quotas in Amazon Rekognition (p. 508).

OK to retry? No

InvalidImageFormatException

Message: *Request has invalid image format.*

The provided image format isn't supported. Use a supported image format (.JPEG and .PNG). For more information, see Guidelines and quotas in Amazon Rekognition (p. 508).

OK to retry? No

InvalidPaginationTokenException

Messages

- *Invalid Token*
- *Invalid Pagination Token*

The pagination token in the request isn't valid. The token might have expired.

OK to retry? No

InvalidParameterException

Message: *Request has invalid parameters.*

An input parameter violated a constraint. Validate your parameters before calling the API operation again.

OK to retry? No

InvalidS3ObjectException

Messages:

- *Request has invalid S3 object.*
- *Unable to get object metadata from S3. Check object key, region and/or access permissions.*

Amazon Rekognition is unable to access the S3 object that was specified in the request. For more information, see Configure Access to S3: AWS S3 Managing Access. For troubleshooting information, see Troubleshooting Amazon S3.

OK to retry? No

LimitExceededException

Messages:

- *Stream processor limit exceeded for account, limit - <Current Limit>.*
- *<Number of Open Jobs> open Jobs for User <User ARN> Maximum limit: <Maximum Limit>*

An Amazon Rekognition service limit was exceeded. For example, if you start too many Amazon Rekognition Video jobs concurrently, calls to start operations, such as `StartLabelDetection`, raise a `LimitExceededException` exception (HTTP status code: 400) until the number of concurrently running jobs is below the Amazon Rekognition service limit.

OK to retry? No

ProvisionedThroughputExceededException

Messages:

- *Provisioned Rate exceeded.*
- *S3 download limit exceeded.*

The number of requests exceeded your throughput limit. For more information, see Amazon Rekognition Service Limits.

To request a limit increase, follow the instructions at the section called "Create a case to change TPS quotas" (p. 510).

OK to retry? Yes

ResourceAlreadyExistsException

Message: *The collection id: <Collection Id> already exists.*

A collection with the specified ID already exists.

OK to retry? No

ResourceInUseException

Messages:

- *Stream processor name already in use.*
- *Specified resource is in use.*
- *Processor not available for stopping stream.*
- *Cannot delete stream processor.*

Retry when the resource is available.

OK to retry? No

ResourceNotFoundException

Message: Various messages depending on the API call.

The specified resource doesn't exist.

OK to retry? No

ThrottlingException

Message: *Slow down; sudden increase in rate of requests.*

Your rate of request increase is too fast. Slow down your request rate and gradually increase it. We recommend that you back off exponentially and retry. By default, the AWS SDKs use automatic retry logic and exponential backoff. For more information, see Error Retries and Exponential Backoff in AWS and Exponential Backoff and Jitter.

OK to retry? Yes

VideoTooLargeException

Message: *Video size in bytes: <Video Size> is more than the maximum limit of: <Max Size> bytes.*

The file size or duration of the supplied media is too large. For more information, see Guidelines and quotas in Amazon Rekognition (p. 508).

OK to retry? No

HTTP status code 5xx

An HTTP `5xx` status code indicates a problem that must be resolved by AWS. This might be a transient error. If it is, you can retry your request until it succeeds. Otherwise, go to the AWS Service Health Dashboard to see if there are any operational issues with the service.

InternalServerError (HTTP 500)

Message: *Internal server error*

Amazon Rekognition experienced a service issue. Try your call again. You should back off exponentially and retry. By default, the AWS SDKs use automatic retry logic and exponential backoff. For more information, see Error Retries and Exponential Backoff in AWS and Exponential Backoff and Jitter.

OK to retry? Yes

ThrottlingException (HTTP 500)

Message: *Service Unavailable*

Amazon Rekognition is temporarily unable to process the request. Try your call again. We recommend that you back off exponentially and retry. By default, the AWS SDKs use automatic retry logic and exponential backoff. For more information, see Error Retries and Exponential Backoff in AWS and Exponential Backoff and Jitter.

OK to retry? Yes

Error handling in your application

For your application to run smoothly, you need to add logic to catch errors and respond to them. Typical approaches include using `try-catch` blocks or `if-then` statements.

The AWS SDKs perform their own retries and error checking. If you encounter an error while using one of the AWS SDKs, the error code and description can help you troubleshoot it.

You should also see a `Request ID` in the response. The `Request ID` can be helpful if you need to work with AWS Support to diagnose an issue.

The following Java code snippet attempts to detect objects in an image and performs rudimentary error handling. (In this case, it informs the user that the request failed.)

```
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) {
    System.err.println("Could not complete operation");
    System.err.println("Error Message:  " + e.getMessage());
    System.err.println("HTTP Status:    " + e.getStatusCode());
    System.err.println("AWS Error Code: " + e.getErrorCode());
    System.err.println("Error Type:     " + e.getErrorType());
    System.err.println("Request ID:     " + e.getRequestId());
}
catch (AmazonClientException ace) {
    System.err.println("Internal error occurred communicating with Rekognition");
    System.out.println("Error Message:  " + ace.getMessage());
}
```

In this code snippet, the `try-catch` construct handles two different kinds of exceptions:

- `AmazonRekognitionException` –This exception occurs if the client request was correctly transmitted to Amazon Rekognition, but Amazon Rekognition couldn't process the request and returned an error response instead.
- `AmazonClientException` – This exception occurs if the client couldn't get a response from a service, or if the client couldn't parse the response from a service.

Using Amazon Rekognition as a FedRAMP authorized service

The AWS FedRAMP compliance program includes Amazon Rekognition as a FedRAMP-authorized service. If you're a federal or commercial customer, you can use the service to process and store sensitive workloads in the AWS US East and US West Regions, with data up to the moderate-impact level. You can use the service for sensitive workloads in the AWS GovCloud (US) Region's authorization boundary, with data up to the high-impact level. For more information about FedRAMP compliance, see AWS FedRAMP Compliance.

To be FedRAMP compliant, you can use a Federal Information Processing Standard (FIPS) endpoint. This gives you access to FIPS 140-2 validated cryptographic modules when you're working with sensitive information. For more information about FIPS endpoints, see FIPS 140-2 Overview.

You can use the AWS Command Line Interface (AWS CLI) or one of the AWS SDKs to specify the endpoint that is used by Amazon Rekognition.

For endpoints that can be used with Amazon Rekognition, see Amazon Rekognition Regions and Endpoints.

The following are examples from the Listing Collections (p. 167) topic in the *Amazon Rekognition Developer Guide*. They are modified to specify the Region and FIPS endpoint through which Amazon Rekognition is accessed.

Java

For Java, use the `withEndpointConfiguration` method when you construct the Amazon Rekognition client. This example shows the collections you have that use the FIPS endpoint in the US East (N.Virginia) Region:

```
//Copyright 2019 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 {


      AmazonRekognition amazonRekognition = AmazonRekognitionClientBuilder.standard()
         .withEndpointConfiguration(new
 AwsClientBuilder.EndpointConfiguration("https://rekognition-fips.us-
east-1.amazonaws.com","us-east-1"))
         .build();


      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

For the AWS CLI, use the `--endpoint-url` argument to specify the endpoint through which Amazon Rekognition is accessed. This example shows the collections you have that use the FIPS endpoint in the US East (Ohio) Region:

```
aws rekognition list-collections --endpoint-url https://rekognition-fips.us-
east-2.amazonaws.com --region us-east-2
```

Python

For Python, use the `endpoint_url` argument in the boto3.client function. Set it to the endpoint that you want to specify. This example shows the collections you have that use the FIPS endpoint in the US West (Oregon) 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

def list_collections():

    max_results=2

    client=boto3.client('rekognition', endpoint_url='https://rekognition-fips.us-west-2.amazonaws.com', region_name='us-west-2')

    #Display all the collections
    print('Displaying collections...')
    response=client.list_collections(MaxResults=max_results)
    collection_count=0
    done=False

    while done==False:
        collections=response['CollectionIds']

        for collection in collections:
            print (collection)
            collection_count+=1
        if 'NextToken' in response:
            nextToken=response['NextToken']

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

        else:
            done=True

    return collection_count

def main():

    collection_count=list_collections()
    print("collections: " + str(collection_count))
if __name__ == "__main__":
    main()
```

Best practices for sensors, input images, and videos

This section contains best practice information for using Amazon Rekognition.

Topics

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 that contains 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 is probably faster than passing image bytes to the operation.

Recommendations for facial comparison input images

The models used for face comparison operations are designed to work for a wide variety of poses, facial expressions, age ranges, rotations, lighting conditions, and sizes. We recommend that you use the following guidelines when choosing reference photos for CompareFaces or for adding faces to a collection using IndexFaces.

- Use an image with a face that is within the recommended range of angles. The pitch should be less than 30 degrees face down and less than 45 degrees face up. The yaw should be less than 45 degrees in either direction. There is no restriction on the roll.

- Use an image of a face with both eyes open and visible.

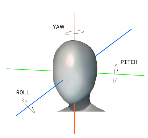
- When creating a collection using `IndexFaces`, use multiple face images of an individual with different pitches and yaws (within the recommended range of angles). We recommend that at least five images of the person are indexed—straight on, face turned left with a yaw of 45 degrees or less, face turned right with a yaw of 45 degrees or less, face tilted down with a pitch of 30 degrees or less, and face tilted up with a pitch of 45 degrees or less. If you want to track that these face instances belong to the

same individual, consider using the external image ID attribute if there is only one face in the image being indexed. For example, five images of John Doe can be tracked in the collection with external image IDs as John_Doe_1.jpg, … John_Doe_5.jpg.

- Use an image of 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 items that block the face, such as headbands and masks.
- Use an image of a face that occupies a large proportion of the image. Images where the face occupies a larger portion of the image are matched with greater accuracy.
- Ensure that images are sufficiently large in terms of resolution. Amazon Rekognition can recognize faces as small as 50 x 50 pixels in image resolutions up to 1920 x 1080. Higher-resolution images require a larger minimum face size. Faces larger than the minimum size provide a more accurate set of facial comparison results.
- Use color images.
- Use images with flat lighting on the face, as opposed to varied lighting such as shadows.
- Use images that have sufficient contrast with the background. A high-contrast monochrome background works well.
- Use images of faces with neutral facial expressions with mouth closed and little to no smile for applications that require high precision.
- Use images that are bright and sharp. Avoid using images that may be blurry due to subject and camera motion as much as possible. DetectFaces can be used to determine the brightness and sharpness of a face.
- Ensure that recent face images are indexed.

Recommendations for camera setup (image and video)

The following recommendations are in addition to Recommendations for facial comparison input images (p. 100).



- Image Resolution – There is no minimum requirement for image resolution, as long as the face resolution is 50 x 50 pixels for images with a total resolution up to 1920 x 1080. Higher-resolution images require a larger minimum face size.

Note

The preceding recommendation is based on the native resolution of the camera. Generating a high-resolution image from a low-resolution image does not produce the results needed for face search (due to artifacts generated by the up-sampling of the image).

- Camera Angle – There are three measurements for camera angle—pitch, roll, and yaw.
  - Pitch – We recommend a pitch of less than 30 degrees when the camera is facing down and less than 45 degrees when the camera is facing up.
  - Roll – There isn’t a minimum requirement for this parameter. Amazon Rekognition can handle any amount of roll.
  - Yaw – We recommend a yaw of less than 45 degrees in either direction.

  The face angle along any axis that is captured by the camera is a combination of both the camera angle facing the scene and the angle at which the subject’s head is in the scene. For example, if the camera is 30 degrees facing down and the person has their head down a further 30 degrees, the actual face pitch as seen by the camera is 60 degrees. In this case, Amazon Rekognition would not be able to recognize the face. We recommend setting up cameras such that the camera angles are based on the assumption that people are generally looking into the camera with the overall pitch (combination of face and camera) at 30 degrees or less.
- Camera Zoom – The recommended minimum face resolution of 50 x 50 pixels should drive this camera setting. We recommend using the zoom setting of a camera so that the desired faces are at a resolution no less than 50 x 50 pixels.
- Camera Height – The recommended camera pitch should drive this parameter.

Recommendations for camera setup (stored and streaming video)

The following recommendations are in addition to Recommendations for camera setup (image and video) (p. 101).

- The codec should be h.264 encoded.
- The recommended frame rate is 30 fps. (It should not be less than 5 fps.)
- The recommended encoder bitrate is 3 Mbps. (It should not be less than 1.5 Mbps.)
- Frame Rate vs. Frame Resolution – If the encoder bitrate is a constraint, we recommend favoring a higher frame resolution over a higher frame rate for better face search results. This ensures that Amazon Rekognition gets the best quality frame within the allocated bitrate. However, there is a downside to this. Because of the low frame rate, the camera misses fast motion in a scene. It's important to understand the trade-offs between these two parameters for a given setup. For example, if the maximum possible bitrate is 1.5 Mbps, a camera can capture 1080p at 5 fps or 720p at 15 fps. The choice between the two is application dependent, as long as the recommended face resolution of 50 x 50 pixels is met.

Recommendations for camera setup (streaming video)

The following recommendation is in addition to Recommendations for camera setup (stored and streaming video) (p. 102).

An additional constraint with streaming applications is internet bandwidth. For live video, Amazon Rekognition only accepts Amazon Kinesis Video Streams as an input. You should understand the dependency between the encoder bitrate and the available network bandwidth. Available bandwidth should, at a minimum, support the same bitrate that the camera is using to encode the live stream. This ensures that whatever the camera captures is relayed through Amazon Kinesis Video Streams. If the available bandwidth is less than the encoder bitrate, Amazon Kinesis Video Streams drops bits based on the network bandwidth. This results in low video quality.

A typical streaming setup involves connecting multiple cameras to a network hub that relays the streams. In this case, the bandwidth should accommodate the cumulative sum of the streams coming from all cameras connected to the hub. For example, if the hub is connected to five cameras encoding at 1.5 Mbps, the available network bandwidth should be at least 7.5 Mbps. To ensure that there are no dropped packets, you should consider keeping the network bandwidth higher than 7.5 Mbps to accommodate for jitters due to dropped connections between a camera and the hub. The actual value depends on the reliability of the internal network.

Detecting labels

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, action, 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 Palm Tree (object), Beach (scene), Running (action), and Outdoors (concept).

To download the full list of labels and object bounding boxes supported by Amazon Rekognition, click here.

> **Note**
> Amazon Rekognition makes gender binary (man, woman, girl, etc.) predictions based on the physical appearance of a person in a particular image. This kind of prediction is not designed to categorize a person’s gender identity, and you shouldn't use Amazon Rekognition to make such a determination. For example, a male actor wearing a long-haired wig and earrings for a role might be predicted as female.
> Using Amazon Rekognition to make gender binary predictions is best suited for use cases where aggregate gender distribution statistics need to be analyzed without identifying specific users. For example, the percentage of users who are women compared to men on a social media platform.
> We don't recommend using gender binary predictions to make decisions that impact an individual's rights, privacy, or access to services.

Amazon Rekognition returns labels in English. You can use Amazon Translate to translate English labels into other languages.

Bounding boxes and parent labels

Amazon Rekognition Image and Amazon Rekognition Video can return the bounding box for common object labels such as cars, furniture, apparel or pets. Bounding box information isn't returned for less common object labels. 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 and Amazon Rekognition Video use 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 and Amazon Rekognition Video both return the version of the label detection model used to detect labels in an image or stored video.

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.

Detecting labels in an image

You can use the DetectLabels operation to detect labels in an image. For an example, see Analyzing images stored in an Amazon S3 bucket (p. 27).

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. 112).

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. 13).

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 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. In the function `main`, 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

def detect_labels(photo, bucket):

    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 ()
    return len(response['Labels'])


def main():
    photo=''
    bucket=''
    label_count=detect_labels(photo, bucket)
    print("Labels detected: " + str(label_count))


if __name__ == "__main__":
    main()
```

.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);
        }
    }
}
```

Ruby

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 2020 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.)

   # Add to your Gemfile
   # gem 'aws-sdk-rekognition'
   require 'aws-sdk-rekognition'
   credentials = Aws::Credentials.new(
      ENV['AWS_ACCESS_KEY_ID'],
      ENV['AWS_SECRET_ACCESS_KEY']
   )
   bucket = 'bucket' # the bucket name without s3://
   photo  = 'photo' # the name of file
   client   = Aws::Rekognition::Client.new credentials: credentials
   attrs = {
     image: {
       s3_object: {
         bucket: bucket,
         name: photo
       },
     },
     max_labels: 10
   }
  response = client.detect_labels attrs
  puts "Detected labels for: #{photo}"
  response.labels.each do |label|
    puts "Label:      #{label.name}"
    puts "Confidence: #{label.confidence}"
    puts "Instances:"
    label['instances'].each do |instance|
      box = instance['bounding_box']
      puts "  Bounding box:"
      puts "    Top:        #{box.top}"
      puts "    Left:       #{box.left}"
      puts "    Width:      #{box.width}"
      puts "    Height:     #{box.height}"
      puts "  Confidence: #{instance.confidence}"
    end
    puts "Parents:"
    label.parents.each do |parent|
      puts "  #{parent.name}"
    end
    puts "------------"
    puts ""
  end
```

Node.js

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.

If you are using TypeScript definitions, you may need to use `import AWS from 'aws-sdk'` instead of `const AWS = require('aws-sdk')`, in order to run the program with Node.js. You can consult the AWS SDK for Javascript for more details. Depending on how you have your configurations set up, you also may need to specify your region with `AWS.config.update({region: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.)


// Load the SDK
var AWS = require('aws-sdk');

const bucket = 'bucket' // the bucketname without s3://
const photo  = 'photo' // the name of file

 const config = new AWS.Config({
  accessKeyId: process.env.AWS_ACCESS_KEY_ID,
  secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
  region: process.env.AWS_REGION
})
const client = new AWS.Rekognition();
const params = {
  Image: {
    S3Object: {
      Bucket: bucket,
      Name: photo
    },
  },
  MaxLabels: 10
}
client.detectLabels(params, function(err, response) {
  if (err) {
    console.log(err, err.stack); // if an error occurred
  } else {
    console.log(`Detected labels for: ${photo}`)
    response.Labels.forEach(label => {
      console.log(`Label:      ${label.Name}`)
      console.log(`Confidence: ${label.Confidence}`)
      console.log("Instances:")
      label.Instances.forEach(instance => {
        let box = instance.BoundingBox
        console.log("  Bounding box:")
        console.log(`    Top:        ${box.Top}`)
        console.log(`    Left:       ${box.Left}`)
        console.log(`    Width:      ${box.Width}`)
        console.log(`    Height:     ${box.Height}`)
        console.log(`  Confidence: ${instance.Confidence}`)
      })
      console.log("Parents:")
      label.Parents.forEach(parent => {
        console.log(`  ${parent.Name}`)
      })
      console.log("------------")
      console.log("")
    }) // for response.labels
  } // if
});
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
   public static void detectImageLabels(RekognitionClient rekClient, String
 sourceImage) {

        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);

            // Create an Image object for the source image.
            Image souImage = Image.builder()
                .bytes(sourceBytes)
                .build();

            DetectLabelsRequest detectLabelsRequest = DetectLabelsRequest.builder()
                .image(souImage)
                .maxLabels(10)
                .build();

            DetectLabelsResponse labelsResponse =
 rekClient.detectLabels(detectLabelsRequest);
            List<Label> labels = labelsResponse.labels();
            System.out.println("Detected labels for the given photo");
            for (Label label: labels) {
                System.out.println(label.name() + ": " +
 label.confidence().toString());
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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 response 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, Vehicle, and Car. Each label has an associated level of confidence. For example, the detection algorithm is 98.991432% 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 Automobile has two parent labels named Vehicle and Transportation.

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`.


```
{

    {
    "Labels": [
        {
            "Name": "Vehicle",
            "Confidence": 99.15271759033203,
            "Instances": [],
            "Parents": [
                {
                    "Name": "Transportation"
                }
            ]
        },
        {
            "Name": "Transportation",
            "Confidence": 99.15271759033203,
            "Instances": [],
            "Parents": []
        },
        {
            "Name": "Automobile",
            "Confidence": 99.15271759033203,
            "Instances": [],
            "Parents": [
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ]
        },
        {
            "Name": "Car",
            "Confidence": 99.15271759033203,
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.10616336017847061,
                        "Height": 0.18528179824352264,
```

```
                "Left": 0.0037978808395564556,
                "Top": 0.5039216876029968
            },
            "Confidence": 99.15271759033203
        },
        {
            "BoundingBox": {
                "Width": 0.2429988533258438,
                "Height": 0.21577216684818268,
                "Left": 0.7309805154800415,
                "Top": 0.5251884460449219
            },
            "Confidence": 99.1286392211914
        },
        {
            "BoundingBox": {
                "Width": 0.14233611524105072,
                "Height": 0.15528248250484467,
                "Left": 0.6494812965393066,
                "Top": 0.5333095788955688
            },
            "Confidence": 98.48368072509766
        },
        {
            "BoundingBox": {
                "Width": 0.11086395382881165,
                "Height": 0.10271988064050674,
                "Left": 0.10355594009160995,
                "Top": 0.5354844927787781
            },
            "Confidence": 96.45606231689453
        },
        {
            "BoundingBox": {
                "Width": 0.06254628300666809,
                "Height": 0.053911514580249786,
                "Left": 0.46083059906959534,
                "Top": 0.5573825240135193
            },
            "Confidence": 93.65448760986328
        },
        {
            "BoundingBox": {
                "Width": 0.10105438530445099,
                "Height": 0.12226245552301407,
                "Left": 0.5743985772132874,
                "Top": 0.534368634223938
            },
            "Confidence": 93.06217193603516
        },
        {
            "BoundingBox": {
                "Width": 0.056389667093753815,
                "Height": 0.17163699865341187,
                "Left": 0.9427769780158997,
                "Top": 0.5235804319381714
            },
            "Confidence": 92.6864013671875
        },
        {
            "BoundingBox": {
                "Width": 0.06003860384225845,
                "Height": 0.06737709045410156,
                "Left": 0.22409997880458832,
                "Top": 0.5441341400146484
            },
```

```
                    "Confidence": 90.4227066040039
                },
                {
                    "BoundingBox": {
                        "Width": 0.02848697081208229,
                        "Height": 0.19150497019290924,
                        "Left": 0.0,
                        "Top": 0.5107086896896362
                    },
                    "Confidence": 86.65286254882812
                },
                {
                    "BoundingBox": {
                        "Width": 0.04067881405353546,
                        "Height": 0.03428703173995018,
                        "Left": 0.316415935754776,
                        "Top": 0.5566273927688599
                    },
                    "Confidence": 85.36471557617188
                },
                {
                    "BoundingBox": {
                        "Width": 0.043411049991846085,
                        "Height": 0.0893595889210701,
                        "Left": 0.18293385207653046,
                        "Top": 0.5394920110702515
                    },
                    "Confidence": 82.21705627441406
                },
                {
                    "BoundingBox": {
                        "Width": 0.031183116137981415,
                        "Height": 0.03989990055561066,
                        "Left": 0.2853088080883026,
                        "Top": 0.5579366683959961
                    },
                    "Confidence": 81.0157470703125
                },
                {
                    "BoundingBox": {
                        "Width": 0.031113790348172188,
                        "Height": 0.056484755128622055,
                        "Left": 0.2580395042896271,
                        "Top": 0.5504819750785828
                    },
                    "Confidence": 56.13441467285156
                },
                {
                    "BoundingBox": {
                        "Width": 0.08586374670267105,
                        "Height": 0.08550430089235306,
                        "Left": 0.5128012895584106,
                        "Top": 0.5438792705535889
                    },
                    "Confidence": 52.37760925292969
                }
            ],
            "Parents": [
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ]
        },
```

```
        {
            "Name": "Human",
            "Confidence": 98.9914321899414,
            "Instances": [],
            "Parents": []
        },
        {
            "Name": "Person",
            "Confidence": 98.9914321899414,
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.19360728561878204,
                        "Height": 0.2742200493812561,
                        "Left": 0.43734854459762573,
                        "Top": 0.35072067379951477
                    },
                    "Confidence": 98.9914321899414
                },
                {
                    "BoundingBox": {
                        "Width": 0.03801717236638069,
                        "Height": 0.06597328186035156,
                        "Left": 0.9155802130699158,
                        "Top": 0.5010883808135986
                    },
                    "Confidence": 85.02790832519531
                }
            ],
            "Parents": []
        }
    ],
    "LabelModelVersion": "2.0"
}


}
```


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. 85).

Amazon Rekognition Video label detection is an asynchronous operation. To start the detection of labels in a video, call StartLabelDetection. 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 to get the detected labels. For information about calling the video analysis API operations, see Calling Amazon Rekognition Video operations (p. 62).

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.

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) element.
- **Label information** – The `LabelDetection` array element contains a (Label) object which contains the label name and the confidence Amazon Rekognition has in the accuracy of the detected label. A `Label` object also includes a hierarchical taxonomy of labels and bounding box information for common labels. `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. 64).
- **Video information** – The response includes information about the video format (`VideoMetadata`) in each page of information returned by `GetLabelDetection`.


```
{
    "Labels": [
        {
            "Timestamp": 0,
            "Label": {
                "Instances": [],
                "Confidence": 60.51791763305664,
                "Parents": [],
                "Name": "Electronics"
            }
        },
        {
            "Timestamp": 0,
            "Label": {
                "Instances": [],
                "Confidence": 99.53411102294922,
                "Parents": [],
                "Name": "Human"
            }
        },
        {
            "Timestamp": 0,
            "Label": {
                "Instances": [
                    {
                        "BoundingBox": {
                            "Width": 0.11109819263219833,
                            "Top": 0.08098889887332916,
                            "Left": 0.8881205320358276,
                            "Height": 0.9073750972747803
                        },
                        "Confidence": 99.5831298828125
                    },
                    {
                        "BoundingBox": {
                            "Width": 0.1268676072359085,
                            "Top": 0.14018426835536957,
                            "Left": 0.0003282368124928324,
                            "Height": 0.7993982434272766
                        },
                        "Confidence": 99.46029663085938
                    }
                ],
                "Confidence": 99.53411102294922,
                "Parents": [],
                "Name": "Person"
```

```
            }
        },
        .
        .
        .

        {
            "Timestamp": 166,
            "Label": {
                "Instances": [],
                "Confidence": 73.6471176147461,
                "Parents": [
                    {
                        "Name": "Clothing"
                    }
                ],
                "Name": "Sleeve"
            }
        }

    ],
    "LabelModelVersion": "2.0",
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
        "Format": "QuickTime / MOV",
        "FrameRate": 23.976024627685547,
        "Codec": "h264",
        "DurationMillis": 5005,
        "FrameHeight": 674,
        "FrameWidth": 1280
    }
}
```


Detecting labels in streaming video events

You can use Amazon Rekognition Video to detect labels in streaming video. To do this, you create a stream processor (CreateStreamProcessor) to start and manage the analysis of streaming video.

Amazon Rekognition Video uses Amazon Kinesis Video Streams to receive and process a video stream. When you create the stream processor, you choose what you want the stream processor to detect. You can choose people, packages and pets, or people and packages. The analysis results are output to your Amazon S3 bucket and in Amazon SNS notifications. Note that Amazon Rekognition Video detects the presence of a person in the video, but does not detect whether the person is a specific individual. To search for a face from a collection in a streaming video, see the section called “Searching faces in a collection in streaming video” (p. 216).

To use Amazon Rekognition Video with streaming video, your application requires the following:

- A Kinesis video stream for sending streaming video to Amazon Rekognition Video. For more information, see the Amazon Kinesis Video Streams Developer Guide.
- An Amazon Rekognition Video stream processor to manage the analysis of the streaming video. For more information, see Overview of Amazon Rekognition Video stream processor operations (p. 90).
- An Amazon S3 bucket. Amazon Rekognition Video publishes session output to the S3 bucket. The output includes the image frame where a person or object of interest was detected for first time. You must be the owner of the S3 bucket.
- An Amazon SNS topic that Amazon Rekognition Video publishes smart alerts and an end-of-session summary to.

Topics

Setting up your Amazon Rekognition Video and Amazon Kinesis resources

The following procedures describe the steps that you take to provision the Kinesis video stream and other resources that are used to detect labels in a streaming video.

Prerequisites

To run this procedure, AWS SDK for Java must be installed. For more information, see Getting started with Amazon Rekognition (p. 11). The AWS account that you use requires access permissions to the Amazon Rekognition API. For more information, see Actions Defined by Amazon Rekognition in the *IAM User Guide*.

To detect labels in a video stream (AWS SDK)

1. Create an Amazon S3 bucket. Note the bucket name and any key prefixes that you want to use. You use this information later.
2. Create an Amazon SNS topic. You can use it to receive notifications when an object of interest is first detected in the video stream. Note the Amazon Resource Name (ARN) for the topic. For more information, see Creating an Amazon SNS topic in the Amazon SNS developer guide.
3. Subscribe an endpoint to the Amazon SNS topic. For more information, see Subscribing to an Amazon SNS topic in the Amazon SNS developer guide.
4. Create a Kinesis video stream and note the Amazon Resource Name (ARN) of the stream.
5. If you didn't already, create an IAM service role to give Amazon Rekognition Video access to your Kinesis video streams, your S3 bucket, and your Amazon SNS topic. For more information, see Giving access for label detection stream processors (p. 118).

You can then create the label detection stream processor (p. 121) and start the stream processor (p. 123) using the stream processor name that you chose.

> **Note**
> Start the stream processor only after you verified that you can ingest media into the Kinesis video stream.

Camera orientation and setup

Amazon Rekognition Video Streaming Video Events can support all cameras that are supported by Kinesis Video Streams. For best results, we recommend placing the camera between 0 to 45 degrees from the ground. The camera needs to be in its canonical upright position. For example, if there is a person in the frame, the person should be oriented vertically, and the head of the person should be higher in the frame than the feet.

Giving access for label detection stream processors

You use an AWS Identity and Access Management (IAM) service role to give Amazon Rekognition Video read access to Kinesis video streams. To do this, use IAM roles to give Amazon Rekognition Video access to your Amazon S3 bucket and to an Amazon SNS topic.

You can create a permissions policy that allows Amazon Rekognition Video access to an existing Amazon SNS topic, Amazon S3 bucket, and Kinesis video stream. For a step-by-step procedure using the AWS CLI, see the section called "AWS CLI commands to set up a label detection IAM role" (p. 120).

To give Amazon Rekognition Video access to resources for label detection

1. Create a new permissions policy with the IAM JSON policy editor, and use the following policy. Replace `kvs-stream-name` with the name of the Kinesis video stream, `topicarn` with the Amazon Resource Name (ARN) of the Amazon SNS topic that you want to use, and `bucket-name` with the name of the Amazon S3 bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "KinesisVideoPermissions",
            "Effect": "Allow",
            "Action": [
                "kinesisvideo:GetDataEndpoint",
                "kinesisvideo:GetMedia"
            ],
            "Resource": [
                "arn:aws:kinesisvideo:::stream/kvs-stream-name/*"
            ]
        },
        {
            "Sid": "SNSPermissions",
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:::sns-topic-name"
            ]
        },
        {
            "Sid": "S3Permissions",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name/*"
            ]
        }
    ]
}
```

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 create the stream processor before you perform video analysis operations.

4. (Optional) If you use your own AWS KMS key to encrypt data sent to your S3 bucket, you must add the following statement with the IAM role. (This is the IAM role that you created for the key policy, which corresponds to the customer managed key that you want to use.)

```
            {
                        "Sid": "Allow use of the key by label detection Role",
                        "Effect": "Allow",
                        "Principal": {
                            "AWS":
 "arn:aws:iam:::role/REPLACE_WITH_LABEL_DETECTION_ROLE_CREATED"
                        },
                        "Action": [
                            "kms:Decrypt",
                            "kms:GenerateDataKey*"
                        ],
                        "Resource": "*"
            }
```

AWS CLI commands to set up a label detection IAM role

If you didn't already, set up and configure the AWS CLI with your credentials.

Enter the following commands into the AWS CLI to set up an IAM role with the necessary permissions for label detection.

1. `export IAM_ROLE_NAME=labels-test-role`
2. `export AWS_REGION=us-east-1`
3. Create a trust relationship policy file (for example, assume-role-rekognition.json) with the following content.

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

4. `aws iam create-role --role-name $IAM_ROLE_NAME --assume-role-policy-document file://path-to-assume-role-rekognition.json --region $AWS_REGION`
5. `aws iam attach-role-policy --role-name $IAM_ROLE_NAME --policy-arn "arn:aws:iam::aws:policy/service-role/AmazonRekognitionServiceRole" --region $AWS_REGION`
6. If the name of your SNS topic that you want to receive notifications with doesn't start with the "AmazonRekognition" prefix, add the following policy:

   ```
   aws iam attach-role-policy --role-name $IAM_ROLE_NAME --policy-arn
   "arn:aws:iam::aws:policy/AmazonSNSFullAccess" --region $AWS_REGION
   ```
7. If you use your own AWS KMS key to encrypt data sent to your Amazon S3 bucket, update the key policy of the customer managed key that you want to use.
   a. Create a file kms_key_policy.json that contains the following content:

```
{
"Sid": "Allow use of the key by label detection Role",
"Effect": "Allow",
"Principal": {
"AWS": "arn:aws:iam:::role/REPLACE_WITH_IAM_ROLE_NAME_CREATED"
},
"Action": [
"kms:Encrypt",
"kms:GenerateDataKey*"
],
"Resource": "*"
}
```

b. `export KMS_KEY_ID=labels-kms-key-id`. Replace KMS_KEY_ID with the KMS key ID that you created.

c. `aws kms put-key-policy --policy-name default --key-id $KMS_KEY_ID --policy file://path-to-kms-key-policy.json`

Label detection operations for streaming video events

Amazon Rekognition Video can detect people or relevant objects in a streaming video and notify you when they're detected. When you create a label detection stream processor, choose what labels that you want Amazon Rekognition Video to detect. These can be people, packages and pets, or people, packages, and pets. Choose only the specific labels that you want to detect. That way, the only relevant labels create notifications. You can configure options to determine when to store video information, and then do additional processing based on the labels that are detected in the frame.

After you set up your resources, the process to detect labels in a streaming video is as follows:

1. Create the stream processor
2. Start the stream processor
3. If an object of interest is detected, you receive an Amazon SNS notification for the first occurrence of each object of interest.
4. The stream processor stops when the time specified in `MaxDurationInSeconds` is complete.
5. You receive a final Amazon SNS notification with an event summary.
6. Amazon Rekognition Video publishes a detailed session summary to your S3 bucket.

Topics

Creating the Amazon Rekognition Video label detection stream processor

Before you can analyze a streaming video, you create an Amazon Rekognition Video stream processor (CreateStreamProcessor).

If you want to create a stream processor to detect labels of interest and people, provide as input a Kinesis video stream (`Input`), Amazon S3 bucket information (`Output`), and an Amazon SNS topic ARN (`StreamProcessorNotificationChannel`). You can also provide a KMS key ID to encrypt the data sent to your S3 bucket. You specify what you want to detect in `Settings`, such as people, packages and people, or pets, people, and packages. You can also specify where in the frame that you want Amazon Rekognition to monitor with `RegionsOfInterest`. The following is a JSON example for the `CreateStreamProcessor` request.

```
{
  "DataSharingPreference": { "OptIn":TRUE
  },
  "Input": {
    "KinesisVideoStream": {
      "Arn": "arn:aws:kinesisvideo:us-east-1:nnnnnnnnnnnn:stream/muh_video_stream/
nnnnnnnnnnnnn"
    }
  },
  "KmsKeyId": "muhkey",
  "Name": "muh-default_stream_processor",
  "Output": {
    "S3Destination": {
      "Bucket": "s3bucket",
      "KeyPrefix": "s3prefix"
    }
  },
  "NotificationChannel": {
    "SNSTopicArn": "arn:aws:sns:us-east-2:nnnnnnnnnnnn:MyTopic"
  },
  "RoleArn": "arn:aws:iam::nnnnnnnnn:role/Admin",
  "Settings": {
    "ConnectedHome": {
      "Labels": [
        "PET"
      ]
    "MinConfidence": 80
    }
  },
  "RegionsOfInterest": [
    {
      "BoundingBox": {
        "Top": 0.11,
        "Left": 0.22,
        "Width": 0.33,
        "Height": 0.44
      }
    },
    {
      "Polygon": [
        {
          "X": 0.11,
          "Y": 0.11
        },
        {
          "X": 0.22,
          "Y": 0.22
        },
        {
          "X": 0.33,
          "Y": 0.33
        }
      ]
```

```
    }
  ]
}
```

Note that you can change the `MinConfidence` value when you specify the `ConnectedHomeSettings` for the stream processor. `MinConfidence` is a numerical value ranging from 0 to 100 that indicates how certain the algorithm is about its predictions. For example, a notification for `person` with a confidence value of 90 means that the algorithm is absolutely certain that the person is present in the video. A confidence value of 10 indicates that there might be a person. You can set `MinConfidence` to a desired value of your choice between 0 and 100 depending on how frequently you want to be notified. For example, if you want to be notified only when Rekognition is absolutely certain there is a package in the video frame then you can set `MinConfidence` to 90.

By default, `MinConfidence` is set to 50. If you want to optimize the algorithm for higher precision, then you can set `MinConfidence` to be higher than 50. You then receive fewer notification, but each notification is more reliable. If you want to optimize the algorithm for higher recall, then you can set `MinConfidence` to be lower than 50 to receive more notifications.

Starting the Amazon Rekognition Video label detection stream processor

You start analyzing streaming video by calling StartStreamProcessor with the stream processor name that you specified in `CreateStreamProcessor`. When you run the `StartStreamProcessor` operation on a label detection stream processor, you input start and stop information to determine the processing time.

When you start the stream processor, the label detection stream processor state changes in the following ways:

1. When you call `StartStreamProcessor`, the label detection stream processor state goes from `STOPPED` or `FAILED` to `STARTING`.
2. While the label detection stream processor runs, it stays in `STARTING`.
3. When the label detection stream processor is done running, the state becomes either `STOPPED` or `FAILED`.

The `StartSelector` specifies the starting point in the Kinesis stream to start processing. You can use the KVS Producer timestamp or the KVS Fragment number. For more information, see Fragment.

> **Note**
> If you use the KVS Producer timestamp, you must input the time in milliseconds.

The `StopSelector` specifies when to stop processing the stream. You can specify a maximum amount of time to process the video. The default is a maximum duration of 10 seconds. Note that the actual processing time might be a bit longer than the maximum duration, depending on the size of individual KVS fragments. If the maximum duration has been reached or exceeded at the end of a fragment, the processing time stops.

The following is a JSON example for the `StartStreamProcessor` request.

```
{
   "Name": "string",
   "StartSelector": {
     "KVSStreamStartSelector": {
         "KVSProducerTimestamp": 1655930623123
```

```
        },
          "StopSelector": {
              "MaxDurationInSeconds": 11
        }
    }
}
```

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

Analyzing label detection results

There are three ways that Amazon Rekognition Video publishes notifications from a label detection stream processor: Amazon SNS notifications for object detection events, an Amazon SNS notification for an end-of-session summary, and a detailed Amazon S3 bucket report.

- Amazon SNS notifications for object detection events.

  If labels are detected in the video stream, you receive Amazon SNS notifications for object detection events. Amazon Rekognition publishes a notification the first time that a person or an object of interest is detected in the video stream. Notifications include information such as the type of label detected, the confidence, and a link to the hero image. They also include a cropped image of the person or object detected and a detection timestamp. The notification has the following format:

```
{"Subject": "Rekognition Stream Processing Event",
    "Message": {
        "inputInformation": {
            "kinesisVideo": {
                "streamArn": string
            }
        },
        "eventNamespace": {
            "type": "LABEL_DETECTED"
        },
        "labels": [{
            "id": string,
            "name": "PERSON" | "PET" | "PACKAGE",
            "frameImageUri": string,
            "croppedImageUri": string,
            "videoMapping": {
                "kinesisVideoMapping": {
                    "fragmentNumber": string,
                    "serverTimestamp": number,
                    "producerTimestamp": number,
                    "frameOffsetMillis": number
                }
            },
            "boundingBox": {
                "left": number,
                "top": number,
                "height": number,
                "width": number
            }
        }],
        "eventId": string,
        "tags": {
            [string]: string
        },
        "sessionId": string,
        "startStreamProcessorRequest": object
```

```
    }
}
```

- Amazon SNS end-of-session summary.

  You also receive an Amazon SNS notification when the stream processing session is finished. This notification lists the metadata for the session. This includes details such as the duration of the stream that was processed. The notification has the following format:

```
{"Subject": "Rekognition Stream Processing Event",
    "Message": {
        "inputInformation": {
            "kinesisVideo": {
                "streamArn": string,
                "processedVideoDurationMillis": number
            }
        },
        "eventNamespace": {
            "type": "STREAM_PROCESSING_COMPLETE"
        },
        "streamProcessingResults": {
            "message": string
        },
        "eventId": string,
        "tags": {
            [string]: string
        },
        "sessionId": string,
        "startStreamProcessorRequest": object
    }
}
```

- Amazon S3 bucket report.

  Amazon Rekognition Video publishes detailed inference results of a video analysis operation to the Amazon S3 bucket that's provided in the `CreateStreamProcessor` operation. These results include image frames where an object of interest or person was detected for first time.

  The frames are available in S3 in the following path: ObjectKeyPrefix/StreamProcessorName/SessionId/*`service_determined_unique_path`*. In this path, **LabelKeyPrefix** is a customer provided optional argument, **StreamProcessorName** is the name of the stream processor resource, and **SessionId** is a unique ID for the stream processing session. Replace these according to your situation.

Detecting custom labels

Amazon Rekognition Custom Labels can identify the objects and scenes in images that are specific to your business needs, such as logos or engineering machine parts. For more information, see What Is Amazon Rekognition Custom Labels? in the *Amazon Rekognition Custom Labels Developer Guide*.

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.

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. 26) and Working with stored video analysis (p. 60).

> **Note**
> The face detection models used by Amazon Rekognition Image and Amazon Rekognition Video don't support the detection of faces in cartoon/animated characters or non-human entities. If you want to detect cartoon characters in images or videos, we recommend using Amazon Rekognition Custom Labels. For more information, see the Amazon Rekognition Custom Labels Developer Guide.

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. 157).

Topics

Overview of face detection and face comparison

There are two primary applications of machine learning that analyze images containing faces: face detection and face comparison. A face detection system is designed to answer the question: is there a face in this picture? A face detection system determines the presence, location, scale, and (possibly) orientation of any face present in a still image or video frame. This system is designed to detect the presence of faces regardless of attributes such as gender, age, and facial hair.

A face comparison system is designed to answer the question: does the face in an image match the face in another image? A face comparison system takes an image of a face and makes a prediction about whether the face matches other faces in a provided database. Face comparison systems are designed to compare and predict potential matches of faces regardless of their expression, facial hair, and age.

Both face detection and face comparison systems can provide an estimate of the confidence level of the prediction in the form of a probability or confidence score. For example, a face detection system may predict that an image region is a face at a confidence score of 90%, and another image region is a face at a confidence score of 60%. The region with the higher confidence score should be more likely to contain a face. If a face detection system does not properly detect a face, or provides a low confidence prediction of an actual face, this is known as a missed detection or false negative. If a facial detection system incorrectly predicts the presence of a face at a high confidence level, this is a false alarm or false positive. Similarly, a facial comparison system may not match two faces belonging to the same person (missed detection/false negative), or may incorrectly predict that two faces from different people are the same person (false alarm/false positive).

Confidence scores are a critical component of face detection and comparison systems. These systems make predictions of whether a face exists in an image or matches a face in another image, with a corresponding level of confidence in the prediction. Users of these systems should consider the confidence score/similarity threshold provided by the system when designing their application and making decisions based on the output of the system. For example, in a photo application used to identify similar looking family members, if the confidence threshold is set at 80%, then the application will return matches when predictions reach an 80% confidence level, but will not return matches below that level. This threshold may be acceptable because the risk of missed detections or false alarms is low for this type of use case. However, for use cases where the risk of missed detection or false alarm is higher, the system should use a higher confidence level. You should use a 99% confidence/similarity threshold in scenarios where highly accurate facial matches are important. For more information on recommended confidence thresholds, see Searching faces in a collection (p. 157).

Guidelines on face attributes

Amazon Rekognition returns a bounding box, landmarks, quality, and the pose for each face it detects. Amazon Rekognition also returns predictions for emotion, gender, age, and other attributes for each face if the parameter for attributes is set to `ALL` in the API request. Each attribute or emotion has a value and a confidence score. For example, a certain face might be predicted as having the gender ‘Female’ with a confidence score of 85% and the emotion ‘Happy’ with a confidence score of 90%.

A gender binary (male/female) prediction is based on the physical appearance of a face in a particular image. It doesn't indicate a person’s gender identity, and you shouldn't use Amazon Rekognition to make such a determination. We don't recommend using gender binary predictions to make decisions that impact
an individual's rights, privacy, or access to services.

Similarly, a prediction of an emotional expression is based on the physical appearance of a person's face in an image. It doesn't indicate a person’s actual internal emotional state, and you shouldn't use Amazon Rekognition to make such a determination. For example, a person pretending to have a happy face in a picture might look happy, but might not be experiencing happiness.

We recommend using a threshold of 99% or more for use cases where the accuracy of classification could have any negative impact on the subjects of the images. The only exception is Age Range, where Amazon Rekognition estimates the lower and upper age for the person. In this case, the wider the age range, the lower the confidence for that prediction. As an approximation, you should use the mid-point of the age range to estimate a single value for the age of the detected face. (The actual age does not necessarily correspond to this number.)

One of the best uses of these attributes is generating aggregate statistics. For example, attributes, such as Smile, Pose, and Sharpness, may be used to select the 'best profile picture' automatically in a social media application. Another common use case is estimating demographics anonymously of a broad sample using the predicted gender and age attributes (for example, at events or retail stores).

For more information about attributes, see FaceDetail.

Detecting faces in an image

Amazon Rekognition Image provides the DetectFaces operation that looks for key facial features such as eyes, nose, and mouth to detect faces in an input image. 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. 13).

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 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();
      }

   }

}
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void detectFacesinImage(RekognitionClient rekClient,String
 sourceImage ) {

        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);

            // Create an Image object for the source image.
            Image souImage = Image.builder()
                .bytes(sourceBytes)
                .build();

            DetectFacesRequest facesRequest = DetectFacesRequest.builder()
                .attributes(Attribute.ALL)
                .image(souImage)
                .build();

            DetectFacesResponse facesResponse =
 rekClient.detectFaces(facesRequest);
            List<FaceDetail> faceDetails = facesResponse.faceDetails();
            for (FaceDetail face : faceDetails) {
                AgeRange ageRange = face.ageRange();
                System.out.println("The detected face is estimated to be between "
                            + ageRange.low().toString() + " and " +
 ageRange.high().toString()
                            + " years old.");

                System.out.println("There is a smile :
 "+face.smile().value().toString());
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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 and other attributes 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

def detect_faces(photo, 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))

  # Access predictions for individual face details and print them
        print("Gender: " + str(faceDetail['Gender']))
        print("Smile: " + str(faceDetail['Smile']))
        print("Eyeglasses: " + str(faceDetail['Eyeglasses']))
        print("Emotions: " + str(faceDetail['Emotions'][0]))

    return len(response['FaceDetails'])
def main():
    photo='photo'
    bucket='bucket'
    face_count=detect_faces(photo, bucket)
    print("Faces detected: " + str(face_count))


if __name__ == "__main__":
    main()
```

.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);
        }
    }
}
```

Ruby

This example displays the estimated age range for detected faces, and lists various 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.)

   # Add to your Gemfile
   # gem 'aws-sdk-rekognition'
   require 'aws-sdk-rekognition'
   credentials = Aws::Credentials.new(
      ENV['AWS_ACCESS_KEY_ID'],
      ENV['AWS_SECRET_ACCESS_KEY']
   )
   bucket = 'bucket' # the bucketname without s3://
   photo  = 'input.jpg'# the name of file
   client   = Aws::Rekognition::Client.new credentials: credentials
   attrs = {
     image: {
       s3_object: {
         bucket: bucket,
         name: photo
```

```
      },
    },
    attributes: ['ALL']
  }
  response = client.detect_faces attrs
  puts "Detected faces for: #{photo}"
  response.face_details.each do |face_detail|
    low  = face_detail.age_range.low
    high = face_detail.age_range.high
    puts "The detected face is between: #{low} and #{high} years old"
    puts "All other attributes:"
    puts "  bounding_box.width:     #{face_detail.bounding_box.width}"
    puts "  bounding_box.height:    #{face_detail.bounding_box.height}"
    puts "  bounding_box.left:      #{face_detail.bounding_box.left}"
    puts "  bounding_box.top:       #{face_detail.bounding_box.top}"
    puts "  age.range.low:          #{face_detail.age_range.low}"
    puts "  age.range.high:         #{face_detail.age_range.high}"
    puts "  smile.value:            #{face_detail.smile.value}"
    puts "  smile.confidence:       #{face_detail.smile.confidence}"
    puts "  eyeglasses.value:       #{face_detail.eyeglasses.value}"
    puts "  eyeglasses.confidence:  #{face_detail.eyeglasses.confidence}"
    puts "  sunglasses.value:       #{face_detail.sunglasses.value}"
    puts "  sunglasses.confidence:  #{face_detail.sunglasses.confidence}"
    puts "  gender.value:           #{face_detail.gender.value}"
    puts "  gender.confidence:      #{face_detail.gender.confidence}"
    puts "  beard.value:            #{face_detail.beard.value}"
    puts "  beard.confidence:       #{face_detail.beard.confidence}"
    puts "  mustache.value:         #{face_detail.mustache.value}"
    puts "  mustache.confidence:    #{face_detail.mustache.confidence}"
    puts "  eyes_open.value:        #{face_detail.eyes_open.value}"
    puts "  eyes_open.confidence:   #{face_detail.eyes_open.confidence}"
    puts "  mout_open.value:        #{face_detail.mouth_open.value}"
    puts "  mout_open.confidence:   #{face_detail.mouth_open.confidence}"
    puts "  emotions[0].type:       #{face_detail.emotions[0].type}"
    puts "  emotions[0].confidence: #{face_detail.emotions[0].confidence}"
    puts "  landmarks[0].type:      #{face_detail.landmarks[0].type}"
    puts "  landmarks[0].x:         #{face_detail.landmarks[0].x}"
    puts "  landmarks[0].y:         #{face_detail.landmarks[0].y}"
    puts "  pose.roll:              #{face_detail.pose.roll}"
    puts "  pose.yaw:               #{face_detail.pose.yaw}"
    puts "  pose.pitch:             #{face_detail.pose.pitch}"
    puts "  quality.brightness:     #{face_detail.quality.brightness}"
    puts "  quality.sharpness:      #{face_detail.quality.sharpness}"
    puts "  confidence:             #{face_detail.confidence}"
    puts "------------"
    puts ""
  end
```

Node.js

This example displays the estimated age range for detected faces, and lists various 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.

If you are using TypeScript definitions, you may need to use `import AWS from 'aws-sdk'` instead of `const AWS = require('aws-sdk')`, in order to run the program with Node.js. You can consult the AWS SDK for Javascript for more details. Depending on how you have your configurations set up, you also may need to specify your region with `AWS.config.update({region: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.)
```

```
const AWS = require('aws-sdk')
const bucket = 'bucket' // the bucketname without s3://
const photo  = 'input.jpg' // the name of file
const config = new AWS.Config({
  accessKeyId: process.env.AWS_ACCESS_KEY_ID,
  secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
  region: process.env.AWS_REGION
})
const client = new AWS.Rekognition();
const params = {
  Image: {
    S3Object: {
      Bucket: bucket,
      Name: photo
    },
  },
  Attributes: ['ALL']
}
client.detectFaces(params, function(err, response) {
  if (err) {
    console.log(err, err.stack); // an error occurred
  } else {
    console.log(`Detected faces for: ${photo}`)
    response.FaceDetails.forEach(data => {
      let low  = data.AgeRange.Low
      let high = data.AgeRange.High
      console.log(`The detected face is between: ${low} and ${high} years old`)
      console.log("All other attributes:")
      console.log(`  BoundingBox.Width:      ${data.BoundingBox.Width}`)
      console.log(`  BoundingBox.Height:     ${data.BoundingBox.Height}`)
      console.log(`  BoundingBox.Left:       ${data.BoundingBox.Left}`)
      console.log(`  BoundingBox.Top:        ${data.BoundingBox.Top}`)
      console.log(`  Age.Range.Low:          ${data.AgeRange.Low}`)
      console.log(`  Age.Range.High:         ${data.AgeRange.High}`)
      console.log(`  Smile.Value:            ${data.Smile.Value}`)
      console.log(`  Smile.Confidence:       ${data.Smile.Confidence}`)
      console.log(`  Eyeglasses.Value:       ${data.Eyeglasses.Value}`)
      console.log(`  Eyeglasses.Confidence:  ${data.Eyeglasses.Confidence}`)
      console.log(`  Sunglasses.Value:       ${data.Sunglasses.Value}`)
      console.log(`  Sunglasses.Confidence:  ${data.Sunglasses.Confidence}`)
      console.log(`  Gender.Value:           ${data.Gender.Value}`)
      console.log(`  Gender.Confidence:      ${data.Gender.Confidence}`)
      console.log(`  Beard.Value:            ${data.Beard.Value}`)
      console.log(`  Beard.Confidence:       ${data.Beard.Confidence}`)
      console.log(`  Mustache.Value:         ${data.Mustache.Value}`)
      console.log(`  Mustache.Confidence:    ${data.Mustache.Confidence}`)
      console.log(`  EyesOpen.Value:         ${data.EyesOpen.Value}`)
      console.log(`  EyesOpen.Confidence:    ${data.EyesOpen.Confidence}`)
      console.log(`  MouthOpen.Value:        ${data.MouthOpen.Value}`)
      console.log(`  MouthOpen.Confidence:   ${data.MouthOpen.Confidence}`)
      console.log(`  Emotions[0].Type:       ${data.Emotions[0].Type}`)
      console.log(`  Emotions[0].Confidence: ${data.Emotions[0].Confidence}`)
      console.log(`  Landmarks[0].Type:      ${data.Landmarks[0].Type}`)
      console.log(`  Landmarks[0].X:         ${data.Landmarks[0].X}`)
      console.log(`  Landmarks[0].Y:         ${data.Landmarks[0].Y}`)
      console.log(`  Pose.Roll:              ${data.Pose.Roll}`)
      console.log(`  Pose.Yaw:               ${data.Pose.Yaw}`)
      console.log(`  Pose.Pitch:             ${data.Pose.Pitch}`)
      console.log(`  Quality.Brightness:     ${data.Quality.Brightness}`)
      console.log(`  Quality.Sharpness:      ${data.Quality.Sharpness}`)
      console.log(`  Confidence:             ${data.Confidence}`)
      console.log("------------")
      console.log("")
    }) // for response.faceDetails
  } // if
```

```
});
```

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. 26).


```
{
    "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.
- **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, such as 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 comparison input images (p. 100).
- **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": 43,
                "Low": 26
            },
            "Beard": {
                "Confidence": 97.48941802978516,
                "Value": true
            },
            "BoundingBox": {
                "Height": 0.6968063116073608,
                "Left": 0.26937249302864075,
                "Top": 0.11424895375967026,
```

```
            "Width": 0.42325547337532043
        },
        "Confidence": 99.99995422363281,
        "Emotions": [
            {
                "Confidence": 0.042965151369571686,
                "Type": "DISGUSTED"
            },
            {
                "Confidence": 0.002022328320890665,
                "Type": "HAPPY"
            },
            {
                "Confidence": 0.4482877850532532,
                "Type": "SURPRISED"
            },
            {
                "Confidence": 0.007082826923578978,
                "Type": "ANGRY"
            },
            {
                "Confidence": 0,
                "Type": "CONFUSED"
            },
            {
                "Confidence": 99.47616577148438,
                "Type": "CALM"
            },
            {
                "Confidence": 0.017732391133904457,
                "Type": "SAD"
            }
        ],
        "Eyeglasses": {
            "Confidence": 99.42405700683594,
            "Value": false
        },
        "EyesOpen": {
            "Confidence": 99.99604797363281,
            "Value": true
        },
        "Gender": {
            "Confidence": 99.722412109375,
            "Value": "Male"
        },
        "Landmarks": [
            {
                "Type": "eyeLeft",
                "X": 0.38549351692199707,
                "Y": 0.3959200084209442
            },
            {
                "Type": "eyeRight",
                "X": 0.5773905515670776,
                "Y": 0.394561767578125
            },
            {
                "Type": "mouthLeft",
                "X": 0.40410104393959045,
                "Y": 0.6479480862617493
            },
            {
                "Type": "mouthRight",
                "X": 0.5623446702957153,
                "Y": 0.647117555141449
            },
```

```
                {
                    "Type": "nose",
                    "X": 0.47763553261756897,
                    "Y": 0.5337067246437073
                },
                {
                    "Type": "leftEyeBrowLeft",
                    "X": 0.3114689588546753,
                    "Y": 0.3376390337944031
                },
                {
                    "Type": "leftEyeBrowRight",
                    "X": 0.4224424660205841,
                    "Y": 0.3232649564743042
                },
                {
                    "Type": "leftEyeBrowUp",
                    "X": 0.36654090881347656,
                    "Y": 0.3104579746723175
                },
                {
                    "Type": "rightEyeBrowLeft",
                    "X": 0.5353175401687622,
                    "Y": 0.3223199248313904
                },
                {
                    "Type": "rightEyeBrowRight",
                    "X": 0.6546239852905273,
                    "Y": 0.3348073363304138
                },
                {
                    "Type": "rightEyeBrowUp",
                    "X": 0.5936762094497681,
                    "Y": 0.3080498278141022
                },
                {
                    "Type": "leftEyeLeft",
                    "X": 0.3524211347103119,
                    "Y": 0.3936865031719208
                },
                {
                    "Type": "leftEyeRight",
                    "X": 0.4229775369167328,
                    "Y": 0.3973258435726166
                },
                {
                    "Type": "leftEyeUp",
                    "X": 0.38467878103256226,
                    "Y": 0.3836822807788849
                },
                {
                    "Type": "leftEyeDown",
                    "X": 0.38629674911499023,
                    "Y": 0.40618783235549927
                },
                {
                    "Type": "rightEyeLeft",
                    "X": 0.5374732613563538,
                    "Y": 0.39637991786003113
                },
                {
                    "Type": "rightEyeRight",
                    "X": 0.609208345413208,
                    "Y": 0.391626238822937
                },
                {
```

```
                "Type": "rightEyeUp",
                "X": 0.5750962495803833,
                "Y": 0.3821527063846588
            },
            {
                "Type": "rightEyeDown",
                "X": 0.5740782618522644,
                "Y": 0.40471214056015015
            },
            {
                "Type": "noseLeft",
                "X": 0.4441811740398407,
                "Y": 0.5608476400375366
            },
            {
                "Type": "noseRight",
                "X": 0.5155643820762634,
                "Y": 0.5569332242012024
            },
            {
                "Type": "mouthUp",
                "X": 0.47968366742134094,
                "Y": 0.6176465749740601
            },
            {
                "Type": "mouthDown",
                "X": 0.4807897210121155,
                "Y": 0.690782368183136
            },
            {
                "Type": "leftPupil",
                "X": 0.38549351692199707,
                "Y": 0.3959200084209442
            },
            {
                "Type": "rightPupil",
                "X": 0.5773905515670776,
                "Y": 0.394561767578125
            },
            {
                "Type": "upperJawlineLeft",
                "X": 0.27245330810546875,
                "Y": 0.3902156949043274
            },
            {
                "Type": "midJawlineLeft",
                "X": 0.31561678647994995,
                "Y": 0.6596118807792664
            },
            {
                "Type": "chinBottom",
                "X": 0.48385748267173767,
                "Y": 0.8160444498062134
            },
            {
                "Type": "midJawlineRight",
                "X": 0.6625112891197205,
                "Y": 0.656606137752533
            },
            {
                "Type": "upperJawlineRight",
                "X": 0.7042999863624573,
                "Y": 0.3863988518714905
            }
        ],
        "MouthOpen": {
```

```
                "Confidence": 99.83820343017578,
                "Value": false
            },
            "Mustache": {
                "Confidence": 72.20288848876953,
                "Value": false
            },
            "Pose": {
                "Pitch": -4.970901966094971,
                "Roll": -1.4911699295043945,
                "Yaw": -10.983647346496582
            },
            "Quality": {
                "Brightness": 73.81391906738281,
                "Sharpness": 86.86019134521484
            },
            "Smile": {
                "Confidence": 99.93638610839844,
                "Value": false
            },
            "Sunglasses": {
                "Confidence": 99.81478881835938,
                "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 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`.

Comparing faces in images

To compare a face in the *source* image with each face in the *target* image, use the CompareFaces 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.

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.

> **Note**
> CompareFaces uses machine learning algorithms, which are probabilistic. A false negative is an incorrect prediction that a face in the target image has a low similarity confidence score

when compared to the face in the source image. To reduce the probability of false negatives, we recommend that you compare the target image against multiple source images. If you plan to use `CompareFaces` to make a decision that impacts an individual's rights, privacy, or access to services, we recommend that you pass the result to a human for review and further validation before taking action.

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. 13).

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 " + match.getSimilarity().toString()
                + "% confidence.");

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

        System.out.println("There was " + uncompared.size()
             + " face(s) that did not match");
    }
}
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
   public static void compareTwoFaces(RekognitionClient rekClient, Float
 similarityThreshold, String sourceImage, String targetImage) {
        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            InputStream tarStream = new FileInputStream(targetImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            SdkBytes targetBytes = SdkBytes.fromInputStream(tarStream);

            // Create an Image object for the source image.
            Image souImage = Image.builder()
                .bytes(sourceBytes)
```

```
                .build();

            Image tarImage = Image.builder()
                .bytes(targetBytes)
                .build();

            CompareFacesRequest facesRequest = CompareFacesRequest.builder()
                .sourceImage(souImage)
                .targetImage(tarImage)
                .similarityThreshold(similarityThreshold)
                .build();

            // Compare the two images.
            CompareFacesResponse compareFacesResult =
 rekClient.compareFaces(facesRequest);
            List<CompareFacesMatch> faceDetails = compareFacesResult.faceMatches();
            for (CompareFacesMatch match: faceDetails){
                ComparedFace face= match.face();
                BoundingBox position = face.boundingBox();
                System.out.println("Face at " + position.left().toString()
                        + " " + position.top()
                        + " matches with " + face.confidence().toString()
                        + "% confidence.");

            }
            List<ComparedFace> uncompared = compareFacesResult.unmatchedFaces();
            System.out.println("There was " + uncompared.size() + " face(s) that
 did not match");
            System.out.println("Source image rotation: " +
 compareFacesResult.sourceImageOrientationCorrection());
            System.out.println("target image rotation: " +
 compareFacesResult.targetImageOrientationCorrection());

        } catch(RekognitionException | FileNotFoundException e) {
            System.out.println("Failed to load source image " + sourceImage);
            System.exit(1);
        }
    }
```

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 `source_file` and `target_file` 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

def compare_faces(sourceFile, targetFile):

    client=boto3.client('rekognition')

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

    response=client.compare_faces(SimilarityThreshold=80,
                                  SourceImage={'Bytes': imageSource.read()},
                                  TargetImage={'Bytes': imageTarget.read()})

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

    imageSource.close()
    imageTarget.close()
    return len(response['FaceMatches'])

def main():
    source_file='source'
    target_file='target'
    face_matches=compare_faces(source_file, target_file)
    print("Face matches: " + str(face_matches))


if __name__ == "__main__":
    main()
```

.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 " + match.Similarity
                  + "% confidence.");
        }

        Console.WriteLine("There was " + compareFacesResponse.UnmatchedFaces.Count
 + " face(s) that did not match");

    }
}
```

Ruby

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

Replace the values of `photo_source` and `photo_target` with the path and file name of the source and target images.

```
 # Add to your Gemfile
  # gem 'aws-sdk-rekognition'
  require 'aws-sdk-rekognition'
  credentials = Aws::Credentials.new(
     ENV['AWS_ACCESS_KEY_ID'],
     ENV['AWS_SECRET_ACCESS_KEY']
  )
  bucket        = 'bucket' # the bucketname without s3://
  photo_source  = 'source.jpg'
  photo_target  = 'target.jpg'
  client   = Aws::Rekognition::Client.new credentials: credentials
  attrs = {
    source_image: {
      s3_object: {
        bucket: bucket,
        name: photo_source
      },
    },
    target_image: {
      s3_object: {
        bucket: bucket,
        name: photo_target
      },
    },
    similarity_threshold: 70
  }
  response = client.compare_faces attrs
  response.face_matches.each do |face_match|
    position   = face_match.face.bounding_box
    similarity = face_match.similarity
    puts "The face at: #{position.left}, #{position.top} matches with
#{similarity} % confidence"
  end
```

Node.js

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

Replace the values of `photo_source` and `photo_target` 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.)


   const AWS = require('aws-sdk')
   const bucket        = 'bucket' // the bucketname without s3://
   const photo_source  = 'source.jpg'
   const photo_target  = 'target.jpg'
   const config = new AWS.Config({
     accessKeyId: process.env.AWS_ACCESS_KEY_ID,
     secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
```

```
    region: process.env.AWS_REGION
  })
  const client = new AWS.Rekognition();
  const params = {
    SourceImage: {
      S3Object: {
        Bucket: bucket,
        Name: photo_source
      },
    },
    TargetImage: {
      S3Object: {
        Bucket: bucket,
        Name: photo_target
      },
    },
    SimilarityThreshold: 70
  }
  client.compareFaces(params, function(err, response) {
    if (err) {
      console.log(err, err.stack); // an error occurred
    } else {
      response.FaceMatches.forEach(data => {
        let position   = data.Face.BoundingBox
        let similarity = data.Similarity
        console.log(`The face at: ${position.Left}, ${position.Top} matches with
${similarity} % confidence`)
      }) // for response.faceDetails
    } // if
  });
```

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. 26).

```
{
    "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.
- **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": 0.4891010989665985 1,
            "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.
- Additional attributes as explained on the the section called "Guidelines on face attributes" (p. 127) page.

Amazon Rekognition Video face detection in stored videos is an asynchronous operation. To start the detection of faces in videos, call StartFaceDetection. 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 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. 62).

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 StartFaceDetection(String bucket, String video) throws
 Exception{

    NotificationChannel channel= new NotificationChannel()
            .withSNSTopicArn(snsTopicArn)
            .withRoleArn(roleArn);

    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 GetFaceDetectionResults() 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);


}
```

In the function `main`, replace the lines:

```
        StartLabelDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetLabelDetectionResults();
```

with:

```
        StartFaceDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetFaceDetectionResults();
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void StartFaceDetection(RekognitionClient rekClient,
                                          NotificationChannel channel,
                                          String bucket,
                                          String video) {

        try {
```

```
            S3Object s3Obj = S3Object.builder()
                .bucket(bucket)
                .name(video)
                .build();

            Video vidOb = Video.builder()
                .s3Object(s3Obj)
                .build();

            StartFaceDetectionRequest  faceDetectionRequest =
StartFaceDetectionRequest.builder()
                .jobTag("Faces")
                .faceAttributes(FaceAttributes.ALL)
                .notificationChannel(channel)
                .video(vidOb)
                .build();

            StartFaceDetectionResponse startLabelDetectionResult =
rekClient.startFaceDetection(faceDetectionRequest);
            startJobId=startLabelDetectionResult.jobId();

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void GetFaceResults(RekognitionClient rekClient) {

        try {
            String paginationToken=null;
            GetFaceDetectionResponse faceDetectionResponse=null;
            boolean finished = false;
            String status;
            int yy=0 ;

            do{
                if (faceDetectionResponse !=null)
                    paginationToken = faceDetectionResponse.nextToken();

                GetFaceDetectionRequest recognitionRequest =
GetFaceDetectionRequest.builder()
                    .jobId(startJobId)
                    .nextToken(paginationToken)
                    .maxResults(10)
                    .build();

                // Wait until the job succeeds
                while (!finished) {

                    faceDetectionResponse =
rekClient.getFaceDetection(recognitionRequest);
                    status = faceDetectionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is null
```

```
                VideoMetadata videoMetaData=faceDetectionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " + videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                // Show face information
                List<FaceDetection> faces= faceDetectionResponse.faces();

                for (FaceDetection face: faces) {
                    String age = face.face().ageRange().toString();
                    String smile = face.face().smile().toString();
                    System.out.println("The detected face is estimated to be"
                                + age + " years old.");
                    System.out.println("There is a smile : "+smile);
                }

            } while (faceDetectionResponse !=null &&
 faceDetectionResponse.nextToken() != null);

        } catch(RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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.)

    # ============= Faces===============
    def StartFaceDetection(self):
        response=self.rek.start_face_detection(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
            NotificationChannel={'RoleArn': self.roleArn, 'SNSTopicArn':
 self.snsTopicArn})

        self.startJobId=response['JobId']
        print('Start Job Id: ' + self.startJobId)

    def GetFaceDetectionResults(self):
        maxResults = 10
        paginationToken = ''
        finished = False

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

            print('Codec: ' + response['VideoMetadata']['Codec'])
            print('Duration: ' + str(response['VideoMetadata']['DurationMillis']))
            print('Format: ' + response['VideoMetadata']['Format'])
            print('Frame rate: ' + str(response['VideoMetadata']['FrameRate']))
            print()

            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
```

In the function `main`, replace the lines:

```
    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()
```

with:

```
    analyzer.StartFaceDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetFaceDetectionResults()
```

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, 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) 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. 64).
- **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 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) 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. To search for faces in a stored video, call StartFaceSearch. To search for faces in a streaming video, call CreateStreamProcessor.

> **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. 10).

You can perform the following management operations on a collection.

- Create a collection with CreateCollection. For more information, see Creating a collection (p. 161).
- List the available collections with ListCollections. For more information, see Listing collections (p. 167).
- Describe a collection with DescribeCollection. For more information, see Describing a collection (p. 171).
- Delete a collection with DeleteCollection. For more information, see Deleting a collection (p. 176).

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 IndexFaces 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. 179).

- The ListFaces operation lists the faces in a collection. For more information, see Adding faces to a collection (p. 179).
- The DeleteFaces operation deletes faces from a collection. For more information, see Deleting faces from a collection (p. 195).

Guidance for using IndexFaces

The following is guidance for using `IndexFaces` in common scenarios.

Critical or public safety applications

- Call IndexFaces with images which contain only one face in each image and associate the returned Face ID with the identifier for the subject of the image.
- You can use DetectFaces ahead of indexing to verify there is only one face in the image. If more than one face is detected, re-submit the image after review and with only one face present. This prevents inadvertently indexing multiple faces and associating them with the same person.

Photo sharing and social media applications

- You should call `IndexFaces` without restrictions on images that contain multiple faces in use cases such as family albums. In such cases, you need to identify each person in every photo and use that information to group photos by the people present in them.

General usage

- Index multiple different images of the same person, particularly with different face attributes (facial poses, facial hair, etc) to improve matching quality.
- Include a review process so that failed matches can be indexed with the correct face identifier to improve subsequent face matching ability.
- For information about image quality, see Recommendations for facial comparison input images (p. 100).

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 (SearchFaces). For more information, see Searching for a face using its face ID (p. 199).
- The largest face in a supplied image (SearchFacesByImage). For more information, see Searching for a face using an image (p. 203).
- Faces in a stored video. For more information, see Searching stored videos for faces (p. 208).
- Faces in a streaming video. For more information, see Working with streaming video events (p. 90).

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. 139).
- `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, SearchFaces, and SearchFacesByImage) 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.

Use cases that involve public safety

In addition to the recommendations listed in Best practices for sensors, input images, and videos (p. 100) and Guidance for using IndexFaces (p. 158), you should use the following best practices when deploying face detection and comparison systems in use cases that involve public safety. First, you should use confidence thresholds of 99% or higher to reduce errors and false positives. Second, you should involve human reviewers to verify results received from a face detection or comparison system, and you should not make decisions based on system output without additional human review. Face detection and comparison systems should serve as a tool to help narrow the field and allow humans to expeditiously review and consider options. Third, we recommend that you should be transparent about the use of face detection and comparison systems in these use cases, including, wherever possible, informing end users and subjects about the use of these systems, obtaining consent for such use, and providing a mechanism where end users and subjects can provide feedback to improve the system.

If you are a law enforcement agency that is using the Amazon Rekognition face comparison feature in connection with criminal investigations, you must follow the requirements listed in the AWS Service Terms. This includes the following.

- Have appropriately trained humans review all decisions to take action that might impact a person's civil liberties or equivalent human rights.
- Train personnel on responsible use of facial recognition systems.
- Provide public disclosures of your use of facial recognition systems.

- Don't use Amazon Rekognition for sustained surveillance of a person without independent review or exigent circumstances.

In all cases, facial comparison matches should be viewed in the context of other compelling evidence, and shouldn't be used as the sole determinant for taking action. However, if facial comparison is used for non-law-enforcement scenarios (for example, for unlocking a phone or authenticating an employee's identity to access a secure, private office building), these decisions wouldn't require a manual audit because they wouldn't impact a person's civil liberties or equivalent human rights.

If you're planning to use a face detection or face comparison system for use cases that involve public safety you should employ the best practices mentioned previously. In addition, you should consult published resources on the use of face comparison. This includes the Face Recognition Policy Development Template For Use In Criminal Intelligence and Investigative Activities provided by the Bureau of Justice Assistance of the Department of Justice. The template provides several facial comparison and biometric-related resources and is designed to provide law enforcement and public safety agencies with a framework for developing face comparison policies that comply with applicable laws, reduce privacy risks, and establish entity accountability and oversight. Additional resources include Best Privacy Practices for Commercial Use of Facial Recognition by the National Telecommunications and Information Administration and Best Practices for Common Uses of Facial Recognition by the staff of the Federal Trade Commission. Other resources may be developed and published in the future, and you should continuously educate yourself on this important topic.

As a reminder, you must comply with all applicable laws in their use of AWS services, and you may not use any AWS service in a manner that violates the rights of others or may be harmful to others. This means that you may not use AWS services for public safety use cases in a way that illegally discriminates against a person or violates a person's due process, privacy, or civil liberties. You should obtain appropriate legal advice as necessary to review any legal requirements or questions regarding your use case.

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 finding possible matches. 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. If you are a law enforcement agency using Amazon Rekognition to assist in identifying a person in connection with a criminal investigation, and actions will be taken based on the identification that could impact that person's civil liberties or equivalent human rights, the decision to take action must be made by an appropriately trained person based on their independent examination of the identification evidence.
- 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.
- When civil rights are a concern, such as use cases involving law enforcement, use confidence thresholds of 99% or higher and employ human review of facial comparison predictions to ensure that a person's civil rights aren't violated.
- 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 operation. For more information, see Guidance for using IndexFaces (p. 158).

In other use cases (such as social media), we recommend you use your best judgement to assess if the Amazon Rekognition results need human review. Also, depending on your application's requirements, the similarity threshold can be lower.

Creating a collection

You can use the CreateCollection operation to create a collection.

For more information, see Managing collections (p. 157).

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. 13).

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());
```

```
    }

}
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void createMyCollection(RekognitionClient rekClient,String
 collectionId ) {

        try {
            CreateCollectionRequest collectionRequest =
 CreateCollectionRequest.builder()
                .collectionId(collectionId)
                .build();

            CreateCollectionResponse collectionResponse =
 rekClient.createCollection(collectionRequest);
            System.out.println("CollectionArn: " +
 collectionResponse.collectionArn());
            System.out.println("Status code: " +
 collectionResponse.statusCode().toString());

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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 `collection_id` 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

def create_collection(collection_id):

    client=boto3.client('rekognition')

    #Create a collection
    print('Creating collection:' + collection_id)
    response=client.create_collection(CollectionId=collection_id)
```

```
    print('Collection ARN: ' + response['CollectionArn'])
    print('Status code: ' + str(response['StatusCode']))
    print('Done...')

def main():
    collection_id='Collection'
    create_collection(collection_id)

if __name__ == "__main__":
    main()
```

.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);

    }
}
```

Node.JS

In the following example, replace the value of `region` with the name of the region associated with your account and replace the value of `collectionName` with the desired name of your collection.

```
import { CreateCollectionCommand} from  "@aws-sdk/client-rekognition";
import  { RekognitionClient } from "@aws-sdk/client-rekognition";

// Set the AWS Region.
const REGION = "region"; //e.g. "us-east-1"
const rekogClient = new RekognitionClient({ region: REGION });

// Name the collecction
```

```
const collection_name = "collectionName"

const createCollection = async (collectionName) => {
    try {
       console.log(`Creating collection: ${collectionName}`)
       const data = await rekogClient.send(new
 CreateCollectionCommand({CollectionId: collectionName}));
       console.log("Collection ARN:")
       console.log(data.CollectionARN)
       console.log("Status Code:")
       console.log(String(data.StatusCode))
       console.log("Success.",  data);
       return data;
    } catch (err) {
      console.log("Error", err.stack);
    }
  };

  createCollection(collection_name)
```

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
}
```

Tagging collections

You can identify, organize, search for, and filter Amazon Rekognition collections by using tags. Each tag is a label consisting of a user-defined key and value.

You can also use tags to control access for a collection by using Identity and Access Management (IAM). For more information, see Controlling access to AWS resources using resource tags.

Topics

Add tags to a new collection

You can add tags to a collection as you create it using the `CreateCollection` operation. Specify one or more tags in the `Tags` array input parameter.

AWS CLI

```
aws rekognition create-collection --collection-id "collection name"\
                --tags '{"key1":"value1","key2":"value2"}'
```

Python

```
import boto3

def create_collection(collection_id):
    client = boto3.client('rekognition')

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

def main():
    collection_id = 'NewCollectionName'
    create_collection(collection_id)

if __name__ == "__main__":
    main()
```

Add tags to an existing collection

To add one or more tags to an existing collection, use the `TagResource` operation. Specify the collection's Amazon Resource Name (ARN) (`ResourceArn`) and the tags (`Tags`) that you want to add. The following example shows how to add two tags.

AWS CLI

```
aws rekognition tag-resource --resource-arn resource-arn \
                --tags '{"key1":"value1","key2":"value2"}'
```

Python

```
import boto3

def create_tag():
    client = boto3.client('rekognition')
    response = client.tag_resource(ResourceArn="arn:aws:rekognition:region-
name:5498347593847598:collection/NewCollectionName",
                                   Tags={
                                       "KeyName": "ValueName"
                                   })
    print(response)
    if "'HTTPStatusCode': 200" in str(response):
        print("Success!!")
```

```
def main():
    create_tag()

if __name__ == "__main__":
    main()
```

Note
If you do not know the collection's Amazon Resource Name, you can use the `DescribeCollection` operation.

List tags in a collection

To list the tags attached to a collection, use the `ListTagsForResource` operation and specify the ARN of the collection (`ResourceArn`). The response is a map of tag keys and values that are attached to the specified collection.

AWS CLI

```
aws rekognition list-tags-for-resource --resource-arn resource-arn
```

Python

```
import boto3

def list_tags():
    client = boto3.client('rekognition')
    response = client.list_tags_for_resource(ResourceArn="arn:aws:rekognition:region-
name:5498347593847598:collection/NewCollectionName")
    print(response)

def main():
    list_tags()

if __name__ == "__main__":
    main()
```

The output displays a list of tags attached to the collection:

```

            {
    "Tags": {
        "Dept": "Engineering",
        "Name": "Ana Silva Carolina",
        "Role": "Developer"
    }
}
```

Delete tags from a collection

To remove one or more tags from a collection, use the `UntagResource` operation. Specify the ARN of the model (`ResourceArn`) and the tag keys (`Tag-Keys`) that you want to remove.

AWS CLI

```
aws rekognition untag-resource --resource-arn resource-arn \
```

```
                --tag-keys '["key1","key2"]'
```

Alternatively, you can specify tag-keys in this format:

```
--tag-keys key1,key2
```

Python

```
import boto3

def list_tags():
    client = boto3.client('rekognition')
    response = client.untag_resource(ResourceArn="arn:aws:rekognition:region-
name:5498347593847598:collection/NewCollectionName", TagKeys=['KeyName'])
    print(response)

def main():
    list_tags()

if __name__ == "__main__":
    main()
```

Listing collections

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

For more information, see Managing collections (p. 157).

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. 13).

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 {
```

```
     AmazonRekognition 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);

   }
}
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
   public static void listAllCollections(RekognitionClient rekClient) {
       try {
           ListCollectionsRequest listCollectionsRequest =
 ListCollectionsRequest.builder()
               .maxResults(10)
               .build();

           ListCollectionsResponse response =
 rekClient.listCollections(listCollectionsRequest);
           List<String> collectionIds = response.collectionIds();
           for (String resultId : collectionIds) {
               System.out.println(resultId);
           }

       } catch (RekognitionException e) {
           System.out.println(e.getMessage());
           System.exit(1);
       }
   }
```

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

def list_collections():

    max_results=2

    client=boto3.client('rekognition')

    #Display all the collections
    print('Displaying collections...')
    response=client.list_collections(MaxResults=max_results)
    collection_count=0
    done=False

    while done==False:
        collections=response['CollectionIds']

        for collection in collections:
            print (collection)
            collection_count+=1
        if 'NextToken' in response:
            nextToken=response['NextToken']

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

        else:
            done=True

    return collection_count

def main():

    collection_count=list_collections()
    print("collections: " + str(collection_count))
if __name__ == "__main__":
    main()
```

.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);
    }
}
```

Node.js

```
import { ListCollectionsCommand } from  "@aws-sdk/client-rekognition";
import  { RekognitionClient } from "@aws-sdk/client-rekognition";

// Set the AWS Region.
const REGION = "region"; //e.g. "us-east-1"
const rekogClient = new RekognitionClient({ region: REGION });


const listCollection = async () => {
    var max_results = 3
    console.log("Displaying collections:")
    var response = await rekogClient.send(new ListCollectionsCommand({MaxResults:
 max_results}))
    var collection_count = 0
    var done = false
    while (done == false){
        var collections = response.CollectionIds
        collections.forEach(collection => {
            console.log(collection)
            collection_count += 1
        });
        if (JSON.stringify(response).includes("NextToken")){
            nextToken = response.NextToken
            response = new ListCollectionsCommand({NextToken:nextToken, MaxResults:
 max_results})
        }
        else{
            done=true
        }
        return collection_count
    }
}

var collect_list = await listCollection()
console.log(collect_list)
```

ListCollections operation request

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

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

If the response has more collections 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 `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. 10).

`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 operation to get the following information about a collection:

- The number of faces that are indexed into the collection.
- The version of the model being used with the collection. For more information, see the section called "Model versioning" (p. 10).
- 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. 13).

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());

   }

}
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
   public static void describeColl(RekognitionClient rekClient, String
 collectionName) {

        try {
            DescribeCollectionRequest describeCollectionRequest =
 DescribeCollectionRequest.builder()
                .collectionId(collectionName)
                .build();

            DescribeCollectionResponse describeCollectionResponse =
 rekClient.describeCollection(describeCollectionRequest);
            System.out.println("Collection Arn : " +
 describeCollectionResponse.collectionARN());
            System.out.println("Created : " +
 describeCollectionResponse.creationTimestamp().toString());

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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 `collection_id` 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

def describe_collection(collection_id):

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

    try:
        response=client.describe_collection(CollectionId=collection_id)
        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 ' + collection_id + ' was not found ')
        else:
            print ('Error other than Not Found occurred: ' + e.response['Error']
['Message'])
    print('Done...')


def main():
    collection_id='MyCollection'
    describe_collection(collection_id)

if __name__ == "__main__":
    main()
```

.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);
    }
}
```

Node.js

```
import { DescribeCollectionCommand } from  "@aws-sdk/client-rekognition";
import  { RekognitionClient } from "@aws-sdk/client-rekognition";
```

```
import { stringify } from "querystring";

// Set the AWS Region.
const REGION = "region"; //e.g. "us-east-1"
const rekogClient = new RekognitionClient({ region: REGION });

// Name the collection
const collection_name = "collectionName"
const resourceArn = "resourceArn"

const describeCollection = async (collectionName) => {
    try {
       console.log(`Attempting to describe collection named - ${collectionName}`)
       var response = await rekogClient.send(new
 DescribeCollectionCommand({CollectionId: collectionName}))
       console.log('Collection Arn:')
       console.log(response.CollectionARN)
       console.log('Face Count:')
       console.log(response.FaceCount)
       console.log('Face Model Version:')
       console.log(response.FaceModelVersion)
       console.log('Timestamp:')
       console.log(response.CreationTimestamp)
       return response; // For unit tests.
    } catch (err) {
      console.log("Error", err.stack);
    }
  };

describeCollection(collection_name)
```

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 being used with the collection, `FaceModelVersion`. For more information, see the section called “Model versioning” (p. 10).
- 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 operation to delete a collection.

For more information, see Managing collections (p. 157).

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. 13).

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());

   }

}
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
   public static void deleteMyCollection(RekognitionClient rekClient,String
 collectionId ) {

        try {
            DeleteCollectionRequest deleteCollectionRequest =
 DeleteCollectionRequest.builder()
                .collectionId(collectionId)
                .build();

            DeleteCollectionResponse deleteCollectionResponse =
 rekClient.deleteCollection(deleteCollectionRequest);
            System.out.println(collectionId + ": " +
 deleteCollectionResponse.statusCode().toString());

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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 `collection_id` 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

def delete_collection(collection_id):


    print('Attempting to delete collection ' + collection_id)
    client=boto3.client('rekognition')
    status_code=0
    try:
        response=client.delete_collection(CollectionId=collection_id)
        status_code=response['StatusCode']
```

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


def main():
    collection_id='UnitTestCollection'
    status_code=delete_collection(collection_id)
    print('Status code: ' + str(status_code))

if __name__ == "__main__":
    main()
```

.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);
    }
}
```

Node.js

```
import { DeleteCollectionCommand } from  "@aws-sdk/client-rekognition";
import  { RekognitionClient } from "@aws-sdk/client-rekognition";

// Set the AWS Region.
const REGION = "region"; //e.g. "us-east-1"
const rekogClient = new RekognitionClient({ region: REGION });
```

```
// Name the collection
const collection_name = "collectionName"

const deleteCollection = async (collectionName) => {
    try {
       console.log(`Attempting to delete collection named - ${collectionName}`)
       var response = await rekogClient.send(new
 DeleteCollectionCommand({CollectionId: collectionName}))
       var status_code = response.StatusCode
       if (status_code = 200){
           console.log("Collection successfully deleted.")
       }
       return response; // For unit tests.
    } catch (err) {
      console.log("Error", err.stack);
    }
  };

deleteCollection(collection_name)
```

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 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 comparison input images (p. 100).

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:

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

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. 10).

Information about indexed faces is returned in an array of FaceRecord 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` chooses a quality bar that's used to filter out faces. You can use the `QualityFilter` input parameter to explicitly set the quality bar. The values are:

- `AUTO` — Amazon Rekognition chooses the quality bar that's used to filter out faces (default value).
- `LOW` — All except the lowest quality faces are indexed.
- `MEDIUM`
- `HIGH` — Only the highest quality faces are indexed.
- `NONE` - No faces are filtered out based on quality.

`IndexFaces` filters 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.
- The face doesn’t have enough detail to be suitable for face search.

Note
To use quality filtering, you need a collection that's associated with version 3, or higher, of the face model. To get the version of the face model associated with a collection, call DescribeCollection.

Information about faces that aren't indexed by `IndexFaces` is returned in an array of UnindexedFace 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. 157).

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. 13).
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 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.QualityFilter;
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);
            }
        }
    }
}
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void addToCollection(RekognitionClient rekClient, String
 collectionId, String sourceImage) {

        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            Image souImage = Image.builder()
                .bytes(sourceBytes)
                .build();

            IndexFacesRequest facesRequest = IndexFacesRequest.builder()
                .collectionId(collectionId)
                .image(souImage)
                .maxFaces(1)
                .qualityFilter(QualityFilter.AUTO)
                .detectionAttributes(Attribute.DEFAULT)
                .build();
```

```
            IndexFacesResponse facesResponse = rekClient.indexFaces(facesRequest);
            System.out.println("Results for the image");
            System.out.println("\n Faces indexed:");
            List<FaceRecord> faceRecords = facesResponse.faceRecords();
            for (FaceRecord faceRecord : faceRecords) {
                System.out.println("  Face ID: " + faceRecord.face().faceId());
                System.out.println("  Location:" +
 faceRecord.faceDetail().boundingBox().toString());
            }

            List<UnindexedFace> unindexedFaces = facesResponse.unindexedFaces();
            System.out.println("Faces not indexed:");
            for (UnindexedFace unindexedFace : unindexedFaces) {
                System.out.println("  Location:" +
 unindexedFace.faceDetail().boundingBox().toString());
                System.out.println("  Reasons:");
                for (Reason reason : unindexedFace.reasons()) {
                    System.out.println("Reason:  " + reason);
                }
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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

def add_faces_to_collection(bucket,photo,collection_id):
```

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

    response=client.index_faces(CollectionId=collection_id,
                                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)
    return len(response['FaceRecords'])

def main():
    bucket='bucket'
    collection_id='collection'
    photo='photo'


    indexed_faces_count=add_faces_to_collection(bucket, photo, collection_id)
    print("Faces indexed count: " + str(indexed_faces_count))


if __name__ == "__main__":
    main()
```

.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 UnindexedFace.


```
{
    "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": "",
    "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. 10).

`OrientationCorrection` is the estimated orientation of the image. Orientation correction information is not returned if you are using a version of the face detection model that is greater than version 3. For more information, see Getting image orientation and bounding box coordinates (p. 53).


```
{
    "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 operation to list the faces in a collection.

For more information, see Managing faces in a collection (p. 157).

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. 13).

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);
   }

}
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void listFacesCollection(RekognitionClient rekClient, String
 collectionId ) {
        try {
            ListFacesRequest facesRequest = ListFacesRequest.builder()
                .collectionId(collectionId)
                .maxResults(10)
                .build();

            ListFacesResponse facesResponse = rekClient.listFaces(facesRequest);
            List<Face> faces = facesResponse.faces();
            for (Face face: faces) {
                System.out.println("Confidence level there is a face:
 "+face.confidence());
                System.out.println("The face Id value is "+face.faceId());
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
         }
```

```
        }
```

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

def list_faces_in_collection(collection_id):


    maxResults=2
    faces_count=0
    tokens=True

    client=boto3.client('rekognition')
    response=client.list_faces(CollectionId=collection_id,
                               MaxResults=maxResults)

    print('Faces in collection ' + collection_id)


    while tokens:

        faces=response['Faces']

        for face in faces:
            print (face)
            faces_count+=1
        if 'NextToken' in response:
            nextToken=response['NextToken']
            response=client.list_faces(CollectionId=collection_id,
                                       NextToken=nextToken,MaxResults=maxResults)
        else:
            tokens=False
    return faces_count
def main():

    collection_id='collection'

    faces_count=list_faces_in_collection(collection_id)
    print("faces count: " + str(faces_count))
if __name__ == "__main__":
    main()
```

.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. 10).
- **Faces** – Information about the faces in the collection. This includes information about BoundingBox, confidence, image identifiers, and the face ID. For more information, see Face.
- **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",
            "IndexFacesModelVersion": "5.0"
        }
    ],
    "NextToken": "sm+5ythT3aeEVIR4WA...."
}
```


Deleting faces from a collection

You can use the DeleteFaces operation to delete faces from a collection. For more information, see Managing faces in a collection (p. 157).

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. 13).

2. Use the following examples to call the `DeleteFaces` 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);
      }
   }
}
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void deleteFacesCollection(RekognitionClient rekClient,
                                             String collectionId,
                                             String faceId) {

        try {
            DeleteFacesRequest deleteFacesRequest = DeleteFacesRequest.builder()
                .collectionId(collectionId)
                .faceIds(faceId)
                .build();

            rekClient.deleteFaces(deleteFacesRequest);
            System.out.println("The face was deleted from the collection.");
```

```
        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

AWS CLI

This AWS CLI command displays the JSON output for the `delete-faces` 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

def delete_faces_from_collection(collection_id, faces):

    client=boto3.client('rekognition')

    response=client.delete_faces(CollectionId=collection_id,
                               FaceIds=faces)

    print(str(len(response['DeletedFaces'])) + ' faces deleted:')
    for faceId in response['DeletedFaces']:
         print (faceId)
    return len(response['DeletedFaces'])

def main():

    collection_id='collection'
    faces=[]
    faces.append("xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")

    faces_count=delete_faces_from_collection(collection_id, faces)
    print("deleted faces count: " + str(faces_count))

if __name__ == "__main__":
    main()
```

.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 operation to search for faces in a collection that match a supplied face ID.

The face ID is returned in the IndexFaces operation response when the face is detected and added to a collection. For more information, see Managing faces in a collection (p. 157).

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. 13).

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.

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
   public static void searchFacebyId(RekognitionClient rekClient,String
 collectionId, String faceId) {

        try {
            SearchFacesRequest searchFacesRequest = SearchFacesRequest.builder()
                .collectionId(collectionId)
                .faceId(faceId)
                .faceMatchThreshold(70F)
                .maxFaces(2)
                .build();

            SearchFacesResponse imageResponse =
 rekClient.searchFaces(searchFacesRequest) ;
            System.out.println("Faces matching in the collection");
            List<FaceMatch> faceImageMatches = imageResponse.faceMatches();
            for (FaceMatch face: faceImageMatches) {
                System.out.println("The similarity level is  "+face.similarity());
                System.out.println();
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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

def search_face_in_collection(face_id,collection_id):
    threshold = 90
    max_faces=2
    client=boto3.client('rekognition')


    response=client.search_faces(CollectionId=collection_id,
                                FaceId=face_id,
                                FaceMatchThreshold=threshold,
                                MaxFaces=max_faces)


    face_matches=response['FaceMatches']
    print ('Matching faces')
    for match in face_matches:
            print ('FaceId:' + match['Face']['FaceId'])
            print ('Similarity: ' + "{:.2f}".format(match['Similarity']) + "%")
            print
    return len(face_matches)

def main():

    face_id='xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
    collection_id='MyCollection'

    faces=[]
    faces.append(face_id)

    faces_count=search_face_in_collection(face_id, collection_id)
    print("faces found: " + str(faces_count))

if __name__ == "__main__":
    main()
```

.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 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. 158).

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. 13).

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 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();
      }
   }
}
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
   public static void searchFaceInCollection(RekognitionClient rekClient,String
 collectionId, String sourceImage) {

        try {
            InputStream sourceStream = new FileInputStream(new File(sourceImage));
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            Image souImage = Image.builder()
                .bytes(sourceBytes)
                .build();

            SearchFacesByImageRequest facesByImageRequest =
 SearchFacesByImageRequest.builder()
                .image(souImage)
                .maxFaces(10)
                .faceMatchThreshold(70F)
                .collectionId(collectionId)
                .build();

            SearchFacesByImageResponse imageResponse =
 rekClient.searchFacesByImage(facesByImageRequest) ;
            System.out.println("Faces matching in the collection");
            List<FaceMatch> faceImageMatches = imageResponse.faceMatches();
            for (FaceMatch face: faceImageMatches) {
                System.out.println("The similarity level is  "+face.similarity());
                System.out.println();
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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 video events (p. 90).

The faces that you search for must first be indexed into a collection by using IndexFaces. For more information, see Adding faces to a collection (p. 179).

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 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 to get the search results. For more information about starting video analysis and getting the results, see Calling Amazon Rekognition Video operations (p. 62).

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. 161).
2. Index a face into the collection (p. 179).
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,
 String collection) throws Exception{

       NotificationChannel channel= new NotificationChannel()
               .withSNSTopicArn(snsTopicArn)
               .withRoleArn(roleArn);

       StartFaceSearchRequest req = new StartFaceSearchRequest()
               .withCollectionId(collection)
               .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 GetFaceSearchCollectionResults() 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);

  }
```

In the function `main`, replace the lines:

```
        StartLabelDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetLabelDetectionResults();
```

with:

```
        String collection="collection";
        StartFaceSearchCollection(bucket, video, collection);

        if (GetSQSMessageSuccess()==true)
         GetFaceSearchCollectionResults();
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
   public static void StartFaceDetection(RekognitionClient rekClient,
                                         NotificationChannel channel,
                                         String bucket,
                                         String video) {

       try {
           S3Object s3Obj = S3Object.builder()
               .bucket(bucket)
               .name(video)
               .build();
```

```
            Video vidOb = Video.builder()
                .s3Object(s3Obj)
                .build();

            StartFaceDetectionRequest  faceDetectionRequest =
StartFaceDetectionRequest.builder()
                .jobTag("Faces")
                .faceAttributes(FaceAttributes.ALL)
                .notificationChannel(channel)
                .video(vidOb)
                .build();

            StartFaceDetectionResponse startLabelDetectionResult =
rekClient.startFaceDetection(faceDetectionRequest);
            startJobId=startLabelDetectionResult.jobId();

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void GetFaceResults(RekognitionClient rekClient) {

        try {
            String paginationToken=null;
            GetFaceDetectionResponse faceDetectionResponse=null;
            boolean finished = false;
            String status;
            int yy=0 ;

            do{
                if (faceDetectionResponse !=null)
                    paginationToken = faceDetectionResponse.nextToken();

                GetFaceDetectionRequest recognitionRequest =
GetFaceDetectionRequest.builder()
                    .jobId(startJobId)
                    .nextToken(paginationToken)
                    .maxResults(10)
                    .build();

                // Wait until the job succeeds
                while (!finished) {

                    faceDetectionResponse =
rekClient.getFaceDetection(recognitionRequest);
                    status = faceDetectionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is null
                VideoMetadata videoMetaData=faceDetectionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " + videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
```

```
                System.out.println("Job");

                // Show face information
                List<FaceDetection> faces= faceDetectionResponse.faces();

                for (FaceDetection face: faces) {
                    String age = face.face().ageRange().toString();
                    String smile = face.face().smile().toString();
                    System.out.println("The detected face is estimated to be"
                                + age + " years old.");
                    System.out.println("There is a smile : "+smile);
                }

            } while (faceDetectionResponse !=null &&
 faceDetectionResponse.nextToken() != null);

        } catch(RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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.)

    # ============= Face Search ===============
    def StartFaceSearchCollection(self,collection):
        response = self.rek.start_face_search(Video={'S3Object':
{'Bucket':self.bucket,'Name':self.video}},
            CollectionId=collection,
            NotificationChannel={'RoleArn':self.roleArn,
 'SNSTopicArn':self.snsTopicArn})

        self.startJobId=response['JobId']

        print('Start Job Id: ' + self.startJobId)


    def GetFaceSearchCollectionResults(self):
        maxResults = 10
        paginationToken = ''

        finished = False

        while finished == False:
            response = self.rek.get_face_search(JobId=self.startJobId,
                                        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()
```

In the function `main`, replace the lines:

```
    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()
```

with:

```
    collection='tests'
    analyzer.StartFaceSearchCollection(collection)

    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetFaceSearchCollectionResults()
```

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 code to replace might be different.

5. Change the value of `collection` 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, exists for each time the person is matched in the video. Each `PersonMatch` includes an array of face matches from the input collection, FaceMatch, information about the matched person, PersonDetail, 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
    }
}
```


Searching faces in a collection in streaming video

You can use Amazon Rekognition Video to detect and recognize faces from a collection in streaming video. With Amazon Rekognition Video you can create a stream processor (CreateStreamProcessor) to start and manage the analysis of streaming video.

To detect a known face in a video stream (face search), 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.

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 the Amazon Kinesis Video Streams Developer Guide.
- An Amazon Rekognition Video stream processor to manage the analysis of the streaming video. For more information, see Overview of Amazon Rekognition Video stream processor operations (p. 90).
- A Kinesis data stream consumer to read the analysis results that Amazon Rekognition Video sends to the Kinesis data stream. For more information, see Kinesis Data Streams Consumers.

This section contains information about writing an application that creates the Kinesis video stream and other necessary resources, streams video into Amazon Rekognition Video, and receives the analysis results.

Topics

Setting up your Amazon Rekognition Video and Amazon Kinesis resources

The following procedures describe the steps you take to provision the Kinesis video stream and other resources that are used 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 Actions Defined by Amazon Rekognition in the *IAM User Guide*.

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 streams using AmazonRekognitionServiceRole (p. 219).
2. Create a collection (p. 161) and note the collection identifier you used.
3. Index the faces (p. 179) 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.

You can then create the face search stream processor (p. 220) and start the stream processor (p. 220) using the stream processor name that you chose.

> **Note**
> You should start the stream processor only after you have verified you can ingest media into the Kinesis video stream.

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.

- If you are streaming from an Matroska (MKV) encoded source, use the PutMedia operation to stream the source video into the Kinesis video stream that you created. For more information, see PutMedia API Example.
- If you are streaming from a device camera, see Streaming using a GStreamer plugin (p. 233).

Giving Amazon Rekognition Video access to your resources

You use an AWS Identity and Access Management (IAM) service role to give Amazon Rekognition Video read access to Kinesis video streams. If you are using a face search stream processor, you use an IAM service role to give Amazon Rekognition Video write access to Kinesis data streams. If you are using a security monitoring stream processor, you use IAM roles to give Amazon Rekognition Video access to your Amazon S3 bucket and to an Amazon SNS topic.

Giving access for face search stream processors

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 for a face search stream processor

1. Create a new permissions policy with the IAM JSON policy editor, and use the following policy. Replace `video-arn` with the ARN of the desired Kinesis video stream. If you are using a face search stream processor, replace `data-arn` with the ARN of the desired Kinesis data 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.

Giving access to streams using AmazonRekognitionServiceRole

As an alternative option for setting up access to Kinesis video streams and data streams, you can use the `AmazonRekognitionServiceRole` permissions policy. 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:

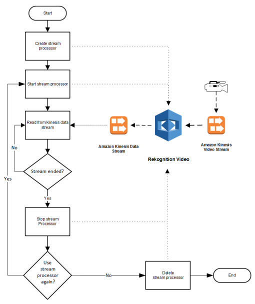
   1. Choose **Rekognition** for the service name.
   2. Choose **Rekognition** for the service role use case.
   3. Choose the **AmazonRekognitionServiceRole** permissions policy, which gives Amazon Rekognition Video write access to Kinesis data streams that are prefixed with *AmazonRekognition* and read access to all your Kinesis video streams.
2. To ensure your account is secure, limit the scope of Rekognition's access to just the resources you are using. This can be done by attaching a trust policy to your IAM service role. For information on how to do this, see Cross-service confused deputy prevention (p. 505).
3. Note the Amazon Resource Name (ARN) of the service role. You need it to start video analysis operations.

Searching 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. 157).

Topics

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



Creating the Amazon Rekognition Video face search stream processor

Before you can analyze a streaming video, you create an Amazon Rekognition Video stream processor (CreateStreamProcessor). 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 face search stream processor

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

```
{
```

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

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

Using stream processors for face searching (Java V2 example)

The following example code shows how to call various stream processor operations, such as CreateStreamProcessor and StartStreamProcessor, using the AWS SDK for Java version 2.

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void listStreamProcessors(RekognitionClient rekClient) {

        ListStreamProcessorsRequest request = ListStreamProcessorsRequest.builder()
            .maxResults(15)
            .build();

        ListStreamProcessorsResponse listStreamProcessorsResult =
 rekClient.listStreamProcessors(request);
        for (StreamProcessor streamProcessor :
 listStreamProcessorsResult.streamProcessors()) {
            System.out.println("StreamProcessor name - " + streamProcessor.name());
            System.out.println("Status - " + streamProcessor.status());
        }
    }

    private static void describeStreamProcessor(RekognitionClient rekClient, String
 StreamProcessorName) {

        DescribeStreamProcessorRequest streamProcessorRequest =
 DescribeStreamProcessorRequest.builder()
            .name(StreamProcessorName)
            .build();

        DescribeStreamProcessorResponse describeStreamProcessorResult =
 rekClient.describeStreamProcessor(streamProcessorRequest);
        System.out.println("Arn - " + describeStreamProcessorResult.streamProcessorArn());
        System.out.println("Input kinesisVideo stream - "
              + describeStreamProcessorResult.input().kinesisVideoStream().arn());
        System.out.println("Output kinesisData stream - "
              + describeStreamProcessorResult.output().kinesisDataStream().arn());
        System.out.println("RoleArn - " + describeStreamProcessorResult.roleArn());
        System.out.println(
              "CollectionId - " +
 describeStreamProcessorResult.settings().faceSearch().collectionId());
        System.out.println("Status - " + describeStreamProcessorResult.status());
        System.out.println("Status message - " +
 describeStreamProcessorResult.statusMessage());
        System.out.println("Creation timestamp - " +
 describeStreamProcessorResult.creationTimestamp());
        System.out.println("Last update timestamp - " +
 describeStreamProcessorResult.lastUpdateTimestamp());
    }

    private static void startSpecificStreamProcessor(RekognitionClient rekClient, String
 StreamProcessorName) {
        try {
            StartStreamProcessorRequest streamProcessorRequest =
 StartStreamProcessorRequest.builder()
                .name(StreamProcessorName)
```

```
                .build();

            rekClient.startStreamProcessor(streamProcessorRequest);
            System.out.println("Stream Processor " + StreamProcessorName + " started.");

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    private static void processCollection(RekognitionClient rekClient, String
StreamProcessorName, String kinInputStream, String kinOutputStream, String collectionName,
String role ) {

        try {
            KinesisVideoStream videoStream = KinesisVideoStream.builder()
                .arn(kinInputStream)
                .build();

            KinesisDataStream dataStream = KinesisDataStream.builder()
                .arn(kinOutputStream)
                .build();

            StreamProcessorOutput processorOutput = StreamProcessorOutput.builder()
                .kinesisDataStream(dataStream)
                .build();

            StreamProcessorInput processorInput = StreamProcessorInput.builder()
                .kinesisVideoStream(videoStream)
                .build();

            FaceSearchSettings searchSettings = FaceSearchSettings.builder()
                .faceMatchThreshold(75f)
                .collectionId(collectionName)
                .build() ;

            StreamProcessorSettings processorSettings = StreamProcessorSettings.builder()
                .faceSearch(searchSettings)
                .build();

            CreateStreamProcessorRequest processorRequest =
CreateStreamProcessorRequest.builder()
                .name(StreamProcessorName)
                .input(processorInput)
                .output(processorOutput)
                .roleArn(role)
                .settings(processorSettings)
                .build();

            CreateStreamProcessorResponse response =
rekClient.createStreamProcessor(processorRequest);
            System.out.println("The ARN for the newly create stream processor is
"+response.streamProcessorArn());

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    private static void deleteSpecificStreamProcessor(RekognitionClient rekClient, String
StreamProcessorName) {

        rekClient.stopStreamProcessor(a->a.name(StreamProcessorName));
        rekClient.deleteStreamProcessor(a->a.name(StreamProcessorName));
```

```
        System.out.println("Stream Processor " + StreamProcessorName + " deleted.");
    }
```

Using stream processors for face searching (Java V1 example)

The following example code shows how to call various stream processor operations, such as CreateStreamProcessor and StartStreamProcessor, using Java V1. 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());
        }
    }
}
```

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. 230).

The Amazon Kinesis Video Streams Parser Library contains example tests that consume Amazon Rekognition Video results and integrates it with the original Kinesis video stream. For more information, see Displaying Rekognition results with Kinesis Video Streams locally (p. 228).

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. 231).

- **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. 232).
- **FaceSearchResponse** – Contains information about faces in the streaming video that match faces in the input collection. FaceSearchResponse (p. 232) contains a DetectedFace (p. 232) object, which is a face that was detected in the analyzed video frame. For each detected face, the array `MatchedFaces` contains an array of matching face objects (MatchedFace (p. 233)) 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. 231) 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. 231) 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.

Displaying Rekognition results with Kinesis Video Streams locally

You can see the results of Amazon Rekognition Video displayed in your feed from Amazon Kinesis Video Streams using the Amazon Kinesis Video Streams Parser Library's example tests provided at KinesisVideo - Rekognition Examples. The `KinesisVideoRekognitionIntegrationExample` displays bounding

boxes over detected faces and renders the video locally through JFrame. This process assumes you have successfully connected a media input from a device camera to a Kinesis video stream and started an Amazon Rekognition Stream Processor. For more information, see Streaming using a GStreamer plugin (p. 233).

Step 1: Installing Kinesis Video Streams Parser Library

To create a directory and download the Github repository, run the following command:

```
$ git clone https://github.com/aws/amazon-kinesis-video-streams-parser-library.git
```

Navigate to the library directory and run the following Maven command to perform a clean installation:

```
$ mvn clean install
```

Step 2: Configuring the Kinesis Video Streams and Rekognition integration example test

Open the `KinesisVideoRekognitionIntegrationExampleTest.java` file. Remove the `@Ignore` right after the class header. Populate the data fields with the information from your Amazon Kinesis and Amazon Rekognition resources. For more information, see Setting up your Amazon Rekognition Video and Amazon Kinesis resources (p. 217). If you are streaming video to your Kinesis video stream, remove the `inputStream` parameter.

See the following code example:

```
RekognitionInput rekognitionInput = RekognitionInput.builder()
  .kinesisVideoStreamArn("arn:aws:kinesisvideo:us-east-1:123456789012:stream/rekognition-
test-video-stream")
  .kinesisDataStreamArn("arn:aws:kinesis:us-east-1:123456789012:stream/AmazonRekognition-
rekognition-test-data-stream")
  .streamingProcessorName("rekognition-test-stream-processor")
  // Refer how to add face collection :
  // https://docs.aws.amazon.com/rekognition/latest/dg/add-faces-to-collection-
procedure.html
  .faceCollectionId("rekognition-test-face-collection")
  .iamRoleArn("rekognition-test-IAM-role")
  .matchThreshold(0.95f)
  .build();

KinesisVideoRekognitionIntegrationExample example =
 KinesisVideoRekognitionIntegrationExample.builder()
  .region(Regions.US_EAST_1)
  .kvsStreamName("rekognition-test-video-stream")
  .kdsStreamName("AmazonRekognition-rekognition-test-data-stream")
  .rekognitionInput(rekognitionInput)
  .credentialsProvider(new ProfileCredentialsProvider())
  // NOTE: Comment out or delete the inputStream parameter if you are streaming video,
 otherwise
  // the test will use a sample video.
  //.inputStream(TestResourceUtil.getTestInputStream("bezos_vogels.mkv"))
  .build();
```

Step 3: Running the Kinesis Video Streams and Rekognition integration example test

Ensure that your Kinesis video stream is receiving media input if you are streaming to it and start analyzing your stream with an Amazon Rekognition Video Stream Processor running. For more information, see Overview of Amazon Rekognition Video stream processor operations (p. 90). Run the `KinesisVideoRekognitionIntegrationExampleTest` class as a JUnit test. After a short delay, a

new window opens with a video feed from your Kinesis video stream with bounding boxes drawn over detected faces.

Note
The faces in the collection used in this example must have External Image Id (the file name) specified in this format in order for bounding box labels to display meaningful text: PersonName1-Trusted, PersonName2-Intruder, PersonName3-Neutral, etc. The labels can also be color-coded and are customizable in the FaceType.java file.

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 video events (p. 90).

Note
The Amazon Rekognition Video API works by comparing the faces in your input stream to a collection of faces, and returning the closest found matches, along with a similarity score.

```
{
    "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": number,
                "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. 231) 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. 232) object

FaceSearchResponse

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

Type: FaceSearchResponse (p. 232) object array

InputInformation

Information about a source video stream that's used by Amazon Rekognition Video. For more information, see Working with streaming video events (p. 90).

KinesisVideo

Type: KinesisVideo (p. 231) object

KinesisVideo

Information about the Kinesis video stream that streams the source video into Amazon Rekognition Video. For more information, see Working with streaming video events (p. 90).

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. For more information, see Working with streaming video events (p. 90).

DetectedFace

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

Type: DetectedFace (p. 232) object

MatchedFaces

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

Type: MatchedFace (p. 233) 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. 233) 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 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 object array

Pose

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

Type: Pose object

Quality

Identifies face image brightness and sharpness.

Type: ImageQuality 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. 232) object.

Type: Face object

Similarity

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

Type: Number

Streaming using a GStreamer plugin

Amazon Rekognition Video can analyze a live streaming video from a device camera. To access media input from a device source, you need to install GStreamer. GStreamer is a third-party multimedia framework software that connects media sources and processing tools together in workflow pipelines. You also need to install the Amazon Kinesis Video Streams Producer Plugin for Gstreamer. This process assumes that you have successfully set up your Amazon Rekognition Video and Amazon Kinesis resources. For more information, see Setting up your Amazon Rekognition Video and Amazon Kinesis resources (p. 217).

Step 1: Install Gstreamer

Download and install Gstreamer, a third-party multi-media platform software. You can use a package management software like Homebrew (Gstreamer on Homebrew) or get it directly from the Freedesktop website.

Verify the successful installation of Gstreamer by launching a video feed with a test source from your command line terminal.

```
$ gst-launch-1.0 videotestsrc ! autovideosink
```

Step 2: Install the Kinesis Video Streams Producer plugin

In this section, you will download the Amazon Kinesis Video Streams Producer Library and install the Kinesis Video Streams Gstreamer plugin.

Create a directory and clone the source code from the Github repository. Be sure to include the `--recursive` parameter.

```
$ git clone --recursive https://github.com/awslabs/amazon-kinesis-video-streams-producer-sdk-cpp.git
```

Follow the instructions provided by the library to configure and build the project. Make sure you use the platform-specific commands for your operating system. Use the `-DBUILD_GSTREAMER_PLUGIN=ON` parameter when you run `cmake` to install the Kinesis Video Streams Gstreamer plugin. This project requires the following additional packages that are included in the installation: GCC or Clang, Curl, Openssl and Log4cplus. If your build fails because of a missing package, verify that the package is installed and in your PATH. If you encounter a "can't run C compiled program" error while building, run the build command again. Sometimes, the correct C compiler is not found.

Verify the installation of the Kinesis Video Streams plugin by running the following command.

```
$ gst-inspect-1.0 kvssink
```

The following information, such as factory and plugin details, should appear:

```


Factory Details:
  Rank                      primary + 10 (266)
  Long-name                 KVS Sink
  Klass                     Sink/Video/Network
  Description               GStreamer AWS KVS plugin
  Author                    AWS KVS <kinesis-video-support@amazon.com>

Plugin Details:
  Name                      kvssink
  Description               GStreamer AWS KVS plugin
  Filename                  /Users/YOUR_USER/amazon-kinesis-video-streams-producer-sdk-cpp/
build/libgstkvssink.so
  Version                   1.0
  License                   Proprietary
  Source module             kvssinkpackage
  Binary package            GStreamer
  Origin URL                http://gstreamer.net/

  ...
```

Step 3: Run Gstreamer with the Kinesis Video Streams plugin

Before you begin streaming from a device camera to Kinesis Video Streams, you might need to convert the media source to an acceptable codec for Kinesis Video Streams. To determine the specifications and format capabilities of devices currently connected to your machine, run the following command.

```
$ gst-device-monitor-1.0
```

To begin streaming, launch Gstreamer with the following sample command and add your credentials and Amazon Kinesis Video Streams information. You should use the access keys and region for the IAM service role you created while giving Amazon Rekognition access to your Kinesis streams. For more information on access keys, see Managing Access Keys for IAM Users in the *IAM User Guide*. Also, you may adjust the video format argument parameters as required by your usage and available from your device.

```
$ gst-launch-1.0 autovideosrc device=/dev/video0 ! videoconvert ! video/x-
raw,format=I420,width=640,height=480,framerate=30/1 !
                x264enc bframes=0 key-int-max=45 bitrate=500 ! video/x-h264,stream-
format=avc,alignment=au,profile=baseline !
                kvssink stream-name="YOUR_STREAM_NAME" storage-size=512 access-
key="YOUR_ACCESS_KEY" secret-key="YOUR_SECRET_ACCESS_KEY" aws-region="YOUR_AWS_REGION"
```

For more launch commands, see Example GStreamer Launch Commands.

> **Note**
> If your launch command terminates with a non-negotiation error, check the output from the Device Monitor and make sure that the `videoconvert` parameter values are valid capabilities of your device.

You will see a video feed from your device camera on your Kinesis video stream after a few seconds. To begin detecting and matching faces with Amazon Rekognition, start your Amazon Rekognition Video stream processor. For more information, see Overview of Amazon Rekognition Video stream processor operations (p. 90).

Troubleshooting streaming video

This topic provides troubleshooting information for using Amazon Rekognition Video with streaming videos.

Topics

I don't know if my stream processor was successfully created

Use the following AWS CLI command to get a list of stream processors and their current status.

```
aws rekognition list-stream-processors
```

You can get additional details by using the following AWS CLI command. Replace `stream-processor-name` with the name of the required stream processor.

```
aws rekognition describe-stream-processor --name stream-processor-name
```

I don't know if I've configured my stream processor correctly

If your code isn't outputting the analysis results from Amazon Rekognition Video, your stream processor might not be configured correctly. Do the following to confirm that your stream processor is configured correctly and able to produce results.

To determine if your solution is configured correctly

1. Run the following command to confirm that your stream processor is in the running state. Change `stream-processor-name` to the name of your stream processor. The stream processor is running if the value of `Status` is `RUNNING`. If the status is `RUNNING` and you aren't getting results, see My stream processor isn't returning results (p. 236). If the status is `FAILED`, see The state of my stream processor is FAILED (p. 237).

```
aws rekognition describe-stream-processor --name stream-processor-name
```

2. If your stream processor is running, run the following Bash or PowerShell command to read data from the output Kinesis data stream.

   Bash

```
SHARD_ITERATOR=$(aws kinesis get-shard-iterator --shard-id shardId-000000000000
 --shard-iterator-type TRIM_HORIZON --stream-name kinesis-data-stream-name --query
 'ShardIterator')
                        aws kinesis get-records --shard-iterator $SHARD_ITERATOR
```

   PowerShell

```
aws kinesis get-records --shard-iterator ((aws kinesis get-shard-iterator --shard-id
 shardId-000000000000 --shard-iterator-type TRIM_HORIZON --stream-name kinesis-data-
stream-name).split('"')[4])
```

3. Use the Decode tool on the Base64 Decode website to decode the output into a human-readable string. For more information, see Step 3: Get the Record.
4. If the commands work and you see face detection results in the Kinesis data stream, then your solution is properly configured. If the command fails, check the other troubleshooting suggestions and see Giving Amazon Rekognition Video access to your resources (p. 218).

Alternatively, you can use the "kinesis-process-record" AWS Lambda blueprint to log messages from the Kinesis data stream to CloudWatch for continuous visualization. This incurs additional costs for AWS Lambda and CloudWatch.

My stream processor isn't returning results

Your stream processor might not return results for several reasons.

Reason 1: Your stream processor isn't configured correctly

Your stream processor might not be configured correctly. For more information, see I don't know if I've configured my stream processor correctly (p. 235).

Reason 2: Your stream processor isn't in the RUNNING state

To troubleshoot the status of a stream processor

1. Check the status of the stream processor with the following AWS CLI command.

```
aws rekognition describe-stream-processor --name stream-processor-name
```

2. If the value of `Status` is `STOPPED`, start your stream processor with the following command:

```
aws rekognition start-stream-processor --name stream-processor-name
```

3. If the value of `Status` is `FAILED`, see The state of my stream processor is FAILED (p. 237).
4. If the value of `Status` is `STARTING`, wait for 2 minutes and check the status by repeating step 1. If the value of Status is still `STARTING`, do the following:

    a. Delete the stream processor with the following command.

    ```
    aws rekognition delete-stream-processor --name stream-processor-name
    ```

    b. Create a new stream processor with the same configuration. For more information, see Working with streaming video events (p. 90).
    c. If you're still having problems, contact AWS Support.
5. If the value of `Status` is `RUNNING`, see Reason 3: There isn't active data in the Kinesis video stream (p. 237).

Reason 3: There isn't active data in the Kinesis video stream

To check if there's active data in the Kinesis video stream

1. Sign in to the AWS Management Console, and open the Amazon Kinesis Video Streams console at https://console.aws.amazon.com/kinesisvideo/.
2. Select the Kinesis video stream that's the input for the Amazon Rekognition stream processor.
3. If the preview states **No data on stream**, then there's no data in the input stream for Amazon Rekognition Video to process.

For information about producing video with Kinesis Video Streams, see Kinesis Video Streams Producer Libraries.

The state of my stream processor is FAILED

You can check the state of a stream processor by using the following AWS CLI command.

```
aws rekognition describe-stream-processor --name stream-processor-name
```

If the value of Status is FAILED, check the troubleshooting information for the following error messages.

Error: "Access denied to Role"

The IAM role that's used by the stream processor doesn't exist or Amazon Rekognition Video doesn't have permission to assume the role.

To troubleshoot access to the IAM role

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. From the left navigation pane, choose **Roles**and confirm that the role exists.
3. If the role exists, check that the role has the *AmazonRekognitionServiceRole* permissions policy.
4. If the role doesn't exist or doesn't have the right permissions, see Giving Amazon Rekognition Video access to your resources (p. 218).
5. Start the stream processor with the following AWS CLI command.

    ```
    aws rekognition start-stream-processor --name stream-processor-name
    ```

Error: "Access denied to Kinesis Video *or* Access denied to Kinesis Data"

The role doesn't have access to the Kinesis Video Streams API operations `GetMedia` and `GetDataEndpoint`. It also might not have access to the Kinesis Data Streams API operations `PutRecord` and `PutRecords`.

To troubleshoot API permissions

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Open the role and make sure that it has the following permissions policy attached.

```
{
    "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"
        }
    ]
}
```

3. If any of the permissions are missing, update the policy. For more information, see Giving Amazon Rekognition Video access to your resources (p. 218).

Error: "Stream *input-video-stream-name* doesn't exist"

The Kinesis video stream input to the stream processor doesn't exist or isn't configured correctly.

To troubleshoot the Kinesis video stream

1. Use the following command to confirm that the stream exists.

```
aws kinesisvideo list-streams
```

2. If the stream exists, check the following.

   - The Amazon Resource Name (ARN) is same as the ARN of the input stream for the stream processor.
   - The Kinesis video stream is in the same Region as the stream processor.

   If the stream processor isn't configured correctly, delete it with the following AWS CLI command.

```
aws rekognition delete-stream-processor --name stream-processor-name
```

3. Create a new stream processor with the intended Kinesis video stream. For more information, see Creating the Amazon Rekognition Video face search stream processor (p. 220).

Error: "Collection not found"

The Amazon Rekognition collection that's used by the stream processor to match faces doesn't exist, or the wrong collection is being used.

To confirm the collection

1. Use the following AWS CLI command to determine if the required collection exists. Change `region` to the AWS Region in which you're running your stream processor.

```
aws rekognition list-collections --region region
```

   If the required collection doesn't exist, create a new collection and add face information. For more information, see Searching faces in a collection (p. 157).
2. In your call to CreateStreamProcessor, check that the value of the `CollectionId` input parameter is correct.
3. Start the stream processor with the following AWS CLI command.

```
aws rekognition start-stream-processor --name stream-processor-name
```

Error: "Stream *output-kinesis-data-stream-name* under account *account-id* not found"

The output Kinesis data stream that's used by the stream processor doesn't exist in your AWS account or isn't in the same AWS Region as your stream processor.

To troubleshoot the Kinesis data stream

1. Use the following AWS CLI command to determine if the Kinesis data stream exists. Change `region` to the AWS Region in which you're using your stream processor.

```
aws kinesis list-streams --region region
```

2. If the Kinesis data stream exists, check that the Kinesis data stream name is same as the name of the output stream that's used by the stream processor.
3. If the Kinesis data stream doesn't exist, it might exist in another AWS Region. The Kinesis data stream must be in the same Region as the stream processor.
4. If necessary, create a new Kinesis data stream.
   a. Create a Kinesis data stream with the same name as the name used by the stream processor. For more information, see Step 1: Create a Data Stream.
   b. Start the stream processor with the following AWS CLI command.

```
aws rekognition start-stream-processor --name stream-processor-name
```

My stream processor isn't returning the expected results

If your stream processor isn't returning the expected face matches, use the following information.

- Searching faces in a collection (p. 157)
- Recommendations for camera setup (streaming video) (p. 102)

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. Amazon Rekognition Video publishes the completion status of the video analysis to an Amazon Simple Notification Service topic. If the video analysis is successful, call GetPersonTracking to get results of the video analysis. For more information about calling Amazon Rekognition Video API operations, see Calling Amazon Rekognition Video operations (p. 62).

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 StartPersonDetection(String bucket, String video)
 throws Exception{


            NotificationChannel channel= new NotificationChannel()
                    .withSNSTopicArn(snsTopicArn)
                    .withRoleArn(roleArn);

         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 GetPersonDetectionResults() 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);

        }
```

In the function `main`, replace the lines:

```
        StartLabelDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetLabelDetectionResults();
```

with:

```
        StartPersonDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetPersonDetectionResults();
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void startPersonLabels(RekognitionClient rekClient,
```

```
                                      NotificationChannel channel,
                                      String bucket,
                                      String video) {
       try {
           S3Object s3Obj = S3Object.builder()
               .bucket(bucket)
               .name(video)
               .build();

           Video vidOb = Video.builder()
               .s3Object(s3Obj)
               .build();

           StartPersonTrackingRequest personTrackingRequest =
StartPersonTrackingRequest.builder()
               .jobTag("DetectingLabels")
               .video(vidOb)
               .notificationChannel(channel)
               .build();

           StartPersonTrackingResponse labelDetectionResponse =
rekClient.startPersonTracking(personTrackingRequest);
           startJobId = labelDetectionResponse.jobId();

       } catch(RekognitionException e) {
           System.out.println(e.getMessage());
           System.exit(1);
       }
   }

   public static void GetPersonDetectionResults(RekognitionClient rekClient) {

       try {
           String paginationToken=null;
           GetPersonTrackingResponse personTrackingResult=null;
           boolean finished = false;
           String status;
           int yy=0 ;

           do{
               if (personTrackingResult !=null)
                   paginationToken = personTrackingResult.nextToken();

               GetPersonTrackingRequest recognitionRequest =
GetPersonTrackingRequest.builder()
                       .jobId(startJobId)
                       .nextToken(paginationToken)
                       .maxResults(10)
                       .build();

               // Wait until the job succeeds
               while (!finished) {

                   personTrackingResult =
rekClient.getPersonTracking(recognitionRequest);
                   status = personTrackingResult.jobStatusAsString();

                   if (status.compareTo("SUCCEEDED") == 0)
                       finished = true;
                   else {
                       System.out.println(yy + " status is: " + status);
                       Thread.sleep(1000);
                   }
                   yy++;
               }
```

```
                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is null
                VideoMetadata videoMetaData = personTrackingResult.videoMetadata();

                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " + videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<PersonDetection> detectedPersons=
 personTrackingResult.persons();
                for (PersonDetection detectedPerson: detectedPersons) {

                    long seconds=detectedPerson.timestamp()/1000;
                    System.out.print("Sec: " + seconds + " ");
                    System.out.println("Person Identifier: " +
 detectedPerson.person().index());
                    System.out.println();
                }

            } while (personTrackingResult !=null &&
 personTrackingResult.nextToken() != null);

        } catch(RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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.)

    # ============= People pathing ===============
    def StartPersonPathing(self):
        response=self.rek.start_person_tracking(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
            NotificationChannel={'RoleArn': self.roleArn, 'SNSTopicArn':
 self.snsTopicArn})

        self.startJobId=response['JobId']
        print('Start Job Id: ' + self.startJobId)

    def GetPersonPathingResults(self):
        maxResults = 10
        paginationToken = ''
        finished = False

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

            print('Codec: ' + response['VideoMetadata']['Codec'])
            print('Duration: ' + str(response['VideoMetadata']['DurationMillis']))
            print('Format: ' + response['VideoMetadata']['Format'])
            print('Frame rate: ' + str(response['VideoMetadata']['FrameRate']))
            print()

            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
```

In the function `main`, replace the lines:

```
    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()
```

with:

```
    analyzer.StartPersonPathing()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetPersonPathingResults()
```

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 code to replace might be 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 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. 64).
- **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
    }
}
```

Detecting personal protective equipment

Amazon Rekognition can detect Personal Protective Equipment (PPE) worn by persons in an image. You can use this information to improve workplace safety practices. For example, you can use PPE detection to help determine if workers on a construction site are wearing head covers, or if medical workers are wearing face covers and hand covers. The following image shows some of the types of PPE that can detected.

To detect PPE in an image you call the DetectProtectiveEquipment API and pass an input image. The response is a JSON structure that includes the following.

- The persons detected in the image.
- The parts of a body where where PPE is worn (face, head, left-hand, and right-hand).
- The types of PPE detected on body parts (face cover, hand cover, and head cover).
- For items of detected PPE, an indicator for whether or not the PPE covers the corresponding body part.

Bounding boxes are returned for the locations of persons and items of PPE detected in the image.

Optionally, you can request a summary of the PPE items and persons detected in an image. For more information, see Summarizing PPE detected in an image (p. 249).

Note
Amazon Rekognition PPE detection doesn't perform facial recognition or facial comparison and can't identify the detected persons.

Types of PPE

DetectProtectiveEquipment detects the following types of PPE. If you want to detect other types of PPE in images, consider using Amazon Rekognition Custom Labels to train a custom model. For more information, see Amazon Rekognition Custom Labels.

Face cover

`DetectProtectiveEquipment` can detect common face covers such as surgical, N95, and masks made of cloth.

Hand cover

`DetectProtectiveEquipment` can detect hand covers such as surgical gloves and safety gloves.

Head cover

`DetectProtectiveEquipment` can detect hard hats and helmets.

The API indicates that a head, hand, or face cover was detected in an image. The API doesn't return information about the type of a specific cover. For example, 'surgical glove' for the type of a hand cover.

PPE detection confidence

Amazon Rekognition makes a prediction about the presence of PPE, persons, and body parts in an image. The API provides a score (50-100) that indicates how confident Amazon Rekognition is in the accuracy of a prediction.

> **Note**
> If you plan to use the `DetectProtectiveEquipment` operation to make a decision that impacts an individual's rights, privacy, or access to services we recommend that you pass the result to a human for review and validation before taking action.

Summarizing PPE detected in an image

You can optionally request a summary of the PPE items and persons detected in an image. You can specify a list of required protective equipment (face cover, hand cover, or head cover) and a minimum confidence threshold (for example, 80%). The response includes a consolidated per-image identifier (ID) summary of persons with the required PPE, persons without the required PPE, and persons where a determination couldn't be made.

The summary allows you to quickly answer questions such as *How many persons are not wearing face covers?* or *Is everyone wearing PPE?* Each detected person in the summary has a unique ID. You can use the ID find out information such as the bounding box location of a person not wearing PPE.

> **Note**
> The ID is randomly generated on a per-image analysis basis and is not consistent across images or multiple analyses of the same image.

You can summarize face covers, head covers, hand covers, or a combination of your choice. To specify the required types of PPE, see Specifying summarization requirements (p. 251). You can also specify a minimum confidence level (50-100) that must be met for detections to be included in the summary.

For more information about the summarization response from `DetectProtectiveEquipment`, see Understanding the DetectProtectiveEquipment response (p. 251).

Tutorial: Creating a AWS Lambda function that detects images with PPE

You can create an AWS Lambda function that detects personal protective equipment (PPE) in images located in an Amazon S3 bucket. See the AWS Documentation SDK examples GitHub repository for this Java V2 tutorial.

Understanding the personal protective equipment detection API

The following information describes the DetectProtectiveEquipment API. For example code, see Detecting personal protective equipment in an image (p. 254).

Supplying an image

You can provide the input image (JPG or PNG format) either as image bytes or reference an image stored in an Amazon S3 bucket.

We recommend using images where the person's face is facing the camera.

If your input image isn't rotated to 0 degrees orientation, we recommend rotating it to 0 degrees orientation before submitting it to `DetectProtectiveEquipment`. Images in JPG format might contain orientation information in Exchangeable image file format (Exif) metadata. You can use this information to write code that rotates your image. For more information, see Exif version 2.32. PNG format images don't contain image orientation information.

To pass an image from an Amazon S3 bucket, use an IAM user with at least AmazonS3ReadOnlyAccess priviliges. Use an IAM user user with `AmazonRekognitionFullAccess` priviliges to call `DetectProtectiveEquipment.`

In the following example input JSON, the image is passed in an Amazon S3 bucket. For more information, see Working with images (p. 26). The example requests a summary of all PPE types (head cover, hand cover, and face cover) with a minimum detection confidence (`MinConfidence`) of 80%. You should specify a `MinConfidence` value that is between 50-100% as `DetectProtectiveEquipment` returns predictions only where the detection confidence is between 50% - 100%. If you specify a value that is less than 50%, the results are the same specifying a value of 50%. For more information, see Specifying summarization requirements (p. 251).

```
{
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "worker.jpg"
        }
    },
    "SummarizationAttributes": {
        "MinConfidence": 80,
        "RequiredEquipmentTypes": [
            "FACE_COVER",
            "HAND_COVER",
            "HEAD_COVER"
```

```
        ]
    }
}
```

If you have a large collection of images to process, consider using AWS Batch to process calls to `DetectProtectiveEquipment` in batches in the background.

Specifying summarization requirements

You can optionally use the `SummarizationAttributes` (ProtectiveEquipmentSummarizationAttributes) input parameter to request summary information for the types of PPE detected in an image.

To specify the types of PPE to summarize, use the `RequiredEquipmentTypes` array field. In the array, include one or more of `FACE_COVER`, `HAND_COVER` or `HEAD_COVER`.

Use the `MinConfidence` field to specify a minimum detection confidence (50-100). The summary doesn't include Persons, body parts, body part coverage, and items of PPE, detected with a confidence lower than `MinConfidence`.

For information about the summary response from `DetectProtectiveEquipment`, see Understanding the DetectProtectiveEquipment response (p. 251).

Understanding the DetectProtectiveEquipment response

`DetectProtectiveEquipment` returns an array of persons detected in the input image. For each person, information about detected body parts and detected items of PPE is returned. The JSON for the following image of a worker wearing a head cover, hand cover, and a face cover is as follows.

In the JSON, note the following.

- **Detected Persons** – `Persons` is an array of persons detected on the image (including persons not wearing PPE). `DetectProtectiveEquipment` can detect PPE on up to 15 persons detected in an image. Each ProtectiveEquipmentPerson object in the array contains a person ID, a bounding box for the person, detected body parts, and detected items of PPE. The value of `Confidence` in

`ProtectiveEquipmentPerson` indicates the percentage confidence that Amazon Rekognition has that the bounding box contains a person.

- **Body Parts** – `BodyParts` is an array of body parts (ProtectiveEquipmentBodyPart) detected on a person (including body parts not covered by PPE). Each `ProtectiveEquipmentBodyPart` includes the name (`Name`) of the detected body part. `DetectProtectEquipment` can detect face, head, left-hand, and right-hand body parts. The `Confidence` field in `ProtectiveEquipmentBodyPart` indicates the percentage confidence that Amazon Rekognition has in the detection accuracy of the body part.
- **PPE Items** – The array `EquipmentDetections` in an `ProtectiveEquipmentBodyPart` object contains an array of detected PPE items. Each EquipmentDetection object contains the following fields.
  - `Type` – The type of the detected PPE.
  - `BoundingBox` – a bounding box around the detected PPE.
  - `Confidence` – The confidence Amazon Rekognition has that the bounding box contains the detected PPE.
  - `CoversBodyPart` – Indicates if the detected PPE is on the corresponding body part.

  The CoversBodyPart field `Value` is a boolean value that indicates if the detected PPE is on the corresponding body part. The field `Confidence` indicates the confidence in the prediction. You can use `CoversBodyPart` to filter out cases where the detected PPE is in the image, but not actually on the person.

  > **Note**
  > `CoversBodyPart` doesn't indicate, or imply, that the person is adequately protected by the protective equipment or that the protective equipment itself is properly worn.
- **Summary Information** – `Summary` contains the summary information specified in the `SummarizationAttributes` input parameter. For more information, see Specifying summarization requirements (p. 251).

  `Summary` is an object of type ProtectiveEquipmentSummary which contains the following information.
  - `PersonsWithRequiredEquipment` – An array of the IDs of persons where each person meets the following criteria.
    - The person is wearing all of the PPE specified in the `SummarizationAttributes` input parameter.
    - The `Confidence` for the person (`ProtectiveEquipmentPerson`), body part (`ProtectiveEquipmentBodyPart`), protective equipment (`EquipmentDetection`) is equal to or greater than the specified minimum confidence threshold (`MinConfidence`).
    - The value of `CoversBodyPart` for all items of PPE is true.
  - `PersonsWithoutRequiredEquipment` – An array of the IDs of persons that meet one of the following criteria.
    - The `Confidence` value for the person (`ProtectiveEquipmentPerson`), body part (`ProtectiveEquipmentBodyPart`), and body part coverage (`CoversBodyPart`) are greater than the specified minimum confidence threshold (`MinConfidence`), but the person is missing one or more specified PPE (`SummarizationAttributes`).
    - The value of `CoversBodyPart` is false for any specified PPE (`SummarizationAttributes`) that has a `Confidence` value greater than the specified minimum confidence threshold (`MinConfidence`). The person also has all the specified PPE (`SummarizationAttributes`) and the `Confidence` values for person (`ProtectiveEquipmentPerson`), body part (`ProtectiveEquipmentBodyPart`), and protective equipment (`EquipmentDetection`) are greater than or equal to the minimum confidence threshold (`MinConfidence`.
  - `PersonsIndeterminate` – An array of the IDs of persons detected where the `Confidence` value for the person (`ProtectiveEquipmentPerson`), body part (`ProtectiveEquipmentBodyPart`), protective equipment (`EquipmentDetection`), or `CoversBodyPart` boolean is lower than specified minimum confidence threshold (`MinConfidence`).

Use the array size to get a count for a particular summary. For example, the size of `PersonsWithRequiredEquipment` tells you the number of people detected as wearing the specified type of PPE.

You can use the person ID to find out further information about a person, such as the bounding box location of the person. The person ID maps to the ID field of a `ProtectiveEquipmentPerson`) object returned in `Persons` (array of`ProtectiveEquipmentPerson`). You can then get the Bounding box and other information from the corresponding `ProtectiveEquipmentPerson` object.


```
{
   "ProtectiveEquipmentModelVersion": "1.0",
   "Persons": [
       {
           "BodyParts": [
               {
                   "Name": "FACE",
                   "Confidence": 99.99861145019531,
                   "EquipmentDetections": [
                       {
                           "BoundingBox": {
                               "Width": 0.14528800547122955,
                               "Height": 0.14956723153591156,
                               "Left": 0.4363413453102112,
                               "Top": 0.34203192591667175
                           },
                           "Confidence": 99.90001678466797,
                           "Type": "FACE_COVER",
                           "CoversBodyPart": {
                               "Confidence": 98.0676498413086,
                               "Value": true
                           }
                       }
                   ]
               },
               {
                   "Name": "LEFT_HAND",
                   "Confidence": 96.9786376953125,
                   "EquipmentDetections": [
                       {
                           "BoundingBox": {
                               "Width": 0.14495663344860077,
                               "Height": 0.12936046719551086,
                               "Left": 0.5114737153053284,
                               "Top": 0.5744519829750061
                           },
                           "Confidence": 83.72270965576172,
                           "Type": "HAND_COVER",
                           "CoversBodyPart": {
                               "Confidence": 96.9288558959961,
                               "Value": true
                           }
                       }
                   ]
               },
               {
                   "Name": "RIGHT_HAND",
                   "Confidence": 99.82939147949219,
                   "EquipmentDetections": [
                       {
                           "BoundingBox": {
                               "Width": 0.20971858501434326,
```

```
                                    "Height": 0.20528452098369598,
                                    "Left": 0.2711356580257416,
                                    "Top": 0.6750612258911133
                                },
                                "Confidence": 95.70789337158203,
                                "Type": "HAND_COVER",
                                "CoversBodyPart": {
                                    "Confidence": 99.85433197021484,
                                    "Value": true
                                }
                            }
                        ]
                    },
                    {
                        "Name": "HEAD",
                        "Confidence": 99.9999008178711,
                        "EquipmentDetections": [
                            {
                                "BoundingBox": {
                                    "Width": 0.24350935220718384,
                                    "Height": 0.34623199701309204,
                                    "Left": 0.43011072278022766,
                                    "Top": 0.01103297434747219
                                },
                                "Confidence": 83.88762664794922,
                                "Type": "HEAD_COVER",
                                "CoversBodyPart": {
                                    "Confidence": 99.96485900878906,
                                    "Value": true
                                }
                            }
                        ]
                    }
                ],
                "BoundingBox": {
                    "Width": 0.7403100728988647,
                    "Height": 0.9412225484848022,
                    "Left": 0.02214839495718479,
                    "Top": 0.03134796395897865
                },
                "Confidence": 99.98855590820312,
                "Id": 0
            }
        ],
        "Summary": {
            "PersonsWithRequiredEquipment": [
                0
            ],
            "PersonsWithoutRequiredEquipment": [],
            "PersonsIndeterminate": []
        }
    }
```


Detecting personal protective equipment in an image

To detect Personal Protective Equipment (PPE) on persons in an image, use the DetectProtectiveEquipment non-storage API operation.

You can provide the input image as an image byte array (base64-encoded image bytes) or as an Amazon S3 object, by using the AWS SDK or the AWS Command Line Interface (AWS CLI). These examples use an image stored in an Amazon S3 bucket. For more information, see Working with images (p. 26).

To detect PPE on persons 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. 13).

2. Upload an image (that contains one or more persons wearing PPE) to your S3 bucket.

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

3. Use the following examples to call the `DetectProtectiveEquipment` operation. For information about displaying bounding boxes in an image, see Displaying bounding boxes (p. 46).

   Java

   This example displays information about the PPE items detected on persons detected in an image.

   Change the value of `bucket` to the name of the Amazon S3 bucket that contains your image. Change the value of `photo` to your image file name.

```
//Copyright 2020 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.client.builder.AwsClientBuilder;
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.ProtectiveEquipmentBodyPart;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.rekognition.model.ProtectiveEquipmentPerson;
import
 com.amazonaws.services.rekognition.model.ProtectiveEquipmentSummarizationAttributes;

import java.util.List;
import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.model.DetectProtectiveEquipmentRequest;
import com.amazonaws.services.rekognition.model.DetectProtectiveEquipmentResult;
import com.amazonaws.services.rekognition.model.EquipmentDetection;


public class DetectPPE {

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

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


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

```
        ProtectiveEquipmentSummarizationAttributes summaryAttributes = new
ProtectiveEquipmentSummarizationAttributes()
                .withMinConfidence(80F)
                .withRequiredEquipmentTypes("FACE_COVER", "HAND_COVER",
"HEAD_COVER");

        DetectProtectiveEquipmentRequest request = new
DetectProtectiveEquipmentRequest()
                .withImage(new Image()
                        .withS3Object(new S3Object()
                                .withName(photo).withBucket(bucket)))
                .withSummarizationAttributes(summaryAttributes);

        try {
            System.out.println("Detected PPE for people in image " + photo);
            System.out.println("Detected people\n---------------");
            DetectProtectiveEquipmentResult result =
rekognitionClient.detectProtectiveEquipment(request);


            List <ProtectiveEquipmentPerson> persons = result.getPersons();


            for (ProtectiveEquipmentPerson person: persons) {
                System.out.println("ID: " + person.getId());
                List<ProtectiveEquipmentBodyPart> bodyParts=person.getBodyParts();
                if (bodyParts.isEmpty()){
                    System.out.println("\tNo body parts detected");
                } else
                    for (ProtectiveEquipmentBodyPart bodyPart: bodyParts) {
                        System.out.println("\t" + bodyPart.getName() + ".
Confidence: " + bodyPart.getConfidence().toString());



                        List<EquipmentDetection>
equipmentDetections=bodyPart.getEquipmentDetections();

                        if (equipmentDetections.isEmpty()){
                            System.out.println("\t\tNo PPE Detected on " +
bodyPart.getName());

                        }
                        else {
                            for (EquipmentDetection item: equipmentDetections) {
                                System.out.println("\t\tItem: " + item.getType() +
". Confidence: " + item.getConfidence().toString());
                                System.out.println("\t\tCovers body part: "
                                        +
item.getCoversBodyPart().getValue().toString() + ". Confidence: " +
item.getCoversBodyPart().getConfidence().toString());

                                System.out.println("\t\tBounding Box");
                                BoundingBox box =item.getBoundingBox();

                                System.out.println("\t\tLeft: "
+box.getLeft().toString());
                                System.out.println("\t\tTop: " +
box.getTop().toString());
                                System.out.println("\t\tWidth: " +
box.getWidth().toString());
                                System.out.println("\t\tHeight: " +
box.getHeight().toString());
                                System.out.println("\t\tConfidence: " +
item.getConfidence().toString());
```

```
                                System.out.println();
                            }
                        }

                    }
            }
            System.out.println("Person ID Summary\n-----------------");

            //List<Integer> list=;
            DisplaySummary("With required equipment",
result.getSummary().getPersonsWithRequiredEquipment());
            DisplaySummary("Without required equipment",
result.getSummary().getPersonsWithoutRequiredEquipment());
            DisplaySummary("Indeterminate",
result.getSummary().getPersonsIndeterminate());


        } catch(AmazonRekognitionException e) {
            e.printStackTrace();
        }
    }
    static void DisplaySummary(String summaryType,List<Integer> idList)
    {
        System.out.print(summaryType + "\n\tIDs  ");
        if (idList.size()==0) {
            System.out.println("None");
        }
        else {
            int count=0;
            for (Integer id: idList ) {
                if (count++ == idList.size()-1) {
                    System.out.println(id.toString());
                }
                else {
                    System.out.print(id.toString() + ", ");
                }
            }
        }

        System.out.println();


    }
}
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void displayGear(S3Client s3,
                                   RekognitionClient rekClient,
                                   String sourceImage,
                                   String bucketName) {

        byte[] data = getObjectBytes (s3, bucketName, sourceImage);
        InputStream is = new ByteArrayInputStream(data);

        try {
            ProtectiveEquipmentSummarizationAttributes summarizationAttributes =
ProtectiveEquipmentSummarizationAttributes.builder()
                .minConfidence(80F)
                .requiredEquipmentTypesWithStrings("FACE_COVER", "HAND_COVER",
"HEAD_COVER")
```

```
                .build();

            SdkBytes sourceBytes = SdkBytes.fromInputStream(is);
            software.amazon.awssdk.services.rekognition.model.Image souImage =
Image.builder()
                .bytes(sourceBytes)
                .build();

            DetectProtectiveEquipmentRequest request =
DetectProtectiveEquipmentRequest.builder()
                .image(souImage)
                .summarizationAttributes(summarizationAttributes)
                .build();

            DetectProtectiveEquipmentResponse result =
rekClient.detectProtectiveEquipment(request);
            List<ProtectiveEquipmentPerson> persons = result.persons();
            for (ProtectiveEquipmentPerson person: persons) {
                System.out.println("ID: " + person.id());
                List<ProtectiveEquipmentBodyPart> bodyParts=person.bodyParts();
                if (bodyParts.isEmpty()){
                    System.out.println("\tNo body parts detected");
                } else
                    for (ProtectiveEquipmentBodyPart bodyPart: bodyParts) {
                        System.out.println("\t" + bodyPart.name() + ". Confidence:
" + bodyPart.confidence().toString());
                        List<EquipmentDetection>
equipmentDetections=bodyPart.equipmentDetections();

                        if (equipmentDetections.isEmpty()){
                            System.out.println("\t\tNo PPE Detected on " +
bodyPart.name());
                        } else {
                            for (EquipmentDetection item: equipmentDetections) {
                                System.out.println("\t\tItem: " + item.type() + ".
Confidence: " + item.confidence().toString());
                                System.out.println("\t\tCovers body part: "
                                        + item.coversBodyPart().value().toString()
+ ". Confidence: " + item.coversBodyPart().confidence().toString());

                                System.out.println("\t\tBounding Box");
                                BoundingBox box =item.boundingBox();
                                System.out.println("\t\tLeft: "
+box.left().toString());
                                System.out.println("\t\tTop: " +
box.top().toString());
                                System.out.println("\t\tWidth: " +
box.width().toString());
                                System.out.println("\t\tHeight: " +
box.height().toString());
                                System.out.println("\t\tConfidence: " +
item.confidence().toString());
                                System.out.println();
                            }
                        }
                    }
            }
            System.out.println("Person ID Summary\n-----------------");

            displaySummary("With required equipment",
result.summary().personsWithRequiredEquipment());
            displaySummary("Without required equipment",
result.summary().personsWithoutRequiredEquipment());
            displaySummary("Indeterminate",
result.summary().personsIndeterminate());
```

```
        } catch (RekognitionException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static byte[] getObjectBytes (S3Client s3, String bucketName, String
 keyName) {

        try {
            GetObjectRequest objectRequest = GetObjectRequest
                .builder()
                .key(keyName)
                .bucket(bucketName)
                .build();

            ResponseBytes<GetObjectResponse> objectBytes =
 s3.getObjectAsBytes(objectRequest);
            return objectBytes.asByteArray();

        } catch (S3Exception e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
    }

    static void displaySummary(String summaryType,List<Integer> idList) {
        System.out.print(summaryType + "\n\tIDs  ");
        if (idList.size()==0) {
            System.out.println("None");
        } else {
            int count=0;
            for (Integer id: idList ) {
                if (count++ == idList.size()-1) {
                    System.out.println(id.toString());
                } else {
                    System.out.print(id.toString() + ", ");
                }
            }
        }
        System.out.println();
    }
```

AWS CLI

This AWS CLI command requests a PPE summary and displays the JSON output for the `detect-protective-equipment` CLI operation.

Change `bucketname` to the name of an Amazon S3 bucket that contains an image. Change `input.jpg` to the name of the image that you want to use.

```
aws rekognition detect-protective-equipment \
  --image "S3Object={Bucket=bucketname,Name=input.jpg}" \
  --summarization-attributes
 "MinConfidence=80,RequiredEquipmentTypes=['FACE_COVER','HAND_COVER','HEAD_COVER']"
```

This AWS CLI command displays the JSON output for the `detect-protective-equipment` CLI operation.

Change `bucketname` to the name of an Amazon S3 bucket that contains an image. Change `input.jpg` to the name of the image that you want to use.

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

Python

This example displays information about the PPE items detected on persons detected in an image.

Change the value of `bucket` to the name of the Amazon S3 bucket that contains your image. Change the value of `photo` to your image file name.

```
#Copyright 2020 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

def detect_labels(photo, bucket):

    client=boto3.client('rekognition')

    response = client.detect_protective_equipment(Image={'S3Object':
{'Bucket':bucket,'Name':photo}},
        SummarizationAttributes={'MinConfidence':80, 'RequiredEquipmentTypes':
['FACE_COVER', 'HAND_COVER', 'HEAD_COVER']})


    print('Detected PPE for people in image ' + photo)
    print('\nDetected people\n---------------')
    for person in response['Persons']:

        print('Person ID: ' + str(person['Id']))
        print ('Body Parts\n----------')
        body_parts = person['BodyParts']
        if len(body_parts) == 0:
                print ('No body parts found')
        else:
            for body_part in body_parts:
                print('\t'+ body_part['Name'] + '\n\t\tConfidence: ' +
 str(body_part['Confidence']))
                print('\n\t\tDetected PPE\n\t\t------------')
                ppe_items = body_part['EquipmentDetections']
                if len(ppe_items) ==0:
                    print ('\t\tNo PPE detected on ' + body_part['Name'])
                else:
                    for ppe_item in ppe_items:
                        print('\t\t' + ppe_item['Type'] + '\n\t\t\tConfidence: ' +
 str(ppe_item['Confidence']))
                        print('\t\tCovers body part: ' +
 str(ppe_item['CoversBodyPart']['Value']) + '\n\t\t\tConfidence: ' +
 str(ppe_item['CoversBodyPart']['Confidence']))
                        print('\t\tBounding Box:')
                        print ('\t\t\tTop: ' + str(ppe_item['BoundingBox']['Top']))
                        print ('\t\t\tLeft: ' + str(ppe_item['BoundingBox']
['Left']))
                        print ('\t\t\tWidth: ' +  str(ppe_item['BoundingBox']
['Width']))
                        print ('\t\t\tHeight: ' +  str(ppe_item['BoundingBox']
['Height']))
                        print ('\t\t\tConfidence: ' + str(ppe_item['Confidence']))
            print()
        print()
```

```
    print('Person ID Summary\n----------------')
    display_summary('With required equipment',response['Summary']
['PersonsWithRequiredEquipment'] )
    display_summary('Without required equipment',response['Summary']
['PersonsWithoutRequiredEquipment'] )
    display_summary('Indeterminate',response['Summary']['PersonsIndeterminate'] )

    print()
    return len(response['Persons'])

#Display summary information for supplied summary.
def display_summary(summary_type, summary):
    print (summary_type + '\n\tIDs: ',end='')
    if (len(summary)==0):
        print('None')
    else:
        for num, id in enumerate(summary, start=0):
            if num==len(summary)-1:
                print (id)
            else:
                print (str(id) + ', ' , end='')



def main():
    photo='photo'
    bucket='bucket'
    person_count=detect_labels(photo, bucket)
    print("Persons detected: " + str(person_count))


if __name__ == "__main__":
    main()
```

Example: Drawing bounding boxes around face covers

The following examples shows you how to draw bounding boxes around face covers detected on persons. For an example that uses AWS Lambda and Amazon DynamoDB, see the AWS Documentation SDK examples GitHub repository.

To detect face covers you use the DetectProtectiveEquipment non-storage API operation. The image is loaded from the local file system. You provide the input image to `DetectProtectiveEquipment` as an image byte array (base64-encoded image bytes). For more information, see Working with images (p. 26).

The example displays a bounding box around detected face covers. The bounding box is green if the face cover fully covers the body part. Otherwise a red bounding box is displayed. As a warning, a yellow bounding box is displayed within the face cover bounding box, if the detection confidence is lower than the specified confidence value. If a face cover is not detected, a red bounding box is drawn around the person.

The image output is similar to the following.

To display bounding boxes on detected face covers

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. 13).

2. Use the following examples to call the `DetectProtectiveEquipment` operation. For information about displaying bounding boxes in an image, see Displaying bounding boxes (p. 46).

   Java

   In the function `main`, change the following:

   - The value of `photo` to path and file name of a local image file (PNG or JPEG).
   - The value of `confidence` to the desired confidence level (50-100).

```
//Loads images, detects faces and draws bounding boxes.Determines exif orientation,
 if necessary.
package com.amazonaws.samples;


import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.*;
```

```
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 com.amazonaws.util.IOUtils;

import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.model.DetectProtectiveEquipmentRequest;
import com.amazonaws.services.rekognition.model.DetectProtectiveEquipmentResult;
import com.amazonaws.services.rekognition.model.EquipmentDetection;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.ProtectiveEquipmentBodyPart;
import com.amazonaws.services.rekognition.model.ProtectiveEquipmentPerson;

// Calls DetectFaces and displays a bounding box around each detected image.
public class PPEBoundingBox extends JPanel {

    private static final long serialVersionUID = 1L;

    BufferedImage image;
    static int scale;
    DetectProtectiveEquipmentResult result;
    float confidence=80;

    public PPEBoundingBox(DetectProtectiveEquipmentResult ppeResult, BufferedImage
 bufImage, float requiredConfidence) throws Exception {
        super();
        scale = 2; // increase to shrink image size.

        result = ppeResult;
        image = bufImage;

        confidence=requiredConfidence;
    }
    // Draws the bounding box around the detected faces.
    public void paintComponent(Graphics g) {
        float left = 0;
        float top = 0;
        int height = image.getHeight(this);
        int width = image.getWidth(this);
        int offset=20;

        Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.

        // Draw the image.
        g2d.drawImage(image, 0, 0, width / scale, height / scale, this);
        g2d.setColor(new Color(0, 212, 0));

        // Iterate through detected persons and display bounding boxes.
        List<ProtectiveEquipmentPerson> persons = result.getPersons();

        for (ProtectiveEquipmentPerson person: persons) {
            BoundingBox boxPerson = person.getBoundingBox();
            left = width * boxPerson.getLeft();
            top = height * boxPerson.getTop();
            Boolean foundMask=false;

            List<ProtectiveEquipmentBodyPart> bodyParts=person.getBodyParts();

            if (bodyParts.isEmpty()==false)
```

```
            {
                   //body parts detected

                   for (ProtectiveEquipmentBodyPart bodyPart: bodyParts) {

                       List<EquipmentDetection>
equipmentDetections=bodyPart.getEquipmentDetections();

                       for (EquipmentDetection item: equipmentDetections) {

                           if (item.getType().contentEquals("FACE_COVER"))
                           {
                               // Draw green or red bounding box depending on mask
coverage.
                               foundMask=true;
                               BoundingBox box =item.getBoundingBox();
                               left = width * box.getLeft();
                               top = height * box.getTop();
                               Color maskColor=new Color( 0, 212, 0);

                               if (item.getCoversBodyPart().getValue()==false) {
                                   // red bounding box
                                   maskColor=new Color( 255, 0, 0);
                               }
                               g2d.setColor(maskColor);
                               g2d.drawRect(Math.round(left / scale),
Math.round(top / scale),
                                       Math.round((width * box.getWidth()) /
scale), Math.round((height * box.getHeight())) / scale);

                               // Check confidence is > supplied confidence.
                               if (item.getCoversBodyPart().getConfidence()<
confidence)
                               {
                                   // Draw a yellow bounding box inside face mask
bounding box
                                   maskColor=new Color( 255, 255, 0);
                                   g2d.setColor(maskColor);
                                   g2d.drawRect(Math.round((left + offset) /
scale),
                                            Math.round((top + offset) / scale),
                                            Math.round((width * box.getWidth())-
(offset * 2 ))/ scale,
                                            Math.round((height * box.getHeight())
-( offset* 2)) / scale);
                               }

                           }
                       }

                   }

               }

           // Didn't find a mask, so draw person bounding box red
           if (foundMask==false) {

               left = width * boxPerson.getLeft();
               top = height * boxPerson.getTop();
               g2d.setColor(new Color(255, 0, 0));
               g2d.drawRect(Math.round(left / scale), Math.round(top / scale),
                       Math.round(((width) * boxPerson.getWidth()) / scale),
Math.round((height * boxPerson.getHeight())) / scale);
           }
        }
```

```
    }


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

        String photo = "photo";

        float confidence =80;


        int height = 0;
        int width = 0;

        BufferedImage image = null;
        ByteBuffer imageBytes;

        // Get image bytes for call to DetectProtectiveEquipment
        try (InputStream inputStream = new FileInputStream(new File(photo))) {
            imageBytes = ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
        }

        //Get image for display
        InputStream imageBytesStream;
        imageBytesStream = new ByteArrayInputStream(imageBytes.array());

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

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


        // Call DetectProtectiveEquipment
        DetectProtectiveEquipmentRequest request = new
 DetectProtectiveEquipmentRequest()
                .withImage(new Image()
                        .withBytes(imageBytes));

        DetectProtectiveEquipmentResult result =
 rekognitionClient.detectProtectiveEquipment(request);


        // Create frame and panel.
        JFrame frame = new JFrame("Detect PPE");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        PPEBoundingBox panel = new PPEBoundingBox(result, image, confidence);
        panel.setPreferredSize(new Dimension(image.getWidth() / scale,
 image.getHeight() / scale));
        frame.setContentPane(panel);
        frame.pack();
        frame.setVisible(true);

    }
}
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
  public static void displayGear(S3Client s3,
                                    RekognitionClient rekClient,
                                    String sourceImage,
                                    String bucketName) {
      float confidence = 80;
      byte[] data = getObjectBytes(s3, bucketName, sourceImage);
      InputStream is = new ByteArrayInputStream(data);

      try {
          ProtectiveEquipmentSummarizationAttributes summarizationAttributes =
ProtectiveEquipmentSummarizationAttributes.builder()
              .minConfidence(70F)
              .requiredEquipmentTypesWithStrings("FACE_COVER")
              .build();

          SdkBytes sourceBytes = SdkBytes.fromInputStream(is);
          image = ImageIO.read(sourceBytes.asInputStream());

          // Create an Image object for the source image.
          software.amazon.awssdk.services.rekognition.model.Image souImage =
Image.builder()
              .bytes(sourceBytes)
              .build();

          DetectProtectiveEquipmentRequest request =
DetectProtectiveEquipmentRequest.builder()
              .image(souImage)
              .summarizationAttributes(summarizationAttributes)
              .build();

          DetectProtectiveEquipmentResponse result =
rekClient.detectProtectiveEquipment(request);
          JFrame frame = new JFrame("Detect PPE");
          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          PPEBoundingBoxFrame panel = new PPEBoundingBoxFrame(result, image,
confidence);
          panel.setPreferredSize(new Dimension(image.getWidth() / scale,
image.getHeight() / scale));
          frame.setContentPane(panel);
          frame.pack();
          frame.setVisible(true);

      } catch (RekognitionException e) {
          e.printStackTrace();
          System.exit(1);
      } catch (Exception e) {
          e.printStackTrace();
      }
  }

   public static byte[] getObjectBytes (S3Client s3, String bucketName, String
keyName) {

       try {
           GetObjectRequest objectRequest = GetObjectRequest
               .builder()
               .key(keyName)
               .bucket(bucketName)
               .build();

           ResponseBytes<GetObjectResponse> objectBytes =
s3.getObjectAsBytes(objectRequest);
           return objectBytes.asByteArray();
```

```
        } catch (S3Exception e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return null;
     }

   public PPEBoundingBoxFrame(DetectProtectiveEquipmentResponse ppeResult,
BufferedImage bufImage, float requiredConfidence) {
        super();
        scale = 1; // increase to shrink image size.
        result = ppeResult;
        image = bufImage;
        confidence=requiredConfidence;
   }

   // Draws the bounding box around the detected masks.
   public void paintComponent(Graphics g) {
        float left = 0;
        float top = 0;
        int height = image.getHeight(this);
        int width = image.getWidth(this);
        int offset=20;

        Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.

        // Draw the image.
        g2d.drawImage(image, 0, 0, width / scale, height / scale, this);
        g2d.setColor(new Color(0, 212, 0));

        // Iterate through detected persons and display bounding boxes.
        List<ProtectiveEquipmentPerson> persons = result.persons();
        for (ProtectiveEquipmentPerson person: persons) {

            List<ProtectiveEquipmentBodyPart> bodyParts=person.bodyParts();
            if (!bodyParts.isEmpty()){
                for (ProtectiveEquipmentBodyPart bodyPart: bodyParts) {
                    List<EquipmentDetection>
equipmentDetections=bodyPart.equipmentDetections();
                    for (EquipmentDetection item: equipmentDetections) {

                        String myType = item.type().toString();
                        if (myType.compareTo("FACE_COVER") ==0) {

                            // Draw green bounding box depending on mask coverage.
                            BoundingBox box =item.boundingBox();
                            left = width * box.left();
                            top = height * box.top();
                            Color maskColor=new Color( 0, 212, 0);

                            if (item.coversBodyPart().equals(false)) {
                                // red bounding box.
                                maskColor=new Color( 255, 0, 0);
                            }
                            g2d.setColor(maskColor);
                            g2d.drawRect(Math.round(left / scale), Math.round(top /
scale),
                                        Math.round((width * box.width()) / scale),
Math.round((height * box.height())) / scale);

                            // Check confidence is > supplied confidence.
                            if (item.coversBodyPart().confidence() < confidence) {
                                // Draw a yellow bounding box inside face mask
bounding box.
                                maskColor=new Color( 255, 255, 0);
                                g2d.setColor(maskColor);
```

```
                                g2d.drawRect(Math.round((left + offset) / scale),
                                        Math.round((top + offset) / scale),
                                        Math.round((width * box.width())- (offset *
 2 ))/ scale,
                                        Math.round((height * box.height()) -
( offset* 2)) / scale);
                            }
                        }
                    }
                }
            }
        }
    }
```

Python

In the function `main`, change the following:

- The value of `photo` to path and file name of a local image file (PNG or JPEG).
- The value of `confidence` to the desired confidence level (50-100).

```
#Copyright 2020 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, ImageDraw, ExifTags, ImageColor

def detect_ppe(photo, confidence):

    fill_green='#00d400'
    fill_red='#ff0000'
    fill_yellow='#ffff00'
    line_width=3

    #open image and get image data from stream.
    image = Image.open(open(photo,'rb'))
    stream = io.BytesIO()
    image.save(stream, format=image.format)
    image_binary = stream.getvalue()
    imgWidth, imgHeight = image.size
    draw = ImageDraw.Draw(image)

    client=boto3.client('rekognition')

    response = client.detect_protective_equipment(Image={'Bytes': image_binary})

    for person in response['Persons']:

        found_mask=False

        for body_part in person['BodyParts']:
            ppe_items = body_part['EquipmentDetections']

            for ppe_item in ppe_items:
                #found a mask
                if ppe_item['Type'] == 'FACE_COVER':
                    fill_color=fill_green
                    found_mask=True
                    # check if mask covers face
                    if ppe_item['CoversBodyPart']['Value'] == False:
                        fill_color=fill='#ff0000'
```

```
                    # draw bounding box around mask
                    box = ppe_item['BoundingBox']
                    left = imgWidth * box['Left']
                    top = imgHeight * box['Top']
                    width = imgWidth * box['Width']
                    height = imgHeight * box['Height']
                    points = (
                            (left,top),
                            (left + width, top),
                            (left + width, top + height),
                            (left , top + height),
                            (left, top)
                        )
                    draw.line(points, fill=fill_color, width=line_width)

                     # Check if confidence is lower than supplied value
                    if ppe_item['CoversBodyPart']['Confidence'] < confidence:
                        #draw warning yellow bounding box within face mask bounding
 box
                        offset=line_width+ line_width
                        points = (
                                    (left+offset,top + offset),
                                    (left + width-offset, top+offset),
                                    ((left) + (width-offset), (top-offset) +
 (height)),
                                    (left+ offset , (top) + (height -offset)),
                                    (left + offset, top + offset)
                                )
                        draw.line(points, fill=fill_yellow, width=line_width)

        if found_mask==False:
            # no face mask found so draw red bounding box around body
            box = person['BoundingBox']
            left = imgWidth * box['Left']
            top = imgHeight * box['Top']
            width = imgWidth * box['Width']
            height = imgHeight * box['Height']
            points = (
                (left,top),
                (left + width, top),
                (left + width, top + height),
                (left , top + height),
                (left, top)
                )
            draw.line(points, fill=fill_red, width=line_width)

    image.show()

def main():
    photo='photo'
    confidence=80
    detect_ppe(photo, confidence)

if __name__ == "__main__":
    main()
```

Recognizing celebrities

Amazon Rekognition makes it easy for customers to automatically recognize tens of thousands of well-known personalities in images and videos using machine learning. The metadata provided by the celebrity recognition API significantly reduces the repetitive manual effort required to tag content and make it readily searchable.

The rapid proliferation of image and video content means that media companies often struggle to organize, search, and utilize their media catalogs at scale. News channels and sports broadcasters often need to find images and videos quickly, in order to respond to current events and create relevant programming. Insufficient metadata makes these tasks difficult, but with Amazon Rekognition you can automatically tag large volumes of new or archival content to make it easily searchable for a comprehensive set of international, widely known celebrities like actors, sportspeople, and online content creators.

Amazon Rekognition celebrity recognition is designed to be used exclusively 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. 157).

> **Note**
> If you are a celebrity and don't want to be included in this feature, contact AWS Support or email `<rekognition-celebrity-opt-out@amazon.com>`.

Topics

Celebrity recognition compared to face search

Amazon Rekognition offers both celebrity recognition and face recognition functionality. These functionalities have some key differences in their use cases and best practices.

Celebrity recognition comes pre-trained with the ability to recognize hundreds of thousands of popular people in fields such as sports, media, politics, and business. This functionality is designed to help you search large volumes of images or videos in order to identify a small set that is likely to contain a particular celebrity. It's not intended to be used to match faces between different people that are not celebrities. In situations where the accuracy of the celebrity match is important, we recommend also using human operators to look through this smaller amount of marked content to help ensure a high level of accuracy and the proper application of human judgment. Celebrity recognition should not be used in a manner that could result in a negative impact on civil liberties.

In contrast, face recognition is a more general functionality that allows you to create your own face collections with your own face vectors to verify identities or search for any person, not just celebrities. Face recognition can be used for applications such as authenticating building access, public safety, and social media. In all these cases, it's recommended that you use best practices, appropriate confidence thresholds (including 99% for public safety use cases), and human review in situations where the accuracy of the match is important.

For more information, see Searching faces in a collection (p. 157).

Recognizing celebrities in an image

To recognize celebrities within images and get additional information about recognized celebrities, use the RecognizeCelebrities 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 64 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. 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 procedures, you use an image that's loaded from your local file system. For information about input image recommendations, see Working with images (p. 26).

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. 13).

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.");
   }
}
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
   public static void recognizeAllCelebrities(RekognitionClient rekClient, String
 sourceImage) {

        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            Image souImage = Image.builder()
                .bytes(sourceBytes)
                .build();

            RecognizeCelebritiesRequest request =
 RecognizeCelebritiesRequest.builder()
                .image(souImage)
                .build();

            RecognizeCelebritiesResponse result =
 rekClient.recognizeCelebrities(request) ;
            List<Celebrity> celebs=result.celebrityFaces();
            System.out.println(celebs.size() + " celebrity(s) were recognized.\n");
            for (Celebrity celebrity: celebs) {
                System.out.println("Celebrity recognized: " + celebrity.name());
                System.out.println("Celebrity ID: " + celebrity.id());

                System.out.println("Further information (if available):");
                for (String url: celebrity.urls()){
                    System.out.println(url);
                }
                System.out.println();
            }
            System.out.println(result.unrecognizedFaces().size() + " face(s) were
 unrecognized.");

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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

def recognize_celebrities(photo):


    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 ('KnownGender: ' + celebrity['KnownGender'])
        print ('Smile: ' + celebrity['Smile'])
        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
    return len(response['CelebrityFaces'])

def main():
    photo='moviestars.jpg'

    celeb_count=recognize_celebrities(photo)
    print("Celebrities detected: " + str(celeb_count))


if __name__ == "__main__":
    main()
```

Node.Js

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. Change the value of `bucket` to the name of the S3 bucket containing the provided image file. Change the value of `REGION` to the name of the region associated with your account.

```
// Import required AWS SDK clients and commands for Node.js
import { RecognizeCelebritiesCommand } from  "@aws-sdk/client-rekognition";
import  { RekognitionClient } from "@aws-sdk/client-rekognition";

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Create SNS service object.
const rekogClient = new RekognitionClient({ region: REGION });

const bucket = 'bucket-name'
const photo = 'photo-name'

// Set params
const params = {
    Image: {
      S3Object: {
        Bucket: bucket,
        Name: photo
```

```
      },
    },
  }

const recognize_celebrity = async() => {
    try {
        const response = await rekogClient.send(new
 RecognizeCelebritiesCommand(params));
        console.log(response.Labels)
        response.CelebrityFaces.forEach(celebrity =>{
            console.log(`Name: ${celebrity.Name}`)
            console.log(`ID: ${celebrity.Id}`)
            console.log(`KnownGender: ${celebrity.KnownGender.Type}`)
            console.log(`Smile: ${celebrity.Smile}`)
            console.log('Position: ')
            console.log(`   Left: ${celebrity.Face.BoundingBox.Height}`)
            console.log(`  Top : ${celebrity.Face.BoundingBox.Top}`)

        })
        return response.length; // For unit tests.
      } catch (err) {
        console.log("Error", err);
      }
}

recognize_celebrity()
```

.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. 288).

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. 26).

```
{
    "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 object in the array contains the celebrity name and a list of URLs pointing to related content—for example, the celebrity's IMDB or Wikidata link. Amazon Rekognition returns an ComparedFace 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 API operation.

- **Unrecognized faces** – `UnrecognizedFaces` is an array of faces that didn't match any known celebrities. Each ComparedFace object in the array contains a bounding box (as well as other information) that you can use to locate the face in the image.

```
{
    "CelebrityFaces": [{
        "Face": {
            "BoundingBox": {
                "Height": 0.617123007774353,
                "Left": 0.15641026198863983,
                "Top": 0.10864841192960739,
                "Width": 0.3641025722026825
            },
            "Confidence": 99.99589538574219,
            "Emotions": [{
                "Confidence": 96.3981749057023,
                "Type": "Happy"
                }
            ],
            "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
            },
            "Smile": {
            "Confidence": 95.45394855702342,
            "Value": True
            }
        },
        "Id": "3Ir0du6",
        "KnownGender": {
            "Type": "Male"
        },
        "MatchConfidence": 98.0,
        "Name": "Jeff Bezos",
```

```
        "Urls": ["www.imdb.com/name/nm1757263"]
    }],
    "OrientationCorrection": "NULL",
    "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
        },
        "Smile": {
            "Confidence": 95.45394855702342,
            "Value": True
        }
    }]
}
```

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 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. to get the analysis results. For more information about starting video analysis and getting the results, see Calling Amazon Rekognition Video operations (p. 62).

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 StartCelebrityDetection(String bucket, String video)
 throws Exception{

            NotificationChannel channel= new NotificationChannel()
                    .withSNSTopicArn(snsTopicArn)
                    .withRoleArn(roleArn);

           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 GetCelebrityDetectionResults() 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);

      }
```

In the function `main`, replace the line:

```
      StartLabelDetection(bucket, video);

      if (GetSQSMessageSuccess()==true)
       GetLabelDetectionResults();
```

with:

```
      StartCelebrityDetection(bucket, video);

      if (GetSQSMessageSuccess()==true)
       GetCelebrityDetectionResults();
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void StartCelebrityDetection(RekognitionClient rekClient,
                                               NotificationChannel channel,
                                               String bucket,
                                               String video){
        try {
            S3Object s3Obj = S3Object.builder()
                .bucket(bucket)
                .name(video)
                .build();

            Video vidOb = Video.builder()
                .s3Object(s3Obj)
                .build();

            StartCelebrityRecognitionRequest recognitionRequest =
StartCelebrityRecognitionRequest.builder()
                .jobTag("Celebrities")
```

```
                .notificationChannel(channel)
                .video(vidOb)
                .build();

            StartCelebrityRecognitionResponse startCelebrityRecognitionResult =
rekClient.startCelebrityRecognition(recognitionRequest);
            startJobId = startCelebrityRecognitionResult.jobId();

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void GetCelebrityDetectionResults(RekognitionClient rekClient) {

        try {
            String paginationToken=null;
            GetCelebrityRecognitionResponse recognitionResponse = null;
            boolean finished = false;
            String status;
            int yy=0 ;

            do{
                if (recognitionResponse !=null)
                    paginationToken = recognitionResponse.nextToken();

                GetCelebrityRecognitionRequest recognitionRequest =
GetCelebrityRecognitionRequest.builder()
                    .jobId(startJobId)
                    .nextToken(paginationToken)
                    .sortBy(CelebrityRecognitionSortBy.TIMESTAMP)
                    .maxResults(10)
                    .build();

                // Wait until the job succeeds
                while (!finished) {
                    recognitionResponse =
rekClient.getCelebrityRecognition(recognitionRequest);
                    status = recognitionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is null.
                VideoMetadata videoMetaData=recognitionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " + videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<CelebrityRecognition> celebs=
recognitionResponse.celebrities();
                for (CelebrityRecognition celeb: celebs) {
                    long seconds=celeb.timestamp()/1000;
                    System.out.print("Sec: " + seconds + " ");
                    CelebrityDetail details=celeb.celebrity();
```

```
                    System.out.println("Name: " + details.name());
                    System.out.println("Id: " + details.id());
                    System.out.println();
                }

            } while (recognitionResponse.nextToken() != null);

        } catch(RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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.)

    # ============== Celebrities ===============
    def StartCelebrityDetection(self):
        response=self.rek.start_celebrity_recognition(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
            NotificationChannel={'RoleArn': self.roleArn, 'SNSTopicArn':
 self.snsTopicArn})

        self.startJobId=response['JobId']
        print('Start Job Id: ' + self.startJobId)

    def GetCelebrityDetectionResults(self):
        maxResults = 10
        paginationToken = ''
        finished = False

        while finished == False:
            response = self.rek.get_celebrity_recognition(JobId=self.startJobId,
                                                        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
```

In the function `main`, replace the lines:

```
    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()
```

with:

```
    analyzer.StartCelebrityDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetCelebrityDetectionResults()
```

Node.JS

In the following Node.Js code example, replace the value of `bucket` with the name of the S3 bucket containing your video and the value of `videoName` with the name of the video file. You'll also need to replace the value of `roleArn` with the Arn associated with your IAM service role. Finally, replace the value of `region` with the name of the operating region associated with your account.

```
// Import required AWS SDK clients and commands for Node.js
  import { CreateQueueCommand, GetQueueAttributesCommand, GetQueueUrlCommand,
    SetQueueAttributesCommand, DeleteQueueCommand, ReceiveMessageCommand,
 DeleteMessageCommand } from  "@aws-sdk/client-sqs";
  import {CreateTopicCommand, SubscribeCommand, DeleteTopicCommand } from "@aws-
sdk/client-sns";
  import  { SQSClient } from "@aws-sdk/client-sqs";
  import  { SNSClient } from "@aws-sdk/client-sns";
  import  { RekognitionClient, StartLabelDetectionCommand,
 GetLabelDetectionCommand,
    StartCelebrityRecognitionCommand, GetCelebrityRecognitionCommand} from "@aws-
sdk/client-rekognition";
  import { stdout } from "process";

  // Set the AWS Region.
  const region = "region-name"; //e.g. "us-east-1"
  // Create SNS service object.
  const sqsClient = new SQSClient({ region: region });
  const snsClient = new SNSClient({ region: region });
  const rekClient = new RekognitionClient({ region: region });

  // Set bucket and video variables
  const bucket = "bucket-name";
  const videoName = "video-name";
  const roleArn = "RoleArn"
  var startJobId = ""

  var ts = Date.now();
  const snsTopicName = "AmazonRekognitionExample" + ts;
  const snsTopicParams = {Name: snsTopicName}
  const sqsQueueName = "AmazonRekognitionQueue-" + ts;

  // Set the parameters
  const sqsParams = {
    QueueName: sqsQueueName, //SQS_QUEUE_URL
    Attributes: {
      DelaySeconds: "60", // Number of seconds delay.
      MessageRetentionPeriod: "86400", // Number of seconds delay.
    },
  };

  const createTopicandQueue = async () => {
    try {
      // Create SNS topic
      const topicResponse = await snsClient.send(new
 CreateTopicCommand(snsTopicParams));
      const topicArn = topicResponse.TopicArn
      console.log("Success", topicResponse);
      // Create SQS Queue
      const sqsResponse = await sqsClient.send(new CreateQueueCommand(sqsParams));
```

```
     console.log("Success", sqsResponse);
     const sqsQueueCommand = await sqsClient.send(new
GetQueueUrlCommand({QueueName: sqsQueueName}))
     const sqsQueueUrl = sqsQueueCommand.QueueUrl
     const attribsResponse = await sqsClient.send(new
GetQueueAttributesCommand({QueueUrl: sqsQueueUrl, AttributeNames: ['QueueArn']}))
     const attribs = attribsResponse.Attributes
     console.log(attribs)
     const queueArn = attribs.QueueArn
     // subscribe SQS queue to SNS topic
     const subscribed = await snsClient.send(new SubscribeCommand({TopicArn:
topicArn, Protocol:'sqs', Endpoint: queueArn}))
     const policy = {
       Version: "2012-10-17",
       Statement: [
         {
           Sid: "MyPolicy",
           Effect: "Allow",
           Principal: {AWS: "*"},
           Action: "SQS:SendMessage",
           Resource: queueArn,
           Condition: {
             ArnEquals: {
               'aws:SourceArn': topicArn
             }
           }
         }
       ]
     };

     const response = sqsClient.send(new SetQueueAttributesCommand({QueueUrl:
sqsQueueUrl, Attributes: {Policy: JSON.stringify(policy)}}))
     console.log(response)
     console.log(sqsQueueUrl, topicArn)
     return [sqsQueueUrl, topicArn]

   } catch (err) {
     console.log("Error", err);
   }
 };

 const startCelebrityDetection = async(roleArn, snsTopicArn) =>{
   try {
       //Initiate label detection and update value of startJobId with returned Job
ID
       const response = await rekClient.send(new
StartCelebrityRecognitionCommand({Video:{S3Object:{Bucket:bucket,
Name:videoName}},
           NotificationChannel:{RoleArn: roleArn, SNSTopicArn: snsTopicArn}}))
           startJobId = response.JobId
           console.log(`Start Job ID: ${startJobId}`)
           return startJobId
     } catch (err) {
       console.log("Error", err);
     }
   };

 const getCelebrityRecognitionResults = async(startJobId) =>{
   try {
       //Initiate label detection and update value of startJobId with returned Job
ID
       var maxResults = 10
       var paginationToken = ''
       var finished = false

       while (finished == false){
```

```
            var response = await rekClient.send(new 
GetCelebrityRecognitionCommand({JobId: startJobId, MaxResults: maxResults,
                NextToken: paginationToken}))
            console.log(response.VideoMetadata.Codec)
            console.log(response.VideoMetadata.DurationMillis)
            console.log(response.VideoMetadata.Format)
            console.log(response.VideoMetadata.FrameRate)
            response.Celebrities.forEach(celebrityRecognition => {
                console.log(`Celebrity: ${celebrityRecognition.Celebrity.Name}`)
                console.log(`Timestamp: ${celebrityRecognition.Timestamp}`)
                console.log()
            })
            // Searh for pagination token, if found, set variable to next token
            if (String(response).includes("NextToken")){
                paginationToken = response.NextToken

            }else{
                finished = true
            }
        }
      } catch (err) {
        console.log("Error", err);
      }
    };

 // Checks for status of job completion
 const getSQSMessageSuccess = async(sqsQueueUrl, startJobId) => {
    try {
      // Set job found and success status to false initially
      var jobFound = false
      var succeeded = false
      var dotLine = 0
      // while not found, continue to poll for response
      while (jobFound == false){
        var sqsReceivedResponse = await sqsClient.send(new 
ReceiveMessageCommand({QueueUrl:sqsQueueUrl, 
          MaxNumberOfMessages:'ALL', MaxNumberOfMessages:10}));
        if (sqsReceivedResponse){
          var responseString = JSON.stringify(sqsReceivedResponse)
          if (!responseString.includes('Body')){
            if (dotLine < 40) {
              console.log('.')
              dotLine = dotLine + 1
            }else {
              console.log('')
              dotLine = 0
            };
            stdout.write('', () => {
              console.log('');
            });
            await new Promise(resolve => setTimeout(resolve, 5000));
            continue
          }
        }

        // Once job found, log Job ID and return true if status is succeeded
        for (var message of sqsReceivedResponse.Messages){
          console.log("Retrieved messages:")
          var notification = JSON.parse(message.Body)
          var rekMessage = JSON.parse(notification.Message)
          var messageJobId = rekMessage.JobId
          if (String(rekMessage.JobId).includes(String(startJobId))){
            console.log('Matching job found:')
            console.log(rekMessage.JobId)
            jobFound = true
            console.log(rekMessage.Status)
```

```
            if (String(rekMessage.Status).includes(String("SUCCEEDED"))){
              succeeded = true
              console.log("Job processing succeeded.")
              var sqsDeleteMessage = await sqsClient.send(new
DeleteMessageCommand({QueueUrl:sqsQueueUrl,
ReceiptHandle:message.ReceiptHandle}));
            }
          }else{
            console.log("Provided Job ID did not match returned ID.")
            var sqsDeleteMessage = await sqsClient.send(new
DeleteMessageCommand({QueueUrl:sqsQueueUrl,
ReceiptHandle:message.ReceiptHandle}));
          }
        }
      }
    return succeeded
    } catch(err) {
      console.log("Error", err);
    }
 };

 // Start label detection job, sent status notification, check for success status
 // Retrieve results if status is "SUCEEDED", delete notification queue and topic
 const runCelebRecognitionAndGetResults = async () => {
    try {
      const sqsAndTopic = await createTopicandQueue();
      //const startLabelDetectionRes = await startLabelDetection(roleArn,
sqsAndTopic[1]);
      //const getSQSMessageStatus = await getSQSMessageSuccess(sqsAndTopic[0],
startLabelDetectionRes)
      const startCelebrityDetectionRes = await startCelebrityDetection(roleArn,
sqsAndTopic[1]);
      const getSQSMessageStatus = await getSQSMessageSuccess(sqsAndTopic[0],
startCelebrityDetectionRes)
      console.log(getSQSMessageSuccess)
      if (getSQSMessageSuccess){
        console.log("Retrieving results:")
        const results = await
getCelebrityRecognitionResults(startCelebrityDetectionRes)
      }
      const deleteQueue = await sqsClient.send(new DeleteQueueCommand({QueueUrl:
sqsAndTopic[0]}));
      const deleteTopic = await snsClient.send(new DeleteTopicCommand({TopicArn:
sqsAndTopic[1]}));
      console.log("Successfully deleted.")
    } catch (err) {
      console.log("Error", err);
    }
 };

 runCelebRecognitionAndGetResults()
```

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 code to replace might be 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 object exists for each time the celebrity is recognized in the video. Each `CelebrityRecognition` contains information about a recognized celebrity (CelebrityDetail) 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`), the celebrity's known gender(`KnownGender`), and a list of URLs pointing to related content (`Urls`). It also includes the confidence level that Amazon Rekognition Video has in the accuracy of the recognition, and details about the celebrity's face, FaceDetail. If you need to get the related content later, you can use `ID` with getCelebrityInfo.
- **VideoMetadata** – Information about the video that was analyzed.


```
{
    "Celebrities": [
        {
            "Celebrity": {
                "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",
                "KnownGender": {
                    "Type": "Female"
                },
                "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 API operation. The celebrity is identified by using the celebrity ID that's returned from a previous call to RecognizeCelebrities.

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. 271).

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. 13).

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. 271).

```
//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);
      }
   }
}
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void getCelebrityInfo(RekognitionClient rekClient, String id) {

        try {
            GetCelebrityInfoRequest info = GetCelebrityInfoRequest.builder()
                .id(id)
                .build();

            GetCelebrityInfoResponse response = rekClient.getCelebrityInfo(info);
            System.out.println("celebrity name: " + response.name());
            System.out.println("Further information (if available):");
            for (String url: response.urls()){
                System.out.println(url);
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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. 271).

```
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. 271).

```
#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

def get_celebrity_info(id):


    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)

def main():
    id="nnnnnnnn"
    celebrity_info=get_celebrity_info(id)



if __name__ == "__main__":
    main()
```

.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. 271).

```
//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 operation response

`GetCelebrityInfo` returns an array (`Urls`) of links to information about the requested celebrity.

```
{
    "Name": "Celebrity Name",
    "Urls": [
        "www.imdb.com/name/nmnnnnnnn"
    ]
}
```

Moderating content

You can use Amazon Rekognition to detect content that is inappropriate, unwanted, or offensive. You can use Rekognition moderation APIs in social media, broadcast media, advertising, and e-commerce situations to create a safer user experience, provide brand safety assurances to advertisers, and comply with local and global regulations.

Today, many companies rely entirely on human moderators to review third-party or user-generated content, while others simply react to user complaints to take down offensive or inappropriate images, ads, or videos. However, human moderators alone cannot scale to meet these needs at sufficient quality or speed, which leads to a poor user experience, high costs to achieve scale, or even a loss of brand reputation. By using Rekognition for image and video moderation, human moderators can review a much smaller set of content, typically 1-5% of the total volume, already flagged by machine learning. This enables them to focus on more valuable activities and still achieve comprehensive moderation coverage at a fraction of their existing cost. To set up human workforces and perform human review tasks, you can use Amazon Augmented AI, which is already integrated with Rekognition.

Topics

Using the image and video moderation APIs

In the Amazon Rekognition Image API, you can use the DetectModerationLabels operation to detect inappropriate or offensive content in images. You can use the Amazon Rekognition Video API to detect inappropriate content asynchronously by using the StartContentModeration and GetContentModeration operations.

Amazon Rekognition uses a two-level hierarchical taxonomy to label categories of inappropriate or offensive content. 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
	Illustrated Explicit Nudity
	Adult Toys
Suggestive	Female Swimwear Or Underwear
	Male Swimwear Or Underwear
	Partial Nudity

Top-Level Category	Second-Level Category
	Barechested Male
	Revealing Clothes
	Sexual Situations
Violence	Graphic Violence Or Gore
	Physical Violence
	Weapon Violence
	Weapons
	Self Injury
Visually Disturbing	Emaciated Bodies
	Corpses
	Hanging
	Air Crash
	Explosions And Blasts
Rude Gestures	Middle Finger
Drugs	Drug Products
	Drug Use
	Pills
	Drug Paraphernalia
Tobacco	Tobacco Products
	Smoking
Alcohol	Drinking
	Alcoholic Beverages
Gambling	Gambling
Hate Symbols	Nazi Party
	White Supremacy
	Extremist

You determine the suitability of content 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 labels array that's returned by `DetectModerationLabels` (images) and by `GetContentModeration` (videos).

You can set the confidence threshold that Amazon Rekognition uses to detect inappropriate content by specifying the `MinConfidence` input parameter. Labels aren't returned for inappropriate content that is detected with a lower confidence than `MinConfidence`.

Specifying a value for `MinConfidence` that is less than 50% is likely to return a high number of false-positive results. We recommend that 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 inappropriate content that is 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, Amazon Rekognition might return “Explicit Nudity” with a high confidence score as a top-level label. That might be enough for your filtering needs. However, if it's necessary, you can use the confidence score of a second-level label (such as "Graphic Male Nudity") to obtain more granular filtering. For an example, see Detecting inappropriate images (p. 294).

Amazon Rekognition Image and Amazon Rekognition Video both return the version of the moderation detection model that is used to detect inappropriate content (`ModerationModelVersion`).

> **Note**
> Amazon Rekognition isn't an authority on, and doesn't in any way claim to be an exhaustive filter of, inappropriate or offensive content. Additionally, the image and video moderation APIs don't detect whether an image includes illegal content, such as child pornography.

Detecting inappropriate images

You can use the DetectModerationLabels operation to determine if an image contains inappropriate or offensive content. For a list of moderation labels in Amazon Rekognition, see Using the image and video moderation APIs.

Detecting inappropriate content in an image

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 or the appropriate AWS SDK 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 Actions Defined by Amazon Rekognition.

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. 13).

2. Upload an image to your S3 bucket.

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

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

   Java

   This example outputs detected inappropriate content 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();
       }
    }
}
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
   public static void detectModLabels(RekognitionClient rekClient, String
 sourceImage) {
```

```
        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            Image souImage = Image.builder()
                .bytes(sourceBytes)
                .build();

            DetectModerationLabelsRequest moderationLabelsRequest =
 DetectModerationLabelsRequest.builder()
                .image(souImage)
                .minConfidence(60F)
                .build();

            DetectModerationLabelsResponse moderationLabelsResponse =
 rekClient.detectModerationLabels(moderationLabelsRequest);
            List<ModerationLabel> labels =
 moderationLabelsResponse.moderationLabels();
            System.out.println("Detected labels for image");

            for (ModerationLabel label : labels) {
                System.out.println("Label: " + label.name()
                    + "\n Confidence: " + label.confidence().toString() + "%"
                    + "\n Parent:" + label.parentName());
            }

        } catch (RekognitionException | FileNotFoundException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
```

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 inappropriate or offensive content label names, confidence levels, and the parent label for detected inappropriate content labels.

In the function `main`, 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

def moderate_image(photo, 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'])
    return len(response['ModerationLabels'])



def main():
    photo='photo'
    bucket='bucket'
    label_count=moderate_image(photo, bucket)
    print("Labels detected: " + str(label_count))


if __name__ == "__main__":
    main()
```

.NET

This example outputs detected inappropriate or offensive content 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:

- **Inappropriate Image Detection information** – The example shows a list of labels for inappropriate or offensive 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 inappropriate stored videos

Amazon Rekognition Video inappropriate or offensive content detection in stored videos is an asynchronous operation. To start detecting inappropriate or offensive content, call StartContentModeration. Amazon Rekognition Video publishes the completion status of the video analysis to an Amazon Simple Notification Service topic. If the video analysis is successful, call GetContentModeration to get the analysis results. For more information about starting video analysis and getting the results, see Calling Amazon Rekognition Video operations (p. 62). For a list of moderation labels in Amazon Rekognition, see Using the image and video moderation APIs.

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 inappropriate or offensive 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 StartUnsafeContentDetection(String bucket, String
 video) throws Exception{

            NotificationChannel channel= new NotificationChannel()
                    .withSNSTopicArn(snsTopicArn)
                    .withRoleArn(roleArn);

            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 GetUnsafeContentDetectionResults() 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);

        }
```

In the function `main`, replace the lines:

```
        StartLabelDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetLabelDetectionResults();
```

with:

```
        StartUnsafeContentDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetUnsafeContentDetectionResults();
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void startModerationDetection(RekognitionClient rekClient,
                                                NotificationChannel channel,
                                                String bucket,
                                                String video) {

        try {
            S3Object s3Obj = S3Object.builder()
                .bucket(bucket)
                .name(video)
                .build();

            Video vidOb = Video.builder()
                .s3Object(s3Obj)
                .build();

            StartContentModerationRequest modDetectionRequest =
StartContentModerationRequest.builder()
                .jobTag("Moderation")
                .notificationChannel(channel)
                .video(vidOb)
                .build();

            StartContentModerationResponse startModDetectionResult =
rekClient.startContentModeration(modDetectionRequest);
            startJobId=startModDetectionResult.jobId();

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void GetModResults(RekognitionClient rekClient) {

        try {
            String paginationToken=null;
            GetContentModerationResponse modDetectionResponse=null;
            boolean finished = false;
            String status;
            int yy=0 ;

            do{
                if (modDetectionResponse !=null)
                    paginationToken = modDetectionResponse.nextToken();

                GetContentModerationRequest modRequest =
GetContentModerationRequest.builder()
                    .jobId(startJobId)
                    .nextToken(paginationToken)
                    .maxResults(10)
                    .build();

                // Wait until the job succeeds
                while (!finished) {
                    modDetectionResponse =
rekClient.getContentModeration(modRequest);
                    status = modDetectionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
```

```
                    finished = true;
                else {
                    System.out.println(yy + " status is: " + status);
                    Thread.sleep(1000);
                }
                yy++;
            }

            finished = false;

            // Proceed when the job is done - otherwise VideoMetadata is null
            VideoMetadata videoMetaData=modDetectionResponse.videoMetadata();
            System.out.println("Format: " + videoMetaData.format());
            System.out.println("Codec: " + videoMetaData.codec());
            System.out.println("Duration: " + videoMetaData.durationMillis());
            System.out.println("FrameRate: " + videoMetaData.frameRate());
            System.out.println("Job");

            List<ContentModerationDetection> mods =
 modDetectionResponse.moderationLabels();
            for (ContentModerationDetection mod: mods) {
                long seconds=mod.timestamp()/1000;
                System.out.print("Mod label: " + seconds + " ");
                System.out.println(mod.moderationLabel().toString());
                System.out.println();
            }

        } while (modDetectionResponse !=null &&
 modDetectionResponse.nextToken() != null);

    } catch(RekognitionException | InterruptedException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
}
```

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.)

    # ============= Unsafe content ==============
    def StartUnsafeContent(self):
        response=self.rek.start_content_moderation(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
            NotificationChannel={'RoleArn': self.roleArn, 'SNSTopicArn':
 self.snsTopicArn})

        self.startJobId=response['JobId']
        print('Start Job Id: ' + self.startJobId)

    def GetUnsafeContentResults(self):
        maxResults = 10
        paginationToken = ''
        finished = False

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

            print('Codec: ' + response['VideoMetadata']['Codec'])
            print('Duration: ' + str(response['VideoMetadata']['DurationMillis']))
```

```
            print('Format: ' + response['VideoMetadata']['Format'])
            print('Frame rate: ' + str(response['VideoMetadata']['FrameRate']))
            print()

            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
```

In the function `main`, replace the lines:

```
    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()
```

with:

```
    analyzer.StartUnsafeContent()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetUnsafeContentResults()
```

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 code to replace might be different.

3. Run the code. A list of inappropriate content labels detected in the video is shown.

GetContentModeration operation response

The response from `GetContentModeration` is an array, `ModerationLabels`, of ContentModerationDetection objects. The array contains an element for each time an inappropriate content label is detected. Within a `ContentModerationDetectionObject` object, ModerationLabel contains information for a detected item of inappropriate or offensive 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 labels detected by inappropriate content image analysis. For more information, see Moderating content (p. 292).

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
    }
}
```


Reviewing inappropriate content with Amazon Augmented AI

Amazon Augmented AI (Amazon A2I) enables you to build the workflows that are required for human review of machine learning predictions.

Amazon Rekognition is directly integrated with Amazon A2I so that you can easily implement human review for the use case of detecting unsafe images. Amazon A2I provides a human review workflow for image moderation. This enables you to easily review predictions from Amazon Rekognition. You can define confidence thresholds for your use case and adjust them over time. With Amazon A2I, you can use

a pool of reviewers within your own organization or Amazon Mechanical Turk. You can also use workforce vendors that are prescreened by AWS for quality and adherence to security procedures.

The following steps walk you through how to set up Amazon A2I with Amazon Rekognition. First, you create a flow definition with Amazon A2I that has the conditions that trigger human review. Then, you pass the flow definition's Amazon Resource Name (ARN) to the Amazon Rekognition `DetectModerationLabel` operation. In the `DetectModerationLabel` response, you can see if human review is required. The results of human review are available in an Amazon S3 bucket that is set by the flow definition.

To view an end-to-end demonstration of how to use Amazon A2I with Amazon Rekognition, see one of the following tutorials in the *Amazon SageMaker Developer Guide*.

- Demo: Get Started in the Amazon A2I Console
- Demo: Get Started Using the Amazon A2I API

  To get started using the API, you can also run an example Jupyter notebook. See Use a SageMaker Notebook Instance with Amazon A2I Jupyter Notebook to use the notebook Amazon Augmented AI (Amazon A2I) integration with Amazon Rekognition [Example] in a SageMaker notebook instance.

Running DetectModerationLabels with Amazon A2I

> **Note**
> Create all of your Amazon A2I resources and Amazon Rekognition resources in the same AWS Region.

1. Complete the prerequisites that are listed in Getting Started with Amazon Augmented AI in the *SageMaker Documentation*.

   Additionally, remember to set up your IAM permissions as in the page Permissions and Security in Amazon Augmented AI in the *SageMaker Documentation*.
2. Follow the instructions for Creating a Human Review Workflow in the *SageMaker Documentation*.

   A human review workflow manages the processing of an image. It holds the conditions that trigger a human review, the work team that the image is sent to, the UI template that the work team uses, and the Amazon S3 bucket that the work team's results are sent to.

   Within your `CreateFlowDefinition` call, you need to set the `HumanLoopRequestSource` to "AWS/Rekognition/DetectModerationLabels/Image/V3". After that, you need to decide how you want to set up your conditions that trigger human review.

   With Amazon Rekognition you have two options for `ConditionType`: `ModerationLabelConfidenceCheck`, and `Sampling`.

   `ModerationLabelConfidenceCheck` creates a human loop when confidence of a moderation label is within a range. Finally, `Sampling` sends a random percent of the documents processed for human review. Each `ConditionType` uses a different set of `ConditionParameters` to set what results in human review.

   `ModerationLabelCondifenceCheck` has the `ConditionParameters` `ModerationLableName` which sets the key that needs to be reviewed by humans. Additionally, it has confidence, which set the percentage range for sending to human review with LessThan, GreaterThan, and Equals. `Sampling` has `RandomSamplingPercentage` which sets a percent of documents that will be sent to human review.

   The following code example is a partial call of `CreateFlowDefinition`. It sends an image for human review if it's rated less than 98% on the label "Suggestive", and more than 95% on the label "Female Swimwear or Underwear". This means that if the image isn't considered suggestive but does have a woman in underwear or swimwear, you can double check the image by using human review.

```
def create_flow_definition():
'''
Creates a Flow Definition resource

Returns:
struct: FlowDefinitionArn
'''
humanLoopActivationConditions = json.dumps(
    {
        "Conditions": [
            {
              "And": [
                {
                    "ConditionType": "ModerationLabelConfidenceCheck",
                    "ConditionParameters": {
                        "ModerationLabelName": "Suggestive",
                        "ConfidenceLessThan": 98
                    }
                },
                {
                    "ConditionType": "ModerationLabelConfidenceCheck",
                    "ConditionParameters": {
                        "ModerationLabelName": "Female Swimwear Or Underwear",
                        "ConfidenceGreaterThan": 95
                    }
                }
              ]
           }
        ]
    }
)
```

`CreateFlowDefinition` returns a `FlowDefinitionArn`, which you use in the next step when you call `DetectModerationLabels`.

For more information see CreateFlowDefinition in the *SageMaker API Reference*.

3. Set the `HumanLoopConfig` parameter when you call `DetectModerationLabels`, as in Detecting inappropriate images (p. 294). See step 4 for examples of a `DetectModerationLabels` call with `HumanLoopConfig` set.

   a. Within the `HumanLoopConfig` parameter, set the `FlowDefinitionArn` to the ARN of the flow definition that you created in step 2.
   b. Set your `HumanLoopName`. This should be unique within a Region and must be lowercase.
   c. (Optional) You can use `DataAttributes` to set whether or not the image you passed to Amazon Rekognition is free of personally identifiable information. You must set this parameter in order to send information to Amazon Mechanical Turk.

4. Run `DetectModerationLabels`.

   The following examples show how to use the AWS CLI and AWS SDK for Python (Boto3) to run `DetectModerationLabels` with `HumanLoopConfig` set.

   AWS SDK for Python (Boto3)

   The following request example uses the SDK for Python (Boto3). For more information, see detect_moderation_labels in the *AWS SDK for Python (Boto) API Reference*.

```
import boto3

rekognition = boto3.client("rekognition", aws-region)

response = rekognition.detect_moderation_labels( \
        Image={'S3Object': {'Bucket': bucket_name, 'Name': image_name}}, \
        HumanLoopConfig={ \
            'HumanLoopName': 'human_loop_name', \
            'FlowDefinitionArn': , "arn:aws:sagemaker:aws-
region:aws_account_number:flow-definition/flow_def_name" \
            'DataAttributes': {'ContentClassifiers':
 ['FreeOfPersonallyIdentifiableInformation','FreeOfAdultContent']}
         })
```

AWS CLI

The following request example uses the AWS CLI. For more information, see detect-moderation-labels in the *AWS CLI Command Reference*.

```
$ aws rekognition detect-moderation-labels \
    --image "S3Object={Bucket='bucket_name',Name='image_name'}" \
    --human-loop-config
 HumanLoopName="human_loop_name",FlowDefinitionArn="arn:aws:sagemaker:aws-
region:aws_account_number:flow-
definition/
flow_def_name",DataAttributes='{ContentClassifiers=["FreeOfPersonallyIdentifiableInformation",
 "FreeOfAdultContent"]}'
```

```
$ aws rekognition detect-moderation-labels \
    --image "S3Object={Bucket='bucket_name',Name='image_name'}" \
    --human-loop-config \
        '{"HumanLoopName": "human_loop_name", "FlowDefinitionArn":
 "arn:aws:sagemaker:aws-region:aws_account_number:flow-
definition/flow_def_name", "DataAttributes": {"ContentClassifiers":
 ["FreeOfPersonallyIdentifiableInformation", "FreeOfAdultContent"]}}'
```

When you run `DetectModerationLabels` with `HumanLoopConfig` enabled, Amazon Rekognition calls the SageMaker API operation `StartHumanLoop`. This command takes the response from `DetectModerationLabels` and checks it against the flow definition's conditions in the example. If it meets the conditions for review, it returns a `HumanLoopArn`. This means that the members of the work team that you set in your flow definition now can review the image. Calling the Amazon Augmented AI runtime operation `DescribeHumanLoop` provides information about the outcome of the loop. For more information, see DescribeHumanLoop in the *Amazon Augmented AI API Reference documentation*.

After the image has been reviewed, you can see the results in the bucket that is specified in your flow definition's output path. Amazon A2I will also notify you with Amazon CloudWatch Events when the review is complete. To see what events to look for, see CloudWatch Events in the *SageMaker Documentation*.

For more information, see Getting Started with Amazon Augmented AI in the *SageMaker Documentation*.

Detecting text

Amazon Rekognition can detect text in images and videos. It can then convert the detected text into machine-readable text. You can use machine-readable text detection in images to implement solutions such as:

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

For text detection in videos, you can implement solutions such as:

- Searching videos for clips with specific text keywords, such as a guest's name on a graphic in a news show.
- Moderating content for compliance with organizational standards by detecting accidental text, profanity, or spam.
- Finding all text overlays on the video timeline for further processing, such as replacing text with text in another language for content internationalization.
- Finding text locations, so that other graphics can be aligned accordingly.

To detect text in images in JPEG or PNG format, use the DetectText operation. To asynchronously detect text in video, use the StartTextDetection and GetTextDetection operations. Both image and video text detection operations support most fonts, including highly stylized ones. After detecting text, Amazon Rekognition creates a representation of detected words and lines of text, shows the relationship between them, and tells you where the text is on an image or video frame.

The `DetectText` and `GetTextDetection` operations detect words and lines. A *word* is one or more script characters that aren't separated by spaces. `DetectText` can detect up to 100 words in an image. `GetTextDetection` can also detect up to 100 words per frame of video.

A word is one or more script characters that are not separated by spaces. Amazon Rekognition is designed to detect words in English, Arabic, Russian, German, French, Italian, Portuguese and Spanish.

A *line* is a string of equally spaced words. A line isn't necessarily a complete sentence (periods don't indicate the end of a line). For example, Amazon Rekognition detects a driver's license number as a line. A line ends when there is no aligned text after it or when there's a large gap between words, relative to the length of the words. Depending on the gap between words, Amazon Rekognition might detect multiple lines in text that are aligned in the same direction. If a sentence spans multiple lines, the operation returns multiple lines.

Consider the following image.

The blue boxes represent information about the detected text and the location of the text that's returned by the `DetectText` operation. In this example, Amazon Rekognition detects "IT'S", "MONDAY", "but", "keep", and "Smiling" as words. Amazon Rekognition detects "IT'S", "MONDAY", "but keep", and "Smiling" as lines. To be detected, text must be within +/- 90 degrees orientation of the horizontal axis.

For an example, see Detecting text in an image (p. 309).

Topics

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, complete the following prerequisites.
   a. Create or update an AWS Identity and Access Management (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 Command Line Interface and the AWS SDKs. For more information, see Step 2: Set up the AWS CLI and AWS SDKs (p. 13).
2. Upload the image that contains text to your S3 bucket.

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

   Java

   The following example code displays lines and words that were detected in an image.

   Replace the values of `bucket` and `photo` with the names of the 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();
      }
   }
}
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void detectTextLabels(RekognitionClient rekClient, String
 sourceImage) {

        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            Image souImage = Image.builder()
                .bytes(sourceBytes)
                .build();

            DetectTextRequest textRequest = DetectTextRequest.builder()
```

```
                .image(souImage)
                .build();

            DetectTextResponse textResponse = rekClient.detectText(textRequest);
            List<TextDetection> textCollection = textResponse.textDetections();
            System.out.println("Detected lines and words");
            for (TextDetection text: textCollection) {
                System.out.println("Detected: " + text.detectedText());
                System.out.println("Confidence: " + text.confidence().toString());
                System.out.println("Id : " + text.id());
                System.out.println("Parent Id: " + text.parentId());
                System.out.println("Type: " + text.type());
                System.out.println();
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

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 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 lines and words detected in an image.

Replace the values of `bucket` and `photo` with the names of the S3bucket 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

def detect_text(photo, bucket):

    client=boto3.client('rekognition')

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

    textDetections=response['TextDetections']
    print ('Detected text\n----------')
    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()
    return len(textDetections)
```

```
def main():

    bucket='bucket'
    photo='photo'
    text_count=detect_text(photo,bucket)
    print("Text detected: " + str(text_count))


if __name__ == "__main__":
    main()
```

.NET

The following example code displays lines and words detected in an image.

Replace the values of `bucket` and `photo` with the names of the 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);
```

```
        }
    }
}
```

Node.JS

The following example code displays lines and words detected in an image.

Replace the values of `bucket` and `photo` with the names of the S3bucket and image that you used in step 2. Replace the value of `region` with the region found in your .aws credentials.

```
var AWS = require('aws-sdk');

const bucket = 'bucket' // the bucketname without s3://
const photo  = 'photo' // the name of file

const config = new AWS.Config({
  accessKeyId: process.env.AWS_ACCESS_KEY_ID,
  secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
})
AWS.config.update({region:'region'});
const client = new AWS.Rekognition();
const params = {
  Image: {
    S3Object: {
      Bucket: bucket,
      Name: photo
    },
  },
}
client.detectText(params, function(err, response) {
  if (err) {
    console.log(err, err.stack); // handle error if an error occurred
  } else {
    console.log(`Detected Text for: ${photo}`)
    console.log(response)
    response.TextDetections.forEach(label => {
      console.log(`Detected Text: ${label.DetectedText}`),
      console.log(`Type: ${label.Type}`),
      console.log(`ID: ${label.Id}`),
      console.log(`Parent ID: ${label.ParentId}`),
      console.log(`Confidence: ${label.Confidence}`),
      console.log(`Polygon: `)
      console.log(label.Geometry.Polygon)
    }
    )
  }
});
```

DetectText operation request

In the `DetectText` operation, you supply an input image either as a base64-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"
        }
```

```
    }
}
```

Filters

Filtering by text region, size and confidence score provides you with additional flexibility to control your text detection output. By using regions of interest, you can easily limit text detection to the regions that are relevant to you, for example, the top right of profile photo or a fixed location in relation to a reference point when reading parts numbers from an image of a machine. Word bounding box size filter can be used to avoid small background text which may be noisy or irrelevant. And lastly, word confidence filter enables you to remove results that may be unreliable due to being blurry or smudged.. You can use the following filters:

- **MinConfidence** –Sets the confidence level of word detection. Words with detection confidence below this level are excluded from the result. Values should be between 0 and 100. The default MinConfidence is 0.
- **MinBoundingBoxWidth** – Sets the minimum width of the word bounding box. Words with bounding boxes that are smaller than this value are excluded from the result. The value is relative to the image frame width.
- **MinBoundingBoxHeight** – Sets the minimum height of the word bounding box. Words with bounding box heights less than this value are excluded from the result. The value is relative to the image frame height.
- **RegionsOfInterest** – Limits detection to a specific region of the image frame. The values are relative to the frame's dimensions. For text only partially within a region, the response is undefined.

DetectText operation response

The `DetectText` operation analyzes the image and returns an array, TextDetections, where each element (`TextDetection`) 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. A word is one or more script characters not separated by spaces. `DetectText` can detect up to 100 words in an image. 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 example 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) 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 object
- A finer-grained polygon that's made up of multiple X and Y coordinates in a Point 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.

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"
        }
    ]
}
```


Detecting text in a stored video

Amazon Rekognition Video text detection in stored videos is an asynchronous operation. To start detecting text, call StartTextDetection. Amazon Rekognition Video publishes the completion status of the video analysis to an Amazon SNS topic. If the video analysis is successful, call GetTextDetection to get the analysis results. For more information about starting video analysis and getting the results, see Calling Amazon Rekognition Video operations (p. 62).

This procedure expands on the code in Analyzing a video stored in an Amazon S3 bucket with Java or Python (SDK) (p. 68). It uses an Amazon SQS queue to get the completion status of a video analysis request.

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

1. Perform the steps in 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` 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 StartTextDetection(String bucket, String video) throws
 Exception{

    NotificationChannel channel= new NotificationChannel()
            .withSNSTopicArn(snsTopicArn)
            .withRoleArn(roleArn);

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


    StartTextDetectionResult startTextDetectionResult =
 rek.startTextDetection(req);
    startJobId=startTextDetectionResult.getJobId();

}

private static void GetTextDetectionResults() throws Exception{
```

```
    int maxResults=10;
    String paginationToken=null;
    GetTextDetectionResult textDetectionResult=null;

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

        }


        textDetectionResult = rek.getTextDetection(new GetTextDetectionRequest()
             .withJobId(startJobId)
             .withNextToken(paginationToken)
             .withMaxResults(maxResults));

        VideoMetadata videoMetaData=textDetectionResult.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 text, confidence values
        List<TextDetectionResult> textDetections =
 textDetectionResult.getTextDetections();


        for (TextDetectionResult text: textDetections) {
            long seconds=text.getTimestamp()/1000;
            System.out.println("Sec: " + Long.toString(seconds) + " ");
            TextDetection detectedText=text.getTextDetection();

            System.out.println("Text Detected: " +
 detectedText.getDetectedText());
                System.out.println("Confidence: " +
 detectedText.getConfidence().toString());
                System.out.println("Id : " + detectedText.getId());
                System.out.println("Parent Id: " + detectedText.getParentId());
                System.out.println("Bounding Box" +
 detectedText.getGeometry().getBoundingBox().toString());
                System.out.println("Type: " + detectedText.getType());
                System.out.println();
        }
    } while (textDetectionResult !=null && textDetectionResult.getNextToken() !=
 null);


}
```

In the function `main`, replace the lines:

```
        StartLabelDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetLabelDetectionResults();
```

with:

```
        StartTextDetection(bucket, video);
```

```
        if (GetSQSMessageSuccess()==true)
         GetTextDetectionResults();
```

Java V2

This code is taken from the AWS Documentation SDK examples GitHub repository. See the full example here.

```
    public static void startTextLabels(RekognitionClient rekClient,
                                    NotificationChannel channel,
                                    String bucket,
                                    String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                .bucket(bucket)
                .name(video)
                .build();

            Video vidOb = Video.builder()
                .s3Object(s3Obj)
                .build();

            StartTextDetectionRequest labelDetectionRequest =
StartTextDetectionRequest.builder()
                .jobTag("DetectingLabels")
                .notificationChannel(channel)
                .video(vidOb)
                .build();

            StartTextDetectionResponse labelDetectionResponse =
rekClient.startTextDetection(labelDetectionRequest);
            startJobId = labelDetectionResponse.jobId();

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void GetTextResults(RekognitionClient rekClient) {

        try {
            String paginationToken=null;
            GetTextDetectionResponse textDetectionResponse=null;
            boolean finished = false;
            String status;
            int yy=0 ;

            do{
                if (textDetectionResponse !=null)
                    paginationToken = textDetectionResponse.nextToken();

                GetTextDetectionRequest recognitionRequest =
GetTextDetectionRequest.builder()
                    .jobId(startJobId)
                    .nextToken(paginationToken)
                    .maxResults(10)
                    .build();

                // Wait until the job succeeds.
                while (!finished) {
                    textDetectionResponse =
rekClient.getTextDetection(recognitionRequest);
                    status = textDetectionResponse.jobStatusAsString();
```

```
                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is null.
                VideoMetadata videoMetaData=textDetectionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " + videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<TextDetectionResult> labels=
 textDetectionResponse.textDetections();
                for (TextDetectionResult detectedText: labels) {
                    System.out.println("Confidence: " +
 detectedText.textDetection().confidence().toString());
                    System.out.println("Id : " +
 detectedText.textDetection().id());
                    System.out.println("Parent Id: " +
 detectedText.textDetection().parentId());
                    System.out.println("Type: " +
 detectedText.textDetection().type());
                    System.out.println("Text: " +
 detectedText.textDetection().detectedText());
                    System.out.println();
                }

            } while (textDetectionResponse !=null &&
 textDetectionResponse.nextToken() != null);

        } catch(RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

Python

```
#Copyright 2019 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 StartTextDetection(self):
        response=self.rek.start_text_detection(Video={'S3Object': {'Bucket':
 self.bucket, 'Name': self.video}},
            NotificationChannel={'RoleArn': self.roleArn, 'SNSTopicArn':
 self.snsTopicArn})

        self.startJobId=response['JobId']
        print('Start Job Id: ' + self.startJobId)

    def GetTextDetectionResults(self):
        maxResults = 10
        paginationToken = ''
        finished = False
```

```
        while finished == False:
            response = self.rek.get_text_detection(JobId=self.startJobId,
                                                MaxResults=maxResults,
                                                NextToken=paginationToken)

            print('Codec: ' + response['VideoMetadata']['Codec'])

            print('Duration: ' + str(response['VideoMetadata']['DurationMillis']))
            print('Format: ' + response['VideoMetadata']['Format'])
            print('Frame rate: ' + str(response['VideoMetadata']['FrameRate']))
            print()

            for textDetection in response['TextDetections']:
                text=textDetection['TextDetection']

                print("Timestamp: " + str(textDetection['Timestamp']))
                print("   Text Detected: " + text['DetectedText'])
                print("   Confidence: " +  str(text['Confidence']))
                print ("      Bounding box")
                print ("        Top: " + str(text['Geometry']['BoundingBox']
['Top']))
                print ("        Left: " + str(text['Geometry']['BoundingBox']
['Left']))
                print ("        Width: " +  str(text['Geometry']['BoundingBox']
['Width']))
                print ("        Height: " +  str(text['Geometry']['BoundingBox']
['Height']))
                print ("   Type: " + str(text['Type']) )
                print()

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

In the function `main`, replace the lines:

```
    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()
```

with:

```
    analyzer.StartTextDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetTextDetectionResults()
```

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 code to replace might be different.

3. Run the code. Text that was detected in the video is shown in a list.

Filters

Filters are optional request parameters that can be used when you call `StartTextDetection`. Filtering by text region, size and confidence score provides you with additional flexibility to control your text detection output. By using regions of interest, you an easily limit text detection to the regions that are

relevant, for example, a bottom third region for graphics or a top left corner for reading scoreboards in a soccer game. Word bounding box size filter can be used to avoid small background text which may be noisy or irrelevant. And lastly, word confidence filter enables you to remove results that may be unreliable due to being blurry or smudged. You can use the following filters:

- **MinConfidence** –Sets the confidence level of word detection. Words with detection confidence below this level are excluded from the result. Values should be between 0 and 100. The default MinConfidence is 80.
- **MinBoundingBoxWidth** – Sets the minimum width of the word bounding box. Words with bounding boxes that are smaller than this value are excluded from the result. The value is relative to the video frame width.
- **MinBoundingBoxHeight** – Sets the minimum height of the word bounding box. Words with bounding box heights less than this value are excluded from the result. The value is relative to the video frame height.
- **RegionsOfInterest** – Limits detection to a specific region of the frame. The values are relative to the frame dimensions. For objects only partially within the regions, the response is undefined.

GetTextDetection response

`GetTextDetection` returns an array (`TextDetectionResults`) that contains information about the text detected in the video. An array element, TextDetection, exists for each time a word or line is detected in the video. The array elements are sorted by time (in milliseconds) since the start of the video.

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

- **Text information** – The `TextDetectionResult` array element contains information about the detected text (TextDetection) and the time that the text was detected in the video (`Timestamp`).
- **Paging information** – The example shows one page of text detection information. You can specify how many text elements to return in the `MaxResults` input parameter for `GetTextDetection`. If more results than `MaxResults` exist, or there are more results than the default maximum, `GetTextDetection` 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. 64).
- **Video information** – The response includes information about the video format (`VideoMetadata`) in each page of information that's returned by `GetTextDetection`.


```
{
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 174441,
        "Format": "QuickTime / MOV",
        "FrameRate": 29.970029830932617,
        "FrameHeight": 480,
        "FrameWidth": 854
    },
    "TextDetections": [
        {
            "Timestamp": 967,
            "TextDetection": {
                "DetectedText": "Twinkle Twinkle Little Star",
                "Type": "LINE",
                "Id": 0,
                "Confidence": 99.91780090332031,
                "Geometry": {
                    "BoundingBox": {
```

```
                        "Width": 0.8337579369544983,
                        "Height": 0.08365312218666077,
                        "Left": 0.08313830941915512,
                        "Top": 0.4663468301296234
                    },
                    "Polygon": [
                        {
                            "X": 0.08313830941915512,
                            "Y": 0.4663468301296234
                        },
                        {
                            "X": 0.9168962240219116,
                            "Y": 0.4674469828605652
                        },
                        {
                            "X": 0.916861355304718,
                            "Y": 0.5511001348495483
                        },
                        {
                            "X": 0.08310343325138092,
                            "Y": 0.5499999523162842
                        }
                    ]
                }
            }
        },
        {
            "Timestamp": 967,
            "TextDetection": {
                "DetectedText": "Twinkle",
                "Type": "WORD",
                "Id": 1,
                "ParentId": 0,
                "Confidence": 99.98338317871094,
                "Geometry": {
                    "BoundingBox": {
                        "Width": 0.2423887550830841,
                        "Height": 0.0833333358168602,
                        "Left": 0.08313817530870438,
                        "Top": 0.46666666865348816
                    },
                    "Polygon": [
                        {
                            "X": 0.08313817530870438,
                            "Y": 0.46666666865348816
                        },
                        {
                            "X": 0.3255269229412079,
                            "Y": 0.46666666865348816
                        },
                        {
                            "X": 0.3255269229412079,
                            "Y": 0.550000011920929
                        },
                        {
                            "X": 0.08313817530870438,
                            "Y": 0.550000011920929
                        }
                    ]
                }
            }
        },
        {
            "Timestamp": 967,
            "TextDetection": {
                "DetectedText": "Twinkle",
```

```
                "Type": "WORD",
                "Id": 2,
                "ParentId": 0,
                "Confidence": 99.982666015625,
                "Geometry": {
                    "BoundingBox": {
                        "Width": 0.2423887550830841,
                        "Height": 0.08124999701976776,
                        "Left": 0.3454332649707794,
                        "Top": 0.46875
                    },
                    "Polygon": [
                        {
                            "X": 0.3454332649707794,
                            "Y": 0.46875
                        },
                        {
                            "X": 0.5878220200538635,
                            "Y": 0.46875
                        },
                        {
                            "X": 0.5878220200538635,
                            "Y": 0.550000011920929
                        },
                        {
                            "X": 0.3454332649707794,
                            "Y": 0.550000011920929
                        }
                    ]
                }
            }
        },
        {
            "Timestamp": 967,
            "TextDetection": {
                "DetectedText": "Little",
                "Type": "WORD",
                "Id": 3,
                "ParentId": 0,
                "Confidence": 99.8787612915039,
                "Geometry": {
                    "BoundingBox": {
                        "Width": 0.16627635061740875,
                        "Height": 0.08124999701976776,
                        "Left": 0.6053864359855652,
                        "Top": 0.46875
                    },
                    "Polygon": [
                        {
                            "X": 0.6053864359855652,
                            "Y": 0.46875
                        },
                        {
                            "X": 0.7716627717018127,
                            "Y": 0.46875
                        },
                        {
                            "X": 0.7716627717018127,
                            "Y": 0.550000011920929
                        },
                        {
                            "X": 0.6053864359855652,
                            "Y": 0.550000011920929
                        }
                    ]
                }
```

```
            }
        },
        {
            "Timestamp": 967,
            "TextDetection": {
                "DetectedText": "Star",
                "Type": "WORD",
                "Id": 4,
                "ParentId": 0,
                "Confidence": 99.82640075683594,
                "Geometry": {
                    "BoundingBox": {
                        "Width": 0.12997658550739288,
                        "Height": 0.08124999701976776,
                        "Left": 0.7868852615356445,
                        "Top": 0.46875
                    },
                    "Polygon": [
                        {
                            "X": 0.7868852615356445,
                            "Y": 0.46875
                        },
                        {
                            "X": 0.9168618321418762,
                            "Y": 0.46875
                        },
                        {
                            "X": 0.9168618321418762,
                            "Y": 0.550000011920929
                        },
                        {
                            "X": 0.7868852615356445,
                            "Y": 0.550000011920929
                        }
                    ]
                }
            }
        }
    ],
    "NextToken": "NiHpGbZFnkM/S8kLcukMni15wb05iKtquu/Mwc+Qg1LVlMjjKNOD0Z0GusSPg7TONLe
+OZ3P",
    "TextModelVersion": "3.0"
}
```

Detecting video segments in stored video

Amazon Rekognition Video provides an API that identifies useful segments of video, such as black frames and end credits.

Viewers are watching more content than ever. In particular, Over-The-Top (OTT) and Video-On-Demand (VOD) platforms provide a rich selection of content choices anytime, anywhere, and on any screen. With proliferating content volumes, media companies are facing challenges in preparing and managing their content. This is crucial to providing a high-quality viewing experience and better monetizing content. Today, companies use large teams of trained human workforces to perform tasks such as the following.

- Finding where the opening and end credits are in a piece of content
- Choosing the right spots to insert advertisements, such as in silent black frame sequences
- Breaking up videos into smaller clips for better indexing

These manual processes are expensive, slow, and can't scale to keep up with the volume of content that is produced, licensed, and retrieved from archives daily.

You can use Amazon Rekognition Video to automate operational media analysis tasks using fully managed, purpose-built video segment detection APIs powered by machine learning (ML). By using the Amazon Rekognition Video segment APIs, you can easily analyze large volumes of videos and detect markers such as black frames or shot changes. You get SMPTE (Society of Motion Picture and Television Engineers) timecodes, timestamps, and frame numbers from each detection. No ML experience is required.

Amazon Rekognition Video analyzes videos stored in an Amazon Simple Storage Service (Amazon S3) bucket. The SMPTE timecodes that are returned are frame accurate – Amazon Rekognition Video provides the exact frame number of a detected segment of video, and handles various video frame rate formats automatically. You can use the frame accurate metadata from Amazon Rekognition Video to automate certain tasks completely, or to significantly reduce the review workload of trained human operators, so that they can focus on more creative work. You can perform tasks such as preparing content, inserting advertisements, and adding "binge-markers" to content at scale in the cloud.

For information about pricing, see Amazon Rekognition pricing.

Amazon Rekognition Video segment detection supports two types of segmentation tasks — Technical cues (p. 330) detection and Shot detection (p. 332).

Topics

Technical cues

A *technical cue* identifies black frames, color bars, opening credits, end credits, studio logos, and primary program content in a video.

Black frames

Videos often contain empty black frames with no audio that are used as cues to insert advertisements, or to mark the end of a program segment, such as a scene or opening credits. With Amazon Rekognition Video, you can detect black frame sequences to automate ad insertion, package content for VOD, and demarcate various program segments or scenes. Black frames with audio (such as fade outs or voiceovers) are considered as content and not returned.

Credits

Amazon Rekognition Video can automatically identify the exact frames where the opening and closing credits start and end for a movie or TV show. With this information, you can generate "binge markers" or interactive viewer prompts, such as "Next Episode'"or "Skip Intro," in video on demand (VOD) applications. You can also detect the first and last frame of program content in a video. Amazon Rekognition Video is trained to handle a wide variety of opening and end credit styles ranging from simple rolling credits to more challenging credits alongside content.

Color bars

Amazon Rekognition Video allows you to detect sections of video that display SMPTE color bars, which are a set of colors displayed in specific patterns to ensure color is calibrated correctly on broadcast monitors, programs, and on cameras. For more information about SMPTE color bars, see SMPTE color bar. This metadata is useful to prepare content for VOD applications by removing color bar segments from the content, or to detect issues such as loss of broadcast signals in a recording, when color bars are shown continuously as a default signal instead of content.

Slates

Slates are sections of the video, typically near the beginning, that contain text metadata about the episode, studio, video format, audio channels, and more. Amazon Rekognition Video can identify the start and end of slates, making it easy to use the text metadata or remove the slate when preparing content for final viewing.

Studio logos

Studio logos are sequences that show the logos or emblems of the production studio involved in making the show. Amazon Rekognition Video can detect these sequences so that users can review them to identify studios.

Content

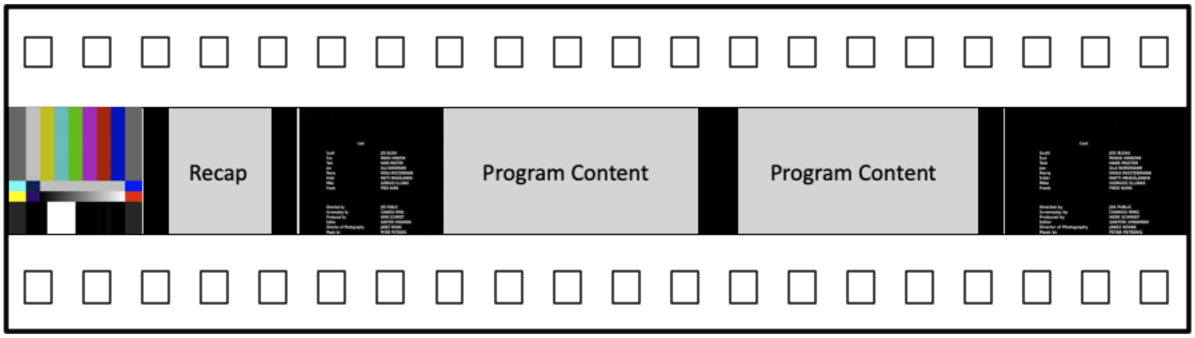
Content is the portions of the TV show or movie that contain the program or related elements. Black frames, credits, color bars, slates, and studio logos are not considered content. Amazon Rekognition Video can detect the start and end of each content segment in the video, so you can find the program run time or specific segments.

Content segments include, but are not limited to, the following:

- Program scenes between two ad breaks
- A quick recap of the previous episode at the beginning of the video
- Bonus post-credit content
- "Textless" content, such as a set of all program scenes that originally contained overlaid text, but where the text has been removed to support translation into other languages.

After Amazon Rekognition Video finishes detecting all of the content segments, you can apply domain knowledge or send them for human review to further categorize each segment. For example, if you use videos that always start with a recap, you could categorize the first content segment as a recap.

The following diagram shows technical cue segments on a show or movie's timeline. Note the color bars and opening credits, content segments such as the recap and main program, black frames throughout the video, and the end credits.



Shot detection

A shot is a series of interrelated consecutive pictures taken contiguously by a single camera and representing a continuous action in time and space. With Amazon Rekognition Video, you can detect the start, end, and duration of each shot, as well as a count for all the shots in a piece of content. You can use shot metadata for tasks such as the following.

- Creating promotional videos using selected shots.
- Inserting advertisements in locations that don’t disrupt the viewer's experience, such as the middle of a shot when someone is speaking.
- Generating a set of preview thumbnails that avoid transitional content between shots.

A shot detection is marked at the exact frame where there is a hard cut to a different camera. If there is a soft transition from one camera to another, Amazon Rekognition Video omits the transition. This ensures that shot start and end times don’t include sections without actual content.

The following diagram illustrates shot detection segments on a strip of film. Note that each shot is identified by a cut from one camera angle or location to the next.

About the Amazon Rekognition Video Segment detection API

To segment a stored video you use the asynchronous StartSegmentDetection and GetSegmentDetection API operations to start a segmentation job and fetch the results. Segment detection accepts videos stored in an Amazon S3 bucket and returns a JSON output. You can choose to detect only technical cues, only shot changes, or both together by configuring the `StartSegmentdetection` API request. You can also filter detected segments by setting thresholds for a minimum prediction confidence. For more information, see Using the Amazon Rekognition Segment API (p. 333). For example code, see Example: Detecting segments in a stored video (p. 337).

Using the Amazon Rekognition Segment API

Amazon Rekognition Video segment detection in stored videos is an Amazon Rekognition Video asynchronous operation. The Amazon Rekognition Segment API is a composite API where you choose the type of analysis (technical cues or shot detection) from a single API call. For information about calling asynchronous operations, see Calling Amazon Rekognition Video operations (p. 62).

Topics

Starting segment analysis

To start the detection of segments in a stored video call StartSegmentDetection. The input parameters are the same as other Amazon Rekognition Video operations with the addition of segment type selection and result filtering. For more information, see Starting video analysis (p. 62).

The following is example JSON passed by `StartSegmentDetection`. The request specifies that both technical cue and shot detection segments are detected. Different filters for the minimum detection confidence are requested for technical cue segments (90%) and shot detection segments (80%).

```
{
  "Video": {
    "S3Object": {
      "Bucket": "test_files",
      "Name": "test_file.mp4"
    }
    "SegmentTypes":["TECHNICAL_CUES", "SHOT"]
    "Filters": {
      "TechnicalCueFilter": {
         "MinSegmentConfidence": 90,
         "BlackFrame" : {
            "MaxPixelThreshold": 0.1,
            "MinCoveragePercentage": 95
         }
      },
      "ShotFilter" : {
          "MinSegmentConfidence": 60
      }
  }
}
```

Choosing a segment type

Use the `SegmentTypes` array input parameter to detect technical cue and/or shot detection segments in the input video.

- TECHNICAL_CUE — identifies frame-accurate timestamps for the start, end, and duration of technical cues (black frames, color bars, opening credits, end credits, studio logos, and primary program content) detected in a video. For example, you can use technical cues to find the start of the end credits. For more information, see Technical cues (p. 330).
- SHOT — Identifies the start, end, and duration of a shot. For example, you can use shot detection to identify candidate shots for a final edit of a video. For more information, see Shot detection (p. 332).

Filtering the analysis results

You can use the `Filters` (StartSegmentDetectionFilters) input parameter to specify the minimum detection confidence returned in the response. Within `Filters`, use `ShotFilter` (StartShotDetectionFilter) to filter detected shots. Use `TechnicalCueFilter` (StartTechnicalCueDetectionFilter) to filter technical cues.

For example code, see Example: Detecting segments in a stored video (p. 337).

Getting segment analysis results

Amazon Rekognition Video publishes the completion status of the video analysis to an Amazon Simple Notification Service topic. If the video analysis is successful, call GetSegmentDetection to get results of the video analysis.

The following is an example `GetSegmentDetection` request. The `JobId` is the job identifier returned from the call to `StartSegmentDetection`. For information about the other input parameters, see Getting Amazon Rekognition Video analysis results (p. 64).

```
{
    "JobId": "270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1784ca90fc180e3",
    "MaxResults": 10,
    "NextToken": "XfXnZKiyMOGDhzBzYUhS5puM+g1IgezqFeYpv/H/+5noP/LmM57FitUAwSQ5D6G4AB/
PNwolrw=="
}
```

`GetSegmentDetection` returns results for the requested analysis and general information about the stored video.

General information

`GetSegmentDection` returns the following general information.

- **Audio information** — The response includes audio metadata in an array, `AudioMetadata`, of AudioMetadata objects. There can be multiple audio streams. Each `AudioMetadata` object contains metadata for a single audio stream. Audio information in an `AudioMetadata` objects includes the audio codec, the number of audio channels, the duration of the audio stream, and the sample rate. Audio metadata is returned in each page of information returned by `GetSegmentDetection`.
- **Video information** – Currently, Amazon Rekognition Video returns a single VideoMetadata object in the `VideoMetadata` array. The object contains information about the video stream in the input file that Amazon Rekognition Video chose to analyze. The `VideoMetadata` object includes the video codec, video format and other information. Video metadata is returned in each page of information returned by `GetSegmentDetection`.

- **Paging information** – The example shows one page of segment information. You can specify how many elements to return in the `MaxResults` input parameter for `GetSegmentDetection`. If more results than `MaxResults` exist, `GetSegmentDetection` returns a token (`NextToken`) used to get the next page of results. For more information, see Getting Amazon Rekognition Video analysis results (p. 64).
- **Request information** – The type of analysis requested in the call to `StartSegmentDetection` is returned in the `SelectedSegmentTypes` field.

Segments

Technical cues and shot information detected in a video is returned in an array, `Segments`, of SegmentDetection objects. The array is sorted by the segment types (TECHNICAL_CUE or SHOT) specified in the `SegmentTypes` input parameter of `StartSegmentDetection`. Within each segment type the array is sorted by timestamp values. Each `SegmentDetection` object includes information about the type of detected segment (Technical cue or shot detection) and general information, such as the start time, end time, and the duration of the segment.

Time information is returned in three formats.

- **Milliseconds**

  The number of milliseconds since the start of the video. The fields `DurationMillis`, `StartTimestampMillis`, and `EndTimestampMillis` are in millisecond format.
- **Timecode**

  Amazon Rekognition Video timecodes are in SMPTE format where each frame of video has a unique timecode value. The format is *hh:mm:ss:frame*. For example, a timecode value of 01:05:40:07, would be read as one hour, five minutes, forty seconds and seven frames. Drop frame rate use cases are supported by Amazon Rekognition Video. The drop rate timecode format is *hh:mm:ss;frame*. The fields `DurationSMPTE`, `StartTimecodeSMPTE`, and `EndTimecodeSMPTE` are in timecode format.
- **Frame Counters**

  The duration of each video segment is also expressed with the number of frames. The field `StartFrameNumber` gives the frame number at the start of a video segment, and `EndFrameNumber` gives the frame number at the end of a video segment. `DurationFrames` gives the total number of frames in a video segment. These values are calculated using a frame index that starts with 0.

You can use the `SegmentType` field to determine the type of a segment returned by Amazon Rekognition Video.

- **Technical Cues** – the `TechnicalCueSegment` field is an TechnicalCueSegment object that contains the detection confidence and the type of a technical cue. The types of technical cue are `ColorBars`, `EndCredits`, `BlackFrames`, `OpeningCredits`, `StudioLogo`, `Slate`, and `Content`.
- **Shot** – the `ShotSegment` field is a ShotSegment object than contains the detection confidence and an identifier for the shot segment within the video.

The following example is the JSON response from `GetSegmentDetection`.


```
{
    "SelectedSegmentTypes": [
        {
            "ModelVersion": "2.0",
            "Type": "SHOT"
        },
        {
            "ModelVersion": "2.0",
```

```
            "Type": "TECHNICAL_CUE"
        }
    ],
    "Segments": [
        {
            "DurationFrames": 299,
            "DurationSMPTE": "00:00:09;29",
            "StartFrameNumber": 0,
            "EndFrameNumber": 299,
            "EndTimecodeSMPTE": "00:00:09;29",
            "EndTimestampMillis": 9976,
            "StartTimestampMillis": 0,
            "DurationMillis": 9976,
            "StartTimecodeSMPTE": "00:00:00;00",
            "Type": "TECHNICAL_CUE",
            "TechnicalCueSegment": {
                "Confidence": 90.45006561279297,
                "Type": "BlackFrames"
            }
        },
        {
            "DurationFrames": 150,
            "DurationSMPTE": "00:00:05;00",
            "StartFrameNumber": 299,
            "EndFrameNumber": 449,
            "EndTimecodeSMPTE": "00:00:14;29",
            "EndTimestampMillis": 14981,
            "StartTimestampMillis": 9976,
            "DurationMillis": 5005,
            "StartTimecodeSMPTE": "00:00:09;29",
            "Type": "TECHNICAL_CUE",
            "TechnicalCueSegment": {
                "Confidence": 100.0,
                "Type": "Content"
            }
        },
        {
            "DurationFrames": 299,
            "ShotSegment": {
                "Index": 0,
                "Confidence": 99.9982681274414
            },
            "DurationSMPTE": "00:00:09;29",
            "StartFrameNumber": 0,
            "EndFrameNumber": 299,
            "EndTimecodeSMPTE": "00:00:09;29",
            "EndTimestampMillis": 9976,
            "StartTimestampMillis": 0,
            "DurationMillis": 9976,
            "StartTimecodeSMPTE": "00:00:00;00",
            "Type": "SHOT"
        },
        {
            "DurationFrames": 149,
            "ShotSegment": {
                "Index": 1,
                "Confidence": 99.9982681274414
            },
            "DurationSMPTE": "00:00:04;29",
            "StartFrameNumber": 300,
            "EndFrameNumber": 449,
            "EndTimecodeSMPTE": "00:00:14;29",
            "EndTimestampMillis": 14981,
            "StartTimestampMillis": 10010,
            "DurationMillis": 4971,
            "StartTimecodeSMPTE": "00:00:10;00",
```

```
            "Type": "SHOT"
        }
    ],
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": [
        {
            "Format": "QuickTime / MOV",
            "FrameRate": 29.970029830932617,
            "Codec": "h264",
            "DurationMillis": 15015,
            "FrameHeight": 1080,
            "FrameWidth": 1920,
            "ColorRange": "LIMITED"

        }
    ],
    "AudioMetadata": [
        {
            "NumberOfChannels": 1,
            "SampleRate": 48000,
            "Codec": "aac",
            "DurationMillis": 15007
        }
    ]
}
```


For example code, see Example: Detecting segments in a stored video (p. 337).

Example: Detecting segments in a stored video

The following procedure shows how to detect technical cue segments and shot detection segments in a video stored in an Amazon S3 bucket. The procedure also shows how to filter detected segments based on the confidence that Amazon Rekognition Video has in the accuracy of the detection.

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 segments 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 to the code that you used in step 1.

   Java

   1. Add the following imports.

```
import com.amazonaws.services.rekognition.model.GetSegmentDetectionRequest;
import com.amazonaws.services.rekognition.model.GetSegmentDetectionResult;
import com.amazonaws.services.rekognition.model.SegmentDetection;
import com.amazonaws.services.rekognition.model.SegmentType;
import com.amazonaws.services.rekognition.model.SegmentTypeInfo;
import com.amazonaws.services.rekognition.model.ShotSegment;
import com.amazonaws.services.rekognition.model.StartSegmentDetectionFilters;
import com.amazonaws.services.rekognition.model.StartSegmentDetectionRequest;
import com.amazonaws.services.rekognition.model.StartSegmentDetectionResult;
import com.amazonaws.services.rekognition.model.StartShotDetectionFilter;
import
 com.amazonaws.services.rekognition.model.StartTechnicalCueDetectionFilter;
import com.amazonaws.services.rekognition.model.TechnicalCueSegment;
```

```
import com.amazonaws.services.rekognition.model.AudioMetadata;
```

2. Add the following code to the class `VideoDetect`.

```
    //Copyright 2020 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 StartSegmentDetection(String bucket, String video)
 throws Exception{

        NotificationChannel channel= new NotificationChannel()
                .withSNSTopicArn(snsTopicArn)
                .withRoleArn(roleArn);

        float minTechnicalCueConfidence = 80F;
        float minShotConfidence = 80F;

        StartSegmentDetectionRequest req = new StartSegmentDetectionRequest()
                .withVideo(new Video()
                        .withS3Object(new S3Object()
                                .withBucket(bucket)
                                .withName(video)))
                .withSegmentTypes("TECHNICAL_CUE" , "SHOT")
                .withFilters(new StartSegmentDetectionFilters()
                        .withTechnicalCueFilter(new
 StartTechnicalCueDetectionFilter()

 .withMinSegmentConfidence(minTechnicalCueConfidence))
                        .withShotFilter(new StartShotDetectionFilter()
                                .withMinSegmentConfidence(minShotConfidence)))
                .withJobTag("DetectingVideoSegments")
                .withNotificationChannel(channel);

        StartSegmentDetectionResult startLabelDetectionResult =
 rek.startSegmentDetection(req);
        startJobId=startLabelDetectionResult.getJobId();

    }

    private static void GetSegmentDetectionResults() throws Exception{

        int maxResults=10;
        String paginationToken=null;
        GetSegmentDetectionResult segmentDetectionResult=null;
        Boolean firstTime=true;


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

            GetSegmentDetectionRequest segmentDetectionRequest= new
 GetSegmentDetectionRequest()
                    .withJobId(startJobId)
                    .withMaxResults(maxResults)
                    .withNextToken(paginationToken);

            segmentDetectionResult =
 rek.getSegmentDetection(segmentDetectionRequest);

            if(firstTime) {
                System.out.println("\nStatus\n------");
```

```
                System.out.println(segmentDetectionResult.getJobStatus());
                System.out.println("\nRequested features\n------------------");
                 for (SegmentTypeInfo requestedFeatures :
segmentDetectionResult.getSelectedSegmentTypes()) {
                    System.out.println(requestedFeatures.getType());
                }
                 int count=1;
                 List<VideoMetadata> videoMetaDataList =
segmentDetectionResult.getVideoMetadata();
                 System.out.println("\nVideo Streams\n-------------");
                 for (VideoMetadata videoMetaData: videoMetaDataList) {
                     System.out.println("Stream: " + count++);
                     System.out.println("\tFormat: " +
videoMetaData.getFormat());
                     System.out.println("\tCodec: " +
videoMetaData.getCodec());
                     System.out.println("\tDuration: " +
videoMetaData.getDurationMillis());
                     System.out.println("\tFrameRate: " +
videoMetaData.getFrameRate());
                 }


                 List<AudioMetadata> audioMetaDataList =
segmentDetectionResult.getAudioMetadata();
                 System.out.println("\nAudio streams\n-------------");

                 count=1;
                 for (AudioMetadata audioMetaData: audioMetaDataList) {
                     System.out.println("Stream: " + count++);
                     System.out.println("\tSample Rate: " +
audioMetaData.getSampleRate());
                     System.out.println("\tCodec: " +
audioMetaData.getCodec());
                     System.out.println("\tDuration: " +
audioMetaData.getDurationMillis());
                     System.out.println("\tNumber of Channels: " +
audioMetaData.getNumberOfChannels());
                 }
                 System.out.println("\nSegments\n--------");

                firstTime=false;
            }


            //Show segment information

            List<SegmentDetection> detectedSegments=
segmentDetectionResult.getSegments();

            for (SegmentDetection detectedSegment: detectedSegments) {

               if
(detectedSegment.getType().contains(SegmentType.TECHNICAL_CUE.toString())) {
                   System.out.println("Technical Cue");
                   TechnicalCueSegment
segmentCue=detectedSegment.getTechnicalCueSegment();
                   System.out.println("\tType: " + segmentCue.getType());
                   System.out.println("\tConfidence: " +
segmentCue.getConfidence().toString());
                }
               if
(detectedSegment.getType().contains(SegmentType.SHOT.toString())) {
                   System.out.println("Shot");
                   ShotSegment segmentShot=detectedSegment.getShotSegment();
                   System.out.println("\tIndex " + segmentShot.getIndex());
```

```
                    System.out.println("\tConfidence: " +
segmentShot.getConfidence().toString());
                }
                long seconds=detectedSegment.getDurationMillis();
                System.out.println("\tDuration : " + Long.toString(seconds) + "
milliseconds");
                System.out.println("\tStart time code: " +
detectedSegment.getStartTimecodeSMPTE());
                System.out.println("\tEnd time code: " +
detectedSegment.getEndTimecodeSMPTE());
                System.out.println("\tDuration time code: " +
detectedSegment.getDurationSMPTE());
                System.out.println();

            }

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

    }
```

3. In the function `main`, replace the lines:

```
        StartLabelDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetLabelDetectionResults();
```

with:

```
        StartSegmentDetection(bucket, video);

        if (GetSQSMessageSuccess()==true)
         GetSegmentDetectionResults();
```

Java V2

```
    public static void StartSegmentDetection (RekognitionClient rekClient,
                                NotificationChannel channel,
                                String bucket,
                                String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                .bucket(bucket)
                .name(video)
                .build();

            Video vidOb = Video.builder()
                .s3Object(s3Obj)
                .build();

            StartShotDetectionFilter cueDetectionFilter =
StartShotDetectionFilter.builder()
                .minSegmentConfidence(60F)
                .build();

            StartTechnicalCueDetectionFilter technicalCueDetectionFilter =
StartTechnicalCueDetectionFilter.builder()
                .minSegmentConfidence(60F)
                .build();
```

```
            StartSegmentDetectionFilters filters =
StartSegmentDetectionFilters.builder()
                .shotFilter(cueDetectionFilter)
                .technicalCueFilter(technicalCueDetectionFilter)
                .build();

            StartSegmentDetectionRequest segDetectionRequest =
StartSegmentDetectionRequest.builder()
                .jobTag("DetectingLabels")
                .notificationChannel(channel)
                .segmentTypes(SegmentType.TECHNICAL_CUE , SegmentType.SHOT)
                .video(vidOb)
                .filters(filters)
                .build();

            StartSegmentDetectionResponse segDetectionResponse =
rekClient.startSegmentDetection(segDetectionRequest);
            startJobId = segDetectionResponse.jobId();

        } catch(RekognitionException e) {
            e.getMessage();
            System.exit(1);
        }
    }

    public static void getSegmentResults(RekognitionClient rekClient) {

        try {
            String paginationToken = null;
            GetSegmentDetectionResponse segDetectionResponse = null;
            boolean finished = false;
            String status;
            int yy = 0;

            do {
                if (segDetectionResponse != null)
                    paginationToken = segDetectionResponse.nextToken();

                GetSegmentDetectionRequest recognitionRequest =
GetSegmentDetectionRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds.
                while (!finished) {
                    segDetectionResponse =
rekClient.getSegmentDetection(recognitionRequest);
                    status = segDetectionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }
                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is null.
                List<VideoMetadata> videoMetaData =
segDetectionResponse.videoMetadata();
                for (VideoMetadata metaData : videoMetaData) {
                    System.out.println("Format: " + metaData.format());
```

```
                    System.out.println("Codec: " + metaData.codec());
                    System.out.println("Duration: " + metaData.durationMillis());
                    System.out.println("FrameRate: " + metaData.frameRate());
                    System.out.println("Job");
                }

                List<SegmentDetection> detectedSegments =
segDetectionResponse.segments();
                for (SegmentDetection detectedSegment : detectedSegments) {
                    String type = detectedSegment.type().toString();
                    if (type.contains(SegmentType.TECHNICAL_CUE.toString())) {
                        System.out.println("Technical Cue");
                        TechnicalCueSegment segmentCue =
detectedSegment.technicalCueSegment();
                        System.out.println("\tType: " + segmentCue.type());
                        System.out.println("\tConfidence: " +
segmentCue.confidence().toString());
                    }

                    if (type.contains(SegmentType.SHOT.toString())) {
                        System.out.println("Shot");
                        ShotSegment segmentShot = detectedSegment.shotSegment();
                        System.out.println("\tIndex " + segmentShot.index());
                        System.out.println("\tConfidence: " +
segmentShot.confidence().toString());
                    }

                    long seconds = detectedSegment.durationMillis();
                    System.out.println("\tDuration : " + seconds + "
milliseconds");
                    System.out.println("\tStart time code: " +
detectedSegment.startTimecodeSMPTE());
                    System.out.println("\tEnd time code: " +
detectedSegment.endTimecodeSMPTE());
                    System.out.println("\tDuration time code: " +
detectedSegment.durationSMPTE());
                    System.out.println();
                }

            } while (segDetectionResponse !=null &&
segDetectionResponse.nextToken() != null);

        } catch(RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

Python

1. Add the following code to the class `VideoDetect` that you created in step 1.

```
# Copyright 2020 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 StartSegmentDetection(self):

        min_Technical_Cue_Confidence = 80.0
        min_Shot_Confidence = 80.0
        max_pixel_threshold = 0.1
        min_coverage_percentage = 60

        response = self.rek.start_segment_detection(
```

```
            Video={"S3Object": {"Bucket": self.bucket, "Name": self.video}},
            NotificationChannel={
                "RoleArn": self.roleArn,
                "SNSTopicArn": self.snsTopicArn,
            },
            SegmentTypes=["TECHNICAL_CUE", "SHOT"],
            Filters={
                "TechnicalCueFilter": {
                    "BlackFrame": {
                        "MaxPixelThreshold": max_pixel_threshold,
                        "MinCoveragePercentage": min_coverage_percentage,
                    },
                    "MinSegmentConfidence": min_Technical_Cue_Confidence,
                },
                "ShotFilter": {"MinSegmentConfidence": min_Shot_Confidence},
            }
        )

        self.startJobId = response["JobId"]
        print(f"Start Job Id: {self.startJobId}")

    def GetSegmentDetectionResults(self):
        maxResults = 10
        paginationToken = ""
        finished = False
        firstTime = True

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

            if firstTime == True:
                print(f"Status\n------\n{response['JobStatus']}")
                print("\nRequested Types\n---------------")
                for selectedSegmentType in response['SelectedSegmentTypes']:
                    print(f"\tType: {selectedSegmentType['Type']}")
                    print(f"\t\tModel Version:
{selectedSegmentType['ModelVersion']}")

                print()
                print("\nAudio metadata\n--------------")
                for audioMetadata in response['AudioMetadata']:
                    print(f"\tCodec: {audioMetadata['Codec']}")
                    print(f"\tDuration: {audioMetadata['DurationMillis']}")
                    print(f"\tNumber of Channels:
{audioMetadata['NumberOfChannels']}")
                    print(f"\tSample rate: {audioMetadata['SampleRate']}")
                print()
                print("\nVideo metadata\n--------------")
                for videoMetadata in response["VideoMetadata"]:
                    print(f"\tCodec: {videoMetadata['Codec']}")
                    print(f"\tColor Range: {videoMetadata['ColorRange']}")
                    print(f"\tDuration: {videoMetadata['DurationMillis']}")
                    print(f"\tFormat: {videoMetadata['Format']}")
                    print(f"\tFrame rate: {videoMetadata['FrameRate']}")
                    print("\nSegments\n--------")

                firstTime = False

            for segment in response['Segments']:

                if segment["Type"] == "TECHNICAL_CUE":
                    print("Technical Cue")
```

```
                    print(f"\tConfidence: {segment['TechnicalCueSegment']
['Confidence']}")
                    print(f"\tType: {segment['TechnicalCueSegment']['Type']}")

                if segment["Type"] == "SHOT":
                    print("Shot")
                    print(f"\tConfidence: {segment['ShotSegment']
['Confidence']}")
                    print(f"\tIndex: " + str(segment["ShotSegment"]["Index"]))

                print(f"\tDuration (milliseconds):
 {segment['DurationMillis']}")
                print(f"\tStart Timestamp (milliseconds):
 {segment['StartTimestampMillis']}")
                print(f"\tEnd Timestamp (milliseconds):
 {segment['EndTimestampMillis']}")

                print(f"\tStart timecode: {segment['StartTimecodeSMPTE']}")
                print(f"\tEnd timecode: {segment['EndTimecodeSMPTE']}")
                print(f"\tDuration timecode: {segment['DurationSMPTE']}")

                print(f"\tStart frame number {segment['StartFrameNumber']}")
                print(f"\tEnd frame number: {segment['EndFrameNumber']}")
                print(f"\tDuration frames: {segment['DurationFrames']}")

                print()

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

2. In the function `main`, replace the lines:

```
    analyzer.StartLabelDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetLabelDetectionResults()
```

with:

```
    analyzer.StartSegmentDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetSegmentDetectionResults()
```

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 code to replace might be different.

3. Run the code. Information about the segments detected in the input video are displayed.

Tutorials

These cross-service tutorials demonstrate how to use Rekognition's API operations alongside other AWS services to create sample applications and accomplish a variety of tasks. Most of these tutorials make use of Amazon S3 to store images or video. Other commonly used services include AWS Lambda.

Topics

Storing Amazon Rekognition Data with Amazon RDS and DynamoDB

When using Amazon Rekognition's APIs, it's important to remember that the API operations don't save any of the generated labels. You can save these labels by placing them in database, along with identifiers for the respective images.

This tutorial demonstrates detecting labels and saving those detected labels to a database. The sample application developed in this tutorial will read images from an Amazon S3 bucket call the DetectLabels operation on these images, and store the resulting labels in a database. The application will store data in either an Amazon RDS database instance or a DynamoDB database, depending on which database type you'd like to use.

You'll use the AWS SDK for Python or this tutorial. You can also see the AWS Documentation SDK examples GitHub repo for more Python tutorials.

Topics

Prerequisites

Before you begin this tutorial, you'll need install Python and complete the steps required to set up the Python AWS SDK. Beyond this, ensure that you have:

Created an AWS account and an IAM role

Installed the Python SDK (Boto3)

Properly configured your AWS access credentials

Created Amazon S3 bucket filled it with images

Created a RDS database instance, if using RDS to store data

Getting Labels for Images in an Amazon S3 Bucket

Start by writing a function that will take the name of an image in your Amazon S3 bucket and retrieve that image. This image will be displayed to confirm that the correct images are being passed into a call to DetectLabels which is also in the function.

1. Find the Amazon S3 bucket you would like to use and write down its name. You will make calls to this Amazon S3 bucket and read the images inside it. Ensure your bucket contains some images to pass to the DetectLabels operation.
2. Write the code to connect to your Amazon S3 bucket. You can connect to the Amazon S3 resource with Boto3 to retrieve an image from an Amazon S3 bucket. Once connected to the Amazon S3 resource, you can access your bucket by providing the Bucket method with the name of your Amazon S3 bucket. After connecting to the Amazon S3 bucket, you retrieve images from the bucket by using the Object method. By making use of Matplotlib, you can use this connection to visualize your images as they process. Boto3 is also used to connect to the Rekognition client.

   In the following code, provide your region to the region_name parameter. You will pass the Amazon S3 bucket name and the image name to DetectLabels , which returns the labels for the corresponding image. After selecting just the labels from the response, both the name of the image and the labels are returned.

```
import boto3
from io import BytesIO
from matplotlib import pyplot as plt
from matplotlib import image as mp_img

boto3 = boto3.Session()

def read_image_from_s3(bucket_name, image_name):

    # Connect to the S3 resource with Boto 3
    # get bucket and find object matching image name
    s3 = boto3.resource('s3')
    bucket = s3.Bucket(name=bucket_name)
    Object = bucket.Object(image_name)

    # Downloading the image for display purposes, not necessary for detection of labels
    # You can comment this code out if you don't want to visualize the images
    file_name = Object.key
    file_stream = BytesIO()
    Object.download_fileobj(file_stream)
    img = mp_img.imread(file_stream, format="jpeg")
    plt.imshow(img)
    plt.show()

    # get the labels for the image by calling DetectLabels from Rekognition
    client = boto3.client('rekognition', region_name="region-name")
    response = client.detect_labels(Image={'S3Object': {'Bucket': bucket_name, 'Name':
 image_name}},
                                    MaxLabels=10)

    print('Detected labels for ' + image_name)

    full_labels = response['Labels']
```

```
    return file_name, full_labels
```

3. Save this code in a file called get_images.py.

Creating an Amazyon DynamoDB Table

The following code uses Boto3 to connect to DynamoDB and uses the DynamoDB `CreateTable` method to create a table named Images. The table has a composite primary key consisting of a partition key called Image and a sort key called Labels. The Image key contains the name of the image, while the Labels key stores the labels assigned to that Image.

```
import boto3

def create_new_table(dynamodb=None):
    dynamodb = boto3.resource(
        'dynamodb',)
    # Table defination
    table = dynamodb.create_table(
        TableName='Images',
        KeySchema=[
            {
                'AttributeName': 'Image',
                'KeyType': 'HASH'  # Partition key
            },
            {
                'AttributeName': 'Labels',
                'KeyType': 'RANGE'  # Sort key
            }
        ],
        AttributeDefinitions=[
            {
                'AttributeName': 'Image',
                'AttributeType': 'S'
            },
            {
                'AttributeName': 'Labels',
                'AttributeType': 'S'
            },
        ],
        ProvisionedThroughput={
            'ReadCapacityUnits': 10,
            'WriteCapacityUnits': 10
        }
    )
    return table

if __name__ == '__main__':
    device_table = create_new_table()
    print("Status:", device_table.table_status)
```

Save this code in an editor and run it once to create a DynamoDB table.

Uploading Data to DynamoDB

Now that the DynamoDB database has been created and you have a function to get labels for images, you can store the labels in DynamoDB The following code retrieves all the images in an S3 bucket, get labels for them, and stores the data in DynamoDB.

1. You'll need to write the code for uploading the data to DynamoDB. A function called `get_image_names` is used to connect to your Amazon S3 bucket and it returns the names of all

images in the bucket as a list. You'll pass this list into the `read_image_from_S3` function, which is imported from the `get_images.py` file you created.

```
import boto3
import json
from get_images import read_image_from_s3

boto3 = boto3.Session()

def get_image_names(name_of_bucket):

    s3_resource = boto3.resource('s3')
    my_bucket = s3_resource.Bucket(name_of_bucket)
    file_list = []
    for file in my_bucket.objects.all():
        file_list.append(file.key)
    return file_list
```

2. The `read_image_from_S3` function we created earlier will return the name of the image being processed and the dictionary of labels associated with that image. A function called `find_values` is used to get just the labels from the response. The name of the image and its labels are then ready to be uploaded to your DynamoDB table.

```
def find_values(id, json_repr):
    results = []

    def _decode_dict(a_dict):
        try:
            results.append(a_dict[id])
        except KeyError:
            pass
        return a_dict

    json.loads(json_repr, object_hook=_decode_dict) # Return value ignored.
    return results
```

3. You will use a third function, called `load_data`, to actually load the images and labels into the DynamoDB table you created.

```
def load_data(image_labels, dynamodb=None):

    if not dynamodb:
        dynamodb = boto3.resource('dynamodb')

    table = dynamodb.Table('Images')

    print("Adding image details:", image_labels)
    table.put_item(Item=image_labels)
    print("Success!!")
```

4. Here's where the three functions we defined previously are called, and the operations are carried out. Add the three functions defined above, along with the code below, to a Python file. Run the code.

```
bucket = "bucket_name"
file_list = get_image_names(bucket)

for file in file_list:
    file_name = file
    print("Getting labels for " + file_name)
    image_name, image_labels = read_image_from_s3(bucket, file_name)
    image_json_string = json.dumps(image_labels, indent=4)
```

```
    labels=set(find_values("Name", image_json_string))
    print("Labels found: " + str(labels))
    labels_dict = {}
    print("Saving label data to database")
    labels_dict["Image"] = str(image_name)
    labels_dict["Labels"] = str(labels)
    print(labels_dict)
    load_data(labels_dict)
    print("Success!")
```

You’ve just used DetectLabels to generate labels for your images and stored those labels in an DynamoDB instance. Be sure that you tear down all the resources you created while going through this tutorial. That will prevent you from being charged for resources you aren’t using.

Creating a MySQL Database in Amazon RDS

Before going further, make sure you have completed the setup procedure for Amazon RDS and created a MySQL DB instance using Amazon RDS.

The following code makes use of the PyMySQL library and your Amazon RDS DB instance. It creates a table to hold the names of your images and the labels associated with those images. Amazon RDS receives commands to create tables and insert data into tables. To use Amazon RDS, you must connect to the Amazon RDS host using your host name, username, and password. You'll connect to Amazon RDS by providing these arguments to PyMySQL's `connect` function and creating an instance of a cursor.

1. In the following code, replace the value of host with your Amazon RDS host endpoint and replace the value of user with the master username associated with your Amazon RDS instance. You will also need to replace password with the master password for your main user.

```
import pymysql

host = "host-endpoint"
user = "username"
password = "master-password"
```

2. Create a database and a table to insert your image and label data into. Do this by running and committing a creation query. The following code creates a database. Run this code only once.

```
conn = pymysql.connect(host=host, user=user, passwd=password)
print(conn)
cursor = conn.cursor()
print("Connection successful")

# run once
create_query = "create database rekogDB1"
print("Creation successful!")
cursor.execute(create_query)
cursor.connection.commit()
```

3. Once the database has been created, you must create a table to insert your image names and labels into. To create a table, you will first pass the use SQL command, along with the name of your database, to the `execute` function. After the connection is made, a query to create a table is run. The following code connects to the database and then creates a table with both a primary key, called `image_id`, and a text attribute storing the labels. Use the imports and variables you defined earlier, and run this code to create a table in your database.

```
# connect to existing DB
cursor.execute("use rekogDB1")
```

```
cursor.execute("CREATE TABLE IF NOT EXISTS test_table(image_id VARCHAR (255) PRIMARY
 KEY, image_labels TEXT)")
conn.commit()
print("Table creation - Successful creation!")
```

Uploading Data to a Amazon RDS MySQL Table

After creating the Amazon RDS database and a table in the database, you can get labels for your images and store those labels in the Amazon RDS database.

1. Connect to your Amazon S3 bucket and retrieve the names of all the images in the bucket. These image names will be passed into the `read_image_from_s3` function you created earlier to get the labels for all your images. The following code connects to your Amazon S3 bucket and returns a list of all the images in your bucket.

```
import pymysql
from get_images import read_image_from_s3
import json
import boto3

host = "host-endpoint"
user = "username"
password = "master-password"

conn = pymysql.connect(host=host, user=user, passwd=password)
print(conn)
cursor = conn.cursor()
print("Connection successful")

def get_image_names(name_of_bucket):

    s3_resource = boto3.resource('s3')
    my_bucket = s3_resource.Bucket(name_of_bucket)
    file_list = []
    for file in my_bucket.objects.all():
        file_list.append(file.key)
    return file_list
```

2. The response from the DetectLabels API contains more than just the labels, so write a function to extract only the label values. The following function returns a list full of just the labels.

```
def find_values(id, json_repr):
    results = []

    def _decode_dict(a_dict):
        try:
            results.append(a_dict[id])
        except KeyError:
            pass
        return a_dict

    json.loads(json_repr, object_hook=_decode_dict) # Return value ignored.
    return results
```

3. You will need a function to insert the image names and labels into your table. The following function runs an insertion query and inserts any given pair of image name and labels.

```
def upload_data(image_id, image_labels):

    # insert into db
```

```
    cursor.execute("use rekogDB1")
    query = "INSERT IGNORE INTO test_table(image_id, image_labels) VALUES (%s, %s)"
    values = (image_id, image_labels)
    cursor.execute(query, values)
    conn.commit()
    print("Insert successful!")
```

4. Finally, you must run the functions you defined above. In the following code, the names of all the images in your bucket are collected and provided to the function that calls DetectLabels. Afterward, the labels and the name of the image they apply to are uploaded to your Amazon RDS database. Copy the three functions defined above, along with the code below, into a Python file. Run the Python file.

```
bucket = "bucket-name"
file_list = get_image_names(bucket)

for file in file_list:
    file_name = file
    print("Getting labels for " + file_name)
    image_name, image_labels = read_image_from_s3(bucket, file_name)
    image_json = json.dumps(image_labels, indent=4)
    labels=set(find_values("Name", image_json))
    print("Labels found: " + str(labels))
    unique_labels=set(find_values("Name", image_json))
    print(unique_labels)
    image_name_string = str(image_name)
    labels_string = str(unique_labels)
    upload_data(image_name_string, labels_string)
    print("Success!")
```

You have successfully used DetectLabels to generate labels for your images and stored those labels in a MySQL database using Amazon RDS. Be sure that you tear down all the resources you created while going through this tutorial. This will prevent you from being charged for resources you aren’t using.

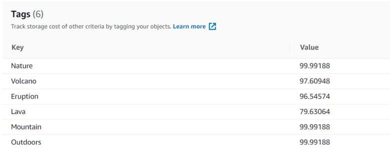
For more AWS multiservice examples, see the AWS Documentation SDK examples GitHub repository.

Using Amazon Rekognition and Lambda to tag assets in an Amazon S3 bucket

In this tutorial, you create an AWS Lambda function that automatically tags digital assets located in an Amazon S3 bucket. The Lambda function reads all objects in a given Amazon S3 bucket. For each object in the bucket, it passes the image to the Amazon Rekognition service to geneate a series of labels. Each label is used to create a tag that is applied to the image. After you execute the Lambda function, it automatically creates tags based on all images in a given Amazon S3 bucket and applies them to the images.

For example, assume you run the Lambda function and you have this image in an Amazon S3 bucket.

The application then automatically creates tags and applies them to the image.



Tags (6)

Track storage cost of other criteria by tagging your objects. Learn more

Key	Value
Nature	99.99188
Volcano	97.60948
Eruption	96.54574
Lava	79.63064
Mountain	99.99188
Outdoors	99.99188

> **Note**
> The services you use in this tutorial are part of the AWS Free Tier. When you are done with the tutorial, we recommend terminating any resources you created during the tutorial so that you are not charged.

This tutorial uses the AWS SDK for Java version 2. See the AWS Documentation SDK examples GitHub repository for additional Java V2 tutorials.

Topics

Prerequisites

Before you begin, you need to complete the steps in Setting Up the AWS SDK for Java. Then make sure that you have the following:

- Java 1.8 JDK.
- Maven 3.6 or higher.
- An Amazon S3 bucket with 5-7 nature images in it. These images are read by the Lambda function.

Configure the IAM Lambda role

This tutorial uses the Amazon Rekognition and Amazon S3 services. Configure the **lambda-support** role to have policies that enable it to invoke these services from a Lambda function.

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**, then choose **Create Role**.
3. Choose **AWS service**, and then choose **Lambda**.
4. Choose the **Permissions** tab.
5. Search for **AWSLambdaBasicExecutionRole**.
6. Choose **Next tags**.
7. Choose **Review**.
8. Name the role **lambda-support**.

9. Choose **Create role**.
10. Choose **lambda-support** to view the overview page.
11. Choose **Attach policies**.
12. Choose *AmazonRekognitionFullAccess* from the list of policies.
13. Choose **Attach policy**.
14. Search for **AmazonS3FullAccess**, and then choose **Attach policy**.

Create the project

Create a new Java project, then configure the Maven pom.xml with the required settings and dependencies. Make sure your pom.xml file looks like the following:

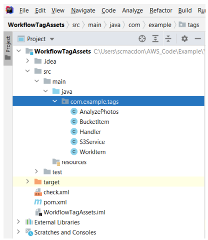
```
<?xml version="1.0" encoding="UTF-8"?>
 <project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.example</groupId>
<artifactId>WorkflowTagAssets</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
<name>java-basic-function</name>
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>bom</artifactId>
            <version>2.10.54</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<dependencies>
   <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-lambda-java-core</artifactId>
        <version>1.2.1</version>
    </dependency>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.8.6</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
        <version>2.10.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.13.0</version>
        <scope>test</scope>
```

```
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-slf4j18-impl</artifactId>
        <version>2.13.3</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.6.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>5.6.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>com.googlecode.json-simple</groupId>
        <artifactId>json-simple</artifactId>
        <version>1.1.1</version>
    </dependency>
    <dependency>
        <groupId>software.amazon.awssdk</groupId>
        <artifactId>s3</artifactId>
    </dependency>
    <dependency>
        <groupId>software.amazon.awssdk</groupId>
        <artifactId>rekognition</artifactId>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.22.2</version>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-shade-plugin</artifactId>
            <version>3.2.2</version>
            <configuration>
                <createDependencyReducedPom>false</createDependencyReducedPom>
            </configuration>
            <executions>
                <execution>
                    <phase>package</phase>
                    <goals>
                        <goal>shade</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
  </build>
```

```
</project>
```

Write the code

Use the AWS Lambda runtime Java API to create the Java class that defines the Lambda function. In this example, there is one Java class for the Lambda function named **Handler** and additional classes required for this use case. The following figure shows the Java classes in the project. Notice that all Java classes are located in a package named **com.example.tags**.



Create the following Java classes for the code:

- **Handler** uses the Lambda Java run-time API and performs the use case described in this AWS tutorial. The application logic that's executed is located in the handleRequest method.
- **S3Service** uses the Amazon S3 API to perform S3 operations.
- **AnalyzePhotos** uses the Amazon Rekognition API to analyze the images.
- **BucketItem** defines a model that stores Amazon S3 bucket information.
- **WorkItem** defines a model that stores Amazon Rekognition data.

Handler class

This Java code represents the **Handler** class. The class reads a flag that is passed to the Lambda function. The **s3Service.ListBucketObjects** method returns a **List** object where each element is a string value that represents the object key. If the flag value is true, then tags are applied by iterating through the list and applying tags to each object by calling the **s3Service.tagAssets** method. If the flag value is false, then the **s3Service.deleteTagFromObject** method is invoked that deletes the tags. Also, notice that you can log messages to Amazon CloudWatch logs by using a **LambdaLogger** object.

> **Note**
> Make sure you assign your bucket name to the **bucketName** variable.

```
package com.example.tags;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class Handler implements RequestHandler<Map<String,String>, String> {

@Override
public String handleRequest(Map<String, String> event, Context context) {
    LambdaLogger logger = context.getLogger();
    String delFlag = event.get("flag");
    logger.log("FLAG IS: " + delFlag);
    S3Service s3Service = new S3Service();
```

```
    AnalyzePhotos photos = new AnalyzePhotos();

    String bucketName = "<Enter your bucket name>";
    List<String> myKeys = s3Service.listBucketObjects(bucketName);
    if (delFlag.compareTo("true") == 0) {

        // Create a List to store the data.
        List<ArrayList<WorkItem>> myList = new ArrayList<>();

        // loop through each element in the List and tag the assets.
        for (String key : myKeys) {

            byte[] keyData = s3Service.getObjectBytes(bucketName, key);

            // Analyze the photo and return a list where each element is a WorkItem.
            ArrayList<WorkItem> item = photos.detectLabels(keyData, key);
            myList.add(item);
        }

        s3Service.tagAssets(myList, bucketName);
        logger.log("All Assets in the bucket are tagged!");

    } else {

        // Delete all object tags.
        for (String key : myKeys) {
            s3Service.deleteTagFromObject(bucketName, key);
            logger.log("All Assets in the bucket are deleted!");
        }
     }
    return delFlag;
  }
 }
```

S3Service class

The following class uses the Amazon S3 API to perform S3 operations. For example, the **getObjectBytes** method returns a byte array that represents the image. Likewise, the **listBucketObjects** method returns a **List** object where each element is a string value that specifies the key name.

```
package com.example.tags;

import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.PutObjectTaggingRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import software.amazon.awssdk.services.s3.model.ListObjectsResponse;
import software.amazon.awssdk.services.s3.model.S3Object;
import software.amazon.awssdk.services.s3.model.GetObjectTaggingResponse;
import software.amazon.awssdk.services.s3.model.ListObjectsRequest;
import java.util.ArrayList;
import java.util.List;
import software.amazon.awssdk.services.s3.model.Tagging;
import software.amazon.awssdk.services.s3.model.Tag;
import software.amazon.awssdk.services.s3.model.GetObjectTaggingRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectTaggingRequest;

public class S3Service {

private S3Client getClient() {
```

```
    Region region = Region.US_WEST_2;
    return S3Client.builder()
            .region(region)
            .build();
}

public byte[] getObjectBytes(String bucketName, String keyName) {

    S3Client s3 = getClient();

    try {

        GetObjectRequest objectRequest = GetObjectRequest
                .builder()
                .key(keyName)
                .bucket(bucketName)
                .build();

        // Return the byte[] from this object.
        ResponseBytes<GetObjectResponse> objectBytes = s3.getObjectAsBytes(objectRequest);
        return objectBytes.asByteArray();

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return null;
}

// Returns the names of all images in the given bucket.
public List<String> listBucketObjects(String bucketName) {

    S3Client s3 = getClient();
    String keyName;

    List<String> keys = new ArrayList<>();

    try {
        ListObjectsRequest listObjects = ListObjectsRequest
                .builder()
                .bucket(bucketName)
                .build();

        ListObjectsResponse res = s3.listObjects(listObjects);
        List<S3Object> objects = res.contents();

        for (S3Object myValue: objects) {
            keyName = myValue.key();
            keys.add(keyName);
        }
        return keys;

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return null;
}

// Tag assets with labels in the given list.
public void tagAssets(List myList, String bucketName) {

    try {

        S3Client s3 = getClient();
        int len = myList.size();
```

```
        String assetName = "";
        String labelName = "";
        String labelValue = "";

        // Tag all the assets in the list.
        for (Object o : myList) {

            // Need to get the WorkItem from each list.
            List innerList = (List) o;
            for (Object value : innerList) {

                WorkItem workItem = (WorkItem) value;
                assetName = workItem.getKey();
                labelName = workItem.getName();
                labelValue = workItem.getConfidence();
                tagExistingObject(s3, bucketName, assetName, labelName, labelValue);
            }
        }

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

// This method tags an existing object.
private void tagExistingObject(S3Client s3, String bucketName, String key, String label,
String LabelValue) {

    try {

        // First need to get existing tag set; otherwise the existing tags are
overwritten.
        GetObjectTaggingRequest getObjectTaggingRequest =
GetObjectTaggingRequest.builder()
                .bucket(bucketName)
                .key(key)
                .build();

        GetObjectTaggingResponse response = s3.getObjectTagging(getObjectTaggingRequest);

        // Get the existing immutable list - cannot modify this list.
        List<Tag> existingList = response.tagSet();
        ArrayList<Tag> newTagList = new ArrayList(new ArrayList<>(existingList));

        // Create a new tag.
        Tag myTag = Tag.builder()
                .key(label)
                .value(LabelValue)
                .build();

        // push new tag to list.
        newTagList.add(myTag);
        Tagging tagging = Tagging.builder()
                .tagSet(newTagList)
                .build();

        PutObjectTaggingRequest taggingRequest = PutObjectTaggingRequest.builder()
                .key(key)
                .bucket(bucketName)
                .tagging(tagging)
                .build();

        s3.putObjectTagging(taggingRequest);
        System.out.println(key + " was tagged with " + label);
```

```
    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
  }

 // Delete tags from the given object.
 public void deleteTagFromObject(String bucketName, String key) {

    try {

        DeleteObjectTaggingRequest deleteObjectTaggingRequest =
 DeleteObjectTaggingRequest.builder()
                .key(key)
                .bucket(bucketName)
                .build();

        S3Client s3 = getClient();
        s3.deleteObjectTagging(deleteObjectTaggingRequest);

    } catch (S3Exception e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
  }
}
```

AnalyzePhotos class

The following Java code represents the **AnalyzePhotos** class. This class uses the Amazon Rekognition API to analyze the images.

```
package com.example.tags;

import software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsRequest;
import software.amazon.awssdk.services.rekognition.model.DetectLabelsResponse;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.ArrayList;
import java.util.List;

public class AnalyzePhotos {

 // Returns a list of WorkItem objects that contains labels.
 public ArrayList<WorkItem> detectLabels(byte[] bytes, String key) {

    Region region = Region.US_EAST_2;
    RekognitionClient rekClient = RekognitionClient.builder()
            .credentialsProvider(EnvironmentVariableCredentialsProvider.create())
            .region(region)
            .build();

    try {

        SdkBytes sourceBytes = SdkBytes.fromByteArray(bytes);

        // Create an Image object for the source image.
        Image souImage = Image.builder()
```

```
                .bytes(sourceBytes)
                .build();

        DetectLabelsRequest detectLabelsRequest = DetectLabelsRequest.builder()
                .image(souImage)
                .maxLabels(10)
                .build();

        DetectLabelsResponse labelsResponse = rekClient.detectLabels(detectLabelsRequest);

        // Write the results to a WorkItem instance.
        List<Label> labels = labelsResponse.labels();
        ArrayList<WorkItem> list = new ArrayList<>();
        WorkItem item ;
        for (Label label: labels) {
            item = new WorkItem();
            item.setKey(key); // identifies the photo.
            item.setConfidence(label.confidence().toString());
            item.setName(label.name());
            list.add(item);
        }
        return list;

    } catch (RekognitionException e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
    return null ;
  }
}
```

BucketItem class

The following Java code represents the **BucketItem** class that stores Amazon S3 object data.

```
package com.example.tags;

public class BucketItem {

 private String key;
 private String owner;
 private String date ;
 private String size ;


 public void setSize(String size) {
    this.size = size ;
 }

 public String getSize() {
    return this.size ;
 }

 public void setDate(String date) {
    this.date = date ;
 }

 public String getDate() {
    return this.date ;
 }

 public void setOwner(String owner) {
    this.owner = owner ;
 }
```

```
 public String getOwner() {
    return this.owner ;
 }

 public void setKey(String key) {
    this.key = key ;
 }

 public String getKey() {
    return this.key ;
 }
}
```

WorkItem class

The following Java code represents the **WorkItem** class.

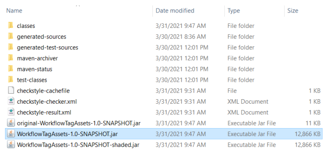
```
 package com.example.tags;

 public class WorkItem {

 private String key;
 private String name;
 private String confidence ;

public void setKey (String key) {
    this.key = key;
}

public String getKey() {
    return this.key;
}

public void setName (String name) {
    this.name = name;
}

public String getName() {
    return this.name;
}

public void setConfidence (String confidence) {
    this.confidence = confidence;
}

public String getConfidence() {
    return this.confidence;
}

}
```

Package the project

Package up the project into a .jar (JAR) file by using the following Maven command.

```
mvn package
```

The JAR file is located in the **target** folder (which is a child folder of the project folder).



Note

Notice the use of the **maven-shade-plugin** in the project's POM file. This plugin is responsible for creating a JAR that contains the required dependencies. If you attempt to package up the project without this plugin, the required dependences are not included in the JAR file and you will encounter a **ClassNotFoundException**.

Deploy the Lambda function

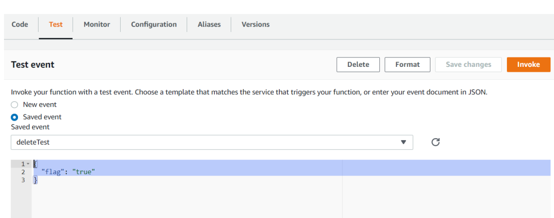
1. Open the Lambda console.
2. Choose **Create Function**.
3. Choose **Author from scratch**.
4. In the **Basic information** section, enter **cron** as the name.
5. In the **Runtime**, choose **Java 8**.
6. Choose **Use an existing role**, and then choose **lambda-support** (the IAM role that you created).
7. Choose **Create function**.
8. For **Code entry type**, choose **Upload a .zip or .jar file**.
9. Choose **Upload**, and then browse to the JAR file that you created.
10. For **Handler**, enter the fully qualified name of the function, for example, **com.example.tags.Handler:handleRequest** (**com.example.tags** specifies the package, **Handler** is the class followed by :: and method name).
11. Choose **Save**.

Test the Lambda method

At this point in the tutorial, you can test the Lambda function.

1. In the Lambda console, click the **Test** tab and then enter the following JSON.

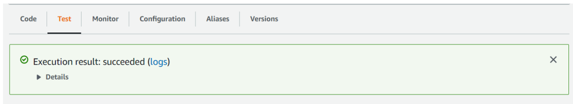
```
                  {
"flag": "true"
 }
```



Note

Passing **true** tags the digital assets and passing **false** deletes the tags.

2. Choose the **Invoke** button. After the Lambda function is invoked, you see a successful message.



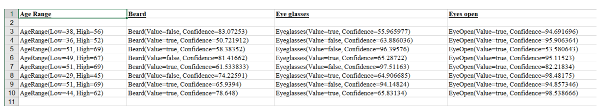
Congratulations, you have created an AWS Lambda function that automactially applies tags to difital assets located in an Amazon S3 bucket. As stated at the beginning of this tutorial, be sure to terminate all of the resources you created while going through this tutorial to ensure that you're not charged.

For more AWS multiservice examples, see the AWS Documentation SDK examples GitHub repository.

Creating AWS video analyzer applications

You can create a Java web application that analyzes videos for label detection by using the AWS SDK for Java version 2. The application created in this AWS tutorial lets you upload a video (MP4 file) to an Amazon S3 bucket. Then the appliction uses the Amazon Rekognition service to analyze the video. The results are used to populate a data model and then a report is generated and emailed to a specific user by using the Amazon Simple Email Service.

The following illustration shows a report that is generated after the application completes analyzing the video.



1	**Age Range**	**Beard**	**Eye glasses**	**Eyes open**
2				
3	AgeRange(Low=38, High=56)	Beard(Value=false, Confidence=83.07253)	Eyeglasses(Value=true, Confidence=55.965977)	EyeOpen(Value=true, Confidence=94.691696)
4	AgeRange(Low=36, High=52)	Beard(Value=true, Confidence=50.721912)	Eyeglasses(Value=false, Confidence=63.886036)	EyeOpen(Value=true, Confidence=95.906364)
5	AgeRange(Low=51, High=69)	Beard(Value=true, Confidence=58.38352)	Eyeglasses(Value=false, Confidence=96.39576)	EyeOpen(Value=true, Confidence=53.580643)
6	AgeRange(Low=49, High=67)	Beard(Value=false, Confidence=81.41662)	Eyeglasses(Value=true, Confidence=65.28722)	EyeOpen(Value=true, Confidence=95.11523)
7	AgeRange(Low=51, High=69)	Beard(Value=true, Confidence=61.533833)	Eyeglasses(Value=false, Confidence=97.51163)	EyeOpen(Value=true, Confidence=82.21834)
8	AgeRange(Low=29, High=45)	Beard(Value=false, Confidence=74.22591)	Eyeglasses(Value=true, Confidence=64.906685)	EyeOpen(Value=true, Confidence=98.48175)
9	AgeRange(Low=51, High=69)	Beard(Value=true, Confidence=65.9394)	Eyeglasses(Value=false, Confidence=94.14824)	EyeOpen(Value=true, Confidence=94.857346)
10	AgeRange(Low=44, High=62)	Beard(Value=true, Confidence=78.648)	Eyeglasses(Value=true, Confidence=65.83134)	EyeOpen(Value=true, Confidence=98.538666)
11				

In this tutorial, you create a Spring Boot application that invokes various AWS services. The Spring Boot APIs are used to build a model, different views, and a controller. For more information, see Spring Boot.

This service uses the following AWS services:

- Amazon Rekognition
- Amazon S3
- Amazon SES
- AWS Elastic Beanstalk

The AWS services included in this tutorial are included in the AWS Free Tier. We recommend that you terminate all of the resources you create in the tutorial when you are finished with them to avoid being charged.

Prerequisites

Before you begin, you need to complete the steps in Setting Up the AWS SDK for Java. Then make sure that you have the following:

- Java 1.8 JDK.
- Maven 3.6 or later.
- An Amazon S3 bucket named **video[somevalue]**. Be sure to use this bucket name in your Amazon S3 Java code. For more information, see Creating a bucket.
- An IAM role. You need this for the **VideoDetectFaces** class that you will create. For more information, see Configuring Amazon Rekognition Video.
- A valid Amazon SNS topic. You need this for the **VideoDetectFaces** class that you will create. For more information, see Configuring Amazon Rekognition Video.

Procedure

In the course of the tutorial, you do the following:

1. Create a project
2. Add the POM dependencies to your project
3. Create the Java classes
4. Create the HTML files
5. Create the script files
6. Package the project into a JAR file
7. Deploy the application to AWS Elastic Beanstalk

To proceed with the tutorial, follow the detailed instructions in the AWS Documentation SDK examples GitHub repository.

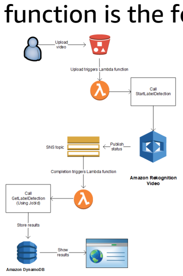
Creating an Amazon Rekognition Lambda function

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

> **Note**
> This tutorial uses the AWS SDK for Java 1.x. For a tutorial using Rekognition and the AWS SDK for Java version 2, see the AWS Documentation SDK examples GitHub repository.

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 StartLabelDetection to start detecting labels in the uploaded video. For information about using Lambda to process event notifications from an Amazon S3 bucket, see Using AWS Lambda with Amazon S3 Events.

A second Lambda function is triggered when the analysis completion status is sent to the registered Amazon SNS topic. The second Lambda function calls GetLabelDetection to get the analysis results. The results are then stored in a database in preparation for displaying on a webpage. This second lambda function is the focus of this tutorial.



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 GetLabelDetection. 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.

> **Note**
> Use the same AWS Region throughout the tutorial.

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. 62).

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.You only need to specifiy the topic name. 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 **Function name**, type a name for your function.
5. In **Runtime**, choose **Java 8**.
6. Choose **Choose or create an execution role**.
7. In **Execution role**, choose **Create a new role with basic Lambda permissions**.
8. Note the name of the new role that's displayed at the bottom of the **Basic information** section.
9. 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. 365). 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 **Edit**. The **Edit basic settings** dialog is shown.
   a. Choose **1024** for **Memory**.
   b. Choose **10** seconds for **Timeout**.
   c. Choose **Save**.
3. In **Designer**, choose **+ Add trigger**. The Add trigger dialog is shown.
4. In **Trigger configuration** choose **SNS**.

   In **SNS topic**, choose the Amazon SNS topic that you created in Create the SNS topic (p. 365).
5. Choose **Enable trigger**.
6. To add the trigger, choose **Add**.
7. Choose **Save** to save the Lambda function.

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 StartLabelDetection, 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 execution role that you created in Create the Lambda function (p. 365).
4. Choose the **Permissions** tab.
5. Choose **Attach policies**.
6. Choose *AmazonRekognitionFullAccess* from the list of policies.
7. Choose **Attach policy**.
8. Again, choose the execution role.
9. 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. 365).

```
{
    "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 GetLabelDetection 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 EclipseAWS Lambda project.
   - For **Project name:**, type a project name of your choosing.
   - For **Class Name:**, enter *JobCompletionHandler*.
   - For **Input type:**, choose **SNS Event**.
   - Leave the other fields unchanged.
2. In the **Eclipse Project** explorer, open the generated Lambda handler method (JobCompletionHandler.java) 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. Save the file.
5. Right-click in your Eclipse code window, choose **AWS Lambda**, and then choose **Upload function to AWS Lambda**.
6. On the **Select Target Lambda Function** page, choose the AWS Region to use.
7. Choose **Choose an existing lambda function**, and select the Lambda function that you created in Create the Lambda function (p. 365).
8. Choose **Next**. The **Function Configuration** dialog box is shown.
9. In **IAM Role** choose the IAM role that you created in Create the Lambda function (p. 365).
10. 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 MOV or MPEG-4 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 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"}" \
--notification-channel "SNSTopicArn=TopicARN,RoleArn=RoleARN" \
--region Region
```

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 `TopicARN` to the ARN of the Amazon SNS topic that you created in Create the SNS topic (p. 365).
- Change `RoleARN` to the ARN of the IAM role that you created in Create the SNS topic (p. 365).
- Change `Region` to the AWS Region that you are using.

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:

Time (UTC +00:00)	Message
2018-02-28	
19:48:01	START RequestId: 47cb1472-1cc0-11e8-860a-4d00aa2ff96b Version: $LATEST
19:48:02	Rekognition Video Operation:=========================
19:48:02	Job id: "9c7c3b1403a375a044c6dbe793d5c78d06014ee16f5efde083ad654b06f6c59a"
19:48:02	Status : "SUCCEEDED"
19:48:02	Job tag : null
19:48:02	Operation : "StartLabelDetection"
19:48:09	Total number of labels : 29
19:48:09	labels : Audience / Badge / Bottle / Clothing / Coat / Crowd / Electric Guitar / Flora / Food / Guitar / Hu
19:48:09	Result: {}
19:48:09	END RequestId: 47cb1472-1cc0-11e8-860a-4d00aa2ff96b

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

Using Amazon Rekognition for Identity Verification

Amazon Rekognition provides users with several operations that enable the simple creation of identity verification systems. Amazon Rekognition allows the user to detect faces in an image and then compare any detected faces to other faces by comparing face data. This face data is stored in server-side containers called Collections. By making use of Amazon Rekognition's face detection, face comparison, and collection management operations, you can create an application with an identity verification solution.

This tutorial will demonstrate two common workflows for the creation of applications requiring identity verification.

The first workflow involves the registration of a new user in a collection. The second workflow involves searching an existing collection for the purposes of logging in a returning user.

You'll use the AWS SDK for Python for this tutorial. You can also see the AWS Documentation SDK examples GitHub repo for more Python tutorials.

Topics

Prerequisites

Before you begin this tutorial, you'll need install Python and complete the steps required to set up the Python AWS SDK. Beyond this, ensure that you have:

- Created an AWS account and an IAM role
- Installed the Python SDK (Boto3)
- Properly configured your AWS access credentials
- Created a Amazon Simple Storage Service bucket and uploaded an image you wish to use as an ID for the purposes of Identity Verification.
- Have selected a second image to to serve as the target image for identity verification.

Creating a Collection

Before you can register a new user in a Collection or search a Collection for a user, you have to have a Collection to work with. An Amazon Rekognition Collection is a server-side container used to store information about detected faces.

Create the Collection

You'll start by writing a function that creates a Collection for use by your application. Amazon Rekognition stores information about faces that have been detected in server-side containers called Collections. You can search facial information stored in a Collection for known faces. To store facial information, you first need to create a Collection using the `CreateCollection` operation.

1. Select a name for the Collection that you would like to create. In the following code, replace the value of `collection_id` with the name of the Collection you'd like to create, and replace the value of `region` with the name of the region defined in your user credentials. You can use the `Tags` argument to apply any tags you would like to the collection, although this isn't required. The `CreateCollection` operation will return information about the collection you have created, including the Arn of the collection. Make a note of the Arn that you receive as a result of running the code.

```
import boto3

def create_collection(collection_id, region):
    client = boto3.client('rekognition', region_name=region)

    # Create a collection
    print('Creating collection:' + collection_id)
    response = client.create_collection(CollectionId=collection_id,
    Tags={"SampleKey1":"SampleValue1"})
    print('Collection ARN: ' + response['CollectionArn'])
    print('Status code: ' + str(response['StatusCode']))
    print('Done...')

collection_id = 'collection-id-name'
```

```
region = "region-name"
create_collection(collection_id, region)
```

2. Save and run the code. Copy down the collection Arn.

   Now that the Rekognition Collection has been created, you can store facial information and identifiers in that Collection. You will also be able to compare faces against the stored information for verification.

New User Registration

You'll want to be able to register new users and add their info to a Collection. The process of registering a new user typically involves the following steps:

Call the `DetectFaces` Operation

Write the code to check the quality of the face image via the `DetectFaces` operation. You'll use the `DetectFaces` operation to determine if an image captured by the camera is suitable for processing by the `SearchFacesByImage` operation. The image should contain only one face. You'll provide a local input image file to the `DetectFaces` operation and receive details for the faces detected in the image. The following sample code provides the input image to `DetectFaces` and then checks to see if only one face has been detected in the image.

1. In the following code example, replace `photo` with the name of the target image in which you'd like to detect faces. You'll also need to replace the value of `region` with the name of the region associated with your account.

```
import boto3
import json

def detect_faces(target_file, region):

    client=boto3.client('rekognition', region_name=region)

    imageTarget = open(target_file, 'rb')

    response = client.detect_faces(Image={'Bytes': imageTarget.read()},
    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))

        # Access predictions for individual face details and print them
        print("Gender: " + str(faceDetail['Gender']))
        print("Smile: " + str(faceDetail['Smile']))
        print("Eyeglasses: " + str(faceDetail['Eyeglasses']))
        print("Emotions: " + str(faceDetail['Emotions'][0]))

    return len(response['FaceDetails'])

photo = 'photo-name'
region = 'region-name'
face_count=detect_faces(photo, region)
print("Faces detected: " + str(face_count))
```

```
if face_count == 1:
    print("Image suitable for use in collection.")
else:
    print("Please submit an image with only one face.")
```

2. Save and run the proceeding code.

Call the `CompareFaces` Operation

Your application will need to be able to register new users in a Collection and confirm the identity of returning users. You will create the functions used to register a new user first. You'll start by using the `CompareFaces` operation to compare a local input/target image of the user and a ID/stored image. If there is a match between the face detected in both images, you can search through the Collection to see if the user has been registered in it.

Start by writing a function that compares an input image to the ID image you have stored in your Amazon S3 bucket. In the following code example, u=you will need to provide the input image yourself, which should be captured after using some form of liveness detector. You will also need to pass the name of an image stored in your Amazon S3 bucket.

1. Replace the value of `bucket` with the name of the Amazon S3 bucket containing your source file. You will also need to replace the value of `source_file` with the name of the source image you are using. Replace the value of `target_file` with the name of the target file you have provided. Replace the value of `region` with the name of the `region` defined in your user credentials.

   You will also want to specify a minimum confidence level in the match that is returned in the response, using the `similarityThreshold` argument. Detected faces will only be returned in the `FaceMatches` array if the confidence is above this threshold. Your chosen `similarityThreshold` should reflect the nature of your specific use case. Any use case involving critical security applications should use 99 as the selected threshold.

```
import boto3

def compare_faces(bucket, sourceFile, targetFile, region):
    client = boto3.client('rekognition', region_name=region)

    imageTarget = open(targetFile, 'rb')

    response = client.compare_faces(SimilarityThreshold=99,
                                    SourceImage={'S3Object':
{'Bucket':bucket,'Name':sourceFile}},
                                    TargetImage={'Bytes': imageTarget.read()})

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

    imageTarget.close()
    return len(response['FaceMatches'])

bucket = 'bucket-name'
source_file = 'source-file-name'
target_file = 'target-file-name'
region = "region-name"
face_matches = compare_faces(bucket, source_file, target_file, region)
print("Face matches: " + str(face_matches))
```

```
if str(face_matches) == "1":
    print("Face match found.")
else:
    print("No face match found.")
```

2. Save and run the proceeding code.

   You will be returned a response object containing information about the matched face and the confidence level.

Call the `SearchFacesByImage` Operation

If the confidence level of the `CompareFaces` operation is above your chosen `SimilarityThreshold`, you'll want to search your Collection for a face that might match the input image. If a match is found in your collection, that means the user is likely already registered in the Collection and there isn't a need to register a new user in your Collection. If there isn't a match, you can register the new user in your Collection.

1. Start by writing the code that will invoke the `SearchFacesByImage` operation. The operation will take in a local image file as an argument and then search your `Collection` for a face that matches the largest detected faces in the provided image.

   In the following code example, change the value of `collectionId` to the Collection that you want to search. Replace the value of `region` with the name of the region associated with your account. You'll also need to replace the value of `photo` with the name of the your input file. You will also want to specify a similarity threshold by replacing the value of `threshold` with a chosen percentile.

```
import boto3

collectionId = 'collection-id-name'
region = "region-name"
photo = 'photo-name'
threshold = 99
maxFaces = 1
client = boto3.client('rekognition', region_name=region)

# input image should be local file here, not s3 file
with open(photo, 'rb') as image:
    response = client.search_faces_by_image(CollectionId=collectionId,
    Image={'Bytes': image.read()},
    FaceMatchThreshold=threshold, MaxFaces=maxFaces)

faceMatches = response['FaceMatches']
print(faceMatches)

for match in faceMatches:
    print('FaceId:' + match['Face']['FaceId'])
    print('ImageId:' + match['Face']['ImageId'])
    print('Similarity: ' + "{:.2f}".format(match['Similarity']) + "%")
    print('Confidence: ' + str(match['Face']['Confidence']))
```

2. Save and run the proceeding code. If there's been a match, it means the person recognized in the image is already part of the Collection and there's no need to go on to the next steps. In this case, you can just allow the user access to the application.

Call the `IndexFaces` Operation

Assuming that no match was found in the Collection you searched, you will want to add the face of the user to your collection. You do this by calling the `IndexFaces` operation. When you call IndexFaces,

Amazon Rekognition extracts the facial features of a face identified in your input image, storing the data in the specified collection.

1. Begin by writing the code to call `IndexFaces`. Replace the value of `image` with the name of the local file you want to use as an input image for the IndexFaces operation. You will also need to replace the value of `photo_name` with the desired name for your input image. Be sure to replace the value of `collection_id` with the ID of the collection you previously created. Next, replace the value of `region` with the name of the region associated with your account. You will also want to specify a value for the `MaxFaces` input parameter, which defines the maximum number of faces in an image that should be indexed. The default value for this parameter is 1.

```
import boto3

def add_faces_to_collection(target_file, photo, collection_id, region):
    client = boto3.client('rekognition', region_name=region)

    imageTarget = open(target_file, 'rb')

    response = client.index_faces(CollectionId=collection_id,
                                  Image={'Bytes': imageTarget.read()},
                                  ExternalImageId=photo,
                                  MaxFaces=1,
                                  QualityFilter="AUTO",
                                  DetectionAttributes=['ALL'])
    print(response)

    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('  Image ID: {}'.format(faceRecord['Face']['ImageId']))
        print('  External Image ID: {}'.format(faceRecord['Face']['ExternalImageId']))
        print('  Confidence: {}'.format(faceRecord['Face']['Confidence']))

    print('Faces not indexed:')
    for unindexedFace in response['UnindexedFaces']:
        print(' Location: {}'.format(unindexedFace['FaceDetail']['BoundingBox']))
        print(' Reasons:')
        for reason in unindexedFace['Reasons']:
            print('   ' + reason)
    return len(response['FaceRecords'])

image = 'image-file-name'
collection_id = 'collection-id-name'
photo_name = 'desired-image-name'
region = "region-name"

indexed_faces_count = add_faces_to_collection(image, photo_name, collection_id, region)
print("Faces indexed count: " + str(indexed_faces_count))
```

2. Save and run the proceeding code. Determine if you would like to save any of the data returned by the `IndexFaces` operation, such as the FaceID assigned to the person in the image. The next section will examine how to save this data. Copy down the returned `FaceId`, `ImageId`, and `Confidence` values before proceeding.

Store Image and FaceID Data in Amazon S3 and Amazon DynamoDB

Once the Face ID for the input image has been obtained, the image data can be saved in Amazon S3, while the face data and image URL can be entered into a database like DynamoDB.

1. Write the code to upload the input image to your Amazon S3 database. In the code sample that follows, replace the value of `bucket` with the name of the bucket to which you'd like to upload the file, then replace the value of `file_name` with the name of the local file you want to store in your Amazon S3 bucket. Provide a key name that will identify the file in the Amazon S3 bucket by replacing the value of `key_name` with a name you'd like to give the image file. The file you want to upload is the same one that was defined in earlier code samples, which is the input file you used for IndexFaces. Finally, replace the value of `region` with the name of the region associated with your account.

```
import boto3
import logging
from botocore.exceptions import ClientError

# store local file in S3 bucket
bucket = "bucket-name"
file_name = "file-name"
key_name = "key-name"
region = "region-name"
s3 = boto3.client('s3', region_name=region)
# Upload the file
try:
    response = s3.upload_file(file_name, bucket, key_name)
    print("File upload successful!")
except ClientError as e:
    logging.error(e)
```

2. Save and run the proceeding code sample to upload your input image to Amazon Amazon S3.

3. You'll want to save the returned Face ID to a database as well. This can be done by creating a DynamoDB database table and then uploading the Face ID to that table. The following code sample creates a DynamoDB table. Note that you only need to run the code that creates this table once. In the following code, replace the value of `region` with the value of the region associated with your account. You will also need to replace the value of `database_name` with the name that you'd like to give the DynamoDB table.

```
import boto3

# Create DynamoDB database with image URL and face data, face ID

def create_dynamodb_table(table_name, region):
    dynamodb = boto3.client("dynamodb", region_name=region)

    table = dynamodb.create_table(
        TableName=table_name,
        KeySchema=[{
                'AttributeName': 'FaceID', 'KeyType': 'HASH'  # Partition key
                },],
            AttributeDefinitions=[
            {
                'AttributeName': 'FaceID', 'AttributeType': 'S'  }, ],
                ProvisionedThroughput={
            'ReadCapacityUnits': 10, 'WriteCapacityUnits': 10  }
    )
    print(table)
    return table
```

```
region = "region-name"
database_name = 'database-name'
dynamodb_table = create_dynamodb_table(database_name, region)
print("Table status:", dynamodb_table)
```

4. Save and run the proceeding code to create your table.
5. After creating the table, you can upload the returned FaceId to it. To do this, you'll establish a connection to the table with the Table function and then use the `put_item` function to upload the data.

   In the following code sample, replace the value of `bucket` with the name of the bucket containing the input image you uploaded to Amazon S3. You'll also need to replace the value of `file_name` with the name of the input file you uploaded to your Amazon S3 bucket and the value of `key_name` with the key that you previously used to identify the input file. Finally, replace the value of `region` with the name of the region associated with your account. These values should match the ones provided in step 1.

   The `AddDBEntry` stores the FaceId, ImageId, and Confidence values assigned to a face in a collection. Provide the function below with the values that you saved during Step 2 of the proceeding `IndexFaces` section.

```
import boto3
from pprint import pprint
from decimal import Decimal
import json

# The local file that was stored in S3 bucket
bucket = "s3-bucket-name"
file_name = "file-name"
key_name = "key-name"
region = "region-name"
# Get URL of file
file_url = "https://s3.amazonaws.com/{}/{}".format(bucket, key_name)
print(file_url)

# upload face-id, face info, and image url
def AddDBEntry(file_name, file_url, face_id, image_id, confidence):
    dynamodb = boto3.resource('dynamodb', region_name=region)
    table = dynamodb.Table('FacesDB-4')
    response = table.put_item(
       Item={
            'ExternalImageID': file_name,
            'ImageURL': file_url,
            'FaceID': face_id,
            'ImageID': image_id,
            'Confidence': json.loads(json.dumps(confidence), parse_float=Decimal)
       }
    )
    return response

# Mock values for face ID, image ID, and confidence - replace them with actual values
 from your collection results
dynamodb_resp = AddDBEntry(file_name, file_url, "FACE-ID-HERE",
    "IMAGE-ID-HERE", confidence-here)
print("Database entry successful.")
pprint(dynamodb_resp, sort_dicts=False)
```

6. Save and run the proceeding code sample to store the returned Face ID data in a table.

Existing User Login

After a user has been registered in a Collection, they can be authenticated upon their return by using the `SearchFacesByImage` operation. You will need to get an input image and then check the quality of the input image using `DetectFaces`. This determines if a suitable image has been used before running the `SearchFacesbyImage` operation.

Call the DetectFaces Operation

1. You'll use the `DetectFaces` operation to check the quality of the face image and determine if an image captured by the camera is suitable for processing by the `SearchFacesByImage` operation. The input image should contain just one face. The following code sample takes an input image and provides it to the `DetectFaces` operation.

   In the following code example, replace the value of `photo` with the name of the local target image and replace the value of `region` with the name of the region associated with your account.

```
import boto3
import json

def detect_faces(target_file, region):

    client=boto3.client('rekognition', region_name=region)

    imageTarget = open(target_file, 'rb')

    response = client.detect_faces(Image={'Bytes': imageTarget.read()},
    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))

        # Access predictions for individual face details and print them
        print("Gender: " + str(faceDetail['Gender']))
        print("Smile: " + str(faceDetail['Smile']))
        print("Eyeglasses: " + str(faceDetail['Eyeglasses']))
        print("Emotions: " + str(faceDetail['Emotions'][0]))

    return len(response['FaceDetails'])

photo = 'photo-name'
region = 'region-name'
face_count=detect_faces(photo, region)
print("Faces detected: " + str(face_count))

if face_count == 1:
    print("Image suitable for use in collection.")
else:
    print("Please submit an image with only one face.")
```

2. Save and run the code.

Call the SearchFacesByImage Operation

1. Write the code to compare the detected face against the faces in the Collection with the `SearchFacesByImage`. You'll use the code shown in the proceeding New User Registration section and provide the input image to the `SearchFacesByImage` operation.

   In the following code example, change the value of `collectionId` to the collection that you want to search. You will also change the value of `bucket` to the name of an Amazon S3 bucket and the value of `fileName` to an image file in that bucket. Replace the value of `region` with the name of the region associated with your account. You will also want to specify a similarity threshold by replacing the value of `threshold` with a chosen percentile.

```
import boto3

bucket = 'bucket-name'
collectionId = 'collection-id-name'
region = "region-name"
fileName = 'file-name'
threshold = 70
maxFaces = 1
client = boto3.client('rekognition', region_name=region)

# input image should be local file here, not s3 file
with open(fileName, 'rb') as image:
    response = client.search_faces_by_image(CollectionId=collectionId,
    Image={'Bytes': image.read()},
    FaceMatchThreshold=threshold, MaxFaces=maxFaces)
```

2. Save and run the code.

Check for the Returned FaceID and Confidence Level

You can now check for information on the matched FaceId by printing out response elements like the FaceId, Similarity, and Confidence attributes.

```
faceMatches = response['FaceMatches']
print(faceMatches)

for match in faceMatches:
    print('FaceId:' + match['Face']['FaceId'])
    print('ImageId:' + match['Face']['ImageId'])
    print('Similarity: ' + "{:.2f}".format(match['Similarity']) + "%")
    print('Confidence: ' + str(match['Face']['Confidence']))
```

Code examples for Amazon Rekognition using AWS SDKs

The following code examples show how to use Amazon Rekognition with an AWS software development kit (SDK). The code examples in this chapter are intended to supplement the code examples found throughout the rest of this guide.

The examples are divided into the following categories:

Actions

Code excerpts that show you how to call individual service functions.

Scenarios

Code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Cross-service examples

Sample applications that work across multiple AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Code examples

Actions for Amazon Rekognition using AWS SDKs

The following code examples demonstrate how to perform individual Amazon Rekognition actions with AWS SDKs. These excerpts call the Amazon Rekognition API and are not intended to be run in isolation. Each example includes a link to GitHub, where you can find instructions on how to set up and run the code in context.

The following examples include only the most commonly used actions. For a complete list, see the Amazon Rekognition API Reference.

Examples

Compare faces in an image against a reference image with Amazon Rekognition using an AWS SDK

The following code examples show how to compare faces in an image against a reference image with Amazon Rekognition.

For more information, see Comparing faces in images.

.NET

AWS SDK for .NET

Tip
To learn how to set up and run this example, see GitHub.

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

/// <summary>
/// Uses the Amazon Rekognition Service to compare faces in two images.
/// The example uses the AWS SDK for .NET 3.7 and .NET Core 5.0.
/// </summary>
public class CompareFaces
{
    public static async Task Main()
    {
        float similarityThreshold = 70F;
        string sourceImage = "source.jpg";
        string targetImage = "target.jpg";

        var 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 ex)
        {
            Console.WriteLine($"Failed to load target image: {targetImage}");
            Console.WriteLine(ex.Message);
            return;
        }
```

```
            var compareFacesRequest = new CompareFacesRequest
            {
                SourceImage = imageSource,
                TargetImage = imageTarget,
                SimilarityThreshold = similarityThreshold,
            };

            // Call operation
            var compareFacesResponse = await
rekognitionClient.CompareFacesAsync(compareFacesRequest);

            // Display results
            compareFacesResponse.FaceMatches.ForEach(match =>
            {
                ComparedFace face = match.Face;
                BoundingBox position = face.BoundingBox;
                Console.WriteLine($"Face at {position.Left} {position.Top} matches
with {match.Similarity}% confidence.");
            });

            Console.WriteLine($"Found {compareFacesResponse.UnmatchedFaces.Count}
face(s) that did not match.");
        }
    }
```

- For API details, see CompareFaces in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Tip
To learn how to set up and run this example, see GitHub.

```
   public static void compareTwoFaces(RekognitionClient rekClient, Float
similarityThreshold, String sourceImage, String targetImage) {
       try {
           InputStream sourceStream = new FileInputStream(sourceImage);
           InputStream tarStream = new FileInputStream(targetImage);
           SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
           SdkBytes targetBytes = SdkBytes.fromInputStream(tarStream);

           // Create an Image object for the source image.
           Image souImage = Image.builder()
               .bytes(sourceBytes)
               .build();

           Image tarImage = Image.builder()
               .bytes(targetBytes)
               .build();

           CompareFacesRequest facesRequest = CompareFacesRequest.builder()
               .sourceImage(souImage)
               .targetImage(tarImage)
               .similarityThreshold(similarityThreshold)
               .build();

           // Compare the two images.
           CompareFacesResponse compareFacesResult =
rekClient.compareFaces(facesRequest);
           List<CompareFacesMatch> faceDetails = compareFacesResult.faceMatches();
```

```
            for (CompareFacesMatch match: faceDetails){
                ComparedFace face= match.face();
                BoundingBox position = face.boundingBox();
                System.out.println("Face at " + position.left().toString()
                        + " " + position.top()
                        + " matches with " + face.confidence().toString()
                        + "% confidence.");

            }
            List<ComparedFace> uncompared = compareFacesResult.unmatchedFaces();
            System.out.println("There was " + uncompared.size() + " face(s) that
 did not match");
            System.out.println("Source image rotation: " +
 compareFacesResult.sourceImageOrientationCorrection());
            System.out.println("target image rotation: " +
 compareFacesResult.targetImageOrientationCorrection());

        } catch(RekognitionException | FileNotFoundException e) {
            System.out.println("Failed to load source image " + sourceImage);
            System.exit(1);
        }
    }
```

- For API details, see CompareFaces in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Tip
To learn how to set up and run this example, see GitHub.

```
suspend fun compareTwoFaces(similarityThresholdVal: Float, sourceImageVal: String,
 targetImageVal: String) {

    val sourceBytes = (File(sourceImageVal).readBytes())
    val targetBytes = (File(targetImageVal).readBytes())

    // Create an Image object for the source image.
    val souImage = Image {
        bytes = sourceBytes
    }

    val tarImage = Image {
        bytes = targetBytes
    }

    val facesRequest = CompareFacesRequest {
        sourceImage = souImage
        targetImage = tarImage
        similarityThreshold = similarityThresholdVal
    }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->

        val compareFacesResult = rekClient.compareFaces(facesRequest)
        val faceDetails = compareFacesResult.faceMatches
```

```
        if (faceDetails != null) {
            for (match: CompareFacesMatch in faceDetails) {
                val face = match.face
                val position = face?.boundingBox
                if (position != null)
                    println("Face at ${position.left} ${position.top} matches with
 ${face.confidence} % confidence.")
            }
        }

        val uncompared = compareFacesResult.unmatchedFaces
        if (uncompared != null)
            println("There was ${uncompared.size} face(s) that did not match")

        println("Source image rotation:
 ${compareFacesResult.sourceImageOrientationCorrection}")
        println("target image rotation:
 ${compareFacesResult.targetImageOrientationCorrection}")
    }
}
```

- For API details, see CompareFaces in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip
To learn how to set up and run this example, see GitHub.

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client

    def compare_faces(self, target_image, similarity):
        """
        Compares faces in the image with the largest face in the target image.

        :param target_image: The target image to compare against.
        :param similarity: Faces in the image must have a similarity value greater
                           than this value to be included in the results.
        :return: A tuple. The first element is the list of faces that match the
                 reference image. The second element is the list of faces that have
                 a similarity value below the specified threshold.
        """
        try:
            response = self.rekognition_client.compare_faces(
                SourceImage=self.image,
```

```
                TargetImage=target_image.image,
                SimilarityThreshold=similarity)
            matches = [RekognitionFace(match['Face']) for match
                       in response['FaceMatches']]
            unmatches = [RekognitionFace(face) for face in
response['UnmatchedFaces']]
            logger.info(
                "Found %s matched faces and %s unmatched faces.",
                len(matches), len(unmatches))
        except ClientError:
            logger.exception(
                "Couldn't match faces from %s to %s.", self.image_name,
                target_image.image_name)
            raise
        else:
            return matches, unmatches
```

- For API details, see CompareFaces in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Create an Amazon Rekognition collection using an AWS SDK

The following code examples show how to create an Amazon Rekognition collection.

For more information, see Creating a collection.

.NET

AWS SDK for .NET

Tip
To learn how to set up and run this example, see GitHub.

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses Amazon Rekognition to create a collection to which you can add
/// faces using the IndexFaces operation. The example was created using
/// the AWS SDK for .NET 3.7 and .NET Core 5.0.
/// </summary>
public class CreateCollection
{
    public static async Task Main()
    {
        var rekognitionClient = new AmazonRekognitionClient();

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

        var createCollectionRequest = new CreateCollectionRequest
        {
```

```
                CollectionId = collectionId,
            };

            CreateCollectionResponse createCollectionResponse = await
rekognitionClient.CreateCollectionAsync(createCollectionRequest);
            Console.WriteLine($"CollectionArn :
{createCollectionResponse.CollectionArn}");
            Console.WriteLine($"Status code :
{createCollectionResponse.StatusCode}");
        }
    }
```

- For API details, see CreateCollection in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Tip
To learn how to set up and run this example, see GitHub.

```
   public static void createMyCollection(RekognitionClient rekClient,String
collectionId ) {

       try {
           CreateCollectionRequest collectionRequest =
CreateCollectionRequest.builder()
               .collectionId(collectionId)
               .build();

           CreateCollectionResponse collectionResponse =
rekClient.createCollection(collectionRequest);
           System.out.println("CollectionArn: " +
collectionResponse.collectionArn());
           System.out.println("Status code: " +
collectionResponse.statusCode().toString());

       } catch(RekognitionException e) {
           System.out.println(e.getMessage());
           System.exit(1);
       }
   }
```

- For API details, see CreateCollection in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Tip
To learn how to set up and run this example, see GitHub.

```
suspend fun createMyCollection(collectionIdVal: String) {
```

```
    val request = CreateCollectionRequest {
        collectionId = collectionIdVal
    }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.createCollection(request)
        println("Collection ARN is ${response.collectionArn}")
        println("Status code is ${response.statusCode}")
    }
}
```

- For API details, see CreateCollection in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip
To learn how to set up and run this example, see GitHub.

```
class RekognitionCollectionManager:
    """
    Encapsulates Amazon Rekognition collection management functions.
    This class is a thin wrapper around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, rekognition_client):
        """
        Initializes the collection manager object.

        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.rekognition_client = rekognition_client

    def create_collection(self, collection_id):
        """
        Creates an empty collection.

        :param collection_id: Text that identifies the collection.
        :return: The newly created collection.
        """
        try:
            response = self.rekognition_client.create_collection(
                CollectionId=collection_id)
            response['CollectionId'] = collection_id
            collection = RekognitionCollection(response, self.rekognition_client)
            logger.info("Created collection %s.", collection_id)
        except ClientError:
            logger.exception("Couldn't create collection %s.", collection_id)
            raise
        else:
            return collection
```

- For API details, see CreateCollection in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Delete an Amazon Rekognition collection using an AWS SDK

The following code examples show how to delete an Amazon Rekognition collection.

For more information, see Deleting a collection.

.NET

AWS SDK for .NET

> **Tip**
> To learn how to set up and run this example, see GitHub.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to delete an existing collection.
    /// The example was created using the AWS SDK for .NET version 3.7 and
    /// .NET Core 5.0.
    /// </summary>
    public class DeleteCollection
    {
        public static async Task Main()
        {
            var rekognitionClient = new AmazonRekognitionClient();

            string collectionId = "MyCollection";
            Console.WriteLine("Deleting collection: " + collectionId);

            var deleteCollectionRequest = new DeleteCollectionRequest()
            {
                CollectionId = collectionId,
            };

            var deleteCollectionResponse = await
 rekognitionClient.DeleteCollectionAsync(deleteCollectionRequest);
            Console.WriteLine($"{collectionId}:
 {deleteCollectionResponse.StatusCode}");
        }
    }
```

- For API details, see DeleteCollection in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

> **Tip**
> To learn how to set up and run this example, see GitHub.

```
    public static void deleteMyCollection(RekognitionClient rekClient,String
 collectionId ) {

        try {
```

```
            DeleteCollectionRequest deleteCollectionRequest =
 DeleteCollectionRequest.builder()
                .collectionId(collectionId)
                .build();

            DeleteCollectionResponse deleteCollectionResponse =
 rekClient.deleteCollection(deleteCollectionRequest);
            System.out.println(collectionId + ": " +
 deleteCollectionResponse.statusCode().toString());

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

- For API details, see DeleteCollection in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Tip
To learn how to set up and run this example, see GitHub.

```
suspend fun deleteMyCollection(collectionIdVal: String) {

    val request = DeleteCollectionRequest {
        collectionId = collectionIdVal
    }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.deleteCollection(request)
        println("The collectionId status is ${response.statusCode}")
    }
}
```

- For API details, see DeleteCollection in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip
To learn how to set up and run this example, see GitHub.

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.
```

```
        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection['CollectionId']
        self.collection_arn, self.face_count, self.created =
self._unpack_collection(
            collection)
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get('CollectionArn'),
            collection.get('FaceCount', 0),
            collection.get('CreationTimestamp'))

    def delete_collection(self):
        """
        Deletes the collection.
        """
        try:

self.rekognition_client.delete_collection(CollectionId=self.collection_id)
            logger.info("Deleted collection %s.", self.collection_id)
            self.collection_id = None
        except ClientError:
            logger.exception("Couldn't delete collection %s.", self.collection_id)
            raise
```

- For API details, see DeleteCollection in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Delete faces from an Amazon Rekognition collection using an AWS SDK

The following code examples show how to delete faces from an Amazon Rekognition collection.

For more information, see Deleting faces from a collection.

.NET

AWS SDK for .NET

Tip
To learn how to set up and run this example, see GitHub.

```
    using System;
```

```
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to delete one or more faces from
    /// a Rekognition collection. This example was created using the AWS SDK
    /// for .NET version 3.7 and .NET Core 5.0.
    /// </summary>
    public class DeleteFaces
    {
        public static async Task Main()
        {
            string collectionId = "MyCollection";
            var faces = new List<string> { "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx" };

            var rekognitionClient = new AmazonRekognitionClient();

            var deleteFacesRequest = new DeleteFacesRequest()
            {
                CollectionId = collectionId,
                FaceIds = faces,
            };

            DeleteFacesResponse deleteFacesResponse = await
 rekognitionClient.DeleteFacesAsync(deleteFacesRequest);
            deleteFacesResponse.DeletedFaces.ForEach(face =>
            {
                Console.WriteLine($"FaceID: {face}");
            });
        }

    }
```

- For API details, see DeleteFaces in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Tip
To learn how to set up and run this example, see GitHub.

```
    public static void deleteFacesCollection(RekognitionClient rekClient,
                                             String collectionId,
                                             String faceId) {

        try {
            DeleteFacesRequest deleteFacesRequest = DeleteFacesRequest.builder()
                .collectionId(collectionId)
                .faceIds(faceId)
                .build();

            rekClient.deleteFaces(deleteFacesRequest);
            System.out.println("The face was deleted from the collection.");

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
```

```
    }
```

- For API details, see DeleteFaces in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see GitHub.

```
suspend fun deleteFacesCollection(collectionIdVal: String?, faceIdVal: String) {

    val deleteFacesRequest = DeleteFacesRequest {
        collectionId = collectionIdVal
        faceIds = listOf(faceIdVal)
    }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        rekClient.deleteFaces(deleteFacesRequest)
        println("$faceIdVal was deleted from the collection")
    }
}
```

- For API details, see DeleteFaces in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see GitHub.

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection['CollectionId']
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection)
        self.rekognition_client = rekognition_client

    @staticmethod
```

```
def _unpack_collection(collection):
    """
    Unpacks optional parts of a collection that can be returned by
    describe_collection.

    :param collection: The collection data.
    :return: A tuple of the data in the collection.
    """
    return (
        collection.get('CollectionArn'),
        collection.get('FaceCount', 0),
        collection.get('CreationTimestamp'))

def delete_faces(self, face_ids):
    """
    Deletes faces from the collection.

    :param face_ids: The list of IDs of faces to delete.
    :return: The list of IDs of faces that were deleted.
    """
    try:
        response = self.rekognition_client.delete_faces(
            CollectionId=self.collection_id, FaceIds=face_ids)
        deleted_ids = response['DeletedFaces']
        logger.info(
            "Deleted %s faces from %s.", len(deleted_ids), self.collection_id)
    except ClientError:
        logger.exception("Couldn't delete faces from %s.", self.collection_id)
        raise
    else:
        return deleted_ids
```

- For API details, see DeleteFaces in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Describe an Amazon Rekognition collection using an AWS SDK

The following code examples show how to describe an Amazon Rekognition collection.

For more information, see Describing a collection.

.NET

AWS SDK for .NET

Tip
To learn how to set up and run this example, see GitHub.

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to describe the contents of a
```

```
    /// collection. The example was created using the AWS SDK for .NET version
    /// 3.7 and .NET Core 5.0.
    /// </summary>
    public class DescribeCollection
    {
        public static async Task Main()
        {
            var rekognitionClient = new AmazonRekognitionClient();

            string collectionId = "MyCollection";
            Console.WriteLine($"Describing collection: {collectionId}");

            var describeCollectionRequest = new DescribeCollectionRequest()
            {
                CollectionId = collectionId,
            };

            var describeCollectionResponse = await
 rekognitionClient.DescribeCollectionAsync(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}");
        }
    }
```

- For API details, see DescribeCollection in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Tip

To learn how to set up and run this example, see GitHub.

```
    public static void describeColl(RekognitionClient rekClient, String
 collectionName) {

        try {
            DescribeCollectionRequest describeCollectionRequest =
 DescribeCollectionRequest.builder()
                .collectionId(collectionName)
                .build();

            DescribeCollectionResponse describeCollectionResponse =
 rekClient.describeCollection(describeCollectionRequest);
            System.out.println("Collection Arn : " +
 describeCollectionResponse.collectionARN());
            System.out.println("Created : " +
 describeCollectionResponse.creationTimestamp().toString());

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

- For API details, see DescribeCollection in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Tip
To learn how to set up and run this example, see GitHub.

```
suspend fun describeColl(collectionName: String) {

    val request = DescribeCollectionRequest {
        collectionId = collectionName
    }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.describeCollection(request)
        println("The collection Arn is ${response.collectionArn}")
        println("The collection contains this many faces ${response.faceCount}")
    }
}
```

- For API details, see DescribeCollection in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip
To learn how to set up and run this example, see GitHub.

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection['CollectionId']
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection)
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.
```

```
        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get('CollectionArn'),
            collection.get('FaceCount', 0),
            collection.get('CreationTimestamp'))

    def describe_collection(self):
        """
        Gets data about the collection from the Amazon Rekognition service.

        :return: The collection rendered as a dict.
        """
        try:
            response = self.rekognition_client.describe_collection(
                CollectionId=self.collection_id)
            # Work around capitalization of Arn vs. ARN
            response['CollectionArn'] = response.get('CollectionARN')
            (self.collection_arn, self.face_count,
             self.created) = self._unpack_collection(response)
            logger.info("Got data for collection %s.", self.collection_id)
        except ClientError:
            logger.exception("Couldn't get data for collection %s.",
 self.collection_id)
            raise
        else:
            return self.to_dict()
```

- For API details, see DescribeCollection in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Detect faces in an image with Amazon Rekognition using an AWS SDK

The following code examples show how to detect faces in an image with Amazon Rekognition.

For more information, see Detecting faces in an image.

.NET

AWS SDK for .NET

Tip
To learn how to set up and run this example, see GitHub.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to detect faces within an image
    /// stored in an Amazon Simple Storage Service (Amazon S3) bucket. This
    /// example uses the AWS SDK for .NET version 3.7 and .NET Core 5.0.
```

```
    /// </summary>
    public class DetectFaces
    {
        public static async Task Main()
        {
            string photo = "input.jpg";
            string bucket = "bucket";

            var rekognitionClient = new AmazonRekognitionClient();

            var 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 = await
 rekognitionClient.DetectFacesAsync(detectFacesRequest);
                bool hasAll = detectFacesRequest.Attributes.Contains("ALL");
                foreach (FaceDetail face in detectFacesResponse.FaceDetails)
                {
                    Console.WriteLine($"BoundingBox: top={face.BoundingBox.Left}
 left={face.BoundingBox.Top} width={face.BoundingBox.Width}
 height={face.BoundingBox.Height}");
                    Console.WriteLine($"Confidence: {face.Confidence}");
                    Console.WriteLine($"Landmarks: {face.Landmarks.Count}");
                    Console.WriteLine($"Pose: pitch={face.Pose.Pitch}
 roll={face.Pose.Roll} yaw={face.Pose.Yaw}");
                    Console.WriteLine($"Brightness:
 {face.Quality.Brightness}\tSharpness: {face.Quality.Sharpness}");

                    if (hasAll)
                    {
                        Console.WriteLine($"Estimated age is between
 {face.AgeRange.Low} and {face.AgeRange.High} years old.");
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

Display bounding box information for all faces in an image.

```
    using System;
    using System.Collections.Generic;
```

```
    using System.Drawing;
    using System.IO;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to display the details of the
    /// bounding boxes around the faces detected in an image. This example was
    /// created using the AWS SDK for .NET version 3.7 and .NET Core 5.0.
    /// </summary>
    public class ImageOrientationBoundingBox
    {
        public static async Task Main()
        {
            string photo = @"D:\Development\AWS-Examples\Rekognition
\target.jpg"; // "photo.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            var image = new Amazon.Rekognition.Model.Image();
            try
            {
                using var 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 (var imageBitmap = new 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
            {
                var detectFacesRequest = new DetectFacesRequest()
                {
                    Image = image,
                    Attributes = new List<string>() { "ALL" },
                };

                DetectFacesResponse detectFacesResponse = await
 rekognitionClient.DetectFacesAsync(detectFacesRequest);
                detectFacesResponse.FaceDetails.ForEach(face =>
                {
                    Console.WriteLine("Face:");
                    ShowBoundingBoxPositions(
                        height,
```

```
                        width,
                        face.BoundingBox,
                        detectFacesResponse.OrientationCorrection);

                    Console.WriteLine($"BoundingBox: top={face.BoundingBox.Left}
 left={face.BoundingBox.Top} width={face.BoundingBox.Width}
 height={face.BoundingBox.Height}");
                    Console.WriteLine($"The detected face is estimated to be
 between {face.AgeRange.Low} and {face.AgeRange.High} years old.\n");
                });
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }

        /// <summary>
        /// Display the bounding box information for an image.
        /// </summary>
        /// <param name="imageHeight">The height of the image.</param>
        /// <param name="imageWidth">The width of the image.</param>
        /// <param name="box">The bounding box for a face found within the image.</
param>
        /// <param name="rotation">The rotation of the face's bounding box.</param>
        public static void ShowBoundingBoxPositions(int imageHeight, int
 imageWidth, BoundingBox box, string rotation)
        {
            float left;
            float top;

            if (rotation == null)
            {
                Console.WriteLine("No 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}");
        }
    }
```

- For API details, see DetectFaces in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Tip
To learn how to set up and run this example, see GitHub.

```
   public static void detectFacesinImage(RekognitionClient rekClient,String
sourceImage ) {

       try {
           InputStream sourceStream = new FileInputStream(sourceImage);
           SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);

           // Create an Image object for the source image.
           Image souImage = Image.builder()
               .bytes(sourceBytes)
               .build();

           DetectFacesRequest facesRequest = DetectFacesRequest.builder()
               .attributes(Attribute.ALL)
               .image(souImage)
               .build();

           DetectFacesResponse facesResponse =
rekClient.detectFaces(facesRequest);
           List<FaceDetail> faceDetails = facesResponse.faceDetails();
           for (FaceDetail face : faceDetails) {
               AgeRange ageRange = face.ageRange();
               System.out.println("The detected face is estimated to be between "
                           + ageRange.low().toString() + " and " +
ageRange.high().toString()
                           + " years old.");

               System.out.println("There is a smile :
"+face.smile().value().toString());
           }

       } catch (RekognitionException | FileNotFoundException e) {
           System.out.println(e.getMessage());
           System.exit(1);
       }
    }
```

- For API details, see DetectFaces in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Tip
To learn how to set up and run this example, see GitHub.

```
suspend fun detectFacesinImage(sourceImage: String?) {

    val souImage = Image {
        bytes = (File(sourceImage).readBytes())
    }

    val request = DetectFacesRequest {
        attributes = listOf(Attribute.All)
        image = souImage
    }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectFaces(request)
        response.faceDetails?.forEach { face ->
            val ageRange = face.ageRange
            println("The detected face is estimated to be between ${ageRange?.low}
 and ${ageRange?.high} years old.")
            println("There is a smile ${face.smile?.value}")
        }
    }
}
```

- For API details, see DetectFaces in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip
To learn how to set up and run this example, see GitHub.

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client

    def detect_faces(self):
        """
        Detects faces in the image.

        :return: The list of faces found in the image.
        """
        try:
            response = self.rekognition_client.detect_faces(
```

```
            Image=self.image, Attributes=['ALL'])
        faces = [RekognitionFace(face) for face in response['FaceDetails']]
        logger.info("Detected %s faces.", len(faces))
    except ClientError:
        logger.exception("Couldn't detect faces in %s.", self.image_name)
        raise
    else:
        return faces
```

- For API details, see DetectFaces in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Detect labels in an image with Amazon Rekognition using an AWS SDK

The following code examples show how to detect labels in an image with Amazon Rekognition.

For more information, see Detecting labels in an image.

.NET

AWS SDK for .NET

Tip
To learn how to set up and run this example, see GitHub.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to detect labels within an image
    /// stored in an Amazon Simple Storage Service (Amazon S3) bucket. This
    /// example was created using the AWS SDK for .NET and .NET Core 5.0.
    /// </summary>
    public class DetectLabels
    {
        public static async Task Main()
        {
            string photo = "del_river_02092020_01.jpg"; // "input.jpg";
            string bucket = "igsmiths3photos"; // "bucket";

            var rekognitionClient = new AmazonRekognitionClient();

            var detectlabelsRequest = new DetectLabelsRequest
            {
                Image = new Image()
                {
                    S3Object = new S3Object()
                    {
                        Name = photo,
                        Bucket = bucket,
                    },
                },
                MaxLabels = 10,
```

```
                MinConfidence = 75F,
            };

            try
            {
                DetectLabelsResponse detectLabelsResponse = await
rekognitionClient.DetectLabelsAsync(detectlabelsRequest);
                Console.WriteLine("Detected labels for " + photo);
                foreach (Label label in detectLabelsResponse.Labels)
                {
                    Console.WriteLine($"Name: {label.Name} Confidence:
{label.Confidence}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

Detect labels in an image file stored on your computer.

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

    /// <summary>
    /// Uses the Amazon Rekognition Service to detect labels within an image
    /// stored locally. This example was created using the AWS SDK for .NET
    /// and .NET Core 5.0.
    /// </summary>
    public class DetectLabelsLocalFile
    {
        public static async Task Main()
        {
            string photo = "input.jpg";

            var image = new Amazon.Rekognition.Model.Image();
            try
            {
                using var 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;
            }

            var rekognitionClient = new AmazonRekognitionClient();

            var detectlabelsRequest = new DetectLabelsRequest
            {
                Image = image,
                MaxLabels = 10,
                MinConfidence = 77F,
            };
```

```
            try
            {
                DetectLabelsResponse detectLabelsResponse = await
rekognitionClient.DetectLabelsAsync(detectlabelsRequest);
                Console.WriteLine($"Detected labels for {photo}");
                foreach (Label label in detectLabelsResponse.Labels)
                {
                    Console.WriteLine($"{label.Name}: {label.Confidence}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

- For API details, see DetectLabels in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Tip
To learn how to set up and run this example, see GitHub.

```
    public static void detectImageLabels(RekognitionClient rekClient, String
sourceImage) {

        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);

            // Create an Image object for the source image.
            Image souImage = Image.builder()
                .bytes(sourceBytes)
                .build();

            DetectLabelsRequest detectLabelsRequest = DetectLabelsRequest.builder()
                .image(souImage)
                .maxLabels(10)
                .build();

            DetectLabelsResponse labelsResponse =
rekClient.detectLabels(detectLabelsRequest);
            List<Label> labels = labelsResponse.labels();
            System.out.println("Detected labels for the given photo");
            for (Label label: labels) {
                System.out.println(label.name() + ": " +
label.confidence().toString());
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

- For API details, see DetectLabels in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Tip
To learn how to set up and run this example, see GitHub.

```
suspend fun detectImageLabels(sourceImage: String) {

    val souImage = Image {
        bytes = (File(sourceImage).readBytes())
    }
    val request = DetectLabelsRequest {
        image = souImage
        maxLabels = 10
    }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectLabels(request)
        response.labels?.forEach { label ->
            println("${label.name} : ${label.confidence}")
        }
    }
}
```

- For API details, see DetectLabels in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip
To learn how to set up and run this example, see GitHub.

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client

    def detect_labels(self, max_labels):
        """
        Detects labels in the image. Labels are objects and people.
```

```
        :param max_labels: The maximum number of labels to return.
        :return: The list of labels detected in the image.
        """
        try:
            response = self.rekognition_client.detect_labels(
                Image=self.image, MaxLabels=max_labels)
            labels = [RekognitionLabel(label) for label in response['Labels']]
            logger.info("Found %s labels in %s.", len(labels), self.image_name)
        except ClientError:
            logger.info("Couldn't detect labels in %s.", self.image_name)
            raise
        else:
            return labels
```

- For API details, see DetectLabels in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Detect moderation labels in an image with Amazon Rekognition using an AWS SDK

The following code examples show how to detect moderation labels in an image with Amazon Rekognition. Moderation labels identify content that may be inappropriate for some audiences.

For more information, see Detecting inappropriate images.

.NET

AWS SDK for .NET

Tip
To learn how to set up and run this example, see GitHub.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to detect unsafe content in a
    /// JPEG or PNG format image. This example was created using the AWS SDK
    /// for .NET version 3.7 and .NET Core 5.0.
    /// </summary>
    public class DetectModerationLabels
    {
        public static async Task Main(string[] args)
        {
            string photo = "input.jpg";
            string bucket = "bucket";

            var rekognitionClient = new AmazonRekognitionClient();

            var detectModerationLabelsRequest = new DetectModerationLabelsRequest()
            {
                Image = new Image()
                {
                    S3Object = new S3Object()
```

```
                    {
                        Name = photo,
                        Bucket = bucket,
                    },
                },
                MinConfidence = 60F,
            };

            try
            {
                var detectModerationLabelsResponse = await
rekognitionClient.DetectModerationLabelsAsync(detectModerationLabelsRequest);
                Console.WriteLine("Detected labels for " + photo);
                foreach (ModerationLabel label in
detectModerationLabelsResponse.ModerationLabels)
                {
                    Console.WriteLine($"Label: {label.Name}");
                    Console.WriteLine($"Confidence: {label.Confidence}");
                    Console.WriteLine($"Parent: {label.ParentName}");
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
```

- For API details, see DetectModerationLabels in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Tip
To learn how to set up and run this example, see GitHub.

```
   public static void detectModLabels(RekognitionClient rekClient, String
sourceImage) {

       try {
           InputStream sourceStream = new FileInputStream(sourceImage);
           SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
           Image souImage = Image.builder()
               .bytes(sourceBytes)
               .build();

           DetectModerationLabelsRequest moderationLabelsRequest =
DetectModerationLabelsRequest.builder()
               .image(souImage)
               .minConfidence(60F)
               .build();

           DetectModerationLabelsResponse moderationLabelsResponse =
rekClient.detectModerationLabels(moderationLabelsRequest);
           List<ModerationLabel> labels =
moderationLabelsResponse.moderationLabels();
           System.out.println("Detected labels for image");

           for (ModerationLabel label : labels) {
               System.out.println("Label: " + label.name()
                   + "\n Confidence: " + label.confidence().toString() + "%"
```

```
                    + "\n Parent:" + label.parentName());
            }

        } catch (RekognitionException | FileNotFoundException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
```

- For API details, see DetectModerationLabels in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see GitHub.

```
suspend fun detectModLabels(sourceImage: String) {

    val myImage = Image {
        this.bytes = (File(sourceImage).readBytes())
    }

    val request = DetectModerationLabelsRequest {
        image = myImage
        minConfidence = 60f
    }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectModerationLabels(request)
        response.moderationLabels?.forEach { label ->
            println("Label: ${label.name} - Confidence: ${label.confidence} %
 Parent: ${label.parentName}")
        }
    }
}
```

- For API details, see DetectModerationLabels in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see GitHub.

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, image, image_name, rekognition_client):
        """
```

```
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client

    def detect_moderation_labels(self):
        """
        Detects moderation labels in the image. Moderation labels identify content
        that may be inappropriate for some audiences.

        :return: The list of moderation labels found in the image.
        """
        try:
            response = self.rekognition_client.detect_moderation_labels(
                Image=self.image)
            labels = [RekognitionModerationLabel(label)
                      for label in response['ModerationLabels']]
            logger.info(
                "Found %s moderation labels in %s.", len(labels), self.image_name)
        except ClientError:
            logger.exception(
                "Couldn't detect moderation labels in %s.", self.image_name)
            raise
        else:
            return labels
```

- For API details, see DetectModerationLabels in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Detect text in an image with Amazon Rekognition using an AWS SDK

The following code examples show how to detect text in an image with Amazon Rekognition.

For more information, see Detecting text in an image.

.NET

AWS SDK for .NET

Tip
To learn how to set up and run this example, see GitHub.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to detect text in an image. The
```

```
    /// example was created using the AWS SDK for .NET version 3.7 and .NET
    /// Core 5.0.
    /// </summary>
    public class DetectText
    {
        public static async Task Main()
        {
            string photo = "Dad_photographer.jpg"; // "input.jpg";
            string bucket = "igsmiths3photos"; // "bucket";

            var rekognitionClient = new AmazonRekognitionClient();

            var detectTextRequest = new DetectTextRequest()
            {
                Image = new Image()
                {
                    S3Object = new S3Object()
                    {
                        Name = photo,
                        Bucket = bucket,
                    },
                },
            };

            try
            {
                DetectTextResponse detectTextResponse = await
 rekognitionClient.DetectTextAsync(detectTextRequest);
                Console.WriteLine($"Detected lines and words for {photo}");
                detectTextResponse.TextDetections.ForEach(text =>
                {
                    Console.WriteLine($"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);
            }
        }
    }
```

- For API details, see DetectText in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Tip
To learn how to set up and run this example, see GitHub.

```
    public static void detectTextLabels(RekognitionClient rekClient, String
 sourceImage) {

        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            Image souImage = Image.builder()
                .bytes(sourceBytes)
```

```
                .build();

            DetectTextRequest textRequest = DetectTextRequest.builder()
                .image(souImage)
                .build();

            DetectTextResponse textResponse = rekClient.detectText(textRequest);
            List<TextDetection> textCollection = textResponse.textDetections();
            System.out.println("Detected lines and words");
            for (TextDetection text: textCollection) {
                System.out.println("Detected: " + text.detectedText());
                System.out.println("Confidence: " + text.confidence().toString());
                System.out.println("Id : " + text.id());
                System.out.println("Parent Id: " + text.parentId());
                System.out.println("Type: " + text.type());
                System.out.println();
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

- For API details, see DetectText in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Tip
To learn how to set up and run this example, see GitHub.

```
suspend fun detectTextLabels(sourceImage: String?) {

    val souImage = Image {
        bytes = (File(sourceImage).readBytes())
    }

    val request = DetectTextRequest {
        image = souImage
    }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectText(request)
        response.textDetections?.forEach { text ->
            println("Detected: ${text.detectedText}")
            println("Confidence: ${text.confidence}")
            println("Id: ${text.id}")
            println("Parent Id:  ${text.parentId}")
            println("Type: ${text.type}")
        }
    }
}
```

- For API details, see DetectText in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip
To learn how to set up and run this example, see GitHub.

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client

    def detect_text(self):
        """
        Detects text in the image.

        :return The list of text elements found in the image.
        """
        try:
            response = self.rekognition_client.detect_text(Image=self.image)
            texts = [RekognitionText(text) for text in response['TextDetections']]
            logger.info("Found %s texts in %s.", len(texts), self.image_name)
        except ClientError:
            logger.exception("Couldn't detect text in %s.", self.image_name)
            raise
        else:
            return texts
```

- For API details, see DetectText in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Get information about celebrities with Amazon Rekognition using an AWS SDK

The following code example shows how to get information about celebrities using Amazon Rekognition.

.NET

AWS SDK for .NET

Tip
To learn how to set up and run this example, see GitHub.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Shows how to use Amazon Rekognition to retrieve information about the
    /// celebrity identified by the supplied celebrity Id. This example was
    /// created using the AWS SDK for .NET version 3.7 and .NET Core 5.0.
    /// </summary>
    public class CelebrityInfo
    {
        public static async Task Main()
        {
            string celebId = "nnnnnnnn";

            var rekognitionClient = new AmazonRekognitionClient();

            var celebrityInfoRequest = new GetCelebrityInfoRequest
            {
                Id = celebId,
            };

            Console.WriteLine($"Getting information for celebrity: {celebId}");

            var celebrityInfoResponse = await
 rekognitionClient.GetCelebrityInfoAsync(celebrityInfoRequest);

            // Display celebrity information.
            Console.WriteLine($"celebrity name: {celebrityInfoResponse.Name}");
            Console.WriteLine("Further information (if available):");
            celebrityInfoResponse.Urls.ForEach(url =>
            {
                Console.WriteLine(url);
            });
        }
    }
```

- For API details, see GetCelebrityInfo in *AWS SDK for .NET API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Index faces to an Amazon Rekognition collection using an AWS SDK

The following code examples show how to index faces in an image and add them to an Amazon Rekognition collection.

For more information, see Adding faces to a collection.

.NET

AWS SDK for .NET

Tip
To learn how to set up and run this example, see GitHub.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to detect faces in an image
/// that has been uploaded to an Amazon Simple Storage Service (Amazon S3)
/// bucket and then adds the information to a collection. The example was
/// created using the AWS SDK for .NET and .NET Core 5.0.
/// </summary>
public class AddFaces
{
    public static async Task Main()
    {
        string collectionId = "MyCollection2";
        string bucket = "doc-example-bucket";
        string photo = "input.jpg";

        var rekognitionClient = new AmazonRekognitionClient();

        var image = new Image
        {
            S3Object = new S3Object
            {
                Bucket = bucket,
                Name = photo,
            },
        };

        var indexFacesRequest = new IndexFacesRequest
        {
            Image = image,
            CollectionId = collectionId,
            ExternalImageId = photo,
            DetectionAttributes = new List<string>() { "ALL" },
        };

        IndexFacesResponse indexFacesResponse = await
rekognitionClient.IndexFacesAsync(indexFacesRequest);

        Console.WriteLine($"{photo} added");
        foreach (FaceRecord faceRecord in indexFacesResponse.FaceRecords)
        {
            Console.WriteLine($"Face detected: Faceid is
{faceRecord.Face.FaceId}");
        }
    }
}
```

- For API details, see IndexFaces in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Tip
To learn how to set up and run this example, see GitHub.

```
    public static void addToCollection(RekognitionClient rekClient, String
 collectionId, String sourceImage) {

        try {
            InputStream sourceStream = new FileInputStream(sourceImage);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            Image souImage = Image.builder()
                .bytes(sourceBytes)
                .build();

            IndexFacesRequest facesRequest = IndexFacesRequest.builder()
                .collectionId(collectionId)
                .image(souImage)
                .maxFaces(1)
                .qualityFilter(QualityFilter.AUTO)
                .detectionAttributes(Attribute.DEFAULT)
                .build();

            IndexFacesResponse facesResponse = rekClient.indexFaces(facesRequest);
            System.out.println("Results for the image");
            System.out.println("\n Faces indexed:");
            List<FaceRecord> faceRecords = facesResponse.faceRecords();
            for (FaceRecord faceRecord : faceRecords) {
                System.out.println("  Face ID: " + faceRecord.face().faceId());
                System.out.println("  Location:" +
 faceRecord.faceDetail().boundingBox().toString());
            }

            List<UnindexedFace> unindexedFaces = facesResponse.unindexedFaces();
            System.out.println("Faces not indexed:");
            for (UnindexedFace unindexedFace : unindexedFaces) {
                System.out.println("  Location:" +
 unindexedFace.faceDetail().boundingBox().toString());
                System.out.println("  Reasons:");
                for (Reason reason : unindexedFace.reasons()) {
                    System.out.println("Reason:  " + reason);
                }
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

- For API details, see IndexFaces in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Tip
To learn how to set up and run this example, see GitHub.

```
suspend fun addToCollection(collectionIdVal: String?, sourceImage: String) {

    val souImage = Image {
```

```
        bytes = (File(sourceImage).readBytes())
    }

    val request = IndexFacesRequest {
        collectionId = collectionIdVal
        image = souImage
        maxFaces = 1
        qualityFilter = QualityFilter.Auto
        detectionAttributes = listOf(Attribute.Default)
    }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val facesResponse = rekClient.indexFaces(request)

        // Display the results.
        println("Results for the image")
        println("\n Faces indexed:")
        facesResponse.faceRecords?.forEach { faceRecord ->
            println("Face ID: ${faceRecord.face?.faceId}")
            println("Location: ${faceRecord.faceDetail?.boundingBox}")
        }

        println("Faces not indexed:")
        facesResponse.unindexedFaces?.forEach { unindexedFace ->
            println("Location: ${unindexedFace.faceDetail?.boundingBox}")
            println("Reasons:")

            unindexedFace.reasons?.forEach { reason ->
                println("Reason:  $reason")
            }
        }
    }
}
```

- For API details, see IndexFaces in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip
To learn how to set up and run this example, see GitHub.

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection['CollectionId']
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection)
        self.rekognition_client = rekognition_client
```

```
    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get('CollectionArn'),
            collection.get('FaceCount', 0),
            collection.get('CreationTimestamp'))

    def index_faces(self, image, max_faces):
        """
        Finds faces in the specified image, indexes them, and stores them in the
        collection.

        :param image: The image to index.
        :param max_faces: The maximum number of faces to index.
        :return: A tuple. The first element is a list of indexed faces.
                 The second element is a list of faces that couldn't be indexed.
        """
        try:
            response = self.rekognition_client.index_faces(
                CollectionId=self.collection_id, Image=image.image,
                ExternalImageId=image.image_name, MaxFaces=max_faces,
                DetectionAttributes=['ALL'])
            indexed_faces = [
                RekognitionFace({**face['Face'], **face['FaceDetail']})
                for face in response['FaceRecords']]
            unindexed_faces = [
                RekognitionFace(face['FaceDetail'])
                for face in response['UnindexedFaces']]
            logger.info(
                "Indexed %s faces in %s. Could not index %s faces.",
 len(indexed_faces),
                image.image_name, len(unindexed_faces))
        except ClientError:
            logger.exception("Couldn't index faces in image %s.", image.image_name)
            raise
        else:
            return indexed_faces, unindexed_faces
```

- For API details, see IndexFaces in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

List Amazon Rekognition collections using an AWS SDK

The following code examples show how to list Amazon Rekognition collections.

For more information, see Listing collections.

.NET

AWS SDK for .NET

Tip
To learn how to set up and run this example, see GitHub.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazpon Rekognition Service to list the collection IDs in the
    /// default user account. This example was crated using the AWS SDK for
    /// .NET version 3.7 and .NET Core 5.0.
    /// </summary>
    public class ListCollections
    {
        public static async Task Main()
        {
            var rekognitionClient = new AmazonRekognitionClient();

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

            var listCollectionsRequest = new ListCollectionsRequest
            {
                MaxResults = limit,
            };

            var listCollectionsResponse = new ListCollectionsResponse();

            do
            {
                if (listCollectionsResponse is not null)
                {
                    listCollectionsRequest.NextToken =
 listCollectionsResponse.NextToken;
                }

                listCollectionsResponse = await
 rekognitionClient.ListCollectionsAsync(listCollectionsRequest);

                listCollectionsResponse.CollectionIds.ForEach(id =>
                {
                    Console.WriteLine(id);
                });
            }
            while (listCollectionsResponse.NextToken is not null);
        }
    }
```

- For API details, see ListCollections in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Tip
To learn how to set up and run this example, see GitHub.

```
    public static void listAllCollections(RekognitionClient rekClient) {
        try {
            ListCollectionsRequest listCollectionsRequest =
 ListCollectionsRequest.builder()
                .maxResults(10)
                .build();

            ListCollectionsResponse response =
 rekClient.listCollections(listCollectionsRequest);
            List<String> collectionIds = response.collectionIds();
            for (String resultId : collectionIds) {
                System.out.println(resultId);
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

- For API details, see ListCollections in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note

This is prerelease documentation for a feature in preview release. It is subject to change.

Tip

To learn how to set up and run this example, see GitHub.

```
suspend fun listAllCollections() {

    val request = ListCollectionsRequest {
        maxResults = 10
    }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.listCollections(request)
        response.collectionIds?.forEach { resultId ->
            println(resultId)
        }
    }
}
```

- For API details, see ListCollections in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see GitHub.

```
class RekognitionCollectionManager:
    """
```

```
    Encapsulates Amazon Rekognition collection management functions.
    This class is a thin wrapper around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, rekognition_client):
        """
        Initializes the collection manager object.

        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.rekognition_client = rekognition_client

    def list_collections(self, max_results):
        """
        Lists collections for the current account.

        :param max_results: The maximum number of collections to return.
        :return: The list of collections for the current account.
        """
        try:
            response =
 self.rekognition_client.list_collections(MaxResults=max_results)
            collections = [
                RekognitionCollection({'CollectionId': col_id},
 self.rekognition_client)
                for col_id in response['CollectionIds']]
        except ClientError:
            logger.exception("Couldn't list collections.")
            raise
        else:
            return collections
```

- For API details, see ListCollections in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

List faces in an Amazon Rekognition collection using an AWS SDK

The following code examples show how to list faces in an Amazon Rekognition collection.

For more information, see Listing faces in a collection.

.NET

AWS SDK for .NET

Tip
To learn how to set up and run this example, see GitHub.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to retrieve the list of faces
    /// stored in a collection. The example was created using AWS SDK for
```

```
    /// .NET version 3.7 and .NET Core 5.0.
    /// </summary>
    public class ListFaces
    {
        public static async Task Main()
        {
            string collectionId = "MyCollection2";

            var rekognitionClient = new AmazonRekognitionClient();

            var listFacesResponse = new ListFacesResponse();
            Console.WriteLine($"Faces in collection {collectionId}");

            var listFacesRequest = new ListFacesRequest
            {
                CollectionId = collectionId,
                MaxResults = 1,
            };

            do
            {
                listFacesResponse = await
 rekognitionClient.ListFacesAsync(listFacesRequest);
                listFacesResponse.Faces.ForEach(face =>
                {
                    Console.WriteLine(face.FaceId);
                });

                listFacesRequest.NextToken = listFacesResponse.NextToken;
            }
            while (!string.IsNullOrEmpty(listFacesResponse.NextToken));
        }
    }
```

- For API details, see ListFaces in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Tip
To learn how to set up and run this example, see GitHub.

```
    public static void listFacesCollection(RekognitionClient rekClient, String
 collectionId ) {
        try {
            ListFacesRequest facesRequest = ListFacesRequest.builder()
                .collectionId(collectionId)
                .maxResults(10)
                .build();

            ListFacesResponse facesResponse = rekClient.listFaces(facesRequest);
            List<Face> faces = facesResponse.faces();
            for (Face face: faces) {
                System.out.println("Confidence level there is a face:
 "+face.confidence());
                System.out.println("The face Id value is "+face.faceId());
            }

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
```

```
        }
    }
```

- For API details, see ListFaces in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Tip
To learn how to set up and run this example, see GitHub.

```
suspend fun listFacesCollection(collectionIdVal: String?) {

    val request = ListFacesRequest {
        collectionId = collectionIdVal
        maxResults = 10
    }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.listFaces(request)
        response.faces?.forEach { face ->
            println("Confidence level there is a face: ${face.confidence}")
            println("The face Id value is ${face.faceId}")
        }
    }
}
```

- For API details, see ListFaces in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip
To learn how to set up and run this example, see GitHub.

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection['CollectionId']
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection)
```

```
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get('CollectionArn'),
            collection.get('FaceCount', 0),
            collection.get('CreationTimestamp'))

    def list_faces(self, max_results):
        """
        Lists the faces currently indexed in the collection.

        :param max_results: The maximum number of faces to return.
        :return: The list of faces in the collection.
        """
        try:
            response = self.rekognition_client.list_faces(
                CollectionId=self.collection_id, MaxResults=max_results)
            faces = [RekognitionFace(face) for face in response['Faces']]
            logger.info(
                "Found %s faces in collection %s.", len(faces), self.collection_id)
        except ClientError:
            logger.exception(
                "Couldn't list faces in collection %s.", self.collection_id)
            raise
        else:
            return faces
```

- For API details, see ListFaces in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Recognize celebrities in an image with Amazon Rekognition using an AWS SDK

The following code examples show how to recognize celebrities in an image with Amazon Rekognition.

For more information, see Recognizing celebrities in an image.

.NET

AWS SDK for .NET

Tip
To learn how to set up and run this example, see GitHub.

```
    using System;
    using System.IO;
    using System.Threading.Tasks;
```

```
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Shows how to use Amazon Rekognition to identify celebrities in a photo.
    /// This example was created using the AWS SDK for .NET version 3.7 and
    /// .NET Core 5.0.
    /// </summary>
    public class CelebritiesInImage
    {
        public static async Task Main(string[] args)
        {
            string photo = "moviestars.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            var recognizeCelebritiesRequest = new RecognizeCelebritiesRequest();

            var img = new Amazon.Rekognition.Model.Image();
            byte[] data = null;
            try
            {
                using var 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");

            var recognizeCelebritiesResponse = await
rekognitionClient.RecognizeCelebritiesAsync(recognizeCelebritiesRequest);

            Console.WriteLine($"{recognizeCelebritiesResponse.CelebrityFaces.Count}
celebrity(s) were recognized.\n");
            recognizeCelebritiesResponse.CelebrityFaces.ForEach(celeb =>
            {
                Console.WriteLine($"Celebrity recognized: {celeb.Name}");
                Console.WriteLine($"Celebrity ID: {celeb.Id}");
                BoundingBox boundingBox = celeb.Face.BoundingBox;
                Console.WriteLine($"position: {boundingBox.Left}
{boundingBox.Top}");
                Console.WriteLine("Further information (if available):");
                celeb.Urls.ForEach(url =>
                {
                    Console.WriteLine(url);
                });
            });


Console.WriteLine($"{recognizeCelebritiesResponse.UnrecognizedFaces.Count} face(s)
were unrecognized.");
        }
    }
```

- For API details, see RecognizeCelebrities in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

Tip
To learn how to set up and run this example, see GitHub.

```
   public static void recognizeAllCelebrities(RekognitionClient rekClient, String
sourceImage) {

       try {
           InputStream sourceStream = new FileInputStream(sourceImage);
           SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
           Image souImage = Image.builder()
               .bytes(sourceBytes)
               .build();

           RecognizeCelebritiesRequest request =
RecognizeCelebritiesRequest.builder()
               .image(souImage)
               .build();

           RecognizeCelebritiesResponse result =
rekClient.recognizeCelebrities(request) ;
           List<Celebrity> celebs=result.celebrityFaces();
           System.out.println(celebs.size() + " celebrity(s) were recognized.\n");
           for (Celebrity celebrity: celebs) {
               System.out.println("Celebrity recognized: " + celebrity.name());
               System.out.println("Celebrity ID: " + celebrity.id());

               System.out.println("Further information (if available):");
               for (String url: celebrity.urls()){
                   System.out.println(url);
               }
               System.out.println();
           }
           System.out.println(result.unrecognizedFaces().size() + " face(s) were
unrecognized.");

       } catch (RekognitionException | FileNotFoundException e) {
           System.out.println(e.getMessage());
           System.exit(1);
       }
   }
```

- For API details, see RecognizeCelebrities in *AWS SDK for Java 2.x API Reference*.

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Tip
To learn how to set up and run this example, see GitHub.

```
suspend fun recognizeAllCelebrities(sourceImage: String?) {
```

```
    val souImage = Image {
        bytes = (File(sourceImage).readBytes())
    }

    val request = RecognizeCelebritiesRequest {
        image = souImage
    }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.recognizeCelebrities(request)
        response.celebrityFaces?.forEach { celebrity ->
            println("Celebrity recognized: ${celebrity.name}")
            println("Celebrity ID:${celebrity.id}")
            println("Further information (if available):")
            celebrity.urls?.forEach { url ->
                println(url)
            }
        }
        println("${response.unrecognizedFaces?.size} face(s) were unrecognized.")
    }
}
```

- For API details, see RecognizeCelebrities in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see GitHub.

```
class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client

    def recognize_celebrities(self):
        """
        Detects celebrities in the image.

        :return: A tuple. The first element is the list of celebrities found in
                 the image. The second element is the list of faces that were
                 detected but did not match any known celebrities.
        """
        try:
            response = self.rekognition_client.recognize_celebrities(
                Image=self.image)
            celebrities = [RekognitionCelebrity(celeb)
```

```
                        for celeb in response['CelebrityFaces']]
            other_faces = [RekognitionFace(face)
                           for face in response['UnrecognizedFaces']]
            logger.info(
                "Found %s celebrities and %s other faces in %s.", len(celebrities),
                len(other_faces), self.image_name)
        except ClientError:
            logger.exception("Couldn't detect celebrities in %s.", self.image_name)
            raise
        else:
            return celebrities, other_faces
```

- For API details, see RecognizeCelebrities in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Search for faces in an Amazon Rekognition collection using an AWS SDK

The following code examples show how to search for faces in an Amazon Rekognition collection that match another face from the collection.

For more information, see Searching for a face (face ID).

.NET

AWS SDK for .NET

Tip
To learn how to set up and run this example, see GitHub.

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to find faces in an image that
/// match the face Id provided in the method request. This example was
/// created using the AWS SDK for .NET version 3.7 and .NET Core 5.0.
/// </summary>
public class SearchFacesMatchingId
{
    public static async Task Main()
    {
        string collectionId = "MyCollection";
        string faceId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

        var rekognitionClient = new AmazonRekognitionClient();

        // Search collection for faces matching the face id.
        var searchFacesRequest = new SearchFacesRequest
        {
            CollectionId = collectionId,
            FaceId = faceId,
            FaceMatchThreshold = 70F,
```

```
                MaxFaces = 2,
            };

            SearchFacesResponse searchFacesResponse = await
rekognitionClient.SearchFacesAsync(searchFacesRequest);

            Console.WriteLine("Face matching faceId " + faceId);

            Console.WriteLine("Matche(s): ");
            searchFacesResponse.FaceMatches.ForEach(face =>
            {
                Console.WriteLine($"FaceId: {face.Face.FaceId} Similarity:
{face.Similarity}");
            });
        }
    }
```

- For API details, see SearchFaces in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

> **Tip**
> To learn how to set up and run this example, see GitHub.

```
    public static void searchFaceInCollection(RekognitionClient rekClient,String
collectionId, String sourceImage) {

        try {
            InputStream sourceStream = new FileInputStream(new File(sourceImage));
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);
            Image souImage = Image.builder()
                .bytes(sourceBytes)
                .build();

            SearchFacesByImageRequest facesByImageRequest =
SearchFacesByImageRequest.builder()
                .image(souImage)
                .maxFaces(10)
                .faceMatchThreshold(70F)
                .collectionId(collectionId)
                .build();

            SearchFacesByImageResponse imageResponse =
rekClient.searchFacesByImage(facesByImageRequest) ;
            System.out.println("Faces matching in the collection");
            List<FaceMatch> faceImageMatches = imageResponse.faceMatches();
            for (FaceMatch face: faceImageMatches) {
                System.out.println("The similarity level is  "+face.similarity());
                System.out.println();
            }

        } catch (RekognitionException | FileNotFoundException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

- For API details, see SearchFaces in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

Tip
To learn how to set up and run this example, see GitHub.

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection['CollectionId']
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection)
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get('CollectionArn'),
            collection.get('FaceCount', 0),
            collection.get('CreationTimestamp'))

    def search_faces(self, face_id, threshold, max_faces):
        """
        Searches for faces in the collection that match another face from the
        collection.

        :param face_id: The ID of the face in the collection to search for.
        :param threshold: The match confidence must be greater than this value
                          for a face to be included in the results.
        :param max_faces: The maximum number of faces to return.
        :return: The list of matching faces found in the collection. This list does
                 not contain the face specified by `face_id`.
        """
        try:
            response = self.rekognition_client.search_faces(
                CollectionId=self.collection_id, FaceId=face_id,
                FaceMatchThreshold=threshold, MaxFaces=max_faces)
            faces = [RekognitionFace(face['Face']) for face in
 response['FaceMatches']]
            logger.info(
                "Found %s faces in %s that match %s.", len(faces),
 self.collection_id,
                face_id)
        except ClientError:
            logger.exception(
```

```
                "Couldn't search for faces in %s that match %s.",
self.collection_id,
                face_id)
            raise
        else:
            return faces
```

- For API details, see SearchFaces in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Search for faces in an Amazon Rekognition collection compared to a reference image using an AWS SDK

The following code examples show how to search for faces in an Amazon Rekognition collection compared to a reference image.

For more information, see Searching for a face (image).

.NET

AWS SDK for .NET

Tip
To learn how to set up and run this example, see GitHub.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses the Amazon Rekognition Service to search for images matching those
    /// in a collection. The example was created using the AWS SDK for .NET
    /// version 3.7 and .NET Core 5.0.
    /// </summary>
    public class SearchFacesMatchingImage
    {
        public static async Task Main()
        {
            string collectionId = "MyCollection";
            string bucket = "bucket";
            string photo = "input.jpg";

            var rekognitionClient = new AmazonRekognitionClient();

            // Get an image object from S3 bucket.
            var image = new Image()
            {
                S3Object = new S3Object()
                {
                    Bucket = bucket,
                    Name = photo,
                },
            };
```

```
            var searchFacesByImageRequest = new SearchFacesByImageRequest()
            {
                CollectionId = collectionId,
                Image = image,
                FaceMatchThreshold = 70F,
                MaxFaces = 2,
            };

            SearchFacesByImageResponse searchFacesByImageResponse = await
rekognitionClient.SearchFacesByImageAsync(searchFacesByImageRequest);

            Console.WriteLine("Faces matching largest face in image from " +
photo);
            searchFacesByImageResponse.FaceMatches.ForEach(face =>
            {
                Console.WriteLine($"FaceId: {face.Face.FaceId}, Similarity:
{face.Similarity}");
            });
        }
    }
```

- For API details, see SearchFacesByImage in *AWS SDK for .NET API Reference*.

Java

SDK for Java 2.x

> **Tip**
> To learn how to set up and run this example, see GitHub.

```
   public static void searchFacebyId(RekognitionClient rekClient,String
collectionId, String faceId) {

       try {
           SearchFacesRequest searchFacesRequest = SearchFacesRequest.builder()
               .collectionId(collectionId)
               .faceId(faceId)
               .faceMatchThreshold(70F)
               .maxFaces(2)
               .build();

           SearchFacesResponse imageResponse =
rekClient.searchFaces(searchFacesRequest) ;
           System.out.println("Faces matching in the collection");
           List<FaceMatch> faceImageMatches = imageResponse.faceMatches();
           for (FaceMatch face: faceImageMatches) {
               System.out.println("The similarity level is  "+face.similarity());
               System.out.println();
           }

       } catch (RekognitionException e) {
           System.out.println(e.getMessage());
           System.exit(1);
       }
   }
```

- For API details, see SearchFacesByImage in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

Tip

To learn how to set up and run this example, see GitHub.

```
class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection['CollectionId']
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection)
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get('CollectionArn'),
            collection.get('FaceCount', 0),
            collection.get('CreationTimestamp'))

    def search_faces_by_image(self, image, threshold, max_faces):
        """
        Searches for faces in the collection that match the largest face in the
        reference image.

        :param image: The image that contains the reference face to search for.
        :param threshold: The match confidence must be greater than this value
                          for a face to be included in the results.
        :param max_faces: The maximum number of faces to return.
        :return: A tuple. The first element is the face found in the reference
 image.
                 The second element is the list of matching faces found in the
                 collection.
        """
        try:
            response = self.rekognition_client.search_faces_by_image(
                CollectionId=self.collection_id, Image=image.image,
                FaceMatchThreshold=threshold, MaxFaces=max_faces)
            image_face = RekognitionFace({
                'BoundingBox': response['SearchedFaceBoundingBox'],
                'Confidence': response['SearchedFaceConfidence']
            })
            collection_faces = [
                RekognitionFace(face['Face']) for face in response['FaceMatches']]
```

```
            logger.info("Found %s faces in the collection that match the largest "
                        "face in %s.", len(collection_faces), image.image_name)
        except ClientError:
            logger.exception(
                "Couldn't search for faces in %s that match %s.",
self.collection_id,
                image.image_name)
            raise
        else:
            return image_face, collection_faces
```

- For API details, see SearchFacesByImage in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Scenarios for Amazon Rekognition using AWS SDKs

The following code examples show you how to implement common scenarios in Amazon Rekognition with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within Amazon Rekognition. Each scenario includes a link to GitHub, where you can find instructions on how to set up and run the code.

Examples

Build an Amazon Rekognition collection and find faces in it using an AWS SDK

The following code example shows how to:

- Create an Amazon Rekognition collection.
- Add images to the collection and detect faces in it.
- Search the collection for faces that match a reference image.
- Delete a collection.

For more information, see Searching faces in a collection.

Python

SDK for Python (Boto3)

Tip
To learn how to set up and run this example, see GitHub.

Create classes that wrap Amazon Rekognition functions.

```
import logging
from pprint import pprint
import boto3
from botocore.exceptions import ClientError
from rekognition_objects import RekognitionFace
from rekognition_image_detection import RekognitionImage

logger = logging.getLogger(__name__)


class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client

    @classmethod
    def from_file(cls, image_file_name, rekognition_client, image_name=None):
        """
        Creates a RekognitionImage object from a local file.

        :param image_file_name: The file name of the image. The file is opened and
 its
                                bytes are read.
        :param rekognition_client: A Boto3 Rekognition client.
        :param image_name: The name of the image. If this is not specified, the
                           file name is used as the image name.
        :return: The RekognitionImage object, initialized with image bytes from the
                 file.
        """
        with open(image_file_name, 'rb') as img_file:
            image = {'Bytes': img_file.read()}
        name = image_file_name if image_name is None else image_name
        return cls(image, name, rekognition_client)

class RekognitionCollectionManager:
    """
    Encapsulates Amazon Rekognition collection management functions.
    This class is a thin wrapper around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, rekognition_client):
        """
        Initializes the collection manager object.

        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.rekognition_client = rekognition_client

    def create_collection(self, collection_id):
        """
        Creates an empty collection.

        :param collection_id: Text that identifies the collection.
```

```
        :return: The newly created collection.
        """
        try:
            response = self.rekognition_client.create_collection(
                CollectionId=collection_id)
            response['CollectionId'] = collection_id
            collection = RekognitionCollection(response, self.rekognition_client)
            logger.info("Created collection %s.", collection_id)
        except ClientError:
            logger.exception("Couldn't create collection %s.", collection_id)
            raise
        else:
            return collection

    def list_collections(self, max_results):
        """
        Lists collections for the current account.

        :param max_results: The maximum number of collections to return.
        :return: The list of collections for the current account.
        """
        try:
            response =
 self.rekognition_client.list_collections(MaxResults=max_results)
            collections = [
                RekognitionCollection({'CollectionId': col_id},
 self.rekognition_client)
                for col_id in response['CollectionIds']]
        except ClientError:
            logger.exception("Couldn't list collections.")
            raise
        else:
            return collections

class RekognitionCollection:
    """
    Encapsulates an Amazon Rekognition collection. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, collection, rekognition_client):
        """
        Initializes a collection object.

        :param collection: Collection data in the format returned by a call to
                           create_collection.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.collection_id = collection['CollectionId']
        self.collection_arn, self.face_count, self.created =
 self._unpack_collection(
            collection)
        self.rekognition_client = rekognition_client

    @staticmethod
    def _unpack_collection(collection):
        """
        Unpacks optional parts of a collection that can be returned by
        describe_collection.

        :param collection: The collection data.
        :return: A tuple of the data in the collection.
        """
        return (
            collection.get('CollectionArn'),
            collection.get('FaceCount', 0),
            collection.get('CreationTimestamp'))
```

```
    def to_dict(self):
        """
        Renders parts of the collection data to a dict.

        :return: The collection data as a dict.
        """
        rendering = {
            'collection_id': self.collection_id,
            'collection_arn': self.collection_arn,
            'face_count': self.face_count,
            'created': self.created
        }
        return rendering

    def describe_collection(self):
        """
        Gets data about the collection from the Amazon Rekognition service.

        :return: The collection rendered as a dict.
        """
        try:
            response = self.rekognition_client.describe_collection(
                CollectionId=self.collection_id)
            # Work around capitalization of Arn vs. ARN
            response['CollectionArn'] = response.get('CollectionARN')
            (self.collection_arn, self.face_count,
             self.created) = self._unpack_collection(response)
            logger.info("Got data for collection %s.", self.collection_id)
        except ClientError:
            logger.exception("Couldn't get data for collection %s.",
 self.collection_id)
            raise
        else:
            return self.to_dict()

    def delete_collection(self):
        """
        Deletes the collection.
        """
        try:

 self.rekognition_client.delete_collection(CollectionId=self.collection_id)
            logger.info("Deleted collection %s.", self.collection_id)
            self.collection_id = None
        except ClientError:
            logger.exception("Couldn't delete collection %s.", self.collection_id)
            raise

    def index_faces(self, image, max_faces):
        """
        Finds faces in the specified image, indexes them, and stores them in the
        collection.

        :param image: The image to index.
        :param max_faces: The maximum number of faces to index.
        :return: A tuple. The first element is a list of indexed faces.
                 The second element is a list of faces that couldn't be indexed.
        """
        try:
            response = self.rekognition_client.index_faces(
                CollectionId=self.collection_id, Image=image.image,
                ExternalImageId=image.image_name, MaxFaces=max_faces,
                DetectionAttributes=['ALL'])
            indexed_faces = [
                RekognitionFace({**face['Face'], **face['FaceDetail']})
```

```
                for face in response['FaceRecords']]
            unindexed_faces = [
                RekognitionFace(face['FaceDetail'])
                for face in response['UnindexedFaces']]
            logger.info(
                "Indexed %s faces in %s. Could not index %s faces.",
len(indexed_faces),
                image.image_name, len(unindexed_faces))
        except ClientError:
            logger.exception("Couldn't index faces in image %s.", image.image_name)
            raise
        else:
            return indexed_faces, unindexed_faces

    def list_faces(self, max_results):
        """
        Lists the faces currently indexed in the collection.

        :param max_results: The maximum number of faces to return.
        :return: The list of faces in the collection.
        """
        try:
            response = self.rekognition_client.list_faces(
                CollectionId=self.collection_id, MaxResults=max_results)
            faces = [RekognitionFace(face) for face in response['Faces']]
            logger.info(
                "Found %s faces in collection %s.", len(faces), self.collection_id)
        except ClientError:
            logger.exception(
                "Couldn't list faces in collection %s.", self.collection_id)
            raise
        else:
            return faces

    def search_faces(self, face_id, threshold, max_faces):
        """
        Searches for faces in the collection that match another face from the
        collection.

        :param face_id: The ID of the face in the collection to search for.
        :param threshold: The match confidence must be greater than this value
                          for a face to be included in the results.
        :param max_faces: The maximum number of faces to return.
        :return: The list of matching faces found in the collection. This list does
                 not contain the face specified by `face_id`.
        """
        try:
            response = self.rekognition_client.search_faces(
                CollectionId=self.collection_id, FaceId=face_id,
                FaceMatchThreshold=threshold, MaxFaces=max_faces)
            faces = [RekognitionFace(face['Face']) for face in
response['FaceMatches']]
            logger.info(
                "Found %s faces in %s that match %s.", len(faces),
self.collection_id,
                face_id)
        except ClientError:
            logger.exception(
                "Couldn't search for faces in %s that match %s.",
self.collection_id,
                face_id)
            raise
        else:
            return faces

    def search_faces_by_image(self, image, threshold, max_faces):
```

```
        """
        Searches for faces in the collection that match the largest face in the
        reference image.

        :param image: The image that contains the reference face to search for.
        :param threshold: The match confidence must be greater than this value
                          for a face to be included in the results.
        :param max_faces: The maximum number of faces to return.
        :return: A tuple. The first element is the face found in the reference
 image.
                 The second element is the list of matching faces found in the
                 collection.
        """
        try:
            response = self.rekognition_client.search_faces_by_image(
                CollectionId=self.collection_id, Image=image.image,
                FaceMatchThreshold=threshold, MaxFaces=max_faces)
            image_face = RekognitionFace({
                'BoundingBox': response['SearchedFaceBoundingBox'],
                'Confidence': response['SearchedFaceConfidence']
            })
            collection_faces = [
                RekognitionFace(face['Face']) for face in response['FaceMatches']]
            logger.info("Found %s faces in the collection that match the largest "
                        "face in %s.", len(collection_faces), image.image_name)
        except ClientError:
            logger.exception(
                "Couldn't search for faces in %s that match %s.",
 self.collection_id,
                image.image_name)
            raise
        else:
            return image_face, collection_faces

class RekognitionFace:
    """Encapsulates an Amazon Rekognition face."""
    def __init__(self, face, timestamp=None):
        """
        Initializes the face object.

        :param face: Face data, in the format returned by Amazon Rekognition
                     functions.
        :param timestamp: The time when the face was detected, if the face was
                          detected in a video.
        """
        self.bounding_box = face.get('BoundingBox')
        self.confidence = face.get('Confidence')
        self.landmarks = face.get('Landmarks')
        self.pose = face.get('Pose')
        self.quality = face.get('Quality')
        age_range = face.get('AgeRange')
        if age_range is not None:
            self.age_range = (age_range.get('Low'), age_range.get('High'))
        else:
            self.age_range = None
        self.smile = face.get('Smile', {}).get('Value')
        self.eyeglasses = face.get('Eyeglasses', {}).get('Value')
        self.sunglasses = face.get('Sunglasses', {}).get('Value')
        self.gender = face.get('Gender', {}).get('Value', None)
        self.beard = face.get('Beard', {}).get('Value')
        self.mustache = face.get('Mustache', {}).get('Value')
        self.eyes_open = face.get('EyesOpen', {}).get('Value')
        self.mouth_open = face.get('MouthOpen', {}).get('Value')
        self.emotions = [emo.get('Type') for emo in face.get('Emotions', [])
                         if emo.get('Confidence', 0) > 50]
        self.face_id = face.get('FaceId')
```

```
        self.image_id = face.get('ImageId')
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the face data to a dict.

        :return: A dict that contains the face data.
        """
        rendering = {}
        if self.bounding_box is not None:
            rendering['bounding_box'] = self.bounding_box
        if self.age_range is not None:
            rendering['age'] = f'{self.age_range[0]} - {self.age_range[1]}'
        if self.gender is not None:
            rendering['gender'] = self.gender
        if self.emotions:
            rendering['emotions'] = self.emotions
        if self.face_id is not None:
            rendering['face_id'] = self.face_id
        if self.image_id is not None:
            rendering['image_id'] = self.image_id
        if self.timestamp is not None:
            rendering['timestamp'] = self.timestamp
        has = []
        if self.smile:
            has.append('smile')
        if self.eyeglasses:
            has.append('eyeglasses')
        if self.sunglasses:
            has.append('sunglasses')
        if self.beard:
            has.append('beard')
        if self.mustache:
            has.append('mustache')
        if self.eyes_open:
            has.append('open eyes')
        if self.mouth_open:
            has.append('open mouth')
        if has:
            rendering['has'] = has
        return rendering
```

Use the wrapper classes to build a collection of faces from a set of images and then search for faces in the collection.

```
def usage_demo():
    print('-'*88)
    print("Welcome to the Amazon Rekognition face collection demo!")
    print('-'*88)

    logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')

    rekognition_client = boto3.client('rekognition')
    images = [
        RekognitionImage.from_file(
            '.media/pexels-agung-pandit-wiguna-1128316.jpg', rekognition_client,
            image_name='sitting'),
        RekognitionImage.from_file(
            '.media/pexels-agung-pandit-wiguna-1128317.jpg', rekognition_client,
            image_name='hopping'),
        RekognitionImage.from_file(
            '.media/pexels-agung-pandit-wiguna-1128318.jpg', rekognition_client,
```

```
            image_name='biking')]

    collection_mgr = RekognitionCollectionManager(rekognition_client)
    collection = collection_mgr.create_collection('doc-example-collection-demo')
    print(f"Created collection {collection.collection_id}:")
    pprint(collection.describe_collection())

    print("Indexing faces from three images:")
    for image in images:
        collection.index_faces(image, 10)
    print("Listing faces in collection:")
    faces = collection.list_faces(10)
    for face in faces:
        pprint(face.to_dict())
    input("Press Enter to continue.")

    print(f"Searching for faces in the collection that match the first face in the
 "
          f"list (Face ID: {faces[0].face_id}.")
    found_faces = collection.search_faces(faces[0].face_id, 80, 10)
    print(f"Found {len(found_faces)} matching faces.")
    for face in found_faces:
        pprint(face.to_dict())
    input("Press Enter to continue.")

    print(f"Searching for faces in the collection that match the largest face in "
          f"{images[0].image_name}.")
    image_face, match_faces = collection.search_faces_by_image(images[0], 80, 10)
    print(f"The largest face in {images[0].image_name} is:")
    pprint(image_face.to_dict())
    print(f"Found {len(match_faces)} matching faces.")
    for face in match_faces:
        pprint(face.to_dict())
    input("Press Enter to continue.")

    collection.delete_collection()
    print('Thanks for watching!')
    print('-'*88)
```

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Detect and display elements in images with Amazon Rekognition using an AWS SDK

The following code example shows how to:

- Detect elements in images by using Amazon Rekognition.
- Display images and draw bounding boxes around detected elements.

For more information, see Displaying bounding boxes.

Python

SDK for Python (Boto3)

Tip
To learn how to set up and run this example, see GitHub.

Create classes to wrap Amazon Rekognition functions.

```
import logging
from pprint import pprint
import boto3
from botocore.exceptions import ClientError
import requests

from rekognition_objects import (
    RekognitionFace, RekognitionCelebrity, RekognitionLabel,
    RekognitionModerationLabel, RekognitionText, show_bounding_boxes,
 show_polygons)

logger = logging.getLogger(__name__)


class RekognitionImage:
    """
    Encapsulates an Amazon Rekognition image. This class is a thin wrapper
    around parts of the Boto3 Amazon Rekognition API.
    """
    def __init__(self, image, image_name, rekognition_client):
        """
        Initializes the image object.

        :param image: Data that defines the image, either the image bytes or
                      an Amazon S3 bucket and object key.
        :param image_name: The name of the image.
        :param rekognition_client: A Boto3 Rekognition client.
        """
        self.image = image
        self.image_name = image_name
        self.rekognition_client = rekognition_client

    @classmethod
    def from_file(cls, image_file_name, rekognition_client, image_name=None):
        """
        Creates a RekognitionImage object from a local file.

        :param image_file_name: The file name of the image. The file is opened and
 its
                                bytes are read.
        :param rekognition_client: A Boto3 Rekognition client.
        :param image_name: The name of the image. If this is not specified, the
                           file name is used as the image name.
        :return: The RekognitionImage object, initialized with image bytes from the
                 file.
        """
        with open(image_file_name, 'rb') as img_file:
            image = {'Bytes': img_file.read()}
        name = image_file_name if image_name is None else image_name
        return cls(image, name, rekognition_client)

    @classmethod
    def from_bucket(cls, s3_object, rekognition_client):
        """
        Creates a RekognitionImage object from an Amazon S3 object.

        :param s3_object: An Amazon S3 object that identifies the image. The image
                          is not retrieved until needed for a later call.
        :param rekognition_client: A Boto3 Rekognition client.
        :return: The RekognitionImage object, initialized with Amazon S3 object
 data.
        """
```

```
        image = {'S3Object': {'Bucket': s3_object.bucket_name, 'Name':
s3_object.key}}
        return cls(image, s3_object.key, rekognition_client)

    def detect_faces(self):
        """
        Detects faces in the image.

        :return: The list of faces found in the image.
        """
        try:
            response = self.rekognition_client.detect_faces(
                Image=self.image, Attributes=['ALL'])
            faces = [RekognitionFace(face) for face in response['FaceDetails']]
            logger.info("Detected %s faces.", len(faces))
        except ClientError:
            logger.exception("Couldn't detect faces in %s.", self.image_name)
            raise
        else:
            return faces

    def detect_labels(self, max_labels):
        """
        Detects labels in the image. Labels are objects and people.

        :param max_labels: The maximum number of labels to return.
        :return: The list of labels detected in the image.
        """
        try:
            response = self.rekognition_client.detect_labels(
                Image=self.image, MaxLabels=max_labels)
            labels = [RekognitionLabel(label) for label in response['Labels']]
            logger.info("Found %s labels in %s.", len(labels), self.image_name)
        except ClientError:
            logger.info("Couldn't detect labels in %s.", self.image_name)
            raise
        else:
            return labels

    def recognize_celebrities(self):
        """
        Detects celebrities in the image.

        :return: A tuple. The first element is the list of celebrities found in
                 the image. The second element is the list of faces that were
                 detected but did not match any known celebrities.
        """
        try:
            response = self.rekognition_client.recognize_celebrities(
                Image=self.image)
            celebrities = [RekognitionCelebrity(celeb)
                           for celeb in response['CelebrityFaces']]
            other_faces = [RekognitionFace(face)
                           for face in response['UnrecognizedFaces']]
            logger.info(
                "Found %s celebrities and %s other faces in %s.", len(celebrities),
                len(other_faces), self.image_name)
        except ClientError:
            logger.exception("Couldn't detect celebrities in %s.", self.image_name)
            raise
        else:
            return celebrities, other_faces

    def compare_faces(self, target_image, similarity):
        """
        Compares faces in the image with the largest face in the target image.
```

```
        :param target_image: The target image to compare against.
        :param similarity: Faces in the image must have a similarity value greater
                           than this value to be included in the results.
        :return: A tuple. The first element is the list of faces that match the
                 reference image. The second element is the list of faces that have
                 a similarity value below the specified threshold.
        """
        try:
            response = self.rekognition_client.compare_faces(
                SourceImage=self.image,
                TargetImage=target_image.image,
                SimilarityThreshold=similarity)
            matches = [RekognitionFace(match['Face']) for match
                       in response['FaceMatches']]
            unmatches = [RekognitionFace(face) for face in
response['UnmatchedFaces']]
            logger.info(
                "Found %s matched faces and %s unmatched faces.",
                len(matches), len(unmatches))
        except ClientError:
            logger.exception(
                "Couldn't match faces from %s to %s.", self.image_name,
                target_image.image_name)
            raise
        else:
            return matches, unmatches

    def detect_moderation_labels(self):
        """
        Detects moderation labels in the image. Moderation labels identify content
        that may be inappropriate for some audiences.

        :return: The list of moderation labels found in the image.
        """
        try:
            response = self.rekognition_client.detect_moderation_labels(
                Image=self.image)
            labels = [RekognitionModerationLabel(label)
                      for label in response['ModerationLabels']]
            logger.info(
                "Found %s moderation labels in %s.", len(labels), self.image_name)
        except ClientError:
            logger.exception(
                "Couldn't detect moderation labels in %s.", self.image_name)
            raise
        else:
            return labels

    def detect_text(self):
        """
        Detects text in the image.

        :return The list of text elements found in the image.
        """
        try:
            response = self.rekognition_client.detect_text(Image=self.image)
            texts = [RekognitionText(text) for text in response['TextDetections']]
            logger.info("Found %s texts in %s.", len(texts), self.image_name)
        except ClientError:
            logger.exception("Couldn't detect text in %s.", self.image_name)
            raise
        else:
            return texts
```

Create helper functions to draw bounding boxes and polygons.

```
import io
import logging
from PIL import Image, ImageDraw

logger = logging.getLogger(__name__)


def show_bounding_boxes(image_bytes, box_sets, colors):
    """
    Draws bounding boxes on an image and shows it with the default image viewer.

    :param image_bytes: The image to draw, as bytes.
    :param box_sets: A list of lists of bounding boxes to draw on the image.
    :param colors: A list of colors to use to draw the bounding boxes.
    """
    image = Image.open(io.BytesIO(image_bytes))
    draw = ImageDraw.Draw(image)
    for boxes, color in zip(box_sets, colors):
        for box in boxes:
            left = image.width * box['Left']
            top = image.height * box['Top']
            right = (image.width * box['Width']) + left
            bottom = (image.height * box['Height']) + top
            draw.rectangle([left, top, right, bottom], outline=color, width=3)
    image.show()

def show_polygons(image_bytes, polygons, color):
    """
    Draws polygons on an image and shows it with the default image viewer.

    :param image_bytes: The image to draw, as bytes.
    :param polygons: The list of polygons to draw on the image.
    :param color: The color to use to draw the polygons.
    """
    image = Image.open(io.BytesIO(image_bytes))
    draw = ImageDraw.Draw(image)
    for polygon in polygons:
        draw.polygon([
            (image.width * point['X'], image.height * point['Y']) for point in
 polygon],
            outline=color)
    image.show()
```

Create classes to parse objects returned by Amazon Rekognition.

```
class RekognitionFace:
    """Encapsulates an Amazon Rekognition face."""
    def __init__(self, face, timestamp=None):
        """
        Initializes the face object.

        :param face: Face data, in the format returned by Amazon Rekognition
                     functions.
        :param timestamp: The time when the face was detected, if the face was
                          detected in a video.
        """
        self.bounding_box = face.get('BoundingBox')
        self.confidence = face.get('Confidence')
        self.landmarks = face.get('Landmarks')
        self.pose = face.get('Pose')
        self.quality = face.get('Quality')
```

```
        age_range = face.get('AgeRange')
        if age_range is not None:
            self.age_range = (age_range.get('Low'), age_range.get('High'))
        else:
            self.age_range = None
        self.smile = face.get('Smile', {}).get('Value')
        self.eyeglasses = face.get('Eyeglasses', {}).get('Value')
        self.sunglasses = face.get('Sunglasses', {}).get('Value')
        self.gender = face.get('Gender', {}).get('Value', None)
        self.beard = face.get('Beard', {}).get('Value')
        self.mustache = face.get('Mustache', {}).get('Value')
        self.eyes_open = face.get('EyesOpen', {}).get('Value')
        self.mouth_open = face.get('MouthOpen', {}).get('Value')
        self.emotions = [emo.get('Type') for emo in face.get('Emotions', [])
                         if emo.get('Confidence', 0) > 50]
        self.face_id = face.get('FaceId')
        self.image_id = face.get('ImageId')
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the face data to a dict.

        :return: A dict that contains the face data.
        """
        rendering = {}
        if self.bounding_box is not None:
            rendering['bounding_box'] = self.bounding_box
        if self.age_range is not None:
            rendering['age'] = f'{self.age_range[0]} - {self.age_range[1]}'
        if self.gender is not None:
            rendering['gender'] = self.gender
        if self.emotions:
            rendering['emotions'] = self.emotions
        if self.face_id is not None:
            rendering['face_id'] = self.face_id
        if self.image_id is not None:
            rendering['image_id'] = self.image_id
        if self.timestamp is not None:
            rendering['timestamp'] = self.timestamp
        has = []
        if self.smile:
            has.append('smile')
        if self.eyeglasses:
            has.append('eyeglasses')
        if self.sunglasses:
            has.append('sunglasses')
        if self.beard:
            has.append('beard')
        if self.mustache:
            has.append('mustache')
        if self.eyes_open:
            has.append('open eyes')
        if self.mouth_open:
            has.append('open mouth')
        if has:
            rendering['has'] = has
        return rendering

class RekognitionCelebrity:
    """Encapsulates an Amazon Rekognition celebrity."""
    def __init__(self, celebrity, timestamp=None):
        """
        Initializes the celebrity object.
```

```
        :param celebrity: Celebrity data, in the format returned by Amazon
 Rekognition
                          functions.
        :param timestamp: The time when the celebrity was detected, if the
 celebrity
                          was detected in a video.
        """
        self.info_urls = celebrity.get('Urls')
        self.name = celebrity.get('Name')
        self.id = celebrity.get('Id')
        self.face = RekognitionFace(celebrity.get('Face'))
        self.confidence = celebrity.get('MatchConfidence')
        self.bounding_box = celebrity.get('BoundingBox')
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the celebrity data to a dict.

        :return: A dict that contains the celebrity data.
        """
        rendering = self.face.to_dict()
        if self.name is not None:
            rendering['name'] = self.name
        if self.info_urls:
            rendering['info URLs'] = self.info_urls
        if self.timestamp is not None:
            rendering['timestamp'] = self.timestamp
        return rendering

class RekognitionPerson:
    """Encapsulates an Amazon Rekognition person."""
    def __init__(self, person, timestamp=None):
        """
        Initializes the person object.

        :param person: Person data, in the format returned by Amazon Rekognition
                       functions.
        :param timestamp: The time when the person was detected, if the person
                          was detected in a video.
        """
        self.index = person.get('Index')
        self.bounding_box = person.get('BoundingBox')
        face = person.get('Face')
        self.face = RekognitionFace(face) if face is not None else None
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the person data to a dict.

        :return: A dict that contains the person data.
        """
        rendering = self.face.to_dict() if self.face is not None else {}
        if self.index is not None:
            rendering['index'] = self.index
        if self.bounding_box is not None:
            rendering['bounding_box'] = self.bounding_box
        if self.timestamp is not None:
            rendering['timestamp'] = self.timestamp
        return rendering

class RekognitionLabel:
    """Encapsulates an Amazon Rekognition label."""
    def __init__(self, label, timestamp=None):
        """
```

```
        Initializes the label object.

        :param label: Label data, in the format returned by Amazon Rekognition
                      functions.
        :param timestamp: The time when the label was detected, if the label
                          was detected in a video.
        """
        self.name = label.get('Name')
        self.confidence = label.get('Confidence')
        self.instances = label.get('Instances')
        self.parents = label.get('Parents')
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the label data to a dict.

        :return: A dict that contains the label data.
        """
        rendering = {}
        if self.name is not None:
            rendering['name'] = self.name
        if self.timestamp is not None:
            rendering['timestamp'] = self.timestamp
        return rendering

class RekognitionModerationLabel:
    """Encapsulates an Amazon Rekognition moderation label."""
    def __init__(self, label, timestamp=None):
        """
        Initializes the moderation label object.

        :param label: Label data, in the format returned by Amazon Rekognition
                      functions.
        :param timestamp: The time when the moderation label was detected, if the
                          label was detected in a video.
        """
        self.name = label.get('Name')
        self.confidence = label.get('Confidence')
        self.parent_name = label.get('ParentName')
        self.timestamp = timestamp

    def to_dict(self):
        """
        Renders some of the moderation label data to a dict.

        :return: A dict that contains the moderation label data.
        """
        rendering = {}
        if self.name is not None:
            rendering['name'] = self.name
        if self.parent_name is not None:
            rendering['parent_name'] = self.parent_name
        if self.timestamp is not None:
            rendering['timestamp'] = self.timestamp
        return rendering

class RekognitionText:
    """Encapsulates an Amazon Rekognition text element."""
    def __init__(self, text_data):
        """
        Initializes the text object.

        :param text_data: Text data, in the format returned by Amazon Rekognition
                          functions.
        """
```

```
        self.text = text_data.get('DetectedText')
        self.kind = text_data.get('Type')
        self.id = text_data.get('Id')
        self.parent_id = text_data.get('ParentId')
        self.confidence = text_data.get('Confidence')
        self.geometry = text_data.get('Geometry')

    def to_dict(self):
        """
        Renders some of the text data to a dict.

        :return: A dict that contains the text data.
        """
        rendering = {}
        if self.text is not None:
            rendering['text'] = self.text
        if self.kind is not None:
            rendering['kind'] = self.kind
        if self.geometry is not None:
            rendering['polygon'] = self.geometry.get('Polygon')
        return rendering
```

Use the wrapper classes to detect elements in images and display their bounding boxes. The images used in this example can be found on GitHub along with instructions and more code.

```
def usage_demo():
    print('-'*88)
    print("Welcome to the Amazon Rekognition image detection demo!")
    print('-'*88)

    logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')
    rekognition_client = boto3.client('rekognition')
    street_scene_file_name = ".media/pexels-kaique-rocha-109919.jpg"
    celebrity_file_name = ".media/pexels-pixabay-53370.jpg"
    one_girl_url = 'https://dhei5unw3vrsx.cloudfront.net/images/
source3_resized.jpg'
    three_girls_url = 'https://dhei5unw3vrsx.cloudfront.net/images/
target3_resized.jpg'
    swimwear_object = boto3.resource('s3').Object(
        'console-sample-images-pdx', 'yoga_swimwear.jpg')
    book_file_name = '.media/pexels-christina-morillo-1181671.jpg'

    street_scene_image = RekognitionImage.from_file(
        street_scene_file_name, rekognition_client)
    print(f"Detecting faces in {street_scene_image.image_name}...")
    faces = street_scene_image.detect_faces()
    print(f"Found {len(faces)} faces, here are the first three.")
    for face in faces[:3]:
        pprint(face.to_dict())
    show_bounding_boxes(
        street_scene_image.image['Bytes'], [[face.bounding_box for face in faces]],
        ['aqua'])
    input("Press Enter to continue.")

    print(f"Detecting labels in {street_scene_image.image_name}...")
    labels = street_scene_image.detect_labels(100)
    print(f"Found {len(labels)} labels.")
    for label in labels:
        pprint(label.to_dict())
    names = []
    box_sets = []
    colors = ['aqua', 'red', 'white', 'blue', 'yellow', 'green']
    for label in labels:
```

```
        if label.instances:
            names.append(label.name)
            box_sets.append([inst['BoundingBox'] for inst in label.instances])
    print(f"Showing bounding boxes for {names} in {colors[:len(names)]}.")
    show_bounding_boxes(
        street_scene_image.image['Bytes'], box_sets, colors[:len(names)])
    input("Press Enter to continue.")

    celebrity_image = RekognitionImage.from_file(
        celebrity_file_name, rekognition_client)
    print(f"Detecting celebrities in {celebrity_image.image_name}...")
    celebs, others = celebrity_image.recognize_celebrities()
    print(f"Found {len(celebs)} celebrities.")
    for celeb in celebs:
        pprint(celeb.to_dict())
    show_bounding_boxes(
        celebrity_image.image['Bytes'],
        [[celeb.face.bounding_box for celeb in celebs]], ['aqua'])
    input("Press Enter to continue.")

    girl_image_response = requests.get(one_girl_url)
    girl_image = RekognitionImage(
        {'Bytes': girl_image_response.content}, "one-girl", rekognition_client)
    group_image_response = requests.get(three_girls_url)
    group_image = RekognitionImage(
        {'Bytes': group_image_response.content}, "three-girls", rekognition_client)
    print("Comparing reference face to group of faces...")
    matches, unmatches = girl_image.compare_faces(group_image, 80)
    print(f"Found {len(matches)} face matching the reference face.")
    show_bounding_boxes(
        group_image.image['Bytes'], [[match.bounding_box for match in matches]],
        ['aqua'])
    input("Press Enter to continue.")

    swimwear_image = RekognitionImage.from_bucket(swimwear_object,
 rekognition_client)
    print(f"Detecting suggestive content in {swimwear_object.key}...")
    labels = swimwear_image.detect_moderation_labels()
    print(f"Found {len(labels)} moderation labels.")
    for label in labels:
        pprint(label.to_dict())
    input("Press Enter to continue.")

    book_image = RekognitionImage.from_file(book_file_name, rekognition_client)
    print(f"Detecting text in {book_image.image_name}...")
    texts = book_image.detect_text()
    print(f"Found {len(texts)} text instances. Here are the first seven:")
    for text in texts[:7]:
        pprint(text.to_dict())
    show_polygons(
        book_image.image['Bytes'], [text.geometry['Polygon'] for text in texts],
 'aqua')

    print("Thanks for watching!")
    print('-'*88)
```

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Detect information in videos using Amazon Rekognition and the AWS SDK

The following code examples show how to:

- Start Amazon Rekognition jobs to detect elements like people, objects, and text in videos.
- Check job status until jobs finish.
- Output the list of elements detected by each job.

Java

SDK for Java 2.x

Tip
To learn how to set up and run this example, see GitHub.

Get celebrity results from a video located in an Amazon S3 bucket.

```
    public static void StartCelebrityDetection(RekognitionClient rekClient,
                                               NotificationChannel channel,
                                               String bucket,
                                               String video){
        try {
            S3Object s3Obj = S3Object.builder()
                .bucket(bucket)
                .name(video)
                .build();

            Video vidOb = Video.builder()
                .s3Object(s3Obj)
                .build();

            StartCelebrityRecognitionRequest recognitionRequest =
 StartCelebrityRecognitionRequest.builder()
                .jobTag("Celebrities")
                .notificationChannel(channel)
                .video(vidOb)
                .build();

            StartCelebrityRecognitionResponse startCelebrityRecognitionResult =
 rekClient.startCelebrityRecognition(recognitionRequest);
            startJobId = startCelebrityRecognitionResult.jobId();

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void GetCelebrityDetectionResults(RekognitionClient rekClient) {

        try {
            String paginationToken=null;
            GetCelebrityRecognitionResponse recognitionResponse = null;
            boolean finished = false;
            String status;
            int yy=0 ;

            do{
                if (recognitionResponse !=null)
                    paginationToken = recognitionResponse.nextToken();
```

```
                GetCelebrityRecognitionRequest recognitionRequest =
GetCelebrityRecognitionRequest.builder()
                    .jobId(startJobId)
                    .nextToken(paginationToken)
                    .sortBy(CelebrityRecognitionSortBy.TIMESTAMP)
                    .maxResults(10)
                    .build();

                // Wait until the job succeeds
                while (!finished) {
                    recognitionResponse =
rekClient.getCelebrityRecognition(recognitionRequest);
                    status = recognitionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is null.
                VideoMetadata videoMetaData=recognitionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " + videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<CelebrityRecognition> celebs=
recognitionResponse.celebrities();
                for (CelebrityRecognition celeb: celebs) {
                    long seconds=celeb.timestamp()/1000;
                    System.out.print("Sec: " + seconds + " ");
                    CelebrityDetail details=celeb.celebrity();
                    System.out.println("Name: " + details.name());
                    System.out.println("Id: " + details.id());
                    System.out.println();
                }

            } while (recognitionResponse.nextToken() != null);

        } catch(RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

Detect labels in a video by a label detection operation.

```
    public static void startLabels(RekognitionClient rekClient,
                                   NotificationChannel channel,
                                   String bucket,
                                   String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                .bucket(bucket)
                .name(video)
                .build();
```

```
            Video vidOb = Video.builder()
                .s3Object(s3Obj)
                .build();

            StartLabelDetectionRequest labelDetectionRequest =
StartLabelDetectionRequest.builder()
                .jobTag("DetectingLabels")
                .notificationChannel(channel)
                .video(vidOb)
                .minConfidence(50F)
                .build();

            StartLabelDetectionResponse labelDetectionResponse =
rekClient.startLabelDetection(labelDetectionRequest);
            startJobId = labelDetectionResponse.jobId();

            boolean ans = true;
            String status = "";
            int yy = 0;
            while (ans) {

                GetLabelDetectionRequest detectionRequest =
GetLabelDetectionRequest.builder()
                    .jobId(startJobId)
                    .maxResults(10)
                    .build();

                GetLabelDetectionResponse result =
rekClient.getLabelDetection(detectionRequest);
                status = result.jobStatusAsString();

                if (status.compareTo("SUCCEEDED") == 0)
                    ans = false;
                else
                    System.out.println(yy +" status is: "+status);

                Thread.sleep(1000);
                yy++;
            }

            System.out.println(startJobId +" status is: "+status);

        } catch(RekognitionException | InterruptedException e) {
            e.getMessage();
            System.exit(1);
        }
    }

    public static void getLabelJob(RekognitionClient rekClient, SqsClient sqs,
String queueUrl) {

        List<Message> messages;
        ReceiveMessageRequest messageRequest = ReceiveMessageRequest.builder()
            .queueUrl(queueUrl)
            .build();

        try {
            messages = sqs.receiveMessage(messageRequest).messages();

            if (!messages.isEmpty()) {
                for (Message message: messages) {
                    String notification = message.body();

                    // Get the status and job id from the 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 in JSON is " + operationJobId);

                    DeleteMessageRequest deleteMessageRequest =
DeleteMessageRequest.builder()
                        .queueUrl(queueUrl)
                        .build();

                    String jobId = operationJobId.textValue();
                    if (startJobId.compareTo(jobId)==0) {
                        System.out.println("Job id: " + operationJobId );
                        System.out.println("Status : " +
operationStatus.toString());

                        if (operationStatus.asText().equals("SUCCEEDED"))
                            GetResultsLabels(rekClient);
                        else
                            System.out.println("Video analysis failed");

                        sqs.deleteMessage(deleteMessageRequest);
                    }

                    else{
                        System.out.println("Job received was not job " +
startJobId);
                        sqs.deleteMessage(deleteMessageRequest);
                    }
                }
            }

        } catch(RekognitionException e) {
            e.getMessage();
            System.exit(1);
        } catch (JsonMappingException e) {
            e.printStackTrace();
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
    }

    // Gets the job results by calling GetLabelDetection
    private static void GetResultsLabels(RekognitionClient rekClient) {

        int maxResults=10;
        String paginationToken=null;
        GetLabelDetectionResponse labelDetectionResult=null;

        try {
            do {
                if (labelDetectionResult !=null)
                    paginationToken = labelDetectionResult.nextToken();


                GetLabelDetectionRequest labelDetectionRequest=
GetLabelDetectionRequest.builder()
                    .jobId(startJobId)
                    .sortBy(LabelDetectionSortBy.TIMESTAMP)
                    .maxResults(maxResults)
                    .nextToken(paginationToken)
                    .build();
```

```
                labelDetectionResult =
rekClient.getLabelDetection(labelDetectionRequest);
                VideoMetadata videoMetaData=labelDetectionResult.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " + videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());

                List<LabelDetection> detectedLabels= labelDetectionResult.labels();
                for (LabelDetection detectedLabel: detectedLabels) {
                    long seconds=detectedLabel.timestamp();
                    Label label=detectedLabel.label();
                    System.out.println("Millisecond: " + seconds + " ");

                    System.out.println("   Label:" + label.name());
                    System.out.println("   Confidence:" +
detectedLabel.label().confidence().toString());

                    List<Instance> instances = label.instances();
                    System.out.println("   Instances of " + label.name());

                    if (instances.isEmpty()) {
                        System.out.println("        " + "None");
                    } else {
                        for (Instance instance : instances) {
                            System.out.println("        Confidence: " +
instance.confidence().toString());
                            System.out.println("        Bounding box: " +
instance.boundingBox().toString());
                        }
                    }
                    System.out.println("   Parent labels for " + label.name() +
":");
                    List<Parent> parents = label.parents();

                    if (parents.isEmpty()) {
                        System.out.println("        None");
                    } else {
                        for (Parent parent : parents) {
                            System.out.println("   " + parent.name());
                        }
                    }
                    System.out.println();
                }
            } while (labelDetectionResult !=null &&
labelDetectionResult.nextToken() != null);

        } catch(RekognitionException e) {
            e.getMessage();
            System.exit(1);
        }
    }
```

Detect faces in a video stored in an Amazon S3 bucket.

```
    public static void startLabels(RekognitionClient rekClient,
                                   NotificationChannel channel,
                                   String bucket,
                                   String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                .bucket(bucket)
                .name(video)
```

```
                .build();

            Video vidOb = Video.builder()
                .s3Object(s3Obj)
                .build();

            StartLabelDetectionRequest labelDetectionRequest =
StartLabelDetectionRequest.builder()
                .jobTag("DetectingLabels")
                .notificationChannel(channel)
                .video(vidOb)
                .minConfidence(50F)
                .build();

            StartLabelDetectionResponse labelDetectionResponse =
rekClient.startLabelDetection(labelDetectionRequest);
            startJobId = labelDetectionResponse.jobId();

            boolean ans = true;
            String status = "";
            int yy = 0;
            while (ans) {

                GetLabelDetectionRequest detectionRequest =
GetLabelDetectionRequest.builder()
                    .jobId(startJobId)
                    .maxResults(10)
                    .build();

                GetLabelDetectionResponse result =
rekClient.getLabelDetection(detectionRequest);
                status = result.jobStatusAsString();

                if (status.compareTo("SUCCEEDED") == 0)
                    ans = false;
                else
                    System.out.println(yy +" status is: "+status);

                Thread.sleep(1000);
                yy++;
            }

            System.out.println(startJobId +" status is: "+status);

        } catch(RekognitionException | InterruptedException e) {
            e.getMessage();
            System.exit(1);
        }
    }

    public static void getLabelJob(RekognitionClient rekClient, SqsClient sqs,
String queueUrl) {

        List<Message> messages;
        ReceiveMessageRequest messageRequest = ReceiveMessageRequest.builder()
            .queueUrl(queueUrl)
            .build();

        try {
            messages = sqs.receiveMessage(messageRequest).messages();

            if (!messages.isEmpty()) {
                for (Message message: messages) {
                    String notification = message.body();

                    // Get the status and job id from the 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 in JSON is " + operationJobId);

                    DeleteMessageRequest deleteMessageRequest =
DeleteMessageRequest.builder()
                        .queueUrl(queueUrl)
                        .build();

                    String jobId = operationJobId.textValue();
                    if (startJobId.compareTo(jobId)==0) {
                        System.out.println("Job id: " + operationJobId );
                        System.out.println("Status : " +
operationStatus.toString());

                        if (operationStatus.asText().equals("SUCCEEDED"))
                            GetResultsLabels(rekClient);
                        else
                            System.out.println("Video analysis failed");

                        sqs.deleteMessage(deleteMessageRequest);
                    }

                    else{
                        System.out.println("Job received was not job " +
startJobId);
                        sqs.deleteMessage(deleteMessageRequest);
                    }
                }
            }

        } catch(RekognitionException e) {
            e.getMessage();
            System.exit(1);
        } catch (JsonMappingException e) {
            e.printStackTrace();
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
    }

    // Gets the job results by calling GetLabelDetection
    private static void GetResultsLabels(RekognitionClient rekClient) {

        int maxResults=10;
        String paginationToken=null;
        GetLabelDetectionResponse labelDetectionResult=null;

        try {
            do {
                if (labelDetectionResult !=null)
                    paginationToken = labelDetectionResult.nextToken();


                GetLabelDetectionRequest labelDetectionRequest=
GetLabelDetectionRequest.builder()
                    .jobId(startJobId)
                    .sortBy(LabelDetectionSortBy.TIMESTAMP)
                    .maxResults(maxResults)
                    .nextToken(paginationToken)
```

```
                    .build();

                labelDetectionResult =
rekClient.getLabelDetection(labelDetectionRequest);
                VideoMetadata videoMetaData=labelDetectionResult.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " + videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());

                List<LabelDetection> detectedLabels= labelDetectionResult.labels();
                for (LabelDetection detectedLabel: detectedLabels) {
                    long seconds=detectedLabel.timestamp();
                    Label label=detectedLabel.label();
                    System.out.println("Millisecond: " + seconds + " ");

                    System.out.println("   Label:" + label.name());
                    System.out.println("   Confidence:" +
detectedLabel.label().confidence().toString());

                    List<Instance> instances = label.instances();
                    System.out.println("   Instances of " + label.name());

                    if (instances.isEmpty()) {
                        System.out.println("        " + "None");
                    } else {
                        for (Instance instance : instances) {
                            System.out.println("        Confidence: " +
instance.confidence().toString());
                            System.out.println("        Bounding box: " +
instance.boundingBox().toString());
                        }
                    }
                    System.out.println("   Parent labels for " + label.name() +
":");
                    List<Parent> parents = label.parents();

                    if (parents.isEmpty()) {
                        System.out.println("        None");
                    } else {
                        for (Parent parent : parents) {
                            System.out.println("   " + parent.name());
                        }
                    }
                    System.out.println();
                }
            } while (labelDetectionResult !=null &&
labelDetectionResult.nextToken() != null);

        } catch(RekognitionException e) {
            e.getMessage();
            System.exit(1);
        }
    }
```

Detect inappropriate or offensive content in a video stored in an Amazon S3 bucket.

```
    public static void startModerationDetection(RekognitionClient rekClient,
                                                NotificationChannel channel,
                                                String bucket,
                                                String video) {

        try {
            S3Object s3Obj = S3Object.builder()
```

```
                .bucket(bucket)
                .name(video)
                .build();

            Video vidOb = Video.builder()
                .s3Object(s3Obj)
                .build();

            StartContentModerationRequest modDetectionRequest =
StartContentModerationRequest.builder()
                .jobTag("Moderation")
                .notificationChannel(channel)
                .video(vidOb)
                .build();

            StartContentModerationResponse startModDetectionResult =
rekClient.startContentModeration(modDetectionRequest);
            startJobId=startModDetectionResult.jobId();

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void GetModResults(RekognitionClient rekClient) {

        try {
            String paginationToken=null;
            GetContentModerationResponse modDetectionResponse=null;
            boolean finished = false;
            String status;
            int yy=0 ;

            do{
                if (modDetectionResponse !=null)
                    paginationToken = modDetectionResponse.nextToken();

                GetContentModerationRequest modRequest =
GetContentModerationRequest.builder()
                    .jobId(startJobId)
                    .nextToken(paginationToken)
                    .maxResults(10)
                    .build();

                // Wait until the job succeeds
                while (!finished) {
                    modDetectionResponse =
rekClient.getContentModeration(modRequest);
                    status = modDetectionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is null
                VideoMetadata videoMetaData=modDetectionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
```

```
                System.out.println("Duration: " + videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<ContentModerationDetection> mods =
modDetectionResponse.moderationLabels();
                for (ContentModerationDetection mod: mods) {
                    long seconds=mod.timestamp()/1000;
                    System.out.print("Mod label: " + seconds + " ");
                    System.out.println(mod.moderationLabel().toString());
                    System.out.println();
                }

            } while (modDetectionResponse !=null &&
modDetectionResponse.nextToken() != null);

        } catch(RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

Detect technical cue segments and shot detection segments in a video stored in an Amazon S3 bucket.

```
    public static void StartSegmentDetection (RekognitionClient rekClient,
                                  NotificationChannel channel,
                                  String bucket,
                                  String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                .bucket(bucket)
                .name(video)
                .build();

            Video vidOb = Video.builder()
                .s3Object(s3Obj)
                .build();

            StartShotDetectionFilter cueDetectionFilter =
StartShotDetectionFilter.builder()
                .minSegmentConfidence(60F)
                .build();

            StartTechnicalCueDetectionFilter technicalCueDetectionFilter =
StartTechnicalCueDetectionFilter.builder()
                .minSegmentConfidence(60F)
                .build();

            StartSegmentDetectionFilters filters =
StartSegmentDetectionFilters.builder()
                .shotFilter(cueDetectionFilter)
                .technicalCueFilter(technicalCueDetectionFilter)
                .build();

            StartSegmentDetectionRequest segDetectionRequest =
StartSegmentDetectionRequest.builder()
                .jobTag("DetectingLabels")
                .notificationChannel(channel)
                .segmentTypes(SegmentType.TECHNICAL_CUE , SegmentType.SHOT)
                .video(vidOb)
                .filters(filters)
                .build();
```

```
            StartSegmentDetectionResponse segDetectionResponse =
rekClient.startSegmentDetection(segDetectionRequest);
            startJobId = segDetectionResponse.jobId();

        } catch(RekognitionException e) {
            e.getMessage();
            System.exit(1);
        }
    }

    public static void getSegmentResults(RekognitionClient rekClient) {

        try {
            String paginationToken = null;
            GetSegmentDetectionResponse segDetectionResponse = null;
            boolean finished = false;
            String status;
            int yy = 0;

            do {
                if (segDetectionResponse != null)
                    paginationToken = segDetectionResponse.nextToken();

                GetSegmentDetectionRequest recognitionRequest =
GetSegmentDetectionRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds.
                while (!finished) {
                    segDetectionResponse =
rekClient.getSegmentDetection(recognitionRequest);
                    status = segDetectionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }
                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is null.
                List<VideoMetadata> videoMetaData =
segDetectionResponse.videoMetadata();
                for (VideoMetadata metaData : videoMetaData) {
                    System.out.println("Format: " + metaData.format());
                    System.out.println("Codec: " + metaData.codec());
                    System.out.println("Duration: " + metaData.durationMillis());
                    System.out.println("FrameRate: " + metaData.frameRate());
                    System.out.println("Job");
                }

                List<SegmentDetection> detectedSegments =
segDetectionResponse.segments();
                for (SegmentDetection detectedSegment : detectedSegments) {
                    String type = detectedSegment.type().toString();
                    if (type.contains(SegmentType.TECHNICAL_CUE.toString())) {
                        System.out.println("Technical Cue");
                        TechnicalCueSegment segmentCue =
detectedSegment.technicalCueSegment();
```

```
                        System.out.println("\tType: " + segmentCue.type());
                        System.out.println("\tConfidence: " +
segmentCue.confidence().toString());
                    }

                    if (type.contains(SegmentType.SHOT.toString())) {
                        System.out.println("Shot");
                        ShotSegment segmentShot = detectedSegment.shotSegment();
                        System.out.println("\tIndex " + segmentShot.index());
                        System.out.println("\tConfidence: " +
segmentShot.confidence().toString());
                    }

                    long seconds = detectedSegment.durationMillis();
                    System.out.println("\tDuration : " + seconds + "
milliseconds");
                    System.out.println("\tStart time code: " +
detectedSegment.startTimecodeSMPTE());
                    System.out.println("\tEnd time code: " +
detectedSegment.endTimecodeSMPTE());
                    System.out.println("\tDuration time code: " +
detectedSegment.durationSMPTE());
                    System.out.println();
                }

            } while (segDetectionResponse !=null &&
segDetectionResponse.nextToken() != null);

        } catch(RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

Detect text in a video stored in a video stored in an Amazon S3 bucket.

```
    public static void startTextLabels(RekognitionClient rekClient,
                                NotificationChannel channel,
                                String bucket,
                                String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                .bucket(bucket)
                .name(video)
                .build();

            Video vidOb = Video.builder()
                .s3Object(s3Obj)
                .build();

            StartTextDetectionRequest labelDetectionRequest =
StartTextDetectionRequest.builder()
                .jobTag("DetectingLabels")
                .notificationChannel(channel)
                .video(vidOb)
                .build();

            StartTextDetectionResponse labelDetectionResponse =
rekClient.startTextDetection(labelDetectionRequest);
            startJobId = labelDetectionResponse.jobId();

        } catch (RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
```

```
        }
    }

    public static void GetTextResults(RekognitionClient rekClient) {

        try {
            String paginationToken=null;
            GetTextDetectionResponse textDetectionResponse=null;
            boolean finished = false;
            String status;
            int yy=0 ;

            do{
                if (textDetectionResponse !=null)
                    paginationToken = textDetectionResponse.nextToken();

                GetTextDetectionRequest recognitionRequest =
GetTextDetectionRequest.builder()
                    .jobId(startJobId)
                    .nextToken(paginationToken)
                    .maxResults(10)
                    .build();

                // Wait until the job succeeds.
                while (!finished) {
                    textDetectionResponse =
rekClient.getTextDetection(recognitionRequest);
                    status = textDetectionResponse.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is null.
                VideoMetadata videoMetaData=textDetectionResponse.videoMetadata();
                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " + videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<TextDetectionResult> labels=
textDetectionResponse.textDetections();
                for (TextDetectionResult detectedText: labels) {
                    System.out.println("Confidence: " +
detectedText.textDetection().confidence().toString());
                    System.out.println("Id : " +
detectedText.textDetection().id());
                    System.out.println("Parent Id: " +
detectedText.textDetection().parentId());
                    System.out.println("Type: " +
detectedText.textDetection().type());
                    System.out.println("Text: " +
detectedText.textDetection().detectedText());
                    System.out.println();
                }

            } while (textDetectionResponse !=null &&
textDetectionResponse.nextToken() != null);
```

```
        } catch(RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

Detect people in a video stored in a video stored in an Amazon S3 bucket.

```
    public static void startPersonLabels(RekognitionClient rekClient,
                                         NotificationChannel channel,
                                         String bucket,
                                         String video) {
        try {
            S3Object s3Obj = S3Object.builder()
                .bucket(bucket)
                .name(video)
                .build();

            Video vidOb = Video.builder()
                .s3Object(s3Obj)
                .build();

            StartPersonTrackingRequest personTrackingRequest =
StartPersonTrackingRequest.builder()
                .jobTag("DetectingLabels")
                .video(vidOb)
                .notificationChannel(channel)
                .build();

            StartPersonTrackingResponse labelDetectionResponse =
rekClient.startPersonTracking(personTrackingRequest);
            startJobId = labelDetectionResponse.jobId();

        } catch(RekognitionException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void GetPersonDetectionResults(RekognitionClient rekClient) {

        try {
            String paginationToken=null;
            GetPersonTrackingResponse personTrackingResult=null;
            boolean finished = false;
            String status;
            int yy=0 ;

            do{
                if (personTrackingResult !=null)
                    paginationToken = personTrackingResult.nextToken();

                GetPersonTrackingRequest recognitionRequest =
GetPersonTrackingRequest.builder()
                        .jobId(startJobId)
                        .nextToken(paginationToken)
                        .maxResults(10)
                        .build();

                // Wait until the job succeeds
                while (!finished) {
```

```
                    personTrackingResult =
rekClient.getPersonTracking(recognitionRequest);
                    status = personTrackingResult.jobStatusAsString();

                    if (status.compareTo("SUCCEEDED") == 0)
                        finished = true;
                    else {
                        System.out.println(yy + " status is: " + status);
                        Thread.sleep(1000);
                    }
                    yy++;
                }

                finished = false;

                // Proceed when the job is done - otherwise VideoMetadata is null
                VideoMetadata videoMetaData = personTrackingResult.videoMetadata();

                System.out.println("Format: " + videoMetaData.format());
                System.out.println("Codec: " + videoMetaData.codec());
                System.out.println("Duration: " + videoMetaData.durationMillis());
                System.out.println("FrameRate: " + videoMetaData.frameRate());
                System.out.println("Job");

                List<PersonDetection> detectedPersons=
personTrackingResult.persons();
                for (PersonDetection detectedPerson: detectedPersons) {

                    long seconds=detectedPerson.timestamp()/1000;
                    System.out.print("Sec: " + seconds + " ");
                    System.out.println("Person Identifier: " +
detectedPerson.person().index());
                    System.out.println();
                }

            } while (personTrackingResult !=null &&
personTrackingResult.nextToken() != null);

        } catch(RekognitionException | InterruptedException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - GetCelebrityRecognition
  - GetContentModeration
  - GetLabelDetection
  - GetPersonTracking
  - GetSegmentDetection
  - GetTextDetection
  - StartCelebrityRecognition
  - StartContentModeration
  - StartLabelDetection
  - StartPersonTracking
  - StartSegmentDetection
  - StartTextDetection

Kotlin

SDK for Kotlin

Note
This is prerelease documentation for a feature in preview release. It is subject to change.

Tip
To learn how to set up and run this example, see GitHub.

Detect faces in a video stored in an Amazon S3 bucket.

```
suspend fun startFaceDetection(channelVal: NotificationChannel?, bucketVal: String,
 videoVal: String) {

    val s3Obj = S3Object {
        bucket = bucketVal
        name = videoVal
    }
    val vidOb = Video {
        s3Object = s3Obj
    }

    val request = StartFaceDetectionRequest {
        jobTag = "Faces"
        faceAttributes = FaceAttributes.All
        notificationChannel = channelVal
        video = vidOb
    }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val startLabelDetectionResult = rekClient.startFaceDetection(request)
        startJobId = startLabelDetectionResult.jobId.toString()
    }
}

suspend fun getFaceResults() {

    var finished = false
    var status: String
    var yy = 0
    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        var response: GetFaceDetectionResponse? = null

        val recognitionRequest = GetFaceDetectionRequest {
            jobId = startJobId
            maxResults = 10
        }

        // Wait until the job succeeds.
        while (!finished) {
            response = rekClient.getFaceDetection(recognitionRequest)
            status = response.jobStatus.toString()
            if (status.compareTo("SUCCEEDED") == 0)
                finished = true
            else {
                println("$yy status is: $status")
                delay(1000)
            }
            yy++
        }

        // Proceed when the job is done - otherwise VideoMetadata is null.
        val videoMetaData = response?.videoMetadata
```

```
        println("Format: ${videoMetaData?.format}")
        println("Codec: ${videoMetaData?.codec}")
        println("Duration: ${videoMetaData?.durationMillis}")
        println("FrameRate: ${videoMetaData?.frameRate}")

        // Show face information.
        response?.faces?.forEach { face ->
            println("Age: ${face.face?.ageRange}")
            println("Face: ${face.face?.beard}")
            println("Eye glasses: ${face?.face?.eyeglasses}")
            println("Mustache: ${face.face?.mustache}")
            println("Smile: ${face.face?.smile}")
        }
    }
}
```

Detect inappropriate or offensive content in a video stored in an Amazon S3 bucket.

```
suspend fun startModerationDetection(channel: NotificationChannel?, bucketVal:
 String?, videoVal: String?) {

    val s3Obj = S3Object {
        bucket = bucketVal
        name = videoVal
    }
    val vidOb = Video {
        s3Object = s3Obj
    }
    val request = StartContentModerationRequest {
        jobTag = "Moderation"
        notificationChannel = channel
        video = vidOb
    }

    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        val startModDetectionResult = rekClient.startContentModeration(request)
        startJobId = startModDetectionResult.jobId.toString()
    }
}

suspend fun getModResults() {
    var finished = false
    var status: String
    var yy = 0
    RekognitionClient { region = "us-east-1" }.use { rekClient ->
        var modDetectionResponse: GetContentModerationResponse? = null

        val modRequest = GetContentModerationRequest {
            jobId = startJobId
            maxResults = 10
        }

        // Wait until the job succeeds.
        while (!finished) {
            modDetectionResponse = rekClient.getContentModeration(modRequest)
            status = modDetectionResponse.jobStatus.toString()
            if (status.compareTo("SUCCEEDED") == 0)
                finished = true
            else {
                println("$yy status is: $status")
                delay(1000)
            }
            yy++
        }
```

```
        // Proceed when the job is done - otherwise VideoMetadata is null.
        val videoMetaData = modDetectionResponse?.videoMetadata
        println("Format: ${videoMetaData?.format}")
        println("Codec: ${videoMetaData?.codec}")
        println("Duration: ${videoMetaData?.durationMillis}")
        println("FrameRate: ${videoMetaData?.frameRate}")

        modDetectionResponse?.moderationLabels?.forEach { mod ->
            val seconds: Long = mod.timestamp / 1000
            print("Mod label: $seconds ")
            println(mod.moderationLabel)
        }
    }
}
```

- For API details, see the following topics in *AWS SDK for Kotlin API reference*.
  - GetCelebrityRecognition
  - GetContentModeration
  - GetLabelDetection
  - GetPersonTracking
  - GetSegmentDetection
  - GetTextDetection
  - StartCelebrityRecognition
  - StartContentModeration
  - StartLabelDetection
  - StartPersonTracking
  - StartSegmentDetection
  - StartTextDetection

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Cross-service examples for Amazon Rekognition using AWS SDKs

The following sample applications use AWS SDKs to combine Amazon Rekognition with other AWS services. Each example includes a link to GitHub, where you can find instructions on how to set up and run the application.

Examples

Detect PPE in images with Amazon Rekognition using an AWS SDK

The following code examples show how to build an app that uses Amazon Rekognition to detect Personal Protective Equipment (PPE) in images.

Java

SDK for Java 2.x

Shows how to create an AWS Lambda function that detects images with Personal Protective Equipment.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

JavaScript

SDK for JavaScript V3

Shows how to use Amazon Rekognition with the AWS SDK for JavaScript to create an application to detect personal protective equipment (PPE) in images located in an Amazon Simple Storage Service (Amazon S3) bucket. The app saves the results to an Amazon DynamoDB table, and sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

Learn how to:

- Create an unauthenticated user using Amazon Cognito.
- Analyze images for PPE using Amazon Rekognition.
- Verify an email address for Amazon SES.
- Update a DynamoDB table with results.
- Send an email notification using Amazon SES.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Detect faces in an image using an AWS SDK

The following code example shows how to:

- Save an image in an Amazon Simple Storage Service Amazon S3) bucket.
- Use Amazon Rekognition (Amazon Rekognition) to detect facial details, such as age range, gender, and emotion (smiling, etc.).
- Display those details.

Rust

SDK for Rust

> **Note**
> This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Save the image in an Amazon Simple Storage Service bucket with an **uploads** prefix, use Amazon Rekognition to detect facial details, such as age range, gender, and emotion (smiling, etc.), and display those details.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Rekognition
- Amazon S3

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Detect objects in images with Amazon Rekognition using an AWS SDK

The following code examples show how to build an app that uses Amazon Rekognition to detect objects by category in images.

.NET

AWS SDK for .NET

Shows how to use Amazon Rekognition .NET API to create an app that uses Amazon Rekognition to identify objects by category in images located in an Amazon Simple Storage Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

Java

SDK for Java 2.x

Shows how to use Amazon Rekognition Java API to create an app that uses Amazon Rekognition to identify objects by category in images located in an Amazon Simple Storage Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

JavaScript

SDK for JavaScript V3

Shows how to use Amazon Rekognition with the AWS SDK for JavaScript to create an app that uses Amazon Rekognition to identify objects by category in images located in an Amazon Simple Storage Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

Learn how to:

- Create an unauthenticated user using Amazon Cognito.
- Analyze images for objects using Amazon Rekognition.
- Verify an email address for Amazon SES.
- Send an email notification using Amazon SES.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

Kotlin

SDK for Kotlin

> **Note**
> This is prerelease documentation for a feature in preview release. It is subject to change.

Shows how to use Amazon Rekognition Kotlin API to create an app that uses Amazon Rekognition to identify objects by category in images located in an Amazon Simple Storage Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

Python

SDK for Python (Boto3)

Shows you how to use the AWS SDK for Python (Boto3) to create a web application that lets you do the following:

- Upload photos to an Amazon Simple Storage Service (Amazon S3) bucket.
- Use Amazon Rekognition to analyze and label the photos.
- Use Amazon Simple Email Service (Amazon SES) to send email reports of image analysis.

This example contains two main components: a webpage written in JavaScript that is built with React, and a REST service written in Python that is built with Flask-RESTful.

You can use the React webpage to:

- Display a list of images that are stored in your S3 bucket.
- Upload images from your computer to your S3 bucket.
- Display images and labels that identify items that are detected in the image.
- Get a report of all images in your S3 bucket and send an email of the report.

The webpage calls the REST service. The service sends requests to AWS to perform the following actions:

- Get and filter the list of images in your S3 bucket.
- Upload photos to your S3 bucket.
- Use Amazon Rekognition to analyze individual photos and get a list of labels that identify items that are detected in the photo.
- Analyze all photos in your S3 bucket and use Amazon SES to email a report.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Detect people and objects in a video with Amazon Rekognition using an AWS SDK

The following code examples show how to detect people and objects in a video with Amazon Rekognition.

Java

SDK for Java 2.x

Shows how to use Amazon Rekognition Java API to create an app to detect faces and objects in videos located in an Amazon Simple Storage Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

JavaScript

SDK for JavaScript V3

Shows how to use Amazon Rekognition with the AWS SDK for JavaScript to create an app to detect faces and objects in videos located in an Amazon Simple Storage Service (Amazon S3) bucket. The app sends the admin an email notification with the results using Amazon Simple Email Service (Amazon SES).

Learn how to:

- Create an unauthenticated user using Amazon Cognito.
- Analyze images for PPE using Amazon Rekognition.
- Verify an email address for Amazon SES.
- Send an email notification using Amazon SES.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Rekognition
- Amazon S3
- Amazon SES

Python

SDK for Python (Boto3)

Use Amazon Rekognition to detect faces, objects, and people in videos by starting asynchronous detection jobs. This example also configures Amazon Rekognition to notify an Amazon Simple Notification Service (Amazon SNS) topic when jobs complete and subscribes an Amazon Simple Queue Service (Amazon SQS) queue to the topic. When the queue receives a message about a job, the job is retrieved and the results are output.

This example is best viewed on GitHub. For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Rekognition

- Amazon SNS
- Amazon SQS

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

Save EXIF and other image information using an AWS SDK

The following code example shows how to:

- Get EXIF information from a a JPG, JPEG, or PNG file.
- Upload the image file to an Amazon Simple Storage Service (Amazon S3) bucket.
- Use Amazon Rekognition (Amazon Rekognition) to identify the three top attributes (labels in Amazon Rekognition) in the file.
- Add the EXIF and label information to a Amazon DynamoDB (DynamoDB) table in the Region.

Rust

SDK for Rust

Note
This documentation is for an SDK in preview release. The SDK is subject to change and should not be used in production.

Get EXIF information from a JPG, JPEG, or PNG file, upload the image file to an Amazon Simple Storage Service bucket, use Amazon Rekognition to identify the three top attributes (*labels* in Amazon Rekognition) in the file, and add the EXIF and label information to a Amazon DynamoDB table in the Region.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- DynamoDB
- Amazon Rekognition
- Amazon S3

For a complete list of AWS SDK developer guides and code examples, see Using Rekognition with an AWS SDK (p. 14). This topic also includes information about getting started and details about previous SDK versions.

API Reference

The Amazon Rekognition API reference is now located at Amazon Rekognition API Reference.

Amazon Rekognition Security

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that are built to meet the requirements of the most security-sensitive organizations.

Use the following topics to learn how to secure your Amazon Rekognition resources.

Topics

Identity and access management for Amazon Rekognition

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon Rekognition resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon Rekognition.

Service user – If you use the Amazon Rekognition service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon Rekognition features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon Rekognition, see Troubleshooting Amazon Rekognition identity and access (p. 491).

Service administrator – If you're in charge of Amazon Rekognition resources at your company, you probably have full access to Amazon Rekognition. It's your job to determine which Amazon Rekognition features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon Rekognition, see How Amazon Rekognition works with IAM (p. 480).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon Rekognition. To view example Amazon Rekognition identity-based policies that you can use in IAM, see Amazon Rekognition identity-based policy examples (p. 488).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *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*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one 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. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *AWS General Reference*.

IAM Users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the *IAM*

User Guide. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, a web identity provider, or the IAM Identity Center identity store. These identities are known as *federated identities*. To assign permissions to federated identities, you can create a role and define permissions for the role. When an external identity authenticates, the identity is associated with the role and is granted the permissions that are defined by it. If you use IAM Identity Center, you configure a permission set. IAM Identity Center correlates the permission set to a role in IAM to control what your identities can access after they authenticate. For more information about identity federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. For more information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for Amazon Rekognition in the *Service Authorization Reference*.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear

in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

- **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 CLI or 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*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

Every IAM entity (user or role) starts with no permissions. By default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Using identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Using resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon Rekognition works with IAM

Before you use IAM to manage access to Amazon Rekognition, you should understand what IAM features are available to use with Amazon Rekognition. To get a high-level view of how Amazon Rekognition and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

Topics

Amazon Rekognition identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon Rekognition supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon Rekognition use the following prefix before the action: `rekognition:`. For example, to grant someone permission to detect objects, scenes, or concepts in an image with the Amazon Rekognition `DeleteLabels` API operation, you include the `rekognition:DetectLabels` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Amazon Rekognition defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [
      "rekognition:action1",
      "rekognition:action2"
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "rekognition:Describe*"
```

To see a list of Amazon Rekognition actions, see Actions Defined by Amazon Rekognition in the *IAM User Guide*.

Resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

For example, to specify the `MyCollection` collection in your statement, use the following ARN:

```
"Resource": "arn:aws:rekognition:us-east-1:123456789012:collection/MyCollection"
```

To specify all instances that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:rekognition:us-east-1:123456789012:collection/*"
```

Some Amazon Rekognition actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

To see a list of Amazon Rekognition resource types and their ARNs, see Resources Defined by Amazon Rekognition in the *IAM User Guide*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon Rekognition.

Condition keys

Amazon Rekognition does not provide any service-specific condition keys, but it does support using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the *IAM User Guide*.

Amazon Rekognition resource-based policies

Amazon Rekognition doesn't support 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.

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/*"
            ]
        }
    ]
```

Amazon Rekognition IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Using temporary credentials with Amazon Rekognition

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

Amazon Rekognition supports using temporary credentials.

Service-linked roles

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

Amazon Rekognition doesn't support service-linked roles.

Service roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Amazon Rekognition supports service roles.

Using a service role may create a security issue where Amazon Rekognition is used to call another service and act on resources it shouldn't have access to. To keep your account secure, you should limit the scope of Amazon Rekognition's access to just the resources you are using. This can be done by attaching a trust policy to your IAM service role. For information on how to do this, see Cross-service confused deputy prevention (p. 505).

Choosing an IAM role in Amazon Rekognition

When you configure Amazon Rekognition to analyze stored videos, you must choose a role to allow Amazon Rekognition to access Amazon SNS on your behalf. If you have previously created a service role or service-linked role, then Amazon Rekognition provides you with a list of roles to choose from. For more information, see the section called “Configuring Amazon Rekognition Video” (p. 66).

AWS managed policies for Amazon Rekognition

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

AWS managed policy: AmazonRekognitionFullAccess

`AmazonRekognitionFullAccess` grants full access to Amazon Rekognition resources including creating and deleting collections.

You can attach the `AmazonRekognitionFullAccess` policy to your IAM identities.

Permissions details

This policy includes the following permissions.

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

AWS managed policy: AmazonRekognitionReadOnlyAccess

`AmazonRekognitionReadOnlyAccess` grants read-only access to Amazon Rekognition resources.

You can attach the `AmazonRekognitionReadOnlyAccess` policy to your IAM identities.

Permissions details

This policy includes the following permissions.

```
{
    "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:GetTextDetection",
                "rekognition:GetSegmentDetection",
                "rekognition:DescribeStreamProcessor",
                "rekognition:ListStreamProcessors",
                "rekognition:DescribeProjects",
                "rekognition:DescribeProjectVersions",
                "rekognition:DetectCustomLabels",
                "rekognition:DetectProtectiveEquipment",
                "rekognition:ListTagsForResource",
                "rekognition:ListDatasetEntries",
                "rekognition:ListDatasetLabels",
                "rekognition:DescribeDataset",
                "rekognition:ListProjectPolicies"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS managed policy: AmazonRekognitionServiceRole

`AmazonRekognitionServiceRole` allows Amazon Rekognition to call Amazon Kinesis Data Streams and Amazon SNS services on your behalf.

You can attach the `AmazonRekognitionServiceRole` policy to your IAM identities.

If using this service role, you should keep your account secure by limiting the scope of Amazon Rekognition's access to just the resources you are using. This can be done by attaching a trust policy to your IAM service role. For information on how to do this, see Cross-service confused deputy prevention (p. 505).

Permissions details

This policy includes the following permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": "arn:aws:sns:*:*:AmazonRekognition*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
            "Resource": "arn:aws:kinesis:*:*:stream/AmazonRekognition*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "kinesisvideo:GetDataEndpoint",
                "kinesisvideo:GetMedia"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS managed policy: AmazonRekognitionCustomLabelsFullAccess

This policy is for Amazon Rekognition Custom Labels; users. Use the AmazonRekognitionCustomLabelsFullAccess policy to allow users full access to the Amazon Rekognition Custom Labels API and full access to the console buckets created by the Amazon Rekognition Custom Labels console.

Permissions details

This policy includes the following permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:ListAllMyBuckets",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:GetObjectTagging",
                "s3:GetObjectVersion",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::*custom-labels*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:CopyProjectVersion",
                "rekognition:CreateProject",
                "rekognition:CreateProjectVersion",
                "rekognition:StartProjectVersion",
                "rekognition:StopProjectVersion",
                "rekognition:DescribeProjects",
                "rekognition:DescribeProjectVersions",
                "rekognition:DetectCustomLabels",
                "rekognition:DeleteProject",
                "rekognition:DeleteProjectVersion"
                "rekognition:TagResource",
                "rekognition:UntagResource",
                "rekognition:ListTagsForResource",
                "rekognition:CreateDataset",
                "rekognition:ListDatasetEntries",
                "rekognition:ListDatasetLabels",
                "rekognition:DescribeDataset",
                "rekognition:UpdateDatasetEntries",
                "rekognition:DistributeDatasetEntries",
```

```
                "rekognition:DeleteDataset",
                "rekognition:PutProjectPolicy",
                "rekognition:ListProjectPolicies",
                "rekognition:DeleteProjectPolicy"

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

Amazon Rekognition updates to AWS managed policies

View details about updates to AWS managed policies for Amazon Rekognition since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon Rekognition Document history page.

Change	Description	Date
Actions for ProjectPolicy and Custom Labels Model Copy have been added to the following managed policies: • AWS managed policy: AmazonRekognitionFullAccess (p. 484) • AWS managed policy: AmazonRekognitionCustomLabelsFullAccess (p. 486)	Amazon Rekognition added the following actions to the AmazonRekognitionCustomLabelsFullAccess and AmazonRekognitionFullAccess managed policies: • `CopyProjectVersion` • `PutProjectPolicy` • `ListProjectPolicies` • `DeleteProjectPolicy`	July 21st, 2022
Actions for ProjectPolicy and Custom Labels Model Copy have been added to the following managed policies: • AWS managed policy: AmazonRekognitionReadOnlyAccess (p. 484)	Amazon Rekognition added the following actions to the AmazonRekognitionReadOnlyAccess managed policy: • `ListProjectPolicies`	July 21st, 2022
Dataset management update for the following managed policies: • AWS managed policy: AmazonRekognitionReadOnlyAccess (p. 484) • AWS managed policy: AmazonRekognitionFullAccess (p. 484) • AWS managed policy: AmazonRekognitionCustomLabelsFullAccess (p. 486)	Amazon Rekognition added the following actions to the AmazonRekognitionReadOnlyAccess, AmazonRekognitionFullOnlyAccess, and AmazonRekognitionCustomLabelsFullAccess managed policies • `CreateDataset` • `ListDatasetEntries` • `ListDatasetLabels` • `DescribeDataset` • `UpdateDatasetEntries` • `DistributeDatasetEntries` • `DeleteDataset`	November 1, 2021

Change	Description	Date
Tagging update for AWS managed policy: AmazonRekognitionReadOnlyAccess (p. 486) and AWS managed policy: AmazonRekognitionFullAccess (p. 484)	Amazon Rekognition added new tagging actions to the AmazonRekognitionFullAccess and AmazonRekognitionReadOnlyAccess policies.	April 2, 2021
Amazon Rekognition started tracking changes	Amazon Rekognition started tracking changes for its AWS managed policies.	April 2, 2021

Amazon Rekognition identity-based policy examples

By default, IAM users and roles don't have permission to create or modify Amazon Rekognition resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the *IAM User Guide*.

Topics

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon Rekognition resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they

are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see IAM Access Analyzer policy validation in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or root users in your account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the Amazon Rekognition console

With the exception of the Amazon Rekognition Custom Labels feature, Amazon Rekognition doesn't require any addition permissions when using the Amazon Rekognition console. For information about Amazon Rekognition Custom Labels, see Step 5: Set Up Amazon Rekognition Custom Labels Console Permissions.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Example Amazon Rekognition Custom Labels policies

You can create identity-based policies for Amazon Rekognition Custom Labels. For more information, see Security.

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:GetTextDetection",
                "rekognition:GetSegmentDetection",
                "rekognition:DescribeStreamProcessor",
                "rekognition:ListStreamProcessors",
                "rekognition:DescribeProjects",
                "rekognition:DescribeProjectVersions",
                "rekognition:DetectCustomLabels",
                "rekognition:DetectProtectiveEquipment",
                "rekognition:ListTagsForResource",
               "rekognition:ListDatasetEntries",
                "rekognition:ListDatasetLabels",
                "rekognition:DescribeDataset"

            ],
            "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": "*"
        }
    ]
}
```

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
```

```
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon Rekognition Resource-Based Policy Examples

Amazon Rekognition Custom Labels uses resource-based polices, known as *project policies*, to manage copy permissions for a model version.

A project policy gives or denies permission to copy a model version from a source project to a destination project. You need a project policy if the destination project is in a different AWS account or if you want to restrict access within an AWS account, For example, you might want to deny copy permissions to a specific IAM role. For more information, see Copying a model.

Giving permission to copy a model version

The following example allows the principal `arn:aws:iam::123456789012:role/Admin` to copy the model version `arn:aws:rekognition:us-east-1:123456789012:project/my_project/version/test_1/1627045542080`.

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Principal":{
        "AWS":"arn:aws:iam::123456789012:role/Admin"
      },
      "Action":"rekognition:CopyProjectVersion",
      "Resource":"arn:aws:rekognition:us-east-1:123456789012:project/my_project/version/
test_1/1627045542080"
    }
  ]
}
```

Troubleshooting Amazon Rekognition identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon Rekognition and IAM.

Topics

- I want to allow people outside of my AWS account to access my Amazon Rekognition resources (p. 493)

I am not authorized to perform an action in Amazon Rekognition

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a *`widget`* but does not have `rekognition:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 rekognition:GetWidget on resource: my-example-widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *`my-example-widget`* resource using the `rekognition:`*`GetWidget`* action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Amazon Rekognition.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon Rekognition. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access Amazon Rekognition

To allow others to access Amazon Rekognition, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amazon Rekognition.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my Amazon Rekognition resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon Rekognition supports these features, see How Amazon Rekognition works with IAM (p. 480).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Data protection in Amazon Rekognition

The AWS shared responsibility model applies to data protection in Amazon Rekognition. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a **Name** field. This includes when you work with Rekognition or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Data encryption

The following information explains where Amazon Rekognition uses data encryption to protect your data.

Encryption at rest

Amazon Rekognition Image

Images

Images passed to Amazon Rekognition API operations may be stored and used to improve the service unless you have opted out by visiting the AI services opt-out policy page and following the process explained there. The stored images are encrypted at rest (Amazon S3) using AWS Key Management Service (SSE-KMS).

Collections

For face comparison operations that store information in a collection, the underlying detection algorithm first detects the faces in the input image, extracts a vector for each face, and then stores the facial vectors in the collection. Amazon Rekognition uses these facial vectors when performing face comparison. Facial vectors are stored as an array of floats. The data is meaningless on its own, effectively acting as a hash, and cannot be reverse engineered. The data is not further encrypted.

Amazon Rekognition Video

Videos

To analyze a video, Amazon Rekognition copies your videos into the service for processing. The video may be stored and used to improve the service unless you have opted out by visiting the AI services opt-out policy page and following the process explained there. The videos are encrypted at rest (Amazon S3) using AWS Key Management Service (SSE-KMS).

Amazon Rekognition Custom Labels

Amazon Rekognition Custom Labels encrypts your data at rest.

Images

To train your model, Amazon Rekognition Custom Labels makes a copy of your source training and test images. The copied images are encrypted at rest in Amazon Simple Storage Service (S3) using server-side encryption with an AWS KMS key that you provide or an AWS owned KMS key. Amazon Rekognition Custom Labels only supports symmetric KMS keys. Your source images are unaffected. For more information, see Training an Amazon Rekognition Custom Labels Model.

Models

By default, Amazon Rekognition Custom Labels encrypts trained models and manifest files stored in Amazon S3 buckets using server-side encryption with an AWS owned key. For more information, see Protecting Data Using Server-Side Encryption. Training results are written to the bucket specified in the `OutputConfig` input parameter to CreateProjectVersion. The training results are encrypted using the configured encryption settings for the bucket (`OutputConfig`).

Console bucket

The Amazon Rekognition Custom Labels console creates an Amazon S3 bucket (console bucket) that you can use to manage your projects. The console bucket is encrypted using the default Amazon S3 encryption. For more information, see Amazon Simple Storage Service default encryption for S3 buckets. If you are using your own KMS key, configure the console bucket after it is created. For more information, see Protecting Data Using Server-Side Encryption. Amazon Rekognition Custom Labels blocks public access to the console bucket.

Encryption in transit

Amazon Rekognition API endpoints only support secure connections over HTTPS. All communication is encrypted with Transport Layer Security (TLS).

Key management

You can use AWS Key Management Service (KMS) to manage keys for the input images and videos you store in Amazon S3 buckets. For more information, see AWS Key Management Service concepts.

Internetwork traffic privacy

An Amazon Virtual Private Cloud (Amazon VPC) endpoint for Amazon Rekognition is a logical entity within a VPC that allows connectivity only to Amazon Rekognition. Amazon VPC routes requests to Amazon Rekognition and routes responses back to the VPC. For more information, see VPC Endpoints in the *Amazon VPC User Guide*. For information about using Amazon VPC endpoints with Amazon Rekognition see Using Amazon Rekognition with Amazon VPC endpoints (p. 503).

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. 25).

Using CloudWatch metrics for Rekognition

To use metrics, you must specify the following information:

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

You can get monitoring data for Rekognition using the AWS Management Console, the AWS CLI, or the CloudWatch API. You can also use the CloudWatch API through one of the Amazon AWS Software Development Kits (SDKs) or the CloudWatch API tools. The console displays a series of graphs based on

the raw data from the CloudWatch API. Depending on your needs, you might prefer to use either the graphs displayed in the console or retrieved from the API.

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

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

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

Access Rekognition metrics

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

To view metrics (console)

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

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

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

- Open the AWS CLI and enter the following command:

  ```
  aws cloudwatch get-metric-statistics --metric-name SuccessfulRequestCount
  --start-time 2017-1-1T19:46:20 --end-time 2017-1-6T19:46:57 --
  ```

```
period 3600 --namespace AWS/Rekognition --statistics Sum --dimensions
Name=Operation,Value=DetectFaces --region us-west-2
```

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

To access metrics (CloudWatch API)

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

Create an alarm

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

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

To set an alarm (console)

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

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

   Note

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

To set an alarm (AWS CLI)

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

  ```
  aws cloudwatch put-metric-alarm --alarm-name UserErrors --alarm-description
  "Alarm when more than 10 user errors occur" --metric-name UserErrorCount --
  ```

```
namespace AWS/Rekognition --statistic Average --period 300 --threshold 10
--comparison-operator GreaterThanThreshold --evaluation-periods 2 --alarm-
actions arn:aws:sns:us-west-2:111111111111:UserError --unit Count
```

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

To set an alarm (CloudWatch API)

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

CloudWatch metrics for Rekognition

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

You can also see an aggregate view of Rekognition metrics from the Rekognition console. For more information, see Exercise 4: See aggregated metrics (console) (p. 25).

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.

Metric	Description
	Unit: Count Valid statistics: `Sum,Average`
DetectedLabelCount	The number of labels detected with the `DetectLabels` operation. Unit: Count Valid statistics: `Sum,Average`
ServerErrorCount	The number of server errors. The response code range for a server error is 500 to 599. Unit: Count Valid statistics: `Sum, Average`
UserErrorCount	The number of user errors (invalid parameters, invalid image, no permission, etc). The response code range for a user error is 400 to 499. Unit: Count Valid statistics: `Sum,Average`
MinInferenceUnits	The minimum number of inference units specified during the `StartProjectVersion` request. Unit: Count Valid statistics: `Average`
MaxInferenceUnits	The maximum number of inference units specified during the `StartProjectVersion` request. Unit: Count Valid statistics: `Average`
DesiredInferenceUnits	The number of inference units to which Rekognition is scaling up or down. Unit: Count Valid statistics: `Average`
InServiceInferenceUnits	The number of inference units that the model is using. Unit: Count Valid statistics: `Average` It is recommended that you use the Average statistic to obtain the 1 minute average of how many instances are used.

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*.

CloudWatch dimension for Rekognition Custom Labels

The following table displays the CloudWatch dimensions available for use with Rekognition Custom Labels:

Dimension	Description
ProjectName	The name of the Rekognition Custom Labels project you created with `CreateProject`.
VersionName	The name of the Rekognition Custom Labels project version you created with `CreateProjectVersion`.

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 all API calls for Amazon Rekognition as events. The calls captured include calls from the Amazon Rekognition console and code calls to the Amazon Rekognition API operations. 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, see the AWS CloudTrail User Guide.

Amazon Rekognition information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When 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 AWS 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 the following:

- 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

All Amazon Rekognition actions are logged by CloudTrail and are documented in the Amazon Rekognition API reference. For example, calls to the `CreateCollection`, `CreateStreamProcessor` and `DetectCustomLabels` actions generate entries in the CloudTrail log files.

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 AWS Identity and Access Management (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.

Understanding 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 aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry with actions for the following API: `StartLabelDetection` and `DetectLabels`.


```
{
    "Records": [

        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AIDAJ45Q7YFFAREXAMPLE",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/JorgeSouza",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AIDAJ45Q7YFFAREXAMPLE",
                        "arn": "arn:aws:iam::111122223333:role/Admin",
                        "accountId": "111122223333",
                        "userName": "Admin"
                    },
                    "webIdFederationData": {},
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2020-06-30T20:10:09Z"
                    }
                }
            },
            "eventTime": "2020-06-30T20:42:14Z",
            "eventSource": "rekognition.amazonaws.com",
            "eventName": "StartLabelDetection",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/3",
            "requestParameters": {
                "video": {
                    "s3Object": {
                        "bucket": "my-bucket",
```

```
                        "name": "my-video.mp4"
                    }
                }
            },
            "responseElements": {
                "jobId": "653de5a7ee03bd5083edde98ea8fce5794fcea66d077bdd4cfb39d71aff8fc25"
            },
            "requestID": "dfcef8fc-479c-4c25-bef0-d83a7f9a7240",
            "eventID": "b602e460-c134-4ecb-ae78-6d383720f29d",
            "readOnly": false,
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        },
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AIDAJ45Q7YFFAREXAMPLE",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/JorgeSouza",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AIDAJ45Q7YFFAREXAMPLE",
                        "arn": "arn:aws:iam::111122223333:role/Admin",
                        "accountId": "111122223333",
                        "userName": "Admin"
                    },
                    "webIdFederationData": {},
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2020-06-30T21:19:18Z"
                    }
                }
            },
            "eventTime": "2020-06-30T21:21:47Z",
            "eventSource": "rekognition.amazonaws.com",
            "eventName": "DetectLabels",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/3",
            "requestParameters": {
                "image": {
                    "s3Object": {
                        "bucket": "my-bucket",
                        "name": "my-image.jpg"
                    }
                }
            },
            "responseElements": null,
            "requestID": "5a683fb2-aec0-4af4-a7df-219018be2155",
            "eventID": "b356b0fd-ea01-436f-a9df-e1186b275bfa",
            "readOnly": true,
            "eventType": "AwsApiCall",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

Using Amazon Rekognition with Amazon VPC endpoints

If you use Amazon Virtual Private Cloud (Amazon VPC) to host your AWS resources, you can establish a private connection between your VPC and Amazon Rekognition. You can use this connection to enable Amazon Rekognition to communicate with your resources on your VPC without going through the public internet.

Amazon VPC is an AWS service that you can use to launch AWS resources in a virtual network that you define. With a VPC, you have control over your network settings, such the IP address range, subnets, route tables, and network gateways. With VPC endpoints, the AWS network handles the routing between the VPC and AWS services.

To connect your VPC to Amazon Rekognition, you define an interface VPC endpoint for Amazon Rekognition. An interface endpoint is an elastic network interface with a private IP address that serves as an entry point for traffic destined to a supported AWS service. The endpoint provides reliable, scalable connectivity to Amazon Rekognition—and it doesn't require an internet gateway, a network address translation (NAT) instance, or a VPN connection. For more information, see What Is Amazon VPC in the *Amazon VPC User Guide*.

Interface VPC endpoints are enabled by AWS PrivateLink. This AWS technology enables private communication between AWS services by using an elastic network interface with private IP addresses.

> **Note**
> All Amazon Rekognition Federal Information Processing Standard (FIPS) endpoints are supported by AWS PrivateLink.

Creating Amazon VPC endpoints for Amazon Rekognition

You can create two types of Amazon VPC endpoints to use with Amazon Rekognition.

- A VPC endpoint to use with Amazon Rekognition operations. For most users, this is the most suitable type of VPC endpoint.
- A VPC endpoint for Amazon Rekognition operations with endpoints that comply with the Federal Information Processing Standard (FIPS) Publication 140-2 US government standard.

To start using Amazon Rekognition with your VPC, use the Amazon VPC console to create an interface VPC endpoint for Amazon Rekognition. For instructions, see the procedure "To create an interface endpoint to an AWS service using the console" in Creating an Interface Endpoint. Note the following procedure steps:

- Step 3 –For **Service category**, choose *AWS services*.
- Step 4 – For **Service Name**, choose one of the following options:
  - *com.amazonaws.region.rekognition* – Creates a VPC endpoint for Amazon Rekognition operations.
  - *com.amazonaws.region.rekognition-fips* – Creates a VPC endpoint for Amazon Rekognition operations with endpoints that comply with the Federal Information Processing Standard (FIPS) Publication 140-2 US government standard.

For more information, see Getting Started in the *Amazon VPC User Guide*.

Create a VPC endpoint policy for Amazon Rekognition

You can create a policy for Amazon VPC endpoints for Amazon Rekognition to specify the following:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling Access to Services with VPC Endpoints in the *Amazon VPC User Guide*.

The following example policy enables users connecting to Amazon Rekognition through the VPC endpoint to call the `DetectFaces` API operation. The policy prevents users from performing other Amazon Rekognition API operations through the VPC endpoint.

Users can still call other Amazon Rekognition API operations from outside the VPC. For information about how to deny access to Amazon Rekognition API operations that are outside the VPC, see Amazon Rekognition identity-based policies (p. 481).

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

To modify the VPC endpoint policy for Amazon Rekognition

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. If you have not already created the endpoint for Amazon Rekognition choose **Create Endpoint**. Then select **com.amazonaws.*Region*.rekognition** and choose **Create endpoint**.
3. In the navigation pane, choose **Endpoints**.
4. Select the **com.amazonaws.*Region*.rekognition** endpoint and choose the **Policy** tab in the lower half of the screen.
5. Choose **Edit Policy** and make the changes to the policy.

Compliance validation for Amazon Rekognition

Third-party auditors assess the security and compliance of Amazon Rekognition as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using Amazon Rekognition is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in Amazon Rekognition

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, Amazon Rekognition offers several features to help support your data resiliency and backup needs.

Configuration and vulnerability analysis in Amazon Rekognition

Configuration and IT controls are a shared responsibility between AWS and you, our customer. For more information, see the AWS shared responsibility model.

Cross-service confused deputy prevention

In AWS, cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to act on another customer's resources even though it shouldn't have the proper permissions, resulting in the confused deputy problem.

To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that Amazon Rekognition gives another service to the resource.

If the value of `aws:SourceArn` does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both keys to limit permissions. If you use both keys and the `aws:SourceArn` value contains the account ID, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement.

Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The value of `aws:SourceArn` must be the ARN of the resource used by Rekognition, which is specified with the following format: `arn:aws:rekognition:region:account:resource`.

The recommended approach to the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full resource ARN.

If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` key with wildcard characters (`*`) for the unknown portions of the ARN. For example, `arn:aws:rekognition:*:111122223333:*`.

In order to protect against the confused deputy problem, carry out the following steps:

1. In the navigation pane of the IAM console choose the **Roles** option. The console will display the roles for your current account.
2. Choose the name of the role that you want to modify. The role you modify should have the **AmazonRekognitionServiceRole** permissions policy. Select the **Trust relationships** tab.
3. Choose **Edit trust policy**.
4. On the **Edit trust policy** page, replace the default JSON policy with a policy that utilizes one or both of the `aws:SourceArn` and `aws:SourceAccount` global condition context keys. See the following example policies.
5. Choose **Update policy**.

The following examples are trust policies that show how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in Amazon Rekognition to prevent the confused deputy problem.

If you are working stored and streaming videos, you could use a policy like the following in your IAM role:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"rekognition.amazonaws.com",
            "AWS":"arn:User ARN"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"Account ID"
            },
            "StringLike":{
               "aws:SourceArn":"arn:aws:rekognition:region:111122223333:streamprocessor/*"
            }
         }
      }
   ]
}
```

If you are working exclusively with stored video, you could use a policy like the following in your IAM role (note that you don't have to include the `StringLike` argument that specifies the `streamprocessor`):

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

Infrastructure security in Amazon Rekognition

As a managed service, Amazon Rekognition is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Amazon Rekognition through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Guidelines and quotas in Amazon Rekognition

The following sections provide guidelines and quotas when using Amazon Rekognition. There are two kinds of quotas. *Set quotas* such as maximum image size cannot be changed. *Default quotas* listed on the AWS Service Quotas page can be changed by following the procedure described in the Default quotas (p. 509) section.

Topics

Supported regions

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

Set quotas

The following is a list of limits in Amazon Rekognition that cannot be changed. For information about limits you can change, such as Transactions Per Second (TPS) limits, see Default quotas (p. 509).

For Amazon Rekognition Custom Labels limits, see Guidelines and Quotas in Amazon Rekognition Custom Labels.

Amazon Rekognition Image

- Maximum image size stored as an Amazon S3 object is limited to 15 MB.
- The minimum image dimensions is 80 pixels for both height and width. The minimum image dimension for `DetectProtectiveEquipment` is 64 pixels for both height and width.
- The maximum image dimensions for `DetectProtectiveEquipment` is 4096 pixels for both width and height.
- To be detected by `DetectProtectiveEquipment`, a person must be no smaller than 100x100 pixels in an image with 800x1300. Images with dimensions higher than 800x1300 pixels will need a larger minimum person size proportionally.
- To be detected, a face must be no smaller than 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. The limit is 4 MB for the `DetectProtectiveEquipment` API.
- 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.
- DetectText can detect up to 100 words in an image.
- `DetectProtectiveEquipment` can detect Personal Protective Equipment on up to 15 people.

For best practice information about images and facial comparison, see Best practices for sensors, input images, and videos (p. 100).

Amazon Rekognition Video stored video

- Amazon Rekognition Video can analyze stored videos up to 10GB in size.
- Amazon Rekognition Video can analyze stored videos up to 6 hours in length.
- Amazon Rekognition Video supports a maximum of 20 concurrent jobs per account.
- Stored videos must be encoded using the H.264 codec. The supported file formats are MPEG-4 and MOV.
- Any Amazon Rekognition Video API that analyzes audio data only supports AAC audio codecs.
- The Time To Live (TTL) period for pagination tokens is 24 hours. Pagination tokens are in the `NextToken` field retured by Get operations such as `GetLabeldetection`.

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.
- Any Amazon Rekognition Video API that analyzes audio data only supports ACC audio codecs.

Default quotas

A list of default quotas can be found at AWS Service Quotas. These limits are defaults and can be changed. To request a limit increase, you create a case. To see your current quota limits (applied quota values), see Amazon Rekognition Service Quotas. To view your TPS utilization history for Amazon Rekognition Image APIs, see the Amazon Rekognition Service Quotas page and choose a specific API operation to see the history for that operation.

Topics

Calculate TPS quota change

What is the new limit you are requesting? Transactions Per Second (TPS) are most relevant at the peak of an expected workload. It is important to understand the max concurrent API calls at the peak of a workload and time for responses (5 - 15 seconds). Please note, 5 seconds should be the minimum. Below are two examples:

- Example 1: The max concurrent Face Authentication (CompareFaces API) users I expect at the beginning of my busiest hour is 1000. These responses will be spread over a period of 10 seconds. Therefore, the TPS required is 100 (1000/10) for the CompareFaces API in my relevant region.
- Example 2: The max concurrent Object Detection (DetectLabels API) calls that are expected at the beginning of my busiest hour is 250. These responses will be spread over a period of 5 seconds. Therefore, the TPS required is 50 (250/5) for the DetectLabels API in my relevant region.

Best practices for TPS quotas

Recommended best practices for Transactions Per Second (TPS) include smoothening spiky traffic, configuring retries, and configuring exponential backoff and jitter.

1. Smooth spiky traffic. Spiky traffic affects throughput. To get maximum throughput for the allotted transactions per second (TPS), use a queueing serverless architecture or another mechanism to “smooth” traffic so it is more consistent. For code samples and references for serverless large-scale image and video processing with Rekognition, see Large scale image and video processing with Amazon Rekognition.
2. Configure retries. Follow the guidelines at the section called “Error handling” (p. 92) to configure retries for the errors that allow them.
3. Configure exponential backoff and jitter. Configuring exponential backoff and jitter as you configure retries allows you to improve the achievable throughput. See Error retries and exponential backoff in AWS.

Create a case to change TPS quotas

To create a case, go to Create Case and answer the following questions:

- Have you implemented the the section called “Best practices for TPS quotas” (p. 510) for smoothening your traffic spikes and configuring retries, exponential backoff, and jitter?
- Have you calculated the TPS quota change you need? If not, see the section called “Calculate TPS quota change” (p. 509).
- Have you checked your TPS usage history to more accurately predict your future needs? To view your TPS usage history, see the Amazon Rekognition Service Quotas page.
- What is your use case?
- What APIs do you plan to use?
- What regions do you plan to use these APIs in?
- Are you able to spread the load across multiple regions?
- How many images do you process daily?
- How long do you expect to sustain this volume (Is it a one-time spike or ongoing)?
- How are you blocked by the default limit? Review the following exception table to confirm the scenario that you are encountering.

Error Code	**Exception**	**Message**	**What does it mean?**	**Can it be retried?**
HTTP status code 400	ProvisionedThroughputExceededException	*Provisioned Rate exceeded.*	Indicates throttling. You can retry or evaluate a limit increase request.	Yes

Error Code	Exception	Message	What does it mean?	Can it be retried?
HTTP status code 400	ThrottlingException	*Slow down; sudden increase in rate of requests.*	You might be sending spiky traffic and encountering throttling. You should shape the traffic and make it more smooth and consistent. Then configure additional retries. See best practices.	Yes
HTTP status code 5xx	ThrottlingException (HTTP 500)	*Service Unavailable*	Indicates that the backend is scaling up to support the action. You should retry the request.	Yes

For a detailed understanding of the error codes, see the section called “Error handling” (p. 92).

Note
These limits depend on the region you are in. Making a case to change a limit affects the API operation you request, in the region you request it. Other API operations and regions are not affected.

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:** April 28th, 2022

update-history-change	update-history-description	update-history-date
Amazon Rekognition Video can now detect labels in streaming video (p. 512)	Amazon Rekognition Video can detect labels such as pets and packages in streaming video. This is done using `ConnectedHome` settings on stream processors created with the `CreateStreamProcessor` operation.	April 28, 2022
API reference moved out of the Amazon Rekognition developer guide (p. 512)	The Amazon Rekognition API reference is now available at Amazon Rekognition API Reference.	February 24, 2022
A new node in the table of contents shows Amazon Rekognition examples hosted on GitHub (p. 512)	Updated code examples from the AWS Code Examples Repository now show up in a separate node in the Amazon Rekognition developer guide for easier access.	October 22, 2021
Amazon Rekognition can detect black frames and primary program content in video segments (p. 512)	Amazon Rekognition can identify black frames, color bars, opening credits, end credits, studio logos, and primary program content as technical cues in a video using the `StartSegmentDetection` and `GetSegmentDetection` operations.	June 7, 2021
Amazon Rekognition DetectText can detect up to 100 words in an image (p. 512)	You can use the Amazon Rekognition `DetectText` operation to detect up to 100 words in an image.	May 21, 2021
Amazon Rekognition now supports tagging (p. 512)	You can now use tags to identify, organize, search for, and filter Amazon Rekognition collections, stream processors, and Custom Labels models.	March 25, 2021

Amazon Rekognition can now detect personal protect equipment (p. 512)	Amazon Rekognition can now detect hand covers, face covers, and head covers on persons in an image.	October 15, 2020
Amazon Rekognition has new content moderation categories (p. 512)	Amazon Rekognition content moderation categories now include 6 new categories: Drugs, Tobacco, Alcohol, Gambling, Rude Gestures, and Hate Symbols.	October 12, 2020
New tutorial for displaying Amazon Rekognition Video results from Kinesis Video Streams locally (p. 512)	You can display the output of Amazon Rekognition Video from a streaming video in Kinesis Video Streams in a local video feed.	July 20, 2020
New Amazon Rekognition tutorial for using Gstreamer (p. 512)	Using Gstreamer, you can ingest a livestream video from a device camera source to Amazon Rekognition Video through Kinesis Video Streams.	July 17, 2020
Amazon Rekognition now supports segmentation of stored videos (p. 512)	With the asynchronous Amazon Rekognition Video segmentation API you can detect black frames, color bars, end credits, and shots in stored videos.	June 22, 2020
Amazon Rekognition now supports Amazon VPC endpoint policies (p. 512)	By specifying a policy you can restrict access to an Amazon Rekognition Amazon VPC endpoint.	March 3, 2020
Amazon Rekognition now supports the detection of text in stored videos (p. 512)	You can use the Amazon Rekognition Video API to asynchronously detect text in a stored video.	February 17, 2020
Amazon Rekognition now supports Augmented AI (Preview) and Amazon Rekognition Custom Labels (p. 512)	With Amazon Rekognition Custom Labels you can detect specialized objects, scenes, and concepts in images by creating your own machine learning model. DetectModerationLabels now supports Amazon Augmented AI (Preview).	December 3, 2019
Amazon Rekognition now supports AWS PrivateLink (p. 512)	With AWS PrivateLink you can establish a private connection between your VPC and Amazon Rekognition.	September 12, 2019

Amazon Rekognition face filtering (p. 512)	Amazon Rekognition adds enhanced face filtering support to the IndexFaces API operation, and introduces face filtering for the CompareFaces and SearchFacesByImage API operations.	September 12, 2019
Amazon Rekognition Video examples updated (p. 512)	Amazon Rekognition Video example code updated to create and configure the Amazon SNS topic and Amazon SQS queue.	September 5, 2019
Ruby and Node.js examples added (p. 512)	Amazon Rekognition Image Ruby and Node.js examples added for synchronous label and face detection.	August 19, 2019
Unsafe content detection updated (p. 512)	Amazon Rekognition unsafe content detection can now detect violent content.	August 9, 2019
GetContentModeration operation updated (p. 512)	GetContentModeration now returns the version of the moderation detection model used to detect unsafe content.	February 13, 2019
GetLabelDetection and DetectModerationLabels operations updated (p. 512)	GetLabelDetection now returns bounding box information for common objects and a hierarchical taxonomy of detected labels. The version of the model used for label detection is now returned. DetectModerationLabels now returns the version of the model used for detecting unsafe content.	January 17, 2019
DetectFaces and IndexFaces operation updated (p. 512)	This release updates the DetectFaces and IndexFaces operation. When the Attributes input parameter is set to ALL, the face location landmarks includes 5 new landmarks: upperJawlineLeft, midJawlineLeft, chinBottom, midJawlineRight, upperJawlineRight.	November 19, 2018
DetectLabels operation updated (p. 512)	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. 512)	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. 512)	You can now get information about an existing collection by calling the DescribeCollection operation.	August 22, 2018
New Python examples (p. 512)	Python examples have been added to the Amazon Rekognition Video content along with some content reorganization.	June 26, 2018
Updated content layout (p. 512)	The Amazon Rekognition Image content has been reorganized along with new Python and C# examples.	May 29, 2018
Amazon Rekognition supports AWS CloudTrail (p. 512)	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. 512)	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. 512)	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. 512)	Amazon Rekognition can now analyze images for celebrities. For more information, see Recognizing Celebrities.	June 8, 2017
Image moderation (p. 512)	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. 512)	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. 512)	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*.