デベロッパーガイド

Amazon Rekognition



Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Rekognition: デベロッパーガイド

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは、Amazon のものではない製品またはサービスにも関連して、お客様に混乱を招いたり Amazon の信用を傷つけたり失わせたりするいかなる形においても使用することはできません。Amazon が所有していない他のすべての商標は、それぞれの所有者の所有物であり、Amazon と提携、接続、または後援されている場合とされていない場合があります。

Table of Contents

Amazon Rekognition とは ... 1
主な機能 ... 1
ユースケース ... 2
利点 ... 3
Amazon Rekognition と HIPAA の適格性 ... 4
Amazon Rekognition ユーザー を初めてお使いになる方向けの情報 ... 4
Rekognition の仕組み ... 6
Rekognition の分析タイプについて ... 7
ラベル ... 9
カスタムラベル ... 10
Face Liveness による検出 ... 11
顔の検出と分析 ... 11
顔検索 ... 11
人物の動線 ... 12
個人用保護具 ... 12
有名人 ... 12
テキストの検出 ... 12
不適切または不快なコンテンツ ... 13
カスタマイズ ... 13
一括分析 ... 14
Rekognition の画像およびビデオオペレーションについて ... 14
Amazon Rekognition Image のオペレーション ... 14
Amazon Rekognition Video オペレーション ... 14
非ストレージ型およびストレージ型のオペレーション ... 15
AWS SDK または HTTP による Amazon Rekognition API オペレーションの呼び出し ... 16
非ストレージ型およびストレージ型の API オペレーション ... 16
非ストレージ型オペレーション ... 16
ストレージ型 API オペレーション ... 18
モデルのバージョニングについて ... 20
入門 ... 22
ステップ 1: AWS アカウントを設定してユーザーを作成する ... 22
AWS アカウントとユーザーを作成する ... 23
ステップ 2: AWS CLI と AWS SDKsを設定する ... 26
プログラマチックアクセス権を付与する ... 28

AWS SDKs の使用 31
ステップ 3: AWS CLI と AWS SDK API の使用を開始する 32
AWS CLI 例のフォーマット 33
次のステップ 33
ステップ 4: コンソールの使用開始 33
コンソールのアクセス権限のセットアップ 34
演習 1: 物体とシーンを検出する (コンソール) 38
演習 2: 顔を分析する (コンソール) 45
演習 3: 顔を比較する (コンソール) 48
演習 4: 集計メトリクスを参照する (コンソール) 51
イメージや動画の操作 53
イメージの操作 53
イメージの仕様 54
Amazon S3 バケットに保存されたイメージの分析 56
ローカルファイルシステムの使用 72
境界ボックスの表示 88
イメージの向きおよび境界ボックス座標の取得 100
保存済み動画分析での作業 111
分析のタイプ 111
Amazon Rekognition Video API の概要 112
Amazon Rekognition Video オペレーションを呼び出す 114
Amazon Rekognition Video の設定 121
保存されたビデオの分析 (SDK) 125
ビデオの分析 (AWS CLI) 154
リファレンス: ビデオ分析結果の通知 157
Amazon Rekognition Video のトラブルシューティング 159
ストリーミングビデオイベントの操作 161
Amazon Rekognition Video ストリームプロセッサオペレーションの概要 162
新しいストリームプロセッサにタグを追加する 163
既存のストリームプロセッサにタグを追加する 163
ストリームプロセッサ内のタグを一覧表示する 164
ストリームプロセッサからタグを削除する 165
エラー処理 165
エラーコンポーネント 165
エラーメッセージおよびコード 166
アプリケーションのエラー処理 171

FedRAMP で Amazon Rekognition の使用 172
センサー、入力イメージ、ビデオのベストプラクティス 177
Amazon Rekognition イメージオペレーションのレイテンシ 177
顔比較用の入力イメージに関する推奨事項 178
顔のオペレーションの入力画像に関する一般推奨事項 178
コレクション内で顔を検索するときの推奨事項 179
カメラセットアップの推奨事項 (画像およびビデオ) 179
カメラセットアップの推奨事項 (保存済みビデオおよびストリーミングビデオ) 181
カメラセットアップの推奨事項 (ストリーミングビデオ) 182
Face Liveness の使用に関する推奨事項 182
オブジェクトおよび概念の検出 184
レスポンスオブジェクトのラベル付け 185
境界ボックス 185
信頼スコア 186
親 186
カテゴリ 187
エイリアス 187
イメージプロパティ 187
モデルのバージョン 189
包含フィルターと除外フィルター 189
結果のソートと集計 190
イメージ内のラベルの検出 190
DetectLabels オペレーションのリクエスト 201
DetectLabels のレスポンス 202
DetectLabels レスポンスの変換 206
ビデオ内のラベルの検出 210
StartLabelDetection リクエスト 210
GetLabelDetection オペレーションレスポンス 212
GetLabelDetection レスポンスの変換 218
ストリーミングビデオイベント内のラベルの検出 227
Amazon Rekognition Video と Amazon Kinesis のリソースを設定する 227
ストリーミングビデオイベントに対するラベル検出オペレーションの呼び出し 232
カスタムラベルの検出 239
顔の検出と分析 240
顔検出および顔比較の概要 241
顔属性のガイドライン 243

イメージ内の顔の検出 ... 243
DetectFaces オペレーションのリクエスト ... 255
DetectFaces オペレーションのレスポンス ... 255
イメージ間の顔の比較 ... 263
CompareFaces オペレーションのリクエスト ... 275
CompareFaces オペレーションのレスポンス ... 275
保存済みビデオ内の顔の検出 ... 278
GetFaceDetection オペレーションレスポンス ... 287
コレクション内での顔の検索 ... 293
顔コレクション、顔、ユーザーの管理 ... 296
コレクションの管理 ... 296
コレクション内の顔の管理 ... 297
コレクション内のユーザーの管理 ... 297
類似度しきい値を使用して顔を関連付け、照合する ... 298
顔の関連付けに類似度しきい値を使用する ... 298
類似度しきい値を使用した顔のマッチング ... 298
一般的なシナリオで顔のインデックスを作成するためのガイダンス ... 299
重要または公共安全のアプリケーション ... 299
写真共有とソーシャルメディアアプリケーション ... 299
一般的な使用 ... 299
コレクション内の顔とユーザーの検索 ... 300
公共安全に関係するユースケース ... 300
Amazon Rekognition を使用して公共の安全を支援する ... 302
コレクションの作成 ... 303
CreateCollection オペレーションのリクエスト ... 309
CreateCollection オペレーションのレスポンス ... 309
タグコレクション ... 310
新しいコレクションにタグを追加する ... 310
既存のコレクションにタグを追加する ... 311
コレクション内のタグを一覧表示する ... 312
コレクションからタグを削除する ... 314
コレクションの一覧表示 ... 314
ListCollections オペレーションのリクエスト ... 321
ListCollections オペレーションのレスポンス ... 321
コレクションの定義 ... 322
DescribeCollection オペレーションのリクエスト ... 329

DescribeCollection オペレーションのレスポンス 329
コレクションの削除 330
DeleteCollection オペレーションのリクエスト 336
DeleteCollection オペレーションのレスポンス 336
コレクションへの顔の追加 337
顔のフィルタリング 338
IndexFaces オペレーションのリクエスト 347
IndexFaces オペレーションのレスポンス 348
コレクション内の顔と関連ユーザーを一覧表示 356
ListFaces オペレーションのリクエスト 362
ListFaces オペレーションのレスポンス 363
コレクションからの顔の削除 364
DeleteFaces オペレーションのリクエスト 370
DeleteFaces オペレーションのレスポンス 370
ユーザーの作成 371
ユーザーの削除 374
ユーザーへの顔の関連付け 377
AssociateFaces オペレーションのレスポンス 380
顔とユーザーとの関連付けの解除 381
DisassociateFaces オペレーションのレスポンス 384
コレクション内のユーザーの一覧表示 385
ListUsers オペレーションのレスポンス 388
顔 (顔 ID) の検索 389
SearchFaces オペレーションのリクエスト 395
SearchFaces オペレーションのレスポンス 395
顔 (イメージ) の検索 396
SearchFacesByImage オペレーションのリクエスト 404
SearchFacesByImage オペレーションのレスポンス 405
ユーザー (顔 ID/ユーザー ID) を検索 406
SearchUsers オペレーションのリクエスト 410
SearchUsers オペレーションのレスポンス 410
ユーザーの検索 (イメージ) 412
SearchUsersByImage オペレーションのリクエスト 415
SearchUsersByImage オペレーションのレスポンス 416
保存したビデオでの顔の検索 417
GetFaceSearch オペレーションレスポンス 426

ストリーミングビデオのコレクション内での顔検索 431
Amazon Rekognition Video と Amazon Kinesis のリソースを設定する 432
ストリーミングビデオの顔検索 436
GStreamer プラグインを使用したストリーミング 459
ストリーミングビデオのトラブルシューティング 462
人物の動線の検出 470
GetPersonTracking オペレーションレスポンス 479
個人用保護具を検出する。 484
PPE の種類です。 485
フェイスカバー 485
ハンドカバー 485
ヘッドカバー 485
PPE 検出の信頼度 486
イメージ内で検出された PPE の概要です。 486
チュートリアル: PPE でイメージを検出する AWS Lambda 関数の作成 487
PPE 検出 API を理解する 487
画像を供給する 487
保護装置の検出レスポンスの概要 488
画像から PPE を検出します。 494
例：バウンディングボックスとフェイスカバー 506
有名人の認識 522
顔検索と比較した有名人の認識 522
イメージ内の有名人の認識 523
RecognizeCelebrities の呼び出し 524
RecognizeCelebrities オペレーションのリクエスト 533
RecognizeCelebrities オペレーションのレスポンス 533
保存済みビデオ内の有名人の認識 536
GetCelebrityRecognition オペレーションのレスポンス 552
有名人に関する情報の取得 554
GetCelebrityInfo の呼び出し 554
GetCelebrityInfo オペレーションのリクエスト 560
GetCelebrityInfo オペレーションの応答 560
コンテンツのモデレーション 561
イメージとビデオのモデレーション API の使用 563
ラベルのカテゴリ 563
コンテンツタイプ 574

信頼度 574
バージョニング 575
並び替えと集計 575
カスタムモデレーションアダプターのステータス 576
コンテンツモデレーションバージョン 7 のテストと API レスポンスの変換 576
AWS コンテンツモデレーションバージョン 7 の SDK と使用ガイド 577
バージョン 6.1 から 7 へのラベルのマッピング 577
不適切なイメージの検出 582
イメージ内の不適切なコンテンツの検出 582
..... 582
DetectModerationLabels オペレーションのリクエスト 589
DetectModerationLabels オペレーションのレスポンス 590
不適切な保存動画の検出 591
GetContentModeration オペレーションレスポンス 600
カスタムモデレーションによる精度の向上 602
アダプターの作成と使用 603
データの準備 607
CLI と SDKs AWS を使用したアダプターの管理 609
カスタムモデレーションのアダプターのチュートリアル 616
アダプターの評価と改善 634
マニフェストファイルの形式 636
トレーニングアダプターのベストプラクティス 641
AutoUpdate のアクセス権限のセットアップ 642
AWS Rekognition のヘルスダッシュボード通知 644
Amazon A2I による不適切なコンテンツの確認 646
テキストの検出 652
イメージ内のテキストの検出 654
DetectText オペレーションリクエスト 662
DetectText オペレーションレスポンス 663
保存済みビデオ内のテキスト検出 669
フィルター 678
GetTextDetection レスポンス 679
ビデオセグメントの検出 685
テクニカルキュー 686
ブラックフレーム 686
クレジット 686

カラーバー ... 687
スレート ... 687
スタジオロゴ ... 687
コンテンツ ... 687
ショット検出 ... 688
Amazon Rekognition Video セグメント検出 API について ... 689
Amazon Rekognition セグメント API を利用する ... 689
セグメント解析の開始 ... 689
セグメント解析結果の取得 ... 691
例: 保存されたビデオ内のセグメントの検出 ... 696
顔のライブネスの検出 ... 709
ユーザー側の Face Liveness 要件 ... 711
アーキテクチャ図とシーケンス図 ... 712
前提条件 ... 714
ステップ 1: AWS アカウントを設定する ... 714
ステップ 2: Face Liveness AWS SDK をセットアップする ... 714
ステップ 3: Amplify AWS リソースを設定する ... 715
Face Liveness を検出するためのベストプラクティス ... 715
Amazon Rekognition Face Liveness API のプログラミング ... 715
ステップ 1: CreateFaceLivenessSession ... 716
ステップ 2: StartFaceLivenessSession ... 717
ステップ 3: GetFaceLivenessSessionResults ... 717
ステップ 4: 結果への対応 ... 719
Face Liveness API を呼び出す ... 719
アプリケーションの設定とカスタマイズ ... 725
アプリケーションの設定 ... 725
アプリケーションをカスタマイズする ... 726
Face Liveness の責任共有モデル ... 726
Face Liveness の更新に関するガイドライン ... 730
バージョニングとタイムフレーム ... 730
バージョンリリースと互換性マトリックス ... 731
新しいリリースの通知 ... 732
Face Liveness に関するよくある質問 ... 732
一括分析 ... 736
イメージの一括処理 ... 736
一括分析ジョブ (CLI) を作成するには ... 736

一括分析出力マニフェスト 738
コンテンツタイプ 739
予測の検証とアダプターのトレーニング 739
チュートリアル 740
Amazon RDS および DynamoDB で Amazon Rekognition データを保存する 740
前提条件 741
Amazon S3 バケット内のイメージのラベルを取得する 741
Amazon DynamoDB テーブルを作成する 743
DynamoDB へのデータのアップロード 744
Amazon RDS で MySQL データベースを作成する 746
Amazon RDS の MySQL テーブルへのデータのアップロード 747
Amazon Rekognition と Lambda を使用して Amazon S3 バケットのアセットにタグを付ける 749
前提条件 751
IAM Lambda ロールを設定する 751
プロジェクトの作成 752
コードを書き込む 755
プロジェクトをパッケージ化する 765
Lambda 関数をデプロイします。 766
Lambda メソッドをテストする 767
AWS ビデオアナライザーアプリケーションの作成 768
前提条件 769
手順 769
Amazon Rekognition Lambda 関数の作成 770
前提条件 772
SNS トピックを作成する 772
Lambda 関数を作成する 772
Lambda 関数を設定 773
IAM Lambda ロールを設定する 774
AWS Toolkit for Eclipse Lambda プロジェクトを作成 775
Lambda 関数をテストする 779
本人確認における Amazon Rekognition の使用 780
前提条件 781
コレクションの作成 781
新規ユーザー登録 782
既存ユーザーのログイン 791

マーケティングアプリケーション用に Amazon Rekognition (REK) を使用してラベルを検出する 793
前提条件 794
DynamoDB テーブルの作成とデータ入力 795
ログと通知の機能を備えた Lambda 関数の作成 797
アクセス許可と Lambda 関数の設定 799
画像をメタデータとともに Amazon S3 にアップロードする 801
クライアントからの CloudWatch Logs のポーリング 802
Lambda と Python を使用したイメージ内のラベルの検出 804
Lambda 関数の作成 (コンソール) 804
(オプション) レイヤーの作成 (コンソール) 806
Python コードの追加 (コンソール) 807
Python コードを追加するには (コンソール) 809
コードの例 813
基本 817
Hello Amazon Rekognition 818
アクション 821
シナリオ 982
コレクションを構築し、その中に顔を検索します。 983
サーバーレスアプリケーションを作成して写真の管理 995
イメージ内の PPE を検出する 999
イメージ内の要素の検出と表示 1000
イメージ内の顔を検出します 1016
ビデオ内の情報を検出する 1017
イメージ内のオブジェクトを検出する 1056
動画内の人物や物体を検出する 1060
EXIF およびその他のイメージ情報を保存します 1061
API リファレンス 1063
セキュリティ 1064
Identity and Access Management 1064
対象者 1065
アイデンティティを使用した認証 1065
ポリシーを使用したアクセスの管理 1068
Amazon Rekognition が IAM と機能する仕組み 1071
AWS マネージドポリシー 1077
ポリシーに関するベストプラクティス 1085

コンソールを使用する 1086
アイデンティティベースのポリシーの使用例 1086
リソースベースのポリシーの例 1089
トラブルシューティング 1090
データ保護 1093
データ暗号化 1094
インターネットトラフィックのプライバシー 1096
Amazon VPC エンドポイントで Amazon Rekognition を使用する 1097
Amazon Rekognition 用の Amazon VPC エンドポイントの作成 1097
Amazon Rekognition の VPC エンドポイントポリシーの作成 1098
コンプライアンス検証 1099
耐障害性 1100
設定と脆弱性の分析 1101
サービス間の混乱した代理の防止 1101
インフラストラクチャセキュリティ 1103
モニタリング 1105
Amazon CloudWatch による Rekognition のモニタリング 1105
Rekognition での CloudWatch メトリクスの使用です。 1106
Rekognition メトリクスにアクセスします。 1107
アラームの作成 1108
Rekognition の CloudWatch メトリクスです。 1110
を使用した Amazon Rekognition API コールのログ記録 AWS CloudTrail 1115
CloudTrail 内の Amazon Rekognition に関する情報 1115
Amazon Rekognition ログファイルエントリを理解する 1116
ガイドラインとクォータ 1119
サポートされるリージョン 1119
クォータの設定 1119
Amazon Rekognition イメージ 1119
Amazon Rekognition Image 一括分析 1119
Amazon Rekognition Video 保存ビデオ 1120
Amazon Rekognition Video ストリーミング ビデオ 1121
デフォルトのクォータ 1121
TPS クォータの変更を計算する 1122
TPS クォータのベストプラクティス 1122
TPS クォータを変更するケースを作成する 1122
ドキュメント履歴 1125

AWS 用語集 1139
..... mcxl

Amazon Rekognition とは

Amazon Rekognition は、クラウドベースの画像およびビデオ分析サービスであり、アプリケーションに高度なコンピュータビジョン機能を簡単に追加できます。このサービスは、実証済みの深層学習テクノロジを活用しており、機械学習の専門知識がなくても利用できます。Amazon Rekognition には、Amazon S3 に保存されているイメージファイルやビデオファイルを即座に分析できる、シンプルで使いやすい API が含まれています。

Rekognition の API を使用することで、物体検出、テキスト検出、安全でないコンテンツの検出、画像/ビデオ分析、顔比較などの機能をアプリケーションに追加できます。Amazon Rekognition の顔認識 API を使用すると、ユーザー認証、カタログ作成、人数のカウント、公共の安全など、さまざまなユースケースで顔を検出、分析、比較できます。

このサービスは、Amazon のコンピュータビジョン科学者が開発したものと同じ、実証済みの非常にスケーラブルな深層学習テクノロジに基づいており、日々何十億もの画像やビデオを分析できます。Rekognition は新しいデータから継続的に学習しており、サービスに新しいラベルや機能が頻繁に追加されています。

詳細については、「Amazon Rekognition のよくある質問」を参照してください。

主な機能

画像分析:

- 物体、シーン、概念の検出 - 画像内の物体、シーン、概念、有名人を検出し、分類します。
- テキスト検出 - さまざまな言語の画像内の印刷テキストや手書きテキストを検出し、認識します。
- 安全でないコンテンツの検出 - 画像内の露骨な描写、不適切な内容、暴力的な表現を検出し、フィルタリングします。詳細なカテゴリの安全でないコンテンツラベルを検出します。
- 有名人の認識 - 政治家、アスリート、俳優、ミュージシャンなど、さまざまなカテゴリの数万人の有名人を画像内で認識します。
- 顔分析 - 顔および顔属性 (性別、年齢、感情など) を検出、分析、比較します。ユースケースには、ユーザー認証、カタログ化、人数カウント、公共安全維持などがあります。
- カスタムラベル - ユースケースに固有の物体 (ロゴ、製品、キャラクターなど) を検出するためのカスタム分類モデルを構築します。

- 画像プロパティ - 画質、色、シャープネス、コントラストなどの画像プロパティを分析します。

ビデオ分析:

- 物体、シーン、概念の検出 - ビデオ内の物体、シーン、概念、有名人を検出し、分類します。
- テキスト検出 - さまざまな言語のビデオ内の印刷テキストや手書きテキストを検出し、認識します。
- 人物の動線の検出 - ビデオフレーム間を移動する特定された人物を追跡します。
- 顔分析 - ストリーミングまたは保存済みビデオ内の顔を検出、分析、比較します。
- 有名人の認識 - 政治家、アスリート、俳優、ミュージシャンなど、さまざまなカテゴリの数万人の有名人を保存済みビデオ内で認識します。
- 安全でないコンテンツの検出 - ビデオ内の露骨な描写、不適切な内容、暴力的な表現を検出します。
- ビデオのセグメンテーション - ブラックフレームやエンドクレジットなど、ビデオの有用なセグメントを自動的に識別します。
- 顔のライブネスチェック - 顔認証時にライブユーザーが存在するかどうかを検出します。

ユースケース

検索可能なメディアライブラリ - Rekognition は画像やビデオ内のラベル、物体、概念、シーンを検出します。この視覚的コンテンツ分析に基づいて、これらのラベルを検索可能にすることができます。検索可能な画像およびビデオライブラリの構築に役立ちます。

顔に基づいたユーザーの本人確認 - 画像内の顔と参照用の顔画像を比較してユーザーの本人確認を行います。アプリケーションでの本人確認に役立ちます。

顔のライブネス検出 - Rekognition の顔のライブネスチェックは、デベロッパーが顔に基づいた本人確認時の不正を抑止できるように設計された、フルマネージドの機械学習 (ML) 機能です。この機能は、ユーザーがカメラの前に物理的に存在していて、ユーザーの顔になりすましている悪質な行為者でないことを確認するのに役立ちます。Rekognition Face Liveness を使用すると、印刷された写真、デジタル写真/ビデオ、3D マスクなど、カメラに提示されたなりすまし攻撃を検出するのに役立ちます。また、事前に録画されたビデオやビデオキャプチャサブシステムに直接挿入されたディープフェイクビデオなど、カメラをバイパスするなりすまし攻撃の検出にも役立ちます。

顔検索 - Rekognition では、コンテナ (顔コレクションとも呼ばれる) に保存されているものと一致する顔がないか、画像、保存済みビデオ、ストリーミングビデオを検索できます。顔コレクションはお

客様が所有および管理する顔のインデックスです。顔に基づいて人物を検索するには、顔をインデックス化してから、顔を検索する必要があります。

安全でないコンテンツの検出 - 画像やビデオ内の露骨な描写、不適切な内容、暴力的な表現を検出し、フィルタリングします。ラベルを使用して、ビジネスニーズに基づいた詳細なフィルタリングが可能です。Content Moderation API は、検出されたラベル (物体や概念) の階層リストと信頼スコアも返します。これらのオブジェクト/ラベルによって、安全でないコンテンツの特定のカテゴリがわかるため、大量のユーザー生成コンテンツ (UGC) を細かくフィルター処理して管理できます。Content Moderation API の出力は、アダプターを使用してカスタマイズできます。これにより、トレーニングデータとして提供した画像と同様のものに対して、処理パフォーマンスが向上します。

個人用保護具の検出 - さまざまな業界の安全コンプライアンスの遵守をモニタリングするために、画像内の個人用保護具を検出します。不適切な装備を検出して安全でない状況に自動的にフラグ付けして、これらの状況に関するアラートを受け取ることができます。これにより、コンプライアンスとトレーニングを改善できます。

有名人の認識 - 政治家、アスリート、俳優、ミュージシャンなど、さまざまなカテゴリの有名人を画像やビデオ内で認識します。有名人の出現は名前を指定しなくても特定できます。

テキスト検出 - 視覚的検索やメタデータ抽出のために画像内のテキストを検出し、抽出します。さまざまなフォントやスタイルに対応しています。標識やバナーのテキストを処理するために向きを検出します。

カスタムラベル - ロゴ検出など、ビジネスユースケースに固有のカスタム物体、概念、シーンを識別します。ニッチな物体や独自の物体を処理するようにカスタム分類モデルをトレーニングできます。これにより、汎用の分類モデルと比べて主要な物体に対する精度が向上します。詳細については、[Amazon Rekognition カスタムラベル デベロッパー ガイド] の [Amazon Rekognition カスタムラベルとは] を参照してください。

利点

強力な画像およびビデオ分析をアプリケーションに統合 - 専門知識がなくても、正確な画像およびビデオ分析をアプリケーションに追加できます。Amazon Rekognition API を使用すると、機械学習の知識がなくても深層学習による分析が可能になります。ウェブ、モバイル、デバイスアプリケーションにコンピュータビジョン機能をすばやく組み込むことができます。

深層学習ベースの画像およびビデオ分析 - 深層学習を使用して画像やビデオを高精度で分析します。Amazon Rekognition はラベル、物体、シーン、顔、有名人を検出できます。検出した結果は、特定のラベルを含めたり除外したりするようにフィルタリングできます。

スケーラブルな画像分析 - 数百万もの画像を分析して、大規模な視覚的データセットを整理します。増大し続ける画像ライブラリとトラフィックを処理するようにスケールします。キャパシティーを計画する必要がなく、使用分のみお支払いいただきます。

画像プロパティに基づく分析とフィルタリング - 画質、色、視覚的コンテンツなどのプロパティに基づいて画像の分析とフィルタリングを行い、画像のシャープネス、明るさ、コントラストを検出します。

他の AWS サービスとの統合 - Amazon Rekognition は S3 および Lambda とすぐに統合できます。Lambda から Amazon Rekognition の API を呼び出して、Amazon S3 内の画像を移動させずに処理できます。Rekognition には、IAM AWS を使用したスケーラビリティとセキュリティが組み込まれています。

低コスト - 従量制料金であり、最低利用料金や契約期間のコミットメントはありません。まずは無料利用枠から開始できます。段階的な料金設定のため、使用量が増えるほどお得になります。自社ソリューションと比べ、より高い費用対効果を得られます。

シンプルなカスタマイズ - アダプターを使用してユースケースに合わせて精度をカスタマイズできます。サンプル画像を提供してアダプターをトレーニングします。特定のドメインに対する物体とラベルの検出が向上します。機械学習の専門知識がなくても、簡単に分析をカスタマイズできます。

詳細については、「Amazon Rekognition のよくある質問」を参照してください。

Amazon Rekognition と HIPAA の適格性

これは HIPAA 対象サービスです。1996 年の AWS米国の医療保険の相互運用性と説明責任に関する法律 (HIPAA)、および AWS のサービスを使用して保護対象医療情報 (PHI) を処理、保存、および送信する方法の詳細については、「HIPAA 概要」を参照してください。

Amazon Rekognition ユーザー を初めてお使いになる方向けの情報

Amazon Rekognition を初めて使用する場合は、以下のセクションを順に読むことをお勧めします。

1. Amazon Rekognition の仕組み – このセクションでは、エンドツーエンドエクスペリエンスを創出するための Amazon Rekognition のさまざまなコンポーネントを紹介します。
2. 「Amazon Rekognition の開始方法」 — このセクションでは、アカウントをセットアップして、選択した言語に対応した SDK をインストールし、Amazon Rekognition API をテストします。Amazon Rekognition がサポートしているプログラミング言語の一覧については、「AWS SDK での Rekognition の使用」を参照してください。

3. イメージの操作 – このセクションでは、Amazon S3 バケットに保存されたイメージおよびローカルファイルシステムからロードされたイメージで Amazon Rekognition を使用する方法について説明します。
4. 保存済み動画分析での作業 – このセクションでは、Amazon S3 バケットに保存されているビデオに Amazon Rekognition を使用する方法について説明します。
5. ストリーミングビデオイベントの操作 – このセクションでは、ストリーミングビデオで Amazon Rekognition を使用する方法について説明します。

Amazon Rekognition の仕組み

Amazon Rekognition には、視覚的分析用に 2 つの API セットが用意されています。

- 画像分析用の Amazon Rekognition Image
- ビデオ分析用の Amazon Rekognition Video

画像分析

Amazon Rekognition Image を使用すると、アプリケーションで以下のことができます。

- 画像内の物体、シーン、概念を検出する
- 有名人を認識する
- さまざまな言語のテキストを検出する
- 画像内の露骨な描写、不適切な内容、暴力的な表現を検出する
- 顔および顔属性 (年齢や感情など) を検出、分析、比較する
- 個人用保護具 (PPE) の有無を検出する

ユースケースとしては、写真アプリケーションの機能強化、画像のカタログ化、コンテンツのモデレーションなどがあります。

ビデオ分析

Amazon Rekognition Video を使用すると、アプリケーションで以下のことができます。

- ビデオフレーム間で人物や物体を追跡する
- 物体を認識する
- 有名人を認識する
- 保存済みビデオとストリーミングビデオ内で対象となる人物を検索する
- 年齢や感情などの顔属性に基づいて顔を分析する
- 画像内の露骨な描写、不適切な内容、暴力的な表現を検出する
- タイムスタンプとセグメント別に分析結果の集計、ソートする
- ストリーミングビデオ内の人、ペット、荷物を検出する

ユースケースには、ビデオの分析、ビデオのカタログ化、不適切なコンテンツのフィルタリングなどがあります。

主な特徴

- 強力な深層学習分析
- 物体、シーン、顔、テキストの高精度な検出
- アプリケーションへの統合が容易な API
- データに合わせてチューニング可能なモデル
- メディアライブラリのスケーラブルな分析

Amazon Rekognition では、カスタムアダプターをトレーニングすることで、特定の深層学習モデルの精度を向上させることができます。例えば、Amazon Rekognition のカスタムモデレーションを使用すると、お客様の画像でカスタムアダプターをトレーニングすることで、Amazon Rekognition の基本画像分析モデルを必要に応じて調整できます。詳細については、「カスタムモデレーションによる精度の向上」を参照してください。

以下のセクションでは、Amazon Rekognition で可能な分析のタイプと、Amazon Rekognition Image および Amazon Rekognition Video のオペレーションの概要について説明します。また、非ストレージ型オペレーションとストレージ型オペレーションの違いについても説明します。

Amazon Rekognition APIs「ステップ 3: AWS CLI と AWS SDK API の使用を開始する」を参照してください。ここでは、 AWS コンソールで Rekognition を試す方法について説明します。

トピック

- Rekognition の分析タイプについて
- Rekognition の画像およびビデオオペレーションについて
- 非ストレージ型およびストレージ型の API オペレーションについて
- モデルのバージョニングについて

Rekognition の分析タイプについて

Amazon Rekognition Image API と Amazon Rekognition Video API で実行できる分析のタイプは以下のとおりです。API の詳細については、「Rekognition の画像およびビデオオペレーションについて」を参照してください。

以下の表は、使用するメディアのタイプおよびユースケースに応じて使用すべきオペレーションを一覧にまとめたものです。

ユースケース	メディアタイプ	オペレーション
コンテンツのモデレーション	イメージ	DetectModerationLabels、StartMediaAnalysisJob、GetMediaAnalysisJob、ListMediaAnalysisJobs
	保存済みビデオ	StartContentModeration、GetContentModeration
本人確認	イメージ	CreateCollection、CreateUser、IndexFaces、AssociateFaces、SearchFacesByImage、SearchUsersByImage
	保存済みビデオ	CreateCollection、IndexFaces、StartFaceSearch、GetFaceSearch
	ストリーミングビデオ (顔のライブネスの検出)	CreateFaceLivenessSession、StartFaceLivenessSession、GetFaceLivenessSessionResults
顔分析	イメージ	DetectFaces、CompareFaces
	保存済みビデオ	StartFaceDetection、GetFaceDetection
	ストリーミングビデオ	CreateStreamProcessor、StartStreamProcessor
オブジェクトとアクティビティの認識	イメージ	DetectLabels

ユースケース	メディアタイプ	オペレーション
	保存済みビデオ	StartLabelDetection、GetLabelDetection
コネクテッドホーム	ストリーミングビデオ	StartStreamProcessor
メディア分析	保存済みビデオ	StartSegmentDetection、GetSegmentDetection
職場の安全	イメージ	DetectProtectiveEquipment
テキストの検出	イメージ	DetectText
	動画	StartTextDetection、GetTextDetection
人物の動線の検出	動画	StartPersonTracking、GetPersonTracking
有名人の認識	イメージ	RecognizeCelebrities
	動画	StartCelebrityRecognition、GetCelebrityRecognition
カスタムラベル検出	イメージ	DetectCustomLabels
	モデルトレーニング	「Custom Labels developer guide」を参照のこと

ラベル

ラベルとは、オブジェクト (花、木、テーブルなど)、イベント (結婚式、卒業式、誕生日会など)、概念 (風景、夜、自然など)、アクティビティ (走る、バスケットボールをするなど) のうちのいずれかを指します。Amazon Rekognition では、イメージやビデオ内のラベルを検出できます。詳細については、「オブジェクトおよび概念の検出」を参照してください。

Rekognition は、イメージや保存済みビデオから、大量のラベルリストを検出できます。また、ストリーミングビデオから少数のラベルを検出することもできます。

ラベルを検出するには、ユースケースに応じて次のオペレーションを使用します。

- イメージ内のラベルを検出するには、DetectLabels を使用します。主要な色や画質といったイメージの特性を識別できます。実行するには、DetectLabels と共に IMAGE_PROPERTIES を入力パラメータとして使用します。
- 保存済みビデオのラベルを検出するには、StartLabelDetection を使用します。保存済みビデオでは、主要な色や画質の検出はサポートされていません。
- ストリーミングビデオのラベルを検出するには、CreateStreamProcessor を使用します。ストリーミングビデオでは、主要な色や画質の検出はサポートされていません。

イメージと保存済みビデオの両方で返すラベルの種類は、包含フィルターと除外フィルターのオプションを使用することで指定できます。

カスタムラベル

Amazon Rekognition カスタムラベルは、機械学習モデルをトレーニングすることで、ビジネスニーズに固有のイメージ内のオブジェクトやシーンを識別できます。たとえば、モデルをトレーニングして、ロゴを検出したり、組立ラインでエンジニアリング機械部品を検出したりできます。

> **Note**
>
> Amazon Rekognition カスタムラベルの詳細については、[Amazon Rekognition カスタムラベル開発者ガイド] を参照してください。

Amazon Rekognition には、機械学習モデルの作成、トレーニング、評価、実行に使用するコンソールが用意されています。詳細については、[Amazon Rekognition カスタムラベル デベロッパーガイド] の [Amazon Rekognition カスタムラベルを使い始める] を参照してください。Amazon Rekognition カスタムラベル API を使用して、モデルをトレーニングおよび実行することもできます。詳細については、[Amazon Rekognition カスタムラベル デベロッパー ガイド] の [Amazon Rekognition カスタムラベル SDK を使い始める] を参照してください。

トレーニングされたモデルを使用してイメージを分析するには、DetectCustomLabels を使用します。

Face Liveness による検出

Amazon Rekognition Face Liveness を使用すると、顔での本人確認をパスしたユーザーが実際にカメラの前に存在しており、そのユーザーをかたった偽者ではないことを、確認できます。Face Liveness は、カメラにしかけられたなりすまし攻撃やカメラを回避する攻撃を検出する機能です。Face Liveness のチェックはユーザーが短い自撮り動画を撮影することで完了し、それにより Liveness のスコアが返されます。Face Liveness は確率的計算で算出され、チェック後に信頼スコア (0 ~ 100) が返されます。スコアが高いほど、チェックした人物が実在している確実性が高くなります。

Face Liveness の詳細については、「顔のライブネスの検出」を参照してください。

顔の検出と分析

Amazon Rekognition では、イメージおよび保存済みビデオ内の顔を検出できます。Amazon Rekognition を使用すると、以下に関する情報を取得できます。

- イメージまたはビデオで顔が検出された場所
- 顔のランドマーク (目の位置など)
- イメージ内で顔を覆っている物の有無
- 確認できる感情 (うれしい、悲しいなど)
- イメージ内にいる人の視線の方向

性別や年齢といった人口統計的情報を読み取ることもできます。また、イメージ内の顔を、別のイメージで検出された顔と比較することもできます。顔に関する情報を保存して、後で取得することもできます。詳細については、「顔の検出と分析」を参照してください。

イメージから顔を検出するときは DetectFaces を使用します。保存済みビデオから顔を検出するときは StartFaceDetection を使用します。

顔検索

Amazon Rekognition では顔を検索することができます。顔情報は、コレクションと呼ばれるコンテナ内にインデックス付けされます。これにより、コレクション内の顔情報を、イメージ、保存済みビデオ、ストリーミングビデオで検出された顔と照合することができます。詳細については、「コレクション内での顔の検索」。

イメージから既知の顔を検索するときは DetectFaces を使用します。保存済みビデオで顔を検索するときは StartFaceDetection を使用します。ストリーミングビデオで既知の顔を検索するときは CreateStreamProcessor を使用します。

人物の動線

Amazon Rekognition では、保存されたビデオで検出された人物のパスを追跡できます。Amazon Rekognition Video は、ビデオで検出された人物のパス追跡、顔の詳細、およびフレーム内の位置情報を提供します。詳細については、「人物の動線の検出」を参照してください。

保存済みビデオから人物を検出するときは StartPersonTracking を使用します。

個人用保護具

Amazon Rekognition では、イメージ内で検出された人物が着用している個人用保護具 (PPE) を検出できます。Amazon Rekognition は、フェイスカバー、ハンドカバー、ヘッドカバーを検出します。Amazon Rekognition は、PPE のアイテムが適切な体の部位をカバーしているかどうかを予測します。検出された人物や PPE アイテムの境界ボックスを取得することもできます。詳細については、「個人用保護具を検出する。」を参照してください。

イメージから PPE を検出するときは DetectProtectiveEquipment を使用します。

有名人

Amazon Rekognition では、イメージや保存されたビデオに写っている多数の有名人を認識できます。イメージで有名人の顔が写っている場所、顔の特徴、表情に関する情報を取得できます。保存済みビデオ全体で表示される有名人の追跡情報を取得できます。また、認識されている有名人について、感情表現や性別の提示などの詳細情報を得ることもできます。詳細については、「有名人の認識」を参照してください。

イメージから有名人を識別するときは RecognizeCelebrities を使用します。保存済みビデオから有名人を識別するときは StartCelebrityRecognition を使用します。

テキストの検出

Amazon Rekognition Text in Image は、イメージ内のテキストを検出し、機械判読可能なテキストに変換することができます。詳細については、「テキストの検出」を参照してください。

イメージからテキストを検出するときは DetectText を使用します。

不適切または不快なコンテンツ

Amazon Rekognition では、イメージや保存されたビデオにアダルトコンテンツや暴力的なコンテンツが含まれていないかどうかを分析できます。詳細については、「コンテンツのモデレーション」を参照してください。

イメージから PPE を検出するときは DetectModerationLabels を使用します。保存済みビデオから適切でない内容を検出するときは StartContentModeration を使用します。

カスタマイズ

Rekognition で利用できる特定のイメージ分析 API を使用すると、ユーザー独自のデータでトレーニングされたカスタムアダプターを作成することにより、深層学習モデルの精度を高めることができます。アダプターとは、Rekognition の事前トレーニング済み深層学習モデルにプラグインされるコンポーネントで、ユーザーのイメージに基づく専門知識によりモデルの精度を高めます。アダプターは、サンプルイメージを提供しこれに注釈をつけることにより、ニーズに合うようにトレーニングします。

アダプターを作成すると、AdapterID が提供されます。この AdapterID をオペレーションに提供することで、作成したアダプターを使用するように指定できます。例えば、AdapterId を、同期的にイメージを分析する DetectModerationLabels API に提供するとします。リクエストの一部で AdapterID を指定すると、Rekognition が自動的にこれを使用し、イメージの予測精度を高めます。こうして、Rekognition の機能を活用しつつニーズに合わせてこれをカスタマイズすることができます。

StartMediaAnalysisJob API を使用して、イメージの予測を一括で取得することもできます。詳細については「Bulk analysis」を参照してください。

Rekognition のコンソールにイメージをアップロードし分析を実行すると、Rekognition のオペレーションの精度を評価できます。Rekognition は選択した機能を使ってイメージに注釈を付けます。その後、予測を検証し、検証済みの予測を使って、アダプターを作成することで恩恵を受けるラベルはどれかを判断できます。

現在ご利用いただけるのは、DetectModerationLabels の付いたアダプターです。アダプターの作成および使用に関する詳細は、「カスタムモデレーションによる精度の向上」を参照してください。

一括分析

Rekognition の一括分析を使用すると、StartMediaAnalysisJob オペレーションと併せてマニフェストファイルを使用することにより、大量のイメージのコレクションを非同期的に処理できます。詳細については「Bulk analysis」を参照してください。

Rekognition の画像およびビデオオペレーションについて

Amazon Rekognition には、画像分析用とビデオ分析用に 2 つの主要な API セットが用意されています。

- Amazon Rekognition Image: この API は画像の分析用に設計されています。
- Amazon Rekognition Video: この API は保存済みビデオとストリーミングビデオの両方の分析に重点を置いています。

どちらの API でも、顔や物体などのさまざまな要素を検出できます。サポートされている比較と検出の両タイプを包括的に理解するには、Rekognition の分析タイプについて に関するセクションを参照してください。

Amazon Rekognition Image のオペレーション

Amazon Rekognition Image のオペレーションは同期的に行われます。入力およびレスポンスは JSON 形式です。Amazon Rekognition Image オペレーションでは、.jpg 形式または .png 形式の入力イメージを分析します。Amazon Rekognition Image オペレーションに渡されたイメージは、Amazon S3 バケットに保存できます。AWS CLI を使用しない場合は、Base 64 エンコードされた画像のバイトを Amazon Rekognition オペレーションに直接渡すこともできます。詳細については、「画像の操作」を参照してください。

Amazon Rekognition Video オペレーション

Amazon Rekognition Video API により、Amazon S3 バケットに保存されたビデオまたは Amazon Kinesis Video Streams を介してストリーミングされるビデオの分析が容易になります。

保存済みビデオのオペレーションについては、以下の点に注意してください。

- オペレーションは非同期的に実行されます。
- 分析は「Start」オペレーション (例えば、保存済みビデオ内の顔検出の場合は StartFaceDetection) で開始する必要があります。

- 分析の完了ステータスは Amazon SNS トピックに発行されます。
- 分析結果を取得するには、対応する「Get」オペレーション (例えば、GetFaceDetection) を使用します。
- 詳細については、「保存済み動画分析での作業」を参照してください。

ストリーミングビデオ分析の場合:

- Rekognition Video コレクションでの顔検索やラベル (物体や概念) 検出などの機能があります。
- ラベルの分析結果は Amazon SNS と Amazon S3 の通知として送信されます。
- 顔検索の結果は Kinesis データストリームに出力されます。
- ストリーミングビデオ分析の管理は、Amazon Rekognition Video ストリームプロセッサを介して行われます (例えば、CreateStreamProcessor を使用してプロセッサを作成します)。
- 詳細については、「ストリーミングビデオイベントの操作」を参照してください。

各ビデオ分析オペレーションでは、分析中のビデオに関するメタデータのほか、ジョブ ID とジョブタグが返されます。ビデオに対するラベルの検出やコンテンツのモデレーションなどのオペレーションでは、タイムスタンプまたはラベル名でソートしたり、タイムスタンプまたはセグメント別に結果を集計したりできます。

非ストレージ型およびストレージ型のオペレーション

Amazon Rekognition オペレーションは次の種類に分類されます。

- [非ストレージ型 API オペレーション] - このオペレーションでは、Amazon Rekognition によって情報は保存されません。入力イメージやビデオを指定すると、オペレーションで分析が実行されてその結果が返されますが、Amazon Rekognition によって情報は保存されません。詳細については、「非ストレージ型オペレーション」を参照してください。
- [ストレージ型 API オペレーション] - Amazon Rekognition サーバーは、検出された顔情報をコレクションと呼ばれるコンテナに保存できます。Amazon Rekognition には、保存された顔情報から一致する顔を検索するための追加の API オペレーションが用意されています。詳細については、「ストレージ型 API オペレーション」を参照してください。

AWS SDK または HTTP による Amazon Rekognition API オペレーションの呼び出し

AWS SDK を使用するか、直接 HTTP を使用して Amazon Rekognition API オペレーションを呼び出すことができます。正当な理由がない限り、常に AWS SDK を使用してください。このセクションの Java の例では、AWS SDK を使用しています。Java プロジェクトファイルは提供されていませんが、Java を使用する AWS アプリケーションは AWS Toolkit for Eclipse で開発できます。

このセクションの .NET 例では AWS SDK for .NET を使用します。AWS Toolkit for Visual Studio では、.NET を使用して AWS アプリケーションを開発できます。便利なテンプレートと AWS Explorer を使用して、アプリケーションをデプロイし、サービスを管理できます。

このガイドの「API Reference」では、HTTP を使用した Amazon Rekognition オペレーションの呼び出しについて説明します。Java に関する参考情報については、「AWS SDK for Java」を参照してください。

使用できる Amazon Rekognition サービスエンドポイントについては、[AWS のリージョンとエンドポイント] を参照してください。

HTTP で Amazon Rekognition を呼び出す場合は、POST HTTP オペレーションを使用します。

非ストレージ型およびストレージ型の API オペレーションについて

Amazon Rekognition には、2 つのタイプの API オペレーションがあります。1 つは Amazon Rekognition によって情報が保存されない非ストレージ型オペレーションで、もう 1 つは Amazon Rekognition によって特定の顔情報が保存されるストレージ型オペレーションです。

非ストレージ型オペレーション

Amazon Rekognition には、以下のイメージ用の非ストレージ型 API オペレーションがあります。

- DetectLabels
- DetectFaces
- CompareFaces
- DetectModerationLabels
- DetectProtectiveEquipment

- RecognizeCelebrities
- DetectText
- GetCelebrityInfo

Amazon Rekognition には、以下のビデオ用の非ストレージ型 API オペレーションがあります。

- StartLabelDetection
- StartFaceDetection
- StartPersonTracking
- StartCelebrityRecognition
- StartContentModeration

これらは、[非ストレージ型] の API オペレーションと呼ばれます。この型のオペレーションを呼び出すと、Amazon Rekognition で入力イメージに関して検出された情報は保持されません。他のすべての Amazon Rekognition API オペレーションと同様に、非ストレージ型の API オペレーションでは入力イメージのバイトが保持されません。

以下のシナリオ例では、非ストレージ型の API オペレーションをアプリケーションに統合できることを示します。各シナリオでは、イメージのローカルリポジトリがあることを想定しています。

Example 1: ローカルリポジトリ内で、特定のラベルを含むイメージを検索するアプリケーション

最初に、次に示すように Amazon Rekognition `DetectLabels` オペレーションを使用してリポジトリ内のイメージごとにラベル (オブジェクトと概念) を検出し、クライアント側のインデックスを構築します。

```
Label        ImageID

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

次に、このインデックスを検索することで、ローカルリポジトリ内で特定のラベルを含むイメージを見つけることができます。たとえば、樹木を含むイメージを表示します。

Amazon Rekognition で検出するラベルごとに、信頼値が関連付けられます。信頼値は、入力イメージにそのラベルが含まれている確率 (信頼度) を示します。この信頼値を使用して、検出の信頼度に関するアプリケーションの要件に応じて、ラベルに対するクライアント側の追加のフィルタリングを実行できます。たとえば、正確なラベルを必要とする場合は、信頼度の高い (95% 以上) ラベルに絞り込んで選択します。信頼値がそれほど高くなくてもよい場合は、信頼値が低い (50% 程度) ラベルに絞り込んで選択してください。

Example 2: 強化した顔イメージを表示するアプリケーション

まず、Amazon Rekognition `DetectFaces` オペレーションを使用してローカルリポジトリ内のイメージごとに顔を検出し、クライアント側のインデックスを構築できます。検出された顔ごとに、境界ボックス、顔の特徴 (口や耳の位置)、顔の属性 (性別など) を含むメタデータが返されます。次に示すように、このメタデータはクライアント側のローカルインデックスに保存できます。

```
ImageID     FaceID     FaceMetaData

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

このインデックスでは、プライマリキーは `ImageID` と `FaceID` の組み合わせです。

次に、このインデックス内の情報を使用して、アプリケーションでローカルリポジトリのイメージを表示するときのイメージを強化できます。たとえば、境界ボックスを顔の周囲に配置したり、顔の特徴を強調したりできます。

ストレージ型 API オペレーション

Amazon Rekognition Image は、イメージ内の顔を検出し、Amazon Rekognition コレクションで検出された顔の特徴の情報を保持するための IndexFaces オペレーションをサポートしています。サービスによって情報がサーバーに保持されるため、これは ストレージベース の API オペレーションの例です。

Amazon Rekognition Image に用意されているストレージ型 API オペレーションは次のとおりです。

- IndexFaces
- ListFaces

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

Amazon Rekognition Video に用意されているストレージ型 API オペレーションは次のとおりです。

- StartFaceSearch
- CreateStreamProcessor

顔の情報を保存するには、まず、アカウントの AWS リージョンのいずれかで顔コレクションを作成する必要があります。この顔コレクションは、IndexFaces オペレーションを呼び出すときに指定します。顔コレクションを作成して、すべての顔の特徴情報を保存すると、コレクション内で一致する顔を検索できます。たとえば、イメージ内で最も大きい顔を検出し、一致する顔がないかコレクションを検索するには、searchFacesByImage. を呼び出します。

IndexFaces によってコレクション内に保存された顔情報は、Amazon Rekognition Video オペレーションで利用することができます。例えば、StartFaceSearch を呼び出すと、既存コレクション内の顔と一致する顔の人物がいないか、ビデオを検索できます。

コレクションの作成および管理については、「コレクション内での顔の検索」を参照してください。

> **Note**
>
> コレクションには、顔の数学的表現である顔ベクトルが保存されます。コレクションには、顔の画像は保存されません。

Example 1: ビルへの立ち入りを認証するアプリケーション

まず、IndexFaces オペレーションを使用して、バッジのスキャンイメージを保存する顔コレクションを作成します。これにより、顔が抽出され、検索可能なイメージベクトルとして保管されます。次に、従業員が建物に入る際に従業員の顔のイメージが撮影されて SearchFacesByImage オ

ペレーションに送信されます。十分に高い類似度スコア (99% など) で顔が一致すると、この社員を認証できます。

モデルのバージョニングについて

Amazon Rekognition では、深層学習モデルを使用して顔検出を実行し、コレクション内の顔を検索します。顧客フィードバックに基づいてモデルの精度の改善を継続的に行い、ディープラーニング研究を進めています。これらの改善点はモデル更新に反映されて提供されます。例えば、バージョン 1.0 のモデルの場合は、IndexFaces を使用すると、イメージ内にある 15 の最大の顔にインデックスを付けることができます。これ以降のバージョンのモデルでは、IndexFaces を使用し、イメージ内の 100 の大きい顔にインデックスを付けることができます。

新しいコレクションを作成すると、最新バージョンのモデルにこのコレクションが関連付けられます。精度を向上させるには、モデルを定期的に更新してください。

新しいバージョンのモデルがリリースされると、以下の動作が発生します。

- 作成する新しいコレクションが最新モデルに関連付けられます。IndexFaces を使用して新しいコレクションに追加する顔は、この最新モデルを使用して検出されます。
- 既存のコレクションでは、このコレクションを作成したバージョンのモデルを引き続き使用します。これらのコレクションに保存されている顔ベクトルは、最新バージョンのモデルに自動的に更新されません。
- 既存のコレクションに追加された新しい顔は、そのコレクションに既に関連付けられているモデルを使用して検出されます。

各バージョンのモデルは互いに互換性はありません。具体的にいうと、異なるバージョンのモデルを使用した複数のコレクションのイメージにインデックスを付けた場合は、検出された顔が同じであっても顔の識別子は異なります。同じモデルに関連付けられた複数のコレクションのイメージにインデックスを付けた場合、顔の識別子は同じになります。

モデルのアップデートを考慮せずにコレクション管理を行うと、アプリケーションで互換性問題が発生する可能性があります。コレクションオペレーション (CreateCollection など) のレスポンスで返される FaceModelVersion フィールドを使用すると、コレクションで使用しているモデルのバージョンを確認できます。DescribeCollection を呼び出して、既存のコレクションのモデルバージョンを取得できます。詳細については、「コレクションの定義」を参照してください。

コレクション内の既存の顔ベクトルは、それ以降のバージョンのモデルに更新することはできません。Amazon Rekognition はソースイメージのバイトを保存しないため、新しいバージョンのモデルを使用して自動的にイメージを再インデックスすることはできません。

既存のコレクションに保存されている顔に対して最新モデルを使用するには、新しいコレクションを作成し (CreateCollection)、この新しいコレクションのイメージにインデックスを付けます (`Indexfaces`)。新しいコレクションの顔の識別子は古いコレクションの顔の識別子とは異なるため、アプリケーションによって保存される顔の識別子をすべて更新する必要があります。古いコレクションが不要になった場合は、DeleteCollection を使用して削除できます。

DetectFaces などのステートレスオペレーションは、モデルの最新バージョンを使用します。

Amazon Rekognition の開始方法

このセクションでは、Amazon Rekognition を使用して開始する方法について説明します。Amazon Rekognition を初めて使用する場合は、最初に Amazon Rekognition の仕組み で説明している概念と用語に目を通すことをお勧めします。

Rekognition を使用する前に、AWS アカウントを作成して AWS アカウント ID を取得する必要があります。また、ユーザーを作成し、Amazon Rekognition システムがそのリソースにアクセスするために必要なアクセス許可を持っているかどうかを判断できるようにします。

アカウントを作成したら、 AWS CLI と AWS SDKsをインストールして設定します。 AWS CLI では、コマンドラインを介して Amazon Rekognition およびその他の サービスとやり取りできます。AWS SDKs では、Java や Python などのプログラミング言語を使用して Amazon Rekognition とやり取りできます。

AWS CLI と AWS SDKs を設定したら、両方を使用する方法の例をいくつか見ることができます。コンソールを使用して Amazon Rekognition と対話する方法の例をいくつか表示することもできます。

Amazon Rekognition コンソールでは、一括分析とカスタムモデレーションのワークフローを使用できます。これらの機能に加えて、コンソールでは Amazon Rekognition のさまざまな機能 (デモ) を試すこともできます。なお、Amazon Rekognition のほとんどの機能を利用するには、API の使用が必要です。

トピック

- ステップ 1: AWS アカウントを設定してユーザーを作成する
- ステップ 2: AWS CLI と AWS SDKsを設定する
- ステップ 3: AWS CLI と AWS SDK API の使用を開始する
- ステップ 4: Amazon Rekognition コンソールの使用開始

ステップ 1: AWS アカウントを設定してユーザーを作成する

Amazon Rekognition を初めて使用する場合は、事前に以下のタスクをすべて完了する必要があります。

1. AWS アカウントにサインアップします。
2. ユーザーを作成します。

デベロッパーガイドのこのセクションでは、 AWS アカウントとユーザーを作成する理由と作成方法について説明します。

トピック

- AWS アカウントとユーザーを作成する

AWS アカウントとユーザーを作成する

AWS アカウント

アマゾン ウェブ サービス (AWS) にサインアップすると、Amazon Rekognition を含む AWS のすべてのサービスに対して AWS アカウントが自動的にサインアップされます。サービスを実際に使用した分の料金のみが請求されます。

Amazon Rekognition は、使用するリソース分のみお支払いいただくだけで利用可能です。

新規の AWS お客様は、無料で Amazon Rekognition の使用を開始できます。詳細については、「AWS 無料利用枠」を参照してください。

アカウントの作成手順については、この後の「にサインアップする AWS アカウント」セクションを参照してください。

アカウントがすでにある場合は AWS 、アカウントのセットアップをスキップして管理ユーザーを作成します。

[ユーザー]

Amazon Rekognition などの AWS のサービスは、サービスにアクセスする際に認証情報を提供する必要があります。これにより、サービスのリソースにアクセスする権限の有無が判定されます。

コンソールの使用時に AWS CLI または APIs にアクセスするための AWS アカウントのアクセスキーを作成するには、パスワードが必要です。ただし、AWS アカウントのルートユーザーの認証情報を使用して AWS にアクセスすることはお勧めしません。代わりに、 AWS Identity and Access Management (IAM) を使用して管理ユーザーを作成することをお勧めします。

それにより、専用の URL と、その管理者ユーザーの認証情報を使用することで AWS にアクセスできます。

AWS にサインアップした時点で自分用の IAM ユーザーをまだ作成していない場合は、IAM コンソールを使用して作成できます。管理者ユーザーの作成方法については、この後の「管理アクセスを持つユーザーを作成する」セクションを参照してください。

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、電話またはテキストメッセージを受け取り、電話キーパッドで検証コードを入力します。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザー が作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティベストプラクティスとして、ユーザーに管理アクセス権を割り当て、ルートユーザーアクセスが必要なタスクの実行にはルートユーザーのみを使用するようにしてください。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] をクリックして、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理することができます。

管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、日常的なタスクにルートユーザーを使用しないように AWS アカウントのルートユーザー、 を保護し AWS IAM Identity Center、 を有効にして管理ユーザーを作成します。

を保護する AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS Management Consoleとして にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドのルートユーザーとしてサインインするを参照してください。
2. ルートユーザーの多要素認証 (MFA) を有効にします。

手順については、IAM ユーザーガイドの AWS アカウント「ルートユーザー (コンソール) の仮想 MFA デバイスを有効にする」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。

2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、「 AWS IAM Identity Center ユーザーガイド」の「デフォルトを使用してユーザーアクセスを設定する IAM アイデンティティセンターディレクトリ」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、AWS サインイン「 ユーザーガイド」の AWS 「 アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。

2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの結合」を参照してください。

ステップ 2: AWS CLI と AWS SDKsを設定する

トピック

- プログラマチックアクセス権を付与する
- AWS SDK での Rekognition の使用

次の手順では、このドキュメントの例で使用されている AWS Command Line Interface（ AWS CLI) と AWS SDKs をインストールする方法を示します。 AWS SDK 呼び出しを認証するには、さまざまな方法があります。このガイドの例では、 AWS CLI コマンドの呼び出しと AWS SDK API オペレーションにデフォルトの認証情報プロファイルを使用していることを前提としています。

使用可能な AWS リージョンのリストについては、の「リージョンとエンドポイント」を参照してくださいAmazon Web Services 全般のリファレンス。

手順に従って、 AWS SDKsをダウンロードして設定します。

AWS CLI と AWS SDKsをセットアップするには

1. 使用する AWS CLIと AWS SDKsをダウンロードしてインストールします。このガイドでは AWS CLI、、Java、Python、Ruby、Node.js、PHP、.NET、JavaScript の例を示します JavaScript 。 AWS SDKs「Amazon Web Services のツール」を参照してください。
2. 「AWS アカウントとユーザーを作成する」で作成したユーザーのアクセスキーを作成します。
   a. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
   b. ナビゲーションペインで [ユーザー] を選択します。
   c. 「AWS アカウントとユーザーを作成する」で作成したユーザーの名前を選択します。
   d. [Security credentials] タブを選択します。
   e. [Create access key] (アクセスキーの作成) を選択します。次に、[.csv ファイルのダウンロード] を選択して、ご使用のコンピュータにアクセスキー ID とシークレットアクセスキーをファイルに保存します。このファイルは安全な場所に保存してください。このダイアログボックスを閉じた後で、シークレットアクセスキーに再度アクセスすることはできません。CSV ファイルをダウンロードしたら、[Close] を選択します。
3. をインストールしている場合は AWS CLI、コマンドプロンプトで `aws configure` を入力して、ほとんどの AWS SDKsの認証情報とリージョンを設定できます。それ以外の場合は、以下の手順をすべて行います。

4. コンピュータでホームディレクトリに移動し .aws ディレクトリを作成します。Linux または macOS のような Unix ベースのシステムでは、次の場所になります。

```
~/.aws
```

Windows では、次の場所になります。

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

5. .aws ディレクトリで credentials という名前の新しいファイルを作成します。
6. ステップ 2 で作成した認証情報 CSV ファイルを開き、コンテンツを以下の形式を使用して credentials ファイルにコピーします。

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

your_access_key_id および your_secret_access_key を、アクセスキー ID とシークレットアクセスキーに置き換えます。

7. Credentials ファイルを保存し、CSV ファイルを削除します。
8. .aws ディレクトリで config という名前の新しいファイルを作成します。
9. config ファイルを開き、リージョンを次の形式で入力します。

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

ご所望の AWS リージョン (例えば us-west-2) に your_aws_region を置き換えます。

> **Note**
>
> リージョンを選択しないと、デフォルトで us-east-1 が使用されます。

10. config ファイルを保存します。

プログラマチックアクセス権を付与する

このガイドの AWS CLI および コード例は、ローカルコンピュータまたは Amazon Elastic Compute Cloud インスタンスなどの他の AWS 環境で実行できます。例を実行するには、例が使用する AWS SDK オペレーションへのアクセスを許可する必要があります。

トピック

- ローカルコンピュータでのコードの実行
- AWS 環境でのコードの実行

ローカルコンピュータでのコードの実行

ローカルコンピュータでコードを実行するには、短期間の認証情報を使用して AWS SDK オペレーションへのアクセス権をユーザーに付与することをお勧めします。ローカルコンピュータでの AWS CLI および コード例の実行に関する具体的な情報については、「」を参照してくださいローカルコンピュータでのプロファイルの使用。

ユーザーが の AWS 外部で を操作する場合は、プログラムによるアクセスが必要です AWS Management Console。プログラムによるアクセスを許可する方法は、 がアクセスするユーザーのタイプによって異なります AWS。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ (IAM アイデンティティセンターで管理されているユーザー)	一時的な認証情報を使用して AWS CLI、、 AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「 を使用する AWS CLI ように を設定する AWS IAM Identity Center」を参照してください。

プログラマチックアクセス権を必要とするユーザー	目的	方法
		• AWS SDKs、ツール、API については、 AWS APIs「 SDK およびツールリファレンスガイド」の「IAM アイデンティティセンター認証」を参照してください。 AWS SDKs
IAM	一時的な認証情報を使用して AWS CLI、、 AWS SDKs、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。
IAM	(非推奨) 長期認証情報を使用して、 AWS CLI、 AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールリファレンスガイド」の「長期認証情報を使用した認証」を参照してください。 AWS SDKs • API AWS APIs「IAM ユーザーガイド」の「IAM ユーザーのアクセスキーの管理」を参照してください。

ローカルコンピュータでのプロファイルの使用

このガイドの AWS CLI および コード例は、 で作成した短期認証情報を使用して実行できますローカルコンピュータでのコードの実行。認証情報や他の設定情報を取得するため、たとえばサンプルでは `profile-name` という名前のプロファイルを使用しています:

```
session = boto3.Session(profile_name="profile-name")
rekognition_client = session.client("rekognition")
```

プロファイルが表すユーザーには、例で必要な Rekognition SDK オペレーションやその他の AWS SDK オペレーションを呼び出すアクセス許可が必要です。

AWS CLI および コード例で動作するプロファイルを作成するには、次のいずれかを選択します。作成するプロファイルの名前が `profile-name` であることを確かめてください。

- IAM が管理するユーザー - 「IAM ロール (AWS CLI) の切り替え」の手順に従います。
- ワークフォース ID (ユーザーが管理 AWS IAM Identity Center) — 使用する AWS CLI の設定 AWS IAM Identity Centerの手順に従います。コード例については統合開発環境 (IDE) を使用することが推奨されます。こちらは、IAM Identity Center による認証を許可する AWS Toolkit をサポートしています。Java の例については「Start building with Java」を参照してください。Python の例については「Start building with Python」を参照してください。その他の詳細については「IAM Identity Center credentials」を参照してください。

> **Note**
>
> コードを使用して、短期間の認証情報を取得できます。詳細については「IAM ロール (AWS API) の切り替え」を参照してください。IAM Identity Center の場合は、「Getting IAM role credentials for CLI access」にある手順に従って、ロールの短期間の認証情報を取得します。

AWS 環境でのコードの実行

AWS Lambda 関数で実行されている本番コードなどの AWS 環境で、ユーザー認証情報を使用して AWS SDK 呼び出しに署名しないでください。代わりに、コードに必要なアクセス権限を定義するロールを設定します。次に、コードを実行する環境にそのロールをアタッチします。ロールをアタッチして一時的な認証情報を利用できるようにする方法は、コードを実行する環境によって異なります。

- AWS Lambda 関数 — Lambda 関数の実行ロールを引き受けるときに Lambda が自動的に関数に提供する一時的な認証情報を使用します。認証情報は Lambda の環境変数で使用できます。プロファイルを指定する必要はありません。詳細については、「Lambda 実行ロール」を参照してください。
- Amazon EC2 - Amazon EC2 のインスタンスメタデータエンドポイント認証情報プロバイダーを使用します。このプロバイダーは、Amazon EC2 インスタンスにアタッチされた Amazon EC2 インスタンスプロファイルを使用して、認証情報を自動的に生成して更新します。詳細については、「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用してアクセス許可を付与する」を参照してください。
- Amazon Elastic Container Service - コンテナ認証情報プロバイダーを使用します。Amazon ECS は認証情報をメタデータエンドポイントに送信して更新します。指定するタスク IAM ロールは、アプリケーションが使用する認証情報を管理するための戦略を提供します。詳細については、「Interact with AWS services」を参照してください。

認証情報プロバイダーの詳細については、「標準認証情報プロバイダー」を参照してください。

AWS SDK での Rekognition の使用

AWS Software Development Kit (SDKsは、多くの一般的なプログラミング言語で使用できます。各 SDK には、デベロッパーが好みの言語でアプリケーションを簡単に構築できるようにする API、コード例、およびドキュメントが提供されています。

SDK ドキュメント	コード例
AWS SDK for C++	AWS SDK for C++ コード例
AWS CLI	AWS CLI コード例
AWS SDK for Go	AWS SDK for Go コード例
AWS SDK for Java	AWS SDK for Java コード例
AWS SDK for JavaScript	AWS SDK for JavaScript コード例
AWS SDK for Kotlin	AWS SDK for Kotlin コード例
AWS SDK for .NET	AWS SDK for .NET コード例

SDK ドキュメント	コード例
AWS SDK for PHP	AWS SDK for PHP コード例
AWS Tools for PowerShell	AWS Tools for PowerShell コード例
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) コード例
AWS SDK for Ruby	AWS SDK for Ruby コード例
AWS SDK for Rust	AWS SDK for Rust コード例
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP コード例
AWS SDK for Swift	AWS SDK for Swift コード例

Rekognition に固有の例については、SDK を使用した Amazon Rekognition のコード例 AWS SDKs を参照してください。

> **可用性の例**
>
> 必要なものが見つからなかった場合。このページの下側にある [Provide feedback (フィードバックを送信)] リンクから、コードの例をリクエストしてください。

ステップ 3: AWS CLI と AWS SDK API の使用を開始する

使用する AWS CLI および AWS SDKs を設定したら、Amazon Rekognition を使用するアプリケーションを構築できます。Amazon Rekognition の一部の機能にはコンソールのワークフローがありますが、Amazon Rekognition とのほとんどのやり取りは API オペレーションを使用して行います。

以下のトピックでは、 AWS CLI または AWS SDKs を使用して Amazon Rekognition Image と Amazon Rekognition Video の使用を開始する方法について説明します。

- イメージの操作 - Amazon Rekognition Image を使用して、画像を分析するプロセスについて説明します。
- 保存済み動画分析での作業 - Amazon Rekognition Video を使用して、ストリーミングではなく保存済みのビデオを分析するプロセスについて説明します。

- ストリーミングビデオイベントの操作 - Amazon Rekognition Video を使用して、ストリーミングビデオを分析するプロセスについて説明します。

上記のセクションには、 AWS CLIを使用した例があります。を使用する場合は AWS CLI、API コールのフォーマット方法の詳細については、次のセクションを参照してください。

AWS CLI 例のフォーマット

このガイド AWS CLI の例は、Linux オペレーティングシステム用にフォーマットされています。Microsoft Windows で例を使用するには、--image パラメータの JSON 形式を変更し、改行をバックスラッシュ (\) からキャレット (^) に変える必要があります。JSON 形式の詳細については、「AWS Command Line Interface のパラメータ値の指定」を参照してください。

以下は、Microsoft Windows 用にフォーマットされた AWS CLI コマンドの例です (これらのコマンドはそのままは実行されず、単なるフォーマット例であることに注意してください)。

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

Microsoft Windows と Linux の両方で動作する短縮バージョンの JSON を提供することもできます。

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

詳細については、「AWS Command Line Interface を使用した短縮構文の使用」を参照してください。

次のステップ

ステップ 4: Amazon Rekognition コンソールの使用開始

ステップ 4: Amazon Rekognition コンソールの使用開始

Amazon Rekognition コンソールでは、Rekognition Custom Labels とカスタムモデレーション機能に関連するリソースを管理できます。Rekognition の他の機能については、デモのみが提供されます。

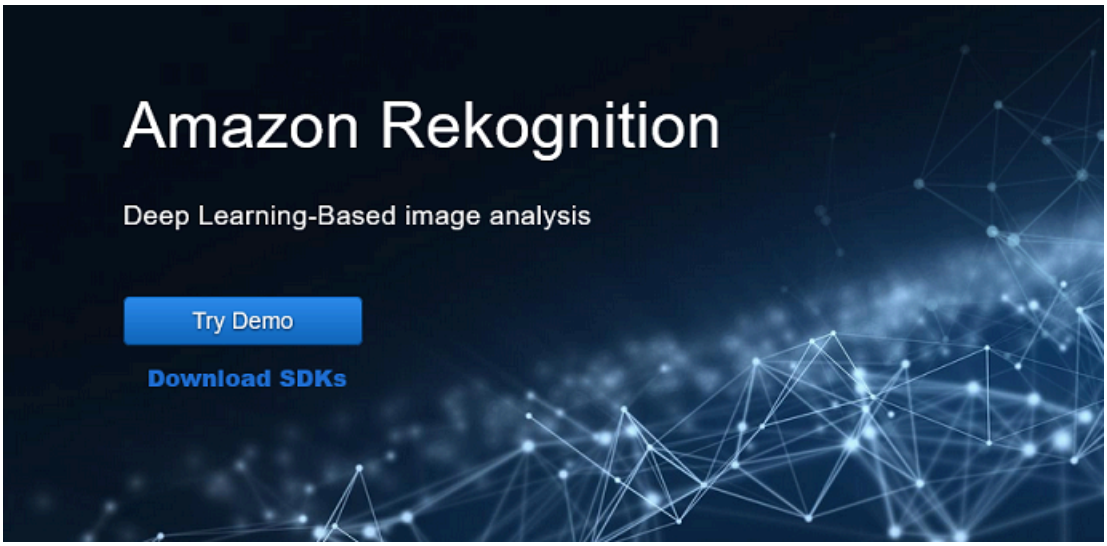
このセクションでは、物体とシーンの検出、顔の分析、イメージ間での顔の比較など、Amazon Rekognition の一部の機能について使用方法を示します。詳細については、「Amazon Rekognition

の仕組み」を参照してください。Amazon Rekognition API または を使用して、オブジェクトやシーン AWS CLI の検出、顔の検出、顔の比較と検索を行うこともできます。詳細については、「ステップ 3: AWS CLI と AWS SDK API の使用を開始する」を参照してください。

このセクションでは、Rekognition コンソール を使用して Amazon CloudWatch のメトリクスの集計を確認する方法についても説明します。

トピック

- コンソールのアクセス権限のセットアップ
- 演習 1: 物体とシーンを検出する (コンソール)
- 演習 2: イメージ内の顔を分析する (コンソール)
- 演習 3: イメージ内の顔を比較する (コンソール)
- 演習 4: 集計メトリクスを参照する (コンソール)



コンソールのアクセス権限のセットアップ

Rekognition のコンソールを使用するには、コンソールにアクセスするロールまたはアカウントに適したアクセス権限が必要です。オペレーションによっては、オペレーション中に処理するファイルを保存するための Amazon S3 バケットが、Rekognition によって自動作成されます。トレーニングファイルをこのコンソールバケット以外のバケットに保存する場合は、追加のアクセス権限が必要になります。

コンソールへのアクセスを許可する

Rekognition のコンソールを使用するときは、以下のような IAM ポリシーを使用します。このポリシーは Amazon S3 と Rekognition のコンソールを対象としています。アクセス権限の割り当てに関する詳細は、以下の「アクセス権限の割り当て」を参照してください。

```
            {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RekognitionFullAccess",
            "Effect": "Allow",
            "Action": [
                "rekognition:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "RekognitionConsoleS3BucketSearchAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation"
            ],
            "Resource": "*"
        },
        {
            "Sid": "RekognitionConsoleS3BucketFirstUseSetupAccess",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutBucketVersioning",
                "s3:PutLifecycleConfiguration",
                "s3:PutEncryptionConfiguration",
                "s3:PutBucketPublicAccessBlock",
                "s3:PutCors",
                "s3:GetCors"
            ],
            "Resource": "arn:aws:s3:::rekognition-custom-projects-*"
        },
```

```
        {
            "Sid": "RekognitionConsoleS3BucketAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:GetBucketVersioning"
            ],
            "Resource": "arn:aws:s3:::rekognition-custom-projects-*"
        },
        {
            "Sid": "RekognitionConsoleS3ObjectAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:HeadObject",
                "s3:DeleteObject",
                "s3:GetObjectAcl",
                "s3:GetObjectTagging",
                "s3:GetObjectVersion",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::rekognition-custom-projects-*/*"
        },
        {
            "Sid": "RekognitionConsoleManifestAccess",
            "Effect": "Allow",
            "Action": [
                "groundtruthlabeling:*",
            ],
            "Resource": "*"
        },
        {
            "Sid": "RekognitionConsoleTagSelectorAccess",
            "Effect": "Allow",
            "Action": [
                "tag:GetTagKeys",
                "tag:GetTagValues"
            ],
            "Resource": "*"
        },
        {
            "Sid": "RekognitionConsoleKmsKeySelectorAccess",
            "Effect": "Allow",
```

```
            "Action": [
                "kms:ListAliases"
            ],
            "Resource": "*"
        }
    ]
}
```

外部の Amazon S3 バケットへのアクセス

新しい AWS リージョンで Rekognition コンソールを初めて開くと、Rekognition はプロジェクトファイルの保存に使用されるバケット (コンソールバケット) を作成します。あるいは、自分の Amazon S3 バケット (外部バケット) を使用してイメージまたはマニフェストファイルをコンソールにアップロードすることもできます。外部バケットを使用するには、前のポリシーに以下のポリシーブロックを追加します。

```
{
            "Sid": "s3ExternalBucketPolicies",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:GetObjectVersion",
                "s3:GetObjectTagging",
                "s3:ListBucket",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket*"
            ]
}
```

権限の割り当て

アクセスを提供するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- AWS IAM アイデンティティセンター (AWS Single Sign-On の後継サービス) のユーザーとグループ:

  アクセス許可セットを作成します。「AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide」の「Create a permission set」の指示に従います。

- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。

- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。
  - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加する。詳細については「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス権限の追加」を参照してください。

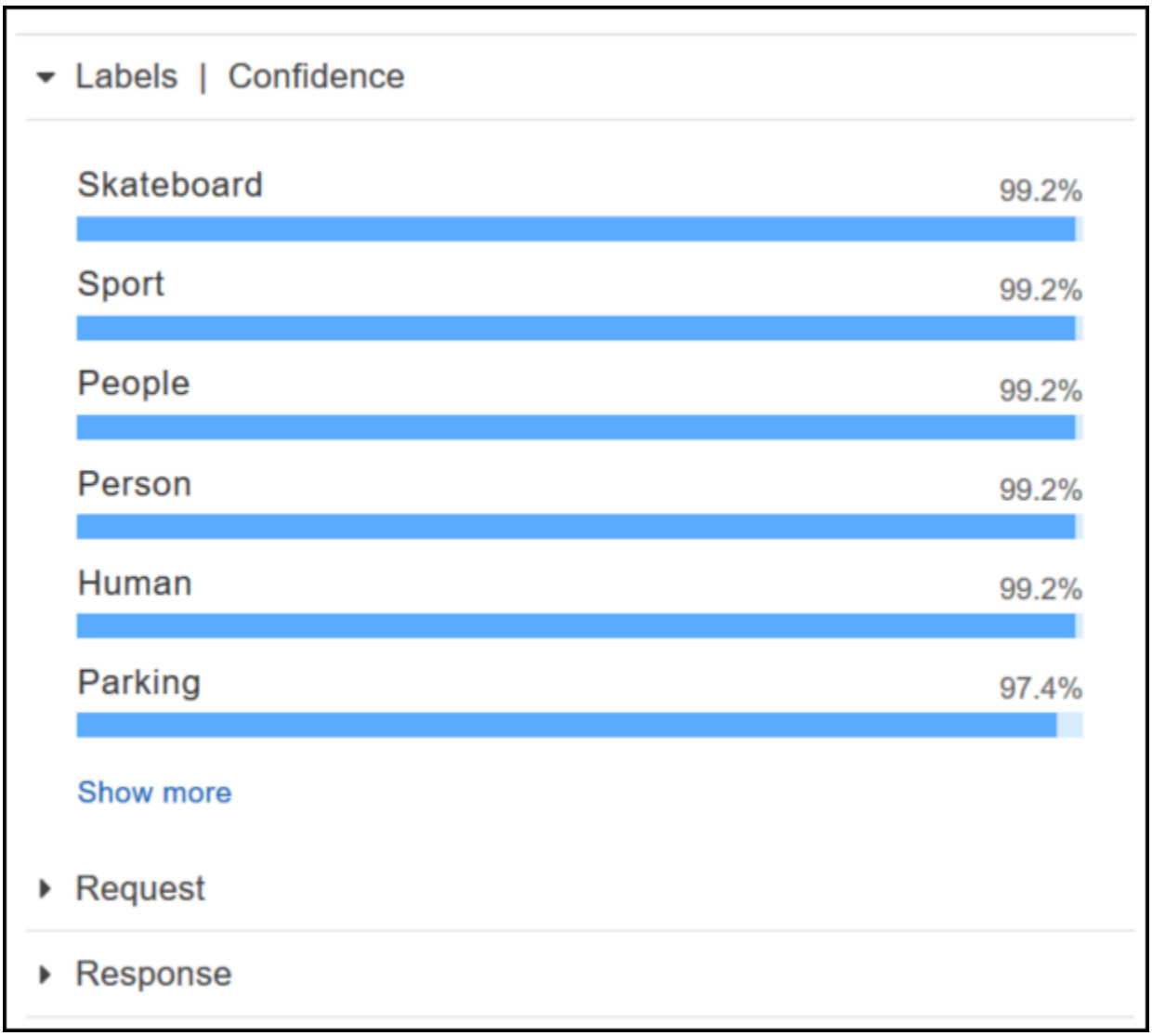
演習 1: 物体とシーンを検出する (コンソール)

このセクションでは、どのように Amazon Rekognition の物体とシーンを高い水準で検出する能力が機能するかを示します。入力としてイメージを指定すると、サービスはイメージ内の物体とシーンを検出して、これらをそれぞれの信頼スコア (%) とともに返します。

たとえば、 Amazon Rekognition は、以下のサンプルイメージから物体やシーンとしてスケートボード、スポーツ、人、自動車、乗り物などを検出します。



Amazon Rekognition は、以下のレスポンスの例に示しているように、サンプルイメージ内で検出した物体ごとの信頼スコアも返します。



このレスポンスで返されるすべての信頼スコアを表示するには、[Labels | Confidence] ペインの [Show more] を選択します。

API へのリクエストと API からのレスポンスも参照できます。

リクエスト

```
{
   "contentString":{
      "Attributes":[
         "ALL"
      ],
      "Image":{
         "S3Object":{
            "Bucket":"console-sample-images",
```

```
            "Name":"skateboard.jpg"
         }
      }
   }
}
```

レスポンス

```
{
   "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"
         }
      ]
}
```


詳細については、「Amazon Rekognition の仕組み」を参照してください。

提供したイメージ内の物体とシーンを検出する

Amazon Rekognition コンソールで、入力として自分でイメージをアップロードするか、イメージへの URL を提供することもできます。Amazon Rekognition は、提供されたイメージ内で検出した物体、シーン、およびシーンごとの信頼スコアを返します。

> **Note**
>
> イメージサイズは 5 MB 未満の JPEG または PNG 形式にする必要があります。

指定したイメージ内の物体とシーンを検出するには

1. Amazon Rekognition コンソールを https://console.aws.amazon.com/rekognition/ で開きます。

2. ラベル検出 を選択します。
3. 次のいずれかを行います：

   - イメージをアップロードする – [Upload] を選択し、イメージの保存先に移動してイメージを選択します。
   - URL を使用する – テキストボックスに URL を入力して、[Go] を選択します。
4. [Labels | Confidence] ペインで、検出されたラベルごとの信頼スコアを確認します。

イメージ解析オプションの詳細については、「the section called “イメージの操作”」を参照してください。

提供するビデオ内の物体や人物を検出する

Amazon Rekognition コンソールに入力として提供したビデオをアップロードできます。Amazon Rekognition は、ビデオ内で検出された人、オブジェクト、ラベルを返します。

> **Note**
>
> デモビデオの長さは 1 分以上または 30 MB を超えないようにする必要があります。MP4 ファイル形式で、H.264 コーデックでエンコードする必要があります。

ビデオ内の人物や物体を検出する

1. Amazon Rekognition コンソールを https://console.aws.amazon.com/rekognition/ で開きます。
2. ナビゲーションバーから [保存済み動画分析] を選択します。
3. [サンプルを選択するか、独自のサンプルをアップロードします] で、ドロップダウンメニューから [独自のビデオ] を選択します。
4. ビデオをドラッグアンドドロップするか、保存した場所からビデオを選択します。

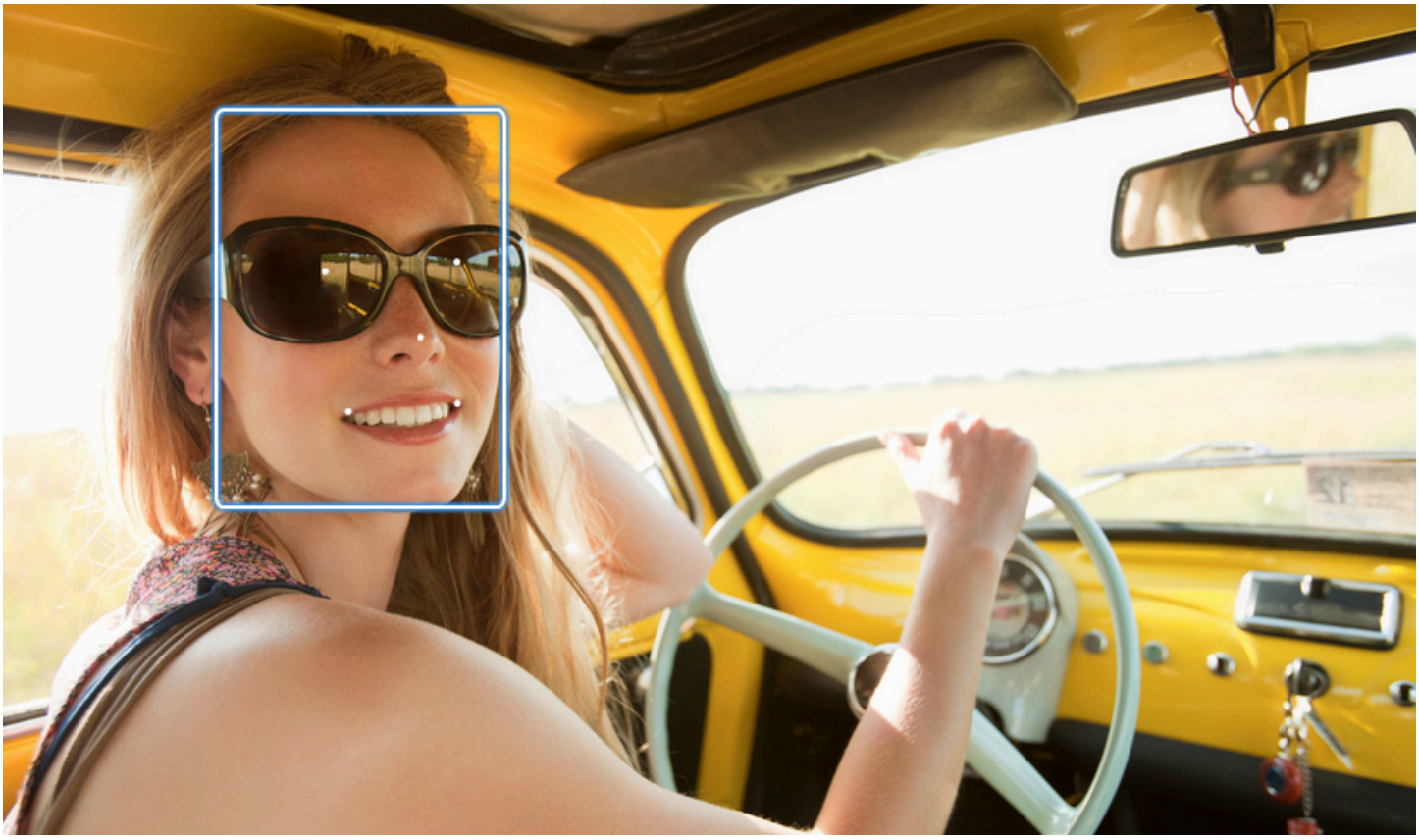
ビデオ分析オプションの詳細については、「the section called “保存済み動画分析での作業” または the section called “ストリーミングビデオイベントの操作”」を参照してください。

演習 2: イメージ内の顔を分析する (コンソール)

このセクションでは、Amazon Rekognition コンソールを使用してイメージ内の顔を検出し、顔属性を分析する方法を示します。顔を含むイメージを入力として指定すると、イメージ内の顔が検出され

て顔属性が分析され、イメージ内で検出された顔と顔属性の信頼スコア (%) が返されます。詳細については、「Amazon Rekognition の仕組み」を参照してください。

たとえば、以下のサンプルイメージを入力として選択すると、Amazon Rekognition によって顔として検出され、検出された顔と顔属性の信頼スコアが返されます。



このレスポンス例は以下のとおりです。

Results



looks like a face	99.8%
appears to be female	100%
age range	23 - 38 years old
smiling	99.4%
appears to be happy	93.2%
wearing eyeglasses	99.9%
wearing sunglasses	97.6%
eyes are open	96.2%
mouth is open	72.5%
does not have a mustache	77.6%
does not have a beard	97.1%

Show less

入力イメージ内に複数の顔がある場合、Rekognition はイメージ内の最大 100 までの顔を検出します。検出された顔ごとに正方形で囲まれます。顔を囲む正方形内をクリックすると、Rekognition はその顔の信頼スコアと検出された属性を [Faces | Confidence] ペインに表示します。

提供したイメージ内の顔を分析する

Amazon Rekognition コンソールで、自分でイメージをアップロードするか、イメージへの URL を提供することができます。

> **Note**
>
> イメージサイズは 5 MB 未満の JPEG または PNG 形式にする必要があります。

指定したイメージ内の顔を分析するには

1. Amazon Rekognition コンソールを https://console.aws.amazon.com/rekognition/ で開きます。
2. [Facial analysis] を選択します。
3. 次のいずれかを行います :
   - イメージをアップロードする – [Upload] を選択し、イメージの保存先に移動してイメージを選択します。
   - URL を使用する – テキストボックスに URL を入力して、[Go] を選択します。
4. 検出された顔のいずれかの信頼スコアと顔の属性を [Faces | Confidence] ペインで確認します。
5. イメージ内に複数の顔がある場合は、別の顔を選択してその属性とスコアを確認します。

演習 3: イメージ内の顔を比較する (コンソール)

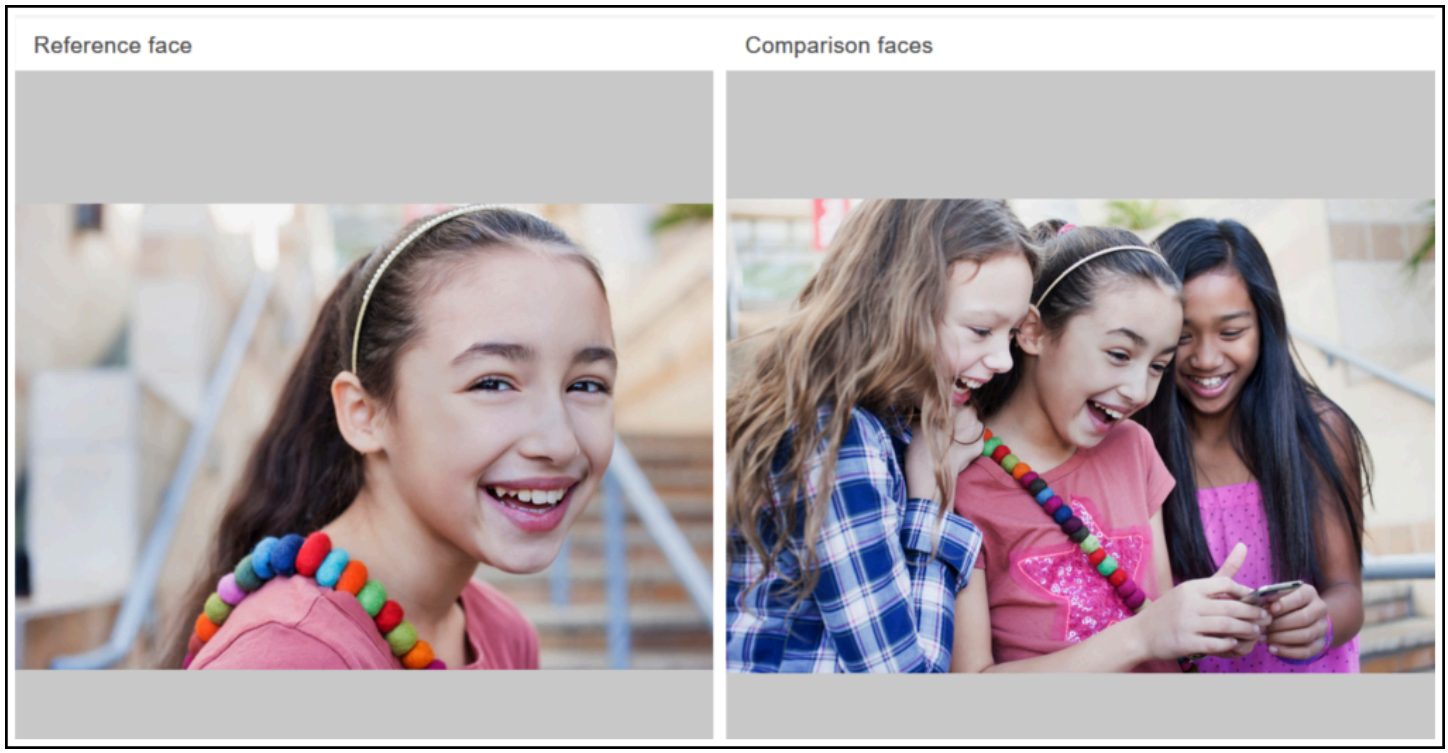
このセクションでは、Amazon Rekognition コンソールを使用して、複数のイメージ内で特定の顔と他の顔を比較する方法を示します。[Reference face] (ソース) と [Comparison faces] (ターゲット) イメージを指定すると、Rekognition はソースイメージ内で最大の顔 (参照顔) とターゲットイメージ内で検出された最大 100 個までの顔 (比較顔) を比較し、ソースの顔とターゲットイメージの顔の類似度を確認します。比較ごとの類似度スコアは [Results] ペインに表示されます。

ターゲットイメージ内に複数の顔がある場合は、Rekognition はソースイメージ内の顔とターゲットイメージ内で検出された最大 100 個の顔を比較し、それぞれの類似度スコアを割り当てます。

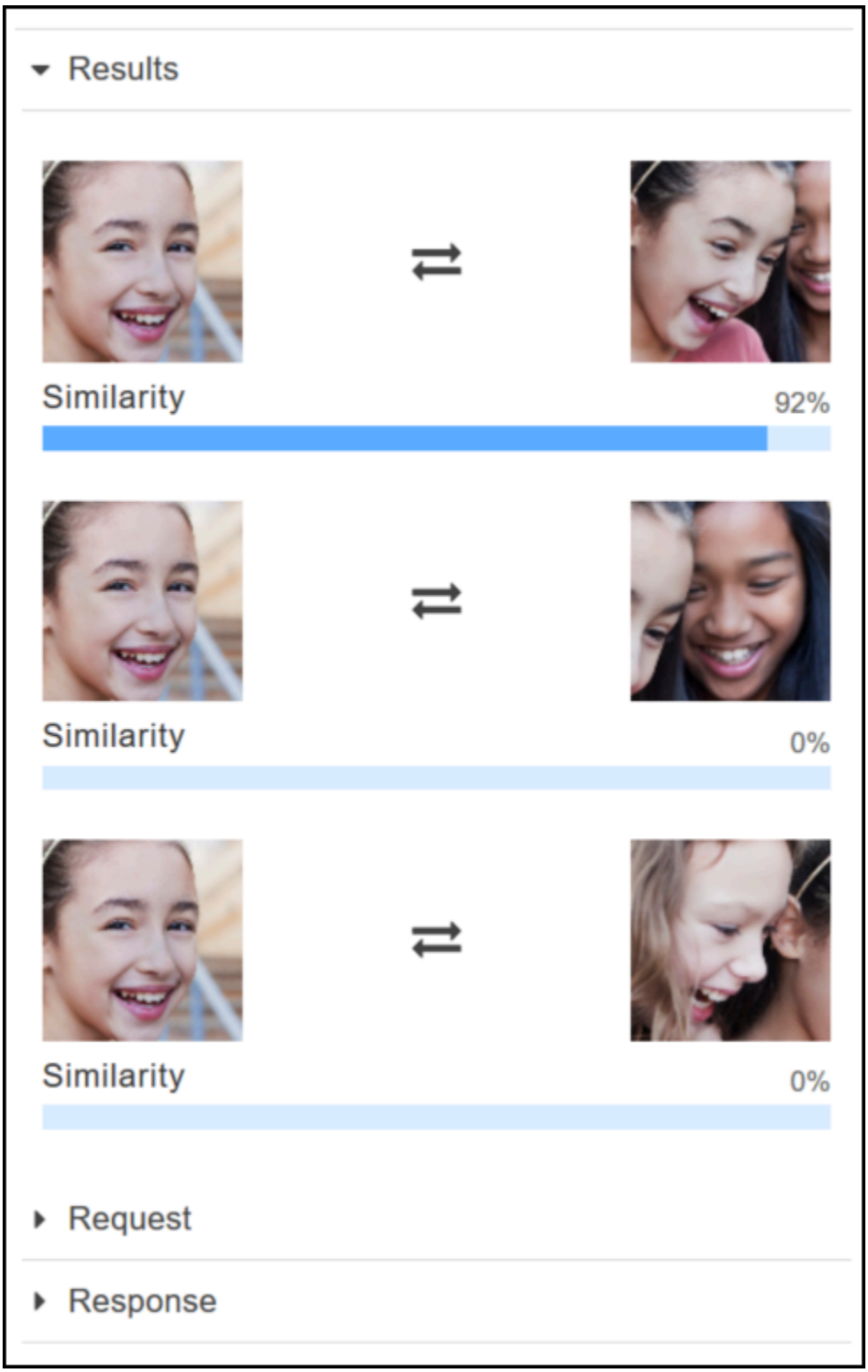
ソースイメージ内に複数の顔がある場合は、ソースイメージ内の最大サイズの顔が検出され、この顔とターゲットイメージ内で検出された各顔が比較されます。

詳細については、「イメージ間の顔の比較」を参照してください。

たとえば、左側のイメージをソースイメージとし、右側のイメージをターゲットイメージとした場合、Rekognition はソースイメージ内の顔を検出して、ターゲットイメージ内の各顔と比較し、ペアごとに類似度スコアを表示します。



ターゲットイメージで検出された顔と顔ごとの類似度スコは以下のとおりです。



提供したイメージ内の顔を比較する

Rekognition でソースイメージとターゲットイメージの顔を比較するには、自分でイメージをアップロードするか、イメージがある場所への URL を指定することができます。

> **Note**
>
> イメージサイズは 5 MB 未満の JPEG または PNG 形式にする必要があります。

イメージ内の顔を比較するには

1. Amazon Rekognition コンソールを https://console.aws.amazon.com/rekognition/ で開きます。
2. [Face comparison] を選択します。
3. ソースイメージについて、次のいずれかを行います。
   - イメージをアップロードする – 左側の [Upload] を選択し、ソースイメージの保存先に移動してイメージを選択します。
   - URL を使用する – テキストボックスにソースイメージの URL を入力して、[Go] を選択します。
4. ターゲットイメージについて、次のいずれかを行います。
   - イメージをアップロードする – 右側の [Upload] を選択し、ソースイメージの保存先に移動してイメージを選択します。
   - URL を使用する – テキストボックスにソースイメージの URL を入力して、[Go] を選択します。
5. Rekognition は、出典イメージ内でターゲットイメージ内の最大 100 個の顔を比較し、各ペアの類似度スコアを [結果] ペインに表示します。

演習 4: 集計メトリクスを参照する (コンソール)

Amazon Rekognition のメトリクスペインには、指定した期間にわたる個別の Rekognition メトリクスが集計されて、アクティビティグラフとして表示されます。たとえば、`SuccessfulRequestCount` の集計メトリクスには、過去 7 日間にわたるすべての Rekognition API オペレーションへの成功したリクエストの総数が表示されます。

次の表は、Rekognition メトリクスペインに表示されるグラフおよび対応する Rekognition メトリクスの一覧です。詳細については、「Rekognition の CloudWatch メトリクスです。」を参照してください。

グラフ	集計メトリクス
成功した呼び出し	SuccessfulRequestCount
クライアントエラー	UserErrorCount
サーバーエラー	ServerErrorCount
スロットル済み	ThrottledCount
検出したラベル	DetectedLabelCount
検出した顔	DetectedFaceCount

各グラフには、指定した期間にわたって収集された集計メトリクスデータが表示されます。同期間の集計メトリクスデータの総数も表示されます。個別の API コールのメトリクスを表示するには、各グラフの下にあるリンクを選択します。

Rekognition メトリクスペインへのアクセスをユーザーに許可するには、ユーザーが CloudWatch と Rekognition に対する適切なアクセス権限を持っていることを確認します。たとえば、`AmazonRekognitionReadOnlyAccess` および `CloudWatchReadOnlyAccess` 管理ポリシーへのアクセス権限を持つユーザーは、メトリクスペインを表示できます。必要なアクセス権限を持っていないユーザーがメトリクスペインを開いても、何のグラフも表示されません。詳細については、「Amazon Rekognition の Identity and Access Management」を参照してください。

CloudWatch を使用した Rekognition のモニタリングの詳細については、「Amazon CloudWatch による Rekognition のモニタリング」を参照してください。

集計メトリクスを表示するには (コンソール)

1. Amazon Rekognition コンソールを https://console.aws.amazon.com/rekognition/ で開きます。
2. ナビゲーションペインで Metrics (メトリクス) を選択します。
3. ドロップダウンで、メトリクスを確認する期間を選択します。
4. グラフを更新するには、[Refresh] ボタンを選択します。
5. 特定の集計メトリクスの詳細な CloudWatch メトリクスを表示するには、メトリクスのグラフの下にある [CloudWatch で詳細を参照] を選択します。

イメージや動画の操作

Amazon Rekognition API オペレーションは、画像、保存済みビデオ、ストリーミングビデオの 3 種類のメディアで使用できます。このセクションでは、Amazon Rekognition にアクセスして各種メディアを処理するコードの記述方法について一般的な情報を提供します。ベストプラクティスや考慮事項については、処理するメディアのタイプに応じて、以下の各セクションを参照してください。

このガイドの他のセクションでは、顔認識など、特定の種類の画像およびビデオ分析に関する情報を示します。

トピック

- イメージの操作
- 保存済み動画分析での作業
- ストリーミングビデオイベントの操作
- エラー処理
- FedRAMP 認定サービスとしての Amazon Rekognition の使用

イメージの操作

このセクションには、Amazon Rekognition Image でイメージに対して実行できる分析のタイプの説明が含まれます。

- オブジェクトとシーンの検出
- 顔の検出と比較
- コレクション内での顔の検索
- 有名人の認識
- イメージモデレーション
- イメージ内のテキスト検出

これらは非ストレージ型 API オペレーションにより実行され、オペレーションで検出された情報は Amazon Rekognition Image によって保持されません。非ストレージ API オペレーションでは入力イメージのバイトは保持されません。詳細については、「非ストレージ型およびストレージ型の API オペレーションについて」を参照してください。

Amazon Rekognition Image では、コレクションに顔のメタデータを保存して後で取得することもできます。詳細については、「コレクション内での顔の検索」を参照してください。

このセクションでは、Amazon Rekognition Image　API オペレーションを使用して、Amazon S3 バケットに保存されたイメージと、ローカルファイルシステムからロードされたイメージのバイトを分析します。このセクションでは、.jpg イメージからイメージの向きの情報を取得する方法についても説明します。

Rekognition は、RGB チャネルのみを使用して推論を実行します。 は、ディスプレイを使用して比較を視覚的に (手動で) 検査する前に、アルファチャネルを削除 AWS することをお勧めします。

トピック

- イメージの仕様
- Amazon S3 バケットに保存されたイメージの分析
- ローカルファイルシステムからロードしたイメージの分析
- 境界ボックスの表示
- イメージの向きおよび境界ボックス座標の取得

イメージの仕様

Amazon Rekognition Image オペレーションでは、.jpg 形式または .png 形式の入力イメージを分析します。

イメージバイトを呼び出しの一部として Amazon Rekognition Image オペレーションに渡すか、既存の Amazon S3 オブジェクトを参照します。Amazon S3 バケットに保存されているイメージの分析例については、「Amazon S3 バケットに保存されたイメージの分析」を参照してください。Amazon Rekognition Image API オペレーションにイメージバイトを渡す例については、「ローカルファイルシステムからロードしたイメージの分析」を参照してください。

HTTP を使用して Amazon Rekognition Image オペレーションの一部として渡す場合のイメージバイトは、base64 でエンコードされた文字列であることが必要です。AWS SDK を使用して API オペレーションの呼び出しの一部としてイメージのバイトを渡す場合、イメージのバイトを Base64 でエンコードする必要があるかどうかは使用言語によって異なります。

次の AWS SDKsイメージを base64 でエンコードします。Amazon Rekognition Image API オペレーションを呼び出す前にイメージのバイトをエンコードする必要はありません。

- Java

- JavaScript
- Python
- PHP

別の AWS SDK を使用しているときに Rekognition API オペレーションの呼び出しに伴ってイメージ形式エラーが発生した場合は、イメージのバイトを base64 でエンコードしてから Rekognition API オペレーションに渡してみてください。

を使用して Amazon Rekognition Image オペレーション AWS CLI を呼び出す場合、呼び出しの一部としてイメージのバイトを渡すことはサポートされていません。最初に Amazon S3 バケットにイメージをアップロードし、次にアップロードしたイメージを参照するオペレーションを呼び出します。

> **Note**
>
> イメージのバイトの代わりに `S3Object` に保存されたイメージを渡す場合は、イメージを base64 でエンコードする必要はありません。

Amazon Rekognition オペレーションのレイテンシーをできるだけ低くする方法の詳細については、「Amazon Rekognition イメージオペレーションのレイテンシ」を参照してください。

イメージの向きの修正

いくつかの Rekognition　API オペレーションでは、分析されたイメージの向きが返されます。イメージの向きを知ることは、表示用にイメージの向きを変えるために重要です。顔を分析する Rekognition API オペレーションでは、イメージ内の顔の位置の境界ボックスも返されます。境界ボックスを使用して、顔またはイメージの周囲にボックスを表示できます。返される境界ボックスの座標は、イメージの向きによって影響を受けます。顔の周囲に正しくボックスを表示するには、境界ボックスの座標の変換が必要になる場合があります。詳細については、「イメージの向きおよび境界ボックス座標の取得」を参照してください。

イメージのサイズ変更

Amazon Rekognition は、分析中に特定のモデルまたはアルゴリズムに最も適した定義済みの範囲のセットを使用して、内部的に画像のサイズを変更します。Amazon Rekognition は、入力イメージの解像度に応じて、異なる数のオブジェクトを検出したり、異なる結果を提供したりすることがあります。たとえば、2 つのイメージがあるとします。最初の画像の解像度は 1024 x 768 ピクセルです。2

番目の画像 (最初のイメージのサイズを変更したバージョン) の解像度は 640 x 480 ピクセルです。イメージを DetectLabels に送信した場合、DetectLabels への 2 つの呼び出しからのレスポンスが若干異なる可能性があります。

Amazon S3 バケットに保存されたイメージの分析

Amazon Rekognition Image は、 バケットに保存されたイメージ、またはイメージのバイトとして提供されたイメージを分析できます。

このトピックでは、 DetectLabels API オペレーションを使用して、Amazon S3 バケットに保存されたイメージ (JPEG または PNG) 内のオブジェクト、概念、シーンを検出します。イメージを Amazon Rekognition Image API オペレーションに渡すには、Image 入力パラメータを使用します。Image 内で S3Object オブジェクトプロパティを指定し、S3 バケットに保存されたイメージを参照します。Amazon S3 バケットに保存されたイメージのバイトは base64 でエンコードする必要はありません。詳細については、「イメージの仕様」を参照してください。

リクエストの例

次の DetectLabels に対する JSON リクエストの例では、ソースイメージ (input.jpg) を amzn-s3-demo-bucket と名前をつけた Amazon S3 バケット からロードします。S3 オブジェクトが含まれている S3 バケットのリージョンと Amazon Rekognition Image オペレーションで使用するリージョンが一致している必要があります。

```
{
    "Image": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "input.jpg"
        }
    },
    "MaxLabels": 10,
    "MinConfidence": 75
}
```

次の例では AWS SDKs と を使用して AWS CLI を呼び出しますDetectLabels。DetectLabels オペレーションからのレスポンスの詳細については、「DetectLabels のレスポンス」を参照してください。

イメージ内のラベルを検出するには

1. まだ実行していない場合:

a. AmazonRekognitionFullAccess と AmazonS3ReadOnlyAccess のアクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。API オペレーションを呼び出すユーザーに、プログラムによるアクセスに必要な適切な権限が付与されていることを確認してください。付与する方法については、「プログラマチックアクセス権を付与する」を参照してください。

2. 1 つ以上のオブジェクト (樹木、家、ボートなど) が含まれているイメージを S3 バケットにアップロードします。イメージは、.jpg 形式または .png 形式にする必要があります。

   手順については、Amazon Simple Storage Service 入門ガイドの「Amazon S3 へのオブジェクトのアップロード」を参照してください。

3. 以下の例を使用して、DetectLabels オペレーションを呼び出します。

   Java

   この例では、入力イメージ内で検出されたラベルのリストを表示します。bucket および photo の値は、ステップ 2 で使用したバケット名とイメージ名に置き換えます。

```
//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

この例では、detect-labels CLI オペレーションの JSON 出力を表示します。bucket および photo の値は、ステップ 2 で使用した Amazon S3 バケット名とイメージ名に置き換えます。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
aws rekognition detect-labels --image '{ "S3Object": { "Bucket": "bucket-name",
 "Name": "file-name" } }' \
--features GENERAL_LABELS IMAGE_PROPERTIES \
--settings '{"ImageProperties": {"MaxDominantColors":1}, {"GeneralLabels":
{"LabelInclusionFilters":["Cat"]}}}' \
--profile profile-name \
```

```
--region us-east-1
```

Windows を使用しているお客様は、場合によって、以下の例のように引用符を エスケープする必要があります。

```
aws rekognition detect-labels --image "{\"S3Object\":{\"Bucket\":\"bucket-
name\",\"Name\":\"file-name\"}}" --features GENERAL_LABELS IMAGE_PROPERTIES --
settings "{\"GeneralLabels\":{\"LabelInclusionFilters\":[\"Car\"]}}" --profile
 profile-name --region us-east-1
```

Java V2

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。完全な例を こちら を参照してください。

```
//snippet-start:[rekognition.java2.detect_labels.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
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 software.amazon.awssdk.services.rekognition.model.S3Object;
import java.util.List;

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DetectLabels {

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
```

```
            "   <bucket> <image>\n\n" +
            "Where:\n" +
            "   bucket - The name of the Amazon S3 bucket that contains the
 image (for example, ,ImageBucket)." +
            "   image - The name of the image located in the Amazon S3 bucket
 (for example, Lake.png). \n\n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String image = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        getLabelsfromImage(rekClient, bucket, image);
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.detect_labels_s3.main]
    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);
        }
    }
 // snippet-end:[rekognition.java2.detect_labels.main]
}
```

Python

この例では、入力イメージ内で検出されたラベルを表示します。bucket と photo の値を、ステップ 2 で使用した Amazon S3 バケットとイメージの名前に置き換えます。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
#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):

     session = boto3.Session(profile_name='profile-name')
     client = session.client('rekognition')

     response = client.detect_labels(Image={'S3Object':
{'Bucket':bucket,'Name':photo}},
     MaxLabels=10,
     # Uncomment to use image properties and filtration settings
     #Features=["GENERAL_LABELS", "IMAGE_PROPERTIES"],
     #Settings={"GeneralLabels": {"LabelInclusionFilters":["Cat"]},
     # "ImageProperties": {"MaxDominantColors":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("Aliases:")
        for alias in label['Aliases']:
            print(" " + alias['Name'])

            print("Categories:")
        for category in label['Categories']:
            print(" " + category['Name'])
            print("----------")
            print()

    if "ImageProperties" in str(response):
        print("Background:")
        print(response["ImageProperties"]["Background"])
        print()
        print("Foreground:")
        print(response["ImageProperties"]["Foreground"])
        print()
        print("Quality:")
        print(response["ImageProperties"]["Quality"])
        print()

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

```
def main():
    photo = 'photo-name'
    bucket = 'amzn-s3-demo-bucket'
    label_count = detect_labels(photo, bucket)
    print("Labels detected: " + str(label_count))

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

Node.Js

この例では、イメージ内で検出されたラベルに関する情報を表示します。

photo の値は、1 つ以上の有名人の顔が含まれているイメージファイルのパスとファイル名に変更します。bucket の値を、提供されたイメージファイルを含む S3 バケット名に変更します。REGION の値を、アカウントに関連付けられているリージョンの名前に変更します。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

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

import {fromIni} from '@aws-sdk/credential-providers';

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"

// Create SNS service object.
const rekogClient = new RekognitionClient({
  region: REGION,
  credentials: fromIni({
       profile: 'profile-name',
  }),
});

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

// Set params
const params = {For example, to grant
```

```
    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

この例では、入力イメージ内で検出されたラベルのリストを表示します。bucket および photo の値は、ステップ 2 で使用した Amazon S3 バケット名とイメージ名に置き換えます。

```
//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 = "amzn-s3-demo-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

この例では、入力イメージ内で検出されたラベルのリストを表示します。bucket と photo の値を、ステップ 2 で使用した Amazon S3 バケットとイメージの名前に置き換えます。

```
 # 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
```

レスポンスの例

DetectLabels からのレスポンスは、イメージ内で検出されたラベルとラベルの検出に使用された信頼度の配列です。

イメージに対する DetectLabels オペレーションを実行するときは、Amazon Rekognition は次の応答例のような出力を返します。

このレスポンスは、オペレーションが Person、Vehicle、Car を含む複数のラベルを検出したことを示しています。各ラベルに信頼度が関連付けられています。たとえば、イメージに人物が含まれている確率として、検出アルゴリズムの信頼度は 98.991432% です。

レスポンスには Parents 配列内のラベルの先祖ラベルも含まれます。たとえば、ラベル Automobile には、Vehicle と Transportation という名前の 2 つの親ラベルがあります。

共通オブジェクトラベルに対するレスポンスは、入力画像上のラベルの位置に対する境界ボックス情報を含みます。例えば、Person ラベルには、2 つの境界ボックスを含むインスタンス配列があります。これらは、画像内で検出された 2 人の人物の位置です。

フィールド LabelModelVersion には DetectLabels で使用される検出モデルのバージョン番号が含まれます。

この DetectLabels オペレーションの使用の詳細については、「オブジェクトおよび概念の検出」を参照してください。

```
{

    {
    "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"
}


}
```

ローカルファイルシステムからロードしたイメージの分析

Amazon Rekognition Image は、Amazon S3 バケットに保存されたイメージ、またはイメージのバイトとして提供されたイメージを分析できます。

以下のトピックでは、ローカルファイルシステムからロードしたファイルを使用して、Amazon Rekognition Image API オペレーションにイメージのバイトを渡す例を示します。イメージのバイトを Amazon Rekognition Image API オペレーションに渡すには Image 入力パラメータを使用します。`Image` 内で `Bytes` プロパティを指定し、base64 エンコードされたイメージのバイトを渡します。

`Bytes` 入力パラメータを使用して Amazon Rekognition API オペレーションに渡すイメージのバイトは、base64 でエンコードする必要があります。以下の例で使用する AWS SDK では、イメージを自動的に base64 でエンコードします。Amazon Rekognition API オペレーションを呼び出す前にイメージのバイトをエンコードする必要はありません。詳細については、「イメージの仕様」を参照してください。

次の DetectLabels に対する JSON リクエストの例では、ソースイメージのバイトを Bytes 入力パラメータで渡します。

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

次の例では AWS SDKs と を使用して AWS CLI を呼び出しますDetectLabels。DetectLabels オペレーションからのレスポンスの詳細については、「DetectLabels のレスポンス」を参照してください。

クライアント側の JavaScript の例については、「JavaScript を使用する」を参照してください。

ローカルイメージ内のラベルを検出するには

1. まだ実行していない場合:
   a. AmazonRekognitionFullAccess と AmazonS3ReadOnlyAccess のアクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。
   b. と AWS SDKsをインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 以下の例を使用して、DetectLabels オペレーションを呼び出します。

   Java

   次の Java の例では、ローカルファイルシステムからイメージをロードし、detectLabels AWS SDK オペレーションを使用してラベルを検出する方法を示します。photo の値は、イメージファイル (.jpg 形式または .png 形式) のパスとファイル名に変更します。

```
//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

次の AWS SDK for Python の例では、ローカルファイルシステムからイメージをロードし、detect_labels オペレーションを呼び出す方法を示します。photo の値は、イメージファイル (.jpg 形式または .png 形式) のパスとファイル名に変更します。

```
#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

次の の例では、ローカルファイルシステムからイメージをロードし、DetectLabels オペレーションを使用してラベルを検出する方法を示します。photo の値は、イメージファイル (.jpg 形式または .png 形式) のパスとファイル名に変更します。

```
//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

次の AWS SDK for PHP の例では、ローカルファイルシステムからイメージをロードし、DetectFaces API オペレーションを呼び出します。photo の値は、イメージファイル (.jpg 形式または .png 形式) のパスとファイル名に変更します。

```
<?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

この例では、入力イメージ内で検出されたラベルのリストを表示します。photo の値は、イメージファイル (.jpg 形式または .png 形式) のパスとファイル名に変更します。

```
#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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。完全な例を こちら を参照してください。

```
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.*;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectLabels {
    public static void main(String[] args) {
        final String usage = """
            Usage: <bucketName> <sourceImage>

            Where:
                bucketName - The name of the Amazon S3 bucket where the image is
 stored
                sourceImage - The name of the image file (for example,
 pic1.png).\s
            """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String bucketName = args[0] ;
        String sourceImage = args[1] ;
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        detectImageLabels(rekClient, bucketName, sourceImage);
        rekClient.close();
    }

    /**
     * Detects the labels in an image stored in an Amazon S3 bucket using the
Amazon Rekognition service.
     *
     * @param rekClient     the Amazon Rekognition client used to make the
detection request
     * @param bucketName    the name of the Amazon S3 bucket where the image is
stored
     * @param sourceImage   the name of the image file to be analyzed
     */
    public static void detectImageLabels(RekognitionClient rekClient, String
bucketName, String sourceImage) {
        try {
            S3Object s3ObjectTarget = S3Object.builder()
                    .bucket(bucketName)
                    .name(sourceImage)
                    .build();

            Image souImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .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 e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

JavaScript を使用する

次の JavaScript ウェブページの例では、ユーザーがイメージを選択し、そのイメージで検出された顔の推定年齢を表示することを許可します。推定年齢は DetectFaces を呼び出すと返されます。

選択したイメージは、JavaScript の `FileReader.readAsDataURL` 関数を使用してロードされます。これに伴って、イメージは base64 でエンコードされます。これは、HTML キャンバスでイメージを表示するのに役立ちます。ただし、イメージバイトを Amazon Rekognition Image オペレーションに渡す前にエンコードされていない状態にする必要があります。この例では、ロードされたイメージバイトをエンコードされていない状態にする方法を示しています。エンコードされたイメージバイトが有用でない場合は、代わりにロードされたイメージがエンコードされない `FileReader.readAsArrayBuffer` を使用します。つまり、最初にイメージバイトをエンコードされていない状態にする処理なしで、Amazon Rekognition Image オペレーションを呼び出すことができます。例については、readAsArrayBuffer を使用するを参照してください。

JavaScript の例を実行するには

1. ソースコード例をエディタにロードします。
2. Amazon Cognito identity プールの識別子を取得します。詳細については、「Amazon Cognito identity プールの識別を取得します。」を参照してください。
3. コード例の `AnonLog` 関数で、`IdentityPoolIdToUse` と `RegionToUse` は「Amazon Cognito identity プールの識別を取得します。」のステップ 9 で書き留めた値に変更します。
4. `DetectFaces` 関数の `RegionToUse` を、前のステップで使用した値に変更します。
5. ソースコード例を `.html` ファイルとして保存します。
6. ブラウザにファイルをロードします。
7. [参照...] ボタンを選択して、1 つ以上の顔が含まれたイメージを選択します。イメージ内で検出された顔ごとの推定年齢が含まれたテーブルが表示されます。

> **Note**
>
> 以下のコード例では、Amazon Cognito に含まれなくなった 2 つのスクリプトを使用しています。これらのファイルを取得するには、aws-cognito-sdk.min.js および amazon-cognito-identity.min.js のリンクをたどり、それぞれのテキストを個別の .js ファイルとして保存します。

JavaScript のコード例

次のコード例では JavaScript V2 を使用しています。JavaScript V3 の例については、AWS「Documentation SDK examples GitHub repository」の「例」を参照してください。

```
<!--
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("")[1]);

        } catch (e) {
          jpg = false;
        }
        if (jpg == false) {
```

```
          try {
            image = atob(e.target.result.split("")[1]);
          } catch (e) {
            alert("Not an image file Rekognition can process");
            return;
          }
        }
        //unencode image bytes for Rekognition DetectFaces API
        var length = image.length;
        imageBytes = new ArrayBuffer(length);
        var ua = new Uint8Array(imageBytes);
        for (var i = 0; i < length; i++) {
          ua[i] = image.charCodeAt(i);
        }
        //Call Rekognition
        DetectFaces(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>
```

readAsArrayBuffer を使用する

次のコードスニペットは、JavaScript V2 を使用したサンプルコード内の `ProcessImage` 関数の代替の実装です。`readAsArrayBuffer` を使用してイメージを読み込み、`DetectFaces` を呼び出し

ます。ロードしたファイルは readAsArrayBuffer で base64 エンコードされないため、Amazon Rekognition オペレーションを呼び出す前にイメージのバイトをエンコード解除する必要はありません。

```
//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);
  }
```

Amazon Cognito identity プールの識別を取得します。

わかりやすいように、この例では、匿名の Amazon Cognito identity プールを使用して、 Amazon Rekognition Image API に認証されていないアクセスを提供します。これは、お客様の二ーズに適している場合もあります。たとえば、認証されていないアクセスを使用して、ユーザーがサインアップする前に、無料または試用でウェブサイトにアクセスできるようにできます。認証されたアクセスを提供するには、Amazon Cognito ユーザープールを使用します。詳細については、Amazon Cognito ID ユーザープール を参照してください。

以下の手順では、認証されていない ID へのアクセスを有効にする ID プールを作成する方法と、コード例で必要な ID プールの ID を取得する方法を説明します。

ID プールの ID を取得するには

1. Amazon Cognito コンソールを開きます。
2. [Create new identity pool] を選択します。
3. [ID プール名*] に ID プールの名前を入力します。
4. [認証されていない ID] で、[認証されていない ID に対してアクセスを有効にする] を選択します。
5. [プールの作成] を選択します。
6. [詳細の表示] を選択して、認証されていない ID のロール名をメモします。
7. [Allow] (許可) を選択します。
8. [プラットフォーム] で [JavaScript] を選択します。
9. [AWS 認証情報の取得] で、コードスニペットに表示される `AWS.config.region` と `IdentityPooldId` の値をメモします。
10. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
11. ナビゲーションペインで Roles (ロール) を選択してください。
12. ステップ 6 で書き留めたロール名を選択します。
13. [アクセス許可] タブで、[ポリシーのアタッチ] を選択します。
14. [AmazonRekognitionReadOnlyAccess] を選択します。
15. [ポリシーをアタッチ] を選択します。

境界ボックスの表示

Amazon Rekognition Image オペレーションは、画像内で検出されたアイテムの境界ボックス座標を返すことができます。例えば、DetectFaces オペレーションは、イメージから検出された各顔の境界ボックス (BoundingBox) を返します。境界ボックス座標を使用して、検出されたアイテムの周囲にボックスを表示できます。たとえば、次の図は、顔を囲む境界ボックスを示しています。



BoundingBox には以下のプロパティがあります。

- Height – 画像全体の高さの比率としての境界ボックスの高さ。

- Left – 画像全体の幅の比率としての境界ボックスの左座標。
- Top – 画像全体の高さの比率としての境界ボックスの上端座標。
- Width – 画像全体の幅の比率としての境界ボックスの幅。

各 BoundingBox プロパティには、0 から 1 の値があります。各プロパティ値は全体の画像幅 (`Left` および `Width`) または高さ (`Height` および `Top`) の比率です。たとえば、入力イメージが 700 x 200 ピクセルの場合、境界ボックスの左上の座標は 350 x 50 ピクセルで、API は `Left` 値 0.5 (350/700) および `Top` 値 0.25 (50/200) を返します。

次の図は、各境界ボックスのプロパティがカバーする画像の範囲を示しています。

適切な場所に適切なサイズで境界ボックスを表示するには、ピクセル値を取得するには、BoundingBox 値にイメージの幅または高さ (必要な値に応じて) を掛ける必要があります。境界ボックスを表示するには、ピクセル値を使用します。たとえば、前の画像のピクセルのディメンションは、幅 608x 高さ 588 です。顔の境界ボックスの値は次のとおりです。

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

顔の境界ボックスの位置 (ピクセル) は次のように計算されます。

```
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
```

これらの値を使用して、面の周囲に境界ボックスを表示します。

> **Note**
>
> 画像の向きはさまざまです。アプリケーションのなかには、画像を回転させて補正方向で表示する必要がある場合があります。境界ボックスの座標は、画像の向きに影響されます。正しい位置に境界ボックスを表示する前に、座標を変換する必要があるかもしれません。詳細については、「イメージの向きおよび境界ボックス座標の取得」を参照してください。

次の例は、DetectFaces を呼び出すことによって検出された顔の周囲に境界ボックスを表示する方法を示しています。例では、画像が 0 度に向けられていると仮定する。この例では、Amazon S3 バケットから画像をダウンロードする方法も示しています。

境界ボックスを表示するには

1. まだ実行していない場合:

   a. AmazonRekognitionFullAccess と AmazonS3ReadOnlyAccess のアクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. 以下の例を使用して、DetectFaces オペレーションを呼び出します。

   Java

   bucket の値を、画像ファイルが保存される Amazon S3 バケットに変更します。photo の値は、イメージファイル (.jpg 形式または .png 形式) のファイル名に変更します。

```
//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

bucket の値を、画像ファイルが保存される Amazon S3 バケットに変更します。photo の値は、イメージファイル (.jpg 形式または .png 形式) のファイル名に変更します。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

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

def show_faces(photo, bucket):

    session = boto3.Session(profile_name='profile-name')
    client = session.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-name"
    photo = "photo-name"
    faces_count = show_faces(photo, bucket)
    print("faces detected: " + str(faces_count))

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

Java V2

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。完全な例を こちら を参照してください。

s3 AWS SDK Amazon S3 クライアントを参照し、rekClient AWS SDK Amazon Rekognition クライアントを参考することを注意ください。

```
//snippet-start:[rekognition.java2.detect_labels.import]
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.*;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.model.Attribute;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;
import software.amazon.awssdk.services.rekognition.model.DetectFacesRequest;
import software.amazon.awssdk.services.rekognition.model.DetectFacesResponse;
import software.amazon.awssdk.services.rekognition.model.FaceDetail;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
```

```
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
//snippet-end:[rekognition.java2.detect_labels.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DisplayFaces extends JPanel {

    static DetectFacesResponse result;
    static BufferedImage image;
    static int scale;

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

        final String usage = "\n" +
            "Usage: " +
            "   <sourceImage> <bucketName>\n\n" +
            "Where:\n" +
            "   sourceImage - The name of the image in an Amazon S3 bucket (for
 example, people.png). \n\n" +
            "   bucketName - The name of the Amazon S3 bucket (for example,
 amzn-s3-demo-bucket). \n\n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String sourceImage = args[0];
        String bucketName = args[1];
        Region region = Region.US_EAST_1;
        S3Client s3 = S3Client.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();
```

```
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        displayAllFaces(s3, rekClient, sourceImage, bucketName);
        s3.close();
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.display_faces.main]
    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);
            DisplayFaces panel = new DisplayFaces(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 DisplayFaces(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);
        }
    }
    // snippet-end:[rekognition.java2.display_faces.main]
}
```

イメージの向きおよび境界ボックス座標の取得

Amazon Rekognition Image を一般的に使用するアプリケーションは、Amazon Rekognition Image オペレーションによって検出されたイメージと、検出された顔の周囲のボックスを表示する必要があります。アプリケーションで正しくイメージを表示するには、イメージの向きを知り、必要に応じて修正する必要があります。この向きを修正することが必要な場合があります。一部の .jpg ファイルでは、イメージの向きは、イメージの Exchangeable イメージファイル形式 (Exif) メタデータに含まれています。

顔の周りにボックスを表示するには、顔の境界ボックスの座標が必要です。ボックスの向きが正しくない場合は、座標の調整が必要になることがあります。Amazon Rekognition イメージの顔検出オペレーションでは、検出された顔ごとに境界ボックスが返されます。しかし Exif メタデータがない.jpg ファイルの座標は推定されません。

下記の例は、イメージで検出された顔の境界ボックスの座標を示す際の例です。

この例の情報を使用して、イメージの向きが正しく、境界ボックスがアプリケーションで正しい位置に表示されるようにします。

イメージと境界ボックスを回転および表示するためのコードは、使用する言語と環境に依存するため、イメージと境界ボックスをコードで表示する方法や、Exif メタデータから向きの情報を取得する方法については説明しません。

イメージの向きの検索

アプリケーションで正しくイメージを表示するには、回転が必要になる場合があります。次のイメージの向きは 0 度で、正しく表示されています。



ただし、次のイメージは反時計回りに 90 度回転しています。これを正しく表示するには、イメージの向きを見つけ、その情報をコードで使用してイメージを 0 度に回転する必要があります。



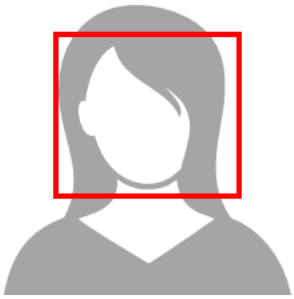
.jpg 形式の一部のイメージには、Exif メタデータに向きの情報が含まれています。使用可能な場合、イメージの Exif メタデータには向きが含まれています。Exif メタデータで、イメージの向きは `orientation` フィールドで見つかります。Amazon Rekognition は Exif メタデータ内のイメージの向きの情報の存在を認識しますが、その情報へのアクセスは提供しません。イメージで Exif メタデータにアクセスするには、サードパーティーのライブラリを使用するか、独自のコードを記述します。詳細については、「Exif バージョン 2.32」を参照してください。

イメージの向きがわかっている場合は、コードを記述して回転し、正しく表示できます。

境界ボックスの表示

イメージの顔を分析する Amazon Rekognition Image オペレーションも、顔を囲む境界ボックスの座標を返します。詳細については、「BoundingBox」を参照してください。

アプリケーションの次のイメージに示すボックスに類似した境界ボックスを顔の周囲に表示するには、コードで境界ボックスの座標を使用します。オペレーションで返される境界ボックスの座標は、イメージの向きを反映しています。イメージを回転して正しく表示する必要がある場合は、境界ボックスの座標の変換が必要になる場合があります。



向きの情報が Exif メタデータに存在するときに境界ボックスを表示する

イメージの向きが Exif メタデータに含まれている場合、Amazon Rekognition Image オペレーションで以下が実行されます。

- オペレーションのレスポンスの向き修正フィールドで null が返されます。イメージを回転するには、コードの Exif メタデータに含まれている向きを使用します。
- すでに向きが設定されている境界ボックスの座標を 0 度に戻します。正しい位置に境界ボックスを表示するには、返された座標を使用します。それらを変換する必要はありません。

例: イメージの向きおよびイメージの境界ボックス座標の取得

以下の例では、AWS SDK を使用して Exif イメージの推定された向きを取得し、RecognizeCelebrities オペレーションで検出された有名人の境界ボックス座標を変換する方法を示します。

> **Note**
>
> OrientationCorrection フィールドを使用した画像方向の見積もりは、2021 年 8 月をもってサポートを終了しています。API レスポンスに含まれるこのフィールドの戻り値は、常に NULL になります。

Java

この例では、ローカルファイルシステムからイメージをロードし、RecognizeCelebrities オペレーションを呼び出して、イメージの高さと幅を判断してから、回転したイメージの顔の境界ボックスの座標を計算します。この例では、Exif メタデータに保存される向きの情報を処理する方法は示しません。

関数 main で、photo の値は、ローカルに保存されているイメージ (.png または .jpg 形式) の名前とパスに置き換えます。

```
//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

この例では、PIL/Pillow イメージライブラリを使用してイメージの幅と高さを取得します。詳細については、「Pillow」を参照してください。この例では、exif メタデータを保存し、アプリケーションの他の場所で利用できるようにします。

関数 main で、photo の値は、ローカルに保存されているイメージ (.png または .jpg 形式) の名前とパスに置き換えます。

```
#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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
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.RecognizeCelebritiesRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import
 software.amazon.awssdk.services.rekognition.model.RecognizeCelebritiesResponse;
import software.amazon.awssdk.services.rekognition.model.Celebrity;
import software.amazon.awssdk.services.rekognition.model.ComparedFace;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.*;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class RotateImage {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <sourceImage>

                Where:
                   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png).\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String sourceImage = args[0];
        Region region = Region.US_WEST_2;
```

```
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Locating celebrities in " + sourceImage);
        recognizeAllCelebrities(rekClient, sourceImage);
        rekClient.close();
    }

    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();
            }
            case "ROTATE_90" -> {
                left = imageHeight * (1 - (box.top() + box.height()));
                top = imageWidth * box.left();
            }
            case "ROTATE_180" -> {
                left = imageWidth - (imageWidth * (box.left() + box.width()));
                top = imageHeight * (1 - (box.top() + box.height()));
            }
            case "ROTATE_270" -> {
                left = imageHeight * box.top();
                top = imageWidth * (1 - box.left() - box.width());
            }
            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()));
        }
    }
```

保存済み動画分析での作業

Amazon Rekognition Video は、ビデオの分析に使用できる API です。Amazon Rekognition Video では、Amazon Simple Storage Service (Amazon S3) バケットに保存されているビデオ内のラベル、顔、人物、有名人、およびアダルト (暗示的および明示的) コンテンツを検出できます。Amazon Rekognition Video は、メディア/エンターテインメントや公共安全などのカテゴリで利用できます。これまでのような物体や人物が撮影されたビデオの人間によるスキャンでは、かなりの時間がかかるだけでなく、エラーも頻繁に発生していました。Amazon Rekognition Video を使うことで、ビデオ全体を通しての項目の検出や、項目が登場したタイミングの検出を自動化することができます。

このセクションでは、Amazon Rekognition Video で実行できる分析のタイプ、API の概要、および Amazon Rekognition Video の使用例について説明します。

トピック

- 分析のタイプ
- Amazon Rekognition Video API の概要
- Amazon Rekognition Video オペレーションを呼び出す
- Amazon Rekognition Video の設定
- Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)
- を使用したビデオの分析 AWS Command Line Interface
- リファレンス: ビデオ分析結果の通知
- Amazon Rekognition Video のトラブルシューティング

分析のタイプ

Amazon Rekognition Video を使用して、以下の情報についてビデオを分析できます。

- ビデオセグメント
- ラベル
- 暗示的および明示的なアダルトコンテンツ
- [テキスト]

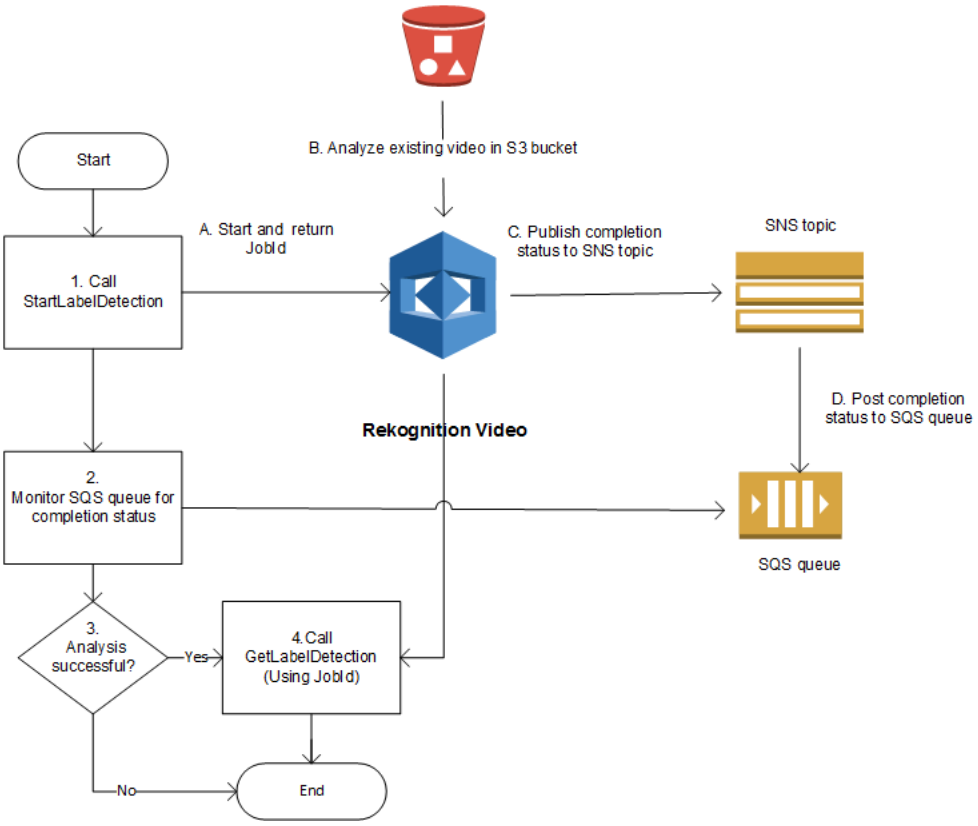
- 有名人
- 顔
- 人員

詳細については、「Amazon Rekognition の仕組み」を参照してください。

Amazon Rekognition Video API の概要

Amazon Rekognition Video は、Amazon S3 バケットに保存されたビデオを処理します。その設計パターンは、複数のオペレーションの非同期セットとなっており、StartLabelDetection などの `Start` オペレーションを呼び出すと、ビデオの分析が開始されます。リクエストの完了ステータスは、Amazon Simple Notification Service (Amazon SNS) トピックに発行されます。Amazon SNS トピックから完了ステータスを取得するには、Amazon Simple Queue Service (Amazon SQS) キューまたは AWS Lambda 関数を使用できます。完了ステータスを取得したら、GetLabelDetection などの `Get` オペレーションを呼び出してリクエストの結果を取得します。

次の図表は、Amazon S3 バケットに保存されているビデオ内のラベルを検出するプロセスを示しています。この図では、Amazon SQS キューが Amazon SNS トピックから完了ステータスを取得しています。または、 AWS Lambda 関数を使用することもできます。



このプロセスは、他の Amazon Rekognition Video オペレーションでも同じです。以下の表は、非ストレージの各 Amazon Rekognition オペレーションの `Start` オペレーションと `Get` オペレーションを一覧したものです。

検出	Start オペレーション	Get オペレーション
ビデオセグメント	StartSegmentDetection	GetSegmentDetection
ラベル	StartLabelDetection	GetLabelDetection
明示的および暗示的なアダルトコンテンツ	StartContentModeration	GetContentModeration
テキスト	StartTextDetection	GetTextDetection
有名人	StartCelebrityRecognition	GetCelebrityRecognition
顔	StartFaceDetection	GetFaceDetection
人員	StartPersonTracking	GetPersonTracking

`Get` 以外の `GetCelebrityRecognition` オペレーションの場合、Amazon Rekognition Video は入力ビデオ全体でエンティティ検出時の追跡情報を返します。

Amazon Rekognition Video の使用の詳細については、Amazon Rekognition Video オペレーションを呼び出す を参照してください。Amazon SQS を使用してビデオ分析を行う例については、Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK) を参照してください。 AWS CLI 例については、「」を参照してくださいを使用したビデオの分析 AWS Command Line Interface。

ビデオの形式とストレージ

Amazon Rekognition のオペレーションでは、Amazon S3 バケットに保存されているビデオを分析できます。ビデオ分析オペレーションの全制限のリストについては、「ガイドラインとクォータ」を参照してください。

ビデオは H.264 コーデックでエンコードする必要があります。サポートされているファイル形式は MPEG-4 および MOV です。

コーデックとは、データ圧縮による高速な配信と、受信データからオリジナル形式への解凍を可能にするソフトウェアまたはハードウェアです。H.264 コーデックは、ビデオコンテンツの記録、圧縮、配信に広く使われています。1 つのビデオ形式には 1 つ以上のコーデックを含めることができます。MOV または MPEG-4 形式のビデオファイルが Amazon Rekognition Video で動作しない場合、H.264 コーデックがビデオのエンコードに使われているかどうかを確認してください。

オーディオデータを分析する Amazon Rekognition Video API は、AAC オーディオコーデックのみをサポートします。

保存済みビデオの最大ファイルサイズは 10 GB です。

人物の検索

コレクションに保存された顔のメタデータを使ってビデオ内の人物を検索できます。例えば、アーカイブされたビデオから特定の人物や複数の人物を検索できます。IndexFaces オペレーションを使用して、ソースイメージの顔のメタデータをコレクションに保存します。次に StartFaceSearch を使用して、コレクション内の顔の検索を非同期で開始します。検索結果の取得には、GetFaceSearch を使用します。詳細については、「保存したビデオでの顔の検索」を参照してください。人物の検索は、代表的なストレージベースの Amazon Rekognition オペレーションです。詳細については、「ストレージ型 API オペレーション」を参照してください。

ストリーミングビデオ内の人物を検索することもできます。詳細については、「ストリーミングビデオイベントの操作」を参照してください。

Amazon Rekognition Video オペレーションを呼び出す

Amazon Rekognition Video は、Amazon Simple Storage Service (Amazon S3) バケットに保存されているビデオの分析に使用できる非同期 API です。StartPersonTracking などの Amazon Rekognition Video の `Start` オペレーションを呼び出して、ビデオの分析を開始します。Amazon Rekognition Video は、分析リクエストの完了ステータスを Amazon Simple Notification Service (Amazon SNS) トピックに発行します。Amazon Simple Queue Service (Amazon SQS) キューまたは AWS Lambda 関数を使用して、Amazon SNS トピックからビデオ分析リクエストの完了ステータスを取得できます。最後に、GetPersonTracking などの Amazon Rekognition `Get` オペレーションを呼び出して、ビデオ分析リクエストの結果を取得します。

以下のセクションでは、ラベル検出オペレーションを使用して、Amazon S3 バケットに保存されているビデオ内のラベル (オブジェクト、イベント、概念、およびアクティビティ) を Amazon Rekognition Video が検出する方法を説明します。この方法は、他の Amazon Rekognition Video オペレーション (StartFaceDetection や StartPersonTracking) でも同様に機能します。この例 Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK) では、Amazon SQS

キューを使用して、Amazon SNS トピックから完了ステータスを取得し、ビデオを分析する方法を説明しています。また、人物の動線の検出 など、他の Amazon Rekognition Video の例のベースとして使用することもできます。 AWS CLI 例については、「」を参照してください[を使用したビデオの分析 AWS Command Line Interface]。

トピック

- ビデオ分析のスタート
- Amazon Rekognition Video 分析リクエストの完了ステータスの取得
- Amazon Rekognition Video の分析結果を取得する

ビデオ分析のスタート

StartLabelDetection を呼び出して、Amazon Rekognition Video のラベル検出リクエストを開始します。以下は、`StartLabelDetection` により渡される JSON リクエストの例を示しています。

```
{
    "Video": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-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"
}
```

入力パラメータ `Video` はビデオファイル名を提供し、`NotificationChannel` から取得する Amazon S3 バケットには、ビデオ分析リクエストが終了したときに Amazon Rekognition Video が通知する Amazon SNS トピックの Amazon リソースネーム (ARN) が含まれます。Amazon SNS トピックは、呼び出し先の エンドポイントと同じ AWS リージョン内に存在する必要があります。`NotificationChannel` には、Amazon Rekognition Video に Amazon SNS トピックの発行を許可するためのロールの ARN も含まれています。IAM サービスロールを作成して、Amazon Rekognition に Amazon SNS トピックへの発行許可を付与します。詳細については、「Amazon Rekognition Video の設定」を参照してください。

JobTag というオプションの入力パラメータを指定して、Amazon SNS トピックに発行された完了ステータスのジョブを特定することもできます。

ClientRequestToken というべき等トークンをオプションで設定すると、分析ジョブが誤って重複するのを防ぐことができます。ClientRequestToken の値を指定すると、Start オペレーションは、StartLabelDetection などの Start オペレーションに対する複数かつ同一の呼び出しに共通する JobId を返します。ClientRequestToken トークンの有効期間は 7 日です。7 日後に再利用することができます。トークンの有効期間中にトークンを再利用すると、以下のことが発生します。

- 同じ Start オペレーション、同じ入力パラメータでトークンを再利用すると、同じ JobId が返されます。ジョブは再度実行されず、Amazon Rekognition Video は、登録されている Amazon SNS トピックに完了ステータスを送信しません。
- 同じ Start オペレーションで、入力パラメータを少し変更してトークンを再利用すると、IdempotentParameterMismatchException (HTTP ステータスコード: 400) 例外を受け取ります。
- Amazon Rekognition から予期しない結果が発生する可能性があるため、トークンを異なる Start オペレーションで再利用しないでください。

StartLabelDetection オペレーションに対する応答は、ジョブ識別子 (JobId) です。Amazon Rekognition Video が完了ステータスを Amazon SNS トピックに発行したら、JobId を使用してリクエストを追跡し、分析結果を取得します。例:

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

同時に開始するジョブが多すぎると、StartLabelDetection の呼び出しによって、同時に実行されるジョブの数が Amazon Rekognition サービスの制限を下回るまで LimitExceededException (HTTP ステータスコード: 400)が発生します。

アクティビティのバーストにより LimitExceededException 例外が発生した場合は、 Amazon SQS キューを使用して受信リクエストを管理することを検討してください。同時リクエストの平均数が Amazon SQS キューで管理できず、まだLimitExceededException例外が発生している場合は、 AWS サポートにお問い合わせください。

Amazon Rekognition Video 分析リクエストの完了ステータスの取得

Amazon Rekognition Video は、登録されている Amazon SNS トピックに分析完了の通知を送信します。通知には、ジョブ識別子およびオペレーション完了ステータスが JSON 文字列で含まれていま

す。ビデオ分析リクエストが正常に実行された場合は SUCCEEDED ステータスとなります。例えば、次の結果はラベル検出ジョブが正常に処理されたことを示しています。

```
{
    "JobId": "270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1nnnnnnnnnnnn",
    "Status": "SUCCEEDED",
    "API": "StartLabelDetection",
    "JobTag": "DetectingLabels",
    "Timestamp": 1510865364756,
    "Video": {
        "S3ObjectName": "video.mp4",
        "S3Bucket": "amzn-s3-demo-bucket"
    }
}
```

詳細については、「リファレンス: ビデオ分析結果の通知」を参照してください。

Amazon Rekognition Video が Amazon SNS トピックに発行したステータス情報を取得するには、次のいずれかのオプションを使用します。

- AWS Lambda – Amazon SNS トピックに書き込む AWS Lambda 関数をサブスクライブできます。この関数は、Amazon Rekognition によってリクエスト完了が Amazon SNS トピックに通知されると呼び出されます。ビデオ分析リクエストの結果をサーバー側のコードで処理する場合は Lambda 関数を使用します。例えば、クライアントアプリケーションに情報を返す前に、ビデオに注釈を付けたり、ビデオコンテンツに関するレポートを作成したりする場合があります。また、Amazon Rekognition API が大量のデータを返す可能性があるため、ビデオの容量が大きい場合もサーバー側での処理をお勧めします。
- [Amazon Simple Queue Service]— Amazon SQS キューを Amazon SNS トピックにサブスクライブできます。次にこの Amazon SQS キューをポーリングすることで、リクエストの完了時に Amazon Rekognition によって発行される完了ステータスを取得できます。詳細については、「Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)」を参照してください。Amazon SQS キューは、Amazon Rekognition Video のオペレーションをクライアントアプリケーションからのみ呼び出す場合に使用します。

> **Important**
>
> リクエストの完了ステータスを取得するために、Amazon Rekognition Video の Get オペレーションを繰り返し呼び出すことは推奨されません。これは、実行されるリクエストが多すぎると、Amazon Rekognition Videoによる Get オペレーションの調整が行われるためで

す。複数のビデオを同時処理する場合は、Amazon Rekognition Video をポーリングして各ビデオのステータスを個別に取得するよりも、1 つの SQS キューの完了通知をモニタリングするほうがシンプルかつ効率的です。

Amazon Rekognition Video の分析結果を取得する

ビデオ分析リクエストの結果を取得するには、まず Amazon SNS トピックから取得された完了ステータスが SUCCEEDED であることを確認します。次に GetLabelDetection を呼び出し、StartLabelDetection から返された JobId の値を渡します。リクエストの JSON は次の例のようになります。

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

JobId はビデオ分析オペレーションの識別子です。ビデオの分析で大量のデータが生成される場合に備えて、1 回の Get オペレーションで返される結果の最大数を MaxResults で指定できます。MaxResults のデフォルト値は 1000 です。1000 より大きい値を指定した場合、最大 1000 件の結果が返されます。オペレーションが結果セット全体を返さない場合、次のページ用のページ分割トークンがオペレーションの応答で返されます。直前の Get リクエストでページ分割トークンが返された場合は、そのトークンと NextToken を使って次の結果ページを取得できます。

> **Note**
>
> Amazon Rekognition はビデオ分析オペレーションの結果を 7 日間保持します。この期間を過ぎると分析結果を取得できなくなります。

GetLabelDetection オペレーションの応答の JSON は次のようになります。

```
{
    "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
    }
}
```

GetLabelDetection および GetContentModeration オペレーションでは、分析結果をタイムスタンプまたはラベル名でソートできます。また、結果をビデオセグメント別またはタイムスタンプ別に集計することもできます。

結果は、検出時間 (ビデオの開始時間からミリ秒単位) または検出されたエンティティ (オブジェクト、顔、有名人、モデレーションラベル、人物) のアルファベット順で並べ替えることができます。時間で並べ替える場合は、SortBy 入力パラメータの値を TIMESTAMP に設定します。SortBy が指定されていない場合のデフォルトの動作は、時間による並べ替えです。上記の例は時間で並べ替えられています。エンティティで並べ替えるには、実行するオペレーションにとって適切な値を SortBy 入力パラメータで使用します。例えば、GetLabelDetection への呼び出しで検出されたラベルで並べ替える、NAME 値を使用します。

結果をタイムスタンプ別に集計するには、AggregateBy パラメータの値を TIMESTAMPS に設定します。ビデオセグメントでは、AggregateBy の値を SEGMENTS に設定します。SEGMENTS 集計

モードは、ラベルを経時的に集計し、TIMESTAMPS は、2 FPS サンプリングとフレームごとの出力を使って、ラベルが検出されたタイムスタンプを表示します (注: 現在のサンプリングレートは変更される可能性があるため、現在のサンプリングレートを基に予測しないようにしてください)。値が指定されていない場合、デフォルトの集計方法は TIMESTAMPS です。

Amazon Rekognition Video の設定

保存したビデオで Amazon Rekognition Video API を使用するには、Amazon SNS トピックにアクセスできるようにユーザーと IAM サービスロールを設定する必要があります。また、Amazon SQS キューを Amazon SNS トピックにサブスクライブする必要があります。

> **Note**
>
> これらの手順を使用して Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK) の例を設定する場合は、ステップ 3、4、5、6 を実行する必要はありません。この例には、Amazon SNS トピックと Amazon SQS キューを作成して設定するコードが含まれています。

このセクションの例では、Amazon Rekognition Video が複数のトピックにアクセスできるようにする手順を使用して、新しい Amazon SNS トピックを作成します。既存の Amazon SNS トピックを使用する場合は、ステップ 3 の 既存の Amazon SNS トピックへのアクセスをataeruする を使用します。

Amazon Rekognition Video を設定するには

1. Amazon Rekognition Video にアクセスする AWS ようにアカウントを設定します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。
2. 必要な AWS SDK をインストールして設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。
3. このデベロッパーガイドのコード例を実行するには、選択したユーザーがプログラムによるアクセス権限を持っていることを確認してください。詳細については「プログラマチックアクセス権を付与する」を参照してください。

   また、ユーザーには少なくとも以下の権限が必要です。

   - AmazonSQSFullAccess
   - AmazonRekognitionFullAccess

- AmazonS3FullAccess
- AmazonSNSFullAccess

IAM アイデンティティセンターを使用して認証を行う場合は、ロールの許可セットにアクセス権限を追加し、それ以外の場合は IAM ロールにアクセス権限を追加します。

4. [Amazon SNS コンソール] を使用して [Amazon SNS トピック] を作成します。トピック名の先頭に AmazonRekognition を追加します。その Amazon リソースネーム (ARN) をメモします。このトピックが、使用している AWS エンドポイントと同じリージョンにあることを確認します。
5. [Amazon SQS コンソール] を使用して、[Amazon SQS スタンダードキュー] を作成します。キューの ARN を書き留めます。
6. ステップ 3 で作成したトピックにキューをサブスクライブします。
7. [Amazon SNS トピックに、Amazon SQS キューへメッセージを送信する許可を与えます]。
8. IAM サービスロールを作成して、Amazon Rekognition Video に Amazon SNS トピックへのアクセスを許可します。このサービスロールの Amazon リソースネーム (ARN) をメモします。詳細については、「複数の Amazon SNS トピックへのアクセスを許可する」を参照してください。
9. アカウントの安全を確保するには、Rekognition のアクセス範囲を、使用しているリソースのみに制限する必要があります。これは、IAM サービスロールに信頼ポリシーをアタッチすることで実行できます。これを行う方法については、「サービス間の混乱した代理の防止」を参照してください。
10. ステップ 1 で作成したユーザーに次のインラインポリシーを追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "MySid",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:Service role ARN from step 7"
        }
    ]
}
```

インラインポリシーに任意の名前を付けます。

11. カスタマーマネージド AWS Key Management Service キーを使用して Amazon S3 バケット内の動画を暗号化する場合は、ステップ 7 で作成したサービスロールに動画の復号を許可するアクセス許可をキーに追加します。サービスロールには少なくとも、kms:GenerateDataKey および kms:Decrypt アクションのアクセス権限が必要です。例:

```
{
    "Sid": "Decrypt only",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/user from step 1"
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*"
}
```

詳細については、「私の Amazon S3 バケットにはカスタムの AWS KMS キーを使用したデフォルト暗号化が設定されています。ユーザーがそのバケットでダウンロードやアップロードを行えるようにするにはどうすればいいですか?」および「Protecting Data Using Server-Side Encryption with KMS keys Stored in AWS Key Management Service (SSE-KMS)」を参照してください。

12. これで、「Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)」と「を使用したビデオの分析 AWS Command Line Interface」の例を実行できます。

複数の Amazon SNS トピックへのアクセスを許可する

IAM サービスロールを使用し、作成したAmazon SNS トピックへのアクセスを Amazon Rekognition Video に付与します。IAM は、Amazon Rekognition Video サービスロール作成用の [Rekognition] ユースケースを提供します。

Amazon Rekognition Videoは、AmazonRekognitionServiceRole 許可ポリシーを使用し、トピック名の前に [AmazonRekognition] を付けることで、複数の Amazon SNS トピックへのアクセスを与えることができます (例: AmazonRekognitionMyTopicName)。

Amazon Rekognition Video を複数の Amazon SNS トピックにアクセスできるようにするには

1. [IAM サービスロールを作成します]。次の情報を使用して、IAM サービスロールを作成します。

1. サービス名の Rekognition を選択します。
2. サービスロールのユースケースの Rekognition を選択します。AmazonRekognitionServiceRole アクセス権限ポリシーが表示されます。[AmazonRekognitionServiceRole] は、[AmazonRekognition] をプレフィックスとする Amazon SNS トピックへのアクセス権を Amazon Rekognition Video に付与します。
3. サービスロールに任意の名前を付けます。

2. サービスロールの ARN をメモしておきます。これは、ビデオ分析のオペレーションを開始する際に必要です。

既存の Amazon SNS トピックへのアクセスをataeruする

既存の Amazon SNS トピックへのアクセスを Amazon Rekognition Video に付与するアクセス許可ポリシーを作成できます。

Amazon Rekognition Video を Amazon SNS トピックにアクセスできるようにするには

1. [IAM JSON ポリシーエディターを使用して新しい許可ポリシーを作成し]、次のポリシーを使用します。`topicarn` を、目的の Amazon SNS トピックの Amazon リソースネーム (ARN) に置き換えます。

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

2. IAM サービスロールを作成する または、既存の IAM サービスロールを更新します。次の情報を使用して、IAM サービスロールを作成します。
    1. サービス名の Rekognition を選択します。
    2. サービスロールのユースケースの Rekognition を選択します。

3. 手順 1 で作成したアクセス権ポリシーを添付します。

3. サービスロールの ARN をメモしておきます。これは、ビデオ分析のオペレーションを開始する際に必要です。

Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)

この手順では、Amazon Rekognition Video ラベル検出オペレーション、Amazon S3 バケットに保存されたビデオ、および Amazon SNS トピックを使用して、ビデオ内のラベルを検出する方法について説明します。また、Amazon SQS キューを使用して Amazon SNS トピックから完了ステータスを取得する方法についても説明します。詳細については、「Amazon Rekognition Video オペレーションを呼び出す」を参照してください。Amazon SQS キューの使用に制限されるわけではありません。たとえば、 AWS Lambda 関数を使用して完了ステータスを取得できます。詳細については、「Amazon SNS 通知を使用した Lambda 関数の呼び出し」を参照してください。

手順に含まれているコード例は、以下を実行する方法を示しています。

1. Amazon SNS トピックを作成します。
2. Amazon SQS キュー を作成します。
3. Amazon Rekognition Video に、ビデオ分析オペレーションの完了ステータスを Amazon SNS トピックに発行するアクセス許可を与えます。
4. Amazon SQS キューを Amazon SNS トピックへサブスクライブします。
5. StartLabelDetection を呼び出してビデオ分析リクエストを開始します。
6. 完了ステータスを Amazon SQS キューから取得します。この例では、`StartLabelDetection` で返されたジョブ識別子 (`JobId`) を追跡し、一致するジョブ識別子の結果だけを完了ステータスから読み取ります。他のアプリケーションで同じキューやトピックを使用している場合、この点を考慮することが重要です。わかりやすいように、この例では一致しないジョブが削除されます。これらを Amazon SQS デッドレターキューに追加して詳しく調査することを検討してください。
7. GetLabelDetection を呼び出して、ビデオ分析結果を取得し表示します。

前提条件

この手順のコード例は Java と Python で提供されています。適切な AWS SDK をインストールする必要があります。詳細については、「Amazon Rekognition の開始方法」を参照してください。使用

する AWS アカウントには、Amazon Rekognition API に対するアクセス許可が必要です。詳細については、[Amazon Rekognition で定義されるアクション] を参照してください。

ビデオ内のラベルを検出するには

1. Amazon Rekognition Video へのユーザーアクセスを設定し、Amazon SNS への Amazon Rekognition Video アクセスを設定します。詳細については、「Amazon Rekognition Video の設定」を参照してください。コード例によって Amazon SNS トピックと Amazon SQS キューが作成されて設定されるため、ステップ 3、4、5、6 を実行する必要はありません。
2. Amazon S3 バケットに MOV または MPEG- 4 形式のビデオファイルをアップロードします。テストの場合、長さが 30 秒以内のビデオをアップロードしてください。

   手順については、[Amazon Simple Storage Service ユーザーガイド] の [Amazon S3 へのオブジェクトのアップロード] を参照してください。

3. 次のコード例を使用して、ビデオ内のラベルを検出します。

   Java

   `main` 関数内:

   - `roleArn` を、Amazon Rekognition Video を設定するには のステップ 7 で作成した IAM サービスロールの ARN に置き換えます。
   - `amzn-s3-demo-bucket` と `video` の値を、ステップ 2 で指定したバケット名とビデオファイル名に置き換えます。

```
//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(snsTopic
                    );


          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

`main` 関数内:

- `roleArn` を、Amazon Rekognition Video を設定するには のステップ 7 で作成した IAM サービスロールの ARN に置き換えます。
- `amzn-s3-demo-bucket` と `video` の値を、ステップ 2 で指定したバケット名とビデオファイル名に置き換えます。

- Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。
- 設定パラメータにフィルター条件を含めることもできます。例えば、目的の値のリストと共に LabelsInclusionFilter または LabelsExclusionFilter を使用できます。以下のコードでは、Features と Settings セクションのコメントを外して独自の値を指定することで、返される結果を興味のあるラベルだけに制限できます。
- GetLabelDetection の呼び出しでは、SortBy および AggregateBy 引数の値を指定できます。時間で並べ替える場合は、SortBy 入力パラメータの値を TIMESTAMP に設定します。エンティティで並べ替えるには、実行するオペレーションにとって適切な値を SortBy 入力パラメータで使用します。結果をタイムスタンプ別に集計するには、AggregateBy パラメータの値を TIMESTAMPS に設定します。ビデオセグメント別に集計するには、SEGMENTS を使用します。

```
## 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 = ''

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

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

    def __init__(self, role, bucket, video, client, rek, sqs, sns):
        self.roleArn = role
        self.bucket = bucket
        self.video = video
        self.client = client
```

```
        self.rek = rek
        self.sqs = sqs
        self.sns = sns

    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},
                                                  MinConfidence=90,
                                                  # Filtration options,
uncomment and add desired labels to filter returned labels
                                                  # Features=['GENERAL_LABELS'],
                                                  # Settings={
                                                  # 'GeneralLabels': {
                                                  # 'LabelInclusionFilters':
['Clothing']
                                                  # }}
                                                   )

        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',
                                                    AggregateBy="TIMESTAMPS")

            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("Aliases:")
                for alias in label['Aliases']:
                    print("   " + alias['Name'])

                print("Categories:")
                for category in label['Categories']:
                    print("   " + category['Name'])
                print("----------")
                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 = 'role-arn'
    bucket = 'bucket-name'
    video = 'video-name'

    session = boto3.Session(profile_name='profile-name')
    client = session.client('rekognition')
    rek = boto3.client('rekognition')
    sqs = boto3.client('sqs')
    sns = boto3.client('sns')

    analyzer = VideoDetect(roleArn, bucket, video, client, rek, sqs, sns)
    analyzer.CreateTopicandQueue()

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

    analyzer.DeleteTopicandQueue()

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

Node.Js

次のサンプルコードでは

- REGION の値を、アカウントのオペレーティングリージョンの名前に置き換えます。
- amzn-s3-demo-bucket の値を、ビデオファイルが保存されているAmazon S3 バケットの名前に置き換えます。
- videoName の値を、Amazon S3 バケット内のビデオファイル名に置き換えます。
- Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。
- roleArn を、Amazon Rekognition Video を設定するには のステップ 7 で作成した IAM サービスロールの ARN に置き換えます。

```
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";
import {fromIni} from '@aws-sdk/credential-providers';

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

// 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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例は [こちら] です。

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.LabelDetectionSortBy;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.LabelDetection;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.Instance;
import software.amazon.awssdk.services.rekognition.model.Parent;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_detect.import]
```

```
/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class VideoDetect {

 private static String startJobId ="";
 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <bucket> <video> <queueUrl> <topicArn> <roleArn>\n\n" +
         "Where:\n" +
         "   bucket - The name of the bucket in which the video is located (for
 example, (for example, amzn-s3-demo-bucket). \n\n"+
         "   video - The name of the video (for example, people.mp4). \n\n" +
         "   queueUrl- The URL of a SQS queue. \n\n" +
         "   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic. \n\n" +
         "   roleArn - The ARN of the AWS Identity and Access Management (IAM)
 role to use. \n\n" ;

     if (args.length != 5) {
         System.out.println(usage);
         System.exit(1);
     }

     String bucket = args[0];
     String video = args[1];
     String queueUrl = args[2];
     String topicArn = args[3];
     String roleArn = args[4];
     Region region = Region.US_WEST_2;
     RekognitionClient rekClient = RekognitionClient.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
         .build();

     SqsClient sqs = SqsClient.builder()
```

```
        .region(Region.US_WEST_2)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

    NotificationChannel channel = NotificationChannel.builder()
        .snsTopicArn(topicArn)
        .roleArn(roleArn)
        .build();

    startLabels(rekClient, channel, bucket, video);
    getLabelJob(rekClient, sqs,  queueUrl);
    System.out.println("This example is done!");
    sqs.close();
    rekClient.close();
}

// snippet-start:[rekognition.java2.recognize_video_detect.main]
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);
    }
}
// snippet-end:[rekognition.java2.recognize_video_detect.main]
}
```

4. コードをビルドして実行します。オペレーションの完了までに時間がかかる場合があります。完了すると、ビデオ内で検出されたラベルのリストが表示されます。詳細については、「ビデオ内のラベルの検出」を参照してください。

を使用したビデオの分析 AWS Command Line Interface

AWS Command Line Interface （ AWS CLI) を使用して Amazon Rekognition Video オペレーションを呼び出すことができます。設計パターンは、 AWS SDK for Java または他の AWS SDK で Amazon Rekognition Video API を使用するのと同じです。 SDKs 詳細については、「Amazon Rekognition Video API の概要」を参照してください。次の手順は、 を使用してビデオ内のラベル AWS CLI を検出する方法を示しています。

ビデオ内のラベルの検出を開始するには、`start-label-detection` を呼び出します。Amazon Rekognition によるビデオの分析が完了すると、`--notification-channel` の `start-label-detection` パラメータで指定されている Amazon SNS トピックに完了ステータスが送信されます。Amazon Simple Queue Service (Amazon SQS) キューを Amazon SNS トピックにサブスクライブすることで、完了ステータスを取得できます。次に [receive-message] をポーリングして Amazon SQS キューから完了ステータスを取得します。

`StartLabelDetection` を呼び出すときは、`LabelsInclusionFilter` および/または `LabelsExclusionFilter` 引数にフィルター引数を指定して結果をフィルタリングできます。詳細については、「ビデオ内のラベルの検出」を参照してください。

完了ステータスの通知は、`receive-message` 応答内の JSON 構造です。この JSON を応答から抽出する必要があります。完了ステータス JSON については、「リファレンス: ビデオ分析結果の通知」を参照してください。JSON 完了ステータスの `Status` フィールドの値が `SUCCEEDED` の場合、`get-label-detection` を呼び出すことでビデオ分析リクエストの結果を取得できます。`GetLabelDetection` を呼び出すときは、`SortBy` および `AggregateBy` 引数を使用して返された結果をソートして集計できます。

以下の手順には、Amazon SQS キューをポーリングするコードは含まれていません。また、Amazon SQS キューから返される JSON を解析するコードも含まれていません。Java での例については、「Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)」を参照してください。

前提条件

この手順を実行するには、 AWS CLI がインストールされている必要があります。詳細については、「Amazon Rekognition の開始方法」を参照してください。使用する AWS アカウントには、Amazon Rekognition API に対するアクセス許可が必要です。詳細については、[Amazon Rekognition によって定義されるアクション] を参照してください。

Amazon Rekognition Video を設定してビデオをアップロードするには

1. Amazon Rekognition Video へのユーザーアクセスを設定し、Amazon SNS への Amazon Rekognition Video アクセスを設定します。詳細については、「Amazon Rekognition Video の設定」を参照してください。
2. S3 バケットに MOV あるいは MPEG-4 形式のビデオファイルをアップロードします。開発やテストには、長さが 30 秒以内の短いビデオを使うことをお勧めします。

   手順については、[Amazon Simple Storage Service ユーザーガイド] の [Amazon S3 へのオブジェクトのアップロード] を参照してください。

ビデオ内のラベルを検出するには

1. 次の AWS CLI コマンドを実行して、ビデオ内のラベルの検出を開始します。

```
aws rekognition start-label-detection --video '{"S3Object":{"Bucket":"amzn-s3-demo-bucket","Name":"video-name"}}' \
 --notification-channel '{"SNSTopicArn":"TopicARN","RoleArn":"RoleARN"}' \
--region region-name  \
--features GENERAL_LABELS \
--profile profile-name \
--settings "{"GeneralLabels":{"LabelInclusionFilters":["Car"]}}
```

   以下の値を更新します。

   - amzn-s3-demo-bucket と videofile を、ステップ 2 で指定した Amazon S3 バケット名とファイル名に変更します。
   - us-east-1 を、使用している AWS リージョンに変更します。
   - Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。
   - TopicARN を、Amazon Rekognition Video の設定 のステップ 3 で作成した Amazon SNS トピックの ARN に変更します。
   - RoleARN を、Amazon Rekognition Video の設定 のステップ 7 で作成した IAM サービスロールの ARN に変更します。
   - 必要に応じて、endpoint-url を指定できます。AWS CLI は、指定されたリージョンに基づいて適切なエンドポイント URL を自動的に決定します。ただし、[プライベート VPC の] エンドポイントを使用している場合は、endpoint-url の指定が必要となる場合があります。

[AWS サービスエンドポイント] リソースに、エンドポイント URL を指定するための構文、および各リージョンの名前とコードが記載されています。

- 設定パラメータにフィルター条件を含めることもできます。例えば、目的の値のリストと共に LabelsInclusionFilter または LabelsExclusionFilter を使用できます。

Windows デバイスで CLI にアクセスする場合は、パーサーエラーの発生に対処するため、一重引用符の代わりに二重引用符を使用し、内側の二重引用符をバックスラッシュ (\) でエスケープします。次の例を参照してください。

```
aws rekognition start-label-detection --video "{\"S3Object\":{\"Bucket\":\"amzn-s3-
demo-bucket",\"Name\":\"video-name\"}}" --notification-channel "{\"SNSTopicArn\":
\"TopicARN\",\"RoleArn\":\"RoleARN\"}" \
--region us-east-1 --features GENERAL_LABELS --settings "{\"GeneralLabels\":
{\"LabelInclusionFilters\":[\"Car\"]}}" --profile profile-name
```

2. 応答内の JobId の値に注意してください。この応答は次の JSON の例のようになります。

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

3. Amazon SQS キューをポーリングして完了ステータス JSON を取得するコードを書き込みます ([receive-message] を使用)。
4. 完了ステータス JSON から Status フィールドを抽出するコードを記述します。
5. の値が Statusの場合SUCCEEDED、次の AWS CLI コマンドを実行してラベル検出結果を表示します。

```
aws rekognition get-label-detection  --job-id JobId \
--region us-east-1 --sort-by TIMESTAMP aggregate-by TIMESTAMPS
```

以下の値を更新します。

- JobId を、手順 2 でメモしたジョブ識別子に合わせて変更します。
- Endpoint と us-east-1 を、使用している AWS エンドポイントとリージョンに変更します。

この結果は、次の JSON の例のようになります。

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

リファレンス: ビデオ分析結果の通知

Amazon Rekognition は、完了ステータスを含む Amazon Rekognition Video の分析リクエストの結果を、Amazon Simple Notification Service (Amazon SNS) トピックに発行します。Amazon SNS トピックから通知を取得するには、Amazon Simple Queue Service キューまたは AWS Lambda 関数を使用します。詳細については、「the section called “Amazon Rekognition Video オペレーションを呼

び出す”」を参照してください。例については、Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)を参照してください。

ペイロードは次の JSON 形式になります。

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

名前	説明
JobId	ジョブ識別子。StartPersonTracking などの Start オペレーションから返されるジョブ識別子と一致します。
ステータス	ジョブのステータス。有効な値は、SUCCEEDED、FAILED、または ERROR です。
API	入力ビデオの分析に使用される Amazon Rekognition Video のオペレーション
JobTag	ジョブ識別子。StartLabelDetection などの開始オペレーションへの呼び出しで JobTag を指定します。
Timestamp	ジョブ完了時の UNIX タイムスタンプ。
動画	処理されたビデオに関する詳細。ファイル名やファイルが保存されている Amazon S3 バケットなどが含まれます。

以下の例は、Amazon SNS トピックに送信された正常な処理の通知です。

```
{
  "JobId": "6de014b0-2121-4bf0-9e31-856a18719e22",
  "Status": "SUCCEEDED",
  "API": "LABEL_DETECTION",
  "Message": "",
  "Timestamp": 1502230160926,
  "Video": {
    "S3ObjectName": "video.mpg",
    "S3Bucket": "amzn-s3-demo-bucket"
  }
}
```

Amazon Rekognition Video のトラブルシューティング

Amazon Rekognition Video および保存したビデオの使用に関するトラブルシューティング情報は以下のとおりです。

Amazon SNS トピックに送信された完了ステータス受け取ることができない

Amazon Rekognition Video は、ビデオの分析が完了すると、Amazon SNS トピックにステータス情報を発行します。通常、Amazon SQS キューまたは Lambda 関数を使用してトピックにサブスクライブすることで、完了ステータスメッセージを取得することができます。調査のために、Amazon SNS のトピックを E メールでサブスクライブしておくと、Amazon SNS のトピックに送られたメッセージを E メール受信トレイで受け取ることができます。詳細については、「Amazon SNS トピックへサブスクライブする」を参照してください。

アプリケーションでメッセージが受信されない場合は、以下の点を確認します。

- 分析が完了済みであることを確認します。Get オペレーションレスポンス (`GetLabelDetection` など) の `JobStatus` 値をチェックします。値が `IN_PROGRESS` である場合、分析は未完了であり、完了ステータスは Amazon SNS トピックにまだ発行されていません。
- Amazon SNS トピックへの発行アクセス許可を Amazon Rekognition Video に付与する IAM サービスロールがあることを確認します。詳細については、「Amazon Rekognition Video の設定」を参照してください。
- 使用している IAM サービスロールでロールの認証情報を使用して Amazon SNS トピックに公開できること、およびサービスロールの許可の範囲が使用中のリソースに安全に設定されていることを確認します。次のステップを実行します。

- ユーザーの Amazon リソースネーム (ARN) を取得します。

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

- ロールの信頼関係にユーザー ARN を追加する 詳細については、「ロールの修正」を参照してください。以下の信頼ポリシー例では、ユーザーのロール認証情報を指定し、サービスロールのアクセス権限を、使用しているリソースのみに制限しています (サービスロールの許可の範囲を安全に制限する方法の詳細については、「サービス間の混乱した代理の防止」を参照してください)。

```
{
  "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/*"
                }
      }
    }
  ]
}
```

- ロールを継承します: `aws sts assume-role --role-arn arn:Role ARN --role-session-name SessionName --profile RekognitionUser`
- Amazon SNS トピックに発行する: `aws sns publish --topic-arn arn:Topic ARN --message "Hello World!" --region us-east-1 --profile RekognitionUser`

AWS CLI コマンドが機能する場合は、メッセージ (E メールでトピックをサブスクライブしている場合は E メールの受信トレイ) を受け取ります。メッセージが受信されない場合は、以下を行います。

  - Amazon Rekognition Video が設定されていることをチェックします。詳細については、「Amazon Rekognition Video の設定」を参照してください。
  - このトラブルシューティングの問いに関する他のヒントを確認します。
- 正しいAmazon SNS トピックを使用していることをチェックします。
  - 1 つの Amazon SNS トピックへのアクセスを Amazon Rekognition Video に許可する IAM サービスロールを使用している場合は、正しい Amazon SNS トピックに対するアクセス許可を付与したことを確認します。詳細については、「既存の Amazon SNS トピックへのアクセスをataeruする」を参照してください。
  - IAM サービスロールを使用して Amazon Rekognition Video に複数の SNS トピックへのアクセスを許可する場合は、正しいトピックを使用しており、トピック名の先頭に [AmazonRekognition] が付加されていることを確認します。詳細については、「複数の Amazon SNS トピックへのアクセスを許可する」を参照してください。
  - AWS Lambda 関数を使用する場合は、Lambda 関数が正しい Amazon SNS トピックにサブスクライブされていることを確認します。詳細については、「Lambda 関数へのファンアウト」を参照してください。
- Amazon SQS キューを Amazon SNS トピックにサブスクライブする場合は、その Amazon SQS キューにメッセージを送信するアクセス許可が Amazon SNS トピックにあることを確認します。詳細については、[Amazon SQS キューにメッセージを送信する許可を Amazon SNS トピックに付与する] を参照してください。

Amazon SNS トピックのトラブルシューティングに関する追加のヘルプが必要です

Amazon SNS AWS X-Ray で を使用して、アプリケーションを通過するメッセージをトレースおよび分析できます。詳細については、Amazon SNSと AWS X-Ray」を参照してください。

その他のヘルプについては、[Amazon Rekognition フォーラム] に質問を投稿するか、[AWS テクニカルサポート] へのサインアップを検討してください。

ストリーミングビデオイベントの操作

Amazon Rekognition Video を使用して、ストリーミングビデオ内の顔を検出/認識したり、オブジェクトを検出したりできます。Amazon Rekognition Video は、Amazon Kinesis Video Streams を使用して、ビデオストリームを受信し処理します。ストリームプロセッサがビデオストリームから何を検出するかを示すパラメータを指定して、ストリームプロセッサを作成します。Rekognition は、ストリーミングビデオイベントのラベルの検出結果を Amazon SNS および Amazon S3 通知として送信します。Rekognition は、顔の検索結果を Kinesis データストリームに出力します。

顔検索ストリームプロセッサは、FaceSearchSettings を使用してコレクションから顔を検索します。顔検索ストリームプロセッサを実装してストリーミングビデオ内の顔を分析する方法については、「the section called “ストリーミングビデオのコレクション内での顔検索”」を参照してください。

ラベル検出ストリームプロセッサは、ConnectedHomeSettings を使用してストリーミングビデオイベント内の人、パッケージ、ペットを検索します。ラベル検出ストリームプロセッサを実装する方法については、「the section called “ストリーミングビデオイベント内のラベルの検出”」を参照してください。

Amazon Rekognition Video ストリームプロセッサオペレーションの概要

ストリーミングビデオの分析を開始するには、Amazon Rekognition Video ストリームプロセッサを起動し、Amazon Rekognition Video にビデオをストリーミングします。Amazon Rekognition Video ストリームプロセッサでは、ストリームプロセッサを起動、停止、管理できます。ストリームプロセッサを作成するには、CreateStreamProcessor を呼び出します。顔検索ストリームプロセッサを作成するリクエストパラメータには、Kinesis ビデオストリーム、Kinesis データストリームの Amazon リソースネーム (ARN)、ストリーミングビデオ内の顔認識に使用されるコレクションの識別子が含まれます。セキュリティモニタリングストリームプロセッサを作成するためのリクエストパラメータには、Kinesis ビデオストリームと Amazon SNS トピックの Amazon リソースネーム (ARN)、ビデオストリームで検出したいオブジェクトのタイプ、出力結果の Amazon S3バケットの情報が含まれます。また、ストリームプロセッサに指定した名前も含めます。

StartStreamProcessor オペレーションを呼び出して、ビデオの処理を開始します。ストリームプロセッサのステータス情報を取得するには、DescribeStreamProcessor を呼び出します。呼び出せるその他のオペレーションには、ストリームプロセッサをタグ付けする TagResource、およびストリームプロセッサを削除する DeleteStreamProcessor があります。顔検索ストリームプロセッサを使用している場合は、StopStreamProcessor を使用してストリームプロセッサを停止することもできます。アカウントのストリームプロセッサのリストを取得するには、ListStreamProcessors を呼び出します。

ストリームプロセッサが起動し始めたら、CreateStreamProcessor で指定した Kinesis ビデオストリームを通じて Amazon Rekognition Video にビデオをストリーミングします。Kinesis Video Streams SDK の PutMedia オペレーションを使用して、Kinesis ビデオストリームにビデオを配信できます。例については、「PutMedia API の例」を参照してください。

アプリケーションが顔検索ストリームプロセッサから Amazon Rekognition Video の分析結果を使用する方法については、「ストリーミングビデオの分析結果の読み取り」を参照してください。

新しいストリームプロセッサにタグを追加する

タグを使用して、Amazon Rekognition ストリームプロセッサを識別、整理、検索、フィルタリングできます。各タグは、ユーザー定義のキーと値で構成されるラベルです。

CreateStreamProcessor オペレーション を使用して、ストリームプロセッサの作成時にタグを追加できます。Tags 配列入力パラメーターで、1 つ以上のタグを指定します。タグを用いて、CreateStreamProcessor リクエストに関する JSON の例を次に示します。

```
{
       "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"
       }
}
```

既存のストリームプロセッサにタグを追加する

タグを使用して、Amazon Rekognition ストリームプロセッサを識別、整理、検索、フィルタリングできます。各タグは、ユーザー定義のキーと値で構成されるラベルです。

既存のストリームプロセッサに 1 つ以上のタグを追加するには、TagResource オペレーションを使用します。ストリームプロセッサの Amazon リソースネーム (ARN) (ResourceArn) と、追加したいタグ (Tags) を指定します。次の例は、2 つのタグを追加する方法を示しています。

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

> **Note**
>
> ストリームプロセッサの Amazon リソース名がわからない場合は、DescribeStreamProcessor オペレーションを使用することができます。

ストリームプロセッサ内のタグを一覧表示する

タグを使用して、Amazon Rekognition ストリームプロセッサを識別、整理、検索、フィルタリングできます。各タグは、ユーザー定義のキーと値で構成されるラベルです。

ストリームプロセッサに付属するタグを一覧表示するには、ListTagsForResource オペレーションを実行し、ストリームプロセッサの ARN (ResourceArn) を指定します。応答は、指定されたストリームプロセッサに添付されるタグのキーと値のマップです。

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

出力には、ストリームプロセッサに添付されたタグのリストが表示されます:

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

ストリームプロセッサからタグを削除する

タグを使用して、Amazon Rekognition ストリームプロセッサを識別、整理、検索、フィルタリングできます。各タグは、ユーザー定義のキーと値で構成されるラベルです。

ストリームプロセッサから 1 つ以上のタグを削除するには、UntagResource オペレーションを使用する。モデル (ResourceArn) の ARN と 削除したいタグキー (Tag-Keys) を指定します。

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

または、次の形式でタグキーを指定できます。

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

エラー処理

このセクションでは、ランタイムエラーとその処理方法について説明します。それは、同時に Amazon Rekognition に特有のエラーメッセージとコードについて説明します。

トピック

- エラーコンポーネント
- エラーメッセージおよびコード
- アプリケーションのエラー処理

エラーコンポーネント

プログラムがリクエストを送信すると、 Amazon Rekognition は処理を試行します。リクエストが成功した場合、Amazon Rekognition はそのリクエストされたオペレーションが出力した結果とともに、 HTTP の成功ステータスコード (200 OK) を返します。

リクエストが正常に行われなかった場合、 Amazon Rekognition はエラーを返します。それぞれのエラーには、次の三つのコンポーネントがあります:

- HTTP ステータスコード (400 など)。
- 例外の名前 (InvalidS3ObjectException など)。

- エラーメッセージ (`Unable to get object metadata from S3. Check object key, region and/or access permissions.` など)。

AWS SDK によりアプリケーションにエラーが伝達されるため、適切なアクションを実行できます。たとえば、Java プログラムでは、`try-catch` を処理する `ResourceNotFoundException` ロジックを記述できます。

AWS SDK を使用していない場合は、Amazon Rekognition からの低レベルのレスポンスの内容も解析する必要があります。以下に、そのようなレスポンスの例を示します。

```
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-
 to get object metadata from S3. Check object key, region and/or access permissions."}
```

エラーメッセージおよびコード

Amazon Rekognition によって返される例外のリストを HTTP ステータスコードごとに以下に示します。再試行してもいいですかがはいであれば、同じリクエストを再度送信できます。再試行してもいいですかがいいえであれば、新しいリクエストを送信する前に、クライアント側で問題を修正する必要があります。

HTTP ステータスコード 400

HTTP `400` ステータスコードは、リクエストに問題があることを示しています。問題の例としては、認証の失敗、必須パラメータの不足、またはオペレーションのプロビジョンドスループットの超過などがあります。リクエストを再度送信する前に、アプリケーションで問題を修正する必要があります。

AccessDeniedException

メッセージ: <Operation> オペレーションを呼び出すときにエラー (AccessDeniedException) が発生しました。ユーザー: <User ARN> には <Operation> をリソース: <Resource ARN> で実行する権限がありません

そのアクションを実行する権限がありません。そのオペレーションを実行するには、承認されたユーザーまたは IAM ロールの Amazon リソースネーム (ARN) を使用します。

再試行してもいいですか。いいえ

GroupFacesInProgressException

メッセージ : ジョブのスケジューリングに失敗しました。このコレクションには既存のグループフェイスジョブがあります。

既存のジョブの完了後にオペレーションを再試行します。

再試行してもいいですか。いいえ

IdempotentParameterMismatchException

メッセージ: 指定した ClientRequestToken: <Token> は既に使用されています。

ClientRequestToken 入力パラメータがオペレーションに再利用されましたが、他の入力パラメータの少なくとも 1 つが、オペレーションに対する前回の呼び出しとは異なります。

再試行してもいいですか。いいえ

ImageTooLargeException

メッセージ: イメージサイズが大きすぎます

入力イメージサイズが制限文字数を超えています。DetectProtectiveEquipment を呼び出しているとき、イメージサイズまたは解像度の許容値を超えています。詳細については、「Amazon Rekognition のガイドラインとクォータ」を参照してください。

再試行してもいいですか。いいえ

InvalidImageFormatException

メッセージ: リクエストに無効なイメージ形式があります

指定されたイメージ形式はサポートされていません。サポートされているイメージ形式 (.JPEG および .PNG) を使用します。詳細については、「Amazon Rekognition のガイドラインとクォータ」を参照してください。

再試行してもいいですか。いいえ

InvalidPaginationTokenException

メッセージ

- トークンが無効です
- ページ分割トークンが無効です

リクエストのページ分割トークンが有効ではありません。トークンの有効期限が切れている可能性があります。

再試行してもいいですか。いいえ

InvalidParameterException

メッセージ: リクエストに無効なパラメータがあります

入力パラメータが制約に違反しています。API オペレーションを再度呼び出す前にパラメータを検証します。

再試行してもいいですか。いいえ

InvalidS3ObjectException

メッセージ:

- リクエストに無効な S3 オブジェクトがあります
- S3 からオブジェクトのメタデータを取得できません。オブジェクトキー、リージョン、またはアクセス権限を確認します。

Amazon Rekognition は、リクエストで指定された S3 オブジェクトにアクセスできません。詳細については、「S3 へのアクセスの設定: AWS S3 へのアクセスの管理」を参照してください。トラブルシューティング情報については、「Amazon S3 のトラブルシューティング」を参照してください。

再試行してもいいですか。いいえ

LimitExceededException

メッセージ:

- アカウントに対するストリームプロセッサの超過制限 - <Current Limit> を超えました。
- ユーザー <User ARN> のオープンジョブの数 <Number of Open Jobs> の最大制限: <Maximum Limit>

Amazon Rekognition サービスの制限を超えました。たとえば、多数の Amazon Rekognition Video ジョブを同時に開始する場合、オペレーションを開始する呼び出し (`StartLabelDetection` など) は、現在同時実行中のジョブ数が Amazon Rekognition のサービスの制限を下回るまで、`LimitExceededException` 例外 (HTTP ステータスコード : 400) を発生させます。

再試行してもいいですか。いいえ

ProvisionedThroughputExceededException

メッセージ:

- プロビジョンドレートを超過しました
- S3 ダウンロード制限を超えました

お客様のスループット制限を超えたリクエストの数。詳細については、「Amazon Rekognition サービスの制限」を参照してください。

制限の拡大をリクエストするには、the section called “TPS クォータを変更するケースを作成する” で次の指示に従ってください。

再試行してもいいですか。はい

ResourceAlreadyExistsException

メッセージ: コレクション id: <Collection Id> はすでに存在しています

指定された ID の付いたコレクションはすでに存在しています。

再試行してもいいですか。いいえ

ResourceInUseException

メッセージ:

- ストリームプロセッサの名前はすでに使用されています
- 指定されたリソースは使用中です
- ストリームを停止するためにプロセッサを利用できません
- ストリームプロセッサを削除できません

リソースが利用可能になったらオペレーションを再試行します。

再試行してもいいですか。いいえ

ResourceNotFoundException

メッセージ: API コールに応じたさまざまなメッセージ。

指定されたリソースは存在しません。

再試行してもいいですか。いいえ

ThrottlingException

メッセージ: リクエストのレートが急激に増加しました。レートを下げてください。

リクエストの増加レートが速すぎます。リクエストレートを下げてから、徐々に引き上げます。エクスポネンシャルバックオフを実行してリクエストを再試行することをお勧めします。デフォルトでは、 AWS SDKs自動再試行ロジックとエクスポネンシャルバックオフを使用します。詳細については、「AWS でのエラー時の再試行とエクスポネンシャルバックオフ」および「エクスポネンシャルバックオフとジッタ」を参照してください。

再試行してもいいですか。はい

VideoTooLargeException

メッセージ: ビデオサイズ: <Video Size> バイトが、最大制限: <Max Size> バイトを上回っています。

指定されたメディアのファイルサイズが大きすぎるか、時間が長すぎます。詳細については、「Amazon Rekognition のガイドラインとクォータ」を参照してください。

再試行してもいいですか。いいえ

HTTP ステータスコード 5xx

HTTP ステータスコード 5xx は、 AWSで解決する必要のある問題を示しています。This might be a transient error。その場合、リクエストを再試行することで成功することがあります。それ以外の場合、サービスに運用上の問題があるかどうかを確認するために、AWS Service Health Dashboard を参照してください。

InternalServerError (HTTP 500)

メッセージ: 内部サーバーエラー

Amazon Rekognition でサービスの問題が発生しました。もう一度やり直してください。エクスポネンシャルバックオフを実行し、リクエストを再試行します。 AWS SDK には、デフォルトで自動再試行ロジックとエクスポネンシャルバックオフが使用されています。詳細については、「AWS でのエラー時の再試行とエクスポネンシャルバックオフ」および「エクスポネンシャルバックオフとジッタ」を参照してください。

再試行してもいいですか。はい

ThrottlingException (HTTP 500)

メッセージ: サービスが使用できません

Amazon Rekognition は一時的にリクエストを処理できませんでした。もう一度やり直してください。エクスポネンシャルバックオフを実行してリクエストを再試行することをお勧めします。デフォルトでは、 AWS SDKs自動再試行ロジックとエクスポネンシャルバックオフを使用します。詳細については、「AWS でのエラー時の再試行とエクスポネンシャルバックオフ」および「エクスポネンシャルバックオフとジッタ」を参照してください。

再試行してもいいですか。はい

アプリケーションのエラー処理

アプリケーションをスムーズに実行するには、エラーをキャッチしエラーに対応するロジックを組み込む必要があります。一般的な方法には、`try-catch`ブロックや`if-then`ステートメントの使用などがあります。

AWS SDK は独自に再試行とエラーチェックを実行します。AWS SDK の使用中にエラーが発生した場合は、エラーコードと説明が問題のトラブルシューティングに役立ちます。

また、レスポンスにRequest IDが表示されます。Request IDは、AWS Support を使用して問題を診断することが必要な場合に便利です。

以下の Java コードスニペットは、イメージ内のオブジェクトの検出を試み、基本的なエラー処理を実行します (この場合、ユーザーにはリクエストが失敗したことが通知されます)。

```
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());
}
```

このコードスニペットでは、try-catchの構成は 2 つの異なるタイプの例外を処理します。

- AmazonRekognitionException クライアントリクエストが Amazon Rekognition に正しく送信されたが、 Amazon Rekognition がリクエストを処理できず、代わりにエラーレスポンスを返した場合に、この例外が発生します。
- AmazonClientException クライアントがサービスからのレスポンスを取得または解析できなかった場合に、この例外が発生します。

FedRAMP 認定サービスとしての Amazon Rekognition の使用

AWS FedRAMP コンプライアンスプログラムには、FedRAMP 認定サービスとして Amazon Rekognition が含まれています。連邦政府または商用のお客様は、 サービスを使用して、影響レベル

が中程度のデータで、機密性の高いワークロードを AWS 米国東部および米国西部リージョンで処理および保存できます。また、 AWS GovCloud (米国) リージョンの認可境界で影響レベルが高程度のデータを使用して、機密ワークロードのサービスを使用できます。FedRAMP コンプライアンスの詳細については、「AWS FedRAMP コンプライアンス」を参照してください。

FedRAMP に準拠するには、連邦情報処理規格 (FIPS) エンドポイントを使用できます。これにより、機密情報を扱うときに、FIPS 140-2 検証済み暗号化モジュールにアクセスできます。FIPS エンドポイントの詳細については、「FIPS 140-2 の概要」を参照してください。

AWS Command Line Interface（ AWS CLI) またはいずれかの AWS SDKs を使用して、Amazon Rekognition で使用されるエンドポイントを指定できます。

Amazon Rekognition で使用可能なエンドポイントについては、「Amazon Rekognition のリージョンとエンドポイント」を参照してください。

以下に、コレクションの一覧表示 トピックの Amazon Rekognition デベロッパーガイド。これらは、Amazon Rekognition にアクセスするリージョンと FIPS エンドポイントを指定するように変更されます。

Java

Java の場合は、Amazon Rekognition クライアントを作成するときに `withEndpointConfiguration` メソッドを使用します。この例では、米国東部 (バージニア北部) リージョンで FIPS エンドポイントを使用するコレクションを示します。

```
//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

では AWS CLI、 `--endpoint-url`引数を使用して、Amazon Rekognition にアクセスするエンドポイントを指定します。この例では、米国東部 (オハイオ) リージョンで FIPS エンドポイントを使用するコレクションを示します。

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

Python

Python の場合は、boto3.client 関数で endpoint_url 引数を使用します。指定するエンドポイントに設定します。この例では、米国西部 (オレゴン) リージョンで FIPS エンドポイントを使用するコレクションを示します。

```
#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()
```

センサー、入力イメージ、ビデオのベストプラクティス

このセクションでは、Amazon Rekognition の使用に関するベストプラクティスを紹介します。これらのベストプラクティスは、呼び出すオペレーションから最適なパフォーマンスを引き出すのに役立ちます。オペレーションから期待される結果を得るのが難しい場合は、ここで説明されているベストプラクティスに従っていることを確認してください。

画像オペレーションのレイテンシーについては、以下を参照してください。

- the section called “Amazon Rekognition イメージオペレーションのレイテンシ”

顔比較および顔検索のオペレーションでは、画像内の顔を検索するために特定のベストプラクティスに従う必要があります。これらの要件は以下のリンク先にも記載されています。

- the section called “顔比較用の入力イメージに関する推奨事項”
- the section called “コレクション内で顔を検索するときの推奨事項”

以下のセクションでは、Amazon Rekognition が分析できるメディアのタイプ別に、カメラを設定する方法について説明します。

- the section called “カメラセットアップの推奨事項 (画像およびビデオ)”
- the section called “カメラセットアップの推奨事項 (保存済みビデオおよびストリーミングビデオ)”
- the section called “カメラセットアップの推奨事項 (ストリーミングビデオ)”

顔のライブネスチェックオペレーションにも独自のベストプラクティスがあり、ライブネスチェックツールから最高のパフォーマンスを引き出すためには従う必要があります。

- the section called “Face Liveness の使用に関する推奨事項”

Amazon Rekognition イメージオペレーションのレイテンシ

Amazon Rekognition イメージ オペレーションの可能な限り低いレイテンシーを実現するには、以下の点を考慮します。

- イメージが含まれている Amazon S3 バケットのリージョンと Amazon Rekognition イメージ API オペレーションで使用するリージョンが一致している必要があります。

- イメージのバイトを使用した Amazon Rekognition イメージ オペレーションの呼び出しは、イメージを Amazon S3 バケットにアップロードしてから、アップロードされたイメージを Amazon Rekognition イメージ オペレーションで参照するよりも高速です。ほぼリアルタイムの処理のためにイメージを Amazon Rekognition イメージ にアップロードする場合は、この手法を検討してください。例えば、IP カメラからアップロードされるイメージや、ウェブポータルを通じてアップロードされるイメージです。
- イメージがすでに Amazon S3 バケットにある場合、Amazon Rekognition イメージ オペレーションで参照すると、イメージのバイトをオペレーションに渡すよりも高速になる可能性があります。

顔比較用の入力イメージに関する推奨事項

顔比較オペレーションに使用されるモデルは、さまざまなポーズ、表情、年齢層、ローテーション、照明条件、サイズで機能するように設計されています。CompareFaces の参考写真を選んだり、IndexFaces を使ってコレクションに顔を追加したりするときは、以下のガイドラインに従うことが推奨されます。

顔のオペレーションの入力画像に関する一般推奨事項

- 明るくシャープな画像を使用します。被写体やカメラの動きによって、ぼやけている可能性がある画像は、できるだけ使用しないでください。明るさとシャープネスは、DetectFaces で判断できます。
- 視線を検出するときは、元の画像を元のサイズおよび画質のままでアップロードすることが推奨されます。
- 推奨される角度範囲内に顔がある画像を使用します。ピッチは、下向きに 30 度未満、上向きに 45 度未満にしてください。ヨーイングはどちらの方向にも 45 度未満である必要があります。ロールに制限はありません。
- 両目が開いて見える顔の画像を使用します。
- 顔がピンぼけしていたり、その大部分が切り取られたりしていない画像を使用します。イメージには人の頭全体と肩が含まれている必要があります。顔の境界ボックスに切り取られている必要はありません。
- ヘッドバンドやマスクなど、顔を遮るものは避けます。
- 顔が画像の大部分を占めている画像を使用します。顔が画像の大部分を占める画像は、より高い精度でマッチングされます。
- 画像の解像度が十分に大きいことを確認します。Amazon Rekognition は、最大 1,920 x 1,080 の解像度のイメージで 50 x 50 ピクセルの小さい顔を認識できます。高解像度のイメージでは、最小

の顔として、より大きいサイズが必要です。最小サイズより大きい顔は、顔比較でより正確な結果のセットを提供します。

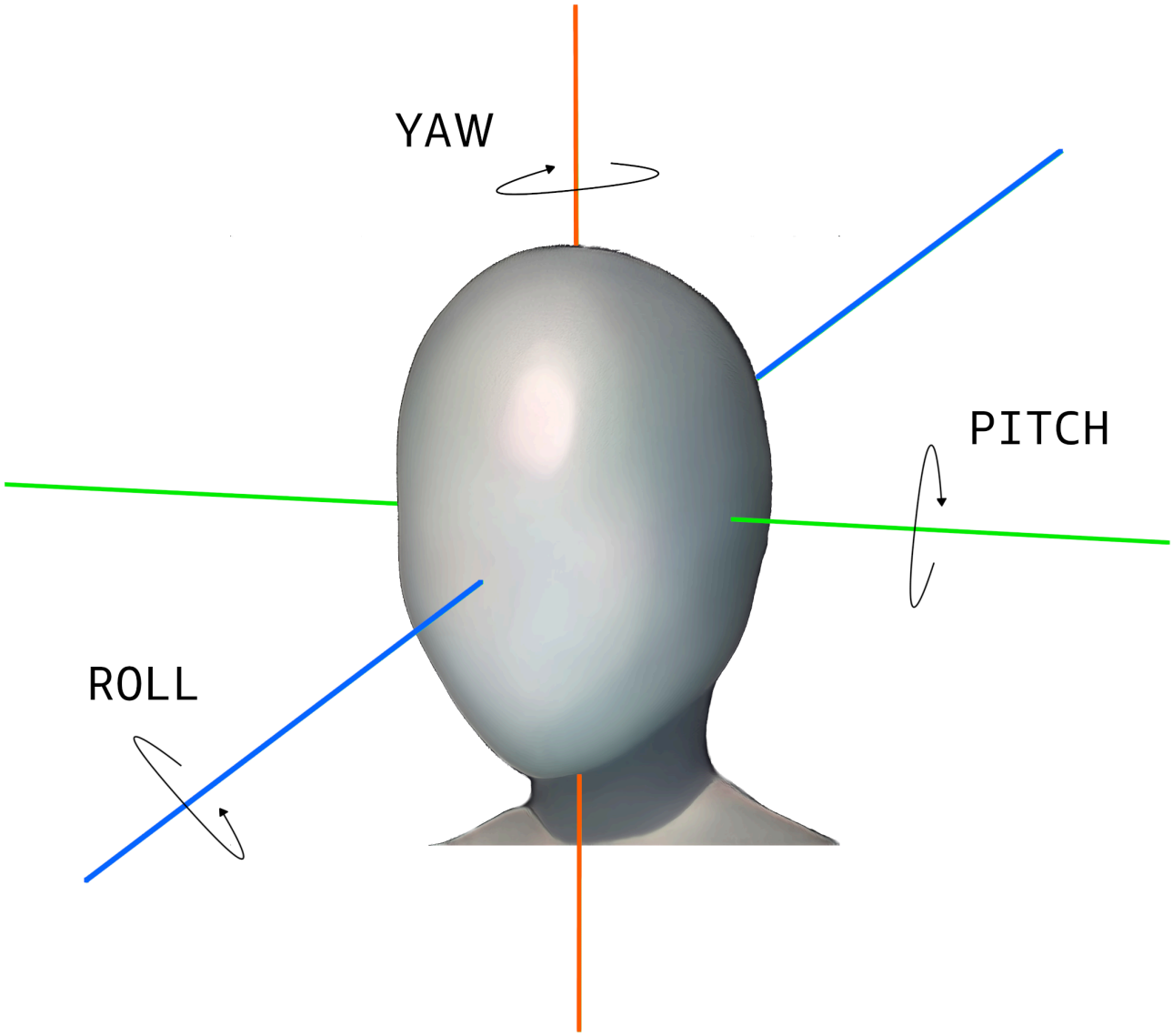
- カラーイメージを使用すること。
- 影などの陰影がなく、照明が顔に均一に当たっている画像を使用します。
- 背景とのコントラストが十分である画像を使用します。高コントラストのモノクロの背景が最適です。
- 高精度を必要とするアプリケーションには、口を閉じて、わずかにほほ笑むかほほ笑まない中立的な表情を持つ顔の画像を使用してください。

コレクション内で顔を検索するときの推奨事項

- コレクション内で顔を検索するときは、最新の顔画像がインデックスに登録されていることを確認します。
- IndexFaces を使用してコレクションを作成する際、ピッチとヨーイングの異なる (推奨角度範囲内の) 複数の個人の顔画像を使用します。少なくともその人物の 5 枚の画像をインデックスに登録することをお勧めします。正面をまっすぐ向いたもの、45 度以下のヨーイングで左を向いた顔、45 度以下のヨーイングで右を向いた顔、30 度以下のピッチで下を向いた顔、45 度以下のピッチで上を向いた顔。これらのフェイスインスタンスが同じ個人に属していることを追跡するときに、インデックス付けされているイメージ内に顔が 1 つしかない場合は、外部イメージ ID 属性を使用することを検討してください。例えば、John_Doe_1.jpg, … John_Doe_5.jpg のような外部のイメージ ID を使用して、コレクションの中から John Doe の 5 枚のイメージを追跡できます。

カメラセットアップの推奨事項 (画像およびビデオ)

以下の推奨事項は 顔比較用の入力イメージに関する推奨事項 に追加されます。



- イメージの解像度 顔の解像度が最大 1,920 x 1,080 の合計解像度を持つイメージに対して 50 x 50 ピクセルである限り、イメージの解像度に最小要件はありません。高解像度のイメージでは、最小の顔として、より大きいサイズが必要です。

> **Note**
>
> 前述の推奨事項は、カメラのネイティブ解像度に基づいています。低解像度画像から高解像度画像を生成しても、顔検索に必要な結果は得られません (画像のアップサンプリングによって生成されるアーティファクトのため)。

- カメラアングル-カメラアングルには、ピッチ、ロール、ヨーイングの 3 つの測定値があります。
  - ピッチ - 30 度未満のピッチをお勧めします。カメラが下を向いているとき、カメラが上を向いているときは 45 度未満にします。
  - ロール-このパラメータには最小要件はありません。Amazon Rekognition はどんな量のロールでも処理できます。
  - ヨーイング - いずれの方向でも 45 度以下のヨーイングを推奨します。

  カメラによってキャプチャされる任意の軸に沿った顔の角度は、シーンに面するカメラアングルと被写体の頭がシーン内にある角度の両方の組み合わせです。例えば、カメラは 30 度下向きに、ユーザーがさらに 30 度頭を下げている場合、カメラから見た実際のフェイスピッチは 60 度です。この場合、Amazon Rekognition は顔を認識することはできません。カメラアングルは、一般的に、30 度以下の全体的なピッチ (顔とカメラを合わせて) でカメラを覗いているという想定に基づいてカメラを設定することをお勧めします。
- カメラのズーム – 推奨される最小顔解像度 (50 x 50 ピクセル) に基づいて、このパラメータを指定します。被写体の顔解像度が 50 X 50 ピクセル以上になるカメラのズーム設定を使用することをお勧めします。
- カメラの高さ – 推奨されるカメラピッチに基づいて、このパラメータを指定します。

カメラセットアップの推奨事項 (保存済みビデオおよびストリーミングビデオ)

以下の推奨事項は カメラセットアップの推奨事項 (画像およびビデオ) に追加されます。

- コーデックは h.264 でエンコードする必要があります。
- 推奨されるフレームレートは 30 fps です (5 fps 未満にすることはできません)。
- エンコーダの推奨ビットレートは 3 Mbps です (1.5 Mbps 未満にすることはできません)。
- フレーム解像度 – エンコーダのビットレートが制約事項である場合、より良い顔検索結果を得るために、フレームレートの高さよりもフレーム解像度の高さを優先することをお勧めします。これ

により、Amazon Rekognition は割り当てられたビットレート内で最高品質のフレームを取得できます。ただし、これには欠点があります。フレームレートが低いため、カメラではシーン内の高速の動きが捉えられません。所定の設定に対する、これら 2 つのパラメータ間の損失評価を理解することが重要です。例えば、指定可能な最大ビットレートが 1.5 Mbps の場合、カメラは 5fps で 1080p、15fps で 720p をキャプチャできます。推奨される 50 x 50 ピクセルの顔解像度が満たされている限り、2 つのいずれを選択するかはアプリケーションに依存します。

カメラセットアップの推奨事項 (ストリーミングビデオ)

次の推奨事項は カメラセットアップの推奨事項 (保存済みビデオおよびストリーミングビデオ) に追加されます。

ストリーミングアプリケーションに関する追加の制約事項はインターネット帯域幅です。ライブ動画では、Amazon Rekognition は入力として Amazon Kinesis Video Streams のみを受け付けます。エンコーダのビットレートと利用可能なネットワーク帯域幅の間の依存関係を理解する必要があります。利用可能な帯域幅は、最小限、カメラがライブストリームのエンコードに使用しているのと同じビットレートをサポートする必要があります。これにより、カメラによってキャプチャされたものはすべて、Amazon Kinesis Video Streams によって中継されるようになります。使用可能な帯域幅がエンコーダビットレートよりも小さい場合、Amazon Kinesis Video Streams はネットワーク帯域幅に基づいてビットを下げます。これにより、ビデオの品質が低下します。

一般的なストリーミング設定では、ストリームを中継するネットワークハブに複数のカメラを接続します。この場合、帯域幅は、ハブに接続されているすべてのカメラからのストリームの累積合計に対応する必要があります。例えば、ハブが 1.5 Mbps でエンコードされている 5 台のカメラに接続されている場合、使用可能なネットワーク帯域幅は少なくとも 7.5 Mbps であることが必要です。削除されたパケットがないことを確認するには、カメラとハブの間の接続が切断されているため、ネットワーク帯域幅を 7.5 Mbps よりも高く維持してジッタに対処することを検討する必要があります。実際の値は、内部ネットワークの信頼性によって異なります。

Face Liveness の使用に関する推奨事項

Rekognition Face Liveness を使用するときは次のベストプラクティスに従うことが推奨されます。

- Face Liveness のチェックは、暗すぎず明るすぎない、照明が均一な環境で行います。
- ウェブブラウザでチェックするときは、ディスプレイの画面の明るさを最大レベルまで上げます。Mobile Native SDK はディスプレイの明るさを自動的に調整します。

- ユースケースの性質を反映した、信頼スコアのしきい値を選択します。セキュリティ上の懸念が大きいユースケースでは高いしきい値を使用します。
- 監査画像には人間によるレビューチェックを定期的に実施し、設定した信頼性しきい値で、なりすまし攻撃が軽減しているか確認します。
- 光に敏感なユーザーや、Rekognition を使用した顔のライブネス判定を希望しないユーザーには、別の判定方法を提案します。
- ライブネスのチェックスコアは、ユーザーアプリケーションに送信したりそこに表示したりすることはできません。送信できるのは成功または失敗の信号のみです。
- 1 つのデバイスで許容されるライブネスチェックの失敗数は 3 分間で 5 回までです。5 回失敗すると、そのユーザーは 30～60 分間タイムアウトします。このパターンが 3～5 回繰り返されると、そのユーザーデバイスはそれ以上の通話ができなくなります。
- 準備完了画面をワークフローに実装すれば、ユーザーはよりスムーズに Face Liveness チェックに成功できるようになります。
- お客様は、Face Liveness によるコンテンツの処理、保存、使用、転送に関して、エンドユーザーに法的に適切なプライバシー通知を提示し、エンドユーザーから必要な同意を取得する責任を負います。

オブジェクトおよび概念の検出

このセクションでは、Amazon Rekognition イメージおよび Amazon Rekognition Video を使用してイメージおよびビデオのラベルを検出する情報について説明します。

ラベルやタグは、イメージやビデオ内でそのコンテンツに基づいて発見されたオブジェクトや概念 (シーンやアクションを含む) です。例えば、南国のビーチでくつろぐ人々のイメージには、ヤシの木 (オブジェクト)、ビーチ (シーン)、ランニング (アクション)、アウトドア (概念) が含まれる場合があります。

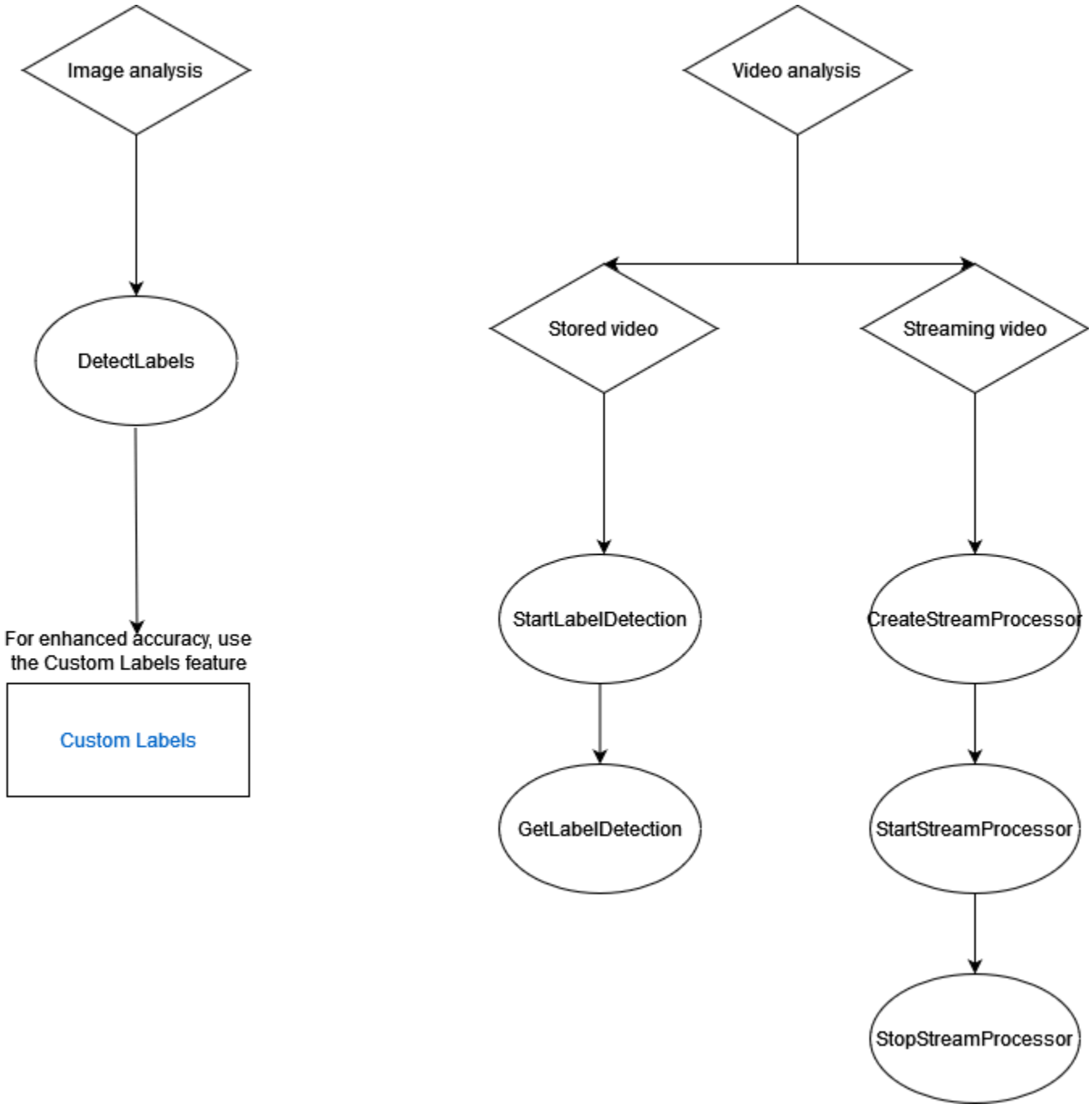
Rekognition のラベル検出オペレーションでサポートされているラベル

- Amazon Rekognition でサポートされているラベルおよびオブジェクト境界ボックスの完全なリストをダウンロードするには、こちらをクリックしてください。
- ラベルおよびオブジェクト境界ボックスの以前のリストをダウンロードするには、こちらをクリックしてください。

> **Note**
>
> Amazon Rekognition は、特定のイメージ内の人物の物理的な外観に基づいて、性別バイナリ（男性、女性、女の子など）の予測を作成します。この種の予測は、人物の性自認を分類するためのものではないため、Amazon Rekognition を使用してこのような判断を下さないようにしましょう。たとえば、ある役のために長髪のかつらやイヤリングを身に着けている男性俳優は、女性として予測されるかもしれません。
> Amazon Rekognition を使用してジェンダーバイナリー予測を作成することは、特定のユーザーを確認せずに性別分布の集計統計を分析する必要があるユースケースに最適です。たとえば、ソーシャルメディアプラットフォームで男性と比較した女性のユーザーの割合。
> 性別二者択一の予測を使用して、個人の権利、プライバシー、またはサービスへのアクセスに影響する決定を行うことはお勧めしません。

Amazon Rekognition は英語でラベルを返します。Amazon Translate を使って、英語のラベルをその他の言語に翻訳することができます。

以下の図では、Amazon Rekognition Image または Amazon Rekognition Video オペレーションの使用目的に応じて、必要なオペレーションを呼び出す順序を示しています。



レスポンスオブジェクトのラベル付け

境界ボックス

Amazon Rekognition イメージおよび Amazon Rekognition Video は、車、家具、アパレル、またはペットなどの一般的なオブジェクトラベルの境界ボックスを返すことができます。境界ボックス情報は、それほど一般的ではないオブジェクトラベルを返しません。境界ボックスを使用して、画像内の

オブジェクトの正確な位置を検出したり、検出された被写体のインスタンスを数えたり、境界ボックスのディメンションを使用して被写体のサイズを測定したりできます。

たとえば、次のイメージでは、Amazon Rekognition イメージは人物、スケートボード、駐車中の車、その他の情報の存在を検出することができます。Amazon Rekognition のイメージは、検出された人、および車や車輪などの他の検出されたオブジェクトの境界ボックスも返します。



信頼スコア

Amazon Rekognition Video および Amazon Rekognition Image は、Amazon Rekognition で検出された各ラベルの精度がどの程度信頼できるかを示す割合のスコアも表示します。

親

Amazon Rekognition イメージおよび Amazon Rekognition Video は、ラベルを分類するために祖先ラベルの階層分類を使用します。たとえば、道路を横切って歩いている人は Pedestrian として検出される可能性があります。Pedestrian の親ラベルは Person です。これらのラベルは両方とも応答で返されます。すべての先祖ラベルが返され、指定されたラベルにはその親および他の先祖ラベルのリ

ストが含まれます。たとえば、祖父母ラベルと曽祖父母ラベル (存在する場合) です。親ラベルを使用して、関連するラベルのグループを作成したり、1 つ以上の画像内の類似ラベルを照会したりできます。たとえば、すべての Vehicles に対するクエリは、ある画像から自動車を、別の画像からバイクを返す可能性があります。

カテゴリ

Amazon Rekognition Image および Amazon Rekognition Video は、ラベルカテゴリに関する情報を返します。ラベルは、「車両と自動車」や「食品と飲料」のように、共通の機能やコンテキストに基づいて個々のラベルをグループ化するカテゴリの一部です。ラベルカテゴリは親カテゴリのサブカテゴリにすることができます。

エイリアス

Amazon Rekognition Image と Amazon Rekognition Video は、ラベルを返すだけでなく、ラベルに関連付けられているエイリアスをすべて返します。エイリアスとは、同じ意味を持つラベル、または返されるプライマリラベルと視覚的に置換可能なラベルです。例えば、「セルフォン」は「携帯電話」のエイリアスです。

以前のバージョンでは、Amazon Rekognition Image は「携帯電話」を含むプライマリラベル名と同じリストに「セルフォン」などのエイリアスを返していました。現在 Amazon Rekognition Image は、「エイリアス」というフィールドに「セルフォン」を返し、プライマリラベル名のリストに「携帯電話」を返すようになりました。アプリケーションが以前のバージョンの Rekognition によって返された構造に依存している場合、必要に応じて、イメージまたはビデオのラベル検出オペレーションによって返される現在のレスポンスを、すべてのラベルとエイリアスがプライマリラベルとして返される以前のレスポンス構造に変換します。

DetectLabels API (イメージ内のラベル検出用) からの現在のレスポンスを以前のレスポンス構造に変換する必要がある場合は、「DetectLabels レスポンスの変換」のコード例を参照してください。

GetLabelDetection API (保存されているビデオのラベル検出用) からの現在のレスポンスを以前のレスポンス構造に変換する必要がある場合は、「GetLabelDetection レスポンスの変換」のコード例を参照してください。

イメージプロパティ

Amazon Rekognition Image は、イメージ全体の画質 (鮮明度、明るさ、コントラスト) に関する情報を返します。イメージの前景と背景の鮮明度と明るさに関する情報も返されます。イメージプロパ

ティを使用して、イメージ全体、前景、背景、および境界ボックスを持つオブジェクトのドミナントカラーを検出することもできます。



以下は、処理中のイメージの DetectLabels オペレーションのレスポンスに含まれる ImageProperties データの例です。

Image Properties	Dominant Colors Examples and Pixel Percentage	Image Quality
Entire Image	Hex code #808080, RGB (128, 128, 128), 15.72	Brightness: 76.08 Sharpness: 89.72 Contrast: 88.42
	Hex code #000000, RGB (0, 0, 0), 15.10	
	Hex code #696969, RGB (105, 105, 105), 14.02	
	Hex code #8fbc8f, RGB (143, 188, 143), 12.70	
	Hex code #5f9ea0, RGB (95, 158, 160), 11.92	
Foreground	Hex code #8fbc8f, RGB (143, 188, 143), 30.18	Brightness: 79.48 Sharpness: 93.47
	Hex code #5f9ea0, RGB (95, 158, 160), 24.29	
	Hex code #000000, RGB (0, 0, 0), 12.02	
	Hex code #2f4f4f, RGB (47, 79, 79), 9.20	
	Hex code #696969, RGB (105, 105, 105), 8.95	
Background	Hex code #808080, RGB (128, 128, 128), 21.16	Brightness: 74.42 Sharpness: 87.84
	Hex code #2f4f4f, RGB (47, 79, 79), 14.61	
	Hex code #000000, RGB (0, 0, 0), 14.23	
	Hex code #696969, RGB (105, 105, 105), 13.19	
	Hex code #ffebcd, RGB (255, 235, 205), 12.80	
Car (example of objects with bounding boxes)	Hex code #5f9ea0, RGB (95, 158, 160), 29.18	Not applicable
	Hex code #8fbc8f, RGB (143, 188, 143), 14.39	
	Hex code #000000, RGB (0, 0, 0), 11.76	
	Hex code #808080, RGB (128, 128, 128), 11.38	
	Hex code #2f4f4f, RGB (47, 79, 79), 9.44	

イメージプロパティは Amazon Rekognition Video では使用できません。

モデルのバージョン

Amazon Rekognition イメージおよび Amazon Rekognition Video は、いずれもイメージまたは保存されたビデオ内のラベルを検出するために使用されたラベル検出モデルのバージョンを返します。

包含フィルターと除外フィルター

Amazon Rekognition Image および Amazon Rekognition Video ラベル検出オペレーションによって返された結果をフィルタリングできます。ラベルとカテゴリのフィルタリング条件を指定して結果をフィルタリングします。ラベルフィルターには、包含フィルターと除外フィルターがあります。

`DetectLabels` を使用して得られた結果のフィルタリングの詳細については、「イメージ内のラベルの検出」を参照してください。

GetLabelDetection によって得られた結果のフィルタリングの詳細については、「ビデオ内のラベルの検出」を参照してください。

結果のソートと集計

特定の Amazon Rekognition Video オペレーションから得られた結果は、タイムスタンプとビデオセグメントに従ってソートおよび集計できます。ラベル検出ジョブまたはコンテンツモデレーションジョブの結果を、GetLabelDetection または GetContentModeration を使用して取得する場合、SortBy および AggregateBy 引数を使用して結果の返し方を指定できます。SortBy を TIMESTAMP または NAME (ラベル名) と一緒に使用したり、TIMESTAMPS または SEGMENTS を AggregateBy 引数と一緒に使用したりできます。

イメージ内のラベルの検出

DetectLabels オペレーションを使用して、イメージ内のラベル (オブジェクトと概念) を検出し、イメージのプロパティに関する情報を取得できます。イメージプロパティには、前景や背景の色、イメージの鮮明度、明るさ、コントラストなどの属性が含まれます。イメージ内のラベルのみ、イメージプロパティのみ、またはその両方を取得できます。例については、Amazon S3 バケットに保存されたイメージの分析を参照してください。

次の例では AWS SDKs と を使用して AWS CLI を呼び出しますDetectLabels。DetectLabels オペレーションからのレスポンスの詳細については、「DetectLabels のレスポンス」を参照してください。

イメージ内のラベルを検出するには

1. まだ実行していない場合:

   a. AmazonRekognitionFullAccess と AmazonS3ReadOnlyAccess のアクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. 1 つ以上のオブジェクト (樹木、家、ボートなど) が含まれているイメージを S3 バケットにアップロードします。イメージは、.jpg 形式または .png 形式にする必要があります。

   手順については、Amazon Simple Storage Service 入門ガイドの「Amazon S3 へのオブジェクトのアップロード」を参照してください。

3. 以下の例を使用して、DetectLabels オペレーションを呼び出します。

Java

この例では、入力イメージ内で検出されたラベルのリストを表示します。bucket および photo の値は、ステップ 2 で使用したバケット名とイメージ名に置き換えます。

```
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

この例では、detect-labels CLI オペレーションの JSON 出力を表示します。bucket および photo の値は、ステップ 2 で使用した Amazon S3 バケット名とイメージ名に置き換えます。profile-name の値を自分のデベロッパープロファイル名に置き換えます。

```
aws rekognition detect-labels --image '{ "S3Object": { "Bucket": "bucket-name",
 "Name": "file-name" } }' \
--features GENERAL_LABELS IMAGE_PROPERTIES \
--settings '{"ImageProperties": {"MaxDominantColors":1}, {"GeneralLabels":
{"LabelInclusionFilters":["Cat"]}}}' \
--profile profile-name \
--region us-east-1
```

Windows デバイスで CLI にアクセスする場合は、パーサーエラーの発生に対処するため、一重引用符の代わりに二重引用符を使用し、内側の二重引用符をバックスラッシュ (\) でエスケープします。例として以下を参照してください。

```
aws rekognition detect-labels --image "{\"S3Object\":{\"Bucket\":\"bucket-name
\",\"Name\":\"file-name\"}}" --features GENERAL_LABELS IMAGE_PROPERTIES \
--settings "{\"GeneralLabels\":{\"LabelInclusionFilters\":[\"Car\"]}}" --profile
 profile-name --region us-east-1
```

Python

この例では、入力イメージ内で検出されたラベルを表示します。関数 `main` で、`bucket` および `photo` の値は、ステップ 2 で使用した Amazon S3 バケット名とイメージ名に置き換えます。Rekognition セッションを作成する行の `profile_name` の値を、自分のデベロッパープロファイル名に置き換えます。

```
#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):

     session = boto3.Session(profile_name='profile-name')
     client = session.client('rekognition')

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

```
    # Uncomment to use image properties and filtration settings
    #Features=["GENERAL_LABELS", "IMAGE_PROPERTIES"],
    #Settings={"GeneralLabels": {"LabelInclusionFilters":["Cat"]},
    # "ImageProperties": {"MaxDominantColors":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("Aliases:")
        for alias in label['Aliases']:
            print(" " + alias['Name'])

            print("Categories:")
        for category in label['Categories']:
            print(" " + category['Name'])
            print("----------")
            print()

    if "ImageProperties" in str(response):
        print("Background:")
        print(response["ImageProperties"]["Background"])
        print()
        print("Foreground:")
        print(response["ImageProperties"]["Foreground"])
        print()
        print("Quality:")
```

```
        print(response["ImageProperties"]["Quality"])
        print()

    return len(response['Labels'])

def main():
    photo = 'photo-name'
    bucket = 'amzn-s3-demo-bucket'
    label_count = detect_labels(photo, bucket)
    print("Labels detected: " + str(label_count))

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

.NET

この例では、入力イメージ内で検出されたラベルのリストを表示します。bucket および photo の値は、ステップ 2 で使用した Amazon S3 バケット名とイメージ名に置き換えます。

```
//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 = "amzn-s3-demo-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

この例では、入力イメージ内で検出されたラベルのリストを表示します。bucket および photo の値は、ステップ 2 で使用した Amazon S3 バケット名とイメージ名に置き換えます。

```
  # 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

この例では、入力イメージ内で検出されたラベルのリストを表示します。bucket および photo の値は、ステップ 2 で使用した Amazon S3 バケット名とイメージ名に置き換えます。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

TypeScript 定義を使用している場合は、import AWS from 'aws-sdk' の代わりにconst AWS = require('aws-sdk') Node.js でプログラムを実行するようになります。詳細については、AWS JavaScriptのSDK をご覧ください。構成の設定方法によって

は、AWS.config.update({region:*region*}); でリージョンを指定する必要がある場合もあります。

```
// Load the SDK
var AWS = require('aws-sdk');
const bucket = 'bucket-name' // the bucketname without s3://
const photo  = 'image-name' // the name of file

var credentials = new AWS.SharedIniFileCredentials({profile: 'profile-name'});
AWS.config.credentials = credentials;
AWS.config.update({region:'region-name'});

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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例は こちら を参照してください。

```
//snippet-start:[rekognition.java2.detect_labels.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
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 software.amazon.awssdk.services.rekognition.model.S3Object;
import java.util.List;

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DetectLabels {

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
```

```
            "   <bucket> <image>\n\n" +
            "Where:\n" +
            "   bucket - The name of the Amazon S3 bucket that contains the
 image (for example, ,ImageBucket)." +
            "   image - The name of the image located in the Amazon S3 bucket
 (for example, Lake.png). \n\n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String image = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        getLabelsfromImage(rekClient, bucket, image);
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.detect_labels_s3.main]
    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);
        }
    }
 // snippet-end:[rekognition.java2.detect_labels.main]
}
```

DetectLabels オペレーションのリクエスト

DetectLabel への入力はイメージです。この JSON 入力の例では、ソースイメージを Amazon S3 バケットからロードします。MaxLabels はレスポンスで返すラベルの最大数です。MinConfidence は Amazon Rekognition イメージが検出済みラベルの精度で保持しなければならない必要最小限の信頼度であり、それのためにレスポンスで返されます。

Features では、返されるイメージの 1 つ以上の特徴を指定でき、GENERAL_LABELS と IMAGE_PROPERTIES を選択できます。GENERAL_LABELS を含めると、入力イメージで検出されたラベルが返され、IMAGE_PROPERTIES を含めるとイメージの色と品質を確認できるようになります。

Settings では、返されるアイテムを GENERAL_LABELS と IMAGE_PROPERTIES 機能の両方でフィルターできます。ラベルには、包含フィルターと除外フィルターを使用できます。特定のラベル、個別のラベル、またはラベルカテゴリ別にフィルタリングすることもできます。

- LabelInclusionFilters - レスポンスに含めるラベルを指定できます。
- LabelExclusionFilters - レスポンスから除外するラベルを指定できます。
- LabelCategoryInclusionFilters - レスポンスに含めるラベルカテゴリを指定できます。
- LabelCategoryExclusionFilters - レスポンスから除外するラベルカテゴリを指定できます。

また、必要に応じて包含フィルターと除外フィルターを組み合わせて、一部のラベルやカテゴリを除外したり、含めたりすることもできます。

IMAGE_PROPERTIES は、イメージのドミナントカラーや、鮮明度、明るさ、コントラストなどの品質属性を指します。IMAGE_PROPERTIES の検出時に、MaxDominantColors パラメータを使用して、返されるドミナントカラーの最大数を指定できます (デフォルトは 10)。

```
{
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "MaxLabels": 10,
    "MinConfidence": 75,
    "Features": [ "GENERAL_LABELS", "IMAGE_PROPERTIES" ],
    "Settings": {
        "GeneralLabels": {
            "LabelInclusionFilters": [<Label(s)>],
            "LabelExclusionFilters": [<Label(s)>],
            "LabelCategoryInclusionFilters": [<Category Name(s)>],
            "LabelCategoryExclusionFilters": [<Category Name(s)>]
        },
        "ImageProperties": {
            "MaxDominantColors":10
        }
    }
}
```

DetectLabels のレスポンス

DetectLabels からのレスポンスは、イメージ内で検出されたラベルとラベルの検出に使用された信頼度の配列です。

以下に、DetectLabels からのレスポンス例を示します。以下のサンプルレスポンスには、GENERAL_LABELS に対して返される次のようなさまざまな属性が含まれています。

- 名前 - 検出されたラベルの名前 この例では、オペレーションによって「携帯電話」というラベルの付いたオブジェクトが検出されました。

- 信頼度 - 各ラベルには信頼度が関連付けられています。この例では、ラベルの信頼度は 99.36% でした。
- 親 - 検出されたラベルの祖先ラベルです。この例では、「携帯電話」というラベルには「電話」という親ラベルが 1 つあります。
- エイリアス - ラベルに含まれている可能がある エイリアスの情報 この例では、「携帯電話」ラベルに「セルフォン」というエイリアスが含まれている可能性があります。
- カテゴリ - 検出されたラベルが属するラベルカテゴリ この例では、「テクノロジーとコンピューティング」です。

共通オブジェクトラベルに対するレスポンスは、入力画像上のラベルの位置に対する境界ボックス情報を含みます。例えば、Person ラベルには、2 つの境界ボックスを含むインスタンス配列があります。これらは、画像内で検出された 2 人の人物の位置です。

レスポンスには IMAGE_PROPERTIES に関する属性も含まれています。IMAGE_PROPERTIES 機能によって提供される属性は以下のとおりです。

- 品質 - 入力イメージの鮮明度、明るさ、コントラストに関する情報です。スコアは 0 ~ 100 です。品質で評価されるのは、イメージ全体、イメージの背景と前景 (可能な場合) です。ただし、コントラストで評価されるのはイメージ全体のみです。鮮明度と明るさでは、背景と前景も評価に含まれます。
- ドミナントカラー - イメージ内のドミナントカラーの配列です。ドミナントカラーはそれぞれ、簡略化された色名、CSS カラーパレット、RGB 値、16 進コードで記述されます。
- 前景 - 入力イメージの前景のドミナントカラー、鮮明度、明るさに関する情報です。
- 背景 - 入力イメージの背景のドミナントカラー、鮮明度、明るさに関する情報です。

GENERAL_LABELS と IMAGE_PROPERTIES を入力パラメータとして一緒に使用すると、Amazon Rekognition Image は境界ボックスを持つオブジェクトのドミナントカラーも返します。

フィールド `LabelModelVersion` には `DetectLabels` で使用される検出モデルのバージョン番号が含まれます。

```
{

   "Labels": [
        {
            "Name": "Mobile Phone",
            "Parents": [
```

```
            {
              "Name": "Phone"
            }
          ],
          "Aliases": [
            {
              "Name": "Cell Phone"
            }
          ],
          "Categories": [
            {
              "Name": "Technology and Computing"
            }
          ],
          "Confidence": 99.9364013671875,
          "Instances": [
              {
                  "BoundingBox": {
                      "Width": 0.26779675483703613,
                      "Height": 0.8562285900115967,
                      "Left": 0.3604024350643158,
                      "Top": 0.09245597571134567,
                  }
                  "Confidence": 99.9364013671875,
                  "DominantColors": [
                  {
              "Red": 120,
              "Green": 137,
              "Blue": 132,
              "HexCode": "3A7432",
              "SimplifiedColor": "red",
              "CssColor": "fuscia",
              "PixelPercentage": 40.10
                  }
                      ],
              }
          ]
      }
  ],
  "ImageProperties": {
      "Quality": {
          "Brightness": 40,
          "Sharpness": 40,
          "Contrast": 24,
```

```
        },
        "DominantColors": [
            {
                "Red": 120,
                "Green": 137,
                "Blue": 132,
                "HexCode": "3A7432",
                "SimplifiedColor": "red",
                "CssColor": "fuscia",
                "PixelPercentage": 40.10
            }
        ],
        "Foreground": {
            "Quality": {
                "Brightness": 40,
                "Sharpness": 40,
            },
            "DominantColors": [
                {
                    "Red": 200,
                    "Green": 137,
                    "Blue": 132,
                    "HexCode": "3A7432",
                    "CSSColor": "",
                    "SimplifiedColor": "red",
                    "PixelPercentage": 30.70
                }
            ],
        }
        "Background": {
            "Quality": {
                "Brightness": 40,
                "Sharpness": 40,
            },
            "DominantColors": [
                {
                    "Red": 200,
                    "Green": 137,
                    "Blue": 132,
                    "HexCode": "3A7432",
                    "CSSColor": "",
                    "SimplifiedColor": "Red",
                    "PixelPercentage": 10.20
                }
```

```
            ],
        },
    },
    "LabelModelVersion": "3.0"
}
```

DetectLabels レスポンスの変換

DetectLabels API を使用する場合、プライマリラベルとエイリアスの両方が同じリストに含まれていた古い API のレスポンス構造を模倣したレスポンス構造が必要になることがあります。

以下は、DetectLabels からの現在の API レスポンスの例です。

```
"Labels": [
        {
            "Name": "Mobile Phone",
            "Confidence": 99.99717712402344,
            "Instances": [],
            "Parents": [
                {
                "Name": "Phone"
                }
             ],
            "Aliases": [
                {
                "Name": "Cell Phone"
                }
             ]
        }
 ]
```

以下は、DetectLabels からの以前の API レスポンスの例です。

```
"Labels": [
        {
            "Name": "Mobile Phone",
            "Confidence": 99.99717712402344,
            "Instances": [],
            "Parents": [
                {
                "Name": "Phone"
                }
```

```
            ]
        },
        {
           "Name": "Cell Phone",
           "Confidence": 99.99717712402344,
           "Instances": [],
           "Parents": [
               {
               "Name": "Phone"
               }
            ]
        },
]
```

必要に応じて、現在のレスポンスを古いレスポンスの形式に従うように変換できます。次のサンプルコードを使用して、最新の API レスポンスを以前の API レスポンス構造に変換できます。

Python

次のコードサンプルは、DetectLabels API からの現在のレスポンスを変換する方法を示しています。以下のコードサンプルでは、*EXAMPLE_INFERENCE_OUTPUT* の値を、実行した DetectLabels オペレーションの出力に置き換えることができます。

```
from copy import deepcopy

LABEL_KEY = "Labels"
ALIASES_KEY = "Aliases"
INSTANCE_KEY = "Instances"
NAME_KEY = "Name"

#Latest API response sample
EXAMPLE_INFERENCE_OUTPUT = {
    "Labels": [
        {
            "Name": "Mobile Phone",
            "Confidence": 97.530106,
            "Categories": [
                {
                    "Name": "Technology and Computing"
                }
            ],
            "Aliases": [
                {
```

```
                    "Name": "Cell Phone"
                }
            ],
            "Instances":[
                {
                    "BoundingBox":{
                        "Height":0.1549897,
                        "Width":0.07747964,
                        "Top":0.50858885,
                        "Left":0.00018205095
                    },
                    "Confidence":98.401276
                }
            ]
        },
        {
            "Name": "Urban",
            "Confidence": 99.99982,
            "Categories": [
                "Colors and Visual Composition"
            ]
        }
    ]
}

def expand_aliases(inferenceOutputsWithAliases):

    if LABEL_KEY in inferenceOutputsWithAliases:
        expandInferenceOutputs = []
        for primaryLabelDict in inferenceOutputsWithAliases[LABEL_KEY]:
            if ALIASES_KEY in primaryLabelDict:
                for alias in primaryLabelDict[ALIASES_KEY]:
                    aliasLabelDict = deepcopy(primaryLabelDict)
                    aliasLabelDict[NAME_KEY] = alias[NAME_KEY]
                    del aliasLabelDict[ALIASES_KEY]
                    if INSTANCE_KEY in aliasLabelDict:
                        del aliasLabelDict[INSTANCE_KEY]
                    expandInferenceOutputs.append(aliasLabelDict)

        inferenceOutputsWithAliases[LABEL_KEY].extend(expandInferenceOutputs)

    return inferenceOutputsWithAliases
```

```
if __name__ == "__main__":

    outputWithExpandAliases = expand_aliases(EXAMPLE_INFERENCE_OUTPUT)
    print(outputWithExpandAliases)
```

変換されたレスポンスの例を以下に示します。

```
#Output example after the transformation
{
    "Labels": [
        {
            "Name": "Mobile Phone",
            "Confidence": 97.530106,
            "Categories": [
                {
                    "Name": "Technology and Computing"
                }
            ],
            "Aliases": [
                {
                    "Name": "Cell Phone"
                }
            ],
            "Instances":[
                {
                    "BoundingBox":{
                        "Height":0.1549897,
                        "Width":0.07747964,
                        "Top":0.50858885,
                        "Left":0.00018205095
                    },
                    "Confidence":98.401276
                }
            ]
        },
        {
            "Name": "Cell Phone",
            "Confidence": 97.530106,
            "Categories": [
                {
                    "Name": "Technology and Computing"
                }
```

```
            ],
            "Instances":[]
        },
        {
            "Name": "Urban",
            "Confidence": 99.99982,
            "Categories": [
                "Colors and Visual Composition"
            ]
        }
    ]
}
```

ビデオ内のラベルの検出

Amazon Rekognition Video では、ビデオ内のラベル (オブジェクトと概念) を検出して、ラベルが検出された時刻を表示できます。SDK のコード例については、「Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)」を参照してください。 AWS CLI 例については、「」を参照してくださいを使用したビデオの分析 AWS Command Line Interface。

Amazon Rekognition Video のラベル検出は、非同期オペレーションです。ビデオ内のラベルの検出を開始するには、StartLabelDetection を呼び出します。

Amazon Rekognition Video は、ビデオ分析の完了ステータスを Amazon Simple Notification Service トピックに発行します。ビデオ分析が成功したら、GetLabelDetection を呼び出して検出されたラベルを取得します。ビデオ分析の API オペレーションの詳細については、「Amazon Rekognition Video オペレーションを呼び出す」を参照してください。

StartLabelDetection リクエスト

以下に `StartLabelDetection` オペレーションのリクエストの例を示します。Amazon S3 バケットに保存されたビデオを使用して `StartLabelDetection` オペレーションを行います。リクエスト JSON の例では、Amazon S3 バケットとビデオ名が `MinConfidence`、`Features`、`Settings`、`NotificationChannel` と共に指定されています。

`MinConfidence` は、Amazon Rekognition Video がレスポンスで返すことができるようにするための、検出されたラベル、またはインスタンスの境界ボックス (検出された場合) の精度に対する最小限の信頼度です。

Features では、GENERAL_LABELS をレスポンスの一部として返すように指定できます。

Settings では、GENERAL_LABELS で返されるアイテムをフィルタリングできます。ラベルには、包含フィルターと除外フィルターを使用できます。特定のラベル、個別のラベル、またはラベルカテゴリ別にフィルタリングすることもできます。

- LabelInclusionFilters - レスポンスに含めるラベルを指定するために使用します。
- LabelExclusionFilters - レスポンスから除外するラベルを指定するために使用します。
- LabelCategoryInclusionFilters - レスポンスに含めるラベルカテゴリを指定するために使用します。
- LabelCategoryExclusionFilters - レスポンスから除外するラベルカテゴリを指定するために使用します。

また、必要に応じて包含フィルターと除外フィルターを組み合わせて、一部のラベルやカテゴリを除外したり、含めたりすることもできます。

NotificationChannel は、Amazon Rekognition Video がラベル検出オペレーションの完了ステータスを公開する Amazon SNS トピックの ARN です。AmazonRekognitionServiceRole 権限ポリシーを使用している場合、Amazon SNS トピックには、Rekognition で始まるトピック名が必要です。

以下は、フィルターを含む JSON 形式の StartLabelDetection リクエストの例です。

```
{
    "ClientRequestToken": "5a6e690e-c750-460a-9d59-c992e0ec8638",
    "JobTag": "5a6e690e-c750-460a-9d59-c992e0ec8638",
    "Video": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "video.mp4"
         }
     },
     "Features": ["GENERAL_LABELS"],
     "MinConfidence": 75,
     "Settings": {
         "GeneralLabels": {
             "LabelInclusionFilters": ["Cat", "Dog"],
             "LabelExclusionFilters": ["Tiger"],
             "LabelCategoryInclusionFilters": ["Animals and Pets"],
             "LabelCategoryExclusionFilters": ["Popular Landmark"]
```

```
        }
    },
    "NotificationChannel": {
        "RoleArn": "arn:aws:iam::012345678910:role/SNSAccessRole",
        "SNSTopicArn": "arn:aws:sns:us-east-1:012345678910:notification-topic",
    }
}
```

GetLabelDetection オペレーションレスポンス

GetLabelDetection は、ビデオ内で検出されたラベルに関する情報が含まれた配列 (Labels) を返します。配列は、時間または SortBy パラメータを指定したとき検出されたラベルでソートできます。また、AggregateBy パラメータを使用してレスポンス項目の集計方法を選択することもできます。

次は、GetLabelDetection の JSON レスポンス例です。レスポンスで、以下の点に注意してください。

- 並べ替え順 – 返されるラベルの配列は時刻別にソートされます。ラベル別に並べ替えるには、GetLabelDetection の SortBy 入力パラメータに NAME を指定します。ビデオ内でラベルが複数回表示されている場合、(LabelDetection) 要素のインスタンスが複数存在しています。既定の並べ替え順序は TIMESTAMP で、副次的な並べ替え順序は NAME です。
- ラベル情報 – LabelDetection 配列要素には (ラベル) オブジェクトが含まれ、そこにはラベル名と、Amazon Rekognition が検出したラベルの精度に対する信頼度が含まれています。Label オブジェクトには、ラベルの階層分類と、一般的なラベルの境界ボックス情報も含まれます。Timestamp は、ラベルが検出された時間であり、ビデオの開始時刻からの経過ミリ秒数として定義されます。

  ラベルに関連付けられているカテゴリやエイリアスに関する情報も返されます。ビデオ SEGMENTS ごとに集計された結果では、StartTimestampMillis、EndTimestampMillis、DurationMillis 構造が返され、それぞれセグメントの開始時間、終了時間、持続時間を定義します。
- 集計 — 結果が返されたときの集計方法を指定します。デフォルトでは TIMESTAMPS によって集計されます。また、SEGMENTS による集計を選択することもできます。この方法では、時間枠の結果が集計されます。SEGMENTS による集計の場合、境界ボックス付きの検出されたインスタンスに関する情報は返されません。このセグメント中に検出されたラベルのみが返されます。

- ページング情報 – この例では、ラベル検出情報が記載された 1 ページを示しています。GetLabelDetection の MaxResults 入力パラメータには、返す LabelDetection オブジェクトの数を指定できます。結果が MaxResults を超える場合、GetLabelDetection は次のページの結果を取得するためのトークン (NextToken) を返します。詳細については、「Amazon Rekognition Video の分析結果を取得する」を参照してください。
- ビデオの情報 – レスポンスには、VideoMetadata から返される情報のページごとにビデオ形式に関する情報 (GetLabelDetection) が含まれます。

以下は、TIMESTAMPS による集計を含む JSON 形式の GetLabelDetection レスポンスのサンプルです。

```
{
    "JobStatus": "SUCCEEDED",
    "LabelModelVersion": "3.0",
    "Labels": [
        {
            "Timestamp": 1000,
            "Label": {
                "Name": "Car",
                "Categories": [
                  {
                    "Name": "Vehicles and Automotive"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Automobile"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Vehicle"
                  }
                ],
                "Confidence": 99.9364013671875, // Classification confidence
                "Instances": [
                    {
                        "BoundingBox": {
                            "Width": 0.26779675483703613,
                            "Height": 0.8562285900115967,
                            "Left": 0.3604024350643158,
```

```
                        "Top": 0.09245597571134567
                    },
                    "Confidence": 99.9364013671875 // Detection confidence
                }
            ]
        }
    },
    {
        "Timestamp": 1000,
        "Label": {
            "Name": "Cup",
            "Categories": [
              {
                "Name": "Kitchen and Dining"
              }
            ],
            "Aliases": [
              {
                "Name": "Mug"
              }
            ],
            "Parents": [],
            "Confidence": 99.9364013671875, // Classification confidence
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.26779675483703613,
                        "Height": 0.8562285900115967,
                        "Left": 0.3604024350643158,
                        "Top": 0.09245597571134567
                    },
                    "Confidence": 99.9364013671875 // Detection confidence
                }
            ]
        }
    },
    {
        "Timestamp": 2000,
        "Label": {
            "Name": "Kangaroo",
            "Categories": [
              {
                "Name": "Animals and Pets"
              }
```

```
                ],
                "Aliases": [
                  {
                    "Name": "Wallaby"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Mammal"
                  }
                ],
                "Confidence": 99.9364013671875,
                "Instances": [
                    {
                        "BoundingBox": {
                            "Width": 0.26779675483703613,
                            "Height": 0.8562285900115967,
                            "Left": 0.3604024350643158,
                            "Top": 0.09245597571134567,
                        },
                        "Confidence": 99.9364013671875
                    }
                ]
            }
        },
        {
            "Timestamp": 4000,
            "Label": {
                "Name": "Bicycle",
                "Categories": [
                  {
                    "Name": "Hobbies and Interests"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Bike"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Vehicle"
                  }
                ],
```

```
                "Confidence": 99.9364013671875,
                "Instances": [
                    {
                        "BoundingBox": {
                            "Width": 0.26779675483703613,
                            "Height": 0.8562285900115967,
                            "Left": 0.3604024350643158,
                            "Top": 0.09245597571134567
                        },
                        "Confidence": 99.9364013671875
                    }
                ]
            }
        }
    ],
    "VideoMetadata": {
        "ColorRange": "FULL",
        "DurationMillis": 5000,
        "Format": "MP4",
        "FrameWidth": 1280,
        "FrameHeight": 720,
        "FrameRate": 24
    }
}
```

以下は、SEGMENTS による集計を含む JSON 形式の GetLabelDetection レスポンスのサンプルです。

```
{
    "JobStatus": "SUCCEEDED",
    "LabelModelVersion": "3.0",
    "Labels": [
        {
            "StartTimestampMillis": 225,
            "EndTimestampMillis": 3578,
            "DurationMillis": 3353,
            "Label": {
                "Name": "Car",
                "Categories": [
                  {
                    "Name": "Vehicles and Automotive"
                  }
                ],
```

```
                "Aliases": [
                  {
                    "Name": "Automobile"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Vehicle"
                  }
                ],
                "Confidence": 99.9364013671875 // Maximum confidence score for Segment
mode
            }
        },
        {
            "StartTimestampMillis": 7578,
            "EndTimestampMillis": 12371,
            "DurationMillis": 4793,
            "Label": {
                "Name": "Kangaroo",
                "Categories": [
                  {
                    "Name": "Animals and Pets"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Wallaby"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Mammal"
                  }
                ],
                "Confidence": 99.9364013671875
            }
        },
        {
            "StartTimestampMillis": 22225,
            "EndTimestampMillis": 22578,
            "DurationMillis": 2353,
            "Label": {
                "Name": "Bicycle",
```

```
                "Categories": [
                  {
                    "Name": "Hobbies and Interests"
                  }
                ],
                "Aliases": [
                  {
                    "Name": "Bike"
                  }
                ],
                "Parents": [
                  {
                    "Name": "Vehicle"
                  }
                ],
                "Confidence": 99.9364013671875
            }
        }
    ],
    "VideoMetadata": {
        "ColorRange": "FULL",
        "DurationMillis": 5000,
        "Format": "MP4",
        "FrameWidth": 1280,
        "FrameHeight": 720,
        "FrameRate": 24
    }
}
```

GetLabelDetection レスポンスの変換

GetLabelDetection API オペレーションを実行して結果を取得する場合、プライマリラベルとエイリアスの両方が同じリストに含まれていた古い API レスポンス構造を模倣したレスポンス構造が必要になることがあります。

前のセクションにある JSON レスポンスの例には、GetLabelDetection からの API レスポンスの現在の形式が表示されています。

次の例は、GetLabelDetection API からのレスポンスを示しています。

```
{
    "Labels": [
        {
```

```
            "Timestamp": 0,
            "Label": {
                "Instances": [],
                "Confidence": 60.51791763305664,
                "Parents": [],
                "Name": "Leaf"
            }
        },
        {
            "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": 99.63411102294922,
                "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
    }
}
```

必要に応じて、現在のレスポンスを古いレスポンスの形式に従うように変換できます。次のサンプルコードを使用して、最新の API レスポンスを以前の API レスポンス構造に変換できます。

```
from copy import deepcopy

VIDEO_LABEL_KEY = "Labels"
LABEL_KEY = "Label"
ALIASES_KEY = "Aliases"
INSTANCE_KEY = "Instances"
NAME_KEY = "Name"
```

```
#Latest API response sample for AggregatedBy SEGMENTS
EXAMPLE_SEGMENT_OUTPUT = {
    "Labels": [
        {
            "Timestamp": 0,
            "Label":{
                "Name": "Person",
                "Confidence": 97.530106,
                "Parents": [],
                "Aliases": [
                    {
                        "Name": "Human"
                    },
                ],
                "Categories": [
                    {
                        "Name": "Person Description"
                    }
                ],
            },
            "StartTimestampMillis": 0,
            "EndTimestampMillis": 500666,
            "DurationMillis": 500666
        },
        {
            "Timestamp": 6400,
            "Label": {
                "Name": "Leaf",
                "Confidence": 89.77790069580078,
                "Parents": [
                    {
                        "Name": "Plant"
                    }
                ],
                "Aliases": [],
                "Categories": [
                    {
                        "Name": "Plants and Flowers"
                    }
                ],

            },
            "StartTimestampMillis": 6400,
            "EndTimestampMillis": 8200,
```

```
            "DurationMillis": 1800
        },
    ]
}

#Output example after the transformation for AggregatedBy SEGMENTS
EXPECTED_EXPANDED_SEGMENT_OUTPUT = {
    "Labels": [
        {
            "Timestamp": 0,
            "Label":{
                "Name": "Person",
                "Confidence": 97.530106,
                "Parents": [],
                "Aliases": [
                    {
                        "Name": "Human"
                    },
                ],
                "Categories": [
                    {
                        "Name": "Person Description"
                    }
                ],
            },
            "StartTimestampMillis": 0,
            "EndTimestampMillis": 500666,
            "DurationMillis": 500666
        },
        {
            "Timestamp": 6400,
            "Label": {
                "Name": "Leaf",
                "Confidence": 89.77790069580078,
                "Parents": [
                    {
                        "Name": "Plant"
                    }
                ],
                "Aliases": [],
                "Categories": [
                    {
                        "Name": "Plants and Flowers"
                    }
```

```
                ],

            },
            "StartTimestampMillis": 6400,
            "EndTimestampMillis": 8200,
            "DurationMillis": 1800
        },
        {
            "Timestamp": 0,
            "Label":{
                "Name": "Human",
                "Confidence": 97.530106,
                "Parents": [],
                "Categories": [
                    {
                        "Name": "Person Description"
                    }
                ],
            },
            "StartTimestampMillis": 0,
            "EndTimestampMillis": 500666,
            "DurationMillis": 500666
        },
    ]
}

#Latest API response sample for AggregatedBy TIMESTAMPS
EXAMPLE_TIMESTAMP_OUTPUT = {
    "Labels": [
        {
            "Timestamp": 0,
            "Label": {
                "Name": "Person",
                "Confidence": 97.530106,
                "Instances": [
                    {
                        "BoundingBox": {
                            "Height": 0.1549897,
                            "Width": 0.07747964,
                            "Top": 0.50858885,
                            "Left": 0.00018205095
                        },
                        "Confidence": 97.530106
                    },
```

```
                ],
                "Parents": [],
                "Aliases": [
                    {
                        "Name": "Human"
                    },
                ],
                "Categories": [
                    {
                        "Name": "Person Description"
                    }
                ],
            },
        },
        {
            "Timestamp": 6400,
            "Label": {
                "Name": "Leaf",
                "Confidence": 89.77790069580078,
                "Instances": [],
                "Parents": [
                    {
                        "Name": "Plant"
                    }
                ],
                "Aliases": [],
                "Categories": [
                    {
                        "Name": "Plants and Flowers"
                    }
                ],
            },
        },
    ]
}

#Output example after the transformation for AggregatedBy TIMESTAMPS
EXPECTED_EXPANDED_TIMESTAMP_OUTPUT = {
    "Labels": [
        {
            "Timestamp": 0,
            "Label": {
                "Name": "Person",
                "Confidence": 97.530106,
```

```
                "Instances": [
                    {
                        "BoundingBox": {
                            "Height": 0.1549897,
                            "Width": 0.07747964,
                            "Top": 0.50858885,
                            "Left": 0.00018205095
                        },
                        "Confidence": 97.530106
                    },
                ],
                "Parents": [],
                "Aliases": [
                    {
                        "Name": "Human"
                    },
                ],
                "Categories": [
                    {
                        "Name": "Person Description"
                    }
                ],
            },
        },
        {
            "Timestamp": 6400,
            "Label": {
                "Name": "Leaf",
                "Confidence": 89.77790069580078,
                "Instances": [],
                "Parents": [
                    {
                        "Name": "Plant"
                    }
                ],
                "Aliases": [],
                "Categories": [
                    {
                        "Name": "Plants and Flowers"
                    }
                ],
            },
        },
        {
```

```
            "Timestamp": 0,
            "Label": {
                "Name": "Human",
                "Confidence": 97.530106,
                "Parents": [],
                "Categories": [
                    {
                        "Name": "Person Description"
                    }
                ],
            },
        },
    ]
}

def expand_aliases(inferenceOutputsWithAliases):

    if VIDEO_LABEL_KEY in inferenceOutputsWithAliases:
        expandInferenceOutputs = []
        for segmentLabelDict in inferenceOutputsWithAliases[VIDEO_LABEL_KEY]:
            primaryLabelDict = segmentLabelDict[LABEL_KEY]
            if ALIASES_KEY in primaryLabelDict:
                for alias in primaryLabelDict[ALIASES_KEY]:
                    aliasLabelDict = deepcopy(segmentLabelDict)
                    aliasLabelDict[LABEL_KEY][NAME_KEY] = alias[NAME_KEY]
                    del aliasLabelDict[LABEL_KEY][ALIASES_KEY]
                    if INSTANCE_KEY in aliasLabelDict[LABEL_KEY]:
                        del aliasLabelDict[LABEL_KEY][INSTANCE_KEY]
                    expandInferenceOutputs.append(aliasLabelDict)

        inferenceOutputsWithAliases[VIDEO_LABEL_KEY].extend(expandInferenceOutputs)

    return inferenceOutputsWithAliases


if __name__ == "__main__":

    segmentOutputWithExpandAliases = expand_aliases(EXAMPLE_SEGMENT_OUTPUT)
    assert segmentOutputWithExpandAliases == EXPECTED_EXPANDED_SEGMENT_OUTPUT

    timestampOutputWithExpandAliases = expand_aliases(EXAMPLE_TIMESTAMP_OUTPUT)
    assert timestampOutputWithExpandAliases == EXPECTED_EXPANDED_TIMESTAMP_OUTPUT
```

ストリーミングビデオイベント内のラベルの検出

Amazon Rekognition Video を使用して、ストリーミングビデオ内のラベルを検出できます。これを行うには、ストリーミングビデオの分析を開始および管理するためのストリームプロセッサ (CreateStreamProcessor) を作成します。

Amazon Rekognition Video は、Amazon Kinesis Video Streams を使用して、ビデオストリームを受信し処理します。ストリームプロセッサを作成するときに、ストリームプロセッサで検出する内容を選択します。人、パッケージ、ペット、または人とパッケージを選択できます。分析結果は、Amazon S3 バケットと Amazon SNS 通知に出力されます。Amazon Rekognition Video はビデオ内の人物の存在を検出しますが、その人物が特定の個人であるかどうかは検出しないことに注意してください。ストリーミングビデオのコレクションから顔を検索するには、「the section called “ストリーミングビデオのコレクション内での顔検索”」を参照してください。

ストリーミングビデオで Amazon Rekognition Video を使用するには、アプリケーションに以下が必要です。

- Amazon Rekognition Video にストリーミングビデオを送信する Kinesis ビデオストリーム。詳細については、「Amazon Kinesis Video Streams デベロッパーガイド」を参照してください。
- ストリーミングビデオの分析を管理するための Amazon Rekognition Video ストリームプロセッサ。詳細については、「Amazon Rekognition Video ストリームプロセッサオペレーションの概要」を参照してください。
- Amazon S3 バケット。Amazon Rekognition Video はセッション出力を S3 バケットに公開します。出力には、対象の人物またはオブジェクトが初めて検出されたイメージフレームが含まれます。S3 バケットを所有している必要があります。
- Amazon Rekognition Video がスマートアラートとセッション終了の概要を公開する Amazon SNS トピック。

トピック

- Amazon Rekognition Video と Amazon Kinesis のリソースを設定する
- ストリーミングビデオイベントに対するラベル検出オペレーションの呼び出し

Amazon Rekognition Video と Amazon Kinesis のリソースを設定する

次の手順では、ストリーミングビデオ内のラベルの検出に使用する Kinesis ビデオストリームとその他のリソースをプロビジョニングするためのステップを説明します。

前提条件

この手順を実行するには、 AWS SDK for Java をインストールする必要があります。詳細については、「Amazon Rekognition の開始方法」を参照してください。使用する AWS アカウントには、Amazon Rekognition API へのアクセス権限が必要です。詳細については、IAM ユーザーガイドの「Amazon Rekognition で定義されるアクション」を参照してください。

ビデオストリーム内のラベルを検出するには (AWS SDK)

1. Amazon S3 バケットを作成する。バケット名と、使用するキープレフィックスをメモします。この情報は後で使用します。
2. Amazon SNS トピックを作成します。これを使用して、ビデオストリームで対象オブジェクトが初めて検出されたときに通知を受け取れます。トピックの Amazon リソースネーム (ARN) をメモします。詳細については、Amazon SNS デベロッパーガイドの「Amazon SNS トピックを作成する」を参照してください。
3. エンドポイントを Amazon SNS トピックにサブスクライブします。詳細については、Amazon SNS デベロッパーガイドの「Amazon SNS トピックへサブスクライブする」を参照してください。
4. Kinesis ビデオストリームを作成して、そのストリームの Amazon リソースネーム (ARN) をメモします。
5. まだ作成していない場合は、Amazon Rekognition Video に Kinesis ビデオストリーム、S3 バケット、Amazon SNS トピックへのアクセス権を付与する IAM サービスロールを作成します。詳細については、「ラベル検出ストリームプロセッサへのアクセス権を付与する」を参照してください。

その後、ラベル検出ストリームプロセッサを作成し、選択したストリームプロセッサ名を使用してストリームプロセッサを起動できます。

> **Note**
>
> ストリームプロセッサは、Kinesis ビデオストリームにメディアを取り込めることを確認した後に起動してください。

カメラの向きとセットアップ

Amazon Rekognition Video ストリーミングビデオイベントでは、Kinesis ビデオストリームでサポートされているすべてのカメラをサポートできます。最良の結果を得るには、カメラを地面と 0 ~ 45 度に配置することをお勧めします。カメラは標準的な直立状態にある必要があります。例えば、フレーム内に人物がいる場合、人物の向きは垂直で、頭は足よりも高い位置にある必要があります。

ラベル検出ストリームプロセッサへのアクセス権を付与する

AWS Identity and Access Management (IAM) サービスロールを使用して、Amazon Rekognition Video に Kinesis ビデオストリームへの読み取りアクセスを許可します。これを行うには、IAM ロールを使用して、Amazon Rekognition Video に Amazon S3 バケットと Amazon SNS トピックへのアクセス権を付与します。

Amazon Rekognition Video が既存の Amazon SNS トピック、Amazon S3 バケット、Kinesis ビデオストリームにアクセスできるようにするアクセス許可ポリシーを作成できます。を使用した手順については AWS CLI、step-by-step「」を参照してくださいthe section called “AWS CLI ラベル検出 IAM ロールを設定する コマンド”。

Amazon Rekognition Video がラベル検出のためにリソースにアクセスできるようにするには

1. [IAM JSON ポリシーエディターを使用して新しい許可ポリシーを作成し]、次のポリシーを使用します。kvs-stream-name を Kinesis ビデオストリームの名前に置き換えます。topicarn は、使用する Amazon SNS トピックの Amazon リソースネーム (ARN) に置き換え、bucket-name は Amazon S3 バケットの名前に置き換えます。

```
{
    "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. IAM サービスロールを作成する または、既存の IAM サービスロールを更新します。次の情報を使用して、IAM サービスロールを作成します。

   1. サービス名の Rekognition を選択します。
   2. サービスロールのユースケースの Rekognition を選択します。
   3. 手順 1 で作成したアクセス権ポリシーを添付します。

3. サービスロールの ARN をメモしておきます。ビデオ分析オペレーションを実行する前に、ストリームプロセッサを作成する必要があります。

4. (オプション) 独自の AWS KMS キーを使用して S3 バケットに送信されるデータを暗号化する場合は、IAM ロールで次のステートメントを追加する必要があります。(これはキーポリシー用に作成した IAM ロールであり、使用するカスタマーマネージドキーに対応します。)

```
            {
                        "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 ラベル検出 IAM ロールを設定する コマンド

まだ設定していない場合は、認証情報 AWS CLI を使用して をセットアップして設定します。

次のコマンドを に入力 AWS CLI して、ラベル検出に必要なアクセス許可を持つ IAM ロールを設定します。

1. `export IAM_ROLE_NAME=labels-test-role`
2. `export AWS_REGION=us-east-1`
3. 以下の内容を含む信頼関係ポリシーファイル (assume-role-rekognition.json など) を作成します。

```
{
  "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. 通知を受け取りたい SNS トピックの名前が「AmazonRekognition」プレフィックスで始まっていない場合は、次のポリシーを追加してください。

   ```
   aws iam attach-role-policy --role-name $IAM_ROLE_NAME --policy-arn "arn:aws:iam::aws:policy/AmazonSNSFullAccess" --region $AWS_REGION
   ```

7. S3 バケットに送信されるデータを暗号化するために独自の AWS KMS キーを使用する場合は、使用するカスタマーマネージドキーのキーポリシーを更新します。

   a. kms_key_policy.json ファイルを作成します。

   ```
   {
   "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. その内容には、`export KMS_KEY_ID=labels-kms-key-id` を含めます。KMS_KEY_ID は、作成した KMS キー ID に置き換えてください。

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

ストリーミングビデオイベントに対するラベル検出オペレーションの呼び出し

Amazon Rekognition Video は、ストリーミングビデオ内の人または関連オブジェクトを検出し、検出されると通知を送信できます。ラベル検出ストリームプロセッサを作成するときは、Amazon Rekognition Video で検出したいラベルを選択します。ラベルには、人、パッケージとペット、また

は人、パッケージ、ペットがあります。検出したい特定のラベルのみを選択してください。これにより、関連するラベルのみが通知を作成します。ビデオ情報を保存するタイミングを決定するオプションを設定し、フレーム内で検出されたラベルに基づいて追加の処理を行えます。

リソースをセットアップした後、ストリーミングビデオ内のラベルを検出するプロセスは次のとおりです。

1. ストリームプロセッサを作成する
2. ストリームプロセッサを開始する
3. 対象オブジェクトが検出されると、対象の各オブジェクトの最初の出現に対して Amazon SNS 通知を受け取ります。
4. ストリームプロセッサは、`MaxDurationInSeconds` で指定された時間が経過すると停止します。
5. イベントの概要が記載された最終的な Amazon SNS 通知を受け取ります。
6. Amazon Rekognition Video は、セッションの詳細な概要を S3 バケットに公開します。

トピック

- Amazon Rekognition Video ラベル検出ストリームプロセッサを作成する
- Amazon Rekognition Video ラベル検出ストリームプロセッサの起動
- ラベル検出結果の分析

Amazon Rekognition Video ラベル検出ストリームプロセッサを作成する

ストリーミングビデオを分析する前に、Amazon Rekognition Video ストリームプロセッサ (CreateStreamProcessor) を作成します。

対象となるラベルや人を検出するストリームプロセッサを作成する場合は、Kinesis ビデオストリーム (`Input`)、Amazon S3 バケット情報 (`Output`)、Amazon SNS トピック ARN (`StreamProcessorNotificationChannel`) を入力として指定します。Amazon S3 バケットに送信されるデータを暗号化する KMS キー ID を指定することもできます。人、パッケージと人、またはペット、人、パッケージなど、`Settings` で検出する内容を指定します。Amazon Rekognition が `RegionsOfInterest` を使用してモニタリングするフレーム内の箇所を指定することもできます。`CreateStreamProcessor` リクエストに関する JSON の例を次に示します。

```
{
  "DataSharingPreference": { "OptIn":TRUE
  },
  "Input": {
    "KinesisVideoStream": {
      "Arn": "arn:aws:kinesisvideo:us-east-1:nnnnnnnnnnnn:stream/muh_video_stream/
nnnnnnnnnnnn"
    }
  },
  "KmsKeyId": "muhkey",
  "Name": "muh-default_stream_processor",
  "Output": {
    "S3Destination": {
      "Bucket": "s3bucket",
      "KeyPrefix": "s3prefix"
    }
  },
  "NotificationChannel": {
    "SNSTopicArn": "arn:aws:sns:us-east-2:nnnnnnnnnnn: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
        }
      ]
    }
  ]
}
```

MinConfidence 値は、ストリームプロセッサの ConnectedHomeSettings を指定するときに変更できます。MinConfidence は 0 ~ 100 の数値で、アルゴリズムの予測がどの程度確実であるかを示します。例えば、信頼度が 90 の person の通知は、アルゴリズムが、その人物がビデオに存在することを確信していることを意味します。信頼値が 10 の場合は、人物が存在する可能性があるということです。通知を受ける頻度に応じて、MinConfidence を 0 ~ 100 の任意の値に設定できます。例えば、Rekognition がビデオフレームにパッケージが含まれていると確信した場合にのみ通知を受け取りたい場合は、MinConfidence を 90 に設定できます。

MinConfidence は、デフォルトで 50 に設定されます。アルゴリズムを最適化して精度を高めたい場合は、MinConfidence を 50 より大きい値に設定できます。これにより、通知を受け取る回数は少なくなりますが、各通知の信頼性は高まります。アルゴリズムを最適化してリコールを高めたい場合は、MinConfidence を 50 未満に設定すれば、より多くの通知を受け取れます。

Amazon Rekognition Video ラベル検出ストリームプロセッサの起動

CreateStreamProcessor で指定したストリームプロセッサの名前を使用して StartStreamProcessor を呼び出し、ストリーミングビデオの分析を開始します。ラベル検出ストリームプロセッサで StartStreamProcessor オペレーションを実行する場合は、開始情報と停止情報を入力して処理時間を決定します。

ストリームプロセッサが起動すると、ラベル検出ストリームプロセッサの状態は次のように変化します。

1. StartStreamProcessor を呼び出すと、ラベル検出ストリームのプロセッサの状態は STOPPED または FAILED から STARTING に変化します。

2. ラベル検出ストリームプロセッサが実行されている間は、状態は STARTING になります。
3. ラベル検出ストリームプロセッサの実行が終了すると、状態は STOPPED または FAILED になります。

StartSelector は、処理を開始する Kinesis ストリームの開始点を指定します。KVS Producer のタイムスタンプまたは KVS フラグメント番号を使用できます。詳細については、「Fragment」を参照してください。

> **Note**
>
> KVS Producer のタイムスタンプを使用する場合は、時間をミリ秒単位で入力する必要があります。

StopSelector は、ストリームの処理を停止するタイミングを指定します。ビデオを処理する最大時間を指定できます。デフォルトでは、最大継続時間が 10 秒に設定されています。個々の KVS フラグメントのサイズによっては、実際の処理時間が最大継続時間より少し長くなる場合があることに注意してください。フラグメントの終端で最大継続時間に達したか、それを超えた場合、処理時間は停止します。

StartStreamProcessor リクエストに関する JSON の例を次に示します。

```
{
   "Name": "string",
   "StartSelector": {
     "KVSStreamStartSelector": {
         "KVSProducerTimestamp": 1655930623123
      },
        "StopSelector": {
            "MaxDurationInSeconds": 11
      }
   }
}
```

ストリームプロセッサが正常に起動した場合は、HTTP 200 レスポンスが返されます。空の JSON 本文が含まれています。

ラベル検出結果の分析

Amazon Rekognition Video がラベル検出ストリームプロセッサから通知を公開する方法は 3 つあります。オブジェクト検出イベントの Amazon SNS 通知、セッション終了の概要に関する Amazon SNS 通知、詳細な Amazon S3 バケットレポートです。

- オブジェクト検出イベントの Amazon SNS 通知

  ビデオストリームでラベルが検出されると、オブジェクト検出イベントの Amazon SNS 通知を受け取ります。Amazon Rekognition は、ビデオストリームで対象オブジェクトまたは人物が初めて検出されたときに通知を公開します。通知には、検出されたラベルのタイプ、信頼度、ヒーローイメージへのリンクなどの情報が含まれています。また、検出された人物またはオブジェクトのトリミングされたイメージや検出タイムスタンプも含まれています。この通知の形式は次のとおりです。

```
{"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 セッション終了の概要

  ストリーム処理セッションが終了すると、Amazon SNS 通知も送信されます。この通知には、セッションのメタデータがリスト表示されます。また、処理されたストリームの継続時間などの詳細情報が含まれています。この通知の形式は次のとおりです。

```
{"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 バケットレポート

  Amazon Rekognition Video は、ビデオ分析オペレーションの詳細な推論結果を、`CreateStreamProcessor` オペレーションで提供された Amazon S3 バケットに公開します。これらの結果には、対象のオブジェクトや人物が初めて検出されたイメージフレームが含まれています。

  フレームは S3 パス ObjectKeyPrefix/StreamProcessorName/SessionId/*`service_determined_unique_path`* で入手できます。このパスでは、LabelKeyPrefix はお客様指定のオプション引数、StreamProcessorName はストリームプロセッサリソースの名前、SessionId はストリーム処理セッションの固有の ID です。これらは状況に応じて置き換えてください。

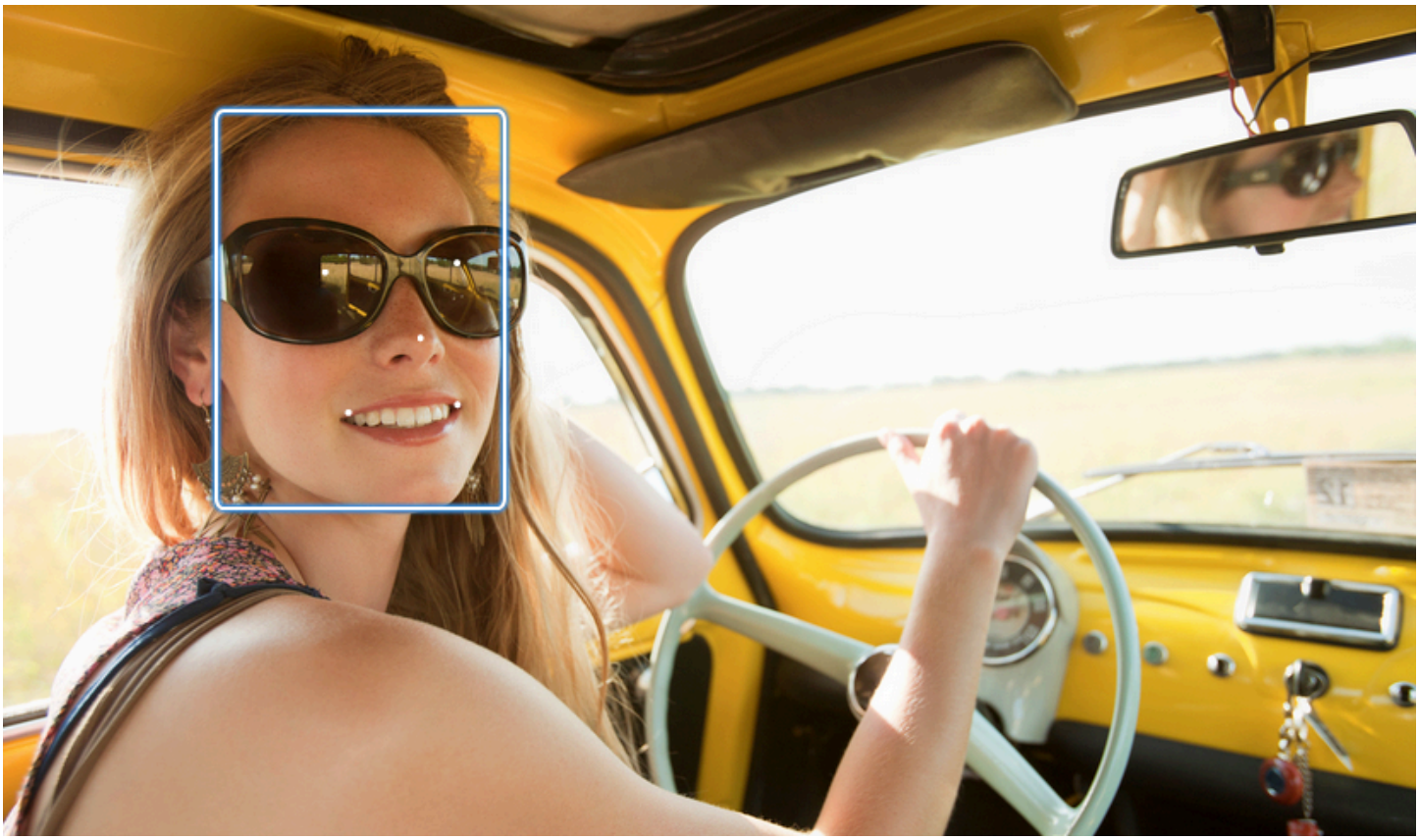
カスタムラベルの検出

Amazon Rekognition カスタムラベルにより、ビジネスニーズに固有のイメージ内のオブジェクトやシーン (ロゴやエンジニアリング機械部品など) を識別できます。詳細については、[Amazon Rekognition カスタムラベル デベロッパー ガイド] の [Amazon Rekognition カスタムラベルについて] を参照してください。

顔の検出と分析

Amazon Rekognition には、画像やビデオ内の顔を検出し分析するために使用できる API が用意されています。このセクションでは、顔分析のための非ストレージ型オペレーションの概要を示します。これらのオペレーションには、顔のランドマーク検出、感情分析、顔比較などの機能が含まれます。

Amazon Rekognition は、顔のランドマーク (目の位置など) を識別し、感情 (喜びや悲しみなど)、その他の属性 (メガネの有無、顔の遮蔽状態など) を検出できます。顔が検出されると、システムは顔の属性を分析し、各属性の信頼スコアを返します。



このセクションでは、画像とビデオの両方のオペレーションの例を示します。

Rekognition の画像オペレーションの使用の詳細については、「イメージの操作」を参照してください。

Rekognition のビデオオペレーションの使用の詳細については、「保存済み動画分析での作業」を参照してください。

なお、これらは非ストレージ型オペレーションです。ストレージ型オペレーションと顔コレクションを使用して、画像内で検出された顔のメタデータを保存できます。保存した顔は、後でイメージとビ

デオの両方で検索できます。たとえば、ビデオで特定の人物を検索できます。詳細については、「コレクション内での顔の検索」を参照してください。

詳細については、「Amazon Rekognition のよくある質問」の「顔」セクションを参照してください。

> **Note**
>
> Amazon Rekognition イメージ と Amazon Rekognition Video で使用される顔検出モデルは、漫画/アニメーションキャラクターや人間以外のエンティティの顔の検出をサポートしていません。イメージまたはビデオで漫画/アニメーションキャラクターを検出する場合は、Amazon Rekognition カスタムラベルを使用することをお勧めします。詳細については、[Amazon Rekognition カスタムラベル デベロッパー ガイド] を参照してください。

トピック

- 顔検出および顔比較の概要
- 顔属性のガイドライン
- イメージ内の顔の検出
- イメージ間の顔の比較
- 保存済みビデオ内の顔の検出

顔検出および顔比較の概要

Amazon Rekognition では、顔の含まれる画像に対し、2 つの主要な機械学習アプリケーションとして、顔検出と顔比較を利用できます。これらは、顔分析や本人確認などの重要な機能を強化し、セキュリティから個人の写真整理まで、さまざまな用途で不可欠なものとなっています。

顔検出

顔検出システムは、「この写真に顔はありますか?」という質問に対応します。顔検出の重要な側面は以下のとおりです。

- 位置と向き: 画像またはビデオフレーム内の顔の存在、位置、大きさ、向きを判定します。
- 顔属性 : 性別、年齢、ひげなどの顔属性に関係なく顔を検出します。
- 追加情報: 顔の遮蔽状態や視線の方向の詳細を示します。

顔比較

顔比較システムは、「この画像内の顔は別の画像内の顔と一致しますか?」という質問に対応します。顔比較システムには以下のような機能があります。

- 顔照合予測: 画像内の顔を、指定したデータベース内の顔と比較して、一致を予測します。
- 顔属性処理: 表情、ひげ、年齢に関係なく顔を比較できるように顔属性を処理します。

信頼スコアと検出漏れ

顔検出システムでも顔比較システムでも信頼スコアを使用します。信頼スコアは、顔の存在や顔の一致などの予測の尤度を示します。このスコアが高いほど、尤度が高くなります。例えば、信頼スコアが 90% の場合は 60% の場合よりも、検出が正しいまたは一致の確率が高いことを示します。

検出システムが顔を適切に検出しなかった場合、または実在する顔に対して低い信頼度の予測を行った場合、これは検出漏れ/偽陰性となります。システムが顔の存在を高い信頼度で誤って予測した場合、これは誤アラーム/偽陽性となります。

同様に、顔比較システムでは、同一人物の顔を別人と予測してしまう場合 (検出漏れ/偽陰性) や、異なる人物の顔を同一人物と誤って予測してしまう場合 (誤アラーム/偽陽性) があります。

アプリケーションの設計としきい値の設定

- 結果を返すために必要な最小信頼度を指定するしきい値を設定できます。適切な信頼度しきい値を選択することは、システム出力に基づくアプリケーション設計と意思決定に不可欠です。
- 選択した信頼度はユースケースを反映している必要があります。ユースケースと信頼度しきい値の例を以下に示します。
- • 写真アプリケーション: 写真内の家族を識別する場合などは、より低いしきい値 (80% など) で十分でしょう。
  - 高リスクシナリオ: セキュリティアプリケーションなど、検出漏れや誤アラームのリスクが高いユースケースでは、システムではより高い信頼度を使用する必要があります。このような場合、正確な顔照合のためには、より高いしきい値 (99% など) が推奨されます。

信頼度しきい値を設定および理解するための詳細については、「コレクション内での顔の検索」を参照してください。

顔属性のガイドライン

Amazon Rekognition が顔属性を処理して返す方法の詳細を以下に示します。

- FaceDetail オブジェクト: 検出された顔ごとに、FaceDetail オブジェクトが返されます。この FaceDetail には、顔のランドマーク、画質、姿勢などのデータが含まれています。
- 属性予測 : 感情、性別、年齢などの属性が予測されます。予測ごとに信頼度が割り当てられ、予測はそれぞれの信頼スコアとともに返されます。機密性の高いユースケースでは、99% の信頼度しきい値が推奨されます。年齢推測では、予測された年齢層の中間値が最適な近似値となります。

なお、性別と感情の予測は見た目に基づいており、実際の性自認や感情の判断には使用しないでください。性別二者択一 (男性/女性) の予測は、特定のイメージ内の顔の物理的な外観に基づいています。人物の性自認を示すものではないため、性別の判断に Rekognition を使用しないでください。性別二者択一の予測を使用して、個人の権利、プライバシー、またはサービスへのアクセスに影響する決定を行うことはお勧めしません。同様に、感情の予測は人物の実際の内面的な感情を示すものではないため、感情の判断に Rekognition を使用しないでください。写真で幸せそうな顔を装っている人物は、幸せそうに見えても、実際には幸せと感じていない可能性があります。

用途とユースケース

これらの属性の実際の用途とユースケースを以下に示します。

- 用途: 笑顔、ポーズ、シャープネスなどの属性は、プロフィール写真の選択や匿名での人口統計推測に利用できます。
- 一般的なユースケース: ソーシャルメディアアプリケーション、およびイベントや小売店での人口統計推測が典型的な例です。

各属性の詳細については、「FaceDetail」を参照してください。

イメージ内の顔の検出

Amazon Rekognition Image には、目、鼻、口などの主な顔の特徴を探して入力イメージ内の顔を検出する DetectFaces オペレーションがあります。Amazon Rekognition Image は、イメージ内の 100 の大きい顔を検出します。

入力イメージとして、イメージのバイト配列 (base64 でエンコードされたイメージのバイト) を指定するか、Amazon S3 オブジェクトを指定することができます。次の手順では、イメージ (JPEG または PNG) を S3 バケットにアップロードし、オブジェクトのキー名を指定します。

画像内の顔を検出するには

1. まだ実行していない場合:

   a. AmazonRekognitionFullAccess と AmazonS3ReadOnlyAccess のアクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. イメージ (1 つ以上の有名人の顔が含まれているもの) を S3 バケットにアップロードします。

   手順については、「Amazon Simple Storage Service ユーザーガイド」の「Amazon S3 へのオブジェクトのアップロード」を参照してください。

3. 以下の例を使用して DetectFaces を呼び出します。

   Java

   この例では、検出した顔の推定年齢層を表示し、すべての検出した顔属性の JSON を一覧表示します。photo の値は、イメージファイル名に変更します。amzn-s3-demo-bucket の値は、イメージの保存先の Amazon S3 バケットに変更します。

```
//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(fac
         }

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

```
      }

   }

}
```

Java V2

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例は [こちら] です。

```
import java.util.List;

//snippet-start:[rekognition.java2.detect_labels.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.DetectFacesRequest;
import software.amazon.awssdk.services.rekognition.model.DetectFacesResponse;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.Attribute;
import software.amazon.awssdk.services.rekognition.model.FaceDetail;
import software.amazon.awssdk.services.rekognition.model.AgeRange;

//snippet-end:[rekognition.java2.detect_labels.import]

public class DetectFaces {

    public static void main(String[] args) {
        final String usage = "\n" +
            "Usage: " +
            "   <bucket> <image>\n\n" +
            "Where:\n" +
            "   bucket - The name of the Amazon S3 bucket that contains the
 image (for example, ,amzn-s3-demo-bucket)." +
            "   image - The name of the image located in the Amazon S3 bucket
 (for example, Lake.png). \n\n";

        if (args.length != 2) {
            System.out.println(usage);
```

```
            System.exit(1);
        }

        String bucket = args[0];
        String image = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        getLabelsfromImage(rekClient, bucket, image);
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.detect_labels_s3.main]
    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();

            DetectFacesRequest facesRequest = DetectFacesRequest.builder()
                    .attributes(Attribute.ALL)
                    .image(myImage)
                    .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 e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
 // snippet-end:[rekognition.java2.detect_labels.main]
}
```

AWS CLI

この例では、 detect-faces AWS CLI オペレーションからの JSON 出力を表示します。fileは、イメージファイル名に置き換えます。amzn-s3-demo-bucket は、イメージファイルが含まれている Amazon S3 バケットの名前に置き換えます。

```
aws rekognition detect-faces --image "{"S3Object":{"Bucket":"amzn-s3-demo-
bucket,"Name":"image-name"}}"\
                            --attributes "ALL" --profile profile-name --region
 region-name
```

Windows デバイスで CLI にアクセスする場合は、パーサーエラーの発生に対処するため、一重引用符の代わりに二重引用符を使用し、内側の二重引用符をバックスラッシュ (\) でエスケープします。例として以下を参照してください。

```
aws rekognition detect-faces --image "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-
bucket\",\"Name\":\"image-name\"}}" --attributes "ALL"
--profile profile-name --region region-name
```

Python

この例では、検出した顔の推定年齢層を表示し、すべての検出した顔属性の JSON を一覧表示します。photo の値は、イメージファイル名に変更します。amzn-s3-demo-bucket の

値は、イメージの保存先の Amazon S3 バケットに変更します。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
import boto3
import json

def detect_faces(photo, bucket, region):

    session = boto3.Session(profile_name='profile-name',
                            region_name=region)
    client = session.client('rekognition', region_name=region)

    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("Face Occluded: " + str(faceDetail['FaceOccluded']))
        print("Emotions: " + str(faceDetail['Emotions'][0]))

    return len(response['FaceDetails'])

def main():
    photo='photo'
    bucket='amzn-s3-demo-bucket'
    region='region'
    face_count=detect_faces(photo, bucket, region)
    print("Faces detected: " + str(face_count))

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

.NET

この例では、検出した顔の推定年齢層を表示し、すべての検出した顔属性の JSON を一覧表示します。photo の値は、イメージファイル名に変更します。amzn-s3-demo-bucket の値は、イメージの保存先の Amazon S3 バケットに変更します。

```
//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 = "amzn-s3-demo-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

この例では、検出した顔の推定年齢層を表示し、さまざまな顔属性を一覧表示します。photo の値は、イメージファイル名に変更します。amzn-s3-demo-bucket の値は、イメージの保存先の Amazon S3 バケットに変更します。

```
  # 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

この例では、検出した顔の推定年齢層を表示し、さまざまな顔属性を一覧表示します。photo の値は、イメージファイル名に変更します。amzn-s3-demo-bucket の値は、イメージの保存先の Amazon S3 バケットに変更します。

Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

TypeScript 定義を使用している場合は、`import AWS from 'aws-sdk'` の代わりに`const AWS = require('aws-sdk')` Node.js でプログラムを実行するようになります。詳細については、AWS JavaScriptのSDK をご覧ください。構成の設定方法によっては、`AWS.config.update({region:`*`region`*`});` でリージョンを指定する必要がある場合もあります。

```
// Load the SDK
var AWS = require('aws-sdk');
const bucket = 'bucket-name' // the bucketname without s3://
const photo  = 'photo-name' // the name of file

var credentials = new AWS.SharedIniFileCredentials({profile: 'profile-name'});
AWS.config.credentials = credentials;
AWS.config.update({region:'region-name'});

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 オペレーションのリクエスト

DetectFaces への入力はイメージです。以下の例では、イメージを Amazon S3 バケットからロードします。Attributes パラメータは、すべての顔属性を返すことを指定します。詳細については、「イメージの操作」を参照してください。

```
{
    "Image": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "input.jpg"
        }
    },
    "Attributes": [
        "ALL"
    ]
}
```

DetectFaces オペレーションのレスポンス

DetectFaces は、検出した顔ごとに以下の情報を返します。

- 境界ボックス – 顔を囲む境界ボックスの座標。
- 信頼度 – 境界ボックス内に顔が含まれている信頼度。

- 顔のランドマーク – 顔のランドマークの配列。ランドマーク (左目、右目、口など) ごとに X 座標と Y 座標がレスポンスで返されます。
- 顔の属性 — 顔が物体で覆われているかどうかなど、顔に関する一連の属性が FaceDetail オブジェクトとして返されます。含まれる属性は AgeRange、Beard、Emotions、EyeDirection、Eyeglasses、EyesOpen、FaceOccluded、Gender、Mout です。レスポンスでは、顔属性ごとに値が返されます。値は、ブール値 (サングラスをしているかどうか)、文字列 (男性か女性か)、角度の値 (視線の前後/上下の動き) など、さまざまな形式で返されます。また、ほとんどの属性では検出した値の信頼度も返されます。FaceOccluded 属性と EyeDirection 属性は、DetectFaces を使用する場合にはサポートされますが、StartFaceDetection と GetFaceDetection を使用してビデオを分析する場合にはサポートされませんのでご注意ください。
- Quality – 顔の明るさとシャープネスを示します。できるだけ最良の顔検出を実現する方法については、「顔比較用の入力イメージに関する推奨事項」を参照してください。
- ポーズ – イメージ内の顔のローテーションを示します。

このリクエストにより、返したい顔の属性の、配列を示せます。顔の属性の DEFAULT サブセット (BoundingBox、Confidence、Pose、Quality、Landmarks) は、常に返されます。特定の顔の属性を (デフォルトのリストに加えて) 返すように、リクエストすることが可能です。それには ["DEFAULT", "FACE_OCCLUDED", "EYE_DIRECTION"] を使用するか、["FACE_OCCLUDED"] のように属性を 1 つだけ使用します。["ALL"] を使用するとすべての顔の属性をリクエストできます。リクエストする属性の数を増やすと応答に時間がかかる場合があります。

以下は、DetectFaces API コールのレスポンスの例です。


```
{
  "FaceDetails": [
    {
      "BoundingBox": {
        "Width": 0.7919622659683228,
        "Height": 0.7510867118835449,
        "Left": 0.08881539851427078,
        "Top": 0.151064932346344
      },
      "AgeRange": {
        "Low": 18,
        "High": 26
      },
      "Smile": {
        "Value": false,
```

```
        "Confidence": 89.77348327636719
      },
      "Eyeglasses": {
        "Value": true,
        "Confidence": 99.99996948242188
      },
      "Sunglasses": {
        "Value": true,
        "Confidence": 93.65237426757812
      },
      "Gender": {
        "Value": "Female",
        "Confidence": 99.85968780517578
      },
      "Beard": {
        "Value": false,
        "Confidence": 77.52591705322266
      },
      "Mustache": {
        "Value": false,
        "Confidence": 94.48904418945312
      },
      "EyesOpen": {
        "Value": true,
        "Confidence": 98.57169342041016
      },
      "MouthOpen": {
        "Value": false,
        "Confidence": 74.33953094482422
      },
      "Emotions": [
        {
          "Type": "SAD",
          "Confidence": 65.56403350830078
        },
        {
          "Type": "CONFUSED",
          "Confidence": 31.277774810791016
        },
        {
          "Type": "DISGUSTED",
          "Confidence": 15.553778648376465
        },
        {
```

```
          "Type": "ANGRY",
          "Confidence": 8.012762069702148
        },
        {
          "Type": "SURPRISED",
          "Confidence": 7.621500015258789
        },
        {
          "Type": "FEAR",
          "Confidence": 7.243380546569824
        },
        {
          "Type": "CALM",
          "Confidence": 5.8196024894714355
        },
        {
          "Type": "HAPPY",
          "Confidence": 2.2830512523651123
        }
      ],
      "Landmarks": [
        {
          "Type": "eyeLeft",
          "X": 0.30225440859794617,
          "Y": 0.41018882393836975
        },
        {
          "Type": "eyeRight",
          "X": 0.6439348459243774,
          "Y": 0.40341562032699585
        },
        {
          "Type": "mouthLeft",
          "X": 0.343580037355423,
          "Y": 0.6951127648353577
        },
        {
          "Type": "mouthRight",
          "X": 0.6306480765342712,
          "Y": 0.6898072361946106
        },
        {
          "Type": "nose",
          "X": 0.47164231538772583,
```

```
          "Y": 0.5763645172119141
        },
        {
          "Type": "leftEyeBrowLeft",
          "X": 0.1732882857322693,
          "Y": 0.34452149271965027
        },
        {
          "Type": "leftEyeBrowRight",
          "X": 0.3655243515968323,
          "Y": 0.33231860399246216
        },
        {
          "Type": "leftEyeBrowUp",
          "X": 0.2671719491481781,
          "Y": 0.31669262051582336
        },
        {
          "Type": "rightEyeBrowLeft",
          "X": 0.5613729953765869,
          "Y": 0.32813435792922974
        },
        {
          "Type": "rightEyeBrowRight",
          "X": 0.7665090560913086,
          "Y": 0.3318614959716797
        },
        {
          "Type": "rightEyeBrowUp",
          "X": 0.6612788438796997,
          "Y": 0.3082450032234192
        },
        {
          "Type": "leftEyeLeft",
          "X": 0.2416982799768448,
          "Y": 0.4085965156555176
        },
        {
          "Type": "leftEyeRight",
          "X": 0.36943578720092773,
          "Y": 0.41230902075767517
        },
        {
          "Type": "leftEyeUp",
```

```
          "X": 0.29974061250686646,
          "Y": 0.3971870541572571
        },
        {
          "Type": "leftEyeDown",
          "X": 0.30360740423202515,
          "Y": 0.42347756028175354
        },
        {
          "Type": "rightEyeLeft",
          "X": 0.5755768418312073,
          "Y": 0.4081145226955414
        },
        {
          "Type": "rightEyeRight",
          "X": 0.7050536870956421,
          "Y": 0.39924031496047974
        },
        {
          "Type": "rightEyeUp",
          "X": 0.642906129360199,
          "Y": 0.39026668667793274
        },
        {
          "Type": "rightEyeDown",
          "X": 0.6423097848892212,
          "Y": 0.41669243574142456
        },
        {
          "Type": "noseLeft",
          "X": 0.4122826159000397,
          "Y": 0.5987403392791748
        },
        {
          "Type": "noseRight",
          "X": 0.5394935011863708,
          "Y": 0.5960900187492371
        },
        {
          "Type": "mouthUp",
          "X": 0.478581964969635,
          "Y": 0.6660456657409668
        },
        {
```

```
          "Type": "mouthDown",
          "X": 0.483366996049881,
          "Y": 0.7497162818908691
        },
        {
          "Type": "leftPupil",
          "X": 0.30225440859794617,
          "Y": 0.41018882393836975
        },
        {
          "Type": "rightPupil",
          "X": 0.6439348459243774,
          "Y": 0.40341562032699585
        },
        {
          "Type": "upperJawlineLeft",
          "X": 0.11031254380941391,
          "Y": 0.3980775475502014
        },
        {
          "Type": "midJawlineLeft",
          "X": 0.19301874935626984,
          "Y": 0.7034031748771667
        },
        {
          "Type": "chinBottom",
          "X": 0.4939905107021332,
          "Y": 0.8877836465835571
        },
        {
          "Type": "midJawlineRight",
          "X": 0.7990140914916992,
          "Y": 0.6899225115776062
        },
        {
          "Type": "upperJawlineRight",
          "X": 0.8548634648323059,
          "Y": 0.38160091638565063
        }
      ],
      "Pose": {
        "Roll": -5.83309268951416,
        "Yaw": -2.4244730472564697,
        "Pitch": 2.6216139793395996
```

```
      },
      "Quality": {
        "Brightness": 96.16363525390625,
        "Sharpness": 95.51618957519531
      },
      "Confidence": 99.99872589111328,
      "FaceOccluded": {
        "Value": true,
        "Confidence": 99.99726104736328
      },
      "EyeDirection": {
        "Yaw": 16.299732,
        "Pitch": -6.407457,
        "Confidence": 99.968704
      }
    }
  ],
  "ResponseMetadata": {
    "RequestId": "8bf02607-70b7-4f20-be55-473fe1bba9a2",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "8bf02607-70b7-4f20-be55-473fe1bba9a2",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "3409",
      "date": "Wed, 26 Apr 2023 20:18:50 GMT"
    },
    "RetryAttempts": 0
  }
}
```


次の点に注意してください:

- Pose データは、検出された顔のローテーションを示します。BoundingBox と Pose データを組み合わせると、アプリケーションで表示する顔の回りに境界ボックスを描画できます。
- Quality は、顔の明るさとシャープネスを示します。複数のイメージ間で顔を比較し、最善の顔を見つける場合に役立ちます。
- 前のレスポンスは、サービスが検出できるすべての顔の landmarks、顔属性、および感情を示しています。これらのすべてをレスポンスで取得するには、attributes パラメータを使用し、値として ALL を指定します。デフォルトでは、DetectFaces API が返す顔属性は BoundingBox、Confidence、Pose、Quality、および landmarks の 5 つのみです。デ

フォルトで返されるランドマークは、eyeLeft、eyeRight、nose、mouthLeft、および mouthRight です。

イメージ間の顔の比較

Rekognition では、CompareFaces オペレーションを使用して 2 つの画像間で顔を比較できます。この機能は、本人確認や写真照合などの用途に役立ちます。

CompareFaces は、ソース画像内の顔をターゲット画像内の各顔と比較します。画像は、以下のいずれかの方法で CompareFaces に渡されます。

- Base64 エンコードされた画像
- Amazon S3 オブジェクト

顔検出と顔比較の違い

顔比較は顔検出とは異なります。顔検出 (DetectFaces を使用) では、画像またはビデオ内の顔の存在と位置のみを識別します。対照的に、顔比較では、ソース画像内の検出された顔をターゲット画像内の顔と比較して、一致するかどうかを判定します。

類似度しきい値

similarityThreshold パラメータを使用して、レスポンスに含める一致の最小信頼度を定義します。デフォルトでは、類似度スコアが 80% 以上の顔のみが応答に返されます。

> **Note**
>
> CompareFaces は確率的な機械学習アルゴリズムを使用します。偽陰性とは、ソースイメージの顔と比較して、ターゲットイメージの顔の類似度信頼スコアが低いという誤った予測です。偽陰性の確率を減らすために、ターゲットイメージを複数のソースイメージと比較することをお勧めします。個人の権利、プライバシー、サービスへのアクセスに影響を与える決定を行うために CompareFaces の操作を使用する予定がある場合は、アクションを起こす前にレビューと検証のために人間に結果を渡すことをお勧めします。

次のコード例は、さまざまな AWS SDKs で CompareFaces オペレーションを使用する方法を示しています。 AWS CLI この例では、2 つの JPEG イメージを Amazon S3 バケットにアップロード

し、オブジェクトキー名を指定します。他の例では、2 つのファイルをローカルファイルシステムからロードし、これらをイメージのバイト配列として入力します。

顔を比較するには

1. まだ実行していない場合:

   a. AmazonRekognitionFullAccess および AmazonS3ReadOnlyAccess (AWS CLI 例のみ) アクセス許可を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. 次のコード例を使用して、CompareFaces オペレーションを呼び出します。

   Java

   この例では、ローカルファイルシステムからロードしたソースイメージとターゲットイメージ間で一致する顔の情報を表示します。

   sourceImage と targetImage の値は、ソースイメージとターゲットイメージのパスとファイル名に置き換えます。

```
//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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例は [こちら] です。

```
import java.util.List;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;
import software.amazon.awssdk.services.rekognition.model.CompareFacesMatch;
import software.amazon.awssdk.services.rekognition.model.CompareFacesRequest;
import software.amazon.awssdk.services.rekognition.model.CompareFacesResponse;
import software.amazon.awssdk.services.rekognition.model.ComparedFace;
import software.amazon.awssdk.core.SdkBytes;
import java.io.FileInputStream;
```

```
import java.io.FileNotFoundException;
import java.io.InputStream;

// snippet-end:[rekognition.java2.detect_faces.import]

/**
 * Before running this Java V2 code example, set up your development
 environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CompareFaces {

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "   <pathSource> <pathTarget>\n\n" +
            "Where:\n" +
            "   pathSource - The path to the source image (for example, C:\\AWS\
\pic1.png). \n " +
            "   pathTarget - The path to the target image (for example, C:\\AWS\
\pic2.png). \n\n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        Float similarityThreshold = 70F;
        String sourceImage = args[0];
        String targetImage = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        compareTwoFaces(rekClient, similarityThreshold, sourceImage,
 targetImage);
```

```
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.compare_faces.main]
    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);
        }
    }
    // snippet-end:[rekognition.java2.compare_faces.main]
}
```

AWS CLI

この例では、 compare-faces AWS CLI オペレーションからの JSON 出力を表示します。

amzn-s3-demo-bucket は、ソースイメージとターゲットイメージが含まれている Amazon S3 バケットの名前に置き換えます。source.jpg と target.jpg は、ソースイメージとターゲットイメージのファイル名に置き換えます。

```
aws rekognition compare-faces --target-image \
"{"S3Object":{"Bucket":"amzn-s3-demo-bucket","Name":"image-name"}}" \
--source-image "{"S3Object":{"Bucket":"amzn-s3-demo-bucket","Name":"image-
name"}}"
--profile profile-name
```

Windows デバイスで CLI にアクセスする場合は、パーサーエラーの発生に対処するため、一重引用符の代わりに二重引用符を使用し、内側の二重引用符をバックスラッシュ (\) でエスケープします。例として以下を参照してください。

```
aws rekognition compare-faces --target-image "{\"S3Object\":{\"Bucket\":\"amzn-
s3-demo-bucket\",\"Name\":\"image-name\"}}" \
--source-image "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-bucket\",\"Name\":
\"image-name\"}}" --profile profile-name
```

Python

この例では、ローカルファイルシステムからロードしたソースイメージとターゲットイメージ間で一致する顔の情報を表示します。

source_file と target_file の値は、ソースイメージとターゲットイメージのパスとファイル名に置き換えます。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
# 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):

    session = boto3.Session(profile_name='profile-name')
    client = session.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-file-name'
    target_file = 'target-file-name'
```

```
    face_matches = compare_faces(source_file, target_file)
    print("Face matches: " + str(face_matches))

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

.NET

この例では、ローカルファイルシステムからロードしたソースイメージとターゲットイメージ間で一致する顔の情報を表示します。

sourceImage と targetImage の値は、ソースイメージとターゲットイメージのパスとファイル名に置き換えます。

```
//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

この例では、ローカルファイルシステムからロードしたソースイメージとターゲットイメージ間で一致する顔の情報を表示します。

photo_source と photo_target の値は、ソースイメージとターゲットイメージのパスとファイル名に置き換えます。

```
 # 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

この例では、ローカルファイルシステムからロードしたソースイメージとターゲットイメージ間で一致する顔の情報を表示します。

photo_source と photo_target の値は、ソースイメージとターゲットイメージのパスとファイル名に置き換えます。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
// Load the SDK
var AWS = require('aws-sdk');
const bucket = 'bucket-name' // the bucket name without s3://
const photo_source  = 'photo-source-name' // path and the name of file
const photo_target = 'photo-target-name'

var credentials = new AWS.SharedIniFileCredentials({profile: 'profile-name'});
AWS.config.credentials = credentials;
AWS.config.update({region:'region-name'});

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 オペレーションのリクエスト

CompareFaces への入力はイメージです。次の例では、ソースイメージとターゲットイメージをローカルファイルシステムからロードします。SimilarityThreshold 入力パラメータで指定した最小限の信頼度以上で一致した顔の比較結果がレスポンスで返されます。詳細については、「イメージの操作」を参照してください。

```
{
    "SourceImage": {
        "Bytes": "/9j/4AAQSk2Q==..."
    },
    "TargetImage": {
        "Bytes": "/9j/4O1Q==..."
    },
    "SimilarityThreshold": 70
}
```

CompareFaces オペレーションのレスポンス

レスポンスは以下のとおりです。

- 顔の一致の配列: 一致する顔のリストであり、一致する各顔の類似度スコアとメタデータを含みます。複数の顔が一致する場合、faceMatches

  配列にはすべての顔の一致が含まれます。

- 顔の一致の詳細: 一致した各顔には、境界ボックス、信頼値、ランドマークの位置、類似度スコアも示されます。
- 一致しない顔のリスト: レスポンスには、ターゲット画像内の顔のうち、ソース画像内の顔と一致しなかったものが含まれます。一致しない各顔の境界ボックスが含まれます。
- ソースの顔情報: 比較に使用されたソース画像内の顔に関する情報 (境界ボックス、信頼値など) が含まれます。

この例では、ターゲット画像内で顔の一致が 1 つ見つかったことを示しています。その顔の一致について、境界ボックスと信頼値 (境界ボックス内に顔が含まれていることを示す Amazon Rekognition の信頼度) が返されています。類似度スコア 99.99% は、顔が非常に似ていることを示しています。この例では、Amazon Rekognition がターゲット画像内で検出した 1 つの顔が、ソース画像内で分析した顔と一致しませんでした。


```
{
    "FaceMatches": [{
        "Face": {
            "BoundingBox": {
                "Width": 0.5521978139877319,
                "Top": 0.1203877404332161,
                "Left": 0.23626373708248138,
                "Height": 0.3126954436302185
            },
            "Confidence": 99.98751068115234,
            "Pose": {
                "Yaw": -82.36799621582031,
                "Roll": -62.13221740722656,
                "Pitch": 0.8652129173278809
            },
            "Quality": {
                "Sharpness": 99.99880981445312,
                "Brightness": 54.49755096435547
            },
            "Landmarks": [{
                    "Y": 0.2996366024017334,
                    "X": 0.41685718297958374,
                    "Type": "eyeLeft"
                },
                {
                    "Y": 0.2658946216106415,
                    "X": 0.4414493441581726,
```

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

```
            },
            {
                "Y": 0.7632885575294495,
                "X": 0.28091415762901306,
                "Type": "eyeRight"
            },
            {
                "Y": 0.7417283654212952,
                "X": 0.3631140887737274,
                "Type": "nose"
            },
            {
                "Y": 0.8081989884376526,
                "X": 0.48565614223480225,
                "Type": "mouthLeft"
            },
            {
                "Y": 0.7548204660415649,
                "X": 0.46090251207351685,
                "Type": "mouthRight"
            }
        ]
    }],
    "SourceImageFace": {
        "BoundingBox": {
            "Width": 0.5521978139877319,
            "Top": 0.1203877404332161,
            "Left": 0.23626373708248138,
            "Height": 0.3126954436302185
        },
        "Confidence": 99.98751068115234
    }
}
```

保存済みビデオ内の顔の検出

Amazon Rekognition Video では、Amazon S3 バケットに保存されているビデオ内の顔を検出し、以下のような情報を提供できます。

- ビデオ内で顔を検出した時間。
- 検出時点におけるビデオフレーム内の顔の位置。
- 顔のランドマーク (左目の位置など)。

- the section called “顔属性のガイドライン” ページで説明されている追加の属性。

保存済みビデオ内の Amazon Rekognition Video のテキスト検出は、非同期オペレーションです。ビデオで顔の検出を開始するには、StartFaceDetection を呼び出します。Amazon Rekognition Video は、動画分析の完了ステータスを Amazon Simple Notification Service (Amazon SNS) トピックに公開します。ビデオの分析が完了したら、GetFaceDetection を呼び出すとビデオ分析の結果を取得できます。ビデオ分析の開始と結果の取得の詳細については、「Amazon Rekognition Video オペレーションを呼び出す」を参照してください。

この手順では、Amazon Simple Queue Service (Amazon SQS) のキューを使用してビデオ分析リクエストの完了ステータスを取得する Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK) のコードを拡張します。

Amazon S3 バケット(SDK) に保存されたビデオ内のテキストを検出するには

1. 「Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)」を実行します。
2. ステップ 1 で作成したクラス `VideoDetect` に次のコードを追加します。

   AWS CLI

   - 以下のコード例では、`amzn-s3-demo-bucket` と `video-name` を、ステップ 2 で指定した Amazon S3 バケットの名前およびファイル名に変更します。
   - `region-name` を、使用している AWS リージョンに変更します。`profile_name` の値を自分のデベロッパープロファイル名に置き換えます。
   - `TopicARN` を、Amazon Rekognition Video の設定 のステップ 3 で作成した Amazon SNS トピックの ARN に変更します。
   - `RoleARN` を、Amazon Rekognition Video の設定 のステップ 7 で作成した IAM サービスロールの ARN に変更します。

```
aws rekognition start-face-detection --video "{"S3Object":{"Bucket":"amzn-s3-
demo-bucket","Name":"Video-Name"}}" --notification-channel
"{"SNSTopicArn":"Topic-ARN","RoleArn":"Role-ARN"}" --region region-name --
profile profile-name
```

Windows デバイスで CLI にアクセスする場合は、パーサーエラーの発生に対処するため、一重引用符の代わりに二重引用符を使用し、内側の二重引用符をバックスラッシュ (\) でエスケープします。例として以下を参照してください。

```
aws rekognition start-face-detection --video "{\"S3Object\":{\"Bucket\":\"amzn-
s3-demo-bucket\",\"Name\":\"Video-Name\"}}" --notification-channel
"{\"SNSTopicArn\":\"Topic-ARN\",\"RoleArn\":\"Role-ARN\"}" --region region-name
 --profile profile-name
```

StartFaceDetection オペレーションを実行してジョブ ID 番号を取得したら、次の GetFaceDetection オペレーションを実行してジョブ ID 番号を指定します。

```
aws rekognition get-face-detection --job-id job-id-number  --profile profile-
name
```

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);
```

```
}
```

関数 main で、以下の行を置き換えます。

```
        StartLabelDetection(amzn-s3-demo-bucket, video);

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

を:

```
        StartFaceDetection(amzn-s3-demo-bucket, video);

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

Java V2

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例は [こちら] です。

```
//snippet-start:[rekognition.java2.recognize_video_faces.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_faces.import]


/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class VideoDetectFaces {
```

```
private static String startJobId ="";
public static void main(String[] args) {

    final String usage = "\n" +
        "Usage: " +
        "   <bucket> <video> <topicArn> <roleArn>\n\n" +
        "Where:\n" +
        "   bucket - The name of the bucket in which the video is located (for
example, (for example, amzn-s3-demo-bucket). \n\n"+
        "   video - The name of video (for example, people.mp4). \n\n" +
        "   topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic. \n\n" +
        "   roleArn - The ARN of the AWS Identity and Access Management (IAM)
role to use. \n\n" ;

    if (args.length != 4) {
        System.out.println(usage);
        System.exit(1);
    }

    String bucket = args[0];
    String video = args[1];
    String topicArn = args[2];
    String roleArn = args[3];

    Region region = Region.US_EAST_1;
    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

    NotificationChannel channel = NotificationChannel.builder()
        .snsTopicArn(topicArn)
        .roleArn(roleArn)
        .build();

    StartFaceDetection(rekClient, channel, bucket, video);
    GetFaceResults(rekClient);
    System.out.println("This example is done!");
    rekClient.close();
}

// snippet-start:[rekognition.java2.recognize_video_faces.main]
```

```
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);
     }
 }
 // snippet-end:[rekognition.java2.recognize_video_faces.main]
}
```

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
```

関数 `main` で、以下の行を置き換えます。

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

を:

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

> **Note**
>
> 「Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)」以外でビデオ例を実行済みである場合、置き換える関数名は異なります。

3. コードを実行します。ビデオ内で検出された顔に関する情報が表示されます。

GetFaceDetection オペレーションレスポンス

`GetFaceDetection` は、ビデオ内で検出された顔に関する情報が含まれた配列 (`Faces`) を返します。配列要素 FaceDetection は、ビデオで顔が検出されるたびに生成されます。配列要素は、ビデオの開始時点からの経過時間 (ミリ秒単位) で並べ替えられて返されます。

次の例は、GetFaceDetection からの JSON レスポンスの一部です。レスポンスで、以下の点に注意してください。

- 境界ボックス – 顔を囲む境界ボックスの座標。
- 信頼度 – 境界ボックス内に顔が含まれている信頼度。
- 顔のランドマーク – 顔のランドマークの配列。ランドマーク (左目、右目、口など) ごとに、x 座標と y 座標がレスポンスで返されます。
- 顔の属性 — 顔の属性は AgeRange、Beard、Emotions、Eyeglasses、EyesOpen、Gender、MouthOpen、Mustache、Smile、Su です。値は、ブール値 (サングラスをしているかどうか) や文字列 (男性か女性か) など、さまざまな型で返される場合があります。また、ほとんどの属性では検出した値の信頼度も返されます。FaceOccluded 属性と EyeDirection 属性は、DetectFaces を使用する場合にはサポートされますが、StartFaceDetection と GetFaceDetection を使用してビデオを分析する場合にはサポートされませんのでご注意ください。
- タイムスタンプ — ビデオ内で顔が検出された時間です。
- ページング情報 – 例は 1 ページの顔検出情報を示しています。人物要素を返す数は、GetFaceDetection の MaxResults 入力パラメータで指定できます。MaxResults の数を超える結果が存在する場合、GetFaceDetection から返されるトークン (NextToken) を使用して次の結果ページを取得できます。詳細については、「Amazon Rekognition Video の分析結果を取得する」を参照してください。
- ビデオ情報 – このレスポンスには、VideoMetadata から返された各情報ページのビデオ形式 (GetFaceDetection) に関する情報が含まれます。
- Quality – 顔の明るさとシャープネスを示します。
- ポーズ – イメージ内の顔の回転を示します。

```
{
    "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
    }
}
```

コレクション内での顔の検索

Amazon Rekognition を使用すると、入力された顔を使用して、保存されている顔のコレクションから一致する顔を検索できます。まず、検出した顔の情報を、サーバー側のコンテナである「コレクション」に保存します。コレクションには個々の顔とユーザー (同一人物の複数の顔) の両方が保存されます。個々の顔は、顔を数学的に表現した顔ベクトルとして保存されます (実在の顔の画像ではなく)。同一人物の異なる画像を使用して、同じコレクションに複数の顔ベクトルを作成し、保存できます。その後、同一人物の複数の顔ベクトルを集計して、ユーザーベクトルを作成できます。ユーザーベクトルは、さまざまなレベルの明るさ、シャープネス、ポーズ、見た目などを含む、より安定した表現であり、より高い精度で顔を検索できます。

コレクションを作成したら、入力した顔を使用して、コレクション内の一致するユーザーベクトルまたは顔ベクトルを検索できます。ユーザーベクトルを検索すると、個々の顔ベクトルを検索する場合よりも精度を大幅に高めることができます。イメージ、保存済みビデオ、ストリーミングビデオ内で検出された顔を使用して、保存した顔ベクトルに対して検索を行うことができます。画像内で検出された顔を使用して、保存されているユーザーベクトルに対して検索を行えます。

顔の情報を保存するときは、次の操作を行う必要があります。

1. コレクションの作成 - 顔情報を保存するには、まずアカウントのいずれかの AWS リージョンで顔コレクションを作成 (CreateCollection) する必要があります。この顔コレクションは、IndexFaces オペレーションを呼び出すときに指定します。
2. 顔にインデックスを付ける - この IndexFaces オペレーションでは、画像内の顔を検出し、顔ベクトルを抽出してコレクションに保存します。このオペレーションを使用して、イメージ内の顔を検出し、検出した顔の特徴に関する情報をコレクション内に保持できます。サービスが顔ベクトル情報をサーバーに保持するため、こちらはストレージベースの API オペレーションの一例です。

ユーザーを作成し、このユーザーに複数の顔ベクトルを関連付けるには、次の操作を行う必要があります。

1. ユーザーを作成する - 最初に CreateUser を使用してユーザーを作成します。同一人物の複数の顔ベクトルをユーザーベクトルに集約することで、顔照合の精度を高めることができます。1 つのユーザーベクトルには、最大で 100 個の顔ベクトルを関連付けることができます。
2. 顔を関連付ける - ユーザーを作成したら、AssociateFaces オペレーションを使って既存の顔ベクトルをそのユーザーに追加できます。顔ベクトルをユーザーベクトルに関連付けるには、顔ベクトルがユーザーベクトルと同じコレクション内に存在している必要があります。

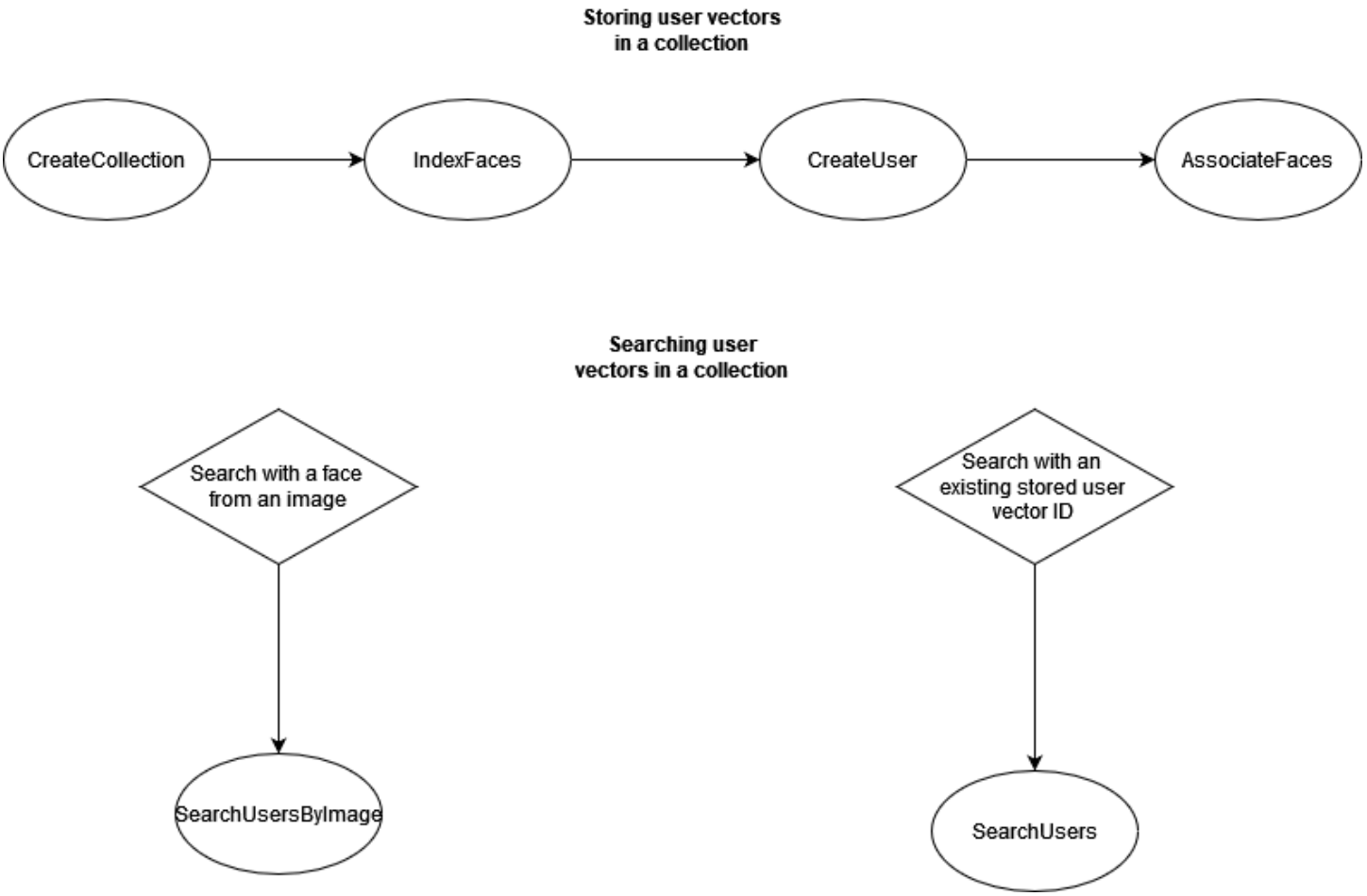
コレクションを作成して顔ベクトルとユーザーベクトルを保存した後に、以下のオペレーションを使って一致する顔を検索できます。

- SearchFacesByImage - 保存されている個々の顔の中から、画像の顔を検索します。
- SearchFaces - 指定した顔 ID を使って、保存されている個々の顔の中から検索します。
- SearchUsers - 指定した顔 ID またはユーザー ID を使って、保存されているユーザーの中から検索します。
- SearchUsersByImage - 保存されているユーザーの中から画像の顔を検索します。
- StartFaceSearch - 保存したビデオ内で顔を検索します。
- CreateStreamProcessor - ストリーミングビデオ内で顔を検索します。

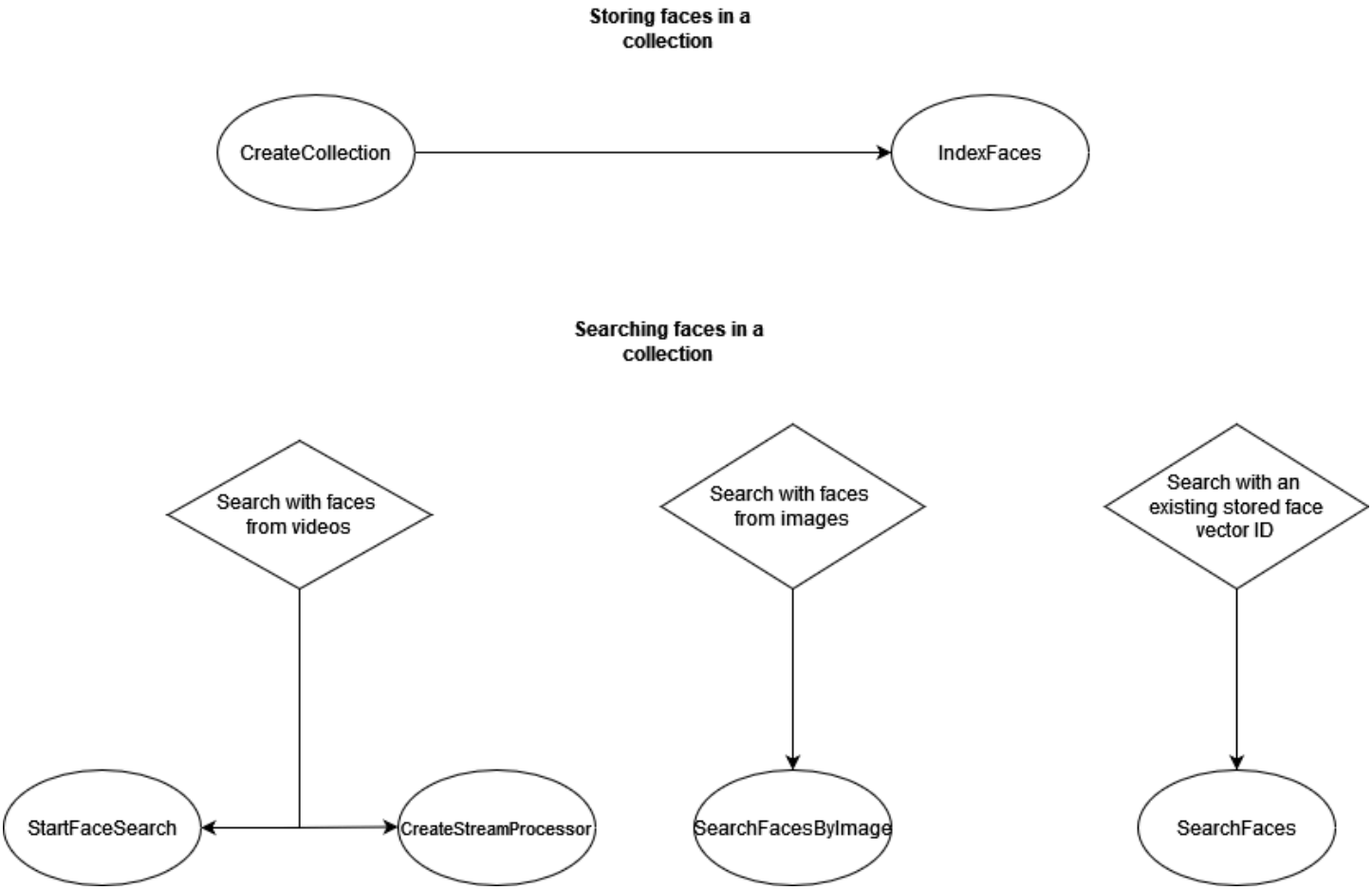
> **Note**
>
> コレクションには、顔の数学的表現である顔ベクトルが保存されます。コレクションには、顔の画像は保存されません。

以下の図は、コレクションの使用目的に応じて、必要なオペレーションを呼び出す順序を示しています。

ユーザーベクトルとの照合精度を最大にするには:



個々の顔ベクトルとの照合精度を高くするには:



コレクションは、さまざまなシナリオで使用できます。例えば、IndexFaces および AssociateFaces オペレーションを使用することにより、スキャンした従業員バッジのイメージや政府発行の ID から検出した顔を保存する顔コレクションを作成できます。社員が建物内に入ると、その顔のイメージがキャプチャされ、SearchUsersByImage オペレーションに送信されます。十分に高い類似度スコア (99% など) で顔が一致すると、この社員を認証できます。

顔コレクション、顔、ユーザーの管理

顔コレクションは、顔とユーザーに関連する情報を保存するために Amazon Rekognition によって使用されるリソースです。さまざまな Rekognition API オペレーションを使用して、コレクション自体と、コレクション内に保存されている顔や顔ベクトルを管理します。

コレクションの管理

顔コレクションは Amazon Rekognition リソースの第一次リソースであり、顔コレクションを作成するたびに一意の Amazon リソースネーム (ARN) が割り当てられます。各顔コレクションは、アカウントの特定の AWS リージョンに作成します。コレクションを作成すると、最新バージョンの顔検出

モデルに関連付けられます。詳細については、「モデルのバージョニングについて」を参照してください。

コレクションに対しては、以下の管理オペレーションを実行できます。

- CreateCollection でコレクションを作成します。詳細については、「コレクションの作成」を参照してください。
- ListCollections で使用可能なコレクションを一覧表示します。詳細については、「コレクションの一覧表示」を参照してください。
- DescribeCollection でコレクションを記述します。詳細については、「コレクションの定義」を参照してください。
- DeleteCollection でコレクションを削除します。詳細については、「コレクションの削除」を参照してください。

コレクション内の顔の管理

顔コレクションを作成したら、それに顔を保存できます。Amazon Rekognition では、以下のオペレーションを使用してコレクション内の顔を管理できます。

- IndexFaces オペレーションでは、入力イメージ (JPEG または PNG) 内の顔を検出し、これを指定された顔コレクションに追加します。イメージで検出された顔ごとに一意の顔 ID が返されます。顔を保存したら、顔コレクション内で一致する顔を検索できます。詳細については、「コレクションへの顔の追加」を参照してください。
- ListFaces オペレーションは、コレクション内の顔を一覧表示します。詳細については、「コレクションへの顔の追加」を参照してください。
- DeleteFaces オペレーションは、コレクションから顔を削除します。詳細については、「コレクションからの顔の削除」を参照してください。

コレクション内のユーザーの管理

同一人物の複数の顔ベクトルを保存した場合、それらすべての顔ベクトルを 1 つのユーザーベクトルに関連付けることで、精度を高めることができます。次のオペレーションを使用するとユーザーを管理できます。

- CreateUser - 指定された一意のユーザー ID を使ってコレクションに新しいユーザーを作成します。

- AssociateUsers - ユーザー ID に 1～100 個の一意の顔 ID を追加します。ユーザーに 1 つ以上の顔 ID をユーザーに関連付けると、コレクションでそのユーザーに一致するものを検索できるようになります。
- ListUsers - コレクション内のユーザーを一覧表示します。
- DeleteUsers - 指定されたユーザー ID のコレクションから特定のユーザーを削除します。
- DisassociateFaces - 特定のユーザーの、1 つまたは複数の顔 ID を削除します。

類似度しきい値を使用して顔を関連付け、照合する

類似度しきい値は、顔の関連付けと照合の両方で使用されます。以下は、両方のユースケースで類似度しきい値を使用するためのガイダンスです。

顔の関連付けに類似度しきい値を使用する

AssociateFaces オペレーションを使用して顔を関連付ける場合、関連付けられる顔がすべて同一人物のものであることを確認することが重要です。そのため、UserMatchThreshold パラメータは、新しい顔を、FaceID を既に 1 つ以上含んでいる UserID に関連付けるために必要な最小限のユーザーマッチ信頼度を指定します。これにより、FaceIds を適切な UserID に関連付けることができます。値の範囲は 0～100 で、デフォルト値は 75 です。

類似度しきい値を使用した顔のマッチング

類似度しきい値を入力パラメータとして指定すると、ユーザーはすべての検索オペレーション (CompareFaces、SearchFaces、SearchFacesByImage、SearchUsers、SearchUsersByImage) の結果をコントロールできます。

FaceMatchThreshold は、SearchFaces と SearchFacesByImage の間における類似度しきい値の入力属性であり、照合した顔の類似度に基づいて返される結果の数をコントロールします。SearchUsers と SearchUsersByImage の類似度しきい値の属性は UserMatchThreshold であり、照合したユーザーベクトルの類似度に基づいて返される結果の数をコントロールします。しきい値属性は、CompareFaces の SimilarityThreshold です。

Similarity レスポンス属性値がしきい値より小さいレスポンスは返されません。このしきい値は、ユースケースに合わせて調整することが重要です。この値により、一致結果に含まれる誤認識の数が変わるためです。これにより、検索結果のリコールが制御されます。しきい値が低いほど、リコールが高くなります。

すべての機械学習システムは確率的です。適切な類似度しきい値の設定には、ユースケースに応じて、お客様の判断が必要です。例えば、見た目が類似した家族を識別する写真アプリケーションを構築する場合は、より低いしきい値 (80% など) を選択できます。一方、多くの法律執行のユースケースでは、偶発的な誤認識を減らすため、99% 以上の高いしきい値を使用することをお勧めします。

FaceMatchThreshold と UserMatchThreshold に加え、偶然的な誤認識を減らす手段として Similarity のレスポンス属性を使用できます。たとえば、低いしきい値 (80% など) を使用して、より多くの結果が返されるようにできます。その後、レスポンス属性 Similarity (類似度の割合) を使用してそれらの結果を絞り込み、アプリケーションで正しいレスポンスが得られるようにフィルタリングできます。ここでも、高い類似度 (99% 以上など) を指定すると、誤認識のリスクが軽減されます。

一般的なシナリオで顔のインデックスを作成するためのガイダンス

一般的なシナリオで IndexFaces を使用するためのガイダンスを以下に示します。

重要または公共安全のアプリケーション

- 各画像に 1 つの顔のみを含む画像で IndexFaces を呼び出し、返された顔 ID を画像の被写体の識別子と関連付けます。
- インデックス作成の前に DetectFaces を使用して、画像内の顔が 1 つのみであることを確認できます。複数の顔が検出された場合は、確認後、顔が 1 つだけの状態で画像を再送信してください。これにより、誤って複数の顔にインデックスを作成し、それらを同じ人物に関連付けることを防ぎます。

写真共有とソーシャルメディアアプリケーション

- 家族のアルバムのようなユースケースでは、複数の顔を含む画像に制限なく、IndexFaces を呼び出す必要があります。このような場合は、すべての写真の各人物を識別し、その情報を使用して写真に写っている人物ごとに写真をまとめる必要があります。

一般的な使用

- 同一人物の複数の異なる画像、特に異なる顔属性 (顔のポーズ、ひげなど) の画像にインデックスを付け、ユーザーを作成し、異なる顔をそのユーザーに関連付けて照合の質を高めます。

- その後の顔照合能力を向上させるために、失敗した照合が正しい顔識別子で索引付けされることができるようにレビュープロセスを含めます。
- 画像品質の詳細については、「顔比較用の入力イメージに関する推奨事項」を参照してください。

コレクション内の顔とユーザーの検索

顔コレクションを作成して顔ベクトルまたはユーザーベクトルを保存したら、顔コレクション内で一致する顔を検索できます。Amazon Rekognition では、以下と一致するコレクション内の顔を検索できます。

- 指定された顔 ID (SearchFaces)。詳細については、「顔 ID を使用した顔の検索」を参照してください。
- 指定した画像内の最大の顔 (SearchFacesByImage)。詳細については、「イメージを使用した顔の検索」を参照してください。
- 保存したビデオ内の顔。詳細については、「 保存したビデオでの顔の検索」を参照してください。
- ストリーミングビデオ内の顔。詳細については、「ストリーミングビデオイベントの操作」を参照してください。

CompareFaces オペレーションを使用すると、ソースイメージの顔とターゲットイメージの顔を比較できます。この比較の範囲は、ターゲットイメージで検出された顔に制限されます。詳細については、「イメージ内の顔を比較する」を参照してください。

以下のリストにあるさまざまな検索オペレーションでは、顔 (FaceId または入力イメージで識別される) を、特定の顔コレクションに保存されているすべての顔と比較します。

- SearchFaces
- SearchFacesByImage
- SearchUsers
- SearchUsersByImage

公共安全に関係するユースケース

センサー、入力イメージ、ビデオのベストプラクティス および 一般的なシナリオで顔のインデックスを作成するためのガイダンス に記載されている推奨事項に加えて、公共安全を伴うユースケース

で顔の検出システムと比較システムをデプロイするときは、次のベストプラクティスを使用してください。まず、エラーおよび誤検知を減らすために、99% 以上のしきい値以上を使用してください。次に、人間のレビュー担当者を採用し、顔検出または比較システムから受け取った結果を検証する必要があります。つまり、人間によるレビューを追加せずにシステムからの出力に基づいた判断をするべきではありません。顔の検出システムと比較システムは、視野を狭め、人間が迅速に選択肢を検討し検討することを可能にするために役立つツールとして機能するべきです。3 番目に、これらのユースケースでは、顔検出および比較システムの使用について透明性を保つ必要があります。これには、可能な限り、これらのシステムの使用についてエンドユーザーおよび被写体に通知すること、当該の使用についての同意を得ること、エンドユーザーおよび被写体がシステムを改善するためのフィードバックを提供できる仕組みを用意することが含まれます。

犯罪捜査に関連して Amazon Rekognition の顔比較機能を使用する法執行機関の場合、AWS サービス条件 に記載されている要件に従う必要があります。これには以下が含まれます。

- 適切なトレーニングを受けた人たちに、市民としての自由または同等の人権に影響を与える可能性のある行動を起こす決定すべてについて確認してもらいます。
- 担当者に顔認識システムの責任ある使用についてトレーニングします。
- 顔認識システムの使用について公に開示します。
- 独立した審査や緊急事態でない限り、Amazon Rekognition を使用して、人の継続的な監視を行わないでください。

いずれの場合も、顔比較の一致は他の重要な証拠とともに考慮される必要があり、実行における唯一の判断材料として使用されるべきではありません。ただし、顔認識が非司法機関によって使用されるシナリオでは (たとえば、電話のロック解除や安全な民間オフィスビルにアクセスするための社員の ID の認証など)、市民としての自由または同等の人権には影響を及ぼすことがないため、上述の決定に手動による監査は必要ではありません。

公共の安全を含むユースケースに顔検出システムまたは顔比較システムを使用することを計画している場合は、前述のベストプラクティスを採用する必要があります。さらに、顔比較の使用に関して公開されている資料を参照する必要があります。これには、米司法省の Bureau of Justice Assistance により提供される『Face Recognition Policy Development Template For Use In Criminal Intelligence and Investigative Activities (犯罪情報および捜査活動における使用のための顔認識ポリシー開発テンプレート)』が含まれます。テンプレートでは、いくつかの顔比較とバイオメトリック関連のリソースが提供されています。また、適用法を遵守し、プライバシーのリスクを軽減して、企業の説明責任と監督を確立する、顔比較ポリシーを作成するためのフレームワークを法執行機関および公安機関に提供するようにつくられています。その他のリソースには、『Best Privacy Practices for Commercial Use of Facial Recognition (顔認識の商業利用のためのプライバシーのベストプラクティス)』(アメリ

力合衆国国家電気通信情報管理庁)、『Best Practices for Common Uses of Facial Recognition (顔認識の一般的な使用のためのベストプラクティス)』(連邦取引委員会職員) などがあります。将来、他のリソースが開発・公開される可能性があります。この重要なトピックについて継続的に自身で学習する必要があります。

繰り返しになりますが、お客様は、AWS のサービスの使用に適用されるすべての法律を遵守する必要があります。また、他人の権利を侵害したり、他人に害を及ぼすような方法で、あらゆる AWS のサービスを使用することはできません。これは、違法に個人を差別したり、個人の適正手続き、プライバシー、または市民の自由を侵害したりするような方法で、公共安全のユースケースのために AWS のサービスを使用してはならないことを意味します。ユースケースに関する法的要件や質問を確認するには、必要に応じて適切な法的助言を受ける必要があります。

Amazon Rekognition を使用して公共の安全を支援する

Amazon Rekognition は、行方不明児童の捜索、人身売買への対抗、犯罪の防止など、公共の安全維持と法律執行のシナリオに役立ちます。公共の安全維持と司法のシナリオでは、以下の点を考慮してください。

- 一致する可能性の検出への最初のステップとして Amazon Rekognition を使用します。Amazon Rekognition の顔オペレーションからのレスポンスにより、さらなる検討の対象と一致する候補をすばやく取得できます。
- 人間による分析が必要なシナリオでは、Amazon Rekognition のレスポンスを使用した自律判断に意思決定を委ねない。犯罪捜査に関連して人物の特定を支援するために Amazon Rekognition を使用する法執行機関であり、個人の市民としての自由または同等の人権に影響を与える可能性のある身元確認に基づいて行動を起こす場合、行動を起こすかどうかの決定は、身分証明の独立した検査に基づいて、適切にトレーニングされた人が行う必要があります。
- 顔の非常に高い類似度によるマッチングが必要なシナリオでは、99% の類似度しきい値を使用する。その一例に建物へのアクセスの認証があります。
- 司法に関連するユースケースなど人権が考慮される場合では、99% 以上の信頼性しきい値を使用し、人権の侵害は生じないよう顔比較の予測に人間による確認を採用することが必要です。
- 可能性のある一連の一致数が多いほどメリットが高くなるシナリオでは、99% 未満の類似度しきい値を使用します。その一例は、行方不明者の捜索です。必要に応じて、Similarity レスポンス属性を使用して、認識する人物に対して一致候補となる場合の類似度を決定してください。
- Amazon Rekognition から返される顔の誤認識の一致への対策を立てる。例えば、IndexFaces オペレーションでインデックスを作成するときに、同じ人物の複数のイメージを使用することでマッチングを改善します。詳細については、「一般的なシナリオで顔のインデックスを作成するためのガイダンス」を参照してください。

その他のユースケース (ソーシャルメディアなど) では、Amazon Rekognition の結果が人によるレビューが必要かどうかを評価し、最良の判断を下すことをお勧めします。また、アプリケーションの要件に応じて、類似度しきい値を小さくしてもかまいません。

コレクションの作成

コレクションは、CreateCollection オペレーションを使って作成します。

詳細については、「コレクションの管理」を参照してください。

コレクションを作成するには (SDK)

1. まだ実行していない場合:
   a. AmazonRekognitionFullAccess アクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。
   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 以下の例を使用して、CreateCollection オペレーションを呼び出します。

   Java

   次の例では、コレクションを作成して、その Amazon リソースネーム (ARN) を表示します。

   collectionId の値は、作成するコレクションの名前に変更します。

```
//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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

Rekognition セッションを作成する行の `profile_name` の値を、自分のデベロッパープロファイル名に置き換えます。

```
//snippet-start:[rekognition.java2.create_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.CreateCollectionResponse;
```

```
import
 software.amazon.awssdk.services.rekognition.model.CreateCollectionRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
//snippet-end:[rekognition.java2.create_collection.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class CreateCollection {

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <collectionName> \n\n" +
         "Where:\n" +
         "   collectionName - The name of the collection. \n\n";

     if (args.length != 1) {
         System.out.println(usage);
         System.exit(1);
     }

     String collectionId = args[0];
     Region region = Region.US_EAST_1;
     RekognitionClient rekClient = RekognitionClient.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
         .build();

     System.out.println("Creating collection: " +collectionId);
     createMyCollection(rekClient, collectionId );
     rekClient.close();
 }

 // snippet-start:[rekognition.java2.create_collection.main]
 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);
    }
}
// snippet-end:[rekognition.java2.create_collection.main]
```

AWS CLI

この AWS CLI コマンドは、 CLI オペレーションの JSON `create-collection` 出力を表示します。

`collection-id` の値は、作成するコレクションの名前に置き換えます。

`profile_name` の値を自分のデベロッパープロファイル名に置き換えます。

```
aws rekognition create-collection --profile profile-name --collection-id
 "collection-name"
```

Python

次の例では、コレクションを作成して、その Amazon リソースネーム (ARN) を表示します。

`collection_id` の値は、作成するコレクションの名前に変更します。Rekognition セッションを作成する行の `profile_name` の値を、自分のデベロッパープロファイル名に置き換えます。

```
# 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):
    session = boto3.Session(profile_name='profile-name')
    client = session.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-id"
    create_collection(collection_id)

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

.NET

次の例では、コレクションを作成して、その Amazon リソースネーム (ARN) を表示します。

collectionId の値は、作成するコレクションの名前に変更します。

```
//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

次の例では、region の値をアカウントに関連付けられたリージョンの名前に置き換え、collectionName の値をコレクションの希望する名前に置き換えます。

Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

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

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Set the profile name
const profileName = "profile-name"
```

```
// Name the collection
const collectionName = "collection-name"
const rekogClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});

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(collectionName)
```

CreateCollection オペレーションのリクエスト

`CreationCollection` への入力は、作成するコレクションの名前です。

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

CreateCollection オペレーションのレスポンス

Amazon Rekognition はコレクションを作成し、新しく作成したコレクションの Amazon リソースネーム (ARN) に返します。

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

タグコレクション

タグを使用して、Amazon Rekognition コレクションを識別、整理、検索、フィルタリングできます。各タグは、ユーザー定義のキーと値で構成されるラベルです。

Identity and Access Management (IAM) を使用して、タグ でコレクションへのアクセスを制御することもできます。詳細については、AWS 「リソースタグを使用したリソースへのアクセスの制御」を参照してください。

トピック

- 新しいコレクションにタグを追加する
- 既存のコレクションにタグを追加する
- コレクション内のタグを一覧表示する
- コレクションからタグを削除する

新しいコレクションにタグを追加する

`CreateCollection` オペレーション を使用して、コレクションの作成時にタグを追加できます。`Tags` 配列入力パラメーターで、 1 つ以上のタグを指定します。

AWS CLI

`profile_name` の値を自分のデベロッパープロファイル名に置き換えます。

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

Windows デバイスの場合:

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

Python

Rekognition セッションを作成する行の `profile_name` の値を、自分のデベロッパープロファイル名に置き換えます。

```
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()
```

既存のコレクションにタグを追加する

既存の証跡に 1 つ以上のタグを追加するには、TagResource オペレーションを使用します。コレクションの Amazon リソースネーム (ARN) (ResourceArn) と、追加したいタグ (Tags) を指定します。次の例は、2 つのタグを追加する方法を示しています。

AWS CLI

profile_name の値を自分のデベロッパープロファイル名に置き換えます。

```
aws rekognition tag-resource --resource-arn collection-arn --tags
 "{"key1":"value1","key2":"value2"}" --profile profile-name
```

Windows デバイスの場合:

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

Python

Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
# 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_tag(collection_id):
    session = boto3.Session(profile_name='default')
    client = session.client('rekognition')
    response = client.tag_resource(ResourceArn=collection_id,
                                   Tags={
                                       "KeyName": "ValueName"
                                   })
    print(response)
    if "'HTTPStatusCode': 200" in str(response):
        print("Success!!")

def main():
    collection_arn = "collection-arn"
    create_tag(collection_arn)

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

> **Note**
>
> コレクションの Amazon リソース名がわからない場合は、DescribeCollection オペレーションを使用することができます。

コレクション内のタグを一覧表示する

コレクションに添付されるタグを一覧表示するには、ListTagsForResource オペレーションを実行し、コレクション (ResourceArn) の ARN を指定します。。応答は、指定されたコレクションに添付されるタグのキーと値のマップです。

AWS CLI

profile_name の値を自分のデベロッパープロファイル名に置き換えます。

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

Python

Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
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()
```

出力には、コレクションに添付されるタグのリストが表示されます。

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

コレクションからタグを削除する

コレクションから 1 つ以上のタグを削除するには、UntagResource オペレーションを使用する。モデル (ResourceArn) の ARN と 削除したいタグキー (Tag-Keys) を指定します。

AWS CLI

profile_name の値を自分のデベロッパープロファイル名に置き換えます。

```
aws rekognition untag-resource --resource-arn resource-arn --profile profile-name --
tag-keys "key1" "key2"
```

または、次の形式でタグキーを指定できます。

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

Python

Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
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()
```

コレクションの一覧表示

ListCollections オペレーションを使用すると、使用しているリージョンのコレクションを一覧表示できます。

詳細については、「コレクションの管理」を参照してください。

コレクションを一覧表示するには (SDK)

1. まだ実行していない場合:

   a. AmazonRekognitionFullAccess アクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. 以下の例を使用して、ListCollections オペレーションを呼び出します。

   Java

   次の例では、現在のリージョンにあるコレクションを一覧表示します。

```
//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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
//snippet-start:[rekognition.java2.list_collections.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.ListCollectionsRequest;
import
 software.amazon.awssdk.services.rekognition.model.ListCollectionsResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;
//snippet-end:[rekognition.java2.list_collections.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
```

```
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class ListCollections {

 public static void main(String[] args) {

     Region region = Region.US_EAST_1;
     RekognitionClient rekClient = RekognitionClient.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
         .build();

     System.out.println("Listing collections");
     listAllCollections(rekClient);
     rekClient.close();
 }

 // snippet-start:[rekognition.java2.list_collections.main]
 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);
     }
 }
 // snippet-end:[rekognition.java2.list_collections.main]
}
```

AWS CLI

この AWS CLI コマンドは、 CLI オペレーションの JSON `list-collections` 出力を表示します。`profile_name` の値を自分のデベロッパープロファイル名に置き換えます。

```
aws rekognition list-collections --profile profile-name
```

Python

次の例では、現在のリージョンにあるコレクションを一覧表示します。

Rekognition セッションを作成する行の `profile_name` の値を、自分のデベロッパープロファイル名に置き換えます。

```
#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

次の例では、現在のリージョンにあるコレクションを一覧表示します。

```
//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

Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

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

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Set the profile name
const profileName = "profile-name"
// Name the collection
const rekogClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});

const listCollection = async () => {
  var max_results = 10
  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
      });
      return collection_count
  }
}

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

ListCollections オペレーションのリクエスト

`ListCollections` への入力は、返されるコレクションの最大数です。

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

レスポンス内に `MaxResults` でリクエストされた数を超える顔がある場合は、返されるトークンを使用して再度 `ListCollections` を呼び出し、残りの結果セットを取得できます。例:

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

ListCollections オペレーションのレスポンス

Amazon Rekognition はコレクション (`CollectionIds`) の配列に戻ります。別の配列 (`FaceModelVersions`) は、各コレクションの顔の分析に使用する顔モデルのバージョンを返します。たとえば、次の JSON レスポンスで、コレクション `MyCollection` はバージョン 2.0 の顔モデルを使用して顔を分析します。コレクション `AnotherCollection` は、バージョン 3.0 の顔モデルを使用します。詳細については、「モデルのバージョニングについて」を参照してください。

NextToken は、ListCollections を再度呼び出して次の結果セットを取得するために使用するトークンです。

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

コレクションの定義

DescribeCollection オペレーションを使用すると、コレクションに関する以下の情報を取得できます。

- コレクション内でインデックス付けされる顔の数。
- コレクションで使用されているモデルのバージョン。詳細については、「the section called “モデルのバージョニングについて”」を参照してください。
- コレクションの Amazon リソースネーム (ARN)
- コレクション作成日時。

コレクションを定義するには (SDK)

1. まだ実行していない場合:
    a. AmazonRekognitionFullAccess アクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。
    b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 以下の例を使用して、DescribeCollection オペレーションを呼び出します。

Java

この例では、コレクションを定義します。

値 collectionId は、目的のコレクションの ID に変更します。

```
//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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.DescribeCollectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.DescribeCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
//snippet-end:[rekognition.java2.describe_collection.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DescribeCollection {

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <collectionName>\n\n" +
         "Where:\n" +
         "   collectionName - The name of the Amazon Rekognition collection. \n
\n";
```

```
    if (args.length != 1) {
        System.out.println(usage);
        System.exit(1);
    }

    String collectionName = args[0];
    Region region = Region.US_EAST_1;
    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

    describeColl(rekClient, collectionName);
    rekClient.close();
}

// snippet-start:[rekognition.java2.describe_collection.main]
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);
    }
}
// snippet-end:[rekognition.java2.describe_collection.main]
}
```

AWS CLI

この AWS CLI コマンドは、 CLI オペレーションの JSON describe-collection 出力を表示します。collection-id の値は、目的のコレクションの ID に変更します。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
aws rekognition describe-collection --collection-id collection-name --profile
 profile-name
```

Python

この例では、コレクションを定義します。

値 collection_id は、目的のコレクションの ID に変更します。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
# 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)

    session = boto3.Session(profile_name='default')
    client = session.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 = 'collection-name'
    describe_collection(collection_id)

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

.NET

この例では、コレクションを定義します。

値 collectionId は、目的のコレクションの ID に変更します。

```
//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

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

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Set the profile name
const profileName = "profile-name"
// Name the collection
const rekogClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});

// Name the collection
const collection_name = "collection-name"

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 オペレーションのリクエスト

DescribeCollection への入力は、以下の JSON の例に示しているように、定義するコレクションの ID です。

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

DescribeCollection オペレーションのレスポンス

レスポンスは以下のとおりです。

- コレクション内でインデックス付けされる顔の数、FaceCount。
- コレクションで使用されているモデルのバージョン、FaceModelVersion。詳細については、「the section called “モデルのバージョニングについて”」を参照してください。
- コレクションの Amazon リソースネーム、CollectionARN。
- コレクション作成日時、CreationTimestamp。CreationTimestamp の値は、Unix エポック時間からコレクション作成までのミリ秒数です。Unix エポック時間は、1970 年 1 月 1 日木曜日の 00:00:00 協定世界時 (UTC) です。詳細については、「Unix 時間」を参照してください。

```
{
```

```
    "CollectionARN": "arn:aws:rekognition:us-east-1:nnnnnnnnnnnn:collection/
MyCollection",
    "CreationTimestamp": 1.533422155042E9,
    "FaceCount": 200,
    "UserCount" : 20,
    "FaceModelVersion": "1.0"
}
```

コレクションの削除

DeleteCollection オペレーションを使用するとコレクションを削除できます。

詳細については、「コレクションの管理」を参照してください。

コレクションを削除するには (SDK)

1. まだ実行していない場合:
   a. AmazonRekognitionFullAccess アクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。
   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 以下の例を使用して、DeleteCollection オペレーションを呼び出します。

   Java

   この例ではコレクションを削除します。

   collectionId の値は、削除するコレクションに変更します。

```
//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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
// snippet-start:[rekognition.java2.delete_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.DeleteCollectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.DeleteCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
```

```
// snippet-end:[rekognition.java2.delete_collection.import]

/**
 * Before running this Java V2 code example, set up your development
 environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteCollection {

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "   <collectionId> \n\n" +
            "Where:\n" +
            "   collectionId - The id of the collection to delete. \n\n";

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        System.out.println("Deleting collection: " + collectionId);
        deleteMyCollection(rekClient, collectionId);
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.delete_collection.main]
    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);
        }
    }
    // snippet-end:[rekognition.java2.delete_collection.main]
}
```

AWS CLI

この AWS CLI コマンドは、 CLI オペレーションの JSON delete-collection 出力を表示します。collection-id の値は、削除するコレクションの名前に置き換えます。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
aws rekognition delete-collection --collection-id collection-name --profile
 profile-name
```

Python

この例ではコレクションを削除します。

collection_id の値は、削除するコレクションに変更します。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
# 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 delete_collection(collection_id):

    print('Attempting to delete collection ' + collection_id)
    session = boto3.Session(profile_name='default')
    client = session.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 = 'collection-name'
    status_code = delete_collection(collection_id)
    print('Status code: ' + str(status_code))

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

.NET

この例ではコレクションを削除します。

collectionId の値は、削除するコレクションに変更します。

```
//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

Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

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

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Set the profile name
const profileName = "profile-name"
```

```
// Name the collection
const rekogClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});

// Name the collection
const collection_name = "collection-name"

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 オペレーションのリクエスト

DeleteCollection への入力は、削除するコレクションの ID です。JSON の例は次のとおりです。

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

DeleteCollection オペレーションのレスポンス

DeleteCollection レスポンスには、オペレーションの成否を示す HTTP ステータスコードが含まれます。コレクションが正常に削除されると、200 が返されます。

```
{"StatusCode":200}
```

コレクションへの顔の追加

IndexFaces オペレーションを使用すると、イメージ内の顔を検出し、コレクションに追加できます。Amazon Rekognition は、検出した顔ごとに顔の特徴を抽出し、その特徴情報をデータベースに保存します。さらに、検出した顔ごとのメタデータを、指定された顔コレクションに保存します。Amazon Rekognition は、実際のイメージのバイトを保存しません。

インデックス作成に適した顔の提供については、「顔比較用の入力イメージに関する推奨事項」を参照してください。

IndexFaces オペレーションでは、顔ごとに以下の情報が保持されます。

- 顔の多次元特徴 – IndexFaces では顔分析を使用して、顔の特徴に関する多次元情報を抽出し、その情報を顔コレクションに保存します。この情報に直接アクセスすることはできません。ただし、Amazon Rekognition は顔コレクション内で一致する顔を検索するときに、この情報を使用します。

- メタデータ – 各顔のメタデータには、境界ボックス、信頼度 (境界ボックスに顔が含まれている確率)、Amazon Rekognition によって割り当てられた ID (顔 ID とイメージ ID)、およびリクエストの外部イメージ ID (指定した場合) が含まれます。この情報は、IndexFaces API コールのレスポンスで返されます。例については、以下のレスポンスの face 要素を参照してください。

  このメタデータは、以下の API コールのレスポンスで返されます。

  - ListFaces
  - 顔検索オペレーション – SearchFaces と SearchFacesByImage のレスポンスは、一致した顔ごとに一致の信頼度と、一致した顔のメタデータを返します。

IndexFaces でインデックス付けされる顔の数は、入力コレクションに関連付けられている顔検出モデルのバージョンによって異なります。詳細については、「モデルのバージョニングについて」を参照してください。

インデックスが付けられた顔に関する情報が FaceRecord オブジェクトの配列で返されます。

インデックスが付けられた顔を、顔を検出したイメージに関連付けることができます。たとえば、イメージのクライアント側のインデックスとイメージ内の顔を保持できます。顔をイメージと関連付け

るには、ExternalImageId リクエストパラメータでイメージ ID を指定します。イメージ ID として、ファイル名を使用するか、別に作成した ID を使用できます。

API は、顔コレクションに保存する前述の情報に加えて、コレクションに保存されない顔の詳細も返します (以下のレスポンス例の faceDetail 要素を参照してください)。

> **Note**
>
> DetectFaces から同じ情報が返されるため、同じイメージに対して DetectFaces と IndexFaces の両方を呼び出す必要はありません。

顔のフィルタリング

IndexFaces オペレーションにより、インデックスが付けられた顔をイメージからフィルタリングすることができます。IndexFaces を使用すると、インデックスの顔の最大数を指定したり、高品質で検出された顔のみにインデックスを付けることを選択したりできます。

MaxFaces 入力パラメーターを使用して IndexFaces によりインデックスが付けられる顔の最大数を指定することができます。この機能は、イメージ内の最も大きい顔にインデックスを付け、背景に立っている人の顔など、小さい顔にインデックスを付けない場合に役立ちます。

デフォルトでは、IndexFaces は顔を除外するために使用する品質基準を選択します。QualityFilter 入力パラメータを使用して、品質基準を明示的に設定できます。 値は次のとおりです。

- AUTO — Amazon Rekognition は、顔のフィルタリングに使用する品質基準を選択します (デフォルト値)。
- LOW - 最低品質の顔を除くすべての顔がインデックス化されます。
- MEDIUM
- HIGH - 最高品質の顔のみがインデックス化されます。
- NONE - 品質に基づいて顔が除外されません。

IndexFaces は、次の理由により顔をフィルタリングします。

- イメージサイズに比べて顔が小さすぎる。
- 顔が極端にぼやけている。
- イメージが暗すぎる。

- 顔が極端なポーズをしている。
- 顔に、顔検索に適したディテールがない。

> **Note**
>
> 品質フィルタリングを使用するには、バージョン 3 以上の顔モデルに関連付けられているコレクションが必要です。コレクションに関連付けられている顔モデルのバージョンを取得するには、DescribeCollection を呼び出します。

IndexFaces によりインデックスが付けられていない顔に関する情報が UnindexedFace オブジェクトの配列で返されます。Reasons 配列には、顔にインデックスが付けられていない理由のリストが含まれています。たとえば、EXCEEDS_MAX_FACES の値は、MaxFaces により指定された顔の数が既に検出されているためにインデックスが付けられていない顔です。

詳細については、「コレクション内の顔の管理」を参照してください。

コレクションに顔を追加するには (SDK)

1. まだ実行していない場合:
   a. AmazonRekognitionFullAccess と AmazonS3ReadOnlyAccess のアクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。
   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。
2. イメージ (1 つ以上の顔を含むもの) を Amazon S3 バケットにアップロードします。

   手順については、「Amazon S3 へのオブジェクトのアップロード」のために、Amazon Simple ストレージサービスユーザーガイド を参照してください。
3. 以下の例を使用して、IndexFaces オペレーションを呼び出します。

   Java

   この例では、コレクションに追加した顔の識別子を表示します。

   collectionId の値は、顔を追加するコレクションの名前に変更します。bucket と photo の値を、ステップ 2 で使用した Amazon S3 バケットとイメージの名前に置き換えま

す。.withMaxFaces(1) パラメーターは、インデックスが付けられた顔の数を 1 に制限します。必要に応じて値を変更または削除します。

```
//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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
//snippet-start:[rekognition.java2.add_faces_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
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.IndexFacesResponse;
import software.amazon.awssdk.services.rekognition.model.IndexFacesRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.QualityFilter;
import software.amazon.awssdk.services.rekognition.model.Attribute;
import software.amazon.awssdk.services.rekognition.model.FaceRecord;
```

```
import software.amazon.awssdk.services.rekognition.model.UnindexedFace;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.Reason;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
//snippet-end:[rekognition.java2.add_faces_collection.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class AddFacesToCollection {

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "    <collectionId> <sourceImage>\n\n" +
         "Where:\n" +
         "    collectionName - The name of the collection.\n" +
         "    sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png). \n\n";

     if (args.length != 2) {
         System.out.println(usage);
         System.exit(1);
     }

     String collectionId = args[0];
     String sourceImage = args[1];
     Region region = Region.US_EAST_1;
     RekognitionClient rekClient = RekognitionClient.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
         .build();

     addToCollection(rekClient, collectionId, sourceImage);
```

```
    rekClient.close();
}

// snippet-start:[rekognition.java2.add_faces_collection.main]
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);
        }
    }
    // snippet-end:[rekognition.java2.add_faces_collection.main]
}
```

AWS CLI

この AWS CLI コマンドは、 CLI オペレーションの JSON index-faces 出力を表示します。

collection-id の値は、顔を保存する先のコレクションの名前に置き換えます。Bucket と Name の値を、ステップ 2 で使用した Amazon S3 バケットとイメージファイルに置き換えます。max-faces パラメーターは、インデックスが付けられた顔の数を 1 に制限します。必要に応じて値を変更または削除します。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
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" --
profile profile-name
```

Windows デバイスで CLI にアクセスする場合は、パーサーエラーの発生に対処するため、一重引用符の代わりに二重引用符を使用し、内側の二重引用符をバックスラッシュ (\) でエスケープします。例として以下を参照してください。

```
aws rekognition index-faces --image "{\"S3Object\":{\"Bucket\":\"bucket-name\",
\"Name\":\"image-name\"}}" \
--collection-id "collection-id" --max-faces 1 --quality-filter "AUTO" --
detection-attributes "ALL" \
--external-image-id "example-image.jpg" --profile profile-name
```

Python

この例では、コレクションに追加した顔の識別子を表示します。

collectionId の値は、顔を追加するコレクションの名前に変更します。bucket と photo の値を、ステップ 2 で使用した Amazon S3 バケットとイメージの名前に置き換えます。MaxFaces 入力パラメーターは、インデックスが付けられた顔の数を 1 に制限します。必要に応じて値を変更または削除します。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
# 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):

    session = boto3.Session(profile_name='profile-name')
    client = session.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 = 'amzn-s3-demo-bucket'
```

```
    collection_id = 'collection-id'
    photo = 'photo-name'

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

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

.NET

この例では、コレクションに追加した顔の識別子を表示します。

collectionId の値は、顔を追加するコレクションの名前に変更します。bucket と photo の値を、ステップ 2 で使用した Amazon S3 バケットとイメージの名前に置き換えます。

```
//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 = "amzn-s3-demo-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 オペレーションのリクエスト

IndexFaces への入力は、インデックス付けするイメージと顔を追加するコレクションです。

```
{
    "CollectionId": "MyCollection",
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "ExternalImageId": "input.jpg",
    "DetectionAttributes": [
        "DEFAULT"
    ],
    "MaxFaces": 1,
    "QualityFilter": "AUTO"
}
```

IndexFaces オペレーションのレスポンス

IndexFaces は、イメージ内で検出された顔に関する情報を返します。たとえば、次の JSON レスポンスは、入力イメージ内で検出された顔のデフォルトの検出属性を返します。この例は、入力パラメータ MaxFaces の値を超えたためにインデックスが付けられていない顔も示しています。-Reasons 配列には、EXCEEDS_MAX_FACES が含まれます。品質の理由で顔にインデックスが付けられない場合、Reasons には LOW_SHARPNESS や LOW_BRIGHTNESS などの値が含まれます。詳細については、「UnindexedFace」を参照してください。

```
{
    "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"
            ]
        }
    ]
}
```

すべての顔の情報を取得するには、DetectionAttributes リクエストパラメータに「ALL」を指定します。次のレスポンス例で、faceDetail 要素の追加情報はサーバーに保持されないことに注意してください。

- 25 の顔ランドマーク (前の例では 5 つのみ)
- 10 種類の顔属性 (眼鏡、あごひげ、歯並び、視線の方向など)
- 感情 (emotion 要素を参照)

face 要素は、サーバーに保持されるメタデータを提供します。

FaceModelVersion は、コレクションに関連付けられている顔モデルのバージョンです。詳細については、「モデルのバージョニングについて」を参照してください。

OrientationCorrection は、イメージの向きの予測です。バージョン 3 よりも新しいバージョンの顔検出モデルを使用している場合、方向補正情報は返されません。詳細については、「イメージの向きおよび境界ボックス座標の取得」を参照してください。

次のレスポンス例は、["ALL"] を指定したときに返される JSON を示しています。


```
{
    "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"
                    }
                ],
                 "EyeDirection": {
                    "yaw": 16.299732,
                    "pitch": -6.407457,
                    "confidence": 99.968704
                              }
                "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": []
}
```

コレクション内の顔と関連ユーザーを一覧表示

ListFaces オペレーションを使うと、コレクション内の顔と関連ユーザーを一覧表示できます。

詳細については、「コレクション内の顔の管理」を参照してください。

コレクション内の顔を一覧表示するには (SDK)

1. まだ実行していない場合:

   a. AmazonRekognitionFullAccess アクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. 以下の例を使用して、ListFaces オペレーションを呼び出します。

Java

この例では、コレクション内の顔を一覧表示します。

collectionId の値は、目的のコレクションに変更します。

```
//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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
// snippet-start:[rekognition.java2.list_faces_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.Face;
import software.amazon.awssdk.services.rekognition.model.ListFacesRequest;
import software.amazon.awssdk.services.rekognition.model.ListFacesResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;
// snippet-end:[rekognition.java2.list_faces_collection.import]

/**
 * Before running this Java V2 code example, set up your development
 environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListFacesInCollection {

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

```
        final String usage = "\n" +
            "Usage: " +
            "   <collectionId>\n\n" +
            "Where:\n" +
            "   collectionId - The name of the collection. \n\n";

        if (args.length < 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        System.out.println("Faces in collection " + collectionId);
        listFacesCollection(rekClient, collectionId) ;
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.list_faces_collection.main]
    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);
        }
      }
    // snippet-end:[rekognition.java2.list_faces_collection.main]
  }
```

AWS CLI

この AWS CLI コマンドは、 CLI オペレーションの JSON `list-faces` 出力を表示します。`collection-id` の値は、顔を一覧表示するコレクションの名前に置き換えます。Rekognition セッションを作成する行の `profile_name` の値を、自分のデベロッパープロファイル名に置き換えます。

```
aws rekognition list-faces --collection-id "collection-id"  --profile profile-
name
```

Python

この例では、コレクション内の顔を一覧表示します。

Rekognition セッションを作成する行の `profile_name` の値を、自分のデベロッパープロファイル名に置き換えます。

```
# 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

    session = boto3.Session(profile_name='profile-name')
    client = session.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-id'
    faces_count = list_faces_in_collection(collection_id)
    print("faces count: " + str(faces_count))

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

.NET

この例では、コレクション内の顔を一覧表示します。

collectionId の値は、目的のコレクションに変更します。

```
//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 オペレーションのリクエスト

ListFaces への入力は、顔を一覧表示するコレクションの ID です。MaxResults は、返される顔の最大数です。また ListFaces は、結果をフィルタリングするための顔 ID のリストと、指定されたユーザーに関連付けられた顔のみを一覧表示するために指定されたユーザー ID も受け取ります。

```
{
    "CollectionId": "MyCollection",
    "MaxResults": 1
}
```

レスポンス内に MaxResults でリクエストされた数を超える顔がある場合は、返されるトークンを使用して再度 ListFaces を呼び出し、残りの結果セットを取得できます。例:

```
{
    "CollectionId": "MyCollection",
    "NextToken": "sm+5ythT3aeEVIR4WA....",
    "MaxResults": 1
}
```

ListFaces オペレーションのレスポンス

ListFaces からのレスポンスは、指定したコレクションに保存されている顔のメタデータに関する情報です。

- FaceModelVersion – コレクションに関連付けられている顔モデルのバージョンです。詳細については、「モデルのバージョニングについて」を参照してください。
- Faces – コレクション内の顔に関する情報です。これには、BoundingBox、信頼度、イメージ識別子、顔 ID に関する情報が含まれます。詳細については、「Face」を参照してください。
- NextToken – 次の結果セットを取得するために使用するトークンです。

```
{
    "FaceModelVersion": "6.0",
    "Faces": [
        {
            "Confidence": 99.76940155029297,
            "IndexFacesModelVersion": "6.0",
            "UserId": "demoUser2",
            "ImageId": "56a0ca74-1c83-39dd-b363-051a64168a65",
            "BoundingBox": {
                "Width": 0.03177810087800026,
                "Top": 0.36568498611450195,
                "Left": 0.3453829884529114,
                "Height": 0.056759100407361984
            },
            "FaceId": "c92265d4-5f9c-43af-a58e-12be0ce02bc3"
        },
        {
            "BoundingBox": {
                "Width": 0.03254450112581253,
                "Top": 0.6080359816551208,
                "Left": 0.5160620212554932,
                "Height": 0.06347999721765518
            },
```

```
            "IndexFacesModelVersion": "6.0",
            "FaceId": "851cb847-dccc-4fea-9309-9f4805967855",
            "Confidence": 99.94369506835938,
            "ImageId": "a8aed589-ceec-35f7-9c04-82e0b546b024"
        },
        {
            "BoundingBox": {
                "Width": 0.03094629943370819,
                "Top": 0.4218429923057556,
                "Left": 0.6513839960098267,
                "Height": 0.05266290158033371
            },
            "IndexFacesModelVersion": "6.0",
            "FaceId": "c0eb3b65-24a0-41e1-b23a-1908b1aaeac1",
            "Confidence": 99.82969665527344,
            "ImageId": "56a0ca74-1c83-39dd-b363-051a64168a65"
        }
    ]
}
```


コレクションからの顔の削除

DeleteFaces オペレーションを使用すると、コレクションから顔を削除できます。詳細については、「コレクション内の顔の管理」を参照してください。

コレクションから顔を削除するには

1. まだ実行していない場合:
   a. AmazonRekognitionFullAccess アクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。
   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 以下の例を使用して、DeleteFaces オペレーションを呼び出します。

   Java

   この例では、コレクションから 1 つの顔を削除します。

collectionId の値は、削除する顔が含まれているコレクションに変更します。faces の値は、削除する顔の ID に変更します。複数の顔を削除するには、顔 ID を faces 配列に追加します。

```
//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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DeleteFacesRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
// snippet-end:[rekognition.java2.delete_faces_collection.import]

/**
 * Before running this Java V2 code example, set up your development
 environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteFacesFromCollection {
    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "   <collectionId> <faceId> \n\n" +
            "Where:\n" +
            "   collectionId - The id of the collection from which faces are
 deleted. \n\n" +
            "   faceId - The id of the face to delete. \n\n";

        if (args.length != 1) {
            System.out.println(usage);
```

```
            System.exit(1);
        }

        String collectionId = args[0];
        String faceId = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        System.out.println("Deleting collection: " + collectionId);
        deleteFacesCollection(rekClient, collectionId, faceId);
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.delete_faces_collection.main]
    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);
        }
    }
    // snippet-end:[rekognition.java2.delete_faces_collection.main]
}
```

AWS CLI

この AWS CLI コマンドは、 CLI オペレーションの JSON `delete-faces` 出力を表示します。`collection-id` の値は、削除する顔が含まれているコレクションの名前に置き換えま

す。face-ids の値は、削除する顔 ID の配列に置き換えます。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
aws rekognition delete-faces --collection-id "collection-id" --face-ids "faceid"
 --profile profile-name
```

Python

この例では、コレクションから 1 つの顔を削除します。

collectionId の値は、削除する顔が含まれているコレクションに変更します。faces の値は、削除する顔の ID に変更します。複数の顔を削除するには、顔 ID を faces 配列に追加します。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
# 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):

    session = boto3.Session(profile_name='profile-name')
    client = session.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-id'
    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

この例では、コレクションから 1 つの顔を削除します。

collectionId の値は、削除する顔が含まれているコレクションに変更します。faces の値は、削除する顔の ID に変更します。複数の顔を削除するには、顔 ID を faces リストに追加します。

```
//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 オペレーションのリクエスト

DeleteFaces への入力は、顔が含まれているコレクションの ID と、削除する顔の ID の配列です。

```
{
    "CollectionId": "MyCollection",
    "FaceIds": [
        "daf29cac-f910-41e9-851f-6eeb0e08f973"
    ]
}
```

DeleteFaces オペレーションのレスポンス

DeleteFaces レスポンスは、削除した顔の ID の配列を返します。

```
{
    "DeletedFaces": [
        "daf29cac-f910-41e9-851f-6eeb0e08f973"
    ]
}
```

入力時に指定した顔 ID が、現在特定のユーザーに関連付けられている場合は、正当な理由と共に UnsuccessfulFaceDeletions の一部として返されます。

```
{
   "DeletedFaces": [
        "daf29cac-f910-41e9-851f-6eeb0e08f973"
    ],
     "UnsuccessfulFaceDeletions" : [
        {
           "FaceId" : "0b683aed-a0f1-48b2-9b5e-139e9cc2a757",
           "UserId" : "demoUser1",
           "Reason" : ["ASSOCIATED_TO_AN_EXISTING_USER"]
        }
     ]
}
```

ユーザーの作成

CreateUser オペレーションを使用すると、指定した一意のユーザー ID を使って、コレクション内に新しいユーザーを作成できます。その後、この新しく作成したユーザーに複数の顔を関連付けることができます。

ユーザーを作成するには (SDK)

1. まだ実行していない場合:

   a. AmazonRekognitionFullAccess アクセス許可を持つ IAM ユーザーアカウントを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. 以下の例を使用して、CreateUser オペレーションを呼び出します。

   Java

   こちらの Java コード例では、ユーザーを作成しています。

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.CreateUserRequest;
import com.amazonaws.services.rekognition.model.CreateUserResult;


public class CreateUser {

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


        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        //Replace collectionId and userId with the name of the user that you
 want to create in that target collection.

        String collectionId = "MyCollection";
        String userId = "demoUser";
        System.out.println("Creating new user: " +
```

```
                userId);

        CreateUserRequest request = new CreateUserRequest()
                .withCollectionId(collectionId)
                .withUserId(userId);

        rekognitionClient.createUser(request);
    }

}
```

AWS CLI

この AWS CLI コマンドは、 CLI create-user オペレーションを使用してユーザーを作成します。

```
aws rekognition create-user --user-id user-id --collection-id collection-name --
region region-name
--client-request-token request-token
```

Python

こちらの Python コード例では、ユーザーを作成しています。

```
# 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
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def create_user(collection_id, user_id):
    """
    Creates a new User within a collection specified by CollectionId.
    Takes UserId as a parameter, which is a user provided ID which
    should be unique within the collection.
```

```
    :param collection_id: The ID of the collection where the indexed faces will
 be stored at.
    :param user_id: ID for the UserID to be created. This ID needs to be unique
 within the collection.

    :return: The indexFaces response
    """
    try:
        logger.info(f'Creating user: {collection_id}, {user_id}')
        client.create_user(
            CollectionId=collection_id,
            UserId=user_id
        )
    except ClientError:
        logger.exception(f'Failed to create user with given user id:
 {user_id}')
        raise

def main():
    collection_id = "collection-id"
    user_id = "user-id"
    create_user(collection_id, user_id)

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

Go

この Go コード例では、Go SDK V2 AWS を使用してユーザーを作成します。

```
package main

import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/rekognition"
)
```

```
func main() {
 // Load the AWS SDK configuration
 cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRegion("us-
west-2"))
 if err != nil {
  log.Fatalf("Failed to load configuration: %v", err)
 }

 // Create a Rekognition client
 client := rekognition.NewFromConfig(cfg)

 // Set up the input parameters
 input := &rekognition.CreateUserInput{
  CollectionId: aws.String("my-new-collection"), // Replace with your collection
 ID
  UserId:       aws.String("user12345678910"),   // Replace with desired user ID
 }

 // Call the CreateUser operation
 result, err := client.CreateUser(context.TODO(), input)
 if err != nil {
  log.Fatalf("Failed to create user: %v", err)
 }

 // Print out the results
 fmt.Printf("User created successfully:\n")

}
```

ユーザーの削除

DeleteUser オペレーションを使用すると、指定した UserID に基づいて、コレクションからユーザーを削除できます。UserID に関連付けられている顔は、指定した UserID が削除される前に UserID との関連付けが解除されることにご注意ください。

ユーザーを削除するには (SDK)

1. まだ実行していない場合:

a. AmazonRekognitionFullAccess アクセス許可を持つ IAM ユーザーアカウントを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. 以下の例を使用して、DeleteUser オペレーションを呼び出します。

Java

こちらの Java コード例では、ユーザーを削除します。

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.DeleteUserRequest;
import com.amazonaws.services.rekognition.model.DeleteUserResult;


public class DeleteUser {

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


        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        //Replace collectionId and userId with the name of the user that you
 want to delete from that target collection.

        String collectionId = "MyCollection";
        String userId = "demoUser";
        System.out.println("Deleting existing user: " +
                userId);

        DeleteUserRequest request = new DeleteUserRequest()
                .withCollectionId(collectionId)
                .withUserId(userId);

        rekognitionClient.deleteUser(request);
    }

}
```

AWS CLI

この AWS CLI コマンドは、 CLI `create-user` オペレーションを使用してユーザーを削除します。

```
aws rekognition delete-user --collection-id MyCollection
--user-id user-id --collection-id collection-name --region region-name
```

Python

こちらの Python コード例では、ユーザーを削除します。

```
# 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
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def delete_user(collection_id, user_id):
        """
        Delete the user from the given collection

        :param collection_id: The ID of the collection where user is stored.
        :param user_id: The ID of the user in the collection to delete.
        """
        logger.info(f'Deleting user: {collection_id}, {user_id}')
        try:
            client.delete_user(
                CollectionId=collection_id,
                UserId=user_id
            )
        except ClientError:
            logger.exception(f'Failed to delete user with given user id:
 {user_id}')
            raise
```

```
def main():
    collection_id = "collection-id"
    user_id = "user-id"
    delete_user(collection_id, user_id)

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

ユーザーへの顔の関連付け

AssociateFaces オペレーションを使用すると、複数の個別の顔を 1 人のユーザーに関連付けることができます。顔をユーザーに関連付けるには、まずコレクションとユーザーを作成する必要があります。顔ベクトルは、ユーザーベクトルが存在するコレクションと同じコレクション内に存在している必要があります。

顔を関連付けるには (SDK)

1. まだ実行していない場合:

   a. AmazonRekognitionFullAccess アクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. 以下の例を使用して、AssociateFaces オペレーションを呼び出します。

   Java

   この Java コード例では、顔をユーザーに関連付けます。

```
import java.util.Arrays;
import java.util.List;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.AssociateFacesRequest;
import com.amazonaws.services.rekognition.model.AssociateFacesResult;


public class AssociateFaces {
```

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


       AmazonRekognition rekognitionClient =
AmazonRekognitionClientBuilder.defaultClient();

       /* Replace the below configurations to allow you successfully run the
example

          @collectionId: The collection where user and faces are stored
          @userId: The user which faces will get associated to
          @faceIds: The list of face IDs that will get associated to the given
user
          @userMatchThreshold: Minimum User match confidence required for the
face to
                               be associated with a User that has at least one
faceID already associated
        */

       String collectionId = "MyCollection";
       String userId = "demoUser";
       String faceId1 = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";
       String faceId2 = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";
       List<String> faceIds = Arrays.asList(faceid1,faceid2);

       float userMatchThreshold = 0f;
       System.out.println("Associating faces to the existing user: " +
               userId);

       AssociateFacesRequest request = new AssociateFacesRequest()
               .withCollectionId(collectionId)
               .withUserId(userId)
               .withFaceIds(faceIds)
               .withUserMatchThreshold(userMatchThreshold);

       AssociateFacesResult result = rekognitionClient.associateFaces(request);

       System.out.println("Successful face associations: " +
result.getAssociatedFaces().size());
       System.out.println("Unsuccessful face associations: " +
result.getUnsuccessfulFaceAssociations().size());
   }
```

```
}
```

AWS CLI

この AWS CLI コマンドは、CLI associate-faces オペレーションを使用して顔をユーザーと関連付けます。

```
aws rekognition associate-faces --user-id user-id --face-ids face-id-1 face-id-2
--collection-id collection-name
--region region-name
```

Python

この Python コード例では、顔をユーザーに関連付けます。

```
from botocore.exceptions import ClientError
import boto3
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def associate_faces(collection_id, user_id, face_ids):
    """
    Associate stored faces within collection to the given user

    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param user_id: The ID of the user that we want to associate faces to
    :param face_ids: The list of face IDs to be associated to the given user

    :return: response of AssociateFaces API
    """
    logger.info(f'Associating faces to user: {user_id}, {face_ids}')
    try:
        response = client.associate_faces(
            CollectionId=collection_id,
            UserId=user_id,
            FaceIds=face_ids
        )
        print(f'- associated {len(response["AssociatedFaces"])} faces')
```

```
    except ClientError:
        logger.exception("Failed to associate faces to the given user")
        raise
    else:
        print(response)
        return response

def main():
    face_ids = ["faceId1", "faceId2"]
    collection_id = "collection-id"
    user_id = "user-id"
    associate_faces(collection_id, user_id, face_ids)

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

AssociateFaces オペレーションのレスポンス

AssociateFaces のレスポンスには、関連付け解除のリクエストのステータスである UserStatus と、関連付ける FaceIds のリストが含まれています。UnsuccessfulFaceAssociations のリストも返されます。AssociateFaces にリクエストを送信した後、オペレーションが完了するまでに 1 分ほどかかる場合があります。

UserStatus が返されるのはそのためです。ステータスには以下のいずれかの値が表示されます。

- CREATED -「ユーザー」が正常に作成され、現時点でそれに関連付けられている顔がないことを示します。「ユーザー」は、「AssociateFaces」の呼び出しが成功するまではこのステータスを維持します。
- UPDATING -「ユーザー」は、新たに関連付けられた/関連付けを解除された顔を反映するように更新中であり、数秒後に ACTIVE になることを示します。検索結果にこのステータスの「ユーザー」が含まれる場合、お客様は、返される結果からそのユーザーを無視することを選択できます。
- ACTIVE -「ユーザー」が更新され、関連付けられた/関連付けを解除されたすべての顔が反映され、検索可能な状態であることを示します。

```
{
    "UnsuccessfulFaceAssociations": [
        {
            "Reasons": [
```

```
                "LOW_MATCH_CONFIDENCE"
            ],
            "FaceId": "f5817d37-94f6-0000-bfee-1a2b3c4d5e6f",
            "Confidence": 0.9375374913215637
        },
        {
            "Reasons": [
                "ASSOCIATED_TO_A_DIFFERENT_IDENTITY"
            ],
            "FaceId": "851cb847-dccc-1111-bfee-1a2b3c4d5e6f",
            "UserId": "demoUser2"
        }
    ],
    "UserStatus": "UPDATING",
    "AssociatedFaces": [
        {
            "FaceId": "35ebbb41-7f67-2222-bfee-1a2b3c4d5e6f"
        }
    ]
}
```

顔とユーザーとの関連付けの解除

DisassociateFaces オペレーションを使用すると、ユーザー ID と顔 ID の関連付けを削除できます。

顔の関連付けを解除するには (SDK)

1. まだ実行していない場合:

   a. AmazonRekognitionFullAccess アクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. 以下の例を使用して、DisassociateFaces オペレーションを呼び出します。

   Java

   この Java の例では、DisassociateFaces オペレーションを使って FaceID と UserID の関連付けを削除します。

```
import java.util.Arrays;
import java.util.List;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.DisassociateFacesRequest;
import com.amazonaws.services.rekognition.model.DisassociateFacesResult;


public class DisassociateFaces {

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


        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();

        /* Replace the below configurations to allow you successfully run the
 example

           @collectionId: The collection where user and faces are stored
           @userId: The user which faces will get disassociated from
           @faceIds: The list of face IDs that will get disassociated from the
 given user
         */

        String collectionId = "MyCollection";
        String userId = "demoUser";
        String faceId1 = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";
        String faceId2 = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";
        List<String> faceIds = Arrays.asList(faceid1,faceid2);

        System.out.println("Disassociating faces from existing user: " +
                userId);

        DisassociateFacesRequest request = new DisassociateFacesRequest()
                .withCollectionId(collectionId)
                .withUserId(userId)
                .withFaceIds(faceIds)

        DisassociateFacesResult result =
 rekognitionClient.disassociateFaces(request);
```

```
        System.out.println("Successful face disassociations: " +
 result.getDisassociatedFaces().size());
        System.out.println("Unsuccessful face disassociations: " +
 result.getUnsuccessfulFaceDisassociations().size());
    }

}
```

AWS CLI

この AWS CLI コマンドは、FaceID と UserID と DisassociateFacesオペレーションとの関連付けを削除します。

```
aws rekognition disassociate-faces --face-ids list-of-face-ids
--user-id user-id --collection-id collection-name --region region-name
```

Python

この例では、DisassociateFaces オペレーションを使って FaceID と UserID の関連付けを削除します。

```
from botocore.exceptions import ClientError
import boto3
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def disassociate_faces(collection_id, user_id, face_ids):
    """
    Disassociate stored faces within collection to the given user

    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param user_id: The ID of the user that we want to disassociate faces from
    :param face_ids: The list of face IDs to be disassociated from the given
 user

    :return: response of AssociateFaces API
    """
    logger.info(f'Disssociating faces from user: {user_id}, {face_ids}')
```

```
    try:
        response = client.disassociate_faces(
            CollectionId=collection_id,
            UserId=user_id,
            FaceIds=face_ids
        )
        print(f'- disassociated {len(response["DisassociatedFaces"])} faces')
    except ClientError:
        logger.exception("Failed to disassociate faces from the given user")
        raise
    else:
        print(response)
        return response

def main():
    face_ids = ["faceId1", "faceId2"]
    collection_id = "collection-id"
    user_id = "user-id"
    disassociate_faces(collection_id, user_id, face_ids)

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

DisassociateFaces オペレーションのレスポンス

DisassociateFaces のレスポンスには、関連付け解除のリクエストのステータスである UserStatus と、関連付けを解除される FaceIds のリストが含まれています。UnsuccessfulFaceDisassociations のリストも返されます。DisassociateFaces にリクエストを送信した後、オペレーションが完了するまでに 1 分ほどかかる場合があります。UserStatus が返されるのはそのためです。ステータスには以下のいずれかの値が表示されます。

- CREATED -「ユーザー」が正常に作成され、現時点でそれに関連付けられている顔がないことを示します。「ユーザー」は、「AssociateFaces」の呼び出しが成功するまではこのステータスを維持します。
- UPDATING -「ユーザー」は、新たに関連付けられた/関連付けを解除された顔を反映するように更新中であり、数秒後に ACTIVE になることを示します。検索結果にこのステータスの「ユーザー」が含まれる場合、お客様は、返される結果からそのユーザーを無視することを選択できます。

- ACTIVE - 「ユーザー」が更新され、関連付けられた/関連付けを解除されたすべての顔が反映され、検索可能な状態であることを示します。

```
{
    "UserStatus": "UPDATING",
    "DisassociatedFaces": [
        {
            "FaceId": "c92265d4-5f9c-43af-a58e-12be0ce02bc3"
        }
    ],
    "UnsuccessfulFaceDisassociations": [
        {
            "Reasons": [
                "ASSOCIATED_TO_A_DIFFERENT_IDENTITY"
            ],
            "FaceId": "f5817d37-94f6-4335-bfee-6cf79a3d806e",
            "UserId": "demoUser1"
        }
    ]
}
```

コレクション内のユーザーの一覧表示

ListUsers オペレーションを使用すると、UserID および UserStatus を一覧表示できます。UserID に関連付けられている FaceID を表示するには、ListFaces オペレーションを使用します。

ユーザーを一覧表示するには (SDK)

1. まだ実行していない場合:

   a. AmazonRekognitionFullAccess アクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. 以下の例を使用して、ListUsers オペレーションを呼び出します。

Java

この Java の例では、ListUsers オペレーションを使用してコレクション内のユーザーを一覧表示します。

```
import java.util.List;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.ListUsersRequest;
import com.amazonaws.services.rekognition.model.ListUsersResult;
import com.amazonaws.services.rekognition.model.User;

public class ListUsers {

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


        AmazonRekognition amazonRekognition =
 AmazonRekognitionClientBuilder.defaultClient();


        System.out.println("Listing users");
        int limit = 10;
        ListUsersResult listUsersResult = null;
        String paginationToken = null;
        do {
            if (listUsersResult != null) {
                paginationToken = listUsersResult.getNextToken();
            }
            ListUsersRequest request = new ListUsersRequest()
                    .withCollectionId(collectionId)
                    .withMaxResults(limit)
                    .withNextToken(paginationToken);
            listUsersResult = amazonRekognition.listUsers(request);

            List<User> users = listUsersResult.getUsers();
            for (User currentUser: users) {
                System.out.println(currentUser.getUserId() + " : " +
 currentUser.getUserStatus());
            }
        } while (listUsersResult.getNextToken() != null);
    }
```

```
}
```

AWS CLI

この AWS CLI コマンドは、 ListUsersオペレーションを使用してコレクション内のユーザーを一覧表示します。

```
aws rekognition list-users --collection-id collection-id --max-results number-
of-max-results
```

Python

次の例では、ListUsers オペレーションを使用してコレクション内のユーザーを一覧表示します。

```
# 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
import logging
from pprint import pprint

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')


def list_users(collection_id):
    """
    List all users from the given collection

    :param collection_id: The ID of the collection where user is stored.

    :return: response that contains list of Users found within given collection
    """
    logger.info(f'Listing the users in collection: {collection_id}')
    try:
        response = client.list_users(
            CollectionId=collection_id
        )
```

```
        pprint(response["Users"])
    except ClientError:
        logger.exception(f'Failed to list all user from given collection:
 {collection_id}')
        raise
    else:
        return response

def main():
    collection_id = "collection-id"
    list_users(collection_id)

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

ListUsers オペレーションのレスポンス

ListUsers へのリクエストに対するレスポンスには、コレクション内の Users のリストが、ユーザーの UsedId および UserStatus と共に含まれます。

```
{
    "NextToken": "B1asJT3bAb/ttuGgPFV8BZoBZyGQzlUHXbuTNLh48a6enU7kXKw43hpOwizW7LOk/
Gk7Em09lznoq6+FcDCcSq2olrn7A98BLkt5keu+ZRVRUTyrXtT6J7Hmp
+ieQ2an6Zu0qzPfcdPeaJ9eAxG2d0WNrzJgi5hvmjoiSTTfKX3MQz1sduWQkvAAs4hZfhZoKFahFlqWofshCXa/
FHAAY3PL1PjxXbkNeSSMq8V7i1MlKCdrPVykCv9MokpPt7jtNvKPEZGUhxgBTFMxNWLEcFnzAiCWDg91dFy/
LalshPjXA9UVc5Gx9vIJNQ/
eO3cQRghAkCT3FOAiXsLAnA015ODTomZpWWVpqB21wKpI3LYmfAVFrDPGzpbTVlRmLsJm41bkmnBBBw9+DHz1Jn7zW
+qc5Fs3yaHu0f51Xg==",
    "Users": [
        {
            "UserId": "demoUser4",
            "UserStatus": "CREATED"
        },
        {
            "UserId": "demoUser2",
            "UserStatus": "CREATED"
        }
    ]
}
```

顔 ID を使用した顔の検索

SearchFaces オペレーションを使用すると、指定したイメージ内で最大の顔と一致する、コレクション内のユーザーを検索できます。

顔が検出されると、顔 ID が IndexFaces オペレーションレスポンスで返され、コレクションに追加されます。詳細については、「コレクション内の顔の管理」を参照してください。

顔 ID を使用してコレクション内の顔を検索するには (SDK)

1. まだ実行していない場合:
   a. AmazonRekognitionFullAccess アクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。
   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 以下の例を使用して、SearchFaces オペレーションを呼び出します。

   Java

   この例では、顔 ID で識別された顔と一致する顔に関する情報を表示します。

   collectionID の値は、必要な顔が含まれているコレクションに変更します。faceId の値は、検索する顔の識別子に変更します。

```
//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-
xxxxxxxxxxx";

    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();
      }
    }

}
```

コード例を実行します。一致する顔に関する情報が表示されます。

Java V2

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
// snippet-start:[rekognition.java2.match_faces_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.SearchFacesRequest;
import software.amazon.awssdk.services.rekognition.model.SearchFacesResponse;
import software.amazon.awssdk.services.rekognition.model.FaceMatch;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;
// snippet-end:[rekognition.java2.match_faces_collection.import]

/**
 * Before running this Java V2 code example, set up your development
 environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SearchFaceMatchingIdCollection {

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "   <collectionId> <sourceImage>\n\n" +
            "Where:\n" +
            "   collectionId - The id of the collection.  \n" +
            "   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png). \n\n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        String faceId = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
```

```
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        System.out.println("Searching for a face in a collections");
        searchFacebyId(rekClient, collectionId, faceId ) ;
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.match_faces_collection.main]
    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);
        }
    }
    // snippet-end:[rekognition.java2.match_faces_collection.main]
}
```

AWS CLI

この AWS CLI コマンドは、 CLI オペレーションの JSON search-faces 出力を表示します。face-id の値は、検索する顔識別子に置き換えます。collection-id の値は、検索

先のコレクションに置き換えます。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
aws rekognition search-faces --face-id face-id --collection-id "collection-id"
 --profile profile-name
```

Python

この例では、顔 ID で識別された顔と一致する顔に関する情報を表示します。

collectionID の値は、必要な顔が含まれているコレクションに変更します。faceId の値は、検索する顔の識別子に変更します。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
# 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

    session = boto3.Session(profile_name='profile-name')
    client = session.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']) + "%")

    return len(face_matches)

def main():
    face_id = 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
```

```
    collection_id = 'collection-id'

    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

この例では、顔 ID で識別された顔と一致する顔に関する情報を表示します。

collectionID の値は、必要な顔が含まれているコレクションに変更します。faceId の値は、検索する顔の識別子に変更します。

```
//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);
    }
}
```

コード例を実行します。一致する顔に関する情報が表示されます。

SearchFaces オペレーションのリクエスト

顔 ID (顔コレクションに保存されている顔ごとに顔 ID があります) を指定すると、SearchFaces は指定した顔コレクション内で類似の顔を検索します。レスポンスには、検索している顔自体は含まれず、類似した顔のみが含まれます。デフォルトでは、アルゴリズムで 80% を超える類似度が検出された顔が SearchFaces から返されます。類似度は、顔が入力イメージの顔と一致している度合いを示します。必要に応じて、FaceMatchThreshold を使用して別の値を指定できます。

```
{
    "CollectionId": "MyCollection",
    "FaceId": "0b683aed-a0f1-48b2-9b5e-139e9cc2a757",
    "MaxFaces": 2,
    "FaceMatchThreshold": 99
}
```

SearchFaces オペレーションのレスポンス

このオペレーションでは、検索された一致する顔の配列と、入力として渡した顔 ID を返します。

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

レスポンスでは、検索された一致する顔ごとに類似度と顔のメタデータを返します。レスポンス例は次のとおりです。


```
{
   ...
    "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"
            }
        }
    ]
}
```


イメージを使用した顔の検索

SearchFacesByImage オペレーションを使用すると、指定したイメージ内の最大の顔と一致する、コレクション内の顔を検索できます。

詳細については、「コレクション内の顔とユーザーの検索」を参照してください。

イメージを使用してコレクション内の顔を検索するには (SDK)

1. まだ実行していない場合:

   a. AmazonRekognitionFullAccess と AmazonS3ReadOnlyAccess のアクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. イメージ (1 つ以上の有名人の顔が含まれているもの) を S3 バケットにアップロードします。

   手順については、「Amazon S3 へのオブジェクトのアップロード」のために、Amazon Simple ストレージ Service 入門ガイド を参照してください。

3. 以下の例を使用して、SearchFacesByImage オペレーションを呼び出します。

   Java

   この例では、イメージ内の最大の顔と一致する顔の情報を表示します。このコード例では、FaceMatchThreshold パラメータと MaxFaces パラメータの両方を指定して、レスポンスで返される結果を制限します。

   次の例で collectionId の値は、検索先のコレクションに変更します。bucket の値は、入力イメージが含まれているバケットに変更します。photo の値は、入力イメージに変更します。

```
//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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
// snippet-start:[rekognition.java2.search_faces_collection.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
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.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.SearchFacesByImageRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import
 software.amazon.awssdk.services.rekognition.model.SearchFacesByImageResponse;
import software.amazon.awssdk.services.rekognition.model.FaceMatch;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
// snippet-end:[rekognition.java2.search_faces_collection.import]

/**
 * Before running this Java V2 code example, set up your development
 environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SearchFaceMatchingImageCollection {

    public static void main(String[] args) {

        final String usage = "\n" +
            "Usage: " +
            "   <collectionId> <sourceImage>\n\n" +
            "Where:\n" +
```

```
            "   collectionId - The id of the collection.  \n" +
            "   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png). \n\n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        String sourceImage = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-
name"))
            .build();

        System.out.println("Searching for a face in a collections");
        searchFaceInCollection(rekClient, collectionId, sourceImage ) ;
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.search_faces_collection.main]
    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);
        }
    }
    // snippet-end:[rekognition.java2.search_faces_collection.main]
}
```

AWS CLI

この AWS CLI コマンドは、 CLI オペレーションの JSON `search-faces-by-image` 出力を表示します。`Bucket` の値は、ステップ 2 で使用した S3 バケットに置き換えます。`Name` の値は、ステップ 2 で使用したイメージファイル名に置き換えます。`collection-id` の値は、検索先のコレクションに置き換えます。Rekognition セッションを作成する行の `profile_name` の値を、自分のデベロッパープロファイル名に置き換えます。

```
aws rekognition search-faces-by-image --image '{"S3Object":{"Bucket":"bucket-
name","Name":"image-name"}}' \
--collection-id "collection-id" --profile profile-name
```

Windows デバイスで CLI にアクセスする場合は、パーサーエラーの発生に対処するため、一重引用符の代わりに二重引用符を使用し、内側の二重引用符をバックスラッシュ (\) でエスケープします。例として以下を参照してください。

```
aws rekognition search-faces-by-image --image "{\"S3Object\":{\"Bucket\":
\"bucket-name\",\"Name\":\"image-name\"}}" \
--collection-id "collection-id" --profile profile-name
```

Python

この例では、イメージ内の最大の顔と一致する顔の情報を表示します。このコード例では、FaceMatchThreshold パラメータと MaxFaces パラメータの両方を指定して、レスポンスで返される結果を制限します。

以下の例で collectionId の値は、検索先のコレクションに変更します。bucket と photo の値は、ステップ 2 で使用した Amazon S3 バケット名とイメージ名に置き換えます。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
#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='amzn-s3-demo-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

この例では、イメージ内の最大の顔と一致する顔の情報を表示します。このコード例では、FaceMatchThreshold パラメータと MaxFaces パラメータの両方を指定して、レスポンスで返される結果を制限します。

以下の例で collectionId の値は、検索先のコレクションに変更します。bucket と photo の値は、ステップ 2 で使用した Amazon S3 バケット名とイメージ名に置き換えます。

```
//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 = "amzn-s3-demo-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 オペレーションのリクエスト

SearchFacesImageByImage への入力パラメータは、検索先のコレクションとソースイメージの場所です。次の例では、ソースイメージが Amazon S3 バケット (S3Object) に保存されています。また、返される顔の最大数 (Maxfaces) と、顔を返すために一致する必要がある最小限の信頼度 (FaceMatchThreshold) も指定されています。

```
{
    "CollectionId": "MyCollection",
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
        }
    },
    "MaxFaces": 2,
    "FaceMatchThreshold": 99
}
```

SearchFacesByImage オペレーションのレスポンス

入力イメージ (.jpeg または .png) を指定すると、オペレーションでは入力イメージ内で顔を検出してから、指定した顔コレクション内で類似する顔を検索します。

> **Note**
>
> 入力イメージ内で複数の顔が検出された場合は、検出された中で最大の顔を使用して顔コレクション内を検索します。

このオペレーションでは、検索された一致する顔の配列と、入力の顔に関する情報を返します。これには、境界ボックスなどの情報と信頼値 (境界ボックス内に顔が含まれている信頼度) が含まれます。

デフォルトでは、アルゴリズムで 80% を超える類似度が検出された顔が SearchFacesByImage から返されます。類似度は、顔が入力イメージの顔と一致している度合いを示します。必要に応じて、FaceMatchThreshold を使用して別の値を指定できます。レスポンスでは、検索された一致する顔ごとに類似度と顔のメタデータが返されます。レスポンス例は次のとおりです。

```
{
    "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
}
```

ユーザー (顔 ID/ユーザー ID) を検索

SearchUsers オペレーションを使用すると、指定したコレクション内で、指定した顔 ID またはユーザー ID と一致する顔を検索できます。このオペレーションでは、リクエストされた UserMatchThreshold を超える、最も高い類似度スコアを持つ UserIds が一覧表示されます。ユーザー ID は CreateUsers オペレーションで作成されます。詳細については、「コレクション内のユーザーの管理」を参照してください。

ユーザーを検索するには (SDK)

1. まだ実行していない場合:
   a. AmazonRekognitionFullAccess アクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。
   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 以下の例を使用して、SearchUsers オペレーションを呼び出します。

   Java

   この Java の例では、SearchUsers オペレーションを使用してコレクション内のユーザーを検索します。

```
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;
import com.amazonaws.services.rekognition.model.UserMatch;
import com.amazonaws.services.rekognition.model.SearchUsersRequest;
import com.amazonaws.services.rekognition.model.SearchUsersResult;
import com.amazonaws.services.rekognition.model.UserMatch;

public class SearchUsers {
    //Replace collectionId and faceId with the values you want to use.
```

```
   public static final String collectionId = "MyCollection";
   public static final String faceId = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

   public static final String userd = 'demo-user';

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

       AmazonRekognition rekognitionClient =
AmazonRekognitionClientBuilder.defaultClient();

       // Search collection for faces matching the user id.
       SearchUsersRequest request = new SearchUsersRequest()
               .withCollectionId(collectionId)
               .withUserId(userId);

       SearchUsersResult result =
               rekognitionClient.searchUsers(request);

       System.out.println("Printing first search result with matched user and
similarity score");
       for (UserMatch match: result.getUserMatches()) {
           System.out.println(match.getUser().getUserId() + " with similarity
score " + match.getSimilarity());
       }

       // Search collection for faces matching the face id.
       SearchUsersRequest request1 = new SearchUsersRequest()
               .withCollectionId(collectionId)
               .withFaceId(faceId);

       SearchUsersResult result1 =
               rekognitionClient.searchUsers(request1);

       System.out.println("Printing second search result with matched user and
similarity score");
       for (UserMatch match: result1.getUserMatches()) {
           System.out.println(match.getUser().getUserId() + " with similarity
score " + match.getSimilarity());
       }
   }
```

AWS CLI

この AWS CLI コマンドは、 SearchUsersオペレーションを使用してコレクション内のユーザーを検索します。

```
aws rekognition search-users --face-id face-id --collection-id collection-id --
region region-name
```

Python

次の例では、SearchUsers オペレーションを使用してコレクション内のユーザーを検索します。

```
# 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
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def search_users_by_face_id(collection_id, face_id):
    """
    SearchUsers operation with face ID provided as the search source

    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param face_id: The ID of the face in the collection to search for.

    :return: response of SearchUsers API
    """
    logger.info(f'Searching for users using a face-id: {face_id}')
    try:
        response = client.search_users(
            CollectionId=collection_id,
            FaceId=face_id
        )
        print(f'- found {len(response["UserMatches"])} matches')
```

```
        print([f'- {x["User"]["UserId"]} - {x["Similarity"]}%' for x in
 response["UserMatches"]])
    except ClientError:
        logger.exception(f'Failed to perform SearchUsers with given face id:
 {face_id}')
        raise
    else:
        print(response)
        return response

def search_users_by_user_id(collection_id, user_id):
    """
    SearchUsers operation with user ID provided as the search source

    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param user_id: The ID of the user in the collection to search for.

    :return: response of SearchUsers API
    """
    logger.info(f'Searching for users using a user-id: {user_id}')
    try:
        response = client.search_users(
            CollectionId=collection_id,
            UserId=user_id
        )
        print(f'- found {len(response["UserMatches"])} matches')
        print([f'- {x["User"]["UserId"]} - {x["Similarity"]}%' for x in
 response["UserMatches"]])
    except ClientError:
        logger.exception(f'Failed to perform SearchUsers with given face id:
 {user_id}')
        raise
    else:
        print(response)
        return response

def main():
    collection_id = "collection-id"
    user_id = "user-id"
    face_id = "face_id"
    search_users_by_face_id(collection_id, face_id)
    search_users_by_user_id(collection_id, user_id)
```

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

SearchUsers オペレーションのリクエスト

FaceID または UserID を指定すると、SearchUsers が、指定された CollectionID でユーザーが一致するものを検索します。デフォルトでは、SearchUsers は類似度スコアが 80% を超える UserID を返します。類似度は、UserID が、指定された FaceID または UserID とどの程度一致しているかを示します。複数の UserID が返された場合、類似度スコアが高いものから順に一覧表示されます。必要に応じて、UserMatchThreshold を使用して別の値を指定できます。詳細については、「コレクション内のユーザーの管理」を参照してください。

以下は、`UserId` を使った SearchUsers リクエストの例です。

```
{
   "CollectionId": "MyCollection",
   "UserId": "demoUser1",
   "MaxUsers": 2,
   "UserMatchThreshold": 99
}
```

以下は、`FaceId` を使った SearchUsers リクエストの例です。

```
{
    "CollectionId": "MyCollection",
    "FaceId": "bff43c40-cfa7-4b94-bed8-8a08b2205107",
    "MaxUsers": 2,
    "UserMatchThreshold": 99
}
```

SearchUsers オペレーションのレスポンス

`FaceId` を使用して検索した場合、SearchUsers のレスポンスには `SearchedFace` の `FaceId`、`UserMatches` のリスト、各ユーザーの `UserId` と `UserStatus` が含まれます。

```
{
    "SearchedFace": {
        "FaceId": "bff43c40-cfa7-4b94-bed8-8a08b2205107"
    },
    "UserMatches": [
        {
            "User": {
                "UserId": "demoUser1",
                "UserStatus": "ACTIVE"
            },
            "Similarity": 100.0
        },
        {
            "User": {
                "UserId": "demoUser2",
                "UserStatus": "ACTIVE"
            },
            "Similarity": 99.97946166992188
        }
    ],
    "FaceModelVersion": "6"
}
```

UserId を使用して検索した場合、SearchUsers のレスポンスには他のレスポンス要素に加えて SearchedUser の UserId が含まれます。

```
{
    "SearchedUser": {
        "UserId": "demoUser1"
    },
    "UserMatches": [
        {
            "User": {
                "UserId": "demoUser2",
                "UserStatus": "ACTIVE"
            },
            "Similarity": 99.97946166992188
        }
    ],
    "FaceModelVersion": "6"
```

```
}
```

ユーザーの検索 (イメージ)

SearchUsersByImage は、指定されたイメージ内で検出された最大の顔に一致するユーザーの、指定された CollectionID を、コレクション内で検索します。デフォルトでは、SearchUsersByImage は類似度スコアが 80% を超える UserID を返します。類似度は、UserID が、指定されたイメージで検出された最大の顔とどの程度一致しているかを示します。複数の UserID が返された場合、類似度スコアが高いものから順に一覧表示されます。必要に応じて、UserMatchThreshold を使用して別の値を指定できます。詳細については、「the section called “コレクション内のユーザーの管理”」を参照してください。

イメージでユーザーを検索するには (SDK)

1. まだ実行していない場合:

   a. AmazonRekognitionFullAccess アクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. 以下の例を使用して、SearchUsersByImage オペレーションを呼び出します。

   Java

   この Java の例では、SearchUsersByImage オペレーションを使用して、入力されたイメージに基づきコレクション内のユーザーを検索します。

```
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.S3Object;
import com.amazonaws.services.rekognition.model.SearchUsersByImageRequest;
import com.amazonaws.services.rekognition.model.SearchUsersByImageResult;
import com.amazonaws.services.rekognition.model.UserMatch;


public class SearchUsersByImage {
```

```
    //Replace bucket, collectionId and photo with your values.
    public static final String collectionId = "MyCollection";
    public static final String s3Bucket = "bucket";
    public static final String s3PhotoFileKey = "input.jpg";

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

        AmazonRekognition rekognitionClient =
 AmazonRekognitionClientBuilder.defaultClient();


        // Get an image object from S3 bucket.
        Image image = new Image()
                .withS3Object(new S3Object()
                        .withBucket(s3Bucket)
                        .withName(s3PhotoFileKey));

        // Search collection for users similar to the largest face in the image.
        SearchUsersByImageRequest request = new SearchUsersByImageRequest()
                .withCollectionId(collectionId)
                .withImage(image)
                .withUserMatchThreshold(70F)
                .withMaxUsers(2);

        SearchUsersByImageResult result =
                rekognitionClient.searchUsersByImage(request);

        System.out.println("Printing search result with matched user and
 similarity score");
        for (UserMatch match: result.getUserMatches()) {
            System.out.println(match.getUser().getUserId() + " with similarity
 score " + match.getSimilarity());
        }
    }
}
```

AWS CLI

この AWS CLI コマンドは、 SearchUsersByImageオペレーションを使用して、入力イメージに基づいてコレクション内のユーザーを検索します。

```
aws rekognition search-users-by-image --image '{"S3Object":
{"Bucket":"s3BucketName","Name":"file-name"}}' --collection-id MyCollectionId --
region region-name
```

Python

次の例では、SearchUsersByImage オペレーションを使用して、入力されたイメージに基づきコレクション内のユーザーを検索します。

```
# 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
import logging
import os

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')

def load_image(file_name):
    """
    helper function to load the image for indexFaces call from local disk

    :param image_file_name: The image file location that will be used by
 indexFaces call.
    :return: The Image in bytes
    """
    print(f'- loading image: {file_name}')
    with open(file_name, 'rb') as file:
        return {'Bytes': file.read()}

def search_users_by_image(collection_id, image_file):
    """
    SearchUsersByImage operation with user ID provided as the search source

    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param image_file: The image that contains the reference face to search
 for.
```

```
    :return: response of SearchUsersByImage API
    """
    logger.info(f'Searching for users using an image: {image_file}')
    try:
        response = client.search_users_by_image(
            CollectionId=collection_id,
            Image=load_image(image_file)
        )
        print(f'- found {len(response["UserMatches"])} matches')
        print([f'- {x["User"]["UserId"]} - {x["Similarity"]}%' for x in
 response["UserMatches"]])
    except ClientError:
        logger.exception(f'Failed to perform SearchUsersByImage with given
 image: {image_file}')
        raise
    else:
        print(response)
        return response

def main():
    collection_id = "collection-id"
    IMAGE_SEARCH_SOURCE = os.getcwd() + '/image_path'
    search_users_by_image(collection_id, IMAGE_SEARCH_SOURCE)

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

SearchUsersByImage オペレーションのリクエスト

SearchUsersByImage のリクエストには、検索先のコレクションとソースイメージの場所が含まれます。次の例では、ソースイメージは Amazon S3 バケット (S3Object) に保存されています。また、返されるユーザーの最大数 (MaxUsers) と、ユーザーを返すために一致する必要がある最小限の信頼度 (UserMatchThreshold) も指定されています。

```
{
    "CollectionId": "MyCollection",
    "Image": {
        "S3Object": {
            "Bucket": "bucket",
            "Name": "input.jpg"
```

```
        }
    },
    "MaxUsers": 2,
    "UserMatchThreshold": 99
}
```

SearchUsersByImage オペレーションのレスポンス

SearchUsersByImage のレスポンスには SearchedFace の FaceDetail オブジェクトと、UserId、Similarity、UserStatus の UserMatches のリストがそれぞれ含まれます。入力したイメージに複数の顔が含まれている場合は、UnsearchedFaces のリストも返されます。

```
{
    "SearchedFace": {
        "FaceDetail": {
            "BoundingBox": {
                "Width": 0.23692893981933594,
                "Top": 0.19235000014305115,
                "Left": 0.39177176356315613,
                "Height": 0.5437348484992981
            }
        }
    },
    "UserMatches": [
        {
            "User": {
                "UserId": "demoUser1",
                "UserStatus": "ACTIVE"
            },
            "Similarity": 100.0
        },
        {
            "User": {
                "UserId": "demoUser2",
                "UserStatus": "ACTIVE"
            },
            "Similarity": 99.97946166992188
        }
    ],
    "FaceModelVersion": "6",
    "UnsearchedFaces": [
```

```
        {
            "FaceDetails": {
                "BoundingBox": {
                    "Width": 0.031677018851041794,
                    "Top": 0.5593535900115967,
                    "Left": 0.6102562546730042,
                    "Height": 0.0682177022099495
                }
            },
            "Reasons": [
                "FACE_NOT_LARGEST"
            ]
        },
        {
            "FaceDetails": {
                "BoundingBox": {
                    "Width": 0.03254449740052223,
                    "Top": 0.6080358028411865,
                    "Left": 0.516062319278717,
                    "Height": 0.06347997486591339
                }
            },
            "Reasons": [
                "FACE_NOT_LARGEST"
            ]
        }
    ]
}
```

保存したビデオでの顔の検索

保存したビデオまたはストリーミングビデオで検出された人物の顔と一致するコレクション内の顔を検索できます。このセクションでは、保存したビデオ内の顔の検索について説明します。ストリーミングビデオ内の顔の検索方法については、「ストリーミングビデオイベントの操作」を参照してください。

検索する顔は、IndexFaces を使用して最初にコレクション内でインデックスを付ける必要があります。詳細については、「コレクションへの顔の追加」を参照してください。

Amazon Rekognition Video のビデオの顔検索は、Amazon S3 バケットに保存されたビデオを分析する他の Amazon Rekognition Video ビデオオペレーションと同じ非同期ワークフローに従い

ます。保存したビデオ内の顔の検索を開始するには、StartFaceSearch を呼び出して、検索するコレクションの ID を指定します。Amazon Rekognition Video は、動画分析の完了ステータスを Amazon Simple Notification Service (Amazon SNS) トピックに公開します。ビデオ分析が成功したら、GetFaceSearch を呼び出して検索結果を取得します。ビデオ分析の開始と結果の取得の詳細については、「Amazon Rekognition Video オペレーションを呼び出す」を参照してください。

次の手順では、ビデオ内で検出された人物の顔と一致するコレクション内の顔を検索する方法を示します。ビデオ内で一致した人物の追跡データを取得する方法も示します。この手順では、Amazon Simple Queue Service (Amazon SQS) のキューを使用してビデオ分析リクエストの完了ステータスを取得する Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK) のコードを拡張します。

ビデオ内で一致する顔を検索するには (SDK)

1. コレクションを作成します。
2. 顔にインデックスを付けてコレクションに追加します。
3. 「Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)」を実行します。
4. ステップ 3 で作成したクラス `VideoDetect` に次のコードを追加します。

   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);

  }
```

関数 main で、以下の行を置き換えます。

```
       StartLabelDetection(bucket, video);

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

を:

```
       String collection="collection";
       StartFaceSearchCollection(bucket, video, collection);
```

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

Java V2

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例は [こちら] です。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetectFaces {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

        if (args.length != 4) {
            System.out.println(usage);
```

```
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];

        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startFaceDetection(rekClient, channel, bucket, video);
        getFaceResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    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()
```

関数 main で、以下の行を置き換えます。

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

を:

```
    collection='tests'
    analyzer.StartFaceSearchCollection(collection)

    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetFaceSearchCollectionResults()
```

Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK) 以外のビデオ例をすでに実行している場合、置き換えるコードは異なる可能性があります。

5. collection の値は、ステップ 1 で作成したコレクションの名前に変更します。
6. コードを実行します。入力コレクション内の顔と一致するビデオ内の顔の人物が一覧表示されます。一致した人物ごとの追跡データも表示されます。

GetFaceSearch オペレーションレスポンス

GetFaceSearch からの JSON レスポンス例は次のとおりです。

レスポンスでは、入力コレクション内の顔と一致する顔の持ち主として、ビデオ内で検出された人物の配列 (Persons) が返されます。配列要素 (PersonMatch) は、ビデオで人物が一致するたびに生成されます。各 PersonMatch には、入力コレクション内の一致した顔の配列 (FaceMatch)、一致した人物に関する情報 (PersonDetail)、およびビデオで人物が一致した時間が含まれます。

```
{
    "JobStatus": "SUCCEEDED",
    "NextToken": "IJdbzkZfvBRqj8GPV82BPiZKkLOGCqDIsNZG/gQsEE5faTVK9JHOz/
xxxxxxxxxxxxxx",
    "Persons": [
        {
            "FaceMatches": [
                {
                    "Face": {
```

```
                    "BoundingBox": {
                        "Height": 0.527472972869873,
                        "Left": 0.33530598878860474,
                        "Top": 0.2161169946193695,
                        "Width": 0.35503000020980835
                    },
                    "Confidence": 99.90239715576172,
                    "ExternalImageId": "image.PNG",
                    "FaceId": "a2f2e224-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
    }
}
```

ストリーミングビデオのコレクション内での顔検索

Amazon Rekognition Video を使用すると、ストリーミングビデオのコレクションの中から顔を検出および認識できます。Amazon Rekognition Video を使用することにより、ストリーミングビデオの分析を開始および管理するためのストリームプロセッサ (CreateStreamProcessor) を作成できます。

ビデオストリーム内の既知の顔を検出する (顔検索) するため、Amazon Rekognition Video は、Amazon Kinesis Video Streams を使用してビデオストリームを受信および処理します。分析結果は Amazon Rekognition Video から Kinesis データストリームに出力され、その後、クライアントのアプリケーションで読み取ります。

ストリーミングビデオに Amazon Rekognition Video を使用するには、アプリケーションに次のものを実装する必要があります。

- Amazon Rekognition Video にストリーミングビデオを送信する Kinesis ビデオストリーム。詳細については、「Amazon Kinesis Video Streams デベロッパーガイド」を参照してください。
- ストリーミングビデオの分析を管理するための Amazon Rekognition Video ストリームプロセッサ。詳細については、「Amazon Rekognition Video ストリームプロセッサオペレーションの概要」を参照してください。

- Amazon Rekognition Video が Kinesis データストリームに送信する分析結果を読み取るための Kinesis データストリームコンシューマ。詳細については、「Kinesis Data Streams Consumers」を参照してください。

このセクションには、Kinesis ビデオストリームおよびその他の必要なリソースを作成し、ビデオを Amazon Rekognition Video にストリーミングして分析結果を受け取るアプリケーションの作成方法に関する情報が含まれています。

トピック

- Amazon Rekognition Video と Amazon Kinesis のリソースを設定する
- ストリーミングビデオの顔検索
- GStreamer プラグインを使用したストリーミング
- ストリーミングビデオのトラブルシューティング

Amazon Rekognition Video と Amazon Kinesis のリソースを設定する

次の手順では、ストリーミングビデオの顔認識に使用する Kinesis ビデオストリームとその他のリソースをプロビジョニングするための手順を説明します。

前提条件

この手順を実行するには、 AWS SDK for Java がインストールされている必要があります。詳細については、「Amazon Rekognition の開始方法」を参照してください。 AWS アカウント 使用するには、Amazon Rekognition API へのアクセス許可が必要です。詳細については、IAM ユーザーガイドの「Amazon Rekognition で定義されるアクション」を参照してください。

ビデオストリーム内の顔を認識するには (AWS SDK)

1. まだの場合、Amazon Rekognition Video に Kinesis Video Streams と Kinesis Data Streams へのアクセスを許可するために IAM サービスロールを作成します。その ARN をメモします。詳細については、「AmazonRekognitionServiceRole を使用してストリームへのアクセス権を付与する」を参照してください。
2. コレクションを作成し、使用したコレクション識別子をメモします。
3. 手順 2 で作成したコレクションで、検索する顔にインデックスを付けます。
4. Kinesis ビデオストリームの作成 と、ストリームの Amazon リソースネーム (ARN) を記録します。

5. Kinesis データストリームを作成する。ストリーム名の先頭に AmazonRekognition を追加し、ストリームの ARN をメモします。

これで、顔検索ストリームプロセッサを作成し、選択したストリームプロセッサ名を使用してストリームプロセッサを起動できます。

> **Note**
>
> ストリームプロセッサは、Kinesis ビデオストリームにメディアを取り込めることを確認した後に起動する必要があります。

Amazon Rekognition Video へのストリーミングビデオ

Amazon Rekognition Video にビデオをストリーミングするには、Amazon Kinesis Video Streams SDK を使用して、Kinesis ビデオストリームを作成し使用します。`PutMedia` オペレーションは、Amazon Rekognition Video が消費するビデオデータ フラグメント を Kinesis ビデオストリームに書き込みます。通常、各ビデオデータフラグメントは 2 ~ 10 秒間の長さで、自己完結型のビデオフレームのシーケンスを含みます。Amazon Rekognition Video は、3 つのタイプのフレーム (I、B、および P) を持つ H.264 エンコードされたビデオをサポートしています。詳細については、「フレーム間」を参照してください。フラグメントの最初のフレームは、I-frame でなければなりません。I-frame は、他のフレームから独立してデコードできます。

Kinesis ビデオストリームにビデオデータが到着すると、Kinesis Video Streams はフラグメントに一意の数値を割り当てます。例については、「PutMedia API の例」を参照してください。

- Matroska (MKV) でエンコードされたソースからストリーミングする場合は、PutMedia オペレーションを使用して、作成した Kinesis ビデオストリームにソースビデオをストリーミングしてください。詳細については、「PutMedia API の例」を参照してください。
- デバイスカメラからストリーミングする場合は、GStreamer プラグインを使用したストリーミング を参照してください。

Amazon Rekognition Video にリソースへのアクセス権を付与する

AWS Identity and Access Management (IAM) サービスロールを使用して、Kinesis ビデオストリームへの読み取りアクセスを Amazon Rekognition Video に付与します。顔検索のストリームプロセッサを使用している場合は、IAM サービスロールを使用して、Amazon Rekognition Video に Kinesis データストリームへの書き込みアクセス権を付与します。セキュリティモニタリングのストリームプ

ロセッサを使用している場合は、IAM ロールを使用して、Amazon Rekognition Video に Amazon S3 バケットと Amazon SNS トピックへのアクセス権を付与します。

顔検索ストリームプロセッサのアクセス権の付与

個々の Kinesis Video Streams と Kinesis Data Streams に Amazon Rekognition Video へのアクセスを許可する権限ポリシーを作成することができます。

Amazon Rekognition Video に顔検索ストリームプロセッサへのアクセス権を付与するには

1. [IAM JSON ポリシーエディターを使用して新しい許可ポリシーを作成し]、次のポリシーを使用します。video-arn を、希望する Kinesis ビデオストリームの ARN に置き換えます。顔検索ストリームプロセッサを使用している場合は、data-arn を、希望する Kinesis データストリームの ARN に置き換えます。

```
{
    "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. IAM サービスロールを作成する または、既存の IAM サービスロールを更新します。次の情報を使用して、IAM サービスロールを作成します。

    1. サービス名の Rekognition を選択します。
    2. サービスロールのユースケースの Rekognition を選択します。

3. 手順 1 で作成したアクセス権ポリシーを添付します。

3. サービスロールの ARN をメモしておきます。これは、ビデオ分析のオペレーションを開始する際に必要です。

AmazonRekognitionServiceRole を使用してストリームへのアクセス権を付与する

Kinesis ビデオストリームとデータストリームへのアクセスを設定する別の方法として、`AmazonRekognitionServiceRole` アクセス権限ポリシーを使用できます。IAM には、Rekognition サービスロールのユースケースが用意されています。このサービスロールを `AmazonRekognitionServiceRole` アクセス権限ポリシーと共に使用すると、複数の Kinesis Date Streams への書き込みやすべての Kinesis Video Streams の読み取りを行うことができます。Amazon Rekognition Video に複数の Kinesis Data Streams への書き込みアクセスを許可するには、Kinesis Data Streams の名前の前に AmazonRekognition を付けます (例えば、`AmazonRekognitionMyDataStreamName`)。

Amazon Rekognition Video に Kinesis ビデオストリームと Kinesis データストリームへのアクセスを与えるには

1. IAM サービスロールを作成します。次の情報を使用して、IAM サービスロールを作成します。

   1. サービス名の Rekognition を選択します。

   2. サービスロールのユースケースの Rekognition を選択します。

   3. AmazonRekognitionServiceRole アクセス権限ポリシーを選択します。これは、AmazonRekognition が先頭に追加されている Kinesis Data Streams への書き込みアクセスと、すべての Kinesis Video Streams への読み取りアクセスを Amazon Rekognition Video に付与するポリシーです。

2. AWS アカウント のセキュリティを確保するには、使用しているリソースのみに Rekognition のアクセス範囲を制限します。これは、IAM サービスロールに信頼ポリシーをアタッチすることで実行できます。これを行う方法については、「サービス間の混乱した代理の防止」を参照してください。

3. このサービスロールの Amazon リソースネーム (ARN) をメモします。これは、ビデオ分析のオペレーションを開始する際に必要です。

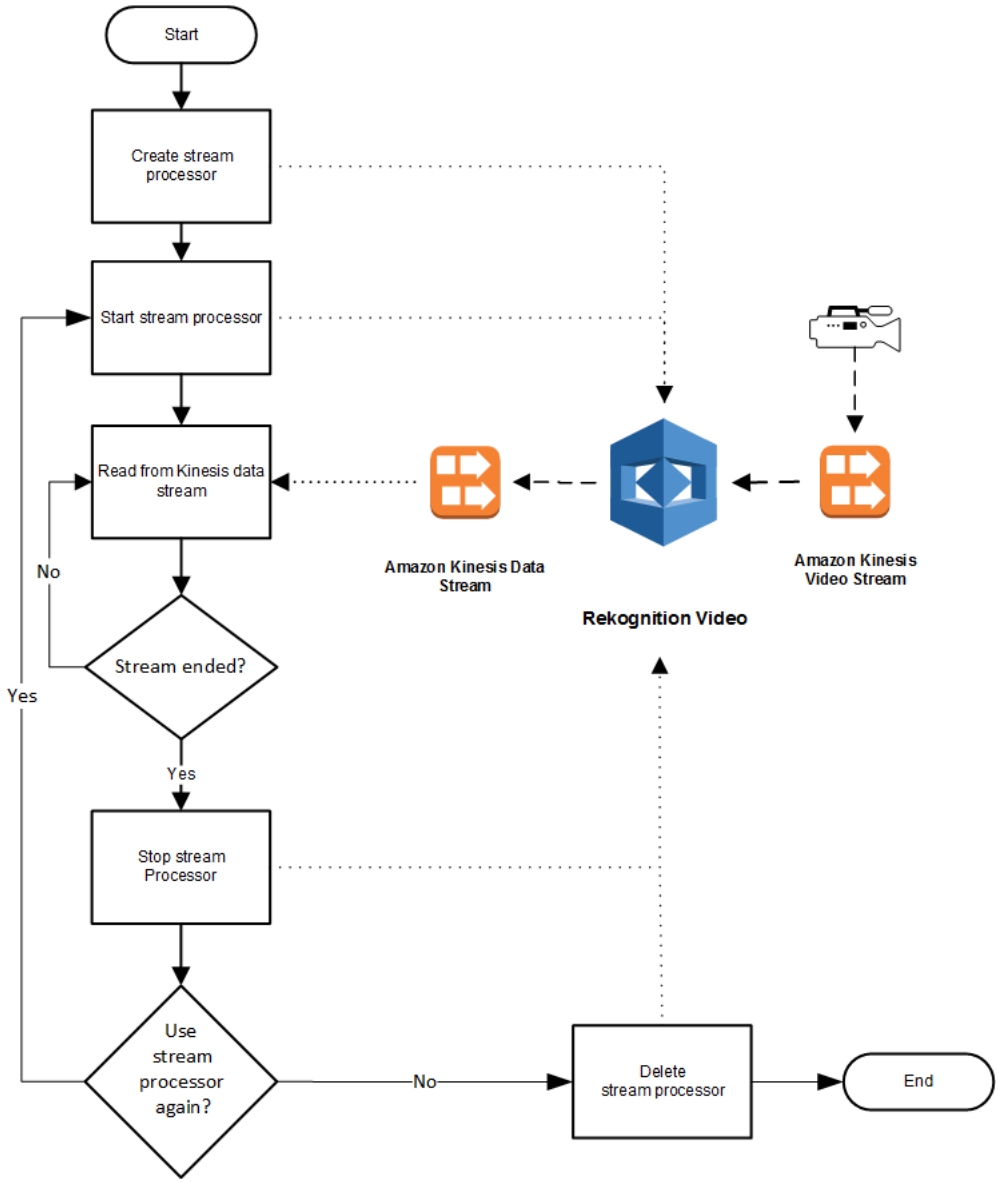
ストリーミングビデオの顔検索

Amazon Rekognition Video では、コレクション内の顔を検索し、ストリーミングビデオで検出された顔と一致する顔を見つけることができます。コレクションの詳細については、「コレクション内での顔の検索」を参照してください。

トピック

- Amazon Rekognition Video 顔検索ストリームプロセッサの作成
- Amazon Rekognition Video 顔検索ストリームプロセッサの起動
- 顔検索用ストリームプロセッサの使用 (Java V2 の例)
- 顔検索用ストリームプロセッサの使用 (Java V1 の例)
- ストリーミングビデオの分析結果の読み取り
- Kinesis Video Streams を使用した Rekognition の結果をローカルに表示する
- Kinesis 顔認識 JSON フレームレコードについて

次の図は、Amazon Rekognition Video がストリーミングビデオ内の顔を検出して認識する原理を示しています。



Amazon Rekognition Video 顔検索ストリームプロセッサの作成

ストリーミングビデオを分析する前に、Amazon Rekognition Video ストリームプロセッサ (CreateStreamProcessor) を作成します。ストリームプロセッサには、Kinesis データストリームと Kinesis ビデオストリームに関する情報が含まれています。また、入力ストリーミングビデオで認識対象とする顔を含むコレクションの識別子も含まれています。ストリームプロセッサの名前も指定します。`CreateStreamProcessor` リクエストに関する JSON の例を次に示します。

```
{
       "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
        }
    }
}
```

以下に、CreateStreamProcessor からのレスポンス例を示します。

```
{
    “StreamProcessorArn”: “arn:aws:rekognition:us-
east-1:nnnnnnnnnnnn:streamprocessor/streamProcessorForCam”
}
```

Amazon Rekognition Video 顔検索ストリームプロセッサの起動

CreateStreamProcessor で指定したストリームプロセッサの名前を使用して StartStreamProcessor を呼び出し、ストリーミングビデオの分析を開始します。StartStreamProcessor リクエストに関する JSON の例を次に示します。

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

ストリームプロセッサが正常に起動した場合は、空の JSON 本文と共に HTTP 200 レスポンスが返されます。

顔検索用ストリームプロセッサの使用 (Java V2 の例)

次のコード例は、 AWS SDK for Java バージョン 2 を使用して CreateStreamProcessor や StartStreamProcessor などのさまざまなストリームプロセッサオペレーションを呼び出す方法を示しています。

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.CreateStreamProcessorRequest;
import software.amazon.awssdk.services.rekognition.model.CreateStreamProcessorResponse;
import software.amazon.awssdk.services.rekognition.model.FaceSearchSettings;
import software.amazon.awssdk.services.rekognition.model.KinesisDataStream;
import software.amazon.awssdk.services.rekognition.model.KinesisVideoStream;
import software.amazon.awssdk.services.rekognition.model.ListStreamProcessorsRequest;
import software.amazon.awssdk.services.rekognition.model.ListStreamProcessorsResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.StreamProcessor;
import software.amazon.awssdk.services.rekognition.model.StreamProcessorInput;
import software.amazon.awssdk.services.rekognition.model.StreamProcessorSettings;
import software.amazon.awssdk.services.rekognition.model.StreamProcessorOutput;
import software.amazon.awssdk.services.rekognition.model.StartStreamProcessorRequest;
import
 software.amazon.awssdk.services.rekognition.model.DescribeStreamProcessorRequest;
import
 software.amazon.awssdk.services.rekognition.model.DescribeStreamProcessorResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class CreateStreamProcessor {
    public static void main(String[] args) {
        final String usage = """
```

```
                Usage:    <role> <kinInputStream> <kinOutputStream> <collectionName>
<StreamProcessorName>

                Where:
                   role - The ARN of the AWS Identity and Access Management (IAM) role
to use. \s
                   kinInputStream - The ARN of the Kinesis video stream.\s
                   kinOutputStream - The ARN of the Kinesis data stream.\s
                   collectionName - The name of the collection to use that contains
content. \s
                   StreamProcessorName - The name of the Stream Processor. \s
                """;

       if (args.length != 5) {
           System.out.println(usage);
           System.exit(1);
       }

       String role = args[0];
       String kinInputStream = args[1];
       String kinOutputStream = args[2];
       String collectionName = args[3];
       String streamProcessorName = args[4];

       Region region = Region.US_EAST_1;
       RekognitionClient rekClient = RekognitionClient.builder()
               .region(region)
               .build();

       processCollection(rekClient, streamProcessorName, kinInputStream,
kinOutputStream, collectionName,
               role);
       startSpecificStreamProcessor(rekClient, streamProcessorName);
       listStreamProcessors(rekClient);
       describeStreamProcessor(rekClient, streamProcessorName);
       deleteSpecificStreamProcessor(rekClient, streamProcessorName);
   }

   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.");
    }
}
```

顔検索用ストリームプロセッサの使用 (Java V1 の例)

次のコード例は、Java V1 を使って、CreateStreamProcessor や StartStreamProcessor など、さまざまなストリームプロセッサのオペレーションを呼び出す方法を示したものです。この例には、ストリームプロセッサのオペレーションを呼び出すメソッドを提供する、ストリームプロセッサマネージャークラス (StreamManager) が含まれています。スタータークラス (Starter) は StreamManager オブジェクトを作成し、さまざまなオペレーションを呼び出します。

例を設定するには:

1. Starter クラスのメンバーフィールドの値を、目的の値に設定します。
2. Starter クラス関数 `main` で、必要な関数呼び出しをコメント解除します。

Starter クラス

```
//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 クラス

```
//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(streamProcessorOutpu

.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());
        }
    }
}
```

ストリーミングビデオの分析結果の読み取り

Amazon Kinesis Data Streams クライアントライブラリを使用して、Amazon Kinesis Data Streams 出力ストリームに送信された分析結果を消費することができます。詳細については、「Kinesis Data Streams からのデータの読み取り」を参照してください。Amazon Rekognition Video は、分析された各フレームの JSON フレームレコードを Kinesis 出力ストリームに配置します。Amazon Rekognition Video は、Kinesis ビデオストリームを通じて渡された全てのフレームを分析するわけではありません。

Kinesis データストリームに送信されたフレームレコードには、フレームがある Kinesis ビデオストリームフラグメント、フレームがあるフラグメント内の場所、フレーム内で認識された顔に関する情報が含まれます。また、ストリームプロセッサに関するステータス情報が含まれます。詳細については、「Kinesis 顔認識 JSON フレームレコードについて」を参照してください。

Amazon Kinesis Video Streams パーサーライブラリには、Amazon Rekognition Video の結果を消費し、オリジナルの Kinesis ビデオストリームと統合するサンプルテストが含まれています。詳細については、「Kinesis Video Streams を使用した Rekognition の結果をローカルに表示する」を参照してください。

Amazon Rekognition Video は、Amazon Rekognition Video の分析情報を Kinesis データストリームにストリーミングします。1 つのレコードの JSON の例を次に示します。

```
{
  "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"
          }
        }
      ]
    }
  ]
}
```


この JSON の例では、以下の点に注意してください。

- InputInformation Amazon Rekognition Video にビデオをストリーミングする際に使用された Kinesis ビデオストリームに関する情報。詳細については、「InputInformation」を参照してください。
- StreamProcessorInformation – Amazon Rekognition Video ストリームプロセッサのステータス情報。Status フィールドに使用できる唯一の値は RUNNING です。詳細については、「StreamProcessorInformation」を参照してください。
- FaceSearchResponse 入力コレクション内の顔と一致するストリーミングビデオ内の顔に関する情報が含まれます。FaceSearchResponse には、DetectedFace オブジェクト (分析されたビデオフレームで検出された顔) が含まれます。検出されたそれぞれの顔について、配列 MatchedFaces には、入力コレクションで見つかった一致する顔オブジェクト (MatchedFace) の配列と類似度スコアが含まれます。

Kinesis ビデオストリームの Kinesis データストリームへのマッピング

Kinesis ビデオストリームフレームを、Kinesis データデータストリームに送信される分析済みフレームにマッピングすることもできます。たとえば、ストリーミングビデオの表示中に、認識された人物の顔の周囲にボックスを表示するなどです。境界ボックスの座標は、Kinesis 顔認識レコードの一部として Kinesis データストリームに送信されます。境界ボックスを正しく表示するには、Kinesis 顔認識レコードと共に送信される時間情報を、ソース Kinesis ビデオストリームの対応するフレームにマッピングする必要があります。

Kinesis ビデオストリームを Kinesis データストリームにマッピングするために使用する手法は、ライブメディア (ライブストリーミングビデオなど) をストリーミングしているか、アーカイブ済みメディア (保存済みビデオなど) をストリーミングしているかによって異なります。

ライブメディアストリーミング中のマッピング

Kinesis ビデオストリームフレームを Kinesis データストリームフレームにマップするには

1. FragmentTimeCodeTypePutMedia オペレーションの入力パラメータ を RELATIVE に設定します。
2. PutMedia を呼び出し、Kinesis ビデオストリームにライブメディアを配信します。
3. Kinesis データストリームから Kinesis 顔認識レコードを受信したら、ProducerTimestamp フィールドから FrameOffsetInSeconds と KinesisVideo の値を保存します。
4. ProducerTimestamp および FrameOffsetInSecondsフィールドの値を一緒に追加して、Kinesis ビデオストリームのフレームに対応するタイムスタンプを計算します。

アーカイブ済みメディアストリーミング中のマッピング

Kinesis ビデオストリームフレームを Kinesis データストリームフレームにマップするには

1. PutMedia を呼び出して、アーカイブ済みメディアを Kinesis ビデオストリームに配信します。
2. Acknowledgement オペレーションの応答から PutMedia オブジェクトを受け取ったら、ペイロード フィールドから FragmentNumber フィールドの値を保存します。FragmentNumber は、MKV クラスタークラスターのフラグメント番号です。
3. Kinesis データストリームから Kinesis 顔認識レコードを受信したら、FrameOffsetInSeconds フィールドから KinesisVideo フィールドの値を保存します。
4. ステップ 2 および 3 で保存した FrameOffsetInSeconds および FragmentNumber の値を使用して、マッピングを計算します。FrameOffsetInSeconds はフラグメントのオフセットであり、FragmentNumber と共に Amazon Kinesis データストリームに送信されます。特定のフラグメント番号のビデオフレームの取得の詳細については、「Amazon Kinesis Video Streams のアーカイブ済みメディア」を参照してください。

Kinesis Video Streams を使用した Rekognition の結果をローカルに表示する

KinesisVideo - Rekognition Examples で提供されている Amazon Kinesis Video Streams パーサーライブラリのテスト例を使用して、Amazon Kinesis Video Streams からフィードに表示された Amazon Rekognition Video の結果を確認することができます。KinesisVideoRekognitionIntegrationExample は、検出されたフェースに境界ボックスを表示し、JFrame を介してローカルにビデオをレンダリングします。このプロセスは、デバイスカメラからのメディア入力を Kinesis ビデオストリームに正常に接続し、Amazon Rekognition ストリームプロセッサを開始したことを前提としています。詳細については、「GStreamer プラグインを使用したストリーミング」を参照してください。

ステップ 1: Kinesis Video Streams パーサーライブラリのインストール

ディレクトリを作成し、GitHub リポジトリをダウンロードするには、次のコマンドを実行します。

```
$ git clone https://github.com/aws/amazon-kinesis-video-streams-parser-library.git
```

ライブラリディレクトリに移動し、次の Maven コマンドを実行して、クリーンインストールを実行します。

```
$ mvn clean install
```

ステップ 2: Kinesis Video Streams と Rekognition の統合サンプルテストの設定

KinesisVideoRekognitionIntegrationExampleTest.java ファイルを開きます。クラスヘッダーの直後の @Ignore を削除する。Amazon Kinesis および Amazon Rekognition リソースからの情報をデータフィールドに入力します。詳細については、「Amazon Rekognition Video と Amazon Kinesis のリソースを設定する」を参照してください。Kinesis ビデオストリームにビデオをストリーミングする場合は、inputStream パラメータを削除してください。

次のコード例を参照してください:

```
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();
```

ステップ 3: Kinesis Video Streams と Rekognition の統合サンプルテストの実行

Kinesis Video Streams にストリーミングしている場合、メディア入力を受信している
ことを確認し、Amazon Rekognition Video Stream プロセッサを実行している状態で

ストリームの分析を開始します。詳細については、「Amazon Rekognition Video ストリームプロセッサオペレーションの概要」を参照してください。JUnit テストとして KinesisVideoRekognitionIntegrationExampleTest クラスを実行します。しばらくすると、新しいウィンドウが開き、検出されたフェースの上に境界ボックスが描画された Kinesis ビデオストリームからのビデオフィードが表示されます。

> **Note**
>
> この例で使用されるコレクション内のフェースは、境界ボックスラベルに意味のあるテキスト (PersonName1-Trusted、PersonName2-Intruder、PersonName3-neutral など) を表示するために、この形式で外部イメージ ID (ファイル名) を指定する必要があります。ラベルは色分けもでき、FaceType.java ファイルでカスタマイズできます。

Kinesis 顔認識 JSON フレームレコードについて

Amazon Rekognition Video は、ストリーミングビデオ内で顔を認識できます。Amazon Rekognition Video は、分析された各フレームの JSON フレームレコードを Kinesis データストリームに配置します。Amazon Rekognition Video では、Kinesis ビデオストリームを通じて渡されたフレームはすべて分析されません。

JSON フレームレコードには、入力ストリームと出力ストリーム、ストリームプロセッサのステータス、および分析されたフレームで認識された顔に関する情報が含まれます。このセクションでは、JSON フレームレコードに関するリファレンス情報を紹介します。

以下に示しているのは、Kinesis データストリームのレコードの JSON 構文です。詳細については、「ストリーミングビデオイベントの操作」を参照してください。

> **Note**
>
> Amazon Rekognition Video API は、入力ストリーム内の顔を顔のコレクションと比較し、見つかった最も近い顔と類似度スコアを返します。

```
{
    "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 レコード

JSON レコードには、Amazon Rekognition Video によって処理されたフレームに関する情報が含まれます。レコードには、ストリーミングビデオ、分析されたフレームのステータス、およびフレームで認識された顔に関する情報が含まれます。

InputInformation

Amazon Rekognition Video にビデオをストリーミングする際に使用された Kinesis ビデオストリームに関する情報。

型: InputInformation オブジェクト

StreamProcessorInformation

Amazon Rekognition Video ストリームプロセッサに関する情報。これには、ストリームプロセッサの現在のステータスに関する情報が含まれます。

型: StreamProcessorInformation オブジェクト

FaceSearchResponse

ストリーミングビデオフレームで検出された顔と入力コレクション内で見つかった一致する顔に関する情報。

型: FaceSearchResponse オブジェクト配列

InputInformation

Amazon Rekognition Video によって使用されたソースビデオストリームに関する情報。詳細については、「ストリーミングビデオイベントの操作」を参照してください。

KinesisVideo

型: KinesisVideo オブジェクト

KinesisVideo

Amazon Rekognition Video にソースビデオをストリーミングする際に使用された Kinesis ビデオストリームに関する情報。詳細については、「ストリーミングビデオイベントの操作」を参照してください。

StreamArn

Kinesis ビデオストリームの Amazon リソースネーム (ARN)。

タイプ: 文字列

FragmentNumber

このレコードを表すフレームが含まれるストリーミングビデオのフラグメント。

タイプ: 文字列

ProducerTimestamp

フラグメントのプロデューサー側の Unix タイムスタンプ。詳細については、「PutMedia」を参照してください。

タイプ: 数値

ServerTimestamp

フラグメントのサーバー側の Unix タイムスタンプ。詳細については、「PutMedia」を参照してください。

タイプ: 数値

FrameOffsetInSeconds

フラグメント内のフレームのオフセット (秒)。

タイプ: 数値

StreamProcessorInformation

ストリームプロセッサに関するステータス情報。

ステータス

ストリームプロセッサの現在のステータス。使用できる唯一の値は RUNNING です。

タイプ: 文字列

FaceSearchResponse

ストリーミングビデオフレームで検出された顔と、検出された顔に一致するコレクション内の顔に関する情報。コレクションは CreateStreamProcessor への呼び出しで指定します。詳細については、「ストリーミングビデオイベントの操作」を参照してください。

DetectedFace

分析されたビデオフレームで検出された顔の詳細情報。

型: DetectedFace オブジェクト

MatchedFaces

DetectedFace で検出された顔と一致するコレクション内の顔の詳細情報の配列。

型: MatchedFace オブジェクト配列

DetectedFace

ストリーミングビデオフレーム内で検出された顔に関する情報。入力コレクション内の一致した顔は MatchedFace オブジェクトフィールドにあります。

BoundingBox

分析されたビデオフレーム内で検出された顔の境界ボックスの座標。BoundingBox オブジェクトには、イメージ分析に使用された BoundingBox オブジェクトと同じプロパティが含まれます。

タイプ: BoundingBox オブジェクト

Confidence

検出された顔が実際に顔であることを Amazon Rekognition Video が信頼するレベル (1 ~ 100)。1 が最も低い信頼度、100 が最も高い信頼度。

タイプ: 数値

Landmarks

顔の特徴の配列。

タイプ: Landmark オブジェクト配列

Pose

ピッチ、ロール、ヨーイングによって特定される顔の表情を示します。

タイプ: Pose オブジェクト

Quality

顔イメージの明るさとシャープネスを示します。

タイプ: ImageQuality オブジェクト

MatchedFace

分析されたビデオフレーム内で検出された顔に一致する顔に関する情報。

Face

DetectedFace オブジェクトの顔と一致する入力コレクション内の顔に関する顔の一致情報。

タイプ: Face オブジェクト

類似度

顔の一致の信頼度のレベル (1 ~ 100)。1 が最も低い信頼度、100 が最も高い信頼度です。

タイプ: 数値

GStreamer プラグインを使用したストリーミング

Amazon Rekognition Video は、デバイスカメラからのライブストリーミングビデオを分析できます。デバイスソースからのメディア入力にアクセスするには、GStreamer をインストールする必要があります。GStreamer は、メディアソースと処理ツールをワークフローパイプラインで接続するサードパーティー製のマルチメディアフレームワークソフトウェアです。また、Gstreamer 用の Amazon Kinesis Video Streams プロデューサープラグイン をインストールする必要があります。このプロセスでは、Amazon Rekognition Video および Amazon Kinesis リソースが正常にセットアッ

プされていることを前提としています。詳細については、「Amazon Rekognition Video と Amazon Kinesis のリソースを設定する」を参照してください。

ステップ 1: GStreamer をインストールする

サードパーティー製のマルチメディアプラットフォームソフトウェアである Gstreamer をダウンロードしてインストールします。Homebrew のようなパッケージ管理ソフト (Gstreamer on Homebrew) を使うか、Freedesktop のウェブサイト から直接入手することができます。

コマンドラインターミナルからテストソースを使用してビデオフィードを起動して、Gstreamer が正常にインストールされたことを確認します。

```
$ gst-launch-1.0 videotestsrc ! autovideosink
```

ステップ 2: Kinesis Video Streams Producer プラグインをインストールする

このセクションでは、Amazon Kinesis Video Streams プロデューサーライブラリ をダウンロードし、Kinesis Video Streams Gstreamer プラグインをインストールします。

ディレクトリを作成し、GitHub リポジトリからソースコードをクローンします。`--recursive` パラメータを必ず入れてください。

```
$ git clone --recursive https://github.com/awslabs/amazon-kinesis-video-streams-
producer-sdk-cpp.git
```

ライブラリが提供する説明書 に従って、プロジェクトの設定とビルドを行います。オペレーティングシステムに対応したプラットフォーム固有のコマンドを使用してください。`-DBUILD_GSTREAMER_PLUGIN=ON` を実行して、Kinesis Video Streams Gstreamer プラグインのインストールする際に、`cmake` パラメータを使用します。このプロジェクトでは、インストールに含まれる次の追加パッケージが必要です。GCC または Clang、Curl、Openssl、および log4CPlus。パッケージが見つからないためにビルドが失敗した場合は、パッケージがインストールされていて PATH にインストールされていることを確認してください。ビルド中に [C コンパイル済みプログラムを実行できません] というエラーが発生した場合は、ビルドコマンドを再度実行してください。正しい C コンパイラが見つからないことがあります。

次のコマンドを実行して Kinesis Video Streams プラグインのインストールを確認します。

```
$ gst-inspect-1.0 kvssink
```

ファクトリやプラグインの詳細など、次の情報が表示されます。

```
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/

  ...
```

ステップ 3: Kinesis Video Streams プラグインで Gstreamer を実行する

デバイスカメラから Kinesis Video Streams へのストリーミングを開始する前に、メディアソースを Kinesis Video Streams で許容されるコーデックに変換する必要がある場合があります。現在マシンに接続されているデバイスの仕様とフォーマット機能を調べるには、次のコマンドを実行します。

```
$ gst-device-monitor-1.0
```

ストリーミングを開始するには、以下のサンプルコマンドで Gstreamer を起動し、認証情報と Amazon Kinesis Video Streams 情報を追加します。Amazon Rekognition に Kinesis ストリームへのアクセスを許可する 際に、作成した IAM サービスロールのアクセスキーとリージョンを使用する必要があります。アクセスキーの詳細については、IAM ユーザーガイド の「IAM ユーザーのアクセスキーの管理」を参照してください。また、ビデオフォーマットの引数パラメータは、使用状況に応じて、お使いのデバイスから利用できるように調整することができます。

```
$ 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"
```

その他の起動コマンドについては、「GStreamer 起動コマンドの例」を参照してください。

> **Note**
>
> 起動コマンドがノンネゴシエーションエラーで終了する場合は、デバイスモニターの出力をチェックし、videoconvert パラメータの値が、デバイスの有効な機能であることを確認してください。

数秒後に Kinesis ビデオストリームにデバイスカメラからのビデオフィードが表示されます。Amazon Rekognition で顔の検出とマッチングを開始するには、Amazon Rekognition Video ストリームプロセッサを起動します。詳細については、「Amazon Rekognition Video ストリームプロセッサオペレーションの概要」を参照してください。

ストリーミングビデオのトラブルシューティング

このトピックでは、Amazon Rekognition Video を使用したストリーミングビデオにおけるトラブルシューティングについて説明します。

トピック

- 使用するストリーミングプロセッサが正常に作成されたかどうかがわかりません。
- ストリームプロセッサを正しく設定しているかがわかりません
- 使用するストリームプロセッサが結果を返しません
- ストリームプロセッサの状態が FAILED (失敗) になる
- 使用するストリームプロセッサが予測された結果を返さない

使用するストリーミングプロセッサが正常に作成されたかどうかがわかりません。

次の AWS CLI コマンドを使用して、ストリームプロセッサとその現在のステータスのリストを取得します。

```
aws rekognition list-stream-processors
```

次の AWS CLI コマンドを使用して、追加の詳細を取得できます。stream-processor-name を必要なストリームプロセッサの名前に置き換えます。

```
aws rekognition describe-stream-processor --name stream-processor-name
```

ストリームプロセッサを正しく設定しているかがわかりません

コードで Amazon Rekognition Video の分析結果が出力されない場合、ストリームプロセッサが正常に設定されていない可能性があります。ストリームプロセッサが正しく設定され、結果を生成することができることを確認するには、次を実行します。

ソリューションが正しく設定されているかどうか判断するには

1. 次のコマンドを実行して、ストリームプロセッサが実行状態にあることを確認します。stream-processor-name を使用するストリームプロセッサの名前に変更します。ストリームプロセッサが実行されている場合、Status の値は RUNNING です。ステータスが RUNNING であり、結果が表示されない場合には、「使用するストリームプロセッサが結果を返しません」を参照してください。ステータスが FAILED の場合には、「ストリームプロセッサの状態が FAILED (失敗) になる」を参照してください。

   ```
   aws rekognition describe-stream-processor --name stream-processor-name
   ```

2. ストリームプロセッサが実行されている場合には、次の Bash あるいは PowerShell コマンドを実行して、出力 Kinesis データストリームからデータを読み込みます。

   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. Base64 デコードウェブサイトでデコードツールを使用して、人間が読み取れる文字列に出力をデコードします。詳細については、「ステップ 3: レコードの取得」を参照してください。

4. コマンドが動作し、Kinesis データストリームに顔検出結果が表示される場合、ソリューションは適切に設定されています。コマンドが失敗した場合は、他のトラブルシューティング解決策を試し、「Amazon Rekognition Video にリソースへのアクセス権を付与する」を参照してください。

または、kinesis-process-record AWS Lambda 」ブループリントを使用して、Kinesis データストリームから CloudWatch にメッセージをログ記録し、継続的に視覚化することもできます。これにより、 AWS Lambda と CloudWatch の追加コストが発生します。

使用するストリームプロセッサが結果を返しません

いくつかの要因によりストリームプロセッサが結果を返さないことがあります。

要因 1: ストリームプロセッサが正しく設定されていない

ストリームプロセッサが正しく設定されていない可能性があります。詳細については、「ストリームプロセッサを正しく設定しているかがわかりません」を参照してください。

要因 2: ストリームプロセッサが RUNNING (実行中) ステータスではない

ストリームプロセッサのステータスをトラブルシューティングするには

1. 次の AWS CLI コマンドを使用して、ストリームプロセッサのステータスを確認します。

```
aws rekognition describe-stream-processor --name stream-processor-name
```

2. Status の値が STOPPED の場合、ストリームプロセッサを次のコマンドで開始します。

```
aws rekognition start-stream-processor --name stream-processor-name
```

3. Status の値が FAILED の場合、「ストリームプロセッサの状態が FAILED (失敗) になる」を参照してください。
4. Status の値が STARTING の場合、2 分ほど待機してから、ステップ 1 を繰り返してステータスを確認します。ステータスの値が依然として STARTING の場合には、次の操作を行います。

   a. 次のコマンドを実行して、ストリームプロセッサを削除します。

   ```
   aws rekognition delete-stream-processor --name stream-processor-name
   ```

   b. 同じ設定で新しいストリームプロセッサを作成します。詳細については、「ストリーミングビデオイベントの操作」を参照してください。

c. それでも問題が解決しない場合は、 AWS サポートにお問い合わせください。

5. `Status` の値が `RUNNING` の場合、「要因 3: Kinesis ビデオストリームにアクティブデータがない」を参照してください。

要因 3: Kinesis ビデオストリームにアクティブデータがない

Kinesis ビデオストリームにアクティブなデータがあるかどうかをチェックするには

1. にサインインし AWS Management Console、https://console.aws.amazon.com/kinesisvideo/ で Amazon Kinesis Video Streams コンソールを開きます。
2. Amazon Rekognition ストリームプロセッサの入力である Kinesis ビデオストリームを選択します。
3. プレビュー状態が [ストリームにデータなし] の場合は、入力ストリームに Amazon Rekognition Video が処理するデータはありません。

Kinesis Video Streams を使用してビデオを作成する詳細については、「Kinesis Video Streams プロデューサーライブラリ」を参照してください。

ストリームプロセッサの状態が FAILED (失敗) になる

ストリームプロセッサの状態は、次の AWS CLI コマンドを使用して確認できます。

```
aws rekognition describe-stream-processor --name stream-processor-name
```

ステータスの値が FAILED (失敗) の場合、トラブルシューティング情報で次のエラーメッセージを参照します。

エラー: "Access denied to Role"

ストリームプロセッサが使用する IAM ロールが存在しない、または Amazon Rekognition Video にこのロールを引き受ける権限がありません。

IAM ロールへのアクセスをトラブルシューティングするには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. 左のナビゲーションペインから [ロール] を選択し、ロールが存在することを確認します。

3. ロールが存在する場合は、このロールに AmazonRekognitionServiceRole アクセス許可ポリシーがあることを確認します。
4. ロールが存在しない、あるいは適切なアクセス許可がない場合には、「Amazon Rekognition Video にリソースへのアクセス権を付与する」を参照してください。
5. 次の AWS CLI コマンドを使用してストリームプロセッサを起動します。

```
aws rekognition start-stream-processor --name stream-processor-name
```

エラー:「Kinesis ビデオへのアクセス拒否またはKinesis データへのアクセス拒否」

ロールには、Kinesis Video Streams API オペレーション `GetMedia` および `GetDataEndpoint` へのアクセス権がありません。また、Kinesis Data Streams API オペレーション `PutRecord` および `PutRecords` へのアクセス権がない可能性もあります。

API アクセス権限をトラブルシューティングするには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. ロールを開き、次のアクセス許可ポリシーがアタッチされていることを確認します。

```
{
    "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. いずれかのアクセス許可がない場合には、ポリシーを更新します。詳細については、「Amazon Rekognition Video にリソースへのアクセス権を付与する」を参照してください。

エラー: "Stream ***input-video-stream-name*** doesn't exist"

ストリームプロセッサへの Kinesis ビデオストリームの入力が存在しないか、正しく設定されていません。

Kinesis ビデオストリームをトラブルシューティングするには

1. 次のコマンドを使用して、ストリームがあることを確認します。

```
aws kinesisvideo list-streams
```

2. ストリームが存在する場合、次を確認します。

   - Amazon リソースネーム (ARN) は、ストリームプロセッサの入力ストリームの ARN と同じです。
   - Kinesis ビデオストリームは、ストリームプロセッサと同じリージョンにあります。

   ストリームプロセッサが正しく設定されていない場合は、次の AWS CLI コマンドを使用して削除します。

```
aws rekognition delete-stream-processor --name stream-processor-name
```

3. 目的の Kinesis ビデオストリームで新しいストリームプロセッサを作成します。詳細については、「Amazon Rekognition Video 顔検索ストリームプロセッサの作成」を参照してください。

エラー: "Collection not found"

顔と一致するためにストリームプロセッサによって使用される Amazon Rekognition コレクションが存在しない、または間違ったコレクションが使用されています。

コレクションを確認するには

1. 次の AWS CLI コマンドを使用して、必要なコレクションが存在するかどうかを確認します。ストリームプロセッサを実行している AWS リージョン`region`に変更します。

```
aws rekognition list-collections --region region
```

必要なコレクションが存在しない場合には、新しいコレクションを作成して顔情報を追加します。詳細については、「コレクション内での顔の検索」を参照してください。

2. CreateStreamProcessor の呼び出しで、CollectionId 入力パラメータの値が正確であることを確認します。
3. 次の AWS CLI コマンドを使用してストリームプロセッサを起動します。

```
aws rekognition start-stream-processor --name stream-processor-name
```

エラー: "Stream ***output-kinesis-data-stream-name*** under account ***account-id*** not found"

ストリームプロセッサによって使用される出力 Kinesis データストリームは、 に存在しない AWS アカウント か、ストリームプロセッサと同じ AWS リージョンにありません。

Kinesis データストリームをトラブルシューティングするには

1. 次の AWS CLI コマンドを使用して、Kinesis データストリームが存在するかどうかを確認します。ストリームプロセッサを使用している AWS リージョンregionに変更します。

```
aws kinesis list-streams --region region
```

2. Kinesis データストリームが存在する場合、Kinesis データストリーム名がストリームプロセッサによって使用される出力ストリームの名前と同じであることを確認します。
3. Kinesis データストリームが存在しない場合は、別の AWS リージョンに存在する可能性があります。Kinesis データストリームは、ストリームプロセッサと同じリージョンにある必要があります。
4. 必要に応じて、新しい Kinesis データストリームを作成します。

   a. ストリームプロセッサによって使用される名前と同じ名前の Kinesis データストリームを作成します。詳細については、「ステップ 1: データストリームを作成する」を参照してください。

   b. 次の AWS CLI コマンドを使用してストリームプロセッサを起動します。

```
aws rekognition start-stream-processor --name stream-processor-name
```

使用するストリームプロセッサが予測された結果を返さない

ストリームプロセッサが予測された顔の一致を返さない場合、次の情報を使用します。

- コレクション内での顔の検索
- カメラセットアップの推奨事項 (ストリーミングビデオ)

人物の動線の検出

> **Note**
>
> サポート終了のお知らせ: 2025 年 10 月 31 日をもって、AWS は Amazon Rekognition の人物の動線の検出機能のサポートを終了いたします。2025 年 10 月 31 日のサポート終了後、Rekognition の人物の動線の検出機能はご利用いただけなくなります。詳細については、こちらのブログ記事をご覧ください。

Amazon Rekognition Video は、ビデオ内の人物を動線を追跡し、以下のような情報を提供できます。

- 動線の追跡時点におけるビデオフレーム内の人物の位置。
- 検出時の顔のランドマーク (左目の位置など)。

Amazon Rekognition Video による保存済みビデオ内の人物の動線の検出は非同期オペレーションです。ビデオ内の人の追跡を開始するには、StartPersonTracking を呼び出します。Amazon Rekognition Video は、ビデオ分析の完了ステータスを Amazon Simple Notification Service トピックに発行します。ビデオ分析が成功したら、GetPersonTracking を呼び出してビデオ分析の結果を取得します。Amazon Rekognition Video API オペレーションの詳細については、Amazon Rekognition Video オペレーションを呼び出す を参照してください。

以下の手順では、Amazon S3 バケットに保存されたビデオを通じた人物の動線の追跡方法を示します。この例では、Amazon Simple Queue Service のキューを使用してビデオ分析リクエストの完了ステータスを取得する Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK) のコードを拡張します。

Amazon S3 バケットに保存されたビデオ内のテキストを検出するには (SDK)

1. 「Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)」を実行します。
2. ステップ 1 で作成したクラス `VideoDetect` に次のコードを追加します。

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);

       }
```

関数 main で、以下の行を置き換えます。

```
       StartLabelDetection(amzn-s3-demo-bucket, video);

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

を:

```
        StartPersonDetection(amzn-s3-demo-bucket, video);

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

Java V2

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例は [こちら] です。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import
 software.amazon.awssdk.services.rekognition.model.StartPersonTrackingRequest;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartPersonTrackingResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetPersonTrackingResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetPersonTrackingRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.PersonDetection;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoPersonDetection {
    private static String startJobId = "";

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

```
        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
(for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
(IAM) role to use.\s
                """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startPersonLabels(rekClient, channel, bucket, video);
        getPersonDetectionResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    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
```

関数 main で、以下の行を置き換えます。

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

を:

```
    analyzer.StartPersonPathing()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetPersonPathingResults()
```

CLI

次の AWS CLI コマンドを実行して、ビデオ内の人物パスを開始します。

```
aws rekognition start-person-tracking --video "{"S3Object":{"Bucket":"amzn-s3-
demo-bucket","Name":"video-name"}}" \
--notification-channel "{"SNSTopicArn":"topic-ARN","RoleArn":"role-ARN"}" \
--region region-name --profile profile-name
```

以下の値を更新します。

- amzn-s3-demo-bucket と video-name を、ステップ 2 で指定した Amazon S3 バケット名とファイル名に変更します。
- region-name を、使用している AWS リージョンに変更します。
- Rekognition セッションを作成する行の profile-name の値を、自分のデベロッパープロファイル名に置き換えます。
- topic-ARN を、Amazon Rekognition Video の設定 のステップ 3 で作成した Amazon SNS トピックの ARN に変更します。
- role-ARN を、Amazon Rekognition Video の設定 のステップ 7 で作成した IAM サービスロールの ARN に変更します。

Windows デバイスで CLI にアクセスする場合は、パーサーエラーの発生に対処するため、一重引用符の代わりに二重引用符を使用し、内側の二重引用符をバックスラッシュ (\) でエスケープします。次の例を参照してください。

```
aws rekognition start-person-tracking --video "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-bucket\",\"Name\":\"video-name\"}}"
--notification-channel "{\"SNSTopicArn\":\"topic-ARN\",\"RoleArn\":\"role-ARN\"}" \
--region region-name --profile profile-name
```

上記のコード例を実行した後、返された jobID をコピーして以下の GetPersonTracking コマンドに渡すと、job-id-number が以前に受け取った jobID に置き換わっている結果が得られます。

```
aws rekognition get-person-tracking --job-id job-id-number
```

> **Note**
>
> Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK) 以外のビデオ例をすでに実行している場合、置き換えるコードは異なる可能性があります。

3. コードを実行します。追跡された人物の一意の ID が、人物の動線が追跡された時間 (秒単位) とともに表示されます。

GetPersonTracking オペレーションレスポンス

GetPersonTracking は、ビデオ内で検出された人の詳細とその動線が追跡されたタイミングに関する情報を含む PersonDetection オブジェクトの配列 Persons を返します。

Persons 入力パラメータを使用して SortBy をソートすることができます。ビデオ内で人物の動線が追跡された時間によって要素をソートするには、TIMESTAMP を指定します。ビデオ内で追跡され

た人物によってソートするには、INDEX を指定します。人物の各結果セット内で、要素は動線の追跡の正確性の信頼度によって降順にソートされます。デフォルトでは、Persons は TIMESTAMP でソートされて返されます。GetPersonDetection の JSON レスポンスの例を次に示します。結果は、ビデオの開始以降にビデオ内で人物の動線が追跡された時間 (ミリ秒単位) によってソートされます。レスポンスで、以下の点に注意してください。

- 個人情報 – PersonDetection 配列要素には、検出された人物に関する情報が含まれています。たとえば、人物が検出された時間 (Timestamp)、検出時のビデオフレームでの人物の位置 (BoundingBox)、人物の正しい検出に関する Amazon Rekognition Video の信頼度 (Confidence) などです。

  人物の動線が追跡されたタイムスタンプごとに顔の特徴は返されません。さらに、状況によっては、追跡された人物の体が見えない場合があり、その場合は顔の位置のみが返されます。
- ページング情報 – 例は 1 ページの人物検出情報を示しています。人物要素を返す数は、GetPersonTracking の MaxResults 入力パラメータで指定できます。結果が MaxResults を超える場合、GetPersonTracking は次のページの結果を取得するためのトークン (NextToken) を返します。詳細については、「Amazon Rekognition Video の分析結果を取得する」を参照してください。
- インデックス –ビデオ全体で人物を識別するための一意の識別子。
- ビデオの情報 – レスポンスには、VideoMetadata から返される情報のページごとにビデオ形式に関する情報 (GetPersonDetection) が含まれます。

```
{
    "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
    }
}
```

個人用保護具を検出する。

Amazon Rekognition は、画像内の人物が身につけている個人用保護具(PPE)を検出することができます。この情報を使用して、職場の安全対策を改善することができます。例えば、建設現場で作業員がヘッドカバーを装着しているかどうか、医療従事者がフェイスカバーとハンドカバーを装着しているかどうかを判断するために、PPE 検出を使用することができます。次のイメージは、検出可能な PPE の種類を示したものです。



イメージ内の PPE を検出するには、DetectProtectiveEquipment API を呼び出して、入力イメージを渡します。レスポンスは JSON 構造で、以下が含まれています。

- 画像内で検出された人物です。
- PPE を装着している身体の部位 (顔、頭、左手、右手)
- 身体部位で検出された PPE の種類 (フェイスカバー、ハンドカバー、ヘッドカバー) です。

- 検出された PPE のアイテムについて、PPE が対応する身体部位をカバーしているかどうかを示すインジケータです。

画像内で検出された PPE の人物およびアイテムの位置に対してバウンディングボックスが返されます。

オプションで、画像内で検出された PPE アイテムと人物の概要をリクエストできます。詳細については、「イメージ内で検出された PPE の概要です。」を参照してください。

> **Note**
>
> Amazon Rekognition PPE 検出は、顔認識や顔比較を行わないため、検出された人物を特定することはできません。

PPE の種類です。

DetectProtectiveEquipment は、以下のタイプの PPE を検出します。イメージ内の他のタイプの PPE を検出する場合は、Amazon Rekognition カスタムラベルを使用して、カスタムモデルをトレーニングすることを検討してください。詳細については、「Amazon Rekognition カスタムラベル」を参照してください。

フェイスカバー

`DetectProtectiveEquipment` は、外科、N95、布製のマスクなどの一般的な顔カバーを検出できます。

ハンドカバー

`DetectProtectiveEquipment` 手術用手袋や安全手袋などのハンドカバーを検出できます。

ヘッドカバー

`DetectProtectiveEquipment` 硬い帽子やヘルメットを検出できます。

API は、頭部、手、または顔のカバーが画像内で検出されたことを示します。API は、特定のカバーのタイプに関する情報を返しません。たとえば、ハンドカバーのタイプの「手術用手袋」などです。

PPE 検出の信頼度

Amazon Rekognition は、画像内の PPE、人、身体部位の存在について予測を行います。API は Amazon Rekognition が、予測の精度にどの程度自信を持っているかを示すスコア (50-100) を提供します。

> **Note**
>
> 個人の権利、プライバシー、サービスへのアクセスに影響を与える決定を行うために DetectProtectiveEquipment の操作を使用する予定がある場合は、アクションを起こす前にレビューと検証のために人間に結果を渡すことをお勧めします。

イメージ内で検出された PPE の概要です。

オプションで、画像内で検出された PPE アイテムと人物の概要をリクエストすることができます。必要な保護具 (フェイスカバー、ハンドカバー、またはヘッドカバー) のリストと最小信頼度 (80% など) を指定できます。応答には、必要な PPE を持つ人、必要な PPE を持たない人、および決定ができない人物の統合イメージごとの識別子 (ID) 要約が含まれます。

このサマリーでは、「何人がフェイスカバーを装着していないか」や「全員がPPEを装着しているか」といった質問に素早く答えることができます。概要内で検出された各ユーザーには、一意の ID があります。IDを使用して、PPE を着用していない人のバウンディングボックスの位置などの情報を見つけることができます。

> **Note**
>
> ID は、画像解析ごとにランダムに生成され、画像間や同じ画像の複数の解析で一貫性はありません。

フェイスカバー、ヘッドカバー、ハンドカバー、またはお好みの組み合わせをまとめることができます。必要なタイプの PPE を指定するには、「要約の要件を指定する」を参照してください。また、サマリーに含まれる検出のために満たす必要のある最小信頼水準 (50 ~ 100) を指定することができます。

DetectProtectiveEquipment の要約レスポンスの詳細については、「保護装置の検出レスポンスの概要 」を参照してください。

チュートリアル: PPE でイメージを検出する AWS Lambda 関数の作成

Amazon S3 バケットにあるイメージ内の個人用保護具 (PPE) を検出する AWS Lambda 関数を作成できます。この Java V2 チュートリアルは、「AWS ドキュメント SDK の例の GitHub リポジトリ」を参照してください。

個人用保護具検出 API を理解する

以下の情報は DetectProtectiveEquipment API について説明しています。サンプルコードについては、「画像から個人用保護具を検出します。」を参照してください。

画像を供給する

入力画像 (JPG または PNG 形式) は、画像バイトとして提供するか、Amazon S3 バケットに保存されている画像を参照することができます。

人物の顔がカメラの方を向いている画像を使用することを推奨します。

入力画像が 0 度回転しない場合は、DetectProtectiveEquipment に送信する前に 0 度回転することをお勧めします。JPG 形式の画像は、Exif (Exchangeable Image File Format) のメタデータに方向情報が含まれている場合があります。この情報を使用して、画像を回転させるコードを記述できます。詳細については、「Exif バージョン 2.32」を参照してください。PNG 形式の画像には、画像の向きに関する情報は含まれていません。

Amazon S3 バケットからイメージを渡すには、少なくとも AmazonS3ReadOnlyAccess 権限を持つユーザーを使用します。AmazonRekognitionFullAccess 権限を持つユーザーを使用して、DetectProtectiveEquipment. を呼び出します。

以下の入力例の JSON では、画像は Amazon S3 バケットで渡されています。詳細については、「イメージの操作」を参照してください。この例では、すべてのPPEタイプ (ヘッドカバー、ハンドカバー、およびフェイスカバー) の概要を最低限の検出信頼度 (MinConfidence) の 80% で要求しています。MinConfidence の値は 50 ~ 100 % の間で指定する必要があり、DetectProtectiveEquipment は検出信頼度が 50 % ~ 100 % の場合のみ予測を返します。50 % 未満の値を指定すると、50 % の値を指定した結果は同じになります。詳細については、「要約の要件を指定する」を参照してください。

```
{
```

```
    "Image": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "worker.jpg"
        }
    },
    "SummarizationAttributes": {
        "MinConfidence": 80,
        "RequiredEquipmentTypes": [
            "FACE_COVER",
            "HAND_COVER",
            "HEAD_COVER"
        ]
    }
}
```

処理する画像のコレクションが大きい場合、AWS Batch を使用してバックグラウンドで DetectProtectiveEquipment への呼び出しをバッチ処理することを検討してください。

要約の要件を指定する

オプションで SummarizationAttributes (ProtectiveEquipmentSummarizationAttributes) 入力パラメータを使用すると、イメージ内で検出された PPE のタイプの要約情報をリクエストできます。

要約する PPE の種類を指定するには、RequiredEquipmentTypes 配列フィールドを使用します。配列には、1 つ以上のものを含めます。FACE_COVER、HAND_COVER または HEAD_COVER です。

MinConfidence フィールドを使用して、検出の最小信頼度 (50-100) を指定します。要約には、MinConfidence より低い信頼度で検出された、人物、身体部位、身体部位範囲、および PPE の項目は含まれません。

DetectProtectiveEquipment のサマリーレスポンスについては、「保護装置の検出レスポンスの概要 」を参照してください。

保護装置の検出レスポンスの概要

DetectProtectiveEquipment は、入力画像から検出された人物の配列を返します。各人について、検出された身体部位、および PPE の検出されたアイテムに関する情報が返されます。ヘッドカバー、ハンドカバー、フェイスカバーを着用した作業員の次の画像の JSON は以下の通りです。



JSON では、以下の点に注意してください。

- 検出された人物 – Persons は、画像上で検出された人物 (PPE を着用していない人物も含む) の配列です。DetectProtectiveEquipment は、画像から検出された最大 15 人の PPE を検出することができます。配列内の各 ProtectiveEquipmentPerson オブジェクトには、人物 ID、人物の境界ボックス、検出された体の部位、検出された PPE のアイテムが含まれます。Confidence の ProtectiveEquipmentPerson の値は、Amazon Rekognition がバウンディングボックスに人が含まれていることを確信する割合を示します。
- 体の部位 – BodyParts は、人物から検出された体の部位 (ProtectiveEquipmentBodyPart) の配列 (PPE でカバーされていない体の部位を含む) です。各 ProtectiveEquipmentBodyPart には、検出された身体部位の名称 (Name) が含まれています。DetectProtectEquipment は、顔、頭、左手、右手の体の一部を検出することができます。Confidence の ProtectiveEquipmentBodyPart フィールドは、Amazon Rekognition が身体部分の検出精度に持つ信頼度のパーセンテージを示します。

- PPE アイテム— 配列 EquipmentDetections で ProtectiveEquipmentBodyPart オブジェクトには、検出された PPE アイテムの配列が含まれています。各 EquipmentDetection オブジェクトには以下のフィールドが含まれています。
  - Type - 検出された PPE のタイプです。
  - BoundingBox - 検出された PPE の周囲のバウンディングボックスです。
  - Confidence - Amazon Rekognition はバウンディングボックスが検出された PPE を含むと確信しています。
  - CoversBodyPart - 検出された PPE が対応する身体部位上にあるかどうかを示します。

  CoversBodyPart フィールド Value は、検出された PPE が対応する体の部位にあるかどうかを示すブーリアン値です。フィールド Confidence 予測の信頼度を示します。CoversBodyPart を使用すると、検出された PPE が画像内にあるが、実際には人物の上にない場合を除外することができます。

  > **Note**
  >
  > CoversBodyPart は、その人が保護具によって十分に保護されていること、または保護具自体が適切に装着されていることを示すものでも、示唆するものでもありません。

- 概要情報 – Summary には、SummarizationAttributes の入力パラメータで指定された概要情報が含まれます。詳細については、「要約の要件を指定する」を参照してください。

  Summary は、以下の情報を含むタイプ ProtectiveEquipmentSummary のオブジェクトです。
  - PersonsWithRequiredEquipment - 各人物が以下の条件を満たす人物の ID の配列です。
    - その人は、SummarizationAttributes 入力パラメータで指定された PPE をすべて着用しています。
    - Confidence の人物 (ProtectiveEquipmentPerson) 、身体部位 (ProtectiveEquipmentBodyPart) 、保護具 (EquipmentDetection) が、指定された最小信頼度 (MinConfidence) と等しいかそれ以上です。
    - PPE の全項目について CoversBodyPart の値が真です。
  - PersonsWithoutRequiredEquipment - 以下の条件のいずれかを満たす人物の ID の配列です。
    - Confidence の値の人物 (ProtectiveEquipmentPerson) 身体部位 (ProtectiveEquipmentBodyPart) 、身体部位カバー率 (CoversBodyPart) は、

指定された最小信頼しきい値 (MinConfidence) よりも大きいが、人物は1つ以上の指定 PPE (SummarizationAttributes) が欠けています。

- CoversBodyPart の値は、指定された PPE (SummarizationAttributes) のうち、Confidence の値が指定された最小信頼しきい値 (MinConfidence) より大きいものに対して偽となります。また、指定された PPE (SummarizationAttributes) をすべて持ち、人物 (Confidence)、身体部位 (ProtectiveEquipmentPerson)、保護具 (ProtectiveEquipmentBodyPart) の EquipmentDetection 値が最小信頼しきい値 (MinConfidence) 以上であることです。

- PersonsIndeterminate - 人物 (Confidence)、身体部位 (ProtectiveEquipmentPerson)、保護具 (ProtectiveEquipmentBodyPart) または EquipmentDetection ブーリアン値の CoversBodyPart 値が、指定された最小信頼しきい値 (MinConfidence) より低い場合に検出される人物の ID の配列です。

配列サイズを使用して、特定の要約の数を取得します。例えば、PersonsWithRequiredEquipment のサイズは、指定されたタイプの PPE を着用していると検出された人の数を示します。

人物 ID を使用して、人物のバウンディングボックスの位置など、人物についてのさらなる情報を調べることができます。人物 ID は、ProtectiveEquipmentPerson (の配列 Persons) で返される ProtectiveEquipmentPerson) オブジェクトの ID フィールドにマッピングされます。そして、対応する ProtectiveEquipmentPerson オブジェクトからバウンディングボックスやその他の情報を取得することができます。

```
{
   "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": []
```

```
    }
}
```

画像から個人用保護具を検出します。

イメージ内の人物の個人用保護具 (PPE) を検出するには、 DetectProtectiveEquipment 非ストレージ型 API オペレーションを使用します。

入力画像は、AWS SDK または AWS Command Line Interface (AWS CLI) のいずれかを使用して、入力画像を画像バイト配列 (base64 でエンコードされた画像バイト) または Amazon S3 オブジェクトとして提供できます。これらの例では、Amazon S3 バケットに保存されている画像を使用しています。詳細については、「イメージの操作」を参照してください。

画像中の人物の PPE を検出するには

1. まだ実行していない場合:

   a. AmazonRekognitionFullAccess と AmazonS3ReadOnlyAccess のアクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. S3 バケットに画像 (PPE を着用した1人以上の人物を含む) をアップロードします。

   手順については、「Amazon Simple Storage Service ユーザーガイド」の「Amazon S3 へのオブジェクトのアップロード」を参照してください。

3. 以下の例を使用して、DetectProtectiveEquipment オペレーションを呼び出します。画像内のバウンディングボックスを表示する方法については、「境界ボックスの表示」を参照してください。。

   Java

   この例では、画像内で検出された人物で検出された PPE アイテムに関する情報を表示します。

   amzn-s3-demo-bucket の値を、画像が格納されている Amazon S3 バケットの名前に変更します。photo の値は、画像ファイル名に変更します。

```
//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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
//snippet-start:[rekognition.java2.detect_ppe.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.ResponseBytes;
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.BoundingBox;
import
 software.amazon.awssdk.services.rekognition.model.DetectProtectiveEquipmentRequest;
import
 software.amazon.awssdk.services.rekognition.model.DetectProtectiveEquipmentResponse;
import software.amazon.awssdk.services.rekognition.model.EquipmentDetection;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentBodyPart;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentSummarizationAttri
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
```

```
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentPerson;
import java.io.ByteArrayInputStream;
import java.io.InputStream;
import java.util.List;
//snippet-end:[rekognition.java2.detect_ppe.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DetectPPE {

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <sourceImage> <bucketName>\n\n" +
         "Where:\n" +
         "   sourceImage - The name of the image in an Amazon S3 bucket (for
 example, people.png). \n\n" +
         "   bucketName - The name of the Amazon S3 bucket (for example, amzn-
s3-demo-bucket). \n\n";

     if (args.length != 2) {
          System.out.println(usage);
          System.exit(1);
     }

     String sourceImage = args[0];
     String bucketName = args[1];
     Region region = Region.US_WEST_2;
     S3Client s3 = S3Client.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
         .build();

     RekognitionClient rekClient = RekognitionClient.builder()
         .region(region)
```

```
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

    displayGear(s3, rekClient, sourceImage, bucketName) ;
    s3.close();
    rekClient.close();
    System.out.println("This example is done!");
}

// snippet-start:[rekognition.java2.detect_ppe.main]
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();
    }
    // snippet-end:[rekognition.java2.detect_ppe.main]
}
```

AWS CLI

この AWS CLI コマンドは PPE の概要をリクエストし、CLI オペレーションの JSON detect-protective-equipment 出力を表示します。

amzn-s3-demo-bucket は、画像が含まれている Amazon S3 バケットの名前に変更します。input.jpg を使用する画像の名前に変更します。

```
aws rekognition detect-protective-equipment \
  --image "S3Object={Bucket=amzn-s3-demo-bucket,Name=input.jpg}" \
  --summarization-attributes
 "MinConfidence=80,RequiredEquipmentTypes=['FACE_COVER','HAND_COVER','HEAD_COVER']"
```

この AWS CLI コマンドは、 CLI オペレーションの JSON detect-protective-equipment 出力を表示します。

amzn-s3-demo-bucket は、画像が含まれている Amazon S3 バケットの名前に変更します。input.jpg を使用する画像の名前に変更します。

```
aws rekognition detect-protective-equipment \
  --image "S3Object={Bucket=amzn-s3-demo-bucket,Name=input.jpg}"
```

Python

この例では、画像内で検出された人物で検出された PPE アイテムに関する情報を表示します。

amzn-s3-demo-bucket の値を、画像が格納されている Amazon S3 バケットの名前に変更します。photo の値は、画像ファイル名に変更します。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
# 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_ppe(photo, bucket):

    session = boto3.Session(profile_name='profile-name')
    client = session.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-name'
    bucket = 'amzn-s3-demo-bucket'
    person_count = detect_ppe(photo, bucket)
    print("Persons detected: " + str(person_count))

if __name__ == "__main__":
```

```
    main()
```

例：フェイスカバーの周りにバウンディングボックスを描画する

次の例は、人物で検出されたフェイスカバーの周りにバウンディングボックスを描画する方法を示しています。 AWS Lambda と Amazon DynamoDB を使用する例については、AWS 「Documentation SDK examples GitHub repository」を参照してください。

フェイスカバーを検出するには、DetectProtectiveEquipment 非ストレージ型 API オペレーションを使用します。画像は、ローカルファイルシステムから読み込まれます。入力画像は、画像バイト配列 (base64 エンコードされた画像バイト) として `DetectProtectiveEquipment` に提供されます。詳細については、「イメージの操作」を参照してください。

この例では、検出された顔面領域のバウンディングボックスが表示されます。フェイスカバーが体の一部を完全に覆っている場合、バウンディングボックスは緑色で表示されます。それ以外の場合は、赤色のバウンディングボックスが表示されます。また、検出の信頼度が指定した信頼度より低い場合は、警告としてフェイスカバーのバウンディングボックス内に黄色のバウンディングボックスが表示されます。フェイスカバーが検出されなかった場合は、人物の周囲に赤色のバウンディングボックスが描画されます。

出力される画像は以下のようなものです。



検出されたフェイスカバーのバウンディングボックスを表示するには

1. まだ実行していない場合:

   a. AmazonRekognitionFullAccess アクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. 以下の例を使用して、DetectProtectiveEquipment オペレーションを呼び出します。画像内のバウンディングボックスの表示については、「境界ボックスの表示」を参照してください。。

   Java

   関数 main の中で、以下を変更します。

- photo の値をローカル画像ファイル（PNG または JPEG）のパスとファイル名に変更します。
- confidence の値を任意の信頼度（50 ~ 100）に変更します。

```
//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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例は こちら です。

```
import java.awt.*;
import java.awt.image.BufferedImage;
```

```
import java.io.*;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.*;
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;
import
 software.amazon.awssdk.services.rekognition.model.DetectProtectiveEquipmentRequest;
import software.amazon.awssdk.services.rekognition.model.EquipmentDetection;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentBodyPart;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentPerson;
import
 software.amazon.awssdk.services.rekognition.model.ProtectiveEquipmentSummarizationAttri
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.S3Exception;
import
 software.amazon.awssdk.services.rekognition.model.DetectProtectiveEquipmentResponse;
//snippet-end:[rekognition.java2.display_mask.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class PPEBoundingBoxFrame extends JPanel {

 DetectProtectiveEquipmentResponse result;
 static BufferedImage image;
 static int scale;
 float confidence;
```

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

     final String usage = "\n" +
         "Usage: " +
         "   <sourceImage> <bucketName>\n\n" +
         "Where:\n" +
         "   sourceImage - The name of the image in an Amazon S3 bucket that
 shows a person wearing a mask (for example, masks.png). \n\n" +
         "   bucketName - The name of the Amazon S3 bucket (for example, amzn-
s3-demo-bucket). \n\n";

     if (args.length != 2) {
         System.out.println(usage);
         System.exit(1);
     }

     String sourceImage = args[0];
     String bucketName = args[1];
     Region region = Region.US_EAST_1;
     S3Client s3 = S3Client.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
         .build();

     RekognitionClient rekClient = RekognitionClient.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
         .build();

     displayGear(s3, rekClient, sourceImage, bucketName);
     s3.close();
     rekClient.close();
 }

 // snippet-start:[rekognition.java2.display_mask.main]
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);
                        }
                    }
                }
            }
```

```
            }
        }
    }
    // snippet-end:[rekognition.java2.display_mask.main]
}
```

Python

関数 main の中で、以下を変更します。

- photo の値をローカル画像ファイル (PNG または JPEG) のパスとファイル名に変更します。
- confidence の値を任意の信頼度 (50 ~ 100) に変更します。
- Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
#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()
```

CLI

次の CLI の例では、以下の引数の値を変更します。

- photo の値をローカル画像ファイル (PNG または JPEG) のパスとファイル名に変更します。
- confidence の値を任意の信頼度 (50 ~ 100) に変更します。
- Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
                                aws rekognition detect-protective-equipment
 --image "{"S3Object":{"Bucket":"amzn-s3-demo-bucket","Name":"image-name"}}" --
profile profile-name \
                                --summarization-attributes
 "{"MinConfidence":MinConfidenceNumber,"RequiredEquipmentTypes":["FACE_COVER"]}"
```

Windows デバイスで CLI にアクセスする場合は、パーサーエラーの発生に対処するため、一重引用符の代わりに二重引用符を使用し、内側の二重引用符をバックスラッシュ (\) でエスケープします。例として以下を参照してください。

```
                                aws rekognition detect-protective-equipment  --
image "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-bucket\",\"Name\":\"image-name
\"}}" \
                                --profile profile-name --summarization-
attributes "{\"MinConfidence\":MinConfidenceNumber,\"RequiredEquipmentTypes\":
[\"FACE_COVER\"]}"
```

有名人の認識

Amazon Rekognition を使うと、機械学習を活用してイメージや動画の中から著名人を数万人単位で簡単に自動認識できます。有名人認識 API によって提供されるメタデータにより、コンテンツにタグを付けて、簡単に検索できるようにするために必要な反復的なマニュアル作業が大幅に削減されます。

イメージや動画コンテンツの急速な普及により、メディア企業はメディアカタログを大規模に整理、検索、活用するのに苦労することがよくあります。ニュースチャンネルやスポーツブロードキャスターは、現在のイベントに対応し、関連するプログラミングを作成するために、イメージや動画をすばやく見つける必要があります。メタデータが不十分なため、これらのタスクが困難になりますが、Amazon Rekognition を使用すると、大量の新規コンテンツまたはアーカイブコンテンツを自動的にタグ付けして、俳優、スポーツ選手、オンラインコンテンツクリエイターなど、国際的で広く知られている有名人の包括的なセットを簡単に検索できます。

Amazon Rekognition の有名人の認識は、イメージやビデオに有名人が写っていると予想される場合にのみ使用されるように設計されています。有名人以外の顔の認識については、「コレクション内での顔の検索」を参照してください。

> **Note**
>
> 有名人の方でこの機能を利用したくない方は、AWS サポート までお問い合わせいただくか、<rekognition-celebrity-opt-out@amazon.com> までEメールを送信してください。

トピック

- 顔検索と比較した有名人の認識
- イメージ内の有名人の認識
- 保存済みビデオ内の有名人の認識
- 有名人に関する情報の取得

顔検索と比較した有名人の認識

Amazon Rekognition では有名人認識と顔認識の両方の機能が用意されています。これらの機能には、ユースケースとベストプラクティスに大きな違いがあります。

有名人の認識は、スポーツ、メディア、政治、ビジネスなどの分野で何十万人もの人気のある人々を認識することができるように事前にトレーニングされています。この機能は、特定の有名人を含む可能性が高い小規模なセットを識別するために、大量の画像やビデオを検索するのに役立つように設計されています。有名人ではないさまざまな人々の間で顔を照合するために使用されることを意図していません。有名人の照合の正確さが重要である状況では、高レベルの正確さと人間の判断力を、確実に適切に適用できるよう、人間のオペレーターも使ってこのマークされた小規模のコンテンツを調べることをお勧めします。有名人の認識は、市民の自由に悪影響を及ぼす可能性のある方法で使用されるべきではありません。

一方、顔認識は、顔認識はより一般的な機能で、有名人だけでなく、自分の顔ベクトルを使って自分の顔コレクションを作成し、同一人物であることを検証したり任意の人物を検索したりできます。顔認識は、建物への出入りの認証、公共の安全、ソーシャルメディアなどの用途に使用できます。これらのすべての場合において、一致の正確性が重要な場合は、ベストプラクティス、適切な信頼性のしきい値 (公共の安全ユースケースの 99% を含む)、人間による確認を使用することをお勧めします。

詳細については、「コレクション内での顔の検索」を参照してください。

イメージ内の有名人の認識

イメージに含まれる有名人を認識し、認識した有名人に関する追加情報を取得するときは、非ストレージ型の API オペレーション RecognizeCelebrities を使用します。たとえば、情報収集がタイムクリティカルなソーシャルメディア、ニュース、エンターテインメント業界では、RecognizeCelebrities オペレーションを使用してイメージ内の最大 64 名の有名人を識別し、有名人のウェブページ (ある場合) へのリンクを返します。どのイメージから有名人を検出したかは、Amazon Rekognition に記憶されません。この情報は、アプリケーションで保存する必要があります。

RecognizeCelebrities によって返された有名人の追加情報を保存しなかった場合に、イメージを再分析することなくその情報を取得するには、GetCelebrityInfo を使用します。GetCelebrityInfo を呼び出すには、Amazon Rekognition から有名人ごとに割り当てられる一意の識別子が必要です。この識別子は、イメージから認識された有名人ごとに RecognizeCelebrities レスポンスの一部として返されます。

大量のイメージのコレクションから有名人を認識する場合は、AWS Batch を使用して RecognizeCelebrities への呼び出しをバックグラウンドでバッチ処理することを検討します。コレクションに新しいイメージを追加すると、 AWS Lambda 関数を使用して、イメージが S3 バケットにアップロードRecognizeCelebritiesされるときに を呼び出すことで有名人を認識できます。

RecognizeCelebrities の呼び出し

() または AWS SDK を使用して、入力イメージをイメージバイト配列 (base64 でエンコードされたイメージバイト AWS Command Line Interface AWS CLI) または Amazon S3 オブジェクトとして指定できます。 AWS CLI 手順では、.jpg または .png 形式のイメージを S3 バケットにアップロードします。 AWS SDK の手順では、ローカルファイルシステムからロードされたイメージを使用します。入力イメージのレコメンデーションの詳細については、「イメージの操作」を参照してください。

この手順を実行するには、1 つ以上の有名人の顔が含まれているイメージファイルが必要です。

画像内の有名人を認識するには

1. まだ実行していない場合:

   a. AmazonRekognitionFullAccess と AmazonS3ReadOnlyAccess のアクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

   b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. 以下の例を使用して、RecognizeCelebrities オペレーションを呼び出します。

   Java

   この例では、イメージ内で検出された有名人に関する情報を表示します。

   photo の値は、1 つ以上の有名人の顔が含まれているイメージファイルのパスとファイル名に変更します。

```
//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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
//snippet-start:[rekognition.java2.recognize_celebs.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.core.SdkBytes;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
import
 software.amazon.awssdk.services.rekognition.model.RecognizeCelebritiesRequest;
import
 software.amazon.awssdk.services.rekognition.model.RecognizeCelebritiesResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.Celebrity;
//snippet-end:[rekognition.java2.recognize_celebs.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
```

```
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class RecognizeCelebrities {

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <sourceImage>\n\n" +
         "Where:\n" +
         "   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png). \n\n";

     if (args.length != 1) {
         System.out.println(usage);
         System.exit(1);
     }

     String sourceImage = args[0];
     Region region = Region.US_EAST_1;
     RekognitionClient rekClient = RekognitionClient.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
         .build();

     System.out.println("Locating celebrities in " + sourceImage);
     recognizeAllCelebrities(rekClient, sourceImage);
     rekClient.close();
 }

 // snippet-start:[rekognition.java2.recognize_celebs.main]
 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);
    }
}
// snippet-end:[rekognition.java2.recognize_celebs.main]
}
```

AWS CLI

この AWS CLI コマンドは、 CLI オペレーションの JSON recognize-celebrities 出力を表示します。

amzn-s3-demo-bucket は、イメージが含まれている Amazon S3 バケットの名前に変更します。input.jpg は、1 つ以上の有名人の顔が含まれているイメージのファイル名に変更します。

profile_name の値を自分のデベロッパープロファイル名に置き換えます。

```
aws rekognition recognize-celebrities \
  --image "S3Object={Bucket=amzn-s3-demo-bucket,Name=input.jpg}"
```

Windows デバイスで CLI にアクセスする場合は、パーサーエラーの発生に対処するため、一重引用符の代わりに二重引用符を使用し、内側の二重引用符をバックスラッシュ (\) でエスケープします。例として以下を参照してください。

```
                                    aws rekognition recognize-celebrities --
image \
                                    "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-
bucket\",\"Name\":\"image-name\"}}" --profile profile-name
```

Python

この例では、イメージ内で検出された有名人に関する情報を表示します。

photo の値は、1 つ以上の有名人の顔が含まれているイメージファイルのパスとファイル名に変更します。

Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
#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 recognize_celebrities(photo):

    session = boto3.Session(profile_name='profile-name')
    client = session.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']['Type'])
        print('Smile: ' + str(celebrity['Face']['Smile']['Value']))
```

```
        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 = 'photo-name'
    celeb_count = recognize_celebrities(photo)
    print("Celebrities detected: " + str(celeb_count))

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

Node.Js

この例では、イメージ内で検出された有名人に関する情報を表示します。

photo の値は、1 つ以上の有名人の顔が含まれているイメージファイルのパスとファイル名に変更します。amzn-s3-demo-bucket の値を、提供されたイメージファイルを含む S3 バケット名に変更します。REGION の値を、ユーザーに関連付けられているリージョンの名前に変更します。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
// 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"
const profileName = "profile-name";

// Create SNS service object.
const rekogClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});
```

```
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

この例では、イメージ内で検出された有名人に関する情報を表示します。

photo の値は、1 つ以上の有名人の顔が含まれているイメージファイル (.jpg 形式または .png 形式) のパスとファイル名に変更します。

```
//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. 表示された有名人 ID のいずれかの値を書き留めます。この ID は、「有名人に関する情報の取得」で必要になります。

RecognizeCelebrities オペレーションのリクエスト

RecognizeCelebrities への入力はイメージです。この例では、イメージをイメージのバイトとして渡します。詳細については、「イメージの操作」を参照してください。

```
{
    "Image": {
        "Bytes": "/AoSiyvFpm....."
    }
}
```

RecognizeCelebrities オペレーションのレスポンス

次の例は、RecognizeCelebrities の JSON 入力および出力です。

RecognizeCelebrities は、認識された有名人の配列と、認識されなかった顔の配列を返します。この の例では、以下の点に注意してください。

- 認識された有名人 – Celebrities は認識された有名人の配列です。配列の各 Celebrity オブジェクトは、有名人の名前および関連コンテンツを参照する URL (有名人の IMDB や Wikidata へのリンクなど) のリストを示します。アプリケーションでは、Amazon Rekognition から返される ComparedFace オブジェクトを使用して、イメージ内における有名人の顔の位置と有名人の一意の識別子を確認できます。この一意の識別子を GetCelebrityInfo API オペレーションで使用し、後で有名人の情報を取得できます。
- 認識されなかった顔 – UnrecognizedFaces は、既知の有名人と一致しなかった顔の配列です。配列の ComparedFace オブジェクトごとに、イメージ内の顔の位置を示す境界ボックス (およびその他の情報) が含まれます。


```
{
    "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
        }
    }]
}
```


保存済みビデオ内の有名人の認識

Amazon Rekognition Video による保存済みビデオ内の有名人の認識は非同期オペレーションです。保存されたビデオの有名人を認識するには、StartCelebrityRecognition を使用してビデオ分析を開始します。Amazon Rekognition Video は、ビデオ分析の完了ステータスを Amazon Simple Notification Service トピックに発行します。ビデオ分析が成功したら、GetCelebrityRecognition を呼び出して分析結果を取得します。ビデオ分析の開始と結果の取得の詳細については、「Amazon Rekognition Video オペレーションを呼び出す」を参照してください。

この手順では、Amazon SQS キューを使用してビデオ分析リクエストの完了ステータスを取得する「Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)」のコードを拡張します。この手順を実行するには、1 つ以上の有名人の顔が含まれているビデオファイルが必要です。

Amazon S3 バケットに保存されたビデオ内の有名人を検出するには (SDK)

1. 「Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)」を実行します。
2. ステップ 1 で作成したクラス VideoDetect に次のコードを追加します。

   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);

       }
```

関数 main で、次の行を置き換えます。

```
       StartLabelDetection(amzn-s3-demo-bucket, video);

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

を:

```
       StartCelebrityDetection(amzn-s3-demo-bucket, video);

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

Java V2

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例は [こちら] です。

```
//snippet-start:[rekognition.java2.recognize_video_celebrity.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartCelebrityRecognitionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.CelebrityRecognitionSortBy;
```

```
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.CelebrityRecognition;
import software.amazon.awssdk.services.rekognition.model.CelebrityDetail;
import
 software.amazon.awssdk.services.rekognition.model.StartCelebrityRecognitionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityRecognitionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityRecognitionResponse;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_celebrity.import]

/**
*  To run this code example, ensure that you perform the Prerequisites as stated
 in the Amazon Rekognition Guide:
*  https://docs.aws.amazon.com/rekognition/latest/dg/video-analyzing-with-
sqs.html
*
* Also, ensure that set up your development environment, including your
 credentials.
*
* For information, see this documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/

public class RecognizeCelebritiesVideo {

private static String startJobId ="";

public static void main(String[] args) {

   final String usage = "\n" +
       "Usage: " +
       "   <bucket> <video> <topicArn> <roleArn>\n\n" +
       "Where:\n" +
       "   bucket - The name of the bucket in which the video is located (for
 example, (for example, amzn-s3-demo-bucket). \n\n"+
       "   video - The name of video (for example, people.mp4). \n\n" +
       "   topicArn - The ARN of the Amazon Simple Notification Service (Amazon
 SNS) topic. \n\n" +
       "   roleArn - The ARN of the AWS Identity and Access Management (IAM)
 role to use. \n\n" ;
```

```
   if (args.length != 4) {
       System.out.println(usage);
       System.exit(1);
   }

   String bucket = args[0];
   String video = args[1];
   String topicArn = args[2];
   String roleArn = args[3];
   Region region = Region.US_EAST_1;
   RekognitionClient rekClient = RekognitionClient.builder()
       .region(region)
       .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
       .build();

  NotificationChannel channel = NotificationChannel.builder()
       .snsTopicArn(topicArn)
       .roleArn(roleArn)
       .build();

  StartCelebrityDetection(rekClient, channel, bucket, video);
  GetCelebrityDetectionResults(rekClient);
  System.out.println("This example is done!");
  rekClient.close();
}

// snippet-start:[rekognition.java2.recognize_video_celebrity.main]
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);
    }
}
// snippet-end:[rekognition.java2.recognize_video_celebrity.main]
}
```

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
```

関数 main で、以下の行を置き換えます。

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

を:

```
    analyzer.StartCelebrityDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetCelebrityDetectionResults()
```

Node.JS

次の Node.Js のコード例では、amzn-s3-demo-bucket の値を、ビデオが含まれている S3 バケットの名前に置き換え、videoName の値をそのビデオファイルの名前に置き換えます。また、roleArn の値を、IAM サービスロールに関連付けられた ARN に置き換える必要があります。最後に、region の値を、お使いのアカウントに関連付けられたオペレーティングリージョンの名前に置き換えます。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
//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 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";
import {fromIni} from '@aws-sdk/credential-providers';

// Set the AWS Region.
const REGION = "region-name"; //e.g. "us-east-1"
// Set the profile name
const profileName = "profile-name"
// Name the collection
// Create SNS service object.
const sqsClient = new SQSClient({ region: REGION,
  credentials: fromIni({profile: profileName,}), });
const snsClient = new SNSClient({ region: REGION,
```

```
  credentials: fromIni({profile: profileName,}), });
const rekClient = new RekognitionClient({region: REGION,
  credentials: fromIni({profile: profileName,}),
});

// 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 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()
```

CLI

次の AWS CLI コマンドを実行して、ビデオ内の有名人の検出を開始します。

```
aws rekognition start-celebrity-recognition --video "{"S3Object":
{"Bucket":"amzn-s3-demo-bucket","Name":"video-name"}}" \
--notification-channel "{"SNSTopicArn":"topic-arn","RoleArn":"role-arn"}" \
--region region-name --profile profile-name
```

以下の値を更新します。

- amzn-s3-demo-bucket と video-name を、ステップ 2 で指定した Amazon S3 バケット名とファイル名に変更します。
- region-name を、使用している AWS リージョンに変更します。
- profile-name の値を自分のデベロッパープロファイル名に置き換えます。
- topic-ARN を、Amazon Rekognition Video の設定 のステップ 3 で作成した Amazon SNS トピックの ARN に変更します。
- role-ARN を、Amazon Rekognition Video の設定 のステップ 7 で作成した IAM サービスロールの ARN に変更します。

Windows デバイスで CLI にアクセスする場合は、パーサーエラーの発生に対処するため、一重引用符の代わりに二重引用符を使用し、内側の二重引用符をバックスラッシュ (\) でエスケープします。次の例を参照してください。

```
aws rekognition start-celebrity-recognition --video "{\"S3Object\":{\"Bucket\":
\"amzn-s3-demo-bucket\",\"Name\":\"video-name\"}}" \
--notification-channel "{\"SNSTopicArn\":\"topic-arn\",\"RoleArn\":\"role-arn
\"}" \
--region region-name --profile profile-name
```

上記のコード例を実行した後、返された jobID をコピーして以下の GetCelebrityRecognition コマンドに渡すと、job-id-number が以前に受け取った jobID に置き換わっている結果が得られます。

```
aws rekognition get-celebrity-recognition --job-id job-id-number --profile
 profile-name
```

> **Note**
>
> Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK) 以外のビデオ例をすでに実行している場合、置き換えるコードは異なる可能性があります。

3. コードを実行します。ビデオ内で認識された有名人に関する情報が表示されます。

GetCelebrityRecognition オペレーションのレスポンス

以下に、JSON レスポンスの例を示します。このレスポンスには、以下が含まれています。

- 認識された有名人 – `Celebrities` はビデオで認識された有名人と認識された時間の配列です。CelebrityRecognition オブジェクトは、ビデオで有名人が認識されるたびに生成されます。各 `CelebrityRecognition` には、ビデオ内で認識された有名人に関する情報 (CelebrityDetail) とその有名人が認識された時間 (`Timestamp`) が含まれています。`Timestamp` は、ビデオの開始からのミリ秒数で示されます。
- CelebrityDetail – 認識された有名人に関する情報を示します。有名人の名前 (`Name`)、識別子 (`ID`)、有名人の既知の性別 (`KnownGender`)、および関連コンテンツを参照する URL のリスト (`Urls`) が含まれます。また、認識の精度に対する Amazon Rekognition Video の信頼度、有名人の顔の詳細 (FaceDetail) も含まれます。後で関連コンテンツを取得する必要がある場合は、getCelebrityInfo で `ID` を使用できます。
- VideoMetadata – 分析したビデオに関する情報です。

```
{
    "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
    }
}
```

有名人に関する情報の取得

以下の手順では、getCelebrityInfo API オペレーションを使用して有名人に関する情報を取得します。有名人を識別するには、前回の RecognizeCelebrities への呼び出しで返された有名人 ID を使用します。

GetCelebrityInfo の呼び出し

以下の手順では、Amazon Rekognition で認識済みの有名人の ID が必要です。「イメージ内の有名人の認識」で書き留めた有名人 ID を使用します。

有名人に関する情報を取得するには (SDK)

1. まだ実行していない場合:

    a. AmazonRekognitionFullAccess と AmazonS3ReadOnlyAccess のアクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

b. AWS CLI と AWS SDKsをインストールして設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. 以下の例を使用して、GetCelebrityInfo オペレーションを呼び出します。

Java

この例では、有名人の名前と情報を表示します。

id は、「イメージ内の有名人の認識」で表示された有名人 ID のいずれかに置き換えます。

```
//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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityInfoRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityInfoResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CelebrityInfo {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <id>

                Where:
                   id - The id value of the celebrity. You can use the
 RecognizeCelebrities example to get the ID value.\s
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String id = args[0];
```

```
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        getCelebrityInfo(rekClient, id);
        rekClient.close();
    }

    /**
     * Retrieves information about a celebrity identified in an image.
     *
     * @param rekClient the Amazon Rekognition client used to make the API call
     * @param id the unique identifier of the celebrity
     * @throws RekognitionException if there is an error retrieving the
 celebrity information
     */
    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

この AWS CLI コマンドは、 CLI オペレーションの JSON get-celebrity-info 出力を表示します。ID は、「イメージ内の有名人の認識」で表示された有名人 ID のいずれかに置き換えます。profile-name の値を自分のデベロッパープロファイル名に置き換えます。

```
aws rekognition get-celebrity-info --id celebrity-id --profile profile-name
```

Python

この例では、有名人の名前と情報を表示します。

id は、「イメージ内の有名人の認識」で表示された有名人 ID のいずれかに置き換えます。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
# 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):

    session = boto3.Session(profile_name='profile-name')
    client = session.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 = "celebrity-id"
    celebrity_info = get_celebrity_info(id)
```

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

.NET

この例では、有名人の名前と情報を表示します。

id は、「イメージ内の有名人の認識」で表示された有名人 ID のいずれかに置き換えます。

```
//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 オペレーションのリクエスト

次の例は、GetCelebrityInfo の JSON 入力および出力です。

GetCelebrityInfo への入力は、必要な有名人の ID です。

```
{
    "Id": "nnnnnnn"
}
```

GetCelebrityInfo オペレーションの応答

GetCelebrityInfo は、必要な有名人に関する情報へのリンクの配列 (Urls) を返します。

```
{
    "Name": "Celebrity Name",
    "Urls": [
        "www.imdb.com/name/nmnnnnnnn"
    ]
}
```

コンテンツのモデレーション

Amazon Rekognition を使用して、不適切、望ましくない、または不快なコンテンツを検出できます。ソーシャルメディア、放送メディア、広告、電子商取引の状況で Rekognition モデレーション API を使用して、より安全なユーザーエクスペリエンスを作成し、広告主にブランド安全を保証し、ローカルおよびグローバル規制に準拠できます。

今日、多くの企業は、第三者またはユーザーが生成したコンテンツをレビューするために人間のモデレーターに完全に依存しており、他の企業はユーザーの苦情に単に反応して、攻撃的または不適切な画像、広告、動画を削除しています。しかし、人間のモデレーターだけでは十分な品質やスピードでこれらのニーズを満たすようにスケールすることはできず、ユーザーエクスペリエンスの低下、スケール達成のための高コスト、ブランドの評判の喪失につながります。Rekognition をイメージおよびビデオのモデレーションに使用することで、人間のモデレーターは、機械学習によってすでにフラグが付けられている総ボリュームの 1 - 5% のかなり小さいコンテンツセットを確認することができます。これにより、より価値ある活動に集中し、既存のコストの一部で包括的なモデレーションカバレッジを実現できます。人間の労働力をセットアップし、ヒューマンレビュータスクを実行するには、すでに Rekognition と統合されている Amazon Augmented AI を使用できます。

カスタムモデレーション機能を使用すると、モデレーション深層学習モデルの精度を高めることができます。カスタムモデレーションでは、イメージをアップロードして注釈を付けることにより、カスタムモデレーションアダプターをトレーニングします。このトレーニング済みのアダプターを DetectModerationLabels オペレーションに提供すると、イメージに関するパフォーマンスを高めることができます。詳細については「カスタムモデレーションによる精度の向上」を参照してください。

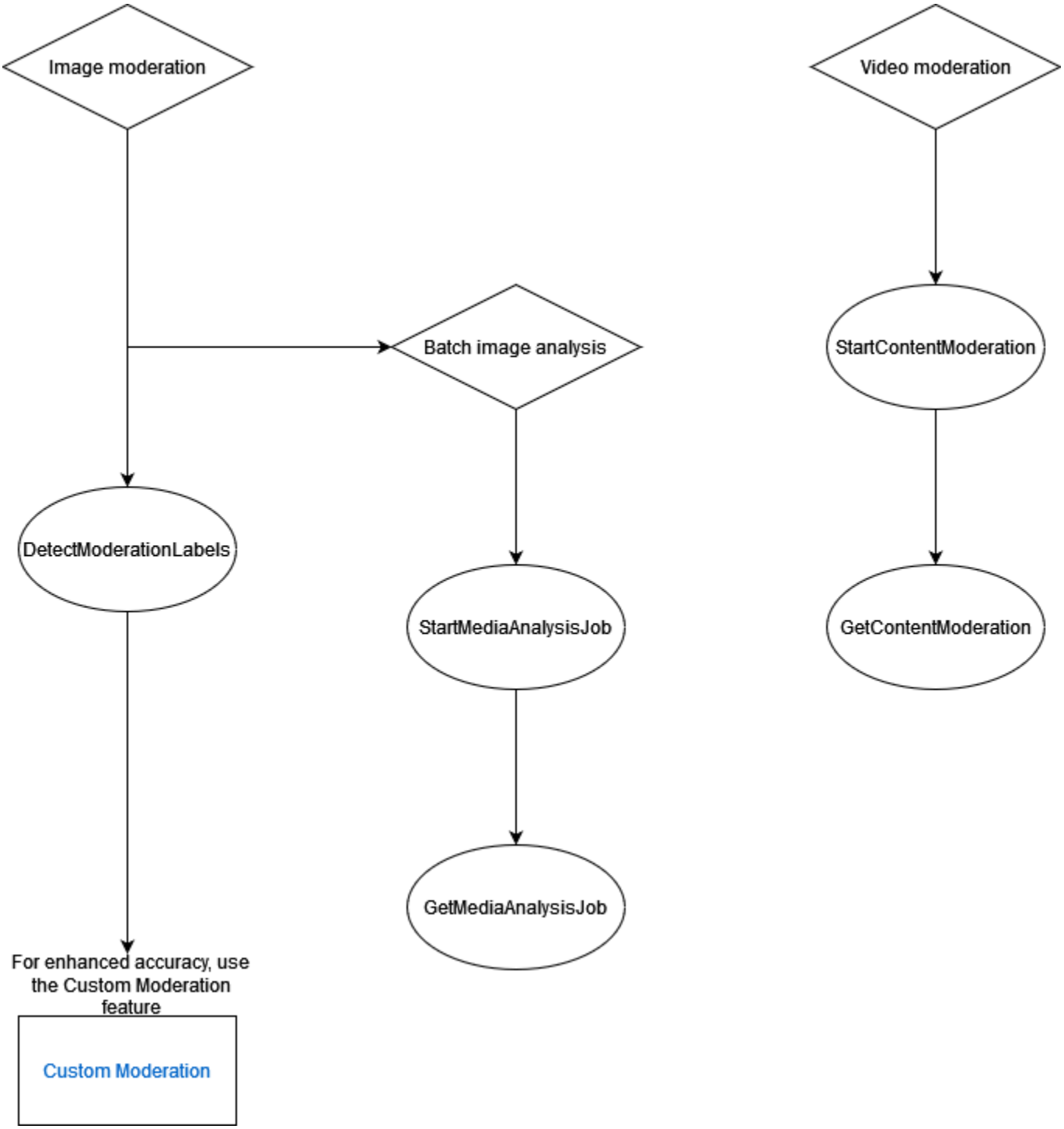
Rekognition のコンテンツのモデレーションオペレーションでサポートされているラベル

- モデレーションラベルのリストをダウンロードするには、ここをクリックします。

トピック

- イメージとビデオのモデレーション API の使用
- コンテンツモデレーションバージョン 7 のテストと API レスポンスの変換
- 不適切なイメージの検出
- 不適切な保存動画の検出
- カスタムモデレーションによる精度の向上
- Amazon Augmented AI による不適切なコンテンツの確認

以下の図は、コンテンツモデレーションの画像またはビデオコンポーネントの使用目的に応じて、必要なオペレーションを呼び出す順序を示しています。



イメージとビデオのモデレーション API の使用

Amazon Rekognition Image API では、DetectModerationLabels オペレーションを使用して同期的に、StartMediaAnalysisJob および GetMediaAnalysisJob オペレーションを使用して非同期的に、不適切、望ましくない、または不快なコンテンツを検出できます。Amazon Rekognition Video API では、StartContentModeration および GetContentModeration オペレーションを使用して、そのようなコンテンツを非同期的に検出できます。

ラベルのカテゴリ

Amazon Rekognition では、3 レベルの階層分類を使用して、不適切、望ましくない、または不快なコンテンツのカテゴリにラベルが付けられます。分類レベル 1 (L1) の各ラベルには、分類レベル 2 (L2) の複数のラベルが含まれます。分類レベル 2 の一部のラベルには、分類レベル 3 (L3) のラベルが含まれることもあります。これにより、コンテンツの階層分類が可能になります。

検出されたモデレーションラベルごとに、API はラベルが属するレベル (1、2、または 3) を含む `TaxonomyLevel` も返します。例えば、画像は以下の分類に従ってラベルが付けられる場合があります。

L1: 局部を含む控えめなヌードやキスの描写、L2: 局部を含まない控えめなヌードの描写、L3: 暗示的なヌードの描写。

> **Note**
>
> コンテンツのモデレーションには、L1 または L2 のカテゴリを使用することをお勧めします。L3 のカテゴリは、モデレーション対象としたくない特定の概念を除外する場合 (つまり、モデレーションポリシーに基づいて不適切、望ましくない、または不快なコンテンツとして分類したくないコンテンツを検出する場合) にのみ使用することをお勧めします。

以下の表では、カテゴリレベルと各レベルで使用可能なラベルの関係を示しています。モデレーションラベルのリストをダウンロードするには、ここをクリックします。

最上位カテゴリ (L1)	第 2 レベルカテゴリ (L2)	第 3 レベルカテゴリ (L3)	定義
Explicit	Explicit Nudity	Exposed Male Genitalia	Human male genitalia, including the penis

	(whether erect or flaccid), the scrotum, and any discernib le pubic hair. This term is applicable in contexts involving sexual activity or any visual content where male genitals are displayed either completely or partially.
Exposed Female Genitalia	External parts of the female reproductive system, encompass ing the vulva, vagina, and any observabl e pubic hair. This term is applicable in scenarios involving sexual activity or any visual content where these aspects of female anatomy are displayed either completely or partially.

Exposed Buttocks or Anus	Human buttocks or anus, including instances where the buttocks are nude or when they are discernible through sheer clothing. The definition specifically applies to situations where the buttocks or anus are directly and completely visible, excluding scenarios where any form of underwear or clothing provides complete or partial coverage.
Exposed Female Nipple	Human female nipples, including fully visible and partially visible aerola (area surrounding the nipples) and nipples.

	Explicit Sexual Activity	N/A	Depiction of actual or simulated sexual acts which encompasses human sexual intercourse, oral sex, as well as male genital stimulation and female genital stimulation by other body parts and objects. The term also includes ejaculation or vaginal fluids on body parts and erotic practices or roleplaying involving bondage, discipline, dominance and submission, and sadomasochism.
	Sex Toys	N/A	Objects or devices used for sexual stimulation or pleasure, e.g., dildo, vibrator, butt plug, beats, etc.
Non-Explicit Nudity of Intimate parts and Kissing	Non-Explicit Nudity	Bare Back	Human posterior part where the majority of the skin is visible from the neck to the end of the spine. This term does not apply when the individual's back is partially or fully occluded.

Exposed Male Nipple	Human male nipples, including partially visible nipples.
Partially Exposed Buttocks	Partially exposed human buttocks. This term includes a partially visible region of the buttocks or butt cheeks due to short clothes, or partially visible top portion of the anal cleft. The term does not apply to cases where the buttocks is fully nude.
Partially Exposed Female Breast	Partially exposed human female breast where one a portion of the female's breast is visible or uncovered while not revealing the entire breast. This term applies when the region of the inner breast fold is visible or when the lower breast crease is visible with nipple fully covered or occluded.

	Implied Nudity	An individual who is nude, either topless or bottomless, but with intimate parts such as buttocks, nipples, or genitalia covered, occluded, or not fully visible.
Obstructed Intimate Parts	Obstructed Female Nipple	Visual depiction of a situation in which a female's nipples is covered by opaque clothing or coverings, but their shapes are clearly visible.
	Obstructed Male Genitalia	Visual depiction of a situation in which a male's genitalia or penis is covered by opaque clothing or coverings, but its shape is clearly visible. This term applies when the obstructed genitalia in the image is in close-up.
Kissing on the Lips	N/A	Depiction of one person's lips making contact with another person's lips.

Swimwear or Underwear	Female Swimwear or Underwear	N/A	Human clothing for female swimwear (e.g., one-piece swimsuits, bikinis, tankinis, etc.) and female underwear (e.g., bras, panties, briefs, lingerie, thongs, etc.)
	Male Swimwear or Underwear	N/A	Human clothing for male swimwear (e.g., swim trunks, boardshorts, swim briefs, etc.) and male underwear (e.g., briefs, boxers, etc.)
Violence	Weapons	N/A	Instruments or devices used to cause harm or damage to living beings, structures, or systems. This includes firearms (e.g., guns, rifles, machine gunes, etc.), sharp weapons (e.g., swords, knives, etc.), explosives and ammunition (e.g., missile, bombs, bullets, etc.).

Graphic Violence	Weapon Violence	The use of weapons to cause harm, damage, injury, or death to oneself, other individuals, or properties.
	Physical Violence	The act of causing harm to other individuals or property (e.g., hitting, fighting, pulling hair, etc.) or other act of violence involving crowd or multiple individuals.
	Self-Harm	The act of causing harm to oneself, often by cutting body parts such as arms or legs, where cuts are typically visible.
	Blood & Gore	Visual represent ation of violence on a person, a group of individuals, or animals, involving open wounds, bloodshed, and mutilated body parts.

		Explosions and Blasts	Depiction of a violent and destructive burst of intense flames with thick smoke or dust and smoke erupting from the ground.
Visually Disturbing	Death and Emaciation	Emaciated Bodies	Human bodies that are extremely thin and undernourished with severe physical wasting and depletion of muscle and fat tissue.
		Corpses	Human corpses in the form of mutilated bodies, hanging corpses, or skeletons.
	Crashes	Air Crash	Incidents of air vehicles, such as airplanes, helicopte rs, or other flying vehicles, resulting in damage, injury, or death. This term applies when parts of the air vehicles are visible.

Drugs & Tobacco	Products	Pills	Small, solid, often round or oval-shaped tables or capsules. This term applies to pills presented as standalones, in a bottle, or a transparent packet and does not apply to a visual depiction of a person taking pills.
	Drugs & Tobacco Paraphernalia & Use	Smoking	The act of inhaling, exhaling, and lighting up burning substances including cigarettes, cigars, e-cigarettes, hookah, or joint.
Alcohol	Alcohol Use	Drinking	The act of drinking alcoholic beverages from bottles or glasses of alcohol or liquor.

	Alcoholic Beverages	N/A	Close up of one or multiple bottles of alcohol or liquor, glasses or mugs with alcohol or liquor, and glasses or mugs with alcohol or liquor held by an individua l. This term does not apply to an individua l drinking from bottles or glasses of alcohol or liquor.
Rude Gestures	Middle Finger	N/A	Visual depiction of a hand gesture with middle finger is extended upward while the other fingers are folded down.
Gambling	N/A	N/A	The act of participa ting in games of chance for a chance to win a prize in casinos, e.g., playing cards, blackjack s, roulette, slot machines at casinos, etc.
Hate Symbols	Nazi Party	N/A	Visual depiction of symbols, flags, or gestures associated with Nazi Party.

White Supremacy	N/A	Visual depiction of symbols or clothings associated with Ku Klux Klan (KKK) and images with confederate flags.
Extremist	N/A	Images containing extremist and terrorist group flags.

L2 カテゴリのすべてのラベルが L3 カテゴリでサポートされるわけではありません。また、「Products」と「Drug and Tobacco Paraphernalia and Use」という L2 ラベルの概念を、その下位の L3 ラベルが網羅しているわけではありません。これらの L2 ラベルは、それらの L3 ラベルが捉えきれない概念も対象としており、その場合、API レスポンスでは L2 ラベルのみが返されます。

用途に合ったコンテンツであるかどうかはお客様が判断します。例えば、暗示的な性質のイメージは受け入れ、ヌードを含むイメージは拒否することができます。イメージをフィルタリングするには、`DetectModerationLabels` (イメージ) および `GetContentModeration` (ビデオ) から返される ModerationLabel ラベル配列を使用します。

コンテンツタイプ

API は、アニメーションまたはイラストのコンテンツタイプも識別でき、レスポンスの一部でコンテンツタイプが返されます。

- アニメーションコンテンツには、ビデオゲームやアニメーション (例: カートゥーン、コミック、マンガ、アニメ) が含まれます。
- イラストコンテンツには、線画、絵画、スケッチが含まれます。

信頼度

`MinConfidence` 入力パラメータを指定することで、安全でないコンテンツを検出するために Amazon Rekognition によって使用される信頼度のしきい値を設定できます。検出された安全でないコンテンツの信頼度が `MinConfidence` よりも低い場合、そのコンテンツのラベルは返されません。

MinConfidence を 50% 未満に設定すると、偽陽性の結果が多く返される傾向があります (つまり、再現率は高くなりますが、精度は低下します)。一方、MinConfidence を 50% 以上に指定すると、偽陽性の結果が少なく返される傾向があります (つまり、再現率は低くなりますが、精度は向上します)。MinConfidence の値を指定しない場合、 Amazon Rekognition は 50% 以上の信頼度で検出された安全でないコンテンツのラベルを返します。

ModerationLabel 配列には、上のカテゴリのラベルと検出されたコンテンツの精度を示す推定信頼度が含まれます。最上位ラベルは、識別された第 2 レベルラベルと共に返されます。例えば、Amazon Rekognition では「明示的なヌード」を最上位のラベルの高い信頼スコアと共に返すことができます。場合によりますが、フィルタ処理のニーズにはこれで十分です。ただし、必要に応じて、第 2 レベルラベル ([Graphic Male Nudity (男性のヌードイメージ)] など) の信頼スコアを使用して、よりきめ細かなフィルタ処理が可能です。例については、不適切なイメージの検出を参照してください。

バージョニング

Amazon Rekognition イメージと Amazon Rekognition Video は、両方とも、不適切なコンテンツの検出に使用されるモデレーション検出のバージョンを返します (ModerationModelVersion)。

並び替えと集計

GetContentModeration を使用して結果を取得すると、結果を並べ替えて集計できます。

並べ替えの順序 – 返されるラベルの配列は時間を基準に並べ替えられます。ラベル別に並べ替えるには、GetContentModeration の SortBy 入力パラメータに NAME を指定します。ビデオ内でラベルが複数回表示されている場合、ModerationLabel 要素のインスタンスは複数になります。

ラベル情報 – ModerationLabels の配列要素には ModerationLabel オブジェクトが含まれ、このオブジェクトには、ラベル名と、検出されたラベルの精度に関する Amazon Rekognition の信頼度が含まれています。タイムスタンプは ModerationLabel が検出された時間を表し、ビデオの開始から経過した時間がミリ秒数として定義されます。ビデオ SEGMENTS ごとに集計された結果では、StartTimestampMillis、EndTimestampMillis、DurationMillis 構造が返され、それぞれセグメントの開始時間、終了時間、持続時間を定義します。

集計 — 結果が返されたときの集計方法を指定します。デフォルトでは TIMESTAMPS によって集計されます。また、SEGMENTS による集計を選択することもできます。この方法では、時間枠の結果が集計されます。このセグメント中に検出されたラベルのみが返されます。

カスタムモデレーションアダプターのステータス

カスタムモデレーションアダプター
は、TRAINING_IN_PROGRESS、TRAINING_COMPLETED、TRAINING_FAILED、DELETING、DEPRECA
のいずれかのステータスになります。アダプターのこれらのステータスの詳細については、「アダプターの管理」を参照してください。

> **Note**
>
> Amazon Rekognitionは、不適切または不快なコンテンツに関する権威ではなく、また網羅的なフィルタリングであると主張するものではありません。また、イメージおよびビデオのモデレーション API は、CSAM などの違法なコンテンツが含まれているかどうかは検出しません。

コンテンツモデレーションバージョン 7 のテストと API レスポンスの変換

Rekognition では、コンテンツモデレーションのラベル検出機能の画像およびビデオコンポーネントに使用する機械学習モデルがバージョン 6.1 から 7 に更新されました。この更新では、全体的な精度が向上し、一部のカテゴリが修正されたほか、いくつかの新しいカテゴリが導入されました。

現在、バージョン 6.1 のビデオユーザーである場合は、バージョン 7 にシームレスに移行するために、以下の対応をお勧めします。

1. AWS プライベート SDK (「the section called “AWS コンテンツモデレーションバージョン 7 の SDK と使用ガイド ”」を参照) をダウンロードし使用して、StartContentModeration API を呼び出します。
2. API レスポンスまたはコンソールで返されるラベルと信頼スコアの更新されたリストを確認します。必要に応じて、アプリケーションの後処理ロジックを調整します。
3. お客様のアカウントは 2024 年 5 月 13 日までバージョン 6.1 のままとなります。バージョン 6.1 を 2024 年 5 月 13 日以降も使用する場合は、2024 年 4 月 30 日までに AWS サポートチームに連絡して、使用期限の延長を申請してください。バージョン 6.1 は 2024 年 6 月 10 日までご利用いただけます。2024 年 4 月 30 日までにご連絡がない場合、お客様のアカウントは 2024 年 5 月 13 日より自動的にバージョン 7.0 に移行されます。

AWS コンテンツモデレーションバージョン 7 の SDK と使用ガイド

選択した開発言語に対応する SDK をダウンロードし、適切なユーザーガイドを参照してください。

SDK へのリンク	インストール/ユーザーガイド
Java-1.X	ガイド - Java 1.pdf
Java-2.X	ガイド - Java 2.pdf
JavaScript v2	ガイド - JavaScript v2.pdf
JavaScript v3	ガイド - JavaScript v3.pdf
Python	ガイド - Python & AWS CLI.pdf
Ruby	ガイド - RubyV3.pdf
go_v1	ガイド - GO V1.pdf
go_v2	ガイド - GO V2.pdf
DotNet	ガイド - .NET.pdf
php	ガイド - PHP.pdf

バージョン 6.1 から 7 へのラベルのマッピング

コンテンツモデレーションバージョン 7 では、新しいラベルカテゴリが追加され、既存のラベル名が変更されました。6.1 ラベルを 7 ラベルにどのようにマッピングするかを決定する際には、the section called “ ラベルのカテゴリ ” にある分類表を参照してください。

ラベルのマッピング例を次のセクションにいくつか示します。アプリケーションの後処理ロジックに基づいて必要な更新を行う前に、これらのマッピングとラベルの定義を確認することをお勧めします。

L1 マッピングスキーマ

最上位カテゴリ (L1) (`Explicit Nudity`、`Suggestive`、`Violence` など) のみでフィルタリングする後処理ロジックを使用している場合は、以下の表を参照してコードを更新してください。

V6.1 L1	V7 L1
Explicit Nudity	Explicit
Suggestive	Non-Explicit Nudity of Intimate parts and Kissing
	Swimwear or Underwear
Violence	Violence
Visually Disturbing	Visually Disturbing
Rude Gestures	Rude Gestures
Drugs	Drugs & Tobacco
Tobacco	Drugs & Tobacco
Alcohol	Alcohol
Gambling	Gambling
Hate Symbols	Hate Symbols

L2 マッピングスキーマ

L1 と L2 の両カテゴリ (`Explicit Nudity / Nudity, Suggestive / Female Swimwear Or Underwear`、`Violence / Weapon Violence` など) でフィルタリングする後処理ロジックを使用している場合は、以下の表を参照してコードを更新してください。

V6.1 L1	V6.1 L2	V7 L1	V7 L2	V7 L3	V7 ContentTypes
Explicit Nudity	Nudity	Explicit	Explicit Nudity	露出した女性の乳首	
				露出した臀部または肛門	

	Graphic Male Nudity	Explicit	Explicit Nudity	Exposed Male Genitalia	
	Graphic Female Nudity	Explicit	Explicit Nudity	Exposed Female Genitalia	
	Sexual Activity	Explicit	Explicit Sexual Activity		
	Illustrated Explicit Nudity	Explicit	Explicit Nudity		Map to "Animated" and "Illustrated"
	Illustrated Explicit Nudity	Explicit	Explicit Sexual Activity		Map to "Animated" and "Illustrated"
	Adult Toys	Explicit	Sex Toys		
Suggestive	Female Swimwear Or Underwear	Swimwear or Underwear	Female Swimwear or Underwear		
	Male Swimwear Or Underwear	Swimwear or Underwear	Male Swimwear or Underwear		
	Partial Nudity	Non-Explicit Nudity of Intimate parts and Kissing	Non-Explicit Nudity	Implied Nudity	

	Barechested Male	Non-Explicit Nudity of Intimate parts and Kissing	Non-Explicit Nudity	Exposed Male Nipple
	Revealing Clothes	Non-Explicit Nudity of Intimate parts and Kissing	Non-Explicit Nudity	
		Non-Explicit Nudity of Intimate parts and Kissing	Obstructed Intimate Parts	
	Sexual Situations	Non-Explicit Nudity of Intimate parts and Kissing	Kissing on the Lips	
Violence	Graphic Violence Or Gore	Violence	Graphic Violence	Blood & Gore
	Physical Violence	Violence	Graphic Violence	Physical Violence
	Weapon Violence	Violence	Graphic Violence	Weapon Violence
	Weapons	Violence	Weapons	
	Self Injury	Violence	Graphic Violence	Self-Harm
Visually Disturbing	Emaciated Bodies	Visually Disturbing	Death and Emaciation	Emaciated Bodies

	Corpses	Visually Disturbing	Death and Emaciation	Corpses
	Hanging	Visually Disturbing	Death and Emaciation	Corpses
	Air Crash	Visually Disturbing	Crashes	Air Crash
	Explosions And Blasts	Violence	Graphic Violence	Explosions and Blasts
Rude Gestures	Middle Finger	Rude Gestures	Middle Finger	
Drugs	Drug Products	Drugs & Tobacco	Products	
	Drug Use	Drugs & Tobacco	Drugs & Tobacco Paraphernalia & Use	
	Pills	Drugs & Tobacco	Products	Pills
	Drug Paraphernalia	Drugs & Tobacco	Drugs & Tobacco Paraphernalia & Use	
Tobacco	Tobacco Products	Drugs & Tobacco	Products	
	Smoking	Drugs & Tobacco	Drugs & Tobacco Paraphernalia & Use	Smoking

Alcohol	Drinking	Alcohol	Alcohol Use	Drinking
	Alcoholic Beverages	Alcohol	Alcoholic Beverages	
Gambling	Gambling	Gambling		
Hate Symbols	Nazi Party	Hate Symbols	Nazi Party	
	White Supremacy	Hate Symbols	White Supremacy	
	Extremist	Hate Symbols	Extremist	

不適切なイメージの検出

DetectModerationLabels オペレーションを使用すると、イメージに不適切なコンテンツや不快なコンテンツが含まれているかどうかを判断できます。Amazon Rekognition のモデレーションラベルのリストについては、イメージおよびビデオのモデレーション API の使用 を参照してください。

イメージ内の不適切なコンテンツの検出

イメージは、.jpg または .png 形式である必要があります。入力イメージとして、イメージのバイト配列 (base64 でエンコードされたイメージのバイト) を指定するか、Amazon S3 オブジェクトを指定することができます。以下の手順では、イメージ (.jpg または .png) を S3 バケットにアップロードします。

これらの手順を実行するには、 AWS CLI または適切な AWS SDK がインストールされている必要があります。詳細については、「Amazon Rekognition の開始方法」を参照してください。使用する AWS アカウントには、Amazon Rekognition API へのアクセス権限が必要です。詳細については、[Amazon Rekognition で定義されるアクション]を参照してください。

イメージでモデレーションラベルを検出するには (SDK)

1. まだ実行していない場合:

   a. `AmazonRekognitionFullAccess` と `AmazonS3ReadOnlyAccess` のアクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. イメージを S3 バケットにアップロードします。

   手順については、「Amazon Simple Storage Service ユーザーガイド」の「Amazon S3 へのオブジェクトのアップロード」を参照してください。

3. 以下の例を使用して、DetectModerationLabels オペレーションを呼び出します。

   Java

   この例では、検出された安全でないコンテンツのラベル名、信頼度、検出されたモデレーションラベルの親ラベルを出力します。

   amzn-s3-demo-bucket と photo の値は、ステップ 2 で使用した S3 バケット名とイメージファイル名に置き換えます。

```
//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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
//snippet-start:[rekognition.java2.recognize_video_text.import]
//snippet-start:[rekognition.java2.detect_mod_labels.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
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.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.Image;
import
 software.amazon.awssdk.services.rekognition.model.DetectModerationLabelsRequest;
```

```
import
 software.amazon.awssdk.services.rekognition.model.DetectModerationLabelsResponse;
import software.amazon.awssdk.services.rekognition.model.ModerationLabel;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
//snippet-end:[rekognition.java2.detect_mod_labels.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class ModerateLabels {

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <sourceImage>\n\n" +
         "Where:\n" +
         "   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png). \n\n";

     if (args.length < 1) {
         System.out.println(usage);
         System.exit(1);
     }

     String sourceImage = args[0];
     Region region = Region.US_WEST_2;
     RekognitionClient rekClient = RekognitionClient.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
         .build();

     detectModLabels(rekClient, sourceImage);
     rekClient.close();
 }
```

```
// snippet-start:[rekognition.java2.detect_mod_labels.main]
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);
    }
}
// snippet-end:[rekognition.java2.detect_mod_labels.main]
```

AWS CLI

この AWS CLI コマンドは、 CLI オペレーションの JSON `detect-moderation-labels` 出力を表示します。

amzn-s3-demo-bucket と input.jpg は、ステップ 2 で使用した S3 バケット名とイメージファイル名に置き換えます。profile_name の値を自分のデベロッパープロファイル名に置き換えます。アダプターを使用するには、project-version パラメータにプロジェクトバージョンの ARN を指定します。

```
aws rekognition detect-moderation-labels --image "{S3Object:{Bucket:<amzn-s3-demo-bucket>,Name:<image-name>}}" \
--profile profile-name \
--project-version "ARN"
```

Windows デバイスで CLI にアクセスする場合は、パーサーエラーの発生に対処するため、一重引用符の代わりに二重引用符を使用し、内側の二重引用符をバックスラッシュ (\) でエスケープします。例として以下を参照してください。

```
aws rekognition detect-moderation-labels --image "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-bucket\",\"Name\":\"image-name\"}}" \
--profile profile-name
```

Python

この例では、検出された安全でないコンテンツのラベル名、信頼度、検出されたモデレーションラベルの親ラベルを出力します。

関数 main で、amzn-s3-demo-bucket と photo の値は、ステップ 2 で使用した S3 バケット名とイメージファイル名に置き換えます。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
#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):

    session = boto3.Session(profile_name='profile-name')
    client = session.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='image-name'
    bucket='amzn-s3-demo-bucket'
    label_count=moderate_image(photo, bucket)
    print("Labels detected: " + str(label_count))

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

.NET

この例では、検出された安全でないコンテンツのラベル名、信頼度、検出されたモデレーションラベルの親ラベルを出力します。

amzn-s3-demo-bucket と photo の値は、ステップ 2 で使用した S3 バケット名とイメージファイル名に置き換えます。

```
//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 = "amzn-s3-demo-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 オペレーションのリクエスト

DetectModerationLabels への入力はイメージです。以下の JSON 入力の例では、ソースイメージを Amazon S3 バケットからロードします。Amazon Rekognition は、必要最小限のMinConfidence の信頼度であり、これ以上の精度の検出済みラベルがレスポンスで返されます。

```
{
    "Image": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
```

```
            "Name": "input.jpg"
        }
    },
    "MinConfidence": 60
}
```

DetectModerationLabels オペレーションのレスポンス

DetectModerationLabels は、S3 バケットから入力イメージを取得できます。または、これらをイメージのバイトとしてユーザーが提供できます。DetectModerationLabels への呼び出しのレスポンス例を以下に示します。

ここで示す JSON レスポンスの例では、以下の点に留意してください。

- 不適切なイメージ検出情報 — この例では、イメージで検出された不適切または不快なコンテンツのラベルが一覧表示されます。この一覧には、イメージで検出された最上位ラベルと第 2 レベルの各ラベルが表示されます。

  ラベル – ラベルごとに、名前、ラベルの精度を示す　Amazon Rekognition の推定信頼度、および親ラベルの名前があります。最上位ラベルの親の名前は "" です。

  ラベルの信頼度 – 各ラベルには0から100の信頼度値があり、 Amazon Rekognition がそのラベルが正しいという確信度を示しています。レスポンスで返させるラベルの必要な信頼度は、API オペレーションリクエストで指定できます。

```
{
    "ModerationLabels": [
        {
            "Confidence": 99.44782257080078,
            "Name": "Smoking",
            "ParentName": "Drugs & Tobacco Paraphernalia & Use",
            "TaxonomyLevel": 3
        },
        {
            "Confidence": 99.44782257080078,
            "Name": "Drugs & Tobacco Paraphernalia & Use",
            "ParentName": "Drugs & Tobacco",
            "TaxonomyLevel": 2
        },
        {
            "Confidence": 99.44782257080078,
```

```
            "Name": "Drugs & Tobacco",
            "ParentName": "",
            "TaxonomyLevel": 1
        }
    ],
    "ModerationModelVersion": "7.0",
    "ContentTypes": [
        {
            "Confidence": 99.9999008178711,
            "Name": "Illustrated"
        }
    ]
}
```

不適切な保存動画の検出

保存済みビデオ内の Amazon Rekognition Video のテキスト検出は、非同期オペレーションです。不適切なコンテンツや不快なコンテンツを検出するには、StartContentModeration を呼び出します。Amazon Rekognition Video は、ビデオ分析の完了ステータスを Amazon Simple Notification Service トピックに発行します。ビデオ分析が完了したら、GetContentModeration を呼び出して分析結果を取得します。ビデオ分析の開始と結果の取得の詳細については、「Amazon Rekognition Video オペレーションを呼び出す」を参照してください。Amazon Rekognition のモデレーションラベルのリストについては、イメージおよびビデオのモデレーション API の使用 を参照してください。

この手順では、Amazon Simple Queue Service キューを使用してビデオ分析リクエストの完了ステータスを取得する「Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)」のコードを拡張します。

Amazon S3 バケット(SDK) に保存されたビデオ内の不適切または不快なコンテンツを検出するには

1. 「Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)」を実行します。
2. ステップ 1 で作成したクラス VideoDetect に次のコードを追加します。

   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);

        }
```

関数 main で、以下の行を置き換えます。

```
       StartLabelDetection(amzn-s3-demo-bucket, video);

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

を:

```
        StartUnsafeContentDetection(amzn-s3-demo-bucket, video);

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

Java V2

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例は [こちら] です。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartContentModerationRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartContentModerationResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetContentModerationResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetContentModerationRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import
 software.amazon.awssdk.services.rekognition.model.ContentModerationDetection;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetectInappropriate {
    private static String startJobId = "";

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

```
        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
(for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
(IAM) role to use.\s
                """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startModerationDetection(rekClient, channel, bucket, video);
        getModResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    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,
                                                SortBy="NAME",
                                                AggregateBy="TIMESTAMPS")

            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
```

関数 `main` で、以下の行を置き換えます。

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

を:

```
    analyzer.StartUnsafeContent()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetUnsafeContentResults()
```

> **Note**
>
> Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK) 以外のビデオ例をすでに実行している場合、置き換えるコードは異なる可能性があります。

3. コードを実行します。ビデオで検出された、安全でないコンテンツのラベルのリストが表示されます。

GetContentModeration オペレーションレスポンス

GetContentModeration からのレスポンスは、ContentModerationDetection オブジェクトの配列、ModerationLabels です。配列には、安全でないコンテンツラベルが検出されるたびに要素が追加されます。ContentModerationDetectionObject オブジェクト内では、ModerationLabel には、検出された不適切なまたは不快なコンテンツ項目の情報が含まれます。Timestamp は、ビデオが開始されてからラベルが検出されるまでに経過した時間 (ミリ秒数) です。ラベルは、安全でないコンテンツイメージの分析で検出されたラベルと同じ方法で階層的に編成されています。詳細については、「コンテンツのモデレーション」を参照してください。

以下は GetContentModeration からのレスポンスの例です。NAME 別に並べ替えられ、TIMESTAMPS によって集計されています。

```
{
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 54100,
        "Format": "QuickTime / MOV",
        "FrameRate": 30.0,
        "FrameHeight": 462,
        "FrameWidth": 884,
        "ColorRange": "LIMITED"
    },
    "ModerationLabels": [
        {
            "Timestamp": 36000,
            "ModerationLabel": {
                "Confidence": 52.451576232910156,
                "Name": "Alcohol",
                "ParentName": "",
                "TaxonomyLevel": 1
            },
            "ContentTypes": [
                {
                    "Confidence": 99.9999008178711,
                    "Name": "Animated"
                }
            ]
        },
        {
            "Timestamp": 36000,
```

```
            "ModerationLabel": {
                "Confidence": 52.451576232910156,
                "Name": "Alcoholic Beverages",
                "ParentName": "Alcohol",
                "TaxonomyLevel": 2
            },
            "ContentTypes": [
                {
                    "Confidence": 99.9999008178711,
                    "Name": "Animated"
                }
            ]
        }
    ],
    "ModerationModelVersion": "7.0",
    "JobId": "a1b2c3d4...",
    "Video": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "video-name.mp4"
        }
    },
    "GetRequestMetadata": {
        "SortBy": "TIMESTAMP",
        "AggregateBy": "TIMESTAMPS"
    }
}
```

以下は GetContentModeration からのレスポンスの例です。NAME 別に並べ替えられ、SEGMENTS によって集計されています。

```
{
    "JobStatus": "SUCCEEDED",
    "VideoMetadata": {
        "Codec": "h264",
        "DurationMillis": 54100,
        "Format": "QuickTime / MOV",
        "FrameRate": 30.0,
        "FrameHeight": 462,
        "FrameWidth": 884,
        "ColorRange": "LIMITED"
    },
    "ModerationLabels": [
```

```
        {
            "Timestamp": 0,
            "ModerationLabel": {
                "Confidence": 0.0003000000142492354,
                "Name": "Alcohol Use",
                "ParentName": "Alcohol",
                "TaxonomyLevel": 2
            },
            "StartTimestampMillis": 0,
            "EndTimestampMillis": 29520,
            "DurationMillis": 29520,
            "ContentTypes": [
                {
                    "Confidence": 99.9999008178711,
                    "Name": "Illustrated"
                },
                {
                    "Confidence": 99.9999008178711,
                    "Name": "Animated"
                }
            ]
        }
    ],
    "ModerationModelVersion": "7.0",
    "JobId": "a1b2c3d4...",
    "Video": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
            "Name": "video-name.mp4"
        }
    },
    "GetRequestMetadata": {
        "SortBy": "TIMESTAMP",
        "AggregateBy": "SEGMENTS"
    }
}
```

カスタムモデレーションによる精度の向上

Amazon Rekognition の DetectModerationLabels API を使用すると、不適切なコンテンツや不快な迷惑コンテンツを検出できます。Rekognition のカスタムモデレーション機能では、アダプターを使用することにより DetectModerationLabels の精度を高めることができます。アダプターは、既存の

Rekognition 深層学習モデルに追加できるモジュール式のコンポーネントであり、トレーニング対象のタスクに合わせて機能を拡張できます。アダプターを作成して DetectModerationLabels オペレーションでこれを指定することにより、特定のユースケースに関連するコンテンツモデレーションタスクの精度をさらに高めることができます。

Rekognition のコンテンツモデレーションモデルを特定のモデレーションラベル用にカスタマイズするときは、プロジェクトを作成し、提供した一連のイメージに基づいて、アダプターをトレーニングします。そうすることで、アダプターのパフォーマンスを繰り返し確認し、希望する精度になるまでアダプターをトレーニングできます。プロジェクトは、異なるバージョンのアダプターを含めるために使用されます。

プロジェクトとアダプターを作成するときは Rekognition コンソールを使用します。または、 AWS SDK および関連する APIs を使用して、プロジェクトの作成、アダプターのトレーニング、アダプターの管理を行うことができます。

アダプターの作成と使用

アダプターは、既存の Rekognition 深層学習モデルに追加できるモジュール式のコンポーネントであり、トレーニング対象のタスクに合わせて機能を拡張できます。アダプターを使って深層学習モデルをトレーニングすることにより、特定のユースケースに関連するイメージ分析タスクの精度を高めることができます。

アダプターを作成して使用するには、Rekognition にトレーニングデータとテストデータを提供する必要があります。そのためには、次のうちいずれかの方法を実行します。

- 一括分析と検証 - Rekognition が分析とラベルの割り当てを行うイメージを一括分析することで、トレーニングデータセットを作成できます。その後、イメージ用に生成された注釈をレビューし、予測を検証または修正します。イメージを一括分析する方法の詳細については、「Bulk analysis」を参照してください。
- 手動での注釈 - イメージをアップロードし注釈を付けることによりトレーニングデータを作成するときはこの方法を使います。テストデータは、イメージをアップロードして注釈を付けるか、自動分割することにより作成します。

詳細については、以下のトピックをご覧ください。

トピック

- 一括分析と検証
- 手動での注釈

一括分析と検証

この方法では、トレーニングデータとして使用する大量のイメージをアップロードし、Rekognition を使ってそれらイメージの予測を取得します。それにより、イメージに自動的にラベルが割り当てられます。これらの予測は、アダプターの出発点として使用できます。予測の精度は検証が可能で、アダプターは検証済みの予測に基づいてトレーニングします。これは AWS コンソールで実行できます。

一括分析とカスタムモデレーション

一括分析用のイメージのアップロード

アダプター用のトレーニングデータセットを作成するには、Rekognition がラベルを予測できるよう、イメージを一括でアップロードします。最善の結果を得るには、10,000 を上限として、できる限り多くのトレーニング用イメージを用意し、それらのイメージで、ユースケースのありとあらゆる側面を表現できるようにします。

AWS コンソールを使用する場合、コンピュータから直接イメージをアップロードしたり、イメージを保存する Amazon Simple Storage Service バケットを提供したりできます。ただし、Rekognition API と SDK を使用する場合は、Amazon Simple Storage Service バケットに保存されているイメージを参照する、マニフェストファイルを提供する必要があります。詳細については「Bulk analysis」を参照してください。

予測のレビュー

イメージを Rekognition コンソールにアップロードすると、Rekognition でそのイメージ用のラベルが生成されます。それにより、これらの予測を真陽性、偽陽性、真陰性、偽陰性のいずれかのカテゴリとして検証できます。予測を検証した後、フィードバックに基づいてアダプターをトレーニングできます。

アダプターのトレーニング

一括分析で返された予測の検証が完了すると、アダプターのトレーニングプロセスを開始できます。

AdapterId の取得

アダプターのトレーニングが完了すると、アダプターの一意の ID を取得して、Rekognition のイメージ分析 API と共に使用できるようになります。

API オペレーションの呼び出し

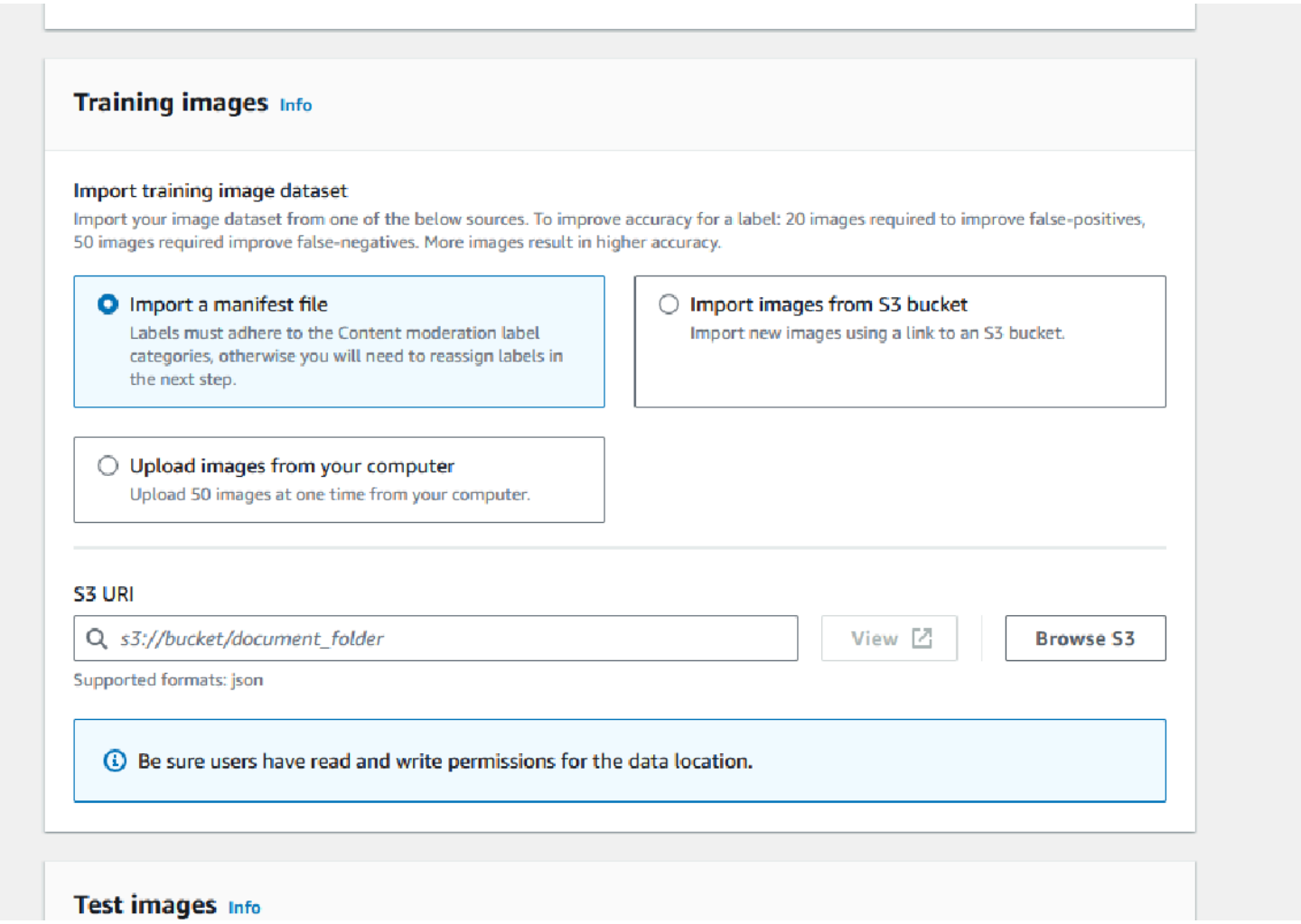
カスタムのアダプターを適用するには、アダプターをサポートしているイメージ分析 API を呼び出す際にその ID を指定します。これにより、イメージの予測精度が高まります。

手動での注釈

イメージを手動でアップロードし注釈を付けることによりトレーニングデータを作成するときは、この方法を使います。テストデータは、テストイメージをアップロードしこれに注釈をつけるか、自動分割により、Rekognition でトレーニングデータの一部をテストイメージとして使用することによって作成します。

イメージのアップロードと注釈付け

アダプターをトレーニングするには、典型的なユースケースの、サンプルイメージのセットをアップロードする必要があります。最善の結果を得るには、10,000 を上限として、できる限り多くのトレーニング用イメージを用意し、それらのイメージで、ユースケースのありとあらゆる側面を表現できるようにします。



AWS コンソールを使用する場合、コンピュータから直接イメージをアップロードしたり、マニフェストファイルを提供したり、イメージを保存する Amazon S3 バケットを提供したりできます。

ただし、Rekognition API と SDK を使用する場合は、Amazon S3 バケットに保存されているイメージを参照する、マニフェストファイルを提供する必要があります。

イメージの注釈は Rekognition コンソールの注釈インターフェイスを使って付けることができます。ラベルを使ってタグ付けすることでイメージに注釈を付けると、トレーニングの「グラウンドトゥルース」が設定されます。また、アダプターをトレーニングする前にトレーニングセットとテストセットを指定するか、自動分割機能を使用する必要があります。データセットの指定とイメージの注釈付けが完了したら、テストセット内の注釈付きイメージに基づいてアダプターを作成できるようになります。その後、アダプターのパフォーマンスを評価できます。

テストセットの作成

注釈付きのテストセットを提供するか、自動分割機能を使用します。トレーニングセットは、アダプターを実際にトレーニングするのに使用します。アダプターは、これらの注釈付きイメージに含まれるパターンを学習します。テストセットは、アダプターを完成させる前にモデルのパフォーマンスを評価するために使用します。

アダプターのトレーニング

トレーニングデータの注釈が完了すると、または、マニフェストファイルを提供すると、アダプターのトレーニングプロセスを開始できます。

アダプター ID の取得

アダプターのトレーニングが完了すると、アダプターの一意の ID を取得して Rekognition のイメージ分析 API と共に使用できるようになります。

API オペレーションの呼び出し

カスタムのアダプターを適用するには、アダプターをサポートしているイメージ分析 API を呼び出す際にその ID を指定します。これにより、イメージの予測精度が高まります。

データの準備

アダプターを作成するには、Rekognition に 2 つのデータセット (トレーニングデータセットとテストデータセット) を提供する必要があります。各データセットは、イメージと注釈/ラベルという 2 つの要素で構成されています。以下のセクションでは、ラベルとイメージの用途、およびこれらを組み合わせてデータセットを作成する方法について説明します。

イメージ

イメージの典型的な例を使ってアダプターをトレーニングします。トレーニング用のイメージを選択するときは、アダプターでターゲットとする各ラベルの、予想される反応を示すイメージを複数含めるようにします。

トレーニングデータセットを作成するには、次の 2 種類のイメージからいずれかを提供する必要があります。

- 偽陽性の予測を含むイメージ。例えば、ベースモデルはアルコールが含まれていると予測したものの、実際は含まれていないイメージ。
- 偽陰性の予測を含むイメージ。例えば、ベースモデルはアルコールが含まれていないと予測したものの、実際は含まれているイメージ。

バランスの取れたデータセットを作成するには、次の 2 種類のイメージからいずれかを提供することが推奨されます。

- 真陽性の予測を含むイメージ。例えば、アルコールが含まれていることをベースモデルが正確に予測しているイメージ。偽陽性のイメージを提供する場合は、これらのイメージを提供することが推奨されます。
- 真陰性の予測を含むイメージ。例えば、アルコールが含まれていないことをベースモデルが正確に予測しているイメージ。偽陰性のイメージを提供する場合は、これらのイメージを提供することが推奨されます。

ラベル

ラベルは、オブジェクト、イベント、概念、アクティビティのいずれかを参照します。コンテンツモデレーションでは、ラベルは、不適切なコンテンツや不快な迷惑コンテンツのインスタンスです。

Rekognition のベースモデルをトレーニングすることでアダプターを作成する場合、イメージにラベルを割り当てることを注釈と言います。Rekognition コンソールでアダプターをトレーニングするときは、コンソールを使って、ラベルを選択し、そのラベルに対応するイメージにタグを付けることで、イメージに注釈を追加します。このプロセスを通じて、モデルは割り当てられたラベルに基づきイメージの要素を識別する方法を学習します。このリンク処理により、モデルは、アダプターの作成時に最も関連性の高いコンテンツに集中することができ、イメージ分析の精度を高めることができます。

あるいはマニフェストファイルを提供することもできます。このファイルには、イメージに関する情報とそれに付随する注釈が含まれています。

トレーニングデータセットとテストデータセットの作成

トレーニングデータセットは、モデルの微調整やカスタムアダプターの作成の土台となるものです。トレーニングデータセットは、モデルが学習できるよう、注釈付きで提供する必要があります。モデルはこのデータセットから学習することにより、指定したタイプのイメージに関するパフォーマンスを高めることができます。

精度を高めるには、イメージに注釈やラベルを付けてトレーニングデータセットを作成する必要があります。このためには以下の 2 つの方法があります。

- 手動でのラベル割り当て - Rekognition コンソールを使って、データセットに含めるイメージをアップロードすることでトレーニングデータセットを作成し、そのイメージに手動でラベルを割り当てます。

- マニフェストファイル — マニフェストファイルを使用してアダプターをトレーニングします。マニフェストファイルには、トレーニングイメージとテストイメージのグラウンドトゥルースの注釈と、トレーニングイメージの場所に関する情報が含まれています。Rekognition APIs を使用してアダプターをトレーニングするとき、または AWS コンソールを使用するときに、マニフェストファイルを指定できます。

テストデータセットは、トレーニング後のアダプターのパフォーマンスを評価する際に使用します。信頼性の高い評価を行うため、テスト用データセットは、元のトレーニングデータセットのうち、モデルがこれまでに処理したことのないものを使用して作成されます。このプロセスによりアダプターのパフォーマンスを常に最新のデータで評価して、正確な測定値と指標を得ることができます。最適な精度の向上については、「トレーニングアダプターのベストプラクティス」を参照してください。

CLI と SDKs AWS を使用したアダプターの管理

Rekognition では、事前トレーニングされたコンピュータビジョンモデルを活用した、複数の機能を利用できます。これらのモデルを使用すると、ラベル検出やコンテンツモデレーションなどのタスクを実行できます。また、こうした特定のモデルを、アダプターを使用してカスタマイズすることもできます。

Rekognition のプロジェクト作成およびプロジェクト管理 API (CreateProject や CreateProjectVersion) を使用すると、アダプターの作成およびトレーニングが行えます。以下のページでは、 API オペレーションを使用して、 AWS コンソール、選択した AWS SDK、または AWS CLI を使用してアダプターを作成、トレーニング、管理する方法について説明します。

アダプターをトレーニングした後は、サポートされている機能を使って推論を行う際にこれを使用できます。現在、アダプターはコンテンツモデレーション機能を使用するときにサポートされています。

AWS SDK を使用してアダプターをトレーニングするときは、グラウンドトゥルースラベル (画像注釈) をマニフェストファイルの形式で指定する必要があります。または、Rekognition コンソールを使用して、アダプターを作成しトレーニングすることもできます。

> **Note**
>
> アダプターはコピーできません。コピーできるのは、Rekognition Custom Labels のプロジェクトバージョンのみです。

トピック

- アダプターのステータス
- プロジェクトの作成
- プロジェクトの記述
- プロジェクトの削除
- プロジェクトバージョンの作成
- プロジェクトバージョンの記述
- プロジェクトバージョンの削除

アダプターのステータス

カスタムモデレーションアダプター (プロジェクトバージョン) は、以下のいずれかのステータスになります。

- TRAINING_IN_PROGRESS - アダプターは、トレーニングドキュメントとして提供したファイルでトレーニング中です。
- TRAINING_COMPLETED - トレーニングが正常に完了し、アダプターはパフォーマンスの確認準備ができています。
- TRAINING_FAILED - アダプターは何らかの理由でトレーニングを完了できませんでした。失敗の原因について、出力マニフェストファイルと出力マニフェスト概要を確認してください。
- DELETING - アダプターは削除中です。
- DEPRECATED - アダプターはコンテンツモデレーション基本モデルの古いバージョンでトレーニングされています。猶予期間中であり、基本モデルの新しいバージョンのリリースから 60〜90 日以内に期限切れになります。猶予期間中も、アダプターを DetectModerationLabels または StartMediaAnalysisJob API オペレーションで推論に使用できます。アダプターの有効期限については、「カスタムモデレーションコンソール」を参照してください。
- EXPIRED - アダプターは、古いバージョンのコンテンツモデレーション基本モデルでトレーニングされており、DetectModerationLabels または StartMediaAnalysisJob API オペレーションでカスタム結果を取得するために使用できなくなりました。期限切れのアダプターを推論リクエストで指定した場合は無視され、代わりに最新バージョンのカスタムモデレーション基本モデルからレスポンスが返されます。

プロジェクトの作成

CreateProject オペレーションでは、Rekognition のラベル検出オペレーション用のアダプターを含むプロジェクトを作成できます。プロジェクトとはリソースのグループであ

り、DetectModerationLabels のようなラベル検出オペレーションの場合、ベースの Rekognition モデルのカスタマイズに使用できるアダプターを、プロジェクトを使って保存できます。CreateProject を呼び出すときに、作成するプロジェクトの名前を、ProjectName 引数で指定します。

AWS コンソールを使用してプロジェクトを作成するには：

- Rekognition コンソールにサインインします。
- [カスタムモデレーション] をクリックします。
- [プロジェクトを作成] を選択します。
- [新しいプロジェクトを作成] または [既存のプロジェクトに追加] を選択します。
- [プロジェクト名] を追加します。
- [アダプター名] を追加します。
- 必要に応じて説明を追加します。
- トレーニングイメージのインポート方法 (マニフェストファイル、S3 バケットから、コンピュータから) を選択します。
- トレーニングデータを自動分割するか、マニフェストファイルをインポートするか、いずれかを選択します。
- プロジェクトを自動的に更新するか否かを選択します。
- [プロジェクトを作成] をクリックします。

CLI と SDK AWS を使用してプロジェクトを作成するには：

1. まだインストールしていない場合は、 AWS CLI と AWS SDKs をインストールして設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次のコードを実行してプロジェクトを作成します。

CLI

```
# Request
# Creating Content Moderation Project
aws rekognition create-project \
    --project-name "project-name" \
    --feature CONTENT_MODERATION \
    --auto-update ENABLED
    --profile profile-name
```

プロジェクトの記述

DescribeProjects API を使用することで、プロジェクトに関する情報 (プロジェクトに関連付けられているすべてのアダプターに関する情報など) を取得できます。

CLI と SDK AWS を使用してプロジェクトを記述するには：

1. まだインストールしていない場合は、 CLI と AWS SDKs AWS をインストールして設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次のコードを実行してプロジェクトを記述します。

CLI

```
# Request
# Getting CONTENT_MODERATION project details
aws rekognition describe-projects \
    --features CONTENT_MODERATION
    --profile profile-name
```

プロジェクトの削除

プロジェクトは、Amazon Rekognition コンソールを使用するか、DeleteProject API を使用することで削除できます。プロジェクトを削除するには、先に、関連付けられたアダプターをすべて削除する必要があります。一度削除したプロジェクトまたはモデルは、元に戻せません。

AWS コンソールを使用してプロジェクトを削除するには：

- Rekognition コンソールにサインインします。
- [カスタムモデレーション] をクリックします。
- プロジェクト自体を削除するときは、先に、それに関連付けられたアダプターを削除する必要があります。プロジェクトに関連付けられているアダプターをすべて削除するには、アダプターを選択して [削除] を選択します。
- プロジェクトを選択し、[削除] をクリックします。

CLI と SDK AWS を使用してプロジェクトを削除するには：

1. まだインストールしていない場合は、 CLI と AWS SDKs AWS をインストールして設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次のコードを実行してプロジェクトを削除します。

CLI

```
aws rekognition delete-project
  --project-arn project_arn \
  --profile profile-name
```

プロジェクトバージョンの作成

アダプターをデプロイ用にトレーニングするには、CreateProjectVersion オペレーションを使用します。CreateProjectVersion は、最初に、プロジェクトに関連付けられたアダプターの新しいバージョンを作成し、その後、アダプターのトレーニングを開始します。CreateProjectVersion のレスポンスは、当該モデルのバージョンの Amazon リソースネーム (ARN) です。トレーニングが完了するまでしばらく時間がかかります。現在のステータスは、DescribeProjectVersions を呼び出すことで確認できます。モデルをトレーニングするとき、Rekognition は、プロジェクトに関連付けられたトレーニングデータセットとテストデータセットを使用します。データセットはコンソールで作成できます。詳細については、データセットのセクションを参照してください。

Rekognition コンソールでプロジェクトを作成するには

- AWS Rekognition コンソールにサインインします。
- [カスタムモデレーション] をクリックします。
- プロジェクトを選択します。
- [プロジェクトの詳細] ページで、[アダプターを作成] を選択します。
- [プロジェクトを作成] ページで、[プロジェクトの詳細]、[トレーニング画像]、[テスト画像] に必要な情報を入力し、[プロジェクトを作成] を選択します。
- [画像にラベルを割り当てる] ページで、イメージにラベルを追加し、完了したら [トレーニングを開始] を選択します。

CLI と SDK AWS を使用してプロジェクトバージョンを作成するには：

1. まだインストールしていない場合は、 CLI と AWS SDKs AWS をインストールして設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次のコードを実行してプロジェクトバージョンを作成します。

CLI

```
# Request
aws rekognition create-project-version \
 --project-arn project-arn \
 --training-data '{Assets=[GroundTruthManifest={S3Object="amzn-s3-demo-source-bucket",Name="manifest.json"}]}' \
 --output-config S3Bucket=amzn-s3-demo-destination-bucket,S3KeyPrefix=my-results \
 --feature-config "ContentModeration={ConfidenceThreshold=70}"
 --profile profile-name
```

プロジェクトバージョンの記述

プロジェクトに関連付けられているアダプターを一覧表示して記述するには、DescribeProjectVersions オペレーションを使用します。ProjectVersionArns では最大 10 個のモデルバージョンを指定できます。値を指定しないと、プロジェクト内のすべてのモデルバージョンの説明が返されます。

CLI と SDK AWS を使用してプロジェクトバージョンを記述するには：

1. まだインストールしていない場合は、 CLI と AWS SDKs AWS をインストールして設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次のコードを実行してプロジェクトバージョンを記述します。

CLI

```
aws rekognition describe-project-versions
  --project-arn project_arn \
  --version-names [versions]
```

プロジェクトバージョンの削除

プロジェクトに関連付けられている Rekognition アダプターを削除するには、DeleteProjectVersion オペレーションを使用してます。実行中またはトレーニング中のアダプターは削除できません。アダプターのステータスを確認するには、DescribeProjectVersions オペレーションを呼び出し、そのオペレーションによって返されるステータスフィールドを確認します。実行中のアダプターを停止するには、StopProjectVersion を呼び出します。モデルがトレーニング中の場合は、トレーニングが終了した後に削除します。プロジェクト自体を削除するときは、先に、それに関連付けられたアダプターを削除する必要があります。

Rekognition コンソールでプロジェクトバージョンを削除するには

- Rekognition コンソールにサインインします。
- [カスタムモデレーション] をクリックします。
- [プロジェクト] タブでは、すべてのプロジェクトと関連するアダプターを確認できます。アダプターを選択し、[削除] をクリックします。

CLI と SDK AWS を使用してプロジェクトバージョンを削除するには：

1. まだインストールしていない場合は、 CLI と AWS SDKs AWS をインストールして設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次のコードを実行してプロジェクトバージョンを削除します。

CLI

```
# Request
aws rekognition delete-project-version
  --project-version-arn model_arn \
  --profile profile-name
```

カスタムモデレーションのアダプターのチュートリアル

このチュートリアルでは、Rekognition コンソールを使用してアダプターを作成、トレーニング、評価、使用、管理する方法について説明します。 AWS SDK でアダプターを作成、使用、管理するには、「」を参照してくださいCLI と SDKs AWS を使用したアダプターの管理。

アダプターを使用すると、Rekognition の API オペレーションの精度を高め、モデルの動作を自分のニーズやユースケースに合わせてカスタマイズできます。このチュートリアルでアダプターを作成すると、DetectModerationLabels などのオペレーションで独自のイメージを分析するときにこれを使用できます。また、アダプターを再トレーニングして今後の改善に役立てることができます。

このチュートリアルでは以下の方法を学びます。

- Rekognition コンソールを使用してプロジェクトを作成する。
- トレーニングデータに注釈を付ける。
- トレーニングデータセットに基づきアダプターをトレーニングする。
- アダプターのパフォーマンスをレビューする。
- アダプターをイメージ解析に使用する。

前提条件

このチュートリアルを始める前に、「アダプターの作成と使用」を読むことをお勧めします。

アダプターを作成するときは、Rekognition コンソールのツールを使用すると、プロジェクトを作成し、イメージをアップロードして注釈を付け、それらのイメージに基づいてアダプターをトレーニングできます。開始するには、プロジェクトの作成およびアダプターのトレーニングを参照してください。

あるいは、Rekognition のコンソールまたは API を使用してイメージの予測を取得し、その予測を検証してから、それらの予測に基づいてアダプターを学習することもできます。開始するには、アダプターの一括分析、予測検証、トレーニングを参照してください。

イメージの注釈

イメージには、Rekognition コンソールでイメージにラベルを付けることで注釈を付けることができます。また、Rekognition の一括分析を使用してイメージに注釈を付け、その後で正しくラベルが付いていることを確認することもできます。注釈を付けるには、次のいずれかのトピックを選択します。

トピック

- プロジェクトの作成およびアダプターのトレーニング
- アダプターの一括分析、予測検証、トレーニング

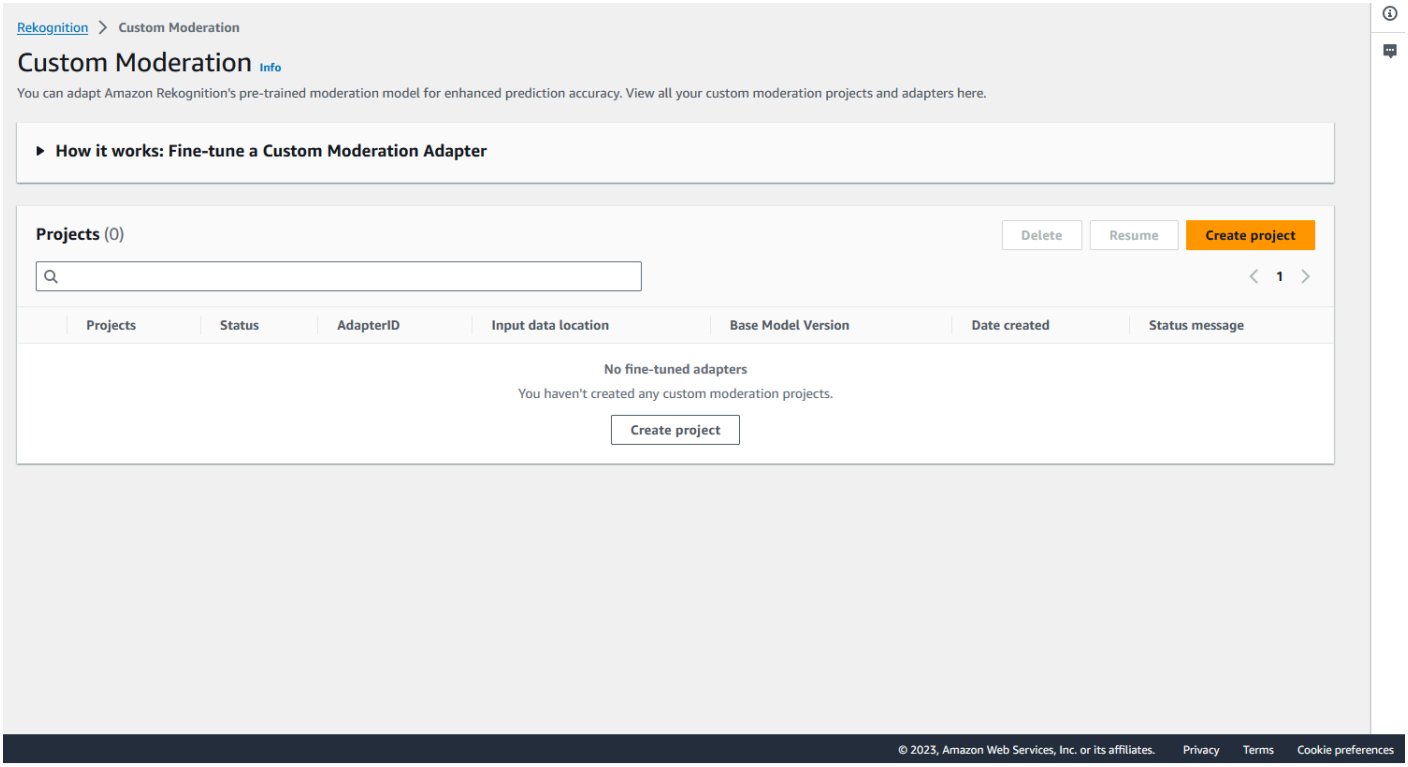
プロジェクトの作成およびアダプターのトレーニング

Rekognition コンソールを使用してイメージに注釈を付けて、アダプターをトレーニングするには、以下のステップを実行します。

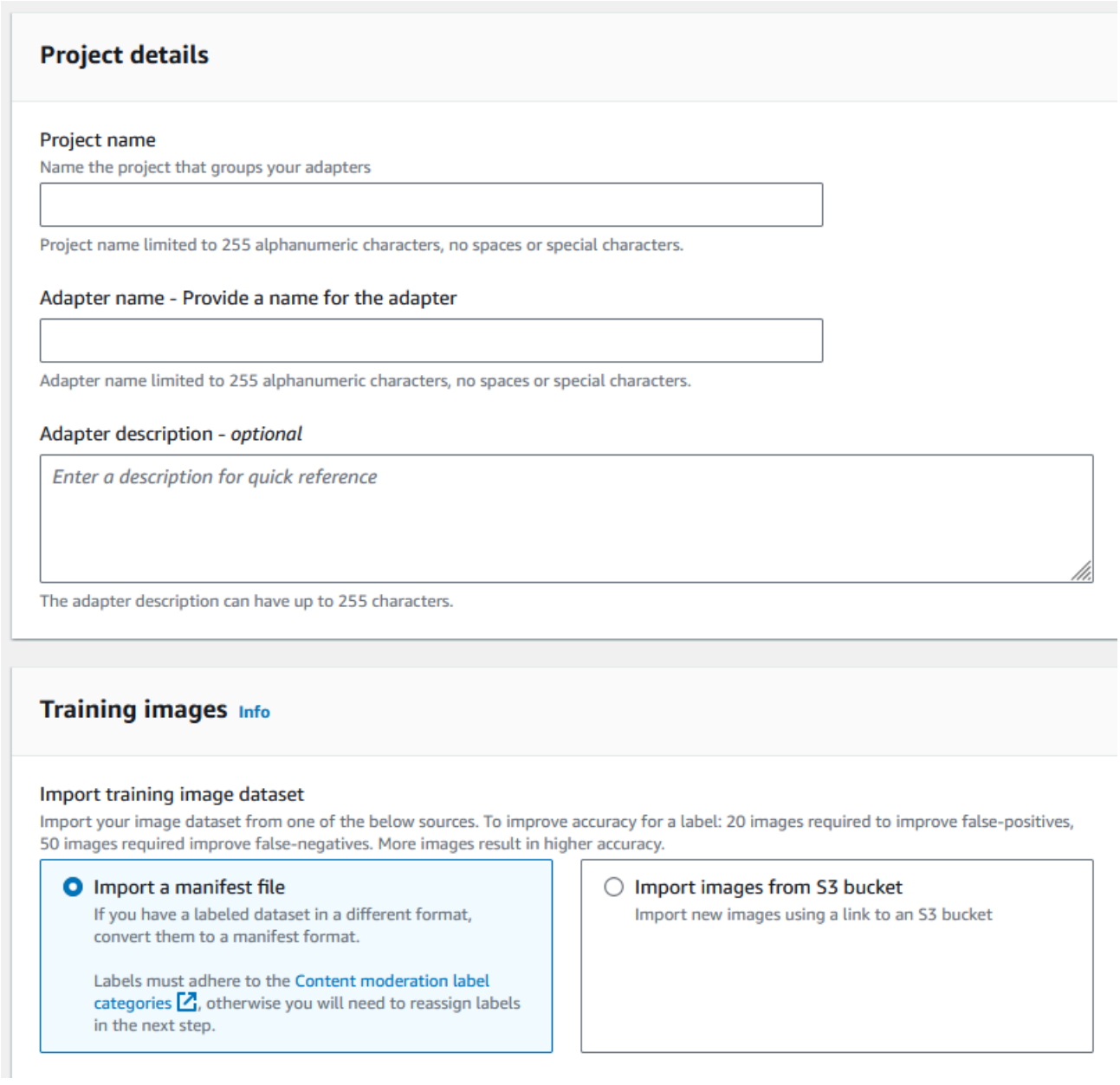
プロジェクトの作成

アダプターをトレーニングまたは使用するときは、先に、アダプターを含めるプロジェクトを作成する必要があります。また、アダプターのトレーニングに使用するイメージも提供する必要があります。プロジェクト、アダプター、イメージデータセットを作成するには

1. AWS マネジメントコンソールにサインインし、https://console.aws.amazon.com/rekognition/ で Rekognition コンソールを開きます。
2. 左側のペインで、[カスタムモデレーション] を選択します。Rekognition カスタムモデレーションのランディングページが開きます。



3. カスタムモデレーションのランディングページにはすべてのプロジェクトとアダプターのリストが表示されています。また、アダプターを作成するボタンもあります。[プロジェクトを作成] を選択し、新規プロジェクトとアダプターを作成します。
4. アダプターを作成するのが初めてである場合は、画面上で、プロジェクトとアダプターに関連するファイルを保存する、Amazon S3 バケットを作成するように指示されます。[Amazon S3 バケットを作成] を選択します。
5. 次のページで、アダプター名とプロジェクト名を入力します。必要に応じてアダプターの説明も入力します。

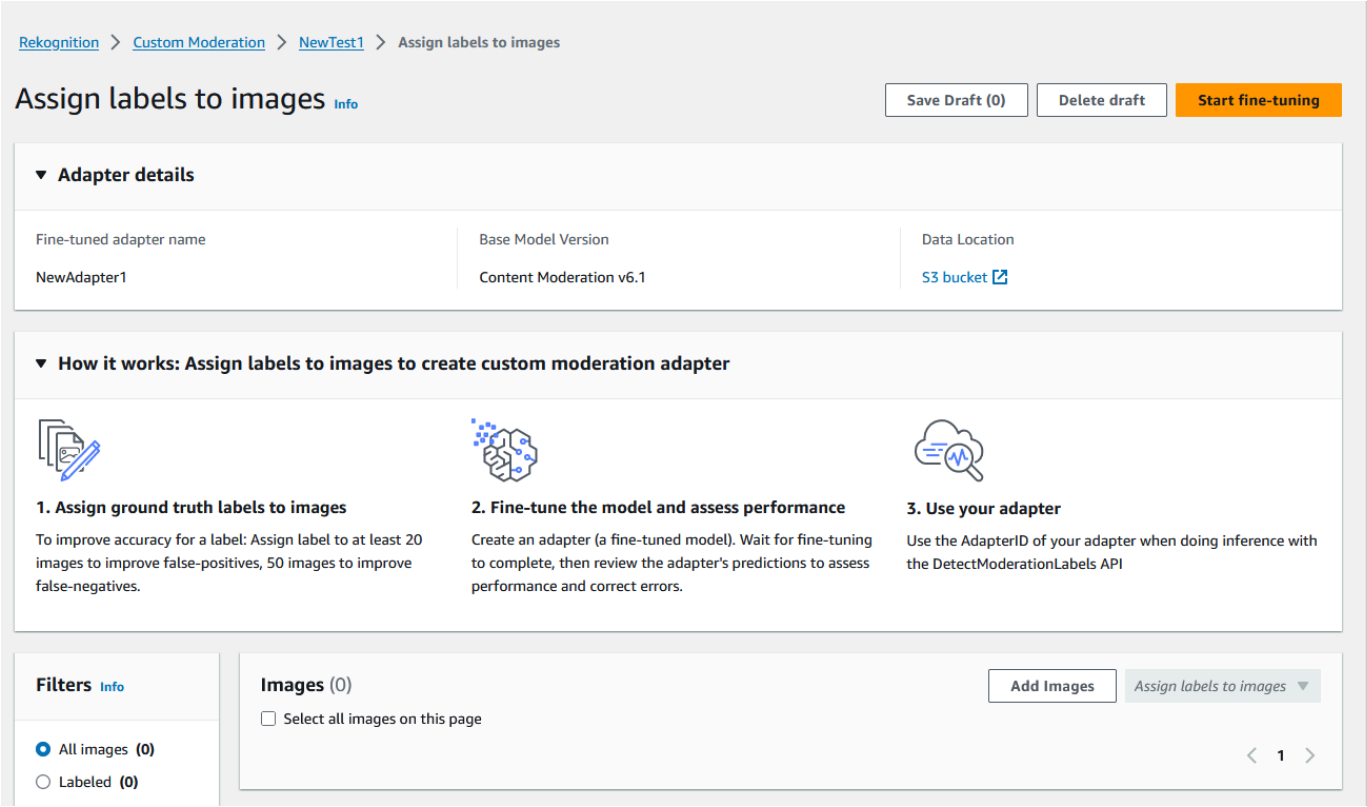


6. このステップでは、アダプターのイメージも提供します。[コンピュータからイメージをインポート]、[マニフェストファイルをインポート]、[Amazon S3 バケットからイメージをインポート] のいずれかを選択できます。Amazon S3 バケットからイメージをインポートする場合は、トレーニングイメージを含むバケットとフォルダへのパスを指定します。コンピュータからイメージを直接アップロードする場合は、一度にアップロードできるイメージの数が最大 30 であることにご注意ください。注釈を含むマニフェストファイルを使用する場合は、以下で説明するイメージ注釈の手順は飛ばして、「アダプターのパフォーマンスのレビュー」セクションに進みます。
7. [テストデータセットの詳細] セクションで、[自動分割] を選択して Rekognition に適切な割合のイメージをテストデータとして自動選択させるか、[マニフェストファイルを手動でインポート] を選択します。
8. この情報を入力したら、[プロジェクトを作成] を選択します。

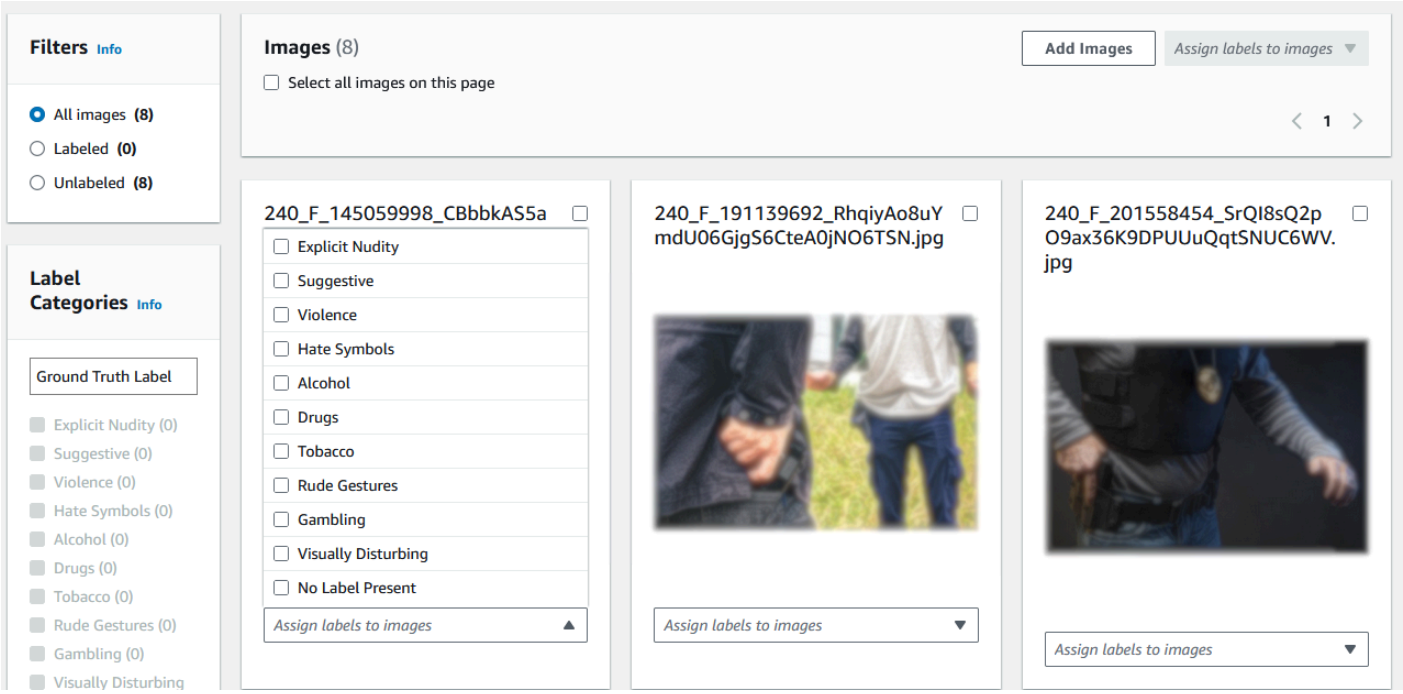
アダプターのトレーニング

注釈の付いていないイメージでアダプターをトレーニングするには

1. アダプターを含むプロジェクトを選択し、[イメージにラベルを割り当てる] オプションを選択します。
2. [イメージにラベルを割り当てる] ページには、トレーニングイメージとしてアップロードされたイメージがすべて表示されています。これらのイメージは、左側にある 2 つの属性選択パネルを使って、ラベル付き/ラベルなしのステータス、およびラベルカテゴリ別に、フィルタリングできます。トレーニングデータセットにイメージを追加するときは、[画像を追加] を選択します。



3. トレーニングデータセットにイメージを追加したら、必ず、ラベルを使ってイメージに注釈を付けます。イメージをアップロードすると、[画像にラベルを割り当てる] ページが更新され、アップロードしたイメージが表示されます。画面上で、Rekognition のモデレーションでサポートされているラベルのドロップダウンリストからイメージに適したラベルを選択するように指示されます。ラベルは複数選べます。
4. トレーニングデータ内のすべてのイメージにラベルを付けるまで、この作業を続けます。
5. すべてのデータにラベルを付けたら、[トレーニングを開始] を選択してモデルのトレーニングを開始し、アダプターを作成します。



6. トレーニングプロセスを開始する前に、必要に応じてアダプターにタグを追加できます。アダプターにカスタム暗号化キーを提供するか、KMS AWS キーを使用することもできます。必要なタグを追加し、暗号化をカスタマイズしたら、[アダプターをトレーニング] をクリックしてアダプターのトレーニングプロセスを開始します。
7. アダプターのトレーニングが完了するまで待機します。トレーニングが完了すると、アダプターの作成が完了したことの通知が届きます。

アダプターのステータスが [トレーニング完了] に変わったら、アダプターのメトリクスをレビューできます。

アダプターの一括分析、予測検証、トレーニング

Rekognition のコンテンツモデレーションモデルによる一括分析予測を検証して次の手順を実行し、アダプターをトレーニングします。

Rekognition のコンテンツモデレーションモデルによる予測を検証してアダプターをトレーニングするには、以下を行う必要があります。

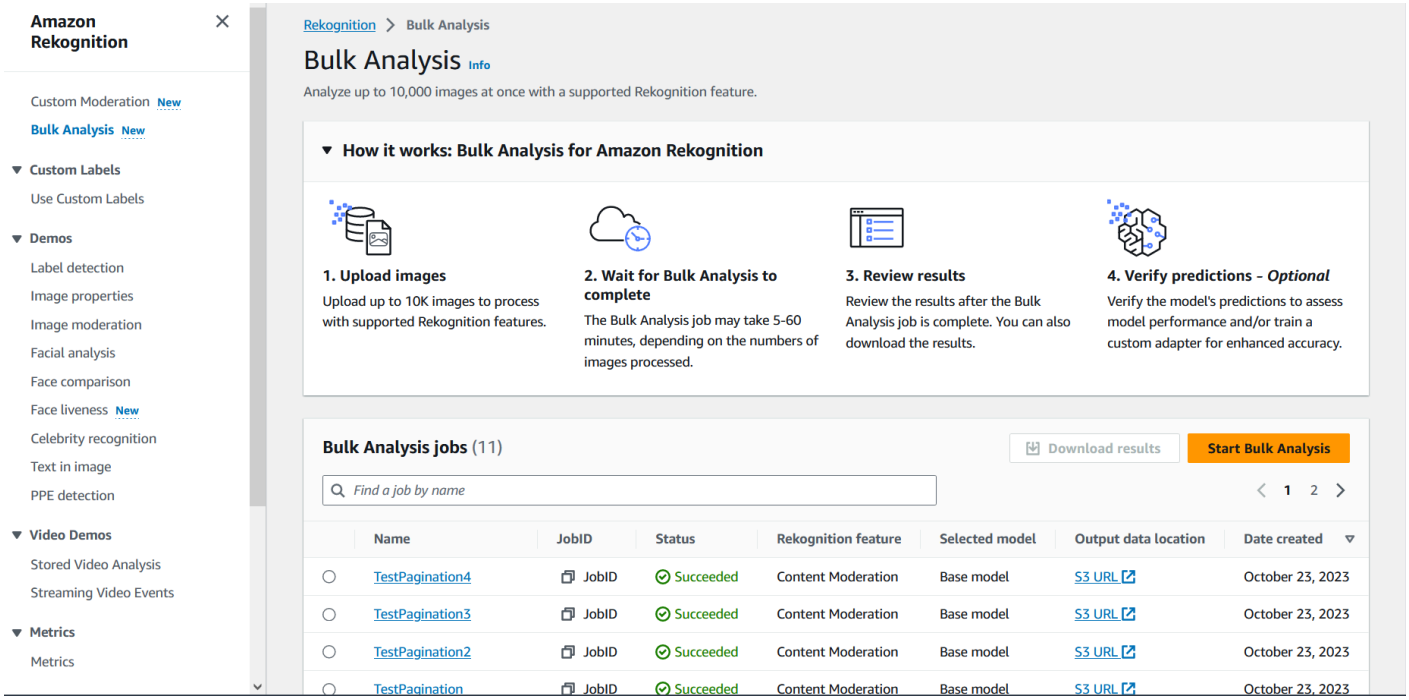
1. イメージの一括分析を実行します。
2. イメージについて返された予測を検証します。

イメージの予測は、Rekognition のベースモデル、または作成済みのアダプターを使用して一括分析を実行することで取得できます。

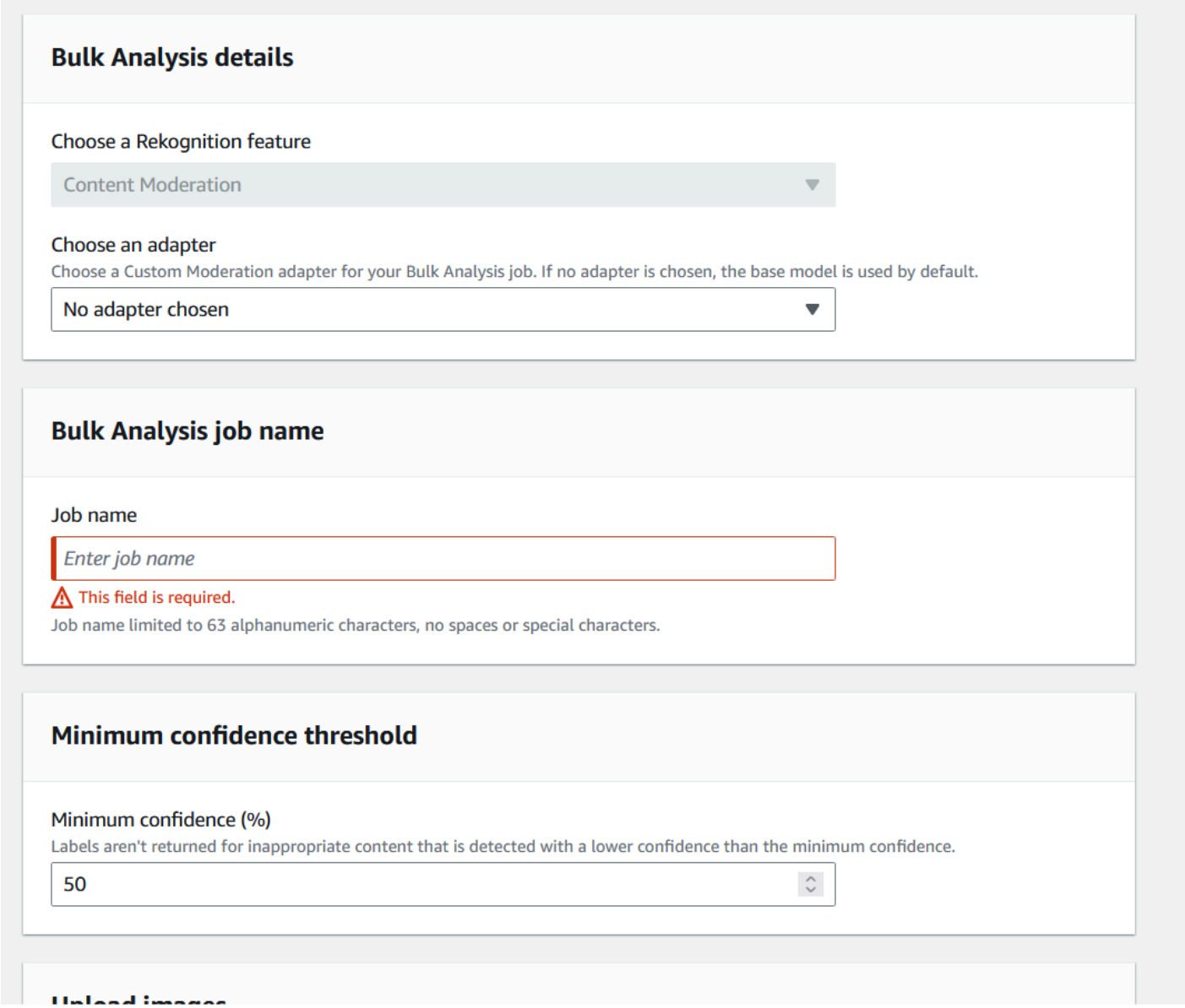
イメージの一括分析を実行

検証した予測に基づいてアダプターをトレーニングするには、最初に、一括分析ジョブを実行して、Rekognition のベースモデルまたは任意のアダプターを使用してイメージのバッチを分析する必要があります。一括分析ジョブを実行するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/rekognition/ で Amazon Rekognition コンソールを開きます。
2. 左側のペインで [一括分析] を選択します。[一括分析] のランディングページが表示されます。[一括分析を開始] を選択します。一括分析機能の概要に、画像のアップロード、分析の待機、結果の確認の手順が表示されており、オプションでモデル予測を検証する手順も示されています。基本モデルを使用したコンテンツモデレーションの最近の一括分析ジョブが一覧表示されます。

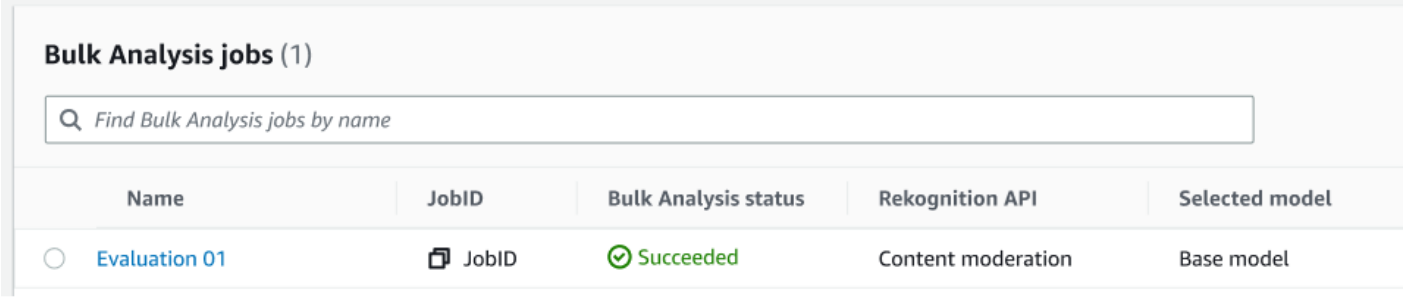


3. アダプターを作成するのが初めてである場合は、画面上で、プロジェクトとアダプターに関連するファイルを保存する、Amazon Simple Storage Service バケットを作成するように指示されます。[Amazon S3 バケットを作成] を選択します。
4. [アダプターを選択する] ドロップダウンメニューから、一括分析に使用するアダプターを選択します。アダプターを選択しない場合は、デフォルトでベースモデルが使用されます。このチュートリアルでは、アダプターを選択しません。

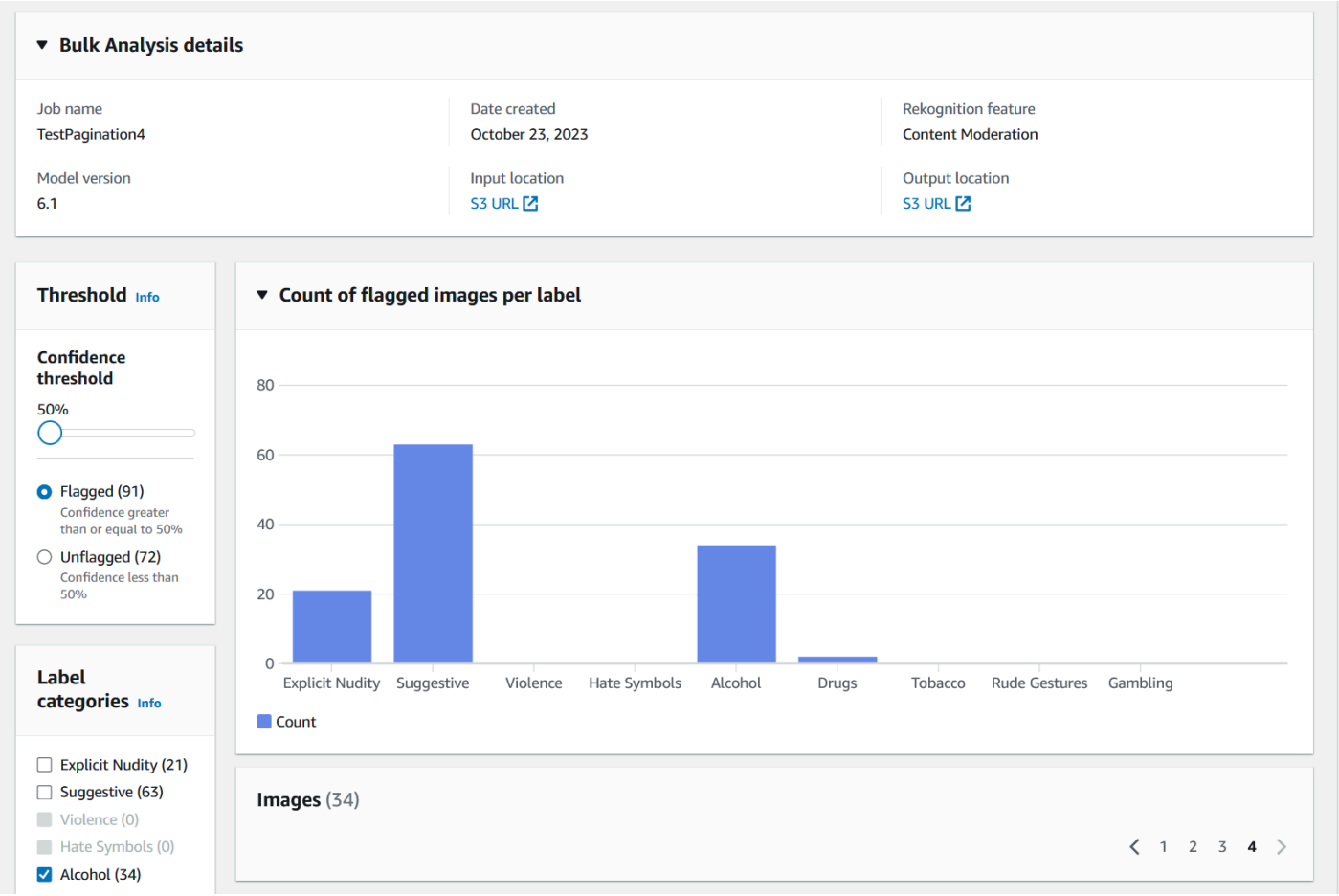


5. [一括分析ジョブ名] フィールドに、一括分析ジョブの名前を入力します。
6. [最小信頼度しきい値] の値を選択します。選択した信頼度しきい値を下回るラベル予測は返されません。選択した最小信頼度しきい値を下回る信頼度しきい値は、後ほどモデルのパフォーマンスを評価するときに、調整できませんのでご注意ください。
7. このステップでは、一括分析で分析するイメージも提供します。これらのイメージはアダプターのトレーニングにも使用できます。[コンピュータからイメージをアップロード] または [Amazon S3 バケットからイメージをインポート] のいずれかを選択できます。Amazon S3 バケットからドキュメントをインポートする場合は、トレーニングイメージを含むバケットとフォルダへのパスを指定します。コンピュータからドキュメントを直接アップロードする場合は、一度にアップロードできるイメージの数が最大 50 であることにご注意ください。

8. この情報を入力したら、[分析を開始する] を選択します。Rekognition のベースモデルを使用した解析プロセスが開始します。
9. 一括分析ジョブのステータスは、一括分析のメインページにある、ジョブの一括分析ステータスから確認できます。一括分析のステータスが「成功」に変わったら、分析の結果をレビューできます。



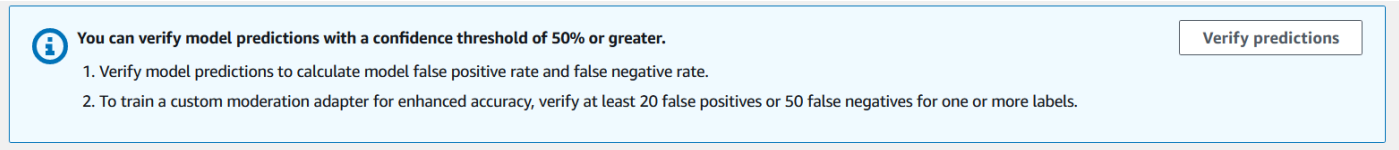
10[一括分析ジョブ] のリストの中から、作成した分析を選択します。
11.一括分析の詳細ページでは、アップロードしたイメージに対して Rekognition のベースモデルが行った予測を確認できます。
12.ベースモデルのパフォーマンスをレビューします。イメージにラベルを割り当てるときにアダプターに必要になる信頼度しきい値は、信頼度しきい値のスライダーを使って変更できます。信頼度しきい値を変更すると、フラグ付きのインスタンスとフラグなしのインスタンスの数が変わります。[ラベルカテゴリ] のパネルには、Rekognition が認識する最上位のカテゴリが表示されます。このリストでカテゴリを選択すると、そのラベルが割り当てられているすべてのイメージを表示できます。



予測の検証

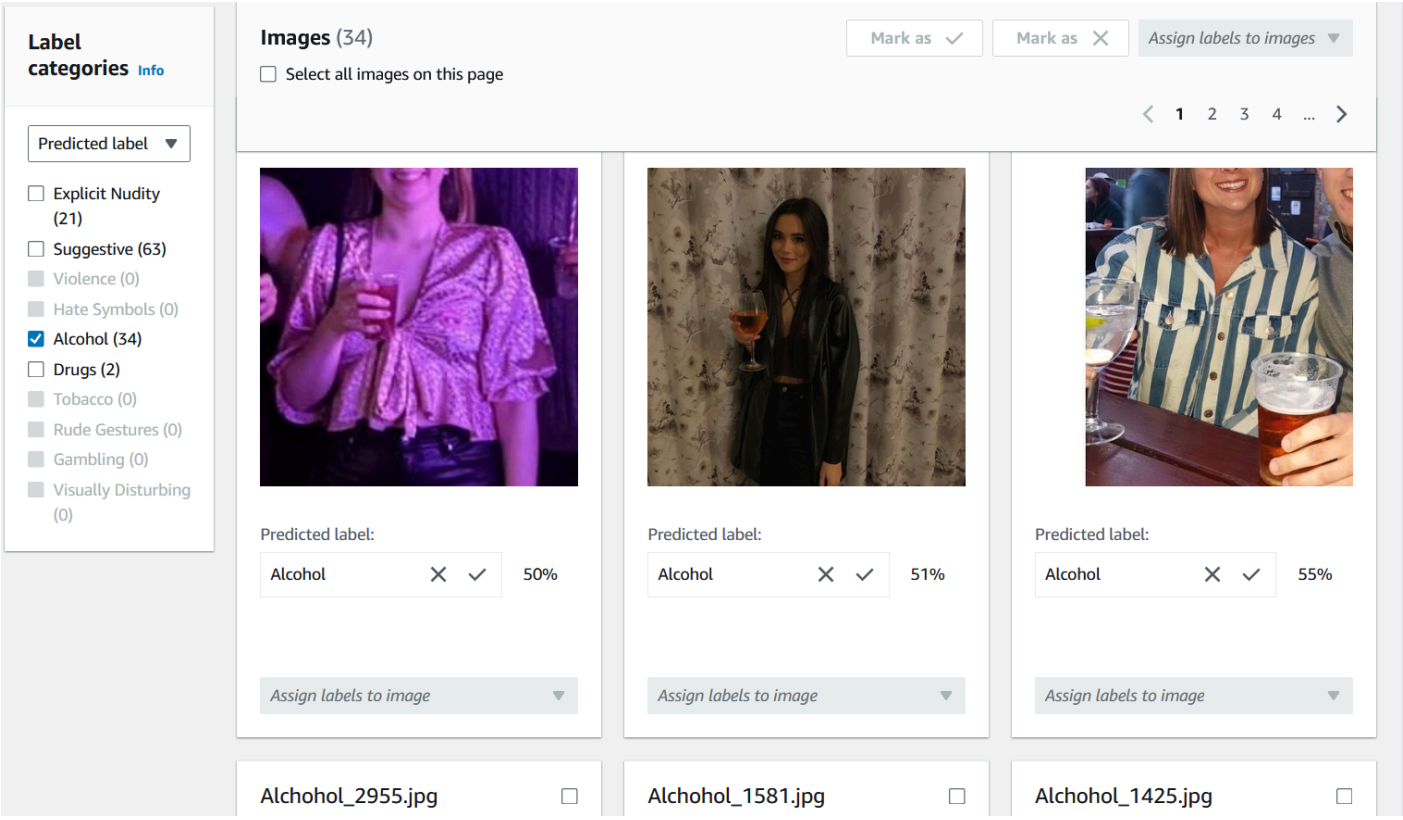
Rekognition のベースモデルまたは選択したアダプターの精度をレビューしてその精度を高めるには、検証ワークフローを使用します。

1. ベースモデルのパフォーマンスのレビューを終えたら、予測を検証する必要があります。予測を修正することで、アダプターをトレーニングできます。[一括分析] ページの上部にある [予測を検証] をクリックします。

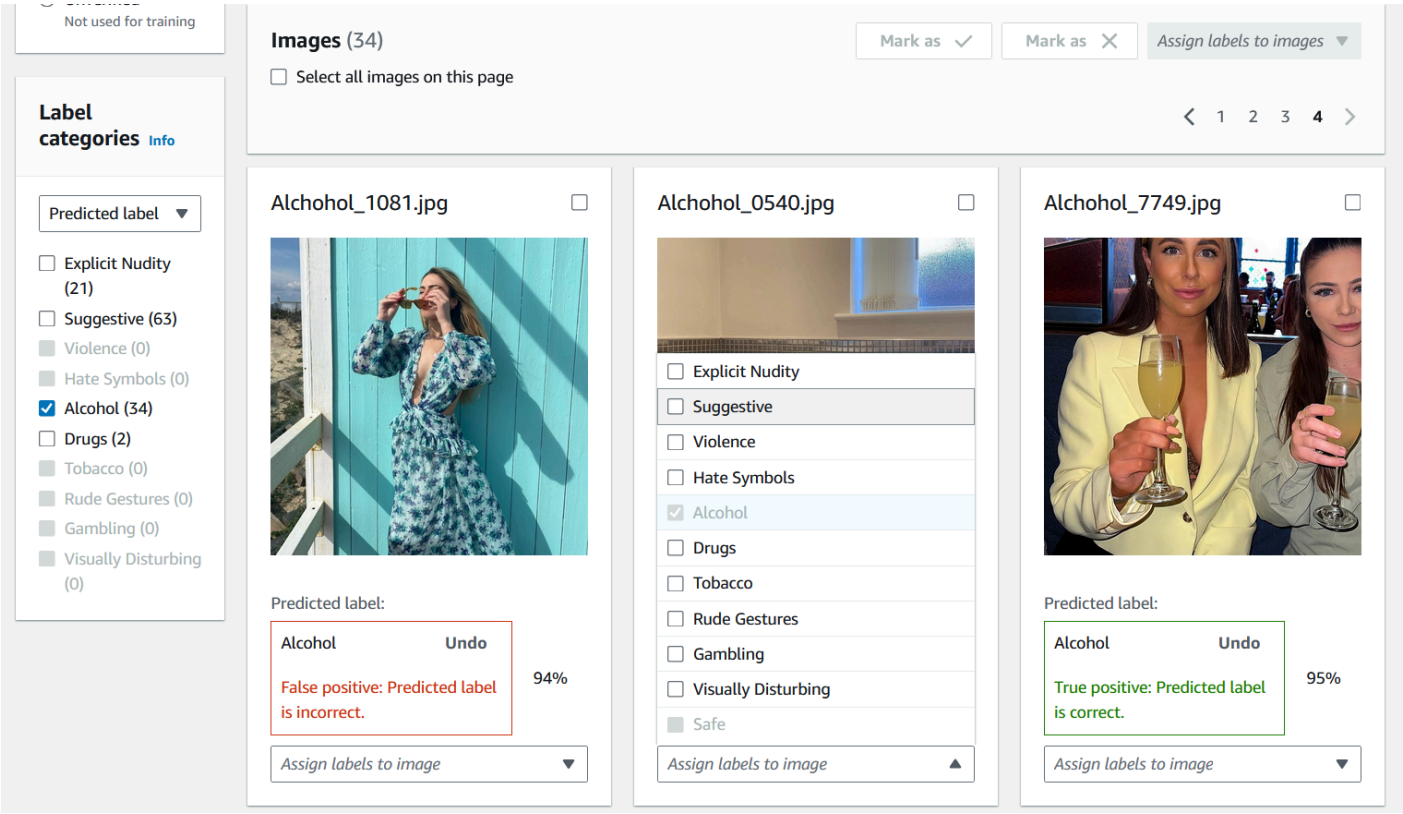


2. [予測を検証] ページには、Rekognition のベースモデルまたは選択したアダプターに提供したすべてのイメージが、各イメージの予測ラベルと共に表示されます。イメージの下にあるボタンを使って、それぞれの予測が正しいか間違っているかを検証します。予測が間違っている場合は [X] を、正しい場合はチェックマークをクリックします。アダプターをトレーニングするには、1 つ

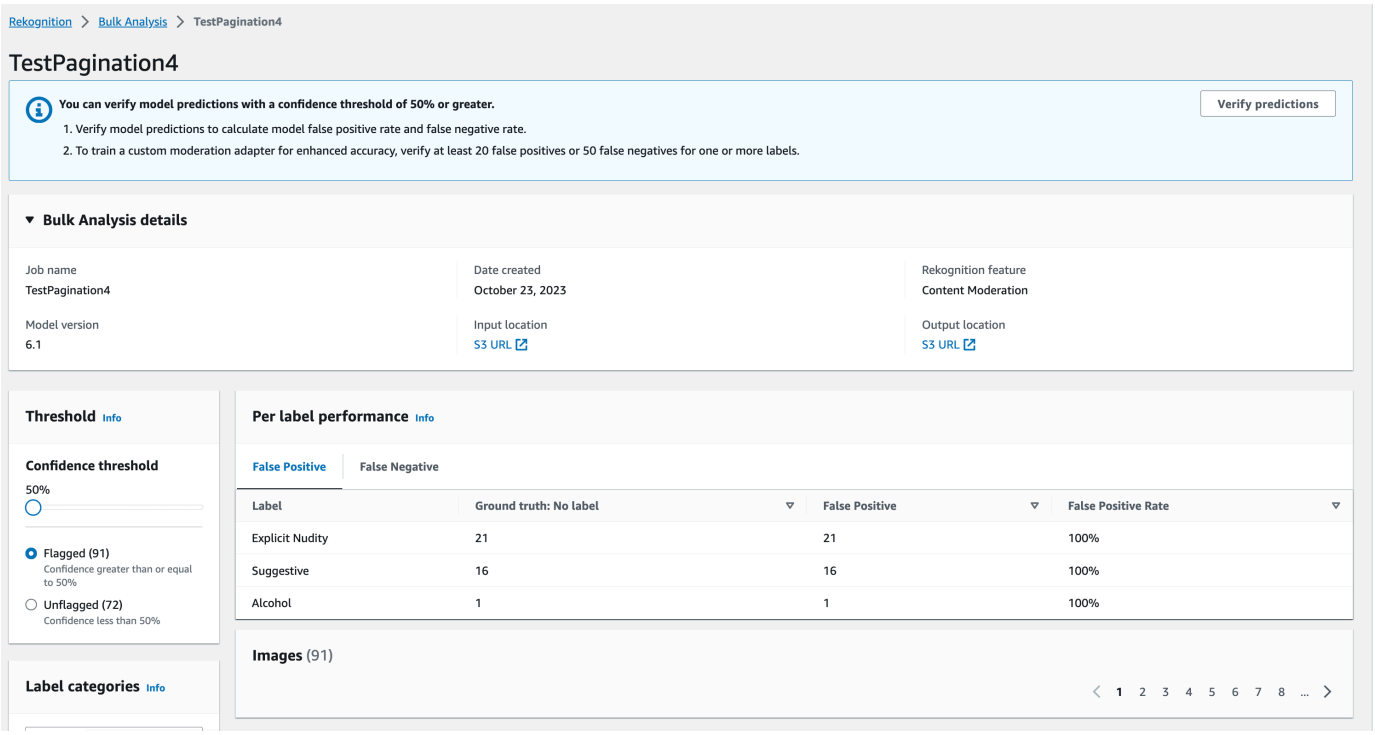
のラベルで 20 件以上の偽陽性予測と 50 件の偽陰性予測を検証する必要があります。検証する予測の数が多ければ多いほど、アダプターのパフォーマンスは向上します。



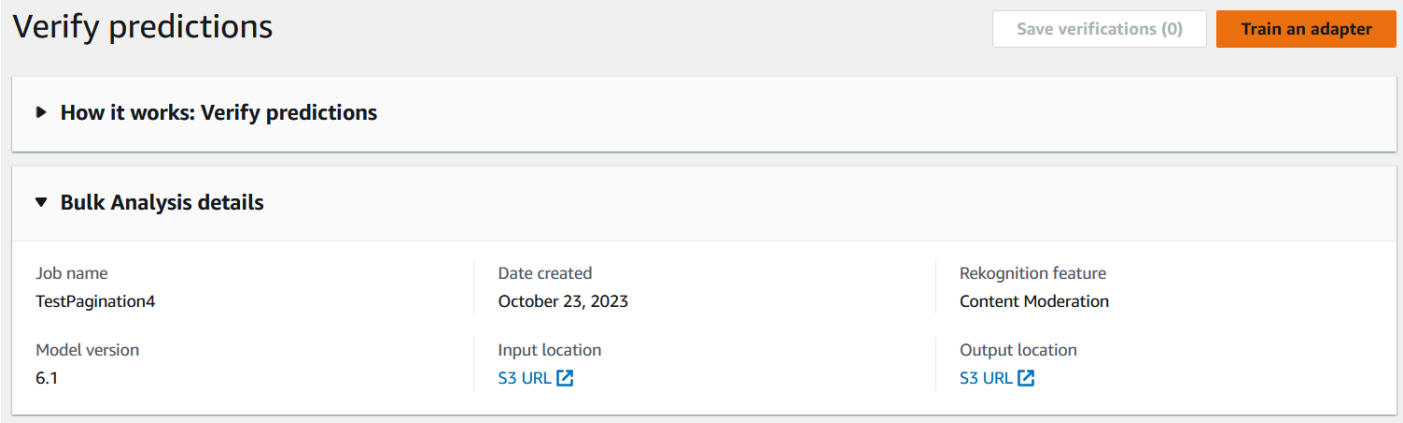
予測を検証すると、イメージの下のテキストが変わり、検証した予測の種類が表示されます。イメージを検証できたら、[画像にラベルを割り当てる] メニューを使ってイメージにラベルを追加することもできます。選択した信頼度しきい値で、モデルによりフラグの付いた/付いてないイメージを確認したり、カテゴリ別にイメージをフィルタリングしたりできます。



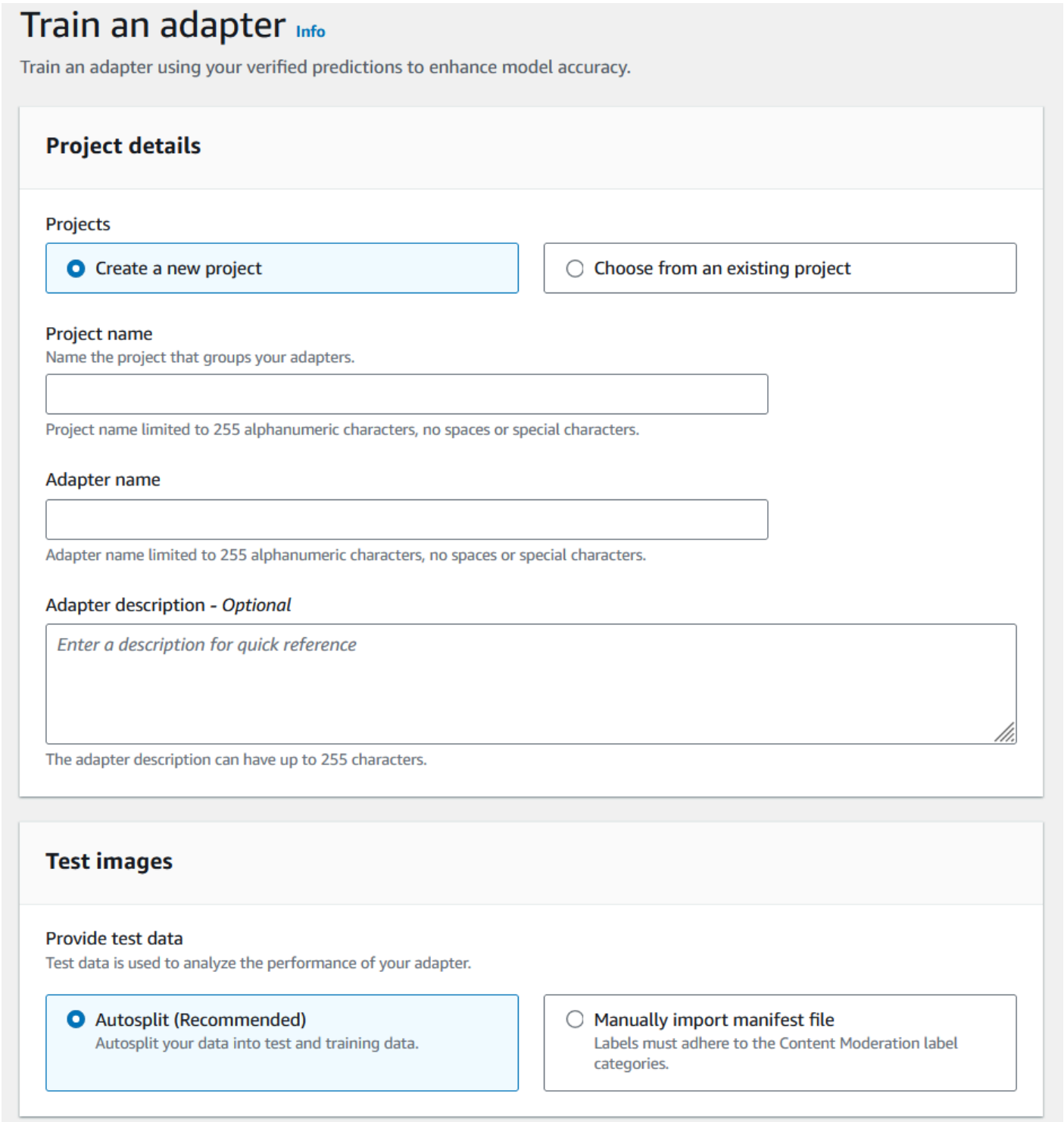
3. 検証の必要な予測をすべて検証すると、[検証] ページの [ラベルごとのパフォーマンス] のセクションに、検証した予測に関する統計が表示されます。これらの統計は、[一括分析の詳細] ページに戻って確認することもできます。



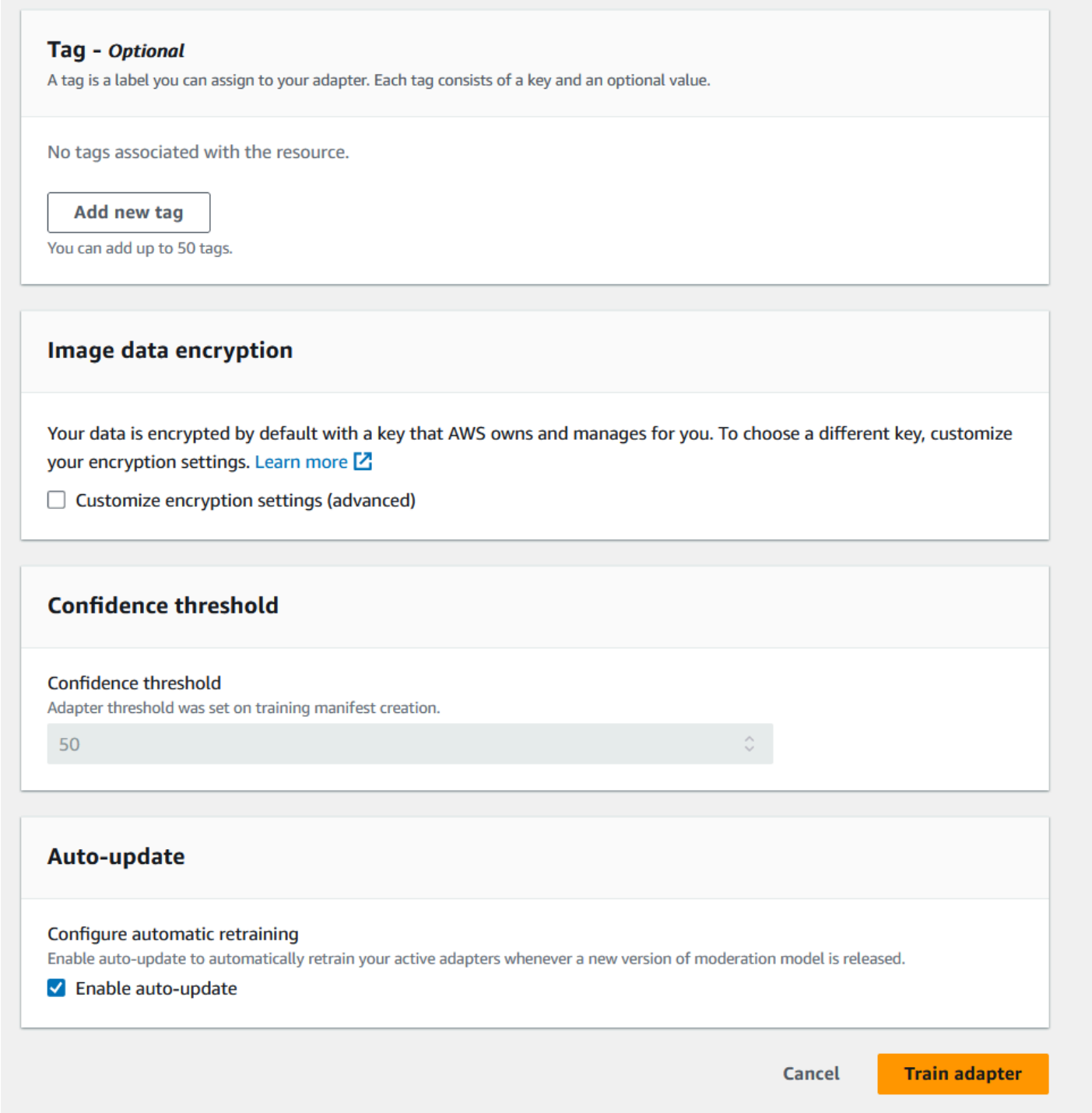
4. [ラベルごとのパフォーマンス] に関する統計に問題がなければ、[予測を検証] ページに戻って、[アダプターをトレーニング] をクリックし、アダプターのトレーニングを開始します。



5. [アダプターをトレーニング] ページでは、プロジェクトを作成するか、既存のプロジェクトを選択するように指示されます。プロジェクトと、そのプロジェクトに含まれるアダプターの名前を入力します。テストイメージのソースも指定する必要があります。イメージを指定するときは、[自動分割] を選択して Rekognition でトレーニングデータの一部をテストイメージとして自動的に使用するか、マニフェストファイルを手動で指定できます。推奨の方法は、[自動分割] を使用することです。



6. 任意のタグを指定し、デフォルト AWS KMS キーを使用しない場合は AWS キーを指定します。[自動更新] は有効のままにしておくことが推奨されます。
7. [アダプターをトレーニング] をクリックします。

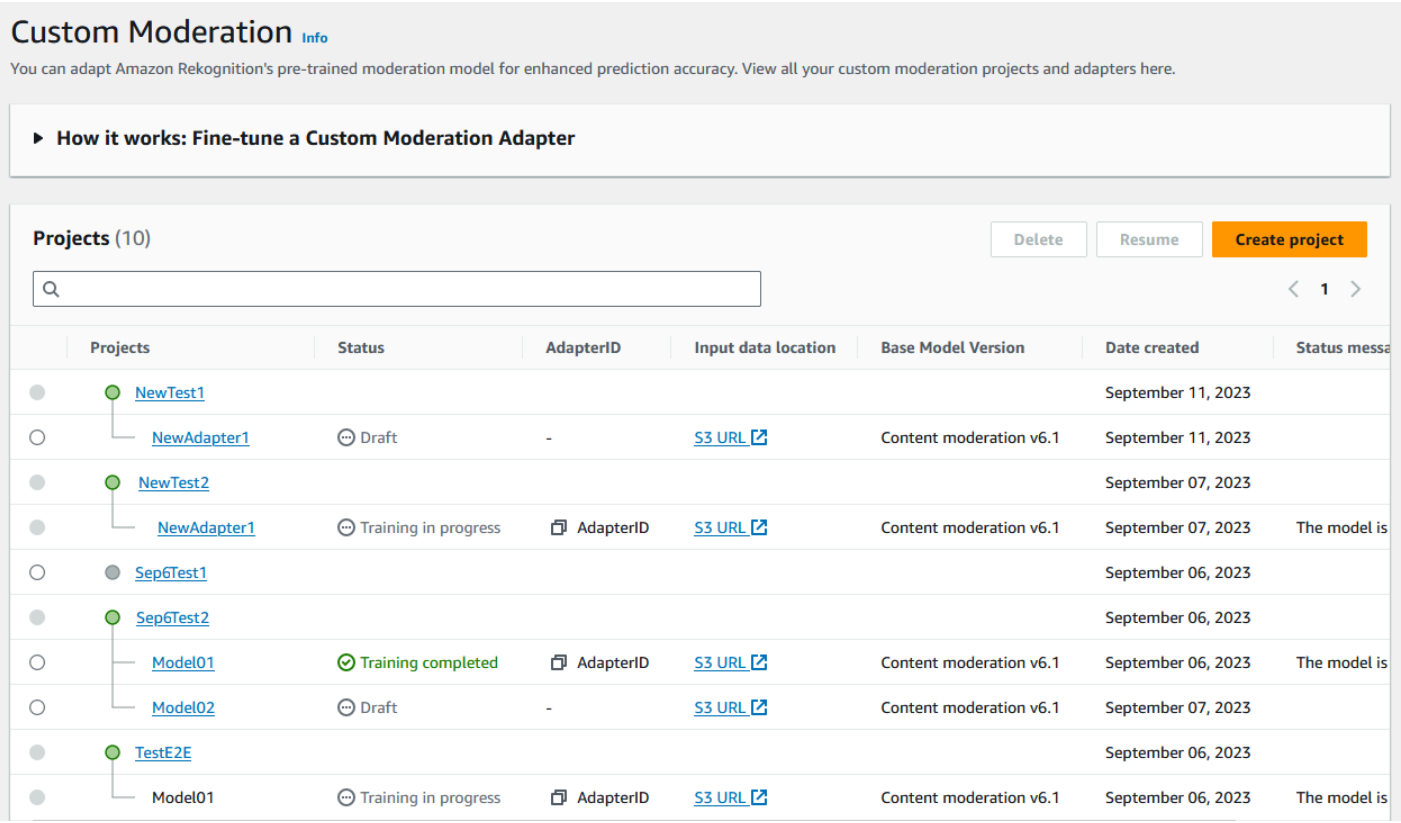


8. カスタムモデレーションのランディングページでアダプターのステータスが [トレーニングの完了] に変わったら、アダプターのパフォーマンスをレビューできます。詳細については「アダプターのパフォーマンスのレビュー」を参照してください。

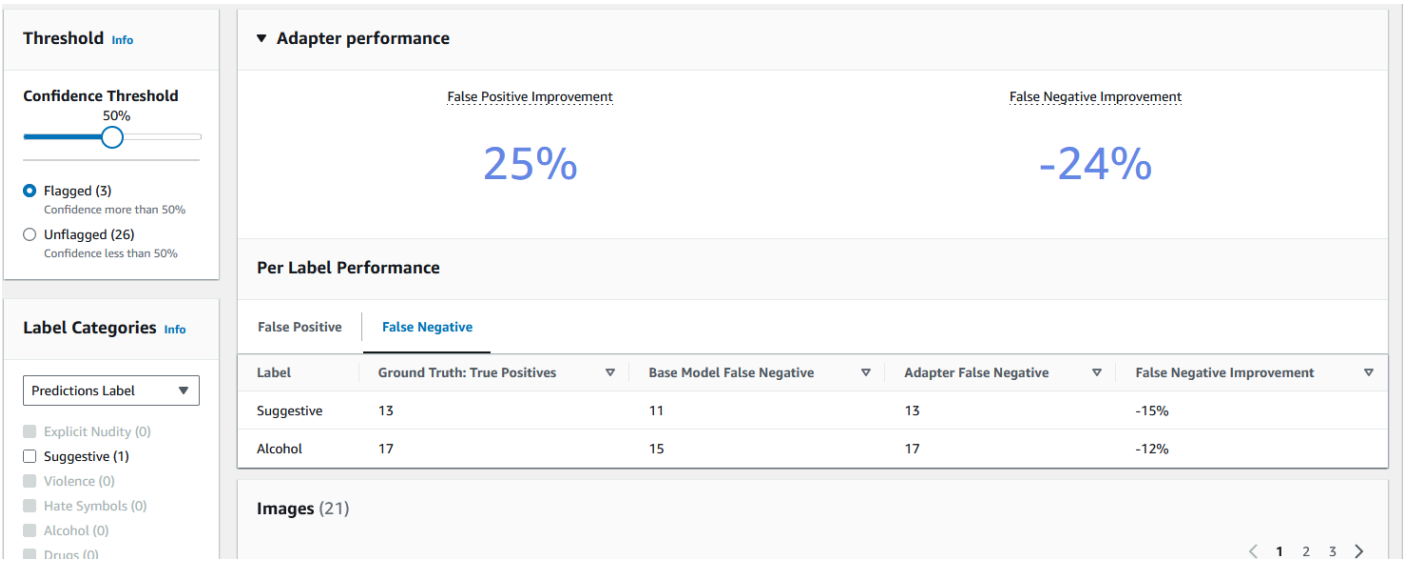
アダプターのパフォーマンスのレビュー

アダプターのパフォーマンスをレビューするには

1. コンソールを使用すると、カスタムモデレーションのランディングページにある [プロジェクト] タブで、プロジェクトに関連付けられているアダプターのステータスを確認できます。カスタムモデレーションのランディングページに移動します。



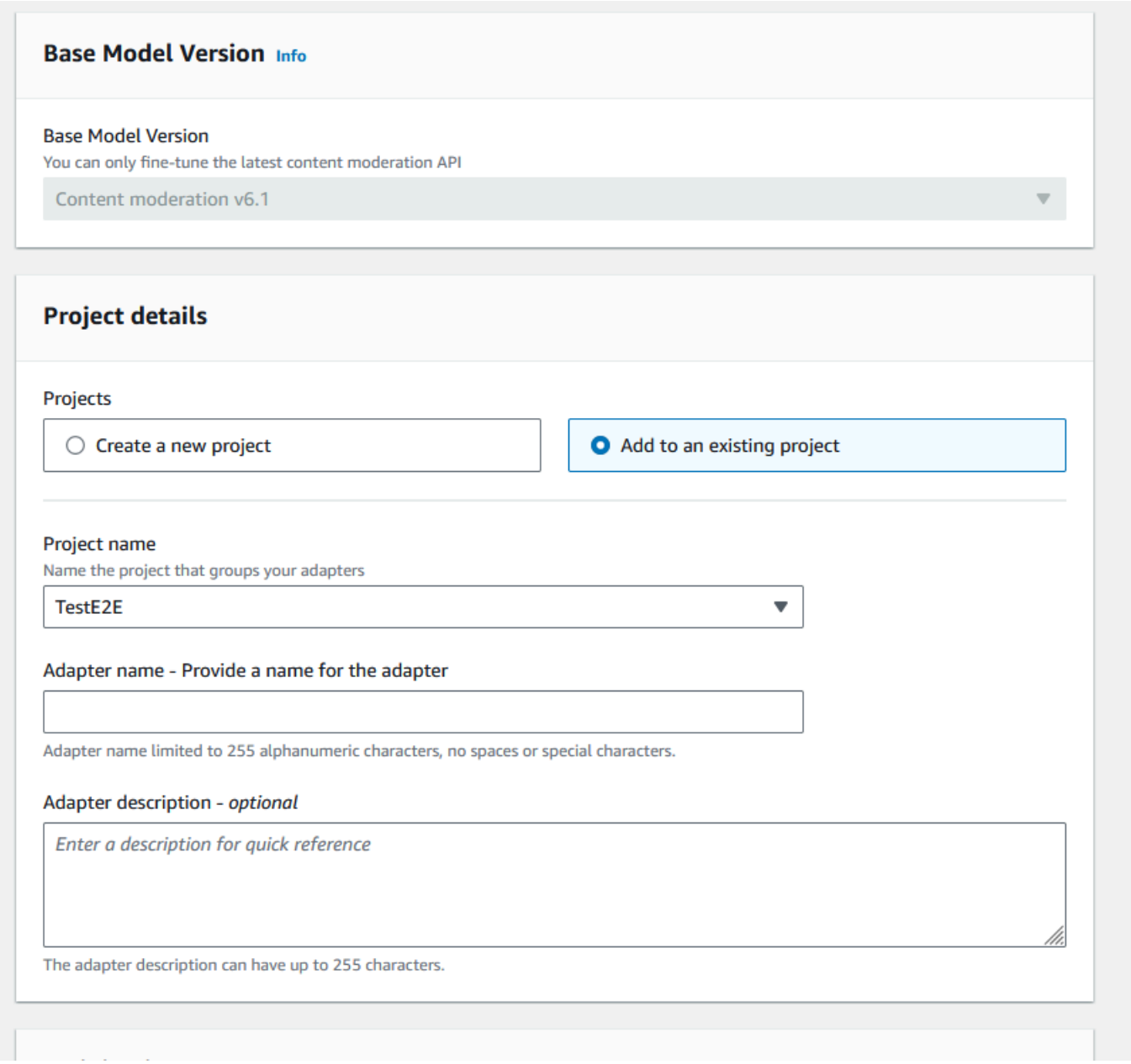
2. レビューするアダプターをこのリストの中から選びます。次の [アダプターの詳細] ページには、アダプターのさまざまなメトリクスが表示されます。



3. [しきい値] パネルでは、イメージにラベルを割り当てるときにアダプターに必要になる最小信頼度しきい値を変更できます。信頼度しきい値を変更すると、フラグ付きのインスタンスとフラグなしのインスタンスの数が変わります。選択したカテゴリのメトリクスは、ラベルカテゴリでフィルタリングして確認することもできます。選択したしきい値を設定します。
4. [アダプターのパフォーマンス] のパネルでメトリクスを調べることにより、テストデータに基づきアダプターのパフォーマンスを評価できます。これらのメトリクスは、アダプターの抽出を、テストセットの「グラウンドトゥルース」の注釈と比較することにより、算出されます。

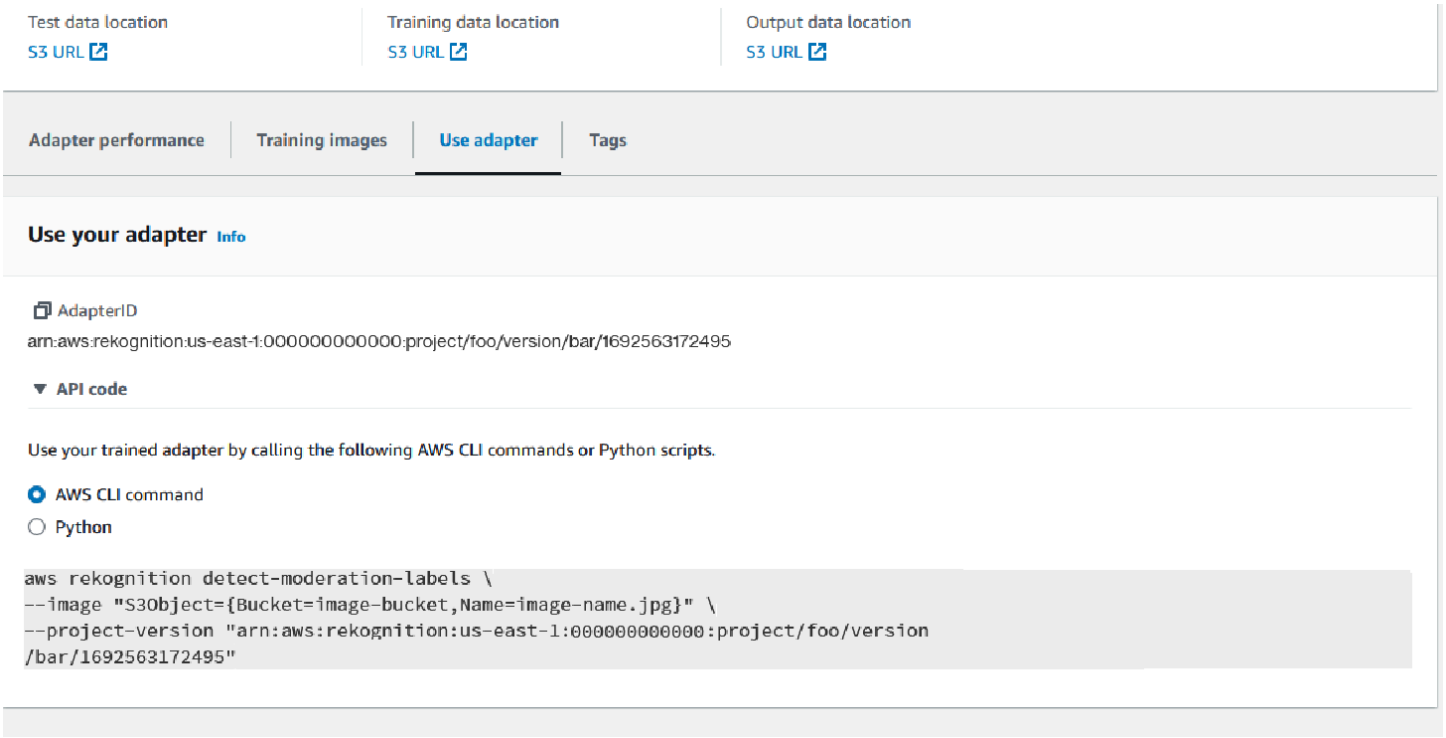
[アダプターのパフォーマンス] パネルには、作成したアダプターの偽陽性の改善と偽陰性の改善が表示されます。[ラベルごとのパフォーマンス] タブでは、各ラベルカテゴリのアダプターと、ベースモデルのパフォーマンスを比較できます。ベースモデルとアダプターの両方による偽陽性予測および偽陰性予測の数が、ラベルカテゴリ別に階層化されて表示されます。これらのメトリクスをレビューすることで、改善が必要な場所を特定できます。これらのメトリクスの詳細については、「アダプターの評価と改善」を参照してください。

パフォーマンスを改善するには、より多くのトレーニングイメージを収集し、プロジェクト内で新しいアダプターを作成します。カスタムモデレーションのランディングページに戻りプロジェクトに新しいアダプターを作成するだけで、アダプターのトレーニング対象となるトレーニングイメージがさらに増えます。今度は、[新規プロジェクトを作成] ではなく [既存のプロジェクトオプションに追加] を選択し、[プロジェクト名] ドロップダウンメニューから、新しいアダプターの作成先となるプロジェクトを選択します。前回と同じように、イメージに注釈を付けるか、注釈付きのマニフェストファイルを提供します。



アダプターの使用

アダプターを作成したら、そのアダプターを DetectModerationLabels などのサポートされている Rekognition オペレーションに提供できます。アダプターで推論を実行するために使用できるコードサンプルを表示するには、「アダプターの使用」タブを選択します。ここでは、CLI AWS と Python の両方のコードサンプルを確認できます。また、アダプターを作成したオペレーションに関する、ドキュメントの各セクションを参照し、その他のコードサンプル、セットアップ手順、サンプル JSON を確認することもできます。



アダプターとプロジェクトの削除

アダプターを個別に削除することも、プロジェクトを削除することもできます。プロジェクト自体を削除するときは、先に、それに関連付けられたアダプターを削除する必要があります。

1. プロジェクトに関連付けられたアダプターを削除するには、そのアダプターを選択し、[削除] をクリックします。
2. プロジェクトを削除するには、削除するプロジェクトを選択して [削除] をクリックします。

アダプターの評価と改善

アダプターをトレーニングしたらその都度、Rekognition コンソールツールのパフォーマンスメトリクスをレビューして、そのアダプターが、必要なパフォーマンス水準をどの程度満たしているかを明らかにします。その後、トレーニングイメージの新しいバッチをアップロードし、プロジェクト内で新しいアダプターをトレーニングすることにより、イメージに対するアダプターの精度をさらに高めることができます。アダプターの改良版を作成できたら、不要になった古いアダプターはコンソールから削除できます。

また、DescribeProjectVersions API オペレーションを使用してメトリクスを取得することもできます。

パフォーマンスメトリクス

トレーニングプロセスを完了してアダプターを作成したら、アダプターが、イメージからどの程度情報を抽出できているかを評価することが重要です。

Rekognition コンソールには、アダプターのパフォーマンスの分析に役立つ、偽陽性の改善と偽陰性の改善という 2 つのメトリクスが用意されています。

これらのメトリクスは、コンソールのアダプターの部分にある [アダプターのパフォーマンス] タブを選択すればどのアダプターでも確認できます。[アダプターのパフォーマンス] パネルには、作成したアダプターの偽陽性の改善と偽陰性の改善が表示されます。

偽陽性の改善では、アダプターによる偽陽性の認識が、ベースモデルに比べどの程度改善しているかを測定します。偽陽性の改善値が 25% であれば、そのアダプターが、テストデータセットでの偽陽性の認識率を 25% 高めたことを意味します。

偽陰性の改善では、アダプターによる偽陰性の認識が、ベースモデルに比べどの程度改善しているかを測定します。偽陰性の改善値が 25% であれば、そのアダプターが、テストデータセットでの偽陰性の認識率を 25% 高めたことを意味します。

[ラベルごとのパフォーマンス] タブでは、各ラベルカテゴリのアダプターと、ベースモデルのパフォーマンスを比較できます。ベースモデルとアダプターの両方による偽陽性予測および偽陰性予測の数が、ラベルカテゴリ別に階層化されて表示されます。これらのメトリクスをレビューすることで、改善が必要な場所を特定できます。

例えば、「アルコール」のラベルカテゴリの、ベースモデルの偽陰性率が 15 で、アダプターの偽陰性率が 15 以上である場合、新しいアダプターを作成するときは、「アルコール」のラベルを含むイメージをさらに追加する必要があることがわかります。

Rekognition API オペレーションを使用する場合、DescribeProjectVersions オペレーションを呼び出すと F1-Score のメトリクスが返されます。

モデルの改善

アダプターのデプロイは反復的なプロセスです。つまり、目標とする精度に達するにはアダプターを複数回トレーニングする必要があります。アダプターを作成してトレーニングしたら、さまざまなタイプのラベルでそのアダプターのパフォーマンスをテストし、評価する必要があります。

アダプターの精度が不十分な領域がある場合は、それらのイメージの新しい例を追加してそれらのラベルに対するアダプターのパフォーマンスを高めます。問題があるケースを反映したさまざまな例

を、アダプターに追加するようにします。アダプターに典型的な、多様なイメージを提供することにより、実社会のさまざまな例に対応できるようになります。

トレーニングセットに新しいイメージを追加したら、アダプターを再トレーニングし、テストセットとラベルを使って再評価します。アダプターが必要なパフォーマンスレベルに達するまでこのプロセスを繰り返します。より忠実な典型的イメージや注釈を提供すれば、偽陽性スコアと偽陰性スコアはトレーニングを繰り返すうちに徐々に改善されていきます。

マニフェストファイルの形式

以下のセクションでは、入力、出力、評価ファイルの、マニフェストファイル形式の例について説明します。

入力マニフェスト

マニフェストファイルは JSON 行で区切られたファイルであり、各行には、1 つのイメージに関する情報を含んだ JSON が含まれています。

入力マニフェストの各エントリには、Amazon S3 バケット内のイメージへのパスを含む `source-ref` フィールドと、カスタムモデレーションの場合は、グラウンド注釈の付いた `content-moderation-groundtruth` フィールドが含まれている必要があります。1 つのデータセット内のイメージはすべて、同じバケットに入れるようにします。この構造は、トレーニングマニフェストファイルとテストマニフェストファイルの両方に共通です。

カスタムモデレーションの `CreateProjectVersion` オペレーションでは、入力マニフェストで提供される情報を使用してアダプターをトレーニングします。

以下の例は、安全でないクラスが 1 つ含まれる 1 つのイメージの、マニフェストファイルに含まれる 1 行です。

```
{
   "source-ref": "s3://foo/bar/1.jpg",
   "content-moderation-groundtruth": {
        "ModerationLabels": [
            {
                "Name": "Rude Gesture"
            }
        ]
   }
}
```

次の例は、複数の安全でないクラス (特に「ヌード」や「失礼なジェスチャー」) を含む、1 つの安全でないイメージの、マニフェストファイルに含まれる 1 行です。

```
{
   "source-ref": "s3://foo/bar/1.jpg",
   "content-moderation-groundtruth": {
        "ModerationLabels": [
            {
                "Name": "Rude Gesture"
            },
            {
                "Name": "Nudity"
            }
        ]
   }
}
```

以下の例は、安全でないクラスを含まない 1 つのイメージの、マニフェストファイルに含まれる 1 行です。

```
{
   "source-ref": "s3://foo/bar/1.jpg",
   "content-moderation-groundtruth": {
        "ModerationLabels": []
   }
}
```

サポートされているラベルの完全なリストについては、「Moderating content」を参照してください。

出力マニフェスト

トレーニングジョブが完了すると、出力マニフェストファイルが返されます。出力マニフェストファイルは JSON 行で区切られたファイルであり、各行には、1 つのイメージに関する情報を含んだ JSON が含まれています。OutputManifest への Amazon S3 パスは、`DescribeProjectVersion` レスポンスから取得できます。

- `TrainingDataResult.Output.Assets[0].GroundTruthManifest.S3Object`、トレーニングデータセット用
- `TestingDataResult.Output.Assets[0].GroundTruthManifest.S3Object`、テストデータセット用

次の情報が、出力マニフェストの各エントリについて返されます。

Key Name	Description
`source-ref`	Reference to an image in s3 that was provided in the input maniefst
`content-moderation-groundtruth`	Ground truth annotations that were provided in the input manifest
`detect-moderation-labels`	Adapter predictions, part of the testing dataset only
`detect-moderation-labels-base-model`	Base model predictions, part of the testing dataset only

アダプターとベースモデルの予測は、DetectModerationLabels レスポンスと同様の形式で、ConfidenceTrehsold 5.0 で返されます。

以下の例は、アダプターとベースモデルの予測の構造を示しています。

```
{
   "ModerationLabels": [
      {
         "Confidence": number,
         "Name": "string",
         "ParentName": "string"
      }
   ],
   "ModerationModelVersion": "string",
   "ProjectVersion": "string"
}
```

返されるラベルの完全なリストについては、「Moderating content」を参照してください。

評価結果マニフェスト

トレーニングジョブが完了すると、評価結果マニフェストファイルが返されます。評価結果マニフェストはトレーニングジョブによって出力される JSON ファイルであり、アダプターがテストデータに対してどの程度うまく機能したかを示す情報が含まれています。

評価結果マニフェストへの Amazon S3 のパスは、DescribeProejctVersion レスポンスの `EvaluationResult.Summary.S3Object` フィールドから取得できます。

次の例は、評価結果マニフェストの構造を示したものです。

```
{
    "AggregatedEvaluationResults": {
       "F1Score": number
    },

    "EvaluationDetails": {
        "EvaluationEndTimestamp": "datetime",
        "Labels": [
            "string"
        ],
        "NumberOfTestingImages": number,
        "NumberOfTrainingImages": number,
        "ProjectVersionArn": "string"
    },

    "ContentModeration": {
        "InputConfidenceThresholdEvalResults": {
            "ConfidenceThreshold": float,
            "AggregatedEvaluationResults": {
                "BaseModel": {
                    "TruePositive": int,
                    "TrueNegative": int,
                    "FalsePositive": int,
                    "FalseNegative": int
                },
                "Adapter": {
                    "TruePositive": int,
                    "TrueNegative": int,
                    "FalsePositive": int,
                    "FalseNegative": int
                }
            },
```

```
            "LabelEvaluationResults": [
                {
                    "Label": "string",
                    "BaseModel": {
                        "TruePositive": int,
                        "TrueNegative": int,
                        "FalsePositive": int,
                        "FalseNegative": int
                    },
                    "Adapter": {
                        "TruePositive": int,
                        "TrueNegative": int,
                        "FalsePositive": int,
                        "FalseNegative": int
                    }
                }
            ]
        }
        "AllConfidenceThresholdsEvalResults": [
            {
                "ConfidenceThreshold": float,
                "AggregatedEvaluationResults": {
                    "BaseModel": {
                        "TruePositive": int,
                        "TrueNegative": int,
                        "FalsePositive": int,
                        "FalseNegative": int
                    },
                    "Adapter": {
                        "TruePositive": int,
                        "TrueNegative": int,
                        "FalsePositive": int,
                        "FalseNegative": int
                    }
                },
                "LabelEvaluationResults": [
                    {
                       "Label": "string",
                        "BaseModel": {
                            "TruePositive": int,
                            "TrueNegative": int,
                            "FalsePositive": int,
                            "FalseNegative": int
                        },
```

```
                        "Adapter": {
                            "TruePositive": int,
                            "TrueNegative": int,
                            "FalsePositive": int,
                            "FalseNegative": int
                        }
                    }
                ]
            }
        ]
    }
}
```

評価マニフェストファイルには以下が含まれます。

- F1Score により定義された集計された結果。
- ProjectVersionArn、トレーニングイメージの数、テストイメージの数、アダプターのトレーニングに使用されたラベルなど、評価ジョブの詳細。
- ベースモデルとアダプターのパフォーマンスの両方に関する TruePositive、TrueNegative、FalsePositive、FalseNegative の集計結果。
- ベースモデルとアダプターのパフォーマンスの両方に関する、入力信頼度しきい値で計算された TruePositive、TrueNegative、FalsePositive、FalseNegative のラベルごとの結果。
- ベースモデルとアダプターのパフォーマンスの両方に関する、入力信頼度しきい値で計算された TruePositive、TrueNegative、FalsePositive、FalseNegative の集計結果およびラベルごとの結果。信頼度しきい値は 5 ~ 100 の範囲の 5 の倍数。

トレーニングアダプターのベストプラクティス

アダプターを作成、トレーニング、使用するときは、次のベストプラクティスに従うことが推奨されます。

1. サンプルイメージのデータでは、顧客が抑えようとしている典型的な工ラーを捉えている必要があります。モデルが、視覚的に類似したイメージで繰り返しミスを犯している場合は、そうしたイメージをトレーニング用に多数用意します。
2. モデルが特定のモデレーションラベルでミスを犯したイメージだけを用意するのではなく、そのモデレーションラベルでミスを犯していないイメージも必ず用意します。

3. トレーニングには最低でも 50 個の偽陰性サンプルまたは 20 個の偽陽性サンプルを用意します。また、テストには最低でも 20 個のサンプルを用意します。ただし、アダプターのパフォーマンスを高めるため、注釈付きのイメージをできるだけ多く提供します。
4. すべてのイメージに関して、重要なラベルのすべてに注釈を付けます。あるイメージで、特定のラベルの出現箇所に注釈を付ける必要があると判断した場合は、他のすべてのイメージでこのラベルの出現箇所に注釈を付けます。
5. サンプルイメージのデータには、ラベルのバリエーションを、本番稼働環境で分析されるイメージの典型的な例を中心にできるだけ多く含める必要があります。

AutoUpdate のアクセス権限のセットアップ

Rekognition はカスタムアダプターの AutoUpdate 機能をサポートしています。したがって、プロジェクトで AutoUpdate フラグが有効になっているときは、自動再トレーニングがベスト工フォートで試行されます。これらの自動更新には、トレーニング/テストデータセットにアクセスするためのアクセス許可と、カスタマーアダプターをトレーニングする AWS KMS キーが必要です。これらのアクセス権限は、以下の手順で付与できます。

Amazon S3 バケット許可

デフォルトでは、すべての Amazon S3 バケットとオブジェクトはプライベートです。バケットを作成した AWS アカウントであるリソース所有者のみが、バケットとそれに含まれるオブジェクトにアクセスできます。ただし、リソース所有者は、バケットポリシーを記述することで他のリソースおよびユーザーにアクセス権限を付与できます。

カスタムアダプターのトレーニングで、入力データセットのソースおよびトレーニング結果の宛先として使用する Amazon S3 バケットを作成または変更する場合は、バケットポリシーをさらに変更する必要があります。Amazon S3 バケットの読み取りまたは書き込みを行うには、Rekognition に次のアクセス権限が必要になります。

Rekognition に必要な Amazon S3 ポリシー

Rekognition には、以下の属性を持つアクセス権限ポリシーが必要です。

- ステートメント SID
- バケット名
- Rekognition のサービスプリンシパル名
- Rekognition に必要なリソース、バケットとそのすべてのコンテンツ

- Rekognition が取るべき必要なアクション

次のポリシーは、自動再トレーニング中に Rekognition が Amazon S3 バケットにアクセスすることを許可します。

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "AllowRekognitionAutoUpdateActions",
            "Principal": {
                "Service": "rekognition.amazonaws.com"
            },
            "Action": [
                "s3:ListBucket",
                "s3:GetObject",
                "s3:PutObject",
                "s3:HeadObject",
                "s3:HeadBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}
```

このガイドに従うことで上記のバケットポリシーを S3 バケットに追加できます。

バケットポリシーの詳細については、こちらを参照してください。

AWS KMS 主要なアクセス許可

Rekognition を使用すると、カスタムアダプターのトレーニング中に、オプションの KmsKeyId を指定できます。指定すると、Rekognition は、このキーを使って、モデルトレーニングのためにサービスにコピーされたトレーニングおよびテストイメージを暗号化します。このキーは、出力の Amazon S3 バケット (OutputConfig) に書き込まれたトレーニング結果とマニフェストファイルを暗号化するためにも使用されます。

カスタムアダプタートレーニングへの入力として KMS キーを指定することを選択すると (つまり `Rekognition:CreateProjectVersion`)、Rekognition サービスプリンシパルが今後この

キーを自動再トレーニングに使用できるよう、KMS キーポリシーをさらに変更する必要があります。Rekognition には次のアクセス権限が必要です。

Rekognition に必要な AWS KMS キーポリシー

Amazon Rekognition には、以下の属性を持つアクセス権限ポリシーが必要です。

- ステートメント SID
- Amazon Rekognition のサービスプリンシパル名
- Amazon Rekognition が取るべき必要なアクション

次のキーポリシーは、自動再トレーニング中に Amazon Rekognition が Amazon KMS キーにアクセスすることを許可します。

```
{
  "Version": "2023-10-06",
  "Statement": [
    {
      "Sid": "KeyPermissions",
      "Effect": "Allow",
      "Principal": {
            "Service": "rekognition.amazonaws.com"
        },
      "Action": [
        "kms:DescribeKey",
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ]
      "Resource": "*"
    }
  ]
}
```

このガイドに従って、上記の AWS KMS ポリシーを AWS KMS キーに追加できます。

AWS KMS ポリシーの詳細については、こちらを参照してください。

AWS Rekognition のヘルスダッシュボード通知

AWS Health Dashboard は、Rekognition からの通知をサポートします。これらの通知は、アプリケーションに影響を与える可能性のある Rekognition モデルの、予定されている変更についての告知

と修正ガイダンス提供します。現在利用できるのは、Rekognition コンテンツモデレーション機能に固有のイベントのみです。

AWS Health Dashboard は AWS Health サービスの一部です。セットアップは一切必要なく、アカウントで認証されるすべてのユーザーが表示できます。詳細については、「Getting started with the AWS Health Dashboard」を参照してください。

次のような通知を受信した場合は、アクションを実行するためのアラームとして処理する必要があります。

通知例: Rekognition コンテンツモデレーションで新しいモデルバージョンの提供が開始されました。

Rekognition はAWS_MODERATION_MODEL_VERSION_UPDATE_NOTIFICATIONイベントを AWS Health Dashboard に発行して、モデレーションモデルの新しいバージョンがリリースされたことを示します。このイベントは、DetectModerationLabels API と、この API を備えたアダプターを使用している場合、重要です。新しいモデルはユースケースによっては品質に影響する可能性があり、最終的に、以前のモデルバージョンから置き換わります。このアラートが表示されたら、モデルの品質を検証して、モデルの更新スケジュールに注意を払うことが推奨されます。

モデルバージョンの更新通知を受け取った場合は、アクションを実行するためのアラームとして取り扱います。アダプターを使用しない場合は、更新したモデルの品質を、既存のユースケースに基づいて評価する必要があります。アダプターを使用する場合は、更新したモデルで新しいアダプターをトレーニングし、その品質を評価する必要があります。自動トレーニングを設定している場合、新しいアダプターは自動的にトレーニングされ、ユーザーはその品質を評価できます。

```
{
   "version": "0",
    "id": "id-number",
    "detail-type": "AWS Health Event",
    "source": "aws.health",
    "account": "123456789012",
    "time": "2023-10-06T06:27:57Z",
    "region": "region",
    "resources": [],
    "detail": {
        "eventArn": "arn:aws:health:us-east-1::event/
AWS_MODERATION_MODEL_UPDATE_NOTIFICATION_event-number",
        "service": "Rekognition",
        "eventTypeCode": "AWS_MODERATION_MODEL_VERSION_UPDATE_NOTIFICATION",
```

```
        "eventScopeCode": "ACCOUNT_SPECIFIC",
        "communicationId": "communication-id-number",
        "eventTypeCategory": "scheduledChange",
        "startTime": "Fri, 05 Apr 2023 12:00:00 GMT",
        "lastUpdatedTime": "Fri, 05 Apr 2023 12:00:00 GMT",
        "statusCode": "open",
        "eventRegion": "us-east-1",
        "eventDescription": [
            {
                "language": "en_US",
                "latestDescription": "A new model version is available for Rekognition
 Content Moderation."
            }
        ]
    }
}
```

EventBridge を使用して Health イベントを検出して対応するには、「Amazon EventBridge による AWS AWS EventBridgeイベントのモニタリング」を参照してください。

Amazon Augmented AI による不適切なコンテンツの確認

Amazon Augmented AI (Amazon A2I) を使用すると、人による機械学習予測のレビューに必要なワークフローを作成できます。

Amazon Rekognition は Amazon A2I と直接統合されているため、安全でないイメージを検出するユースケースで、人によるレビューを容易に実装できます。Amazon A2I は、イメージモデレーションのための人によるレビューワークフローを提供します。これにより、Amazon Rekognition からの予測を容易にレビューできます。ユースケースの信頼しきい値を定義し、時間の経過とともに調整できます。Amazon A2I では、自社組織または Amazon Mechanical Turk 内のレビューアーのプールを使用できます。AWS により品質、セキュリティ手順の順守について、AWS によりあらかじめスクリーニングされた労働力ベンダーを使用することもできます。

次のステップでは、Amazon Rekognition で Amazon A2I を 設定する方法について説明します。まず、Amazon A2I で人によるレビューをトリガーする条件を含むフロー定義を作成します。次に、フロー定義の Amazon リソースネーム (ARN) を Amazon Rekognition `DetectModerationLabel` オペレーションに渡します。`DetectModerationLabel` のレスポンスでは、人によるレビューが必要かどうかを確認できます。人によるレビューの結果は、フロー定義によって設定された Amazon S3 バケットで利用できます。

Amazon Rekognition で Amazon A2I を使用する方法のend-to-endのデモンストレーションを確認するには、「Amazon Amazon SageMaker AI デベロッパーガイド」の以下のチュートリアルのいずれかを参照してください。

- デモ: Amazon A2I コンソールでスタート
- デモ: Amazon A2I API を使用してスタート

  API を使用してスタートするには、Jupyter ノートブックの例を実行することもできます。「Amazon A2I Jupyter Notebook で SageMaker ノートブックインスタンスを使用する」を参照して、SageMaker AI ノートブックインスタンスで Amazon Rekognition [Example] とのノートブック Amazon Augmented AI (Amazon A2I) 統合を使用します。

Amazon A2I での DetectModerationLabels の実行

> **Note**
>
> 同じ AWS リージョン にすべての Amazon A2I リソース と Amazon Rekognition リソースを作成します。

1. SageMaker AI ドキュメントの「Amazon Augmented AI の開始方法」に記載されている前提条件を完了します。

   さらに、SageMaker AI ドキュメントの「Amazon Augmented AI のアクセス許可とセキュリティ」ページにあるように、IAM アクセス許可を設定することを忘れないでください。
2. SageMaker AI ドキュメント」の「人間によるレビューワークフローの作成」の手順に従います。

   人によるレビューワークフローでは、イメージの処理を管理します。これには、人によるレビュー、イメージの送信先となる作業チーム、作業チームが使用する UI テンプレート、および作業チームの結果が送信される Amazon S3 バケットをトリガーする条件が含まれます。

   `CreateFlowDefinition` の呼び出し中に、`HumanLoopRequestSource` を「AWS/Rekognition/DetectModerationLabels/Image/V3」に設定する必要があります。その後、人によるレビューをトリガーする条件をどのように設定するかを決定する必要があります。

   Amazon Rekognition では `ConditionType` 用に、`ModerationLabelConfidenceCheck` および `Sampling` の 2 つのオプションがあります。

ModerationLabelConfidenceCheck は、モデレーションラベルの信頼度が一定範囲内にある場合にヒューマンループを作成します。最後に、Sampling は処理されたドキュメントをランダムな割合で収集し、人によるレビューに送信します。各 ConditionType はそれぞれ異なる ConditionParameters のセットを使用して、人によるレビューに含める結果を設定します。

ModerationLabelConfidenceCheck には、ConditionParameters ModerationLableName があり、人によるレビューが必要なキーを設定します。また、信頼度があり、LessThan、GreaterThan、Equals を使用して、人によるレビューに送信される対象となるパーセンテージの範囲を設定します。Sampling には、RandomSamplingPercentage があり、人によるレビューに送信されるドキュメントの割合を設定します。

次のコード例は、CreateFlowDefinition の部分的な呼び出しです。ラベル「Suggestive」の評価が 98 %未満の場合、およびラベル「Female Swimwear or Underwear」の評価が 95%を超える場合は、人によるレビューにイメージが送信されます。つまり、下着や水着を着用した女性が写っているにもかかわらず、イメージが暗示的であるとは見なされない場合は、人によるレビューを使用してイメージを再確認することができます。

```
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` は `FlowDefinitionArn` を返します。これは、次のステップで `DetectModerationLabels` を呼び出したときに使用します。

詳細については、SageMaker AI API リファレンスの「CreateFlowDefinition」を参照してください。

3. 「不適切なイメージの検出」で説明されているように、`DetectModerationLabels` を呼び出すときに `HumanLoopConfig` パラメータを設定します。`DetectModerationLabels` 設定での `HumanLoopConfig` 呼び出しの例については、ステップ 4 を参照してください。

   a. `HumanLoopConfig` パラメータ内で、`FlowDefinitionArn` をステップ 2 で作成したフロー定義の ARN に設定します。

   b. `HumanLoopName` を設定します。これは、リージョン内で一意で、小文字である必要があります。

   c. (オプション) `DataAttributes` を使用して、Amazon Rekognition に渡されたイメージに個人を特定できる情報がないかどうかを設定できます。情報を Amazon Mechanical Turk に送信するには、このパラメータを設定する必要があります。

4. `DetectModerationLabels` を実行します。

次の例は、 AWS CLI と を使用して `HumanLoopConfig` セット`DetectModerationLabels`で AWS SDK for Python (Boto3) を実行する方法を示しています。

AWS SDK for Python (Boto3)

以下の リクエスト例では、 SDK for Python (Boto3) を使用します。詳細については、AWS SDK for Python (Boto) API リファレンスの「detect_moderation_labels」を参照してください。

```
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

次のリクエスト例では、 CLI AWS を使用しています。詳細については、https://docs.aws.amazon.com/cli/latest/reference/rekognition/detect-moderation-labels.htmlコマンドリファレンスAWS CLI で 検出モデレーションラベル を参照してください。

```
$ 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=["FreeOfPersonallyIdentifiableInforma
 "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"]}}'
```

HumanLoopConfig を有効にDetectModerationLabelsして を実行すると、Amazon Rekognition は SageMaker AI API オペレーション を呼び出しますStartHumanLoop。この

コマンドは、DetectModerationLabels から応答を受け取り、例のフロー定義の条件と照合して確認します。レビューの条件を満たしている場合は、HumanLoopArn を返します。これは、フロー定義で設定した作業チームのメンバーが今イメージを確認できることを意味します。Amazon Augmented AI ランタイムオペレーション DescribeHumanLoop を呼び出すと、ループの結果に関する情報が表示されます。詳細については、Amazon Augmented AI API リファレンス ドキュメント で DescribeHumanLoop を参照してください。。

イメージのレビューが完了すると、フロー定義の出力パスで指定されているバケットで結果を確認できます。また、レビューが完了すると、Amazon A2I から Amazon CloudWatch Events で通知されます。検索するイベントを確認するには、SageMaker AI ドキュメント」のCloudWatch Events」を参照してください。

詳細については、SageMaker AI ドキュメントの「Amazon Augmented AI の開始方法」を参照してください。 SageMaker

テキストの検出

Amazon Rekognition では、イメージやビデオ内のテキストを検出できます。その後、検出されたテキストを機械可読テキストに変換できます。イメージに機械可読テキスト検出を使用して、以下のようなソリューションを実装できます。

- ビジュアル検索。同じテキストが含まれているイメージを取得および表示するソリューションなど。
- コンテンツの洞察。例えば、抽出したビデオフレームから認識されたテキストに含まれるテーマについての洞察を提供する。アプリケーションは、ニュース、スポーツの得点、選手の番号、キャプションなど、認識したテキストから関連コンテンツを検索できます。
- Navigation。視覚障碍者向けの (レストランやショップの名前、道路標識を認識する) 音声対応モバイルアプリを開発するなど。
- 公共の安全と輸送のサポート。例えば、交通監視カメラのイメージから車のナンバープレート番号を検出する。
- フィルタリング。イメージから個人を特定できる情報 (PII) をフィルタリングするなど。

ビデオ内のテキスト検出では、以下のようなソリューションを実装できます。

- 特定のテキストキーワード (ニュース番組のグラフィック上のゲストの名前など) を用いて、クリップをビデオで検索する
- 偶発的なテキスト、冒涜的な表現、スパムを検出して、組織の標準に準拠するようにコンテンツを調整する。
- さらなる処理 (コンテンツの国際化を目的に別の言語のテキストに置き換えるなど) のために、ビデオタイムライン上のすべてのテキストオーバーレイを見つける
- テキストの位置を検索して、それに応じて他のグラフィックを配置できるようにします。

JPEG または PNG 形式のイメージのテキストを検出するには、DetectText オペレーションを使用します。ビデオ内のテキストを非同期的に検出するには、StartTextDetection オペレーションと GetTextDetection オペレーションを使用します。イメージとビデオの両方のテキスト検出オペレーションは、高度に図案化されたフォントも含め、ほとんどのフォントをサポートしています。テキストを検出した後、Amazon Rekognition は、検出した単語とテキストの行の表現を作成し、それらの関係を表示し、テキストがイメージまたはビデオフレームのどこにあるかを示します。

DetectText と GetTextDetection オペレーションは、単語と行を検出します。単語とは、スペースで区切られていない 1 つ以上のスクリプト文字です。DetectText は、イメージ内の単語を最大 100 ワードまで検出できます。GetTextDetection は、ビデオの 1 フレームあたり 100 ワードまで検出できます。

単語とは、スペースで区切られていない、1 個以上のスクリプト文字です。Amazon Rekognition は、英語、アラビア語、ロシア語、ドイツ語、フランス語、イタリア語、ポルトガル語、スペイン語の単語を検出するように設計されています。

行は、等間隔のスペースで区切られた単語の文字列です。1 行は必ずしも完全な文とは限りません (ピリオドは行末を示していません)。例えば、Amazon Rekognition は、運転免許証番号を行として検出します。行の終わりは、その後に整列したテキストがない場合、または単語の長さに対して単語間の間隔に大きなギャップがあるときです。単語間の間隔によっては、Amazon Rekognition は、同じ方向に整列されたテキスト上に、複数の行を検出する場合があります 。文が複数の行にまたがっている場合、このオペレーションは複数の行を返します。

次のイメージについて考えます。



青のボックスは、検出されたテキストと DetectText オペレーションで返されるテキストの位置に関する情報を表します。この例では、Amazon Rekognition は、「IT's」、「MONDAY」、「but」、

「keep」、「Smiling」を単語として検出しています。Amazon Rekognition は、「IT'S」、「MONDAY」、「but keep」、「Smiling」をセリフとして検出します。テキストが検出されるのは、水平軸 から +/- 90 度以内の向き である場合に限ります。

例については、イメージ内のテキストの検出を参照してください。

トピック

- イメージ内のテキストの検出
- 保存済みビデオ内のテキスト検出

イメージ内のテキストの検出

入力イメージとして、イメージのバイト配列 (base64 エンコードされたイメージのバイト) を指定するか、Amazon S3 オブジェクトを指定できます。以下の手順では、JPEG イメージまたは PNG イメージを S3 バケットにアップロードし、そのファイル名を指定します。

イメージ内のテキストを検出するには (API)

1. まだの場合、以下の前提条件を完了します。

   a. `AmazonRekognitionFullAccess` と `AmazonS3ReadOnlyAccess` のアクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

   b. と AWS SDKs をインストール AWS Command Line Interface して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. テキストが含まれているイメージを S3 バケットにアップロードします。

   手順については、Amazon Simple Storage Service ユーザーガイド の「Amazon S3 へのオブジェクトのアップロード」を参照してください。

3. 以下の例を使用して、`DetectText` オペレーションを呼び出します。

   Java

   次のコード例では、イメージ内で検出された行と単語を表示します。

   `amzn-s3-demo-bucket` と `photo` の値は、ステップ 2 で使用した S3 バケット名とイメージ名に置き換えます。

```
//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

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
/**
*  To run this code example, ensure that you perform the Prerequisites as stated
 in the Amazon Rekognition Guide:
*  https://docs.aws.amazon.com/rekognition/latest/dg/video-analyzing-with-
sqs.html
*
* Also, ensure that set up your development environment, including your
 credentials.
*
* For information, see this documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/

//snippet-start:[rekognition.java2.detect_text.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
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.DetectTextRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.DetectTextResponse;
```

```
import software.amazon.awssdk.services.rekognition.model.TextDetection;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
//snippet-end:[rekognition.java2.detect_text.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DetectTextImage {

 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <sourceImage>\n\n" +
         "Where:\n" +
         "   sourceImage - The path to the image that contains text (for
 example, C:\\AWS\\pic1.png). \n\n";

   if (args.length != 1) {
         System.out.println(usage);
         System.exit(1);
     }

     String sourceImage = args[0] ;
     Region region = Region.US_WEST_2;
     RekognitionClient rekClient = RekognitionClient.builder()
         .region(region)
         .credentialsProvider(ProfileCredentialsProvider.create("default"))
         .build();

     detectTextLabels(rekClient, sourceImage );
     rekClient.close();
 }
```

```
// snippet-start:[rekognition.java2.detect_text.main]
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);
    }
}
// snippet-end:[rekognition.java2.detect_text.main]
```

AWS CLI

この AWS CLI コマンドは、 CLI オペレーションの JSON `detect-text` 出力を表示します。

`amzn-s3-demo-bucket` と `Name` の値は、ステップ 2 で使用した S3 バケット名とイメージ名に置き換えます。

`profile_name` の値を自分のデベロッパープロファイル名に置き換えます。

```
aws rekognition detect-text  --image "{"S3Object":{"Bucket":"amzn-s3-demo-
bucket","Name":"image-name"}}" --profile default
```

Windows デバイスで CLI にアクセスする場合は、パーサーエラーの発生に対処するため、一重引用符の代わりに二重引用符を使用し、内側の二重引用符をバックスラッシュ (\) でエスケープします。例として以下を参照してください。

```
aws rekognition detect-text  --image "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-
bucket\",\"Name\":\"image-name\"}}" --profile default
```

Python

以下のコード例では、イメージ内で検出された行と単語を表示します。

amzn-s3-demo-bucket と photo の値は、ステップ 2 で使用した S3 バケット名とイメージ名に置き換えます。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
# 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):

    session = boto3.Session(profile_name='default')
    client = session.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 = 'amzn-s3-demo-bucket'
    photo = 'photo-name'
    text_count = detect_text(photo, bucket)
    print("Text detected: " + str(text_count))

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

.NET

以下のコード例では、イメージ内で検出された行と単語を表示します。

amzn-s3-demo-bucket と photo の値は、ステップ 2 で使用した S3 バケット名とイメージ名に置き換えます。

```
//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 = "amzn-s3-demo-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

次のコード例では、イメージ内で検出された行と単語を表示します。

amzn-s3-demo-bucket と photo の値は、ステップ 2 で使用した S3 バケット名とイメージ名に置き換えます。region の値を、.aws 認証情報に入力されているリージョンに置き換えます。Rekognition セッションを作成する行の profile_name の値を、自分のデベロッパープロファイル名に置き換えます。

```
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 オペレーションリクエスト

DetectText オペレーションでは、入力イメージを base64 でエンコードされたバイト配列、または Amazon S3 バケットに保存されたイメージとして指定します。以下の JSON リクエストの例では、 Amazon S3 バケットからロードしたイメージを表示します。

```
{
    "Image": {
        "S3Object": {
            "Bucket": "amzn-s3-demo-bucket",
```

```
            "Name": "inputtext.jpg"
        }
    }
}
```

フィルター

テキスト領域、サイズ、信頼スコアに基づくフィルタ処理により、テキスト検出出力をさらに柔軟に制御できるようになります。関心領域を使用することで、テキスト検出を関連する領域に簡単に制限できます。例えば、機械のイメージから部品番号を読み取るとき、この領域はプロフィール写真の右上や基準点からの相対位置などです。単語の境界ボックスサイズフィルタを使用すると、ノイズの多いテキストや無関係な小さな背景テキストを回避できます。単語信頼性フィルターを使用すると、ぼやけているか不鮮明であるために信頼できない結果を取り除けます。

フィルターの値については、「DetectTextFilters」を参照してください。

以下のフィルタを使用できます。

- MinConfidence 単語検出の信頼性レベルを設定します。検出の信頼性がこのレベルより低い単語は、結果から除外されます。値は 0 から 100 の間で指定する必要があります。
- MinBoundingBoxWidth 単語境界ボックスの最小幅を設定します。境界ボックスの幅がこの値より小さい単語は、結果から除外されます。値はイメージフレームの幅に対する相対値です。
- MinBoundingBoxHeight 単語境界ボックスの最小の高さを設定します。境界ボックスの高さがこの値より小さい単語は、結果から除外されます。値はイメージフレームの高さに対する相対値です。
- RegionsOfInterest 検出をイメージフレームの特定の領域に制限します。値はフレームの寸法に対する相対値です。領域内に部分的にしか含まれていないテキストの場合、レスポンスは不明となります。

DetectText オペレーションレスポンス

DetectText オペレーションでは、イメージを分析して配列 (TextDetections) を返します。この配列の各要素 (TextDetection) は、イメージ内で検出された行または単語を表します。DetectText により、要素ごとに以下の情報が返されます。

- 検出されたテキスト (DetectedText)
- 単語と行の関係 (Id と ParentId)
- イメージ上のテキストの位置 (Geometry)
- 検出されたテキストの精度を示す Amazon Rekognition の信頼度と境界ボックス (Confidence)

- 検出されたテキストのタイプ (Type)

検出されたテキスト

各 TextDetection 要素には、DetectedText フィールドで認識されたテキスト (単語または行) が含まれます。単語とは、スペースで区切られていない、1 個以上のスクリプト文字です。DetectText は、イメージ内に最大 100 個の単語を検出できます。返されたテキストに含まれる文字によっては、単語が認識できない場合があります。例えば、Cat の代わりに C@t が返される場合があります。TextDetection 要素がテキスト行または単語のいずれであるかを確認するには、Type フィールドを使用します。

各 TextDetection 要素には、検出されたテキストおよびそのテキストを囲む境界ボックスの精度を示す Amazon Rekognition の信頼度 (%) が含まれます。

単語と行の関係

TextDetection 要素ごとに ID フィールド (Id) があります。Id は、行内での単語の位置を示します。要素が単語である場合、親識別子フィールド (ParentId) は単語が検出された行を識別します。行の ParentId は null です。例えば、例のイメージで「but keep」行の Id 値と ParentId 値は以下のとおりです。

テキスト	ID	親 ID
but keep	3	
but	8	3
keep	9	3

イメージ上のテキストの位置

イメージ上で認識されたテキストの位置を確認するには、DetectText から返される境界ボックス (ジオメトリ) 情報を使用します。Geometry オブジェクトには、検出された行と単語に関する次の 2 種類の境界ボックス情報が含まれます。

- BoundingBox オブジェクト内の座標軸に平行な大まかな長方形のアウトライン
- ポイント配列の複数の X 座標と Y 座標で構成された精細な多角形

境界ボックスと多角形の座標は、ソースイメージ上のテキストの位置を示します。座標値は、イメージサイズ全体の比率です。詳細については、「BoundingBox」を参照してください。

次の DetectText オペレーションからの JSON レスポンスは、次のイメージで検出された単語と行を示しています。




```
{
 'TextDetections': [{'Confidence': 99.35693359375,
                     'DetectedText': "IT'S",
                     'Geometry': {'BoundingBox': {'Height': 0.09988046437501907,
                                                  'Left': 0.6684935688972473,
                                                  'Top': 0.18226495385169983,
                                                  'Width': 0.1461552083492279},
                                  'Polygon': [{'X': 0.6684935688972473,
                                               'Y': 0.1838926374912262},
                                              {'X': 0.8141663074493408,
                                               'Y': 0.18226495385169983},
                                              {'X': 0.8146487474441528,
                                               'Y': 0.28051772713661194},
                                              {'X': 0.6689760088920593,
                                               'Y': 0.2821454107761383}]},
                     'Id': 0,
```

```
                    'Type': 'LINE'},
                   {'Confidence': 99.6207275390625,
                    'DetectedText': 'MONDAY',
                    'Geometry': {'BoundingBox': {'Height': 0.11442459374666214,
                                                 'Left': 0.5566731691360474,
                                                 'Top': 0.3525116443634033,
                                                 'Width': 0.39574965834617615},
                                 'Polygon': [{'X': 0.5566731691360474,
                                              'Y': 0.353712260723114},
                                             {'X': 0.9522717595100403,
                                              'Y': 0.3525116443634033},
                                             {'X': 0.9524227976799011,
                                              'Y': 0.4657355844974518},
                                             {'X': 0.5568241477012634,
                                              'Y': 0.46693623065948486}]},
                    'Id': 1,
                    'Type': 'LINE'},
                   {'Confidence': 99.6160888671875,
                    'DetectedText': 'but keep',
                    'Geometry': {'BoundingBox': {'Height': 0.08314694464206696,
                                                 'Left': 0.6398131847381592,
                                                 'Top': 0.5267938375473022,
                                                 'Width': 0.2021435648202896},
                                 'Polygon': [{'X': 0.640289306640625,
                                              'Y': 0.5267938375473022},
                                             {'X': 0.8419567942619324,
                                              'Y': 0.5295097827911377},
                                             {'X': 0.8414806723594666,
                                              'Y': 0.609940767288208},
                                             {'X': 0.6398131847381592,
                                              'Y': 0.6072247624397278}]},
                    'Id': 2,
                    'Type': 'LINE'},
                   {'Confidence': 88.95134735107422,
                    'DetectedText': 'Smiling',
                    'Geometry': {'BoundingBox': {'Height': 0.4326171875,
                                                 'Left': 0.46289217472076416,
                                                 'Top': 0.5634765625,
                                                 'Width': 0.5371078252792358},
                                 'Polygon': [{'X': 0.46289217472076416,
                                              'Y': 0.5634765625},
                                             {'X': 1.0, 'Y': 0.5634765625},
                                             {'X': 1.0, 'Y': 0.99609375},
                                             {'X': 0.46289217472076416,
```

```
                                    'Y': 0.99609375}]},
                  'Id': 3,
                  'Type': 'LINE'},
                 {'Confidence': 99.35693359375,
                  'DetectedText': "IT'S",
                  'Geometry': {'BoundingBox': {'Height': 0.09988046437501907,
                                               'Left': 0.6684935688972473,
                                               'Top': 0.18226495385169983,
                                               'Width': 0.1461552083492279},
                               'Polygon': [{'X': 0.6684935688972473,
                                            'Y': 0.1838926374912262},
                                           {'X': 0.8141663074493408,
                                            'Y': 0.18226495385169983},
                                           {'X': 0.8146487474441528,
                                            'Y': 0.28051772713661194},
                                           {'X': 0.6689760088920593,
                                            'Y': 0.2821454107761383}]},
                  'Id': 4,
                  'ParentId': 0,
                  'Type': 'WORD'},
                 {'Confidence': 99.6207275390625,
                  'DetectedText': 'MONDAY',
                  'Geometry': {'BoundingBox': {'Height': 0.11442466825246811,
                                               'Left': 0.5566731691360474,
                                               'Top': 0.35251158475875854,
                                               'Width': 0.39574965834617615},
                               'Polygon': [{'X': 0.5566731691360474,
                                            'Y': 0.3537122905254364},
                                           {'X': 0.9522718787193298,
                                            'Y': 0.35251158475875854},
                                           {'X': 0.9524227976799011,
                                            'Y': 0.4657355546951294},
                                           {'X': 0.5568241477012634,
                                            'Y': 0.46693626046180725}]},
                  'Id': 5,
                  'ParentId': 1,
                  'Type': 'WORD'},
                 {'Confidence': 99.96778869628906,
                  'DetectedText': 'but',
                  'Geometry': {'BoundingBox': {'Height': 0.0625,
                                               'Left': 0.6402802467346191,
                                               'Top': 0.5283203125,
                                               'Width': 0.08027780801057816},
                               'Polygon': [{'X': 0.6402802467346191,
```

```
                                        'Y': 0.5283203125},
                                       {'X': 0.7205580472946167,
                                        'Y': 0.5283203125},
                                       {'X': 0.7205580472946167,
                                        'Y': 0.5908203125},
                                       {'X': 0.6402802467346191,
                                        'Y': 0.5908203125}]},
                    'Id': 6,
                    'ParentId': 2,
                    'Type': 'WORD'},
                   {'Confidence': 99.26438903808594,
                    'DetectedText': 'keep',
                    'Geometry': {'BoundingBox': {'Height': 0.0818721204996109,
                                                 'Left': 0.7344760298728943,
                                                 'Top': 0.5280686020851135,
                                                 'Width': 0.10748066753149033},
                                 'Polygon': [{'X': 0.7349520921707153,
                                              'Y': 0.5280686020851135},
                                             {'X': 0.8419566750526428,
                                              'Y': 0.5295097827911377},
                                             {'X': 0.8414806127548218,
                                              'Y': 0.6099407076835632},
                                             {'X': 0.7344760298728943,
                                              'Y': 0.6084995269775391}]},
                    'Id': 7,
                    'ParentId': 2,
                    'Type': 'WORD'},
                   {'Confidence': 88.95134735107422,
                    'DetectedText': 'Smiling',
                    'Geometry': {'BoundingBox': {'Height': 0.4326171875,
                                                 'Left': 0.46289217472076416,
                                                 'Top': 0.5634765625,
                                                 'Width': 0.5371078252792358},
                                 'Polygon': [{'X': 0.46289217472076416,
                                              'Y': 0.5634765625},
                                             {'X': 1.0, 'Y': 0.5634765625},
                                             {'X': 1.0, 'Y': 0.99609375},
                                             {'X': 0.46289217472076416,
                                              'Y': 0.99609375}]},
                    'Id': 8,
                    'ParentId': 3,
                    'Type': 'WORD'}],
 'TextModelVersion': '3.0'}
```

保存済みビデオ内のテキスト検出

保存済みビデオ内の Amazon Rekognition Video のテキスト検出は、非同期オペレーションです。テキストの検出を開始するには、StartTextDetection を呼び出します。Amazon Rekognition Video は、ビデオ分析の完了ステータスを Amazon SNS トピックに発行します。ビデオ分析が成功したら、GetTextDetection を呼び出して分析結果を取得します。ビデオ分析の開始と結果の取得の詳細については、「Amazon Rekognition Video オペレーションを呼び出す」を参照してください。

この手順では、「Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)」のコードを拡張します。Amazon SQS キューを使用して、ビデオ分析リクエストの完了ステータスを取得します。

Amazon S3 バケットに保存されたビデオ内のテキストを検出するには (SDK)

1. 「Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)」の手順を実行します。
2. ステップ 1 で作成したクラス `VideoDetect` に以下のコードを追加します。

   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);


}
```

関数 main で、以下の行を置き換えます。

```
        StartLabelDetection(amzn-s3-demo-bucket, video);

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

を:

```
        StartTextDetection(amzn-s3-demo-bucket, video);

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

Java V2

このコードは、 AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例は [こちら] です。

```
//snippet-start:[rekognition.java2.recognize_video_text.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
```

```
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.TextDetectionResult;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_text.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DetectTextVideo {

 private static String startJobId ="";
 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <bucket> <video> <topicArn> <roleArn>\n\n" +
         "Where:\n" +
         "   bucket - The name of the bucket in which the video is located (for
 example, (for example, amzn-s3-demo-bucket). \n\n"+
         "   video - The name of video (for example, people.mp4). \n\n" +
         "   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic. \n\n" +
         "   roleArn - The ARN of the AWS Identity and Access Management (IAM)
 role to use. \n\n" ;

     if (args.length != 4) {
         System.out.println(usage);
```

```
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];

        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
            .region(region)
            .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
            .build();

        NotificationChannel channel = NotificationChannel.builder()
            .snsTopicArn(topicArn)
            .roleArn(roleArn)
            .build();

        startTextLabels(rekClient, channel, bucket, video);
        GetTextResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    // snippet-start:[rekognition.java2.recognize_video_text.main]
    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);
    }
 }
 // snippet-end:[rekognition.java2.recognize_video_text.main]
}
```

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
```

関数 main で、以下の行を置き換えます。

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

を:

```
    analyzer.StartTextDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetTextDetectionResults()
```

CLI

次の AWS CLI コマンドを実行して、ビデオ内のテキストの検出を開始します。

```
 aws rekognition start-text-detection --video "{"S3Object":{"Bucket":"amzn-s3-
demo-bucket","Name":"video-name"}}"\
 --notification-channel "{"SNSTopicArn":"topic-arn","RoleArn":"role-arn"}" \
 --region region-name --profile profile-name
```

以下の値を更新します。

- amzn-s3-demo-bucket と video-name を、ステップ 2 で指定した Amazon S3 バケット名とファイル名に変更します。
- region-name を、使用している AWS リージョンに変更します。

- profile-name の値を自分のデベロッパープロファイル名に置き換えます。
- topic-ARN を、Amazon Rekognition Video の設定 のステップ 3 で作成した Amazon SNS トピックの ARN に変更します。
- role-ARN を、Amazon Rekognition Video の設定 のステップ 7 で作成した IAM サービスロールの ARN に変更します。

Windows デバイスで CLI にアクセスする場合は、パーサーエラーの発生に対処するため、一重引用符の代わりに二重引用符を使用し、内側の二重引用符をバックスラッシュ (\) でエスケープします。次の例を参照してください。

```
aws rekognition start-text-detection --video \
 "{\"S3Object\":{\"Bucket\":\"amzn-s3-demo-bucket\",\"Name\":\"video-name\"}}" \
 --notification-channel "{\"SNSTopicArn\":\"topic-arn\",\"RoleArn\":\"role-arn
\"}" \
 --region region-name --profile profile-name
```

上記のコード例を実行した後、返された jobID をコピーして以下の GetTextDetection コマンドに渡すと、job-id-number が以前に受け取った jobID に置き換わっている結果が得られます。

```
aws rekognition get-text-detection --job-id job-id-number --profile profile-name
```

> **Note**
>
> Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK) 以外のビデオ例をすでに実行している場合、置き換えるコードは異なる可能性があります。

3. コードを実行します。ビデオで検出されたテキストがリストに表示されます。

フィルター

フィルタは、StartTextDetection を呼び出すときに使用できるオプションのリクエストパラメータです。テキスト領域、サイズ、信頼スコアに基づくフィルタ処理により、テキスト検出出力をさらに柔軟に制御できるようになります。関心領域を使用することで、テキスト検出を関連する領域

に簡単に制限できます。例えば、グラフィックスの下部 3 分の 1 の領域や、サッカーゲームのスコアボードを読むための左上隅の領域です。単語の境界ボックスサイズフィルタを使用すると、ノイズの多いテキストや無関係な小さな背景テキストを回避できます。最後に、単語信頼性フィルタを使用すると、ぼやけているか汚れているせいで信頼できない結果を削除できます。

フィルターの値については、「DetectTextFilters」を参照してください。

以下のフィルタを使用できます。

- MinConfidence 単語検出の信頼性レベルを設定します。検出の信頼性がこのレベルより低い単語は、結果から除外されます。値は 0 から 100 の間で指定する必要があります。
- MinBoundingBoxWidth 単語境界ボックスの最小幅を設定します。境界ボックスの幅がこの値より小さい単語は、結果から除外されます。値はビデオフレームの幅に対する相対値です。
- MinBoundingBoxHeight 単語境界ボックスの最小の高さを設定します。境界ボックスの高さがこの値より小さい単語は、結果から除外されます。値はビデオフレームの高さに対する相対値です。
- RegionsOfInterest 検出をフレームの特定の領域に制限します。値はフレームの寸法に対する相対値です。領域内に部分的にしか含まれていないオブジェクトの場合、レスポンスは不明となります。

GetTextDetection レスポンス

GetTextDetection は、ビデオ内で検出されたテキストに関する情報が含まれた配列 (TextDetectionResults) を返します。配列要素 TextDetection は、ビデオで単語や行が検出されるたびに生成されます。配列要素は、ビデオの開始時点からの経過時間 (ミリ秒単位) で並べ替えられます。

以下に示しているのは、GetTextDetection からの JSON レスポンスの一部です。レスポンスで、以下の点に注意してください。

- テキストの情報 – TextDetectionResult 配列要素には、検出されたテキストに関する情報 (TextDetection) と、ビデオ内でテキストが検出された時間 (Timestamp) が含まれます。
- ページング情報 例は 1 ページのテキスト検出情報を示しています。テキスト要素を返す数は、GetTextDetection の MaxResults 入力パラメータで指定できます。MaxResults を超える結果が存在する場合、またはデフォルトの最大値を超える結果がある場合は、GetTextDetection から返されるトークン (NextToken) を使用して次の結果ページを取得できます。詳細については、「Amazon Rekognition Video の分析結果を取得する」を参照してください。

- ビデオ情報 – このレスポンスには、VideoMetadata から返された各情報ページのビデオ形式 (GetTextDetection) に関する情報が含まれます。


```
{
    "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"
}
```

保存されたビデオ内のビデオセグメントの検出

Amazon Rekognition Video は、ブラックフレームやエンドクレジットなど、ビデオの有用なセグメントを特定する API を提供しています。

視聴者は以前にも増して、多くのコンテンツを閲覧するようになっています。特に、オーバーザトップ (OTT) およびビデオオンデマンド (VOD) プラットフォームでは、いつでも、どこでも、どんな画面でも、豊富なコンテンツを選択することができます。コンテンツ量が急増する中、メディア企業はコンテンツの準備と管理に関する課題に直面しています。これは、高品質の視聴体験を提供し、コンテンツの収益化を高めるために不可欠です。今日、企業はトレーニングを受けたスタッフによる大規模なチームを使用して、次のようなタスクを実行しています。

- コンテンツ内のオープニングクレジットとエンドクレジットの位置を見つける。
- サイレントブラックフレームシーケンスなど、広告を挿入するための適切なスポットを選択する
- より良いインデックス作成のため、ビデオをより小さなクリップに分割する。

これらのマニュアルプロセスは高価で時間がかかるため、毎日作成され、ライセンス供与され、アーカイブから取得されるコンテンツのボリュームに合わせてスケールすることができません。

Amazon Rekognition Video を使用すると、機械学習 (ML) を活用したフルマネージドで専用に構築されたビデオセグメント検出 API を使用して、運用メディア分析タスクを自動化することができます。Amazon Rekognition Video セグメント API を使用すると、ラージボリュームのビデオを簡単に分析し、ブラックフレームやショット変更などのマーカーを検出できます。検出ごとに、SMPTE (米国映画テレビ技術者協会) のタイムコードとタイムスタンプ、フレーム数を取得します。ML の経験は必要ありません。

Amazon Rekognition Video は、Amazon Simple Storage Service (Amazon S3) バケットに保存されているビデオを分析します。返される SMPTE タイムコードのフレームは正確です。Amazon Rekognition Video は、検出されたビデオセグメントの正確なフレーム番号を提供し、さまざまなビデオフレームレート形式を自動的に処理します。Amazon Rekognition Video の正確なフレームのメタデータを使用して、特定のタスクを完全に自動化できます。また、トレーニングを受けたオペレーターのレビューのワークロードを大幅に削減して、オペレーターがよりクリエイティブな作業に集中できるようにします。コンテンツの準備、広告の挿入、コンテンツへの [ビンジマーカー] の追加などのタスクを、クラウド内で大規模に実行できます。

料金表の詳細については、「Amazon Rekognition の料金」を参照してください。

Amazon Rekognition Video セグメント検出は、テクニカルキュー 検出と ショット検出 の 2 種類のセグメンテーションタスクをサポートします。

トピック

- テクニカルキュー
- ショット検出
- Amazon Rekognition Video セグメント検出 API について
- Amazon Rekognition セグメント API を利用する
- 例: 保存されたビデオ内のセグメントの検出

テクニカルキュー

テクニカルキュー は、ビデオ内のブラックフレーム、カラーバー、オープニングクレジット、エンドクレジット、スタジオロゴ、および主要な番組コンテンツを識別します。

ブラックフレーム

ビデオには、広告を挿入する合図として、あるいはシーンやオープニングクレジットなどのセグメントの終了時点を区切るために、無音の空のブラックフレームが挿入されることがあります。Amazon Rekognition Video では、ブラックフレームのシーケンスを検出して、広告挿入を自動化したり、VOD のコンテンツをパッケージ化したり、プログラム内のセグメントやシーンを区切ったりすることができます。音声付きのブラックフレーム (フェードアウトやナレーション付きなど) はコンテンツとみなされて、検出されません。

クレジット

Amazon Rekognition Video を使用すると、映画やテレビ番組のオープニングクレジットとエンドクレジットの開始位置と終了位置の正確なフレームを自動的に特定できます。この情報をもとに、ビデオオンデマンド (VOD) アプリケーションで [次のエピソード] または [イントロをスキップ] など、[ビンジマーカー] またはインタラクティブな視聴者向けプロンプトを生成できます。ビデオ内の番組コンテンツの最初と最後のフレームを検出することもできます。Amazon Rekognition Video は、シンプルなローリングクレジットからコンテンツに沿ったより難しいクレジットまで、様々なオープニングとエンドクレジットのスタイルを処理するようにトレーニングされています。

カラーバー

Amazon Rekognition Video は、SMPTE のカラーバーが表示されているセクションの検出を許可します。カラーバーとは、色が正確に調整されているかをブロードキャストモニタリング、プログラム、カメラなどで確認するために、特定のパターンで表示される色のセットのことです。SMPTE カラーバーの詳細については、「SMPTE カラーバー」を参照してください。カラーバーのメタデータは、コンテンツからカラーバーのセグメントを削除するなど、VOD のアプリケーション向けにコンテンツを準備する際に便利です。あるいは、カラーバーがデフォルトの信号としてコンテンツの代わりに継続表示される場合、録画中に放送信号が紛失するといった問題を検出する際にも有用です。

スレート

スレートは、ビデオのセクションで、通常は冒頭付近にあり、エピソード、スタジオ、ビデオフォーマット、オーディオチャンネルなどに関するテキストメタデータを含んでいます。Amazon Rekognition Video は、スレートの開始と終了を識別できるため、最終視聴のためのコンテンツを準備する際に、テキストメタデータを使用したり、スレートを削除したりすることが容易にできます。

スタジオロゴ

スタジオロゴは、番組の制作に携わった制作スタジオのロゴやエンブレムを表示するシーケンスです。Amazon Rekognition Video は、これらのシーケンスを検出し、ユーザーがレビューをしてスタジオを特定できるようにします。

コンテンツ

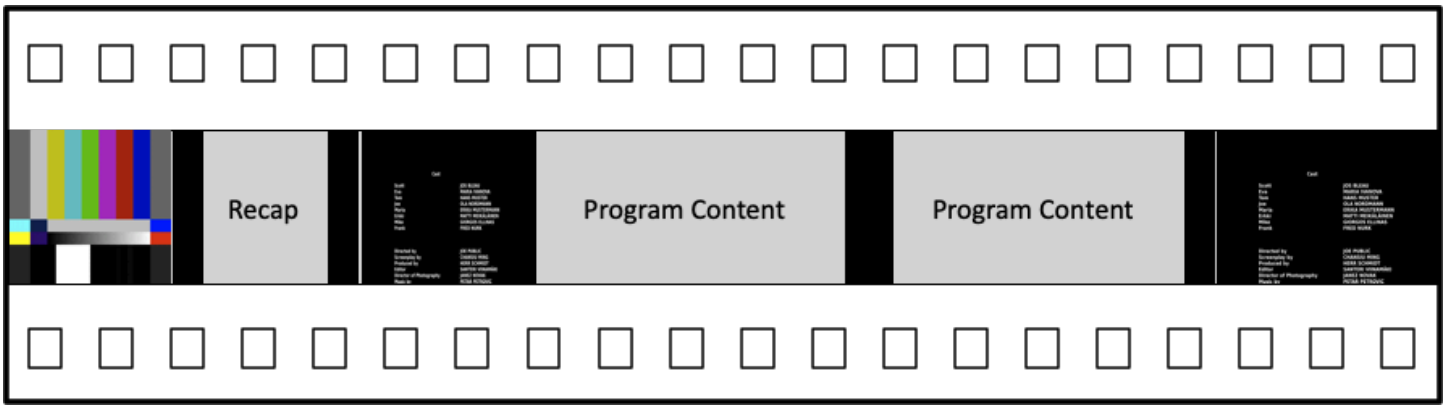
コンテンツとは、テレビ番組や映画などの番組や関連要素を含む部分のことです。ブラックフレーム、クレジット、カラーバー、スレート、スタジオロゴはコンテンツとはみなされません。Amazon Rekognition Video では、ビデオ内の各コンテンツセグメントの開始と終了を検出できるため、番組の実行時間や特定のセグメントを見つけることができます。

コンテンツセグメントには次のようなものがありますが、これに限定されるものではありません:

- 2 つの広告の間のプログラムシーン
- ビデオの冒頭で前回のエピソードを簡単に振り返る
- クレジット後のボーナスコンテンツ
- [テキストレス] コンテンツとは、元々オーバーレイされたテキストが含まれていた番組シーンのうち、他の言語への翻訳をサポートするためにテキストが削除されたものを指します。

Amazon Rekognition Video がすべてのコンテンツセグメントの検出を完了したら、ドメイン知識を適用したり、ヒューマンレビュー用に送信して、各セグメントをさらに分類できます。例えば、常に振り返りから始まる動画を使用する場合、最初のコンテンツセグメントを振り返りに分類できます。

次の図表は、ショーや映画のタイムライン上のテクニカルキューセグメントを示しています。カラーバーとオープニングクレジット、総集編や本編などのコンテンツセグメント、映像全体のブラックフレーム、およびエンドクレジットに注目してください。



ショット検出

ショットとは、1 台のカメラで継続的に撮影された、相互に関係する連続写真の集合のことで、同じ時間および空間内で行われた連続する動きを表現したものです。Amazon Rekognition Video では、各ショットの開始、終了、長さ、およびコンテンツ内のすべてのショットを検出できます。ショットのメタデータは次のようなタスクに使用できます。

- 選択したショットを使用してプロモーションビデオを作成する。
- 視聴者の体験を妨げない場所 (誰かが話しているときのショットの途中など) に広告を挿入する。
- ショット間でトランジションがあるコンテンツを避けるプレビューサムネイルのセットを生成する。

ショット検出は、別のカメラへのハードカットがある正確なフレームでマークされます。あるカメラから別のカメラへのソフトトランジションがある場合、Amazon Rekognition Video はトランジションを省略します。これにより、ショットの開始時間と終了時間に、実際のコンテンツのないセクションが含まれなくなります。

次の図は、フィルムストリップ上のショット検出セグメントを示しています。各ショットは、あるカメラアングルまたは位置から次のカメラアングルまたは位置へのカットによって識別されます。



Amazon Rekognition Video セグメント検出 API について

保存されたビデオをセグメント化するには、非同期 StartSegmentDetection および GetSegmentDetection API オペレーションを使用してセグメント化ジョブを開始し、結果を取得します。セグメント検出は、Amazon S3 バケットに保存されたビデオを受け取り、JSON 出力を返します。`StartSegmentdetection` API リクエストを設定することで、テクニカルキューのみ、ショットの変更のみ、またはその両方を検出することを選択することができます。また、最小予測信頼度のしきい値を設定して、検出されたセグメントをフィルタリングすることもできます。詳細については、「Amazon Rekognition セグメント API を利用する 」を参照してください。サンプルコードについては、「例: 保存されたビデオ内のセグメントの検出」を参照してください。

Amazon Rekognition セグメント API を利用する

保存済み動画内の Amazon Rekognition Video セグメント検出は、Amazon Rekognition Video の非同期オペレーションです。Amazon Rekognition セグメント API は、単一の API コールから分析の種類 (テクニカルキューまたはショット検出) を選択する複合 API です。非同期オペレーションの呼び出しについては、「Amazon Rekognition Video オペレーションを呼び出す」を参照してください。

トピック

- セグメント解析の開始
- セグメント解析結果の取得

セグメント解析の開始

保存されたビデオのセグメント検出を開始するには、StartSegmentDetection を呼び出します。入力パラメータは、他の Amazon Rekognition Video オペレーションと同じで、セグメントタイプの選択

と結果のフィルタリングが追加されています。詳細については、「ビデオ分析のスタート」を参照してください。

以下は、StartSegmentDetection によって渡される JSON の例です。リクエストでは、テクニカルキューとショット検出セグメントの両方が検出されるように指定します。テクニカルキューセグメント (90%) とショット検出セグメント (80%) には、最小検出信頼度として異なるフィルタが要求されます。

```
{
  "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
      }
  }
}
```

セグメントタイプの選択

SegmentTypes 配列入力パラメータを使用して、入力ビデオ内のテクニカルキューやショット検出セグメントを検出します。

- TECHNICAL_CUE ビデオで検出されたテクニカルキュー (ブラックフレーム、カラーバー、オープニングクレジット、エンドクレジット、スタジオロゴ、および主要な番組コンテンツ) の開始、終了、および長さの正確なフレームのタイムスタンプを識別します。たとえば、テクニカルキューを使用してエンドクレジットの開始位置を見つけることができます。詳細については、「テクニカルキュー」を参照してください。

- SHOT ショットの開始、終了、および長さを指定します。たとえば、ショット検出を使用して、ビデオの最終編集の候補ショットを特定できます。詳細については、「ショット検出」を参照してください。

解析結果のフィルタリング

Filters (StartSegmentDetectionFilters) 入力パラメータを使用して、レスポンスで返される最小検出信頼度を指定できます。Filters 内部で ShotFilter (StartShotDetectionFilter) を使用して、検出されたショットをフィルタリングします。テクニカルキューをフィルタリングするには、TechnicalCueFilter (StartTechnicalCueDetectionFilter) を使用します。

サンプルコードについては、「例: 保存されたビデオ内のセグメントの検出」を参照してください。

セグメント解析結果の取得

Amazon Rekognition Video は、ビデオ分析の完了ステータスを Amazon Simple Notification Service トピックに発行します。ビデオ分析が成功したら、GetSegmentDetection を呼び出してビデオ分析の結果を取得します。

GetSegmentDetection リクエストの例を次に示します。JobId は、StartSegmentDetection の呼び出しから返されるジョブ識別子です。他の入力パラメータについての詳細は、「Amazon Rekognition Video の分析結果を取得する」を参照してください。

```
{
    "JobId": "270c1cc5e1d0ea2fbc59d97cb69a72a5495da75851976b14a1784ca90fc180e3",
    "MaxResults": 10,
    "NextToken": "XfXnZKiyMOGDhzBzYUhS5puM+g1IgezqFeYpv/H/+5noP/LmM57FitUAwSQ5D6G4AB/
PNwolrw=="
}
```

GetSegmentDetection は、要求された分析の結果と、保存されたビデオに関する一般的な情報を返します。

一般情報

GetSegmentDection は、次の一般的な情報を返します。

- オーディオ情報 - レスポンスには、AudioMetadata オブジェクトの配列 AudioMetadata のオーディオメタデータが含まれます。複数のオーディオストリームが存在する可能性があります。各 AudioMetadata オブジェクトには、単一のオーディオストリームのメタデータが含まれま

す。AudioMetadata オブジェクトのオーディオ情報には、オーディオコーデック、オーディオチャンネルの数、オーディオストリームの長さ、およびサンプルレートが含まれます。オーディオメタデータは、GetSegmentDetection によって返される情報の各ページに含まれます。

- ビデオ情報 - 現在のところ、Amazon Rekognition Video は、1 つの VideoMetadata オブジェクトを VideoMetadata 配列で返します。このオブジェクトには、Amazon Rekognition Video が分析のために選択した入力ファイル内のビデオストリームに関する情報が含まれます。VideoMetadata オブジェクトには、ビデオコーデック、ビデオ形式、およびその他の情報が含まれます。ビデオのメタデータは、GetSegmentDetection によって返される情報の各ページに含まれます。
- ページング情報 この例では、セグメント情報の 1 ページを示します。要素を返す数は、GetSegmentDetection の MaxResults 入力パラメータで指定できます。結果が MaxResults を超える場合、GetSegmentDetection は次のページの結果を取得するためのトークン (NextToken) を返します。詳細については、「Amazon Rekognition Video の分析結果を取得する」を参照してください。
- リクエスト情報 StartSegmentDetectionの呼び出しで要求される分析のタイプが SelectedSegmentTypes フィールドに返されます。

セグメント

ビデオで検出されたテクニカルキューとショット情報が、SegmentDetection オブジェクトの配列 Segments で返されます。配列は、StartSegmentDetection の SegmentTypes 入力パラメータで指定されたセグメントタイプ (TECHNICAL_CUE または SHOT) でソートされます。各セグメントタイプ内で、配列はタイムスタンプ値でソートされます。各 SegmentDetection オブジェクトには、検出されたセグメントのタイプ (テクニカルキューまたはショット検出) に関する情報と、開始時間、終了時間、セグメントの長さなどの一般情報が含まれます。

時間情報は 3 つの形式で返されます。

- ミリ秒

  ビデオ開始からのミリ秒数。フィールド DurationMillis、StartTimestampMillis、および EndTimestampMillis はミリ秒形式です。
- タイムコード

  Amazon Rekognition Video タイムコードは SMPTE 形式で、ビデオの各フレームに固有のタイムコード値を持っています。形式は hh:mm:ss:frame です。たとえば、01:05:40:07 というタイムコード値は、1 時間、5 分、40 秒、7 フレームとして読み取られます。ドロップフレーム レート

のユースケースは、Amazon Rekognition Video でサポートされています。ドロップレートのタイムコード形式は hh:mm:ss;frame です。フィールド DurationSMPTE、StartTimecodeSMPTE、および EndTimecodeSMPTE はタイムコード形式です。

- フレームカウンタ

  各ビデオセグメントのデュレーションもフレーム数で表されます。フィールド StartFrameNumber は、ビデオセグメントの開始時のフレーム番号を指定し、EndFrameNumber は、ビデオセグメントの終了時のフレーム番号を指定します。DurationFrames は、ビデオセグメント内の総フレーム数を示します。これらの値は、0 から始まるフレームインデックスを使用して計算されます。

SegmentType フィールドを使用して、Amazon Rekognition Video によって返されるセグメントのタイプを決定できます。

- テクニカルキュー - TechnicalCueSegment フィールドは、検出信頼度とテクニカルキューのタイプを含む TechnicalCueSegment オブジェクトです。テクニカルキューの種類は ColorBars、EndCredits、BlackFrames、OpeningCredits、StudioLogo、Slate、および Content です。
- ショット - ShotSegment フィールドは、ビデオ内のショットセグメントの検出信頼度と識別子を含む ShotSegment オブジェクトです。

GetSegmentDetection の JSON レスポンスの例を次に示します。

```
{
    "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
        }
    ]
}
```


サンプルコードについては、「例: 保存されたビデオ内のセグメントの検出」を参照してください。

例: 保存されたビデオ内のセグメントの検出

以下の手順は、Amazon S3 バケットに保存されているビデオ内のテクニカルキューセグメントとショット検出セグメントを検出する方法を示しています。この手順では、Amazon Rekognition Video が持つ検出精度に対する信頼度に基づいて、検出されたセグメントをフィルタリングする方法も示します。

この例では、Amazon Simple Queue Service のキューを使用してビデオ分析リクエストの完了ステータスを取得する Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK) のコードを拡張します。

Amazon S3 バケットに保存されたビデオ内のセグメントを検出するには (SDK)

1. 「Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK)」を実行します。
2. ステップ 1 で使用したコードに以下を追加します。

   Java

   1. 次のインポートを追加します。

```
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. 次のコードを `VideoDetect` クラスに追加します。

```
   //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. 関数 main で、以下の行を置き換えます。

```
       StartLabelDetection(amzn-s3-demo-bucket, video);

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

を:

```
       StartSegmentDetection(amzn-s3-demo-bucket, video);

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

Java V2

```
//snippet-start:[rekognition.java2.recognize_video_text.import]
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.TextDetectionResult;
import java.util.List;
//snippet-end:[rekognition.java2.recognize_video_text.import]

/**
* Before running this Java V2 code example, set up your development environment,
 including your credentials.
*
* For more information, see the following documentation topic:
*
* https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
*/
public class DetectVideoSegments {

 private static String startJobId ="";
 public static void main(String[] args) {

     final String usage = "\n" +
         "Usage: " +
         "   <bucket> <video> <topicArn> <roleArn>\n\n" +
         "Where:\n" +
```

```
        "   bucket - The name of the bucket in which the video is located (for
example, (for example, amzn-s3-demo-bucket). \n\n"+
        "   video - The name of video (for example, people.mp4). \n\n" +
        "   topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic. \n\n" +
        "   roleArn - The ARN of the AWS Identity and Access Management (IAM)
role to use. \n\n" ;

    if (args.length != 4) {
        System.out.println(usage);
        System.exit(1);
    }

    String bucket = args[0];
    String video = args[1];
    String topicArn = args[2];
    String roleArn = args[3];

    Region region = Region.US_WEST_2;
    RekognitionClient rekClient = RekognitionClient.builder()
        .region(region)
        .credentialsProvider(ProfileCredentialsProvider.create("profile-name"))
        .build();

    NotificationChannel channel = NotificationChannel.builder()
        .snsTopicArn(topicArn)
        .roleArn(roleArn)
        .build();

    startTextLabels(rekClient, channel, bucket, video);
    GetTextResults(rekClient);
    System.out.println("This example is done!");
    rekClient.close();
}

// snippet-start:[rekognition.java2.recognize_video_text.main]
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);
        }
    }
    // snippet-end:[rekognition.java2.recognize_video_text.main]
}
```

Python

1. ステップ 1 で作成したクラス VideoDetect に次のコードを追加します。

```
# 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. 関数 main で、以下の行を置き換えます。

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

を:

```
    analyzer.StartSegmentDetection()
    if analyzer.GetSQSMessageSuccess()==True:
        analyzer.GetSegmentDetectionResults()
```

> **Note**
>
> Java または Python を使用した、Amazon S3 バケットに保存されたビデオの分析 (SDK) 以外のビデオ例をすでに実行している場合、置き換えるコードは異なる可能性があります。

3. コードを実行します。入力ビデオで検出されたセグメントに関する情報が表示されます。

顔のライブネスの検出

Amazon Rekognition Face Liveness は、顔認証を受けているユーザーがカメラの前に物理的にいることを確認するのに役立ちます。カメラに提示されたなりすまし攻撃、またはカメラをバイパスしようとする行為を検出します。ユーザーは、短い自撮りビデオを撮影して、一連のプロンプトに従って自分の存在を検証することで、Face Liveness チェックを完了できます。

Face Liveness は確率的計算によって決定され、チェック後に信頼スコア (0 ~ 100) が返されます。スコアが高いほど、チェックした人物が実在している確実性が高くなります。また、Face Liveness は、顔の比較や検索に使用できるリファレンスイメージと呼ばれるフレームも返します。他の確率ベースのシステムと同様に、Face Liveness は完璧な結果を保証することはできません。他の要素と組み合わせて使用することで、ユーザーの個人的アイデンティティについてリスクに基づいた決定を下せます。

Face Liveness は複数のコンポーネントを使用します。

- AWS FaceLivenessDetector コンポーネントを使用した Amplify SDK (React、Swift (iOS)、Android)
- AWS SDKs
- AWS クラウド APIs

Face Liveness 機能と統合するようにアプリケーションを設定すると、次の API オペレーションが使用されます。

- CreateFaceLivenessSession - Face Liveness セッションを開始し、Face Liveness 検出モデルをアプリケーションで使用できるようにします。作成されたセッションの SessionId を返します。ChallengePrefrence を設定して、FaceMovementChallenge オプションを使用することもできます。
- StartFaceLivenessSession - AWS Amplify FaceLivenessDetector によって呼び出されます。現在のセッション内の関連するイベントと属性に関する情報を含むイベントストリームを開始します。
- GetFaceLivenessSessionResults - Face Liveness の信頼スコア、リファレンスイメージ、監査イメージなど、特定の Face Liveness セッションの結果を取得します。

AWS Amplify SDK を使用して、Face Liveness 機能をウェブアプリケーションの顔ベースの検証ワークフローと統合します。ユーザーがアプリケーションを通じてオンボーディングまたは認証され

たら、ユーザーを Amplify SDK の Face Liveness チェックワークフローに送信します。Amplify SDK は、ユーザーが自撮りビデオをキャプチャしている間に、ユーザーインターフェイスとリアルタイムのフィードバックを処理します。

FaceMovementAndLightChallenge を使用すると、ユーザーの顔がデバイスに表示される楕円形に移動すると、Amplify SDK は画面に一連の色付きのライトを表示します。その後、自撮りビデオをクラウド API に安全にストリーミングします。または、FaceMovementChallenge を使用する場合、ユーザーの顔はデバイスに表示される楕円形に移動しますが、色付きのライトのシーケンスはありません。「FaceMovementAndLightChallenge」は精度を最大化するための最適な設定のままですが、「FaceMovementChallenge」を使用すると、お客様は精度よりもより高速なライブネスチェックを優先できます。これらの設定から選択するときは、予想される攻撃タイプ、希望する誤承認率と誤拒否率、位置情報 (IP に基づくなど) 、ワンタイムパスコード (OTPs) などの追加のチェックなど、ユースケースの要件を考慮する必要があります。お客様は、特定のコンテンツに対してさまざまな信頼スコアのしきい値を使用して Liveness のパフォーマンスをテストした後に、この決定を行う必要があります。さらに、両方のライブネスチェックタイプでは、アプリケーション所有者は、ビデオストリームの送信元のデバイスを保護するためのコントロールを実装する必要があります。分析が完了すると、バックエンドに次の情報が表示されます。

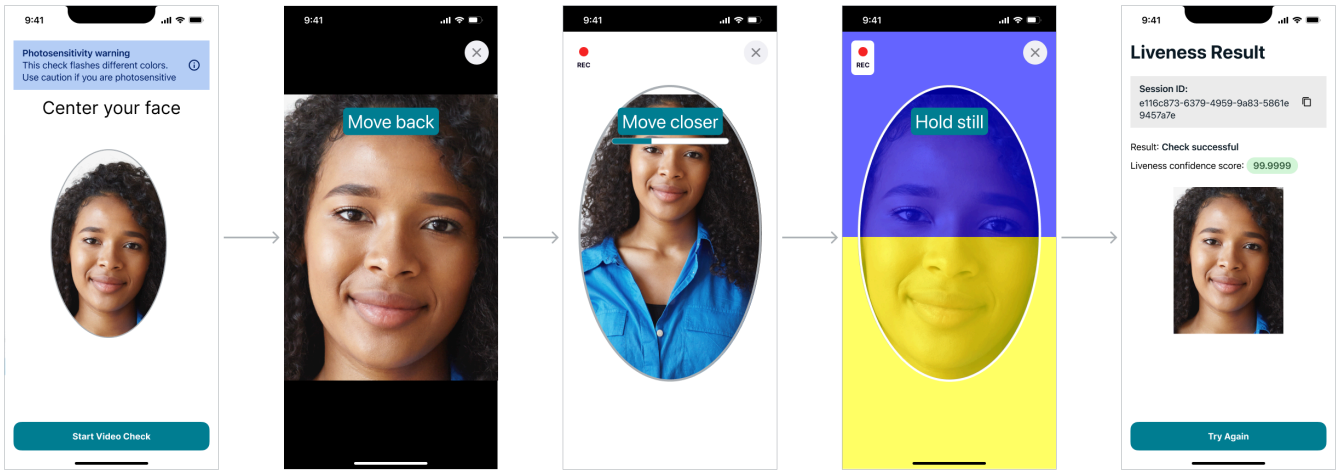
- Face Liveness の信頼スコア (0 ~ 100)
- 顔照合や顔検索に使用できるリファレンスイメージと呼ばれる高品質イメージ
- 自撮りビデオから選択した、監査イメージと呼ばれる最大 4 つのイメージセット

Face Liveness は、さまざまなユースケースで活用できます。例えば、顔のライブネスチェックを顔照合 (CompareFaces および SearchFacesByImage を使用) と併用すると、本人確認を行ったり、年齢ベースのアクセス制限のあるプラットフォームで年齢を推定したり、ボットを抑止しながら本物の人間のユーザーを検出したりできます。

このサービスの想定されるユースケース、サービスでの機械学習 (ML) の使用方法、サービスの責任ある設計と使用における重要な考慮事項については、Rekognition Face Liveness AI サービスカードで詳しく説明されています。

顔のライブネスチェックと顔照合の信頼スコアのしきい値を設定できます。選択したしきい値は、自分のユースケースを反映させたものである必要があります。次に、スコアがしきい値を上回るか下回るかを基準に、ユーザーに ID 認証の承認/拒否を送信します。拒否された場合は、ユーザーに再試行してもらうか、別の方法を紹介します。

以下の図は、指示から Liveness チェック、結果の返信までのユーザーフローを示しています。



ユーザー側の Face Liveness 要件

Amazon Rekognition Face Liveness には、以下の最小仕様が必要です。

デバイス:

- デバイスが前面カメラを備えていること
- デバイスディスプレイの最小リフレッシュレート: 60 Hz
- ディスプレイまたは画面の最小サイズ: 4 インチ
- 端末がジェイルブレイクまたはルート化されていないこと

カメラの仕様:

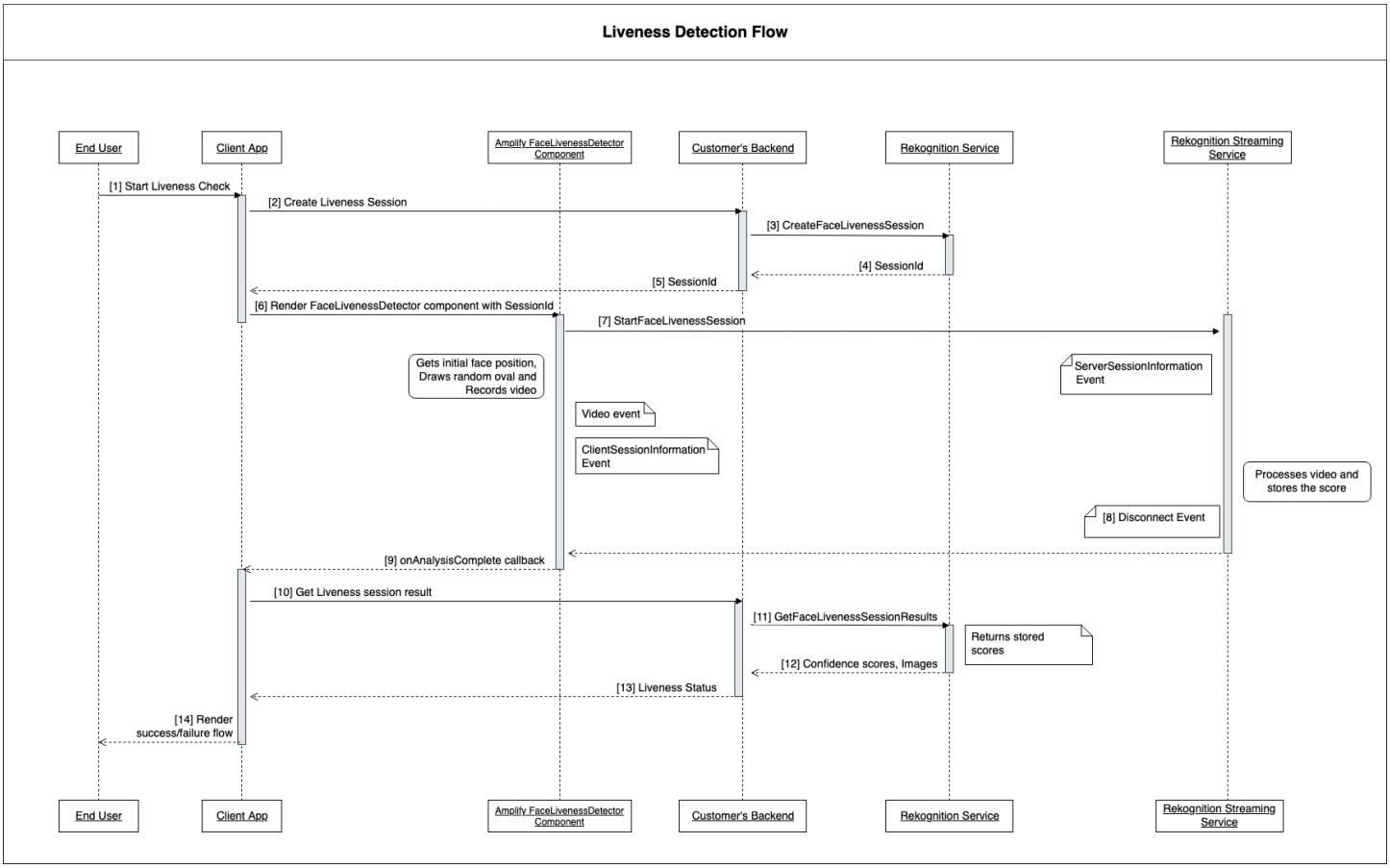
- カラーカメラ: 前面カメラで色を記録できること
- バーチャルカメラやカメラソフトウェアではないこと
- 最小記録機能: 15 フレーム/秒
- 最小ビデオ録画解像度: 480x640px。
- ユーザーがウェブカメラをデスクトップで使用して Face Liveness チェックを行う場合、Face Liveness チェックが開始される画面と同じ画面上にカメラを取り付けることが重要です。

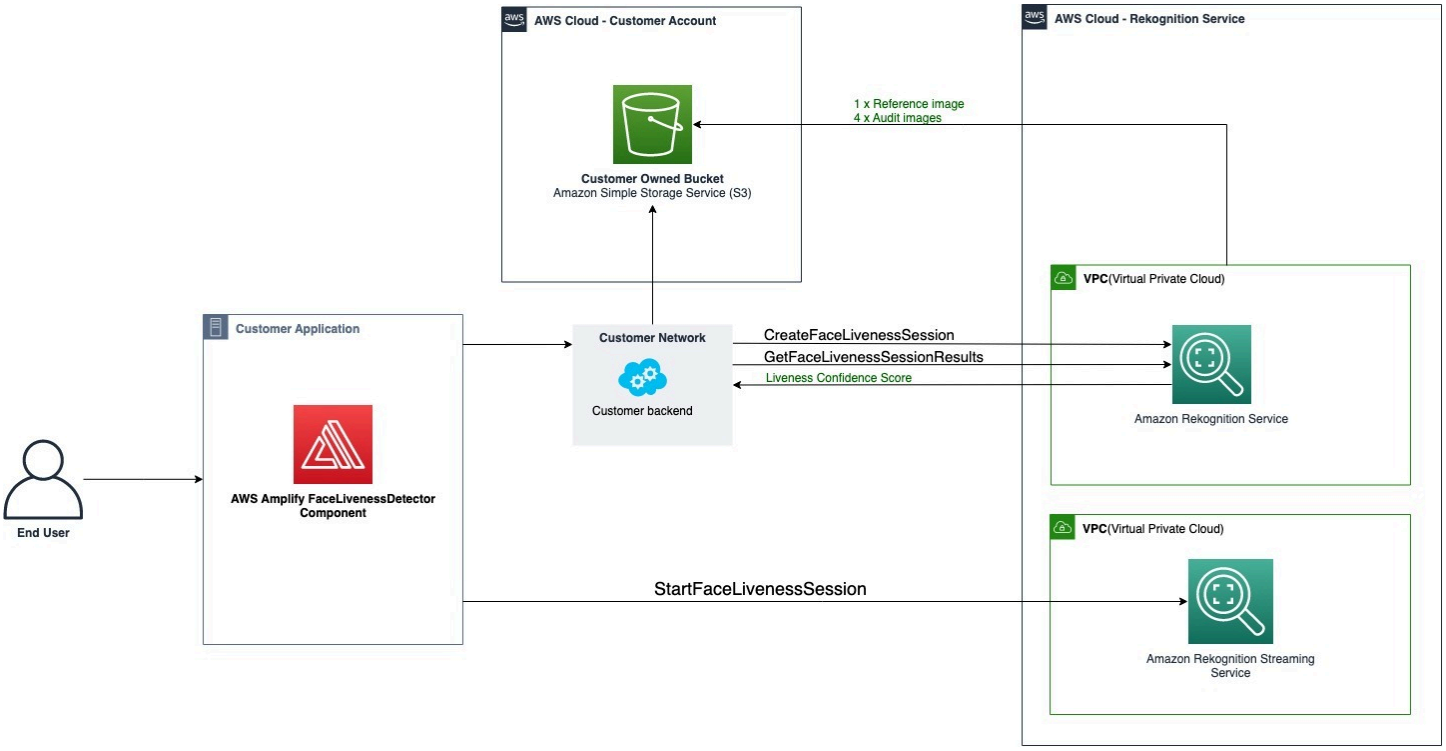
最小帯域幅要件: 100 kbps

対応ブラウザ: Google Chrome、Mozilla Firefox、Apple Safari、Microsoft Edge など、主要ブラウザの最新の 3 バージョン ブラウザのサポートに関する詳細については、「AWS マネジメントコンソールでサポートされているブラウザを教えてください。」を参照してください。

アーキテクチャ図とシーケンス図

以下の図は、Amazon Rekognition Face Liveness が機能のアーキテクチャと一連のオペレーションに関してどのように動作するかを詳しく説明しています。





Face Liveness チェックプロセスには、以下に概説するいくつかのステップが含まれています。

1. ユーザーはクライアントアプリケーションで Face Liveness チェックを開始します。
2. クライアントアプリケーションがお客様のバックエンドを呼び出し、そのバックエンドが Amazon Rekognition サービスを呼び出します。このサービスは Face Liveness セッションを作成し、固有の SessionId を返します。注: SessionId は送信後 3 分で期限切れになるため、以下のステップ 3~7 を完了するまでの時間は 3 分間しかありません。顔のライブネスチェックごとに新しい sessionID を使用する必要があります。同じ sessionID を後続の顔のライブネスチェックで使用すると、チェックは失敗します。さらに、SessionId は送信後 3 分で期限切れになり、セッションに関連付けられたすべてのライブネスデータ (sessionID、参照画像、監査画像など) は使用できなくなります。
3. クライアントアプリケーションは、取得した SessionId と適切なコールバックを使用して FaceLivenessDetector Amplify コンポーネントをレンダリングします。
4. FaceLivenessDetector コンポーネントは Amazon Rekognition ストリーミングサービスへの接続を確立し、ユーザーの画面上に楕円形をレンダリングし、色付きのライトのシーケンスを表示します。FaceLivenessDetector は、ビデオを録画して Amazon Rekognition ストリーミングサービスにリアルタイムでストリーミングします。

5. Amazon Rekognition ストリーミングサービスはビデオをリアルタイムで処理して結果を保存し、ストリーミングが完了すると DisconnectEvent を FaceLivenessDetector コンポーネントに返します。
6. FaceLivenessDetector コンポーネントは、`onAnalysisComplete` コールバックを呼び出して、ストリーミングが完了し、スコアを取得できる状態になったことをクライアントアプリケーションに通知します。
7. クライアントアプリケーションはお客様のバックエンドを呼び出して、ユーザーが稼働中かどうかを示すブール値のフラグを取得します。お客様のバックエンドは、Amazon Rekognition サービスに対して、信頼スコア、リファレンス、監査イメージを取得するためのリクエストを送信します。カスタマーバックエンドはこれらの属性を使用してユーザーが稼働中かどうかを判断し、クライアントアプリケーションに適切なレスポンスを返します。
8. 最後に、クライアントアプリケーションは FaceLivenessDetector コンポーネントにレスポンスを渡します。これにより、成功/失敗のメッセージが適切にレンダリングされ、フローが完了します。

前提条件

Amazon Rekognition Face Liveness を使用するための前提条件は以下のとおりです。

1. AWS アカウントのセットアップ
2. Face AWS SDKsのセットアップ
3. Amplify AWS リソースの設定

ステップ 1: AWS アカウントを設定する

AWS アカウントをまだお持ちでない場合は、「」の手順を実行してアカウントAWS アカウントとユーザーを作成するを作成します。

ステップ 2: Face Liveness AWS SDK をセットアップする

まだインストールしていない場合は、 と AWS SDKsをインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

AWS SDK 呼び出しを認証する方法はいくつかあります。このガイドの例では、CLI コマンドと AWS SDK API AWS オペレーションを呼び出すためにデフォルトの認証情報プロファイルを使用していることを前提としています。

選択した AWS SDK へのアクセス権をユーザーアカウントに付与する方法の詳細については、「プログラムによるアクセスの付与」ページを参照してください。このページでは、ローカルコンピュータでプロファイルを使用する方法と、 AWS 環境でサンプルコードを実行する方法についても説明します。

Face Liveness オペレーションを呼び出すユーザーに、`AmazonRekognitionFullAccess` や `AmazonS3FullAccess` 権限など、オペレーションを呼び出すための正しいアクセス権限があることを確認してください。

ステップ 3: Amplify AWS リソースを設定する

Amazon Rekognition Face Liveness をアプリケーションに統合するには、FaceLivenessDetector Amplify コンポーネントを使用するように AWS Amplify SDK をセットアップする必要があります。

まだインストールしていない場合は、「AWS CLI の開始方法」に記載されている指示に従って AWS コマンドラインインターフェイス (AWS CLI) をセットアップします。CLI をインストールしたら、Amplify UI ドキュメントサイトに表示される認証の設定ステップを完了して、Amplify AWS リソースを設定します。

Face Liveness を検出するためのベストプラクティス

Amazon Rekognition Face Liveness を使用する際は、いくつかのベストプラクティスに従うことをお勧めします。Face Liveness のベストプラクティスには、Face Liveness チェックを実施すべき箇所、監査イメージの使用、信頼スコアのしきい値の選択に関するガイドラインが含まれます。

ベストプラクティスの完全なリストについては、「Face Liveness の使用に関する推奨事項」を参照してください。

Amazon Rekognition Face Liveness API のプログラミング

Amazon Rekognition Face Liveness API を使用するには、以下のステップを実行するバックエンドを作成する必要があります。

1. CreateFaceLivenessSession を呼び出して、Face Liveness セッションを開始します。`CreateFaceLivenessSession` オペレーションが完了すると、UI はユーザーに自撮りビデオの送信を求めるプロンプトを表示します。次に、AWS Amplify の FaceLivenessDetector コンポーネントが StartFaceLivenessSession を呼び出してライブネス検出を実行します。
2. GetFaceLivenessSessionResults を呼び出して、Face Liveness セッションに関連する検出結果を返します。

3. Amplify Liveness ガイドに記載されているステップに従って、FaceLivenessDetector コンポーネントを使用するように React アプリケーションを設定します。

Face Liveness を使用する前に、AWS アカウントを作成し、AWS CLI と AWS SDK 、および AWS Amplify をセットアップしていることを確認してください。また、バックエンド API の IAM ポリシーに、GetFaceLivenessSessionResults と CreateFaceLivenessSession を実行できる権限があることを確認する必要があります。詳細については、「前提条件」セクションを参照してください。

ステップ 1: CreateFaceLivenessSession

CreateFaceLivenessSession API オペレーションは、Face Liveness セッションを作成し、固有の SessionId を返します。

このオペレーションに対する入力の一部として、Amazon S3 バケットの場所を指定することもできます。これにより、Face Liveness セッション中に生成されたリファレンスイメージと監査イメージを保存できます。Amazon S3 バケットは、発信元の AWS アカウント内にあり、かつ Face Liveness エンドポイントと同じリージョンにある必要があります。また、S3 オブジェクトキーは Face Liveness システムによって生成されます。

0 ~ 4 の数字の範囲で AuditImagesLimit を指定することもできます。デフォルトでは、0 に設定されています。返されるイメージ数はベストエフォートであり、自撮りビデオの再生時間に基づいています。

最後に、セッションをデフォルトにするチャレンジのタイプとバージョンを含む ChallengePreference を指定できます。デフォルトでは、FaceLivenessMovementAndLightChallenge を使用しますが、FaceMovementChallenge に設定できます。

リクエストの例

```
{
   "ClientRequestToken": "string",
   "KmsKeyId": "string",
   "Settings": {
      "AuditImagesLimit": number,
      "ChallengePreferences": [
         {
            "Type": "string",
            "Versions": {
```

```
                "Maximum": "string",
                "Minimum": "string"
             }
          }
       ],
       "OutputConfig": {
          "S3Bucket": "string",
          "S3KeyPrefix": "string"
       }
    }
}
```

レスポンスの例

```
{
    {"SessionId": "0f959dbb-37cc-45d8-a08d-dc42cce85fa8"}
}
```

ステップ 2: StartFaceLivenessSession

CreateFaceLivenessSession API オペレーションが完了すると、AWS Amplify コンポーネントは StartFaceLivenessSession API オペレーションを実行します。ユーザーは自撮りビデオをキャプチャするように求められます。チェックを正常に行うには、ユーザーは適切な照明を保ちながら、顔を画面上の楕円形の中に配置する必要があります。詳細については、「Face Liveness の使用に関する推奨事項」を参照してください。

この API オペレーションには、Face Liveness セッション中にキャプチャされたビデオ、CreateFaceLivenessSession API オペレーションから取得した sessionId、`onAnalysisComplete` コールバックが必要です。このコールバックを使用して、GetFaceLivenessSessionResults API オペレーションを呼び出すようにバックエンドに通知できます。これにより、信頼スコア、リファレンス、監査イメージが返されます。

このステップは、クライアントアプリケーションの AWS Amplify FaceLivenessDetector コンポーネントによって実行されることに注意してください。`StartFaceLivenessSession` を呼び出すための追加のセットアップは必要ありません。

ステップ 3: GetFaceLivenessSessionResults

GetFaceLivenessSessionResults API オペレーションは、特定の Face Liveness セッションの結果を取得します。sessionId を入力として必要とし、対応する Face Liveness の信頼スコアを返します。

また、顔の境界ボックスを含むリファレンスイメージと、顔の境界ボックスを含む監査イメージも提供されます。Face Liveness の信頼スコアの範囲は 0 ~ 100 です。

リクエストの例

```
{"SessionId": "0f959dbb-37cc-45d8-a08d-dc42cce85fa8"}
```

レスポンスの例

```
{
    "SessionId": "0f959dbb-37cc-45d8-a08d-dc42cce85fa8",
    "Confidence": 98.9735,
    "ReferenceImage": {
        "S3Object": {
            "Bucket": "s3-bucket-name",
            "Name": "file-name",
        },
        "BoundingBox": {
           "Height": 0.4943420886993408,
            "Left": 0.8435328006744385,
            "Top": 0.8435328006744385,
            "Width": 0.9521094560623169}
    },
    "AuditImages": [{
        "S3Object": {
            "Bucket": "s3-bucket-name",
            "Name": "audit-image-name",
        },
        "BoundingBox": {
           "Width": 0.6399999856948853,
           "Height": 0.47999998927116394,
           "Left": 0.1644444465637207,
           "Top": 0.17666666209697723}
    }],
    "Status": "SUCCEEDED"
}
```

ステップ 4: 結果への対応

Face Liveness セッションの後、チェックの信頼スコアと指定されたしきい値を比較します。スコアがしきい値よりも高い場合、ユーザーは次の画面またはタスクに進めます。チェックに失敗すると、ユーザーに通知され、再試行するよう求められます。

Face Liveness API を呼び出す

Amazon Rekognition Face Liveness は、AWS Python AWS SDK Boto3 や AWS SDK for Java など、サポートされている任意の SDK でテストできます。 Boto3 `CreateFaceLivenessSession` と `GetFaceLivenessSessionResults` API は、選択した SDK を使用して呼び出せます。次のセクションでは、Python および Java SDK を使用してこれらの API を呼び出す方法を説明します。

Face Liveness API を呼び出すには:

- `AmazonRekognitionFullAccess` アクセス権限のあるユーザーをまだ作成または更新していない場合は、ここで行ってください。詳細については、「Step 1: Set up an AWS account and create a User」を参照してください。
- まだ AWS CLI と AWS SDK をインストールして設定していない場合は、ここで行ってください。詳細については、「Step 2: Set up the AWS CLI and AWS SDK」を参照してください。

Python

次のスニペットは、Python アプリケーションでこれらの API を呼び出す方法を示しています。この例を実行するには、少なくともバージョン 1.26.110 の Boto3 SDK を使用する必要がありますが、最新バージョンの SDK が推奨されています。

```
import boto3

session = boto3.Session(profile_name='default')
client = session.client('rekognition')

def create_session():

    response = client.create_face_liveness_session()

    session_id = response.get("SessionId")
    print('SessionId: ' + session_id)
```

```
    return session_id


def get_session_results(session_id):

    response = client.get_face_liveness_session_results(SessionId=session_id)

    confidence = response.get("Confidence")
    status = response.get("Status")

    print('Confidence: ' + "{:.2f}".format(confidence) + "%")
    print('Status: ' + status)

    return status


def main():
    session_id = create_session()
    print('Created a Face Liveness Session with ID: ' + session_id)

    status = get_session_results(session_id)
    print('Status of Face Liveness Session: ' + status)


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

Java

次のスニペットは、Java アプリケーションでこれらの API を呼び出す方法を示しています。

```
package aws.example.rekognition.liveness;

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.CreateFaceLivenessSessionRequest;
import com.amazonaws.services.rekognition.model.CreateFaceLivenessSessionResult;
import
 com.amazonaws.services.rekognition.model.GetFaceLivenessSessionResultsRequest;
```

```
import
 com.amazonaws.services.rekognition.model.GetFaceLivenessSessionResultsResult;

public class DemoLivenessApplication {

    static AmazonRekognition rekognitionClient;

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

        rekognitionClient = AmazonRekognitionClientBuilder.defaultClient();

        try {
            String sessionId = createSession();
            System.out.println("Created a Face Liveness Session with ID: " +
 sessionId);

            String status = getSessionResults(sessionId);
            System.out.println("Status of Face Liveness Session: " + status);

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

    private static String createSession() throws Exception {

        CreateFaceLivenessSessionRequest request = new
 CreateFaceLivenessSessionRequest();
        CreateFaceLivenessSessionResult result =
 rekognitionClient.createFaceLivenessSession(request);

        String sessionId = result.getSessionId();
        System.out.println("SessionId: " + sessionId);

        return sessionId;
    }

    private static String getSessionResults(String sessionId) throws Exception {

        GetFaceLivenessSessionResultsRequest request = new
 GetFaceLivenessSessionResultsRequest().withSessionId(sessionId);
        GetFaceLivenessSessionResultsResult result =
 rekognitionClient.getFaceLivenessSessionResults(request);
```

```
        Float confidence = result.getConfidence();
        String status = result.getStatus();

        System.out.println("Confidence: " + confidence);
        System.out.println("status: " + status);

        return status;
    }
}
```

Java V2

次のスニペットは、Java V2 SDK を使用して Face Liveness APIs AWS を呼び出す方法を示しています。

```
package aws.example.rekognition.liveness;

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.CreateFaceLivenessSessionRequest;
import com.amazonaws.services.rekognition.model.CreateFaceLivenessSessionResult;
import
 com.amazonaws.services.rekognition.model.GetFaceLivenessSessionResultsRequest;
import
 com.amazonaws.services.rekognition.model.GetFaceLivenessSessionResultsResult;

public class DemoLivenessApplication {

    static AmazonRekognition rekognitionClient;

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

        rekognitionClient = AmazonRekognitionClientBuilder.defaultClient();

        try {
            String sessionId = createSession();
            System.out.println("Created a Face Liveness Session with ID: " +
 sessionId);
```

```
            String status = getSessionResults(sessionId);
            System.out.println("Status of Face Liveness Session: " + status);

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

    private static String createSession() throws Exception {

        CreateFaceLivenessSessionRequest request = new
 CreateFaceLivenessSessionRequest();
        CreateFaceLivenessSessionResult result =
 rekognitionClient.createFaceLivenessSession(request);

        String sessionId = result.getSessionId();
        System.out.println("SessionId: " + sessionId);

        return sessionId;
    }

    private static String getSessionResults(String sessionId) throws Exception {

        GetFaceLivenessSessionResultsRequest request = new
 GetFaceLivenessSessionResultsRequest().withSessionId(sessionId);
        GetFaceLivenessSessionResultsResult result =
 rekognitionClient.getFaceLivenessSessionResults(request);

        Float confidence = result.getConfidence();
        String status = result.getStatus();

        System.out.println("Confidence: " + confidence);
        System.out.println("status: " + status);

        return status;
    }
}
```

Node.Js

次のスニペットは、 AWS Node.Js SDK を使用して Face Liveness APIs を呼び出す方法を示しています。

```
const Rekognition = require("aws-sdk/clients/rekognition");

const rekognitionClient = new Rekognition({ region: "us-east-1" });

async function createSession() {
    const response = await rekognitionClient.createFaceLivenessSession().promise();

    const sessionId = response.SessionId;
    console.log("SessionId:", sessionId);

    return sessionId;
}

async function getSessionResults(sessionId) {
    const response = await rekognitionClient
        .getFaceLivenessSessionResults({
            SessionId: sessionId,
        })
        .promise();

    const confidence = response.Confidence;
    const status = response.Status;
    console.log("Confidence:", confidence);
    console.log("Status:", status);

    return status;
}

async function main() {
    const sessionId = await createSession();
    console.log("Created a Face Liveness Session with ID:", sessionId);

    const status = await getSessionResults(sessionId);
    console.log("Status of Face Liveness Session:", status);
}

main();
```

Node.Js (Javascript SDK v3)

次のスニペットは、 AWS Node.Js SDK for Javascript v3 を使用して Face Liveness APIs を呼び出す方法を示しています。

```
import { RekognitionClient, CreateFaceLivenessSessionCommand } from "@aws-sdk/
client-rekognition"; // ES Modules
import const { RekognitionClient, CreateFaceLivenessSessionCommand } =
 require("@aws-sdk/client-rekognition"); // CommonJS import
const client = new RekognitionClient(config);
const input = {
  KmsKeyId: "STRING_VALUE",
  Settings: {
    OutputConfig: { // LivenessOutputConfig
      S3Bucket: "STRING_VALUE", // required
      S3KeyPrefix: "STRING_VALUE",
    },
    AuditImagesLimit: Number("int"),
  },
  ClientRequestToken: "STRING_VALUE",
};
const command = new CreateFaceLivenessSessionCommand(input);
const response = await client.send(command);
// { // CreateFaceLivenessSessionResponse
//   SessionId: "STRING_VALUE", // required
// };
```

アプリケーションの設定とカスタマイズ

アプリケーションの設定

Face Liveness アプリケーションは、モバイルデバイスでもデスクトップウェブブラウザでも動作します。Face Liveness コンポーネントは、選択したソリューションと統合するように設定する必要があります。また、アプリケーションにデバイスのカメラを使用する権限があることも確認する必要があります。Amplify Liveness ガイドには、以下の方法に関する詳細な説明が記載されています。

- AWS Amplify をインストールして設定する
- FaceLivenessDetector コンポーネントをインポートしてレンダリングする
- コールバックをリッスンする

- Amplify のサンプルエラーメッセージ

アプリケーションをカスタマイズする

AWS Amplify を使用して、Liveness アプリケーションの特定のコンポーネントをカスタマイズできます。

翻訳の詳細については、Amplify Authenticator を参照してください。

Amplify コンポーネントとテーマのカスタマイズについては、テーマに関する Amplify のドキュメントを参照してください。

Face Liveness の責任共有モデル

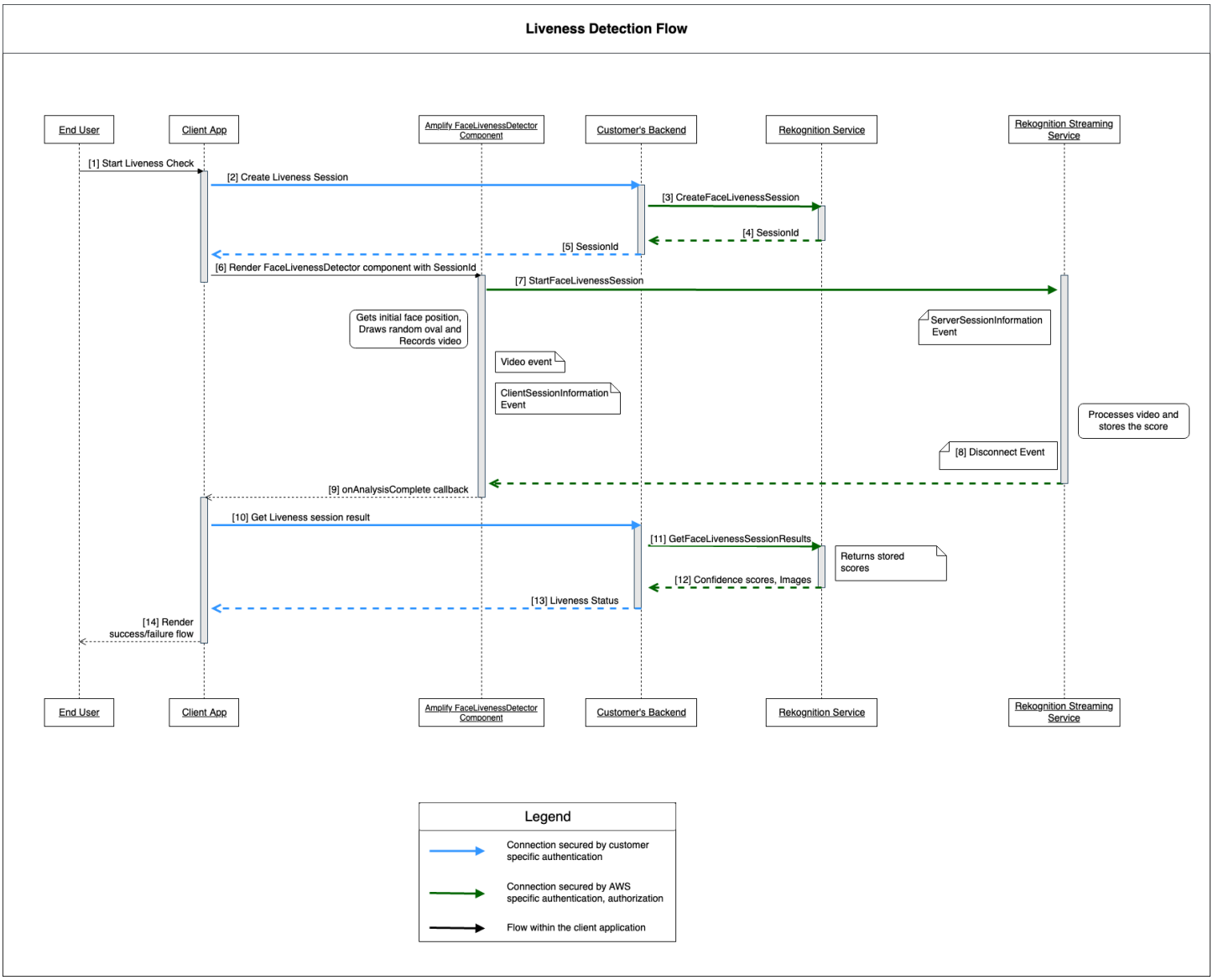
セキュリティとコンプライアンスは、 AWS お客様とお客様との間の責任共有です。責任 AWS 共有モデルの詳細については、こちらを参照してください。

1. AWS サービスへのすべての呼び出し (クライアントアプリケーションまたはカスタマーバックエンド経由) は認証され、認証 (AWS 認証) AWS で承認されます。この認証作業は、Face Liveness のサービスオーナーの責任です。
2. (クライアントアプリケーションからの) お客様のバックエンドへの呼び出しはすべて、お客様を介して認証および承認されます。この責任はお客様にあります。お客様は、クライアントアプリケーションからの呼び出しが認証され、何らかの方法で操作されていないことを確認する必要があります。
3. お客様のバックエンドは、Face Liveness チャレンジを実行しているエンドユーザーを特定する必要があります。エンドユーザーを Face Liveness セッションに結び付けるのはお客様の責任です。Face Liveness サービスはエンドユーザーを区別しません。発信者 AWS ID (顧客が処理する ID) のみを識別できます。
4. AWS では、ユースケースの要件とセキュリティ体制に沿った Face Liveness に加えて、位置情報 (IP に基づくなど) 、ワンタイムパスコード (OTPs) などの追加の検証チェックを適用することを推奨しています。

FaceMovementAndLightChallenge」設定は、ユーザーが顔を画面に向かって移動し、一連の点滅するライトを動かさないようにすることで、Rekognition Liveness に最高の精度を提供します。このデフォルト設定を使用することをお勧めします。または、「FaceMovementChallenge」設定を有効にして、点滅するライトを削除することでチェック時間を数秒短縮することもでき

ます。「FaceMovementAndLightChallenge」は精度を最大化するための最適な設定ですが、「FaceMovementChallenge」を使用すると、お客様はより高速なライブネスチェックを優先できます。これらの設定から選択するときは、予想される攻撃タイプ、希望する誤承認率、誤拒否率などのユースケース要件を検討し、位置情報 (IP に基づくなど) 、ワンタイムパスコード (OTPs) などの追加のチェックを実装する必要があります。お客様は、ユースケースに応じてさまざまな信頼スコアのしきい値で Liveness のパフォーマンスをテストした後に、この決定を行う必要があります。お客様は、動画の送信元のデバイスを保護するためのコントロールを実装する責任があります。

次のフロー図は、どの呼び出しが AWS のサービスまたはお客様によって認証されるかを示しています。



Amazon Rekognition Face Liveness サービスへのすべての呼び出しは AWS 、Auth によって保護されます (署名メカニズムを使用 AWS) 。これには以下の呼び出しが含まれます。

- [3] CreateFaceLivenessSession API コール (お客様のバックエンドから)
- [7] StartFaceLivenessSession API コール (クライアントアプリケーションから)
- [11] GetFaceLivenessSessionResults API コール (お客様のバックエンドから)

お客様のバックエンドへのすべての呼び出しには、認証と認可のメカニズムが必要です。お客様は、使用されているサードパーティーのコード/ライブラリなどがアクティブに保守および開発されていることを確認する必要があります。また、お客様は、正しいエンドユーザーが正しい Face Liveness セッションを呼び出していることを確認する必要があります。お客様は以下のフローを認証して認可する必要があります。

- [2] Face Liveness セッションの作成 (クライアントアプリケーションから)
- [10] Face Liveness セッションの結果の取得 (クライアントアプリケーションから)

お客様は STRIDE セキュリティモデルに従って API コールが保護されていることを確認できます。

タイプ	説明	セキュリティコントロール
Spoofing	Threat action aimed at accessing and use of another user's credentials, such as username and password.	Authentication
Tampering	Threat action intending to maliciously change or modify persistent data. Examples include records in a database, and the alteration of data in transit between two computers over an open network, such as the internet.	Integrity
Repudiation	Threat action aimed at performing prohibited operations in a system that lacks the ability to trace the operations.	Non-Repudiation

Information disclosure	Threat action intending to read a file that one was not granted access to, or to read data in transit.	Confidentiality
Denial of service	Threat action attempting to deny access to valid users, such as by making a web server temporarily unavailable or unusable.	Availability
Elevation of privilege	Threat action intending to gain privileged access to resources in order to gain unauthorized access to information or to compromise a system.	Authorization

AWS は、次の方法で接続を保護します。

1. リクエストの署名を計算し、サービス側で署名を検証します。リクエストはこの署名を使用して認証されます。
2. AWS のお客様は、特定のアクション/オペレーションを承認する適切な IAM ロールを設定する必要があります。これらの IAM ロールは AWS のサービスへの呼び出しを行うために必要なものです。
3. AWS サービスへの HTTPS リクエストのみが許可されます。リクエストは TLS を使用してオープンネットワークで暗号化されます。これにより、リクエストの機密性が保護され、リクエストの完全性が維持されます。
4. AWS サービスは、顧客が行った呼び出しを識別するのに十分なデータをログに記録します。これにより、否認攻撃を防げます。
5. AWS サービスが十分な可用性を維持する

お客様は、以下の方法でサービスと API コールを保護する責任があります。

1. お客様は、適切な認証メカニズムに従っていることを確認する必要があります。リクエストの認証には、さまざまな認証メカニズムを使用できます。お客様は、ダイジェストベースの認証、OAuth、OpenID 接続、その他のメカニズムを利用できます。

2. お客様は、サービス API コールを行うための適切な暗号化チャンネル (TLS/HTTPS など) をサービスがサポートしていることを確認する必要があります。
3. お客様は、API コールと発信者を一意に識別するのに必要なデータをログに記録する必要があります。定義されたパラメータと呼び出し時刻を使用して、API を呼び出しているクライアントを識別できるようにする必要があります。
4. お客様は、システムが利用可能であることと、DDoS 攻撃から保護されていることを確認する必要があります。DDoS 攻撃に対する防御技術の例をいくつか紹介します。

お客様は、アプリケーションを最新の状態に維持する責任があります。詳細については、「Face Liveness の更新に関するガイドライン」を参照してください。

Face Liveness の更新に関するガイドライン

AWS は、定期的に Face AWS SDKs (お客様のバックエンドで使用) AWS と Amplify SDKs (クライアントアプリケーションで使用) の FaceLivenessDetector コンポーネントを更新して、新機能、更新された APIs、拡張セキュリティ、バグ修正、ユーザビリティの向上などを提供します。機能が最適に動作するように、SDK を最新の状態に保つことをお勧めします。古いバージョンの SDK を引き続き使用すると、保守性やセキュリティ上の理由からリクエストがブロックされる可能性があります。

Face Liveness では、Amplify SDKs (React、iOS、Android) に含まれている FaceLivenessDetector AWS コンポーネントを使用する必要があります。

バージョニングとタイムフレーム

Face Liveness 機能の以下の主要コンポーネントは、AWS によってバージョニングされています。その際、セマンティックバージョニング形式に従います。例えば、X.Y.Z のバージョン形式では、X はメジャーバージョン、Y はマイナーバージョン、Z はパッチバージョンを表します。

- Face Liveness のユーザーチャレンジ (FaceMovementAndLightChallenge チャレンジなど) は StartFaceLivenessSession API の一部です。
- Amplify SDKs を介して配信される FaceLivenessDetector AWS コンポーネントは、クライアントアプリケーションで使用されます。

メジャーバージョン: 重大なセキュリティ、API の破損、ユーザビリティの更新の致命的な問題に対処するため、メジャーバージョンの更新が予約されています。Face Liveness の機能を引き続き使用

するには、アプリケーションとお客様のバックエンドをできるだけ早く更新する必要があります。新しいメジャーバージョンがリリースされると、その日から 120 日間、以前のメジャーバージョンがサポートされます。120 日経過すると、以前のメジャーバージョンからのリクエストがブロックされる場合があります。

マイナーバージョン: セキュリティやユーザビリティに関する重要な機能や改善に対応するため、マイナーバージョンの更新が予約されています。これらの更新を適用することを強くお勧めします。マイナー更新の下位互換性をできるだけ長く維持できるよう努めていますが、新しいマイナーバージョンがリリースされてから 180 日後に、以前のマイナーバージョンのサポートの終了を発表することがあります。

パッチバージョン: パッチバージョンの更新は、オプションのバグ修正や改善に対応するためのものです。最高のセキュリティとユーザーエクスペリエンスを実現するために、ご使用のバージョンを最新の状態に保つことをお勧めしますが、新しいメジャーバージョンまたはマイナーバージョンがリリースされるまでは、パッチアップデートの完全な下位互換性を維持できるよう努めています。

バージョニング期間 (メジャーは 120 日、マイナーは 180 日) は、アプリケーション内の SDK を更新したり、アプリケーションストアまたはウェブサイトへのアプリケーションをアップロードしたり、ユーザーがアプリケーションの最新バージョンをダウンロードしたりする場合に適用されます。

バージョンリリースと互換性マトリックス

FaceLivenessDetector コンポーネントのメジャーバージョンまたはユーザーチャレンジのリリースは、多くの場合同時に行われます。バージョンの依存関係を把握しやすくするには、以下の表にリンクされているリソースを参照してください。

SDK バージョンと変更ログ:

FaceLivenessDetector for web SDK		FaceLivenessDetector for iOS SDK	FaceLivenessDetector for Android SDK
現在のバージョン	変更ログ	現在のバージョン/変更ログ	現在のバージョン/変更ログ

ユーザーチャレンジ:

Challenge Name	Version	Release date	Retire date

FaceMovementAndLightChallenge	v1.0.0	4/10/2023	N/A
FaceMovementChallenge	v1.0.0	4/30/2025	N/A

新しいリリースの通知

AWS は、次のチャネルを通じて新しいリリースを通信します。

- Face Liveness アカウント ID に関連付けられたアカウントメールアドレスにサービスヘルスの更新メール通知が送信されます。
- 各 GitHub リポジトリで AWS SDKsおよび関連する通知の更新を公開しました。
- Amplify SDKs AWS の更新と関連する通知を各 GitHub リポジトリに公開しました。

最新情報を入手するには、これらのチャンネルをサブスクライブすることをお勧めします。

Face Liveness に関するよくある質問

以下の FAQ 項目を参照して、Rekognition Face Liveness に関するよくある質問の回答を見つけてください。

- Face Liveness チェックの出力はどのようなものですか?

  Rekognition Face Liveness は、すべてのライブネスチェックに対して以下の出力を提供します。
  - 信頼スコア: 0 ~ 100 の範囲で数値スコアが返されます。このスコアは、自撮りビデオが実在の人物によって撮影されたもので、悪質ななりすまし行為ではない可能性を示します。
  - 高品質イメージ: 自撮りビデオから 1 つの高品質イメージが抽出されます。このフレームは、顔の比較、年齢推定、顔の検索など、さまざまな用途に利用できます。
  - 監査イメージ: 自撮りビデオから最大 4 つのイメージが返され、監査証跡に使用できます。
- Rekognition Face Liveness は iBeta プレゼンテーション攻撃検出 (PAD) テストに準拠していますか?

  iBeta Quality Assurance のプレゼンテーション攻撃検出 (PAD) 試験は ISO/IEC 30107-3 に従って実施されます。iBeta は NIST/NVLAP の認定を受けており、この PAD 規格に準拠したテストと

結果の提供を行っています。Rekognition Face Liveness は、iBeta プレゼンテーション攻撃検出 (PAD) の適合性テスト (レベル 1 とレベル 2) に合格し、PAD スコアは満点でした。レポートは iBeta のウェブページに掲載されています。

- 高品質のフレームと追加のフレームを入手するにはどうすればよいですか?

  CreateFaceLivenessSession API リクエストの設定に応じて、高品質のフレームと追加のフレームを未加工のバイトとして返すか、指定した Amazon S3 バケットにアップロードできます。

- 楕円形や色付きのライトの位置を変更できますか?

  いいえ。楕円の位置と色付きのライトは、より高い精度を可能にする機能であるため、カスタマイズできません。

- アプリケーションに合わせてユーザーインターフェイスをカスタマイズできますか?

  はい。テーマ、色、言語、テキストコンテンツ、フォントなど、ほとんどの画面コンポーネントをアプリケーションに合わせてカスタマイズできます。これらのコンポーネントをカスタマイズする方法の詳細は、React、Swift、Android の UI コンポーネントのドキュメントに記載されています。

- 顔を楕円形の位置に配置する際のカウントダウン時間やフィット時間はカスタマイズできますか?

  いいえ。カウントダウン時間と顔のフィット時間は、1,000 人のユーザーを対象とした大規模な内部研究に基づいて事前に決定されており、より高い精度とレイテンシーを可能にする最適なバランスを提供することを目的としています。そのため、これらの時間設定はカスタマイズできません。

- 異なる色のライトはアクセシビリティガイドラインに準拠していますか?

  はい。当社製品のさまざまな色のライトは、WCAG 2.1 で概説されているアクセシビリティガイドラインに準拠しています。何千回ものユーザーチェックで確認されているように、ユーザーエクスペリエンスでは 1 秒あたり約 2 色が表示されます。これは、1 秒あたり 3 色に制限するという推奨事項に準拠しています。これにより、大勢の人に対してでんかん発作を引き起こす可能性を抑えています。

- SDK は最適な結果を得るために画面の明るさを調整していますか?

  Face Liveness モバイル SDK (Android および iOS 用) は、チェックが開始されると自動的に明るさを調整します。ただし、ウェブ SDK の場合、ウェブページには自動輝度調整ができないという

制限があります。このような場合、ウェブアプリケーションは、最適な結果を得るために画面の明るさを手動で上げるようエンドユーザーに促します。

- 楕円形にする必要はありますか? 他の似たような形は使用できますか?

  いいえ、楕円形のサイズ、形、位置はカスタマイズできません。特定の楕円形のデザインは、顔の動きを正確に捉えて分析する際に効果を発揮できるよう慎重に選択されています。そのため、楕円形は変更できません。

- エンドツーエンドのレイテンシーはどのくらいですか?

  Liveness チェックの完了に必要なアクションをユーザーが開始してから、結果 (合格または不合格) を取得するまでのエンドツーエンドのレイテンシーが測定されます。最適なケースのレイテンシーは 5 秒です。平均すると、約 7 秒になると予想されます。最悪なケースのレイテンシーは 11 秒です。エンドツーエンドのレイテンシーには、ユーザーが必要なアクションを完了する時間 (例えば、顔を楕円形の位置に移動する)、ネットワーク接続、アプリケーションのレイテンシーなどによってばらつきが見られます。

- Amplify SDK がなくても Face Liveness 機能は使用できますか?

  いいえ、Rekognition Face Liveness 機能を使用するには Amplify SDK が必要です。

- Face Liveness に関連するエラー状態はどこで確認できますか?

  Face Liveness のさまざまなエラー状態は、こちらで確認できます。

- 使用中のリージョンで Face Liveness を利用できません。この機能はどうすれば使用できますか?

  Face Liveness が利用可能なリージョンであれば、トラフィックの負荷や近接性に応じて、Face Liveness を呼び出せます。顔のライブネスは現在、次の AWS リージョンで利用できます。

  - 米国東部 (バージニア北部)
  - 米国西部 (オレゴン)
  - 欧州 (アイルランド)
  - アジアパシフィック (東京、ムンバイ)

  AWS アカウントが別のリージョンにある場合でも、レイテンシーの差は大きくないと予想されます。高品質の自撮りフレームと監査イメージは、Amazon S3 の場所または raw バイトで取得でき

ますが、Amazon S3 バケットは Face Liveness の AWS リージョンと一致する必要があります。これらが異なる場合は、イメージを未加工のバイトとして受け取る必要があります。

- Amazon Rekognition Liveness 検出では、カスタマーコンテンツを使用してサービスを改善していますか?

  お客様は、 AWS Organizations のオプトアウトポリシーを使用することにより、Rekognition やその他の Amazon 機械学習/人工知能技術の品質向上または開発を目的としてお客様のイメージやビデオの入力が使用されることを拒否できます。オプトアウトの方法については、「AI サービスのオプトアウトポリシー」を参照してください。

一括分析

Amazon Rekognition の一括分析を使用すると、StartMediaAnalysisJob オペレーションでマニフェストファイルを使用することにより、大量のイメージのコレクションを非同期的に処理できます。個々のイメージの出力は、分析に使用するオペレーションによって返される出力と一致します。

現在、Rekognition は DetectModerationLabels オペレーションの分析をサポートしています。

ユーザーは、ジョブが正常に処理したイメージの数量に基づき、料金を支払います。終了したジョブの結果は、指定された Amazon S3 バケットに出力されます。

一括分析は、Amazon A2I の統合をサポートしていませんのでご注意ください。

API はアニメーションまたはイラストのコンテンツタイプを検出でき、検出されたコンテンツタイプに関する情報はレスポンスの一部として返されます。

トピック

- イメージの一括処理
- 一括分析出力マニフェスト
- コンテンツタイプ
- 予測の検証とアダプターのトレーニング

イメージの一括処理

新しい一括分析ジョブは、マニフェストファイルを送信し、StartMediaAnalysisJob オペレーションを呼び出すことで開始できます。入力マニフェストファイルには Amazon S3 バケット内のイメージへの参照が含まれ、次のようにフォーマットされています。

```
{"source-ref": "s3://foo/bar/1.jpg"}
```

一括分析ジョブ (CLI) を作成するには

1. まだ実行していない場合:

   a. AmazonRekognitionFullAccess と AmazonS3ReadOnlyAccess のアクセス権限を持つユーザーを作成または更新します。詳細については、「ステップ 1: AWS アカウントを設定してユーザーを作成する」を参照してください。

b. と AWS SDKs をインストール AWS CLI して設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. イメージを S3 バケットにアップロードします。

   手順については、Amazon Simple Storage Service ユーザーガイドの「Amazon S3 へのオブジェクトのアップロード」を参照してください。

3. 一括分析ジョブを作成し取得するときは次のコマンドを使用します。

CLI

DetectModerationLabels オペレーションを使って分析を行うときは、次のコマンドで StartMediaAnalysisJob オペレーションを呼び出します。

```
# Requests
# Starting DetectModerationLabels job with default settings
aws rekognition start-media-analysis-job \
--operations-config "DetectModerationLabels={MinConfidence='1'}" \
--input "S3Object={Bucket=amzn-s3-demo-source-bucket,Name=my-input.jsonl}" \
--output-config "S3Bucket=amzn-s3-demo-destination-bucket;,S3KeyPrefix=my-results"
```

GetMediaAnalysisJob オペレーションを使用すると、結果ファイルとサマリーファイルが保存されているバケットの Amazon S3 パスなど、特定のジョブに関する情報を取得できます。そこで、StartMediaAnalysisJob または ListMediaAnalysisJob が返すジョブ ID を指定します。各ジョブの詳細の保持期間は 1 年間のみです。

```
# Request
aws rekognition get-media-analysis-job \
--job-id customer-job-id
```

ListMediaAnalysisJobs ジョブオペレーションを使用すると自身が行った一括分析を一覧表示できます。これによりジョブのページが返されます。max-results 引数を使用すると、1 ページあたりに返されるジョブの最大数を、max-results の値を上限として指定できます。1 ページあたり最大 100 件の結果が返されます。各ジョブの詳細の保持期間は 1 年間のみです。

```
# Request
# Specify number of jobs to return per page, limited to max-results.
aws rekognition list-media-analysis-jobs --max-results 1
```

一括分析出力マニフェスト

一括分析ジョブにより、ジョブ結果を含む出力マニフェストファイルと、入力マニフェストエントリを処理する際のエラーに関する統計と詳細を含むマニフェスト概要が生成されます。

入力マニフェストに重複するエントリが含まれていた場合、そのジョブは固有の入力を除外せず、代わりに提供されたすべてのエントリを処理します。

出力マニフェストファイルは次のようにフォーマットされます。

```
// Output manifest for content moderation
{"source-ref":"s3://foo/bar/1.jpg", "detect-moderation-labels":
 {"ModerationLabels":[],"ModerationModelVersion":"7.0","ContentTypes":
[{"Confidence":72.7257,"Name":"Animated"}]}}
```

出力マニフェスト概要は次のようにフォーマットされます。

```
{
    "version": "1.0",                # Schema version, 1.0 for GA.
    "statistics": {
        "total-json-lines": Number,  # Total number json lines (images) in the input
 manifest.
        "valid-json-lines": Number,  # Total number of JSON Lines (images) that contain
 references to valid images.
        "invalid-json-lines": Number # Total number of invalid JSON Lines. These lines
 were not handled.
    },
    "errors": [
        {
            "line-numer": Number,   # The number of the line in the manifest where the
 error occured.
            "source-ref": "String", # Optional. Name of the file if was parsed.
            "code": "String",       # Error code.
            "message": "String"     # Description of the error.
        }
     ]
}
```

コンテンツタイプ

StartMediaAnalysisJob オペレーションによって分析されたメディアコンテンツのタイプに関する情報は、GetMediaAnalysisJob オペレーションによって返されます。ContentType は以下の 2 つのカテゴリのいずれかになります。

- アニメーションコンテンツ。ビデオゲームやアニメーション (例: カートゥーン、コミック、マンガ、アニメ) が含まれます。
- イラストコンテンツ。線画、絵画、スケッチが含まれます。

予測の検証とアダプターのトレーニング

一括分析は Rekognition のコンソールからも使用でき、イメージのバッチを予測し、その予測を検証し、検証した予測を使ってアダプターを作成できます。アダプターを使用すると、サポートされているすべての Rekognition オペレーションの精度を高めることができます。

現在は、Rekognition カスタムモデレーション機能で使用するアダプターを作成できます。アダプターを作成して DetectModerationLabels オペレーションでこれを指定することにより、特定のユースケースに関連するコンテンツモデレーションタスクの精度をさらに高めることができます。

カスタムモデレーションの詳細については、「カスタムモデレーションによる精度の向上」を参照してください。一括分析による予測の検証方法については、「一括分析と検証」を参照してください。Rekognition のコンソールを使用して予測を検証し、アダプターを作成する方法のチュートリアルについては、「カスタムモデレーションのアダプターのチュートリアル」を参照してください。

チュートリアル

これらのクロスサービスチュートリアルでは、Rekognition の API オペレーションを他の AWS サービスとともに使用してサンプルアプリケーションを作成し、さまざまなタスクを実行する方法を示します。これらのチュートリアルのほとんどは、Amazon S3 を使用して画像や動画を保存します。その他、一般的に利用されているサービスとしては AWS Lambdaがあります。

トピック

- Amazon RDS および DynamoDB で Amazon Rekognition データを保存する
- Amazon Rekognition と Lambda を使用して Amazon S3 バケットのアセットにタグを付ける
- AWS ビデオアナライザーアプリケーションの作成
- Amazon Rekognition Lambda 関数の作成
- 本人確認における Amazon Rekognition の使用
- マーケティングアプリケーション用に Amazon Rekognition (REK) を使用してラベルを検出する
- Lambda と Python を使用したイメージ内のラベルの検出

Amazon RDS および DynamoDB で Amazon Rekognition データを保存する

Amazon Rekognition の API を使用する場合、API オペレーションは生成されたラベルを一切保存しないことを覚えておくことが重要です。これらのラベルは、それぞれのイメージの識別子と一緒にデータベースに登録することで保存できます。

このチュートリアルでは、ラベルを検出し、検出したラベルをデータベースに保存する方法を説明します。このチュートリアルで開発されたサンプルアプリケーションは、Amazon S3 バケットからイメージを読み取り、そのイメージに対して DetectLabels オペレーションを呼び出し、結果のラベルをデータベースに保存します。アプリケーションは、使用したいデータベースタイプに応じて、Amazon RDS データベースインスタンスまたは DynamoDB データベースのいずれかにデータを保存します。

AWS SDK for Python またはこのチュートリアルを使用することになります。その他の Python チュートリアルについては、 AWS Documentation SDK サンプル GitHub リポジトリも参照してください。

トピック

- 前提条件
- Amazon S3 バケット内のイメージのラベルを取得する
- Amazon DynamoDB テーブルを作成する
- DynamoDB へのデータのアップロード
- Amazon RDS で MySQL データベースを作成する
- Amazon RDS の MySQL テーブルへのデータのアップロード

前提条件

このチュートリアルを開始する前に、Python をインストールし、Python AWS SDK のセットアップに必要な手順を完了する必要があります。これ以外にも、次のことを確認してください:

AWS アカウントと IAM ロールを作成しました

Python SDK (Boto3) をインストールしました

AWS アクセス認証情報を適切に設定する

Amazon S3 バケットを作成し、イメージで埋め尽くしました

RDS を使用してデータを保存する場合、RDS データベースインスタンスを作成した

Amazon S3 バケット内のイメージのラベルを取得する

まず、Amazon S3 バケットにあるイメージの名前を受け取り、そのイメージを取得する関数を作成します。このイメージは、正しいイメージが関数内の DetectLabels の呼び出しに渡されていることを確認するために表示されます。

1. 使用したい Amazon S3 バケットを見つけ、名前を書き込みます。この Amazon S3 バケットを呼び出して、その中のイメージを読み取ります。DetectLabels オペレーションに渡すイメージがバケットに含まれていることを確認します。
2. Amazon S3 バケットに接続するためのコードを書き込みます。Boto3 を使用して Amazon S3 リソースに接続して、Amazon S3 バケットからイメージを取得できます。Amazon S3 リソースに接続したら、バケットメソッドに Amazon S3 バケット名を指定して、バケットにアクセスできます。Amazon S3 バケットに接続後、Object メソッドでバケットからイメージを取得します。Matplotlib を利用することで、この接続を使用してイメージを可視化することができます。Boto3 は、Rekognition クライアントへの接続にも使用されます。

以下のコードで、リージョンを region_name パラメータに指定します。DetectLabels に Amazon S3 バケット名とイメージ名を渡すと、対応するイメージのラベルが返されます。レスポンスからラベルのみを選択すると、イメージ名とラベルの両方が返されます。

```
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. このコードを get_images.py というファイルに保存します。

Amazon DynamoDB テーブルを作成する

以下のコードでは、boto3 を使って DynamoDB に接続し、DynamoDB `CreateTable` メソッドを使って、Images という名前のテーブルを作成しています。テーブルには、Image というパーティションキーおよび Labels というソートキーで構成されている複合プライマリキーがあります。Image キーにはイメージの名前が含まれ、Labels キーにはそのイメージに割り当てられたラベルが保存されます。

```
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)
```

このコードをエディタに保存し、一度実行すると DynamoDB テーブルが作成されます。

DynamoDB へのデータのアップロード

DynamoDB データベースが作成され、イメージのラベルを取得する関数ができたので、ラベルを DynamoDB に保存することができます。次のコードは、S3 バケット内のすべてのイメージを取得し、それらのラベルを取得し、データを DynamoDB に保存します。

1. DynamoDB にデータをアップロードするためのコードを書く必要があります。get_image_names という関数で Amazon S3 バケットに接続し、バケット内のすべてのイメージの名前をリストとして返します。このリストを read_image_from_S3 関数に渡すと、作成した get_images.py ファイルからインポートされます。

   ```
   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. 先ほど作成した read_image_from_S3 関数は、処理中のイメージの名前と、そのイメージに関連するラベルのディクショナリを返します。find_values という関数は、レスポンスからラベルのみを取得するために使用されます。イメージの名前とそのラベルを DynamoDB テーブルにアップロードする準備が整いました。

   ```
   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. load_data という 3 番目の関数で、作成した DynamoDB テーブルにイメージとラベルを実際にロードします。

```
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. ここでは、前回定義した 3 つの関数が呼び出され、オペレーションが実行されます。上記で定義した 3 つの関数を、以下のコードとともに Python ファイルに追加します。コードを実行します。

```
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!")
```

DetectLabels を使用してイメージのラベルを生成し、そのラベルを DynamoDB インスタンスに保存します。このチュートリアルの間に作成したリソースはすべて取り外しておいてください。これにより、使用していないリソースに課金されることを防ぐことができます。

Amazon RDS で MySQL データベースを作成する

先に進む前に、Amazon RDS の セットアップ手順 が完了し、Amazon RDS を使用して MySQL DB インスタンスを作成した ことを確認してください。

次のコードは、PyMySQL ライブラリと Amazon RDS DB インスタンスを利用しています。イメージの名前とそのイメージに関連するラベルを保持するためのテーブルを作成します。Amazon RDS は、テーブルの作成とテーブルへのデータ挿入のコマンドを受け取ります。Amazon RDS を使用するには、ホスト名、ユーザーネーム、パスワードを使用して Amazon RDS ホストに接続する必要があります。これらの引数を PyMySQL の `connect` 関数に与え、カーソルのインスタンスを作成することで、 Amazon RDS に接続することになります。

1. 次のコードでは、ホストの値を Amazon RDS ホストエンドポイントに置き換え、ユーザーの値を Amazon RDS インスタンスに関連付けられたマスターユーザーネームに置き換えます。また、パスワードをメインユーザーのマスターパスワードに置き換える必要があります。

```
import pymysql

host = "host-endpoint"
user = "username"
password = "master-password"
```

2. データベースと、イメージとラベルデータを挿入するテーブルを作成します。作成クエリを実行し、コミットすることで行います。次のコードでは、データベースを作成します。このコードは 1 回だけ実行してください。

```
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. データベースを作成したら、イメージ名とラベルを挿入するテーブルを作成する必要があります。テーブルを作成するには、まず use SQL コマンドとデータベースの名前を execute 関数に渡します。接続が完了すると、テーブルを作成するためのクエリが実行されます。次のコードは、データベースに接続し、image_id というプライマリキーと、ラベルを保存するテキスト属性の両方を持つテーブルを作成します。先ほど定義したインポートと変数を使用し、このコードを実行してデータベースにテーブルを作成します。

```
# 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!")
```

Amazon RDS の MySQL テーブルへのデータのアップロード

Amazon RDS データベースとデータベースにテーブルを作成した後、イメージのラベルを取得し、それらのラベルを Amazon RDS データベースに保存することができます。

1. Amazon S3 バケットに接続し、バケット内のすべてのイメージの名前を取得します。これらのイメージ名は、先に作成した read_image_from_s3 関数に渡され、すべてのイメージのラベルを取得します。次のコードは Amazon S3 バケットに接続し、バケット内のすべてのイメージのリストを返します。

```
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. DetectLabels API からのリスポンスにはラベル以外のものも含まれているため、ラベルの値のみを抽出する関数を記述するようにしてください。次の関数は、ラベルだけを集めたリストを返します。

```
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. イメージ名とラベルをテーブルに挿入する関数が必要です。次の関数は、挿入クエリを実行し、与えられたイメージ名とラベルのペアを挿入します。

```
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. 最後に、上で定義した関数を実行する必要があります。次のコードでは、バケット内のイメージすべての名前が収集され、DetectLabels を呼び出す関数に提供されます。その後、ラベルと

ラベルが適用されるイメージの名前が Amazon RDS データベースにアップロードされます。上記で定義した 3 つの関数を、以下のコードとともに Python ファイルにコピーします。Python ファイルを実行します。

```
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!")
```

DetectLabels を使用してイメージのラベルを生成し、Amazon RDS を使用して MySQL データベースにそのラベルを保存しました。このチュートリアルの間に作成したリソースはすべて取り外しておいてください。これにより、使用していないリソースに課金されることを防ぐことができます。

その他の AWS マルチサービスの例については、 AWS 「Documentation SDK examples GitHub repository」を参照してください。

Amazon Rekognition と Lambda を使用して Amazon S3 バケットのアセットにタグを付ける

このチュートリアルでは、Amazon S3 バケットにあるデジタルアセットに自動的にタグ付けする AWS Lambda 関数を作成します。Lambda 関数は、指定された Amazon S3 バケット内のすべてのオブジェクトを読み取ります。バケット内の各オブジェクトについて、イメージを Amazon Rekognition サービスに渡して、一連のラベルを生成します。各ラベルは、イメージに適用されるタグを作成するために使用されます。Lambda 関数を実行すると、指定された Amazon S3 バケット内のすべてのイメージに基づいてタグが自動的に作成され、イメージに適用されます。

例えば、Lambda 関数を実行し、このイメージが Amazon S3 バケット内にあると仮定します。



その後、アプリケーションは自動的にタグを作成し、イメージにそれらを適用します。

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**
>
> このチュートリアルで使用するサービスは、 AWS 無料利用枠の一部です。チュートリアルが終了したら、課金されないように、チュートリアル中に作成したリソースをすべて終了することをお勧めします。

このチュートリアルでは、 AWS SDK for Java バージョン 2 を使用します。追加の Java V2 チュートリアルについては、AWS ドキュメント SDK の例 GitHub リポジトリ を参照してください。

トピック

- 前提条件
- IAM Lambda ロールを設定する
- プロジェクトの作成
- コードを書き込む

- プロジェクトをパッケージ化する
- Lambda 関数をデプロイします。
- Lambda メソッドをテストする

前提条件

開始する前に、AWS 「SDK for Java のセットアップ」の手順を完了する必要があります。次に、以下があることを確認します。

- Java 1.8 JDK。
- Maven 3.6 以上
- ネイチャーイメージが入った Amazon S3 バケット 5-7 これらのイメージは Lambda 関数によって読み取られます。

IAM Lambda ロールを設定する

このチュートリアルでは、Amazon Rekognition および Amazon S3 サービスを使用します。Lambda 関数からこれらのサービスを呼び出すことを可能にするポリシーを持つために、lambda-support ロールを設定する。

ロールを設定するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. ナビゲーションペインで、[Roles (ロール)]、[ロールの作成] の順に選択します。
3. [AWS サービス]、[Lambda] の順に選択します。
4. [アクセス許可] タブを選択します。
5. AWSLambdaBasicExecutionRole を検索する。
6. [次のタグ] を選択します。
7. [Review] (レビュー) を選択します。
8. そのロールに lambda-support という名前を付けます。
9. [ロールの作成] を選択します。
10. lambda-support を選択して、概要ページを表示します。

11. [ポリシーのアタッチ] を選択します。
12. ポリシーのリストから [AmazonRekognitionFullAccess] を選択します。
13. Attach policy] (ポリシーのアタッチ) を選択してください。
14. AmazonS3FullAccess を検索してから、添付ポリシー を選択します。

プロジェクトの作成

新しい Java プロジェクトを作成し、必要な設定と依存関係に必須の Maven pom.xml を構成します。pom.xml ファイルが次のようになっていることを確認します。

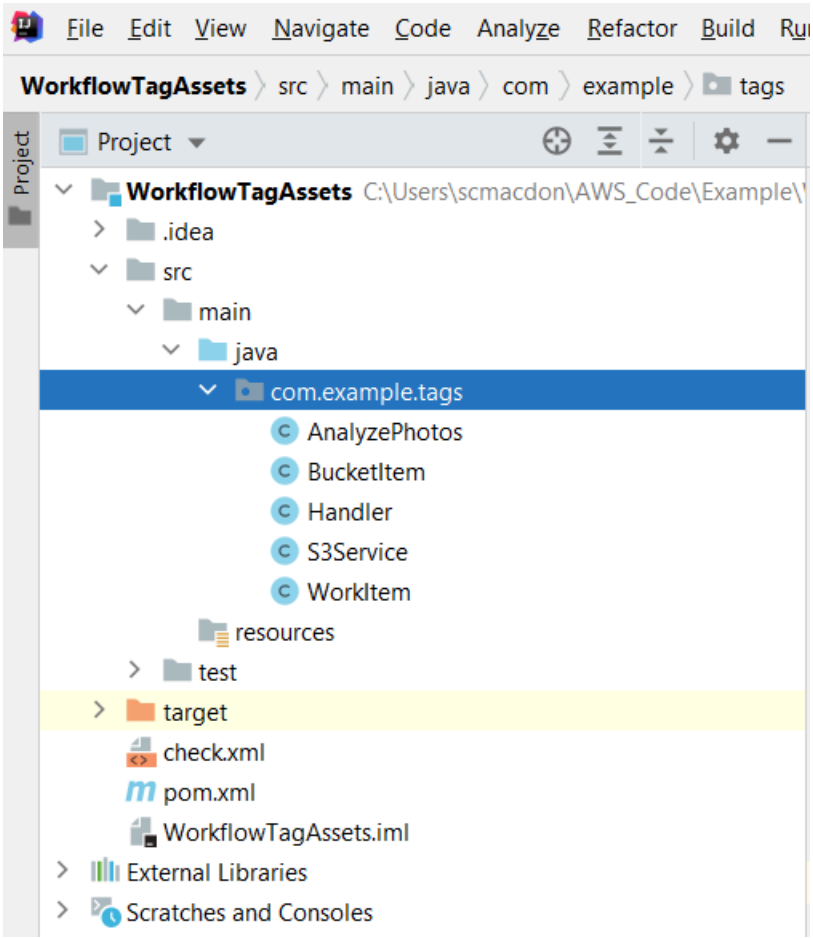
```
<?xml version="1.0" encoding="UTF-8"?>
 <project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.example</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>
```

コードを書き込む

AWS Lambda ランタイム Java API を使用して、Lambda 関数を定義する Java クラスを作成します。この例では、ハンドラー という名前の Lambda 関数用の Java クラスが 1 つあり、このユースケースには追加のクラスが必要です。次の図は、プロジェクトの Java クラスを示します。すべての Java クラスが、com.example.tags という名前のパッケージに配置されていることに注目してください。



コード用に次の Java クラスを作成します。

- ハンドラーは Lambda Java ランタイム API を使用し、この AWS チュートリアルで説明するユースケースを実行します。実行されるアプリケーションロジックは、handleRequest メソッドにあります。
- S3Service は、Amazon S3 API を使用して S3 オペレーションを実行します。
- AnalyzePhotos では、Amazon Rekognition API を使用してイメージを分析します。
- バケット項目 は、Amazon S3 バケット情報を格納するモデルを定義します。
- 作業項目 は Amazon Rekognition データを格納するモデルを定義します。

ハンドラークラス

この Java コードは、Handler のクラスを表します。そのクラスは、Lambda 関数に渡されるフラグを読み取ります。s3Service.listBucketObjects メソッドは、各要素がオブジェクトキーを表す文字列値である リスト オブジェクトを返します。フラグの値が true の場合、リストを繰り返し処理し、s3Service.tagAssets メソッドを呼び出して、タグを各オブジェクトに適用することで、タグが適用されます。フラグの値が false の場合、タグを削除する s3Service.DeleteTagfromObject のメソッドが呼び出されます。また、LambdaLogger オブジェクトを使用して、Amazon CloudWatch のログにメッセージを記録できることに注目してください。

> **Note**
>
> バケット名をbucketName 変数に指定することを確認してください。

```
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;
 }
}
```

S3 サービスクラス

次のクラスは、Amazon S3 API を使用して S3 オペレーションを実行します。例えば、getObjectBytes メソッドは、イメージを表すバイト配列を返します。同様に、listBucketObjects メソッドは、各要素がキー名を指定する文字列値である リスト オブジェクトを返します。

```
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 クラス

次の Java コードは、AnalyzePhotos のクラスを表します。このクラスは Amazon Rekognition API を使用して、イメージを分析します。

```
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 ;
  }
}
```

バケット項目のクラス

次の Java コードは、Amazon S3 オブジェクトデータを格納する バケット項目 のクラスを表します。

```
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 ;
 }
}
```

作業項目のクラス

次の Java コードは、作業項目 のクラスを表します。

```
 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;
}

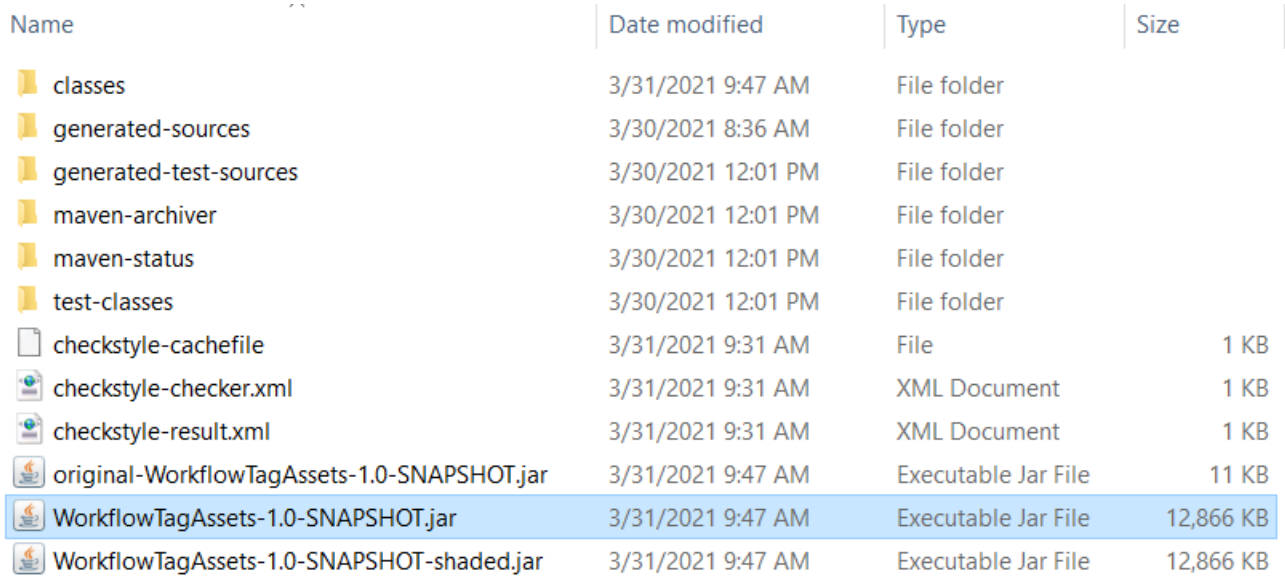
}
```

プロジェクトをパッケージ化する

次の Maven コマンドを使用して、プロジェクトを.jar (JAR) ファイルにパッケージ化します。

```
mvn package
```

JAR ファイルは、ターゲット フォルダ (プロジェクトフォルダの子フォルダ) の中にあります。



Name	Date modified	Type	Size
classes	3/31/2021 9:47 AM	File folder	
generated-sources	3/30/2021 8:36 AM	File folder	
generated-test-sources	3/30/2021 12:01 PM	File folder	
maven-archiver	3/30/2021 12:01 PM	File folder	
maven-status	3/30/2021 12:01 PM	File folder	
test-classes	3/30/2021 12:01 PM	File folder	
checkstyle-cachefile	3/31/2021 9:31 AM	File	1 KB
checkstyle-checker.xml	3/31/2021 9:31 AM	XML Document	1 KB
checkstyle-result.xml	3/31/2021 9:31 AM	XML Document	1 KB
original-WorkflowTagAssets-1.0-SNAPSHOT.jar	3/31/2021 9:47 AM	Executable Jar File	11 KB
WorkflowTagAssets-1.0-SNAPSHOT.jar	3/31/2021 9:47 AM	Executable Jar File	12,866 KB
WorkflowTagAssets-1.0-SNAPSHOT-shaded.jar	3/31/2021 9:47 AM	Executable Jar File	12,866 KB

> **Note**
>
> プロジェクトの POM ファイル内にある maven-shade-plugin の使用に注目してください。このプラグインは、必要な依存関係を含む JAR を作成する役割を担います。このプラグインなしでプロジェクトをパッケージ化しようとすると、必要な依存関係が JAR ファイル内に含まれず、ClassNotFoundException に直面します。

Lambda 関数をデプロイします。

1. Lambdaのコンソールを開きます。
2. [関数を作成] を選択します。
3. Author from scratch (製作者を最初から) を選択します。
4. 基本的な情報 セクションに、名前として cron と入力します。
5. [Runtime] で、[Java 8] を選択します。
6. [Use an existing role (既存のロールの使用)] を選択し、lambda-support (作成した IAM ロール) を選択します。
7. [関数を作成] を選択します。
8. [コードエントリタイプ] で、[.ZIP またはファイルをアップロード] を選択します。
9. アップロード を選択し、作成した JAR ファイルを参照します。

10ハンドラー を使用する場合、関数の完全修飾名を入力します。例えば、com.example.tags.Handler:handleRequest (com.example.tags はパッケージを指定し、ハンドラー は :: およびメソッド名が続くクラスです) 。

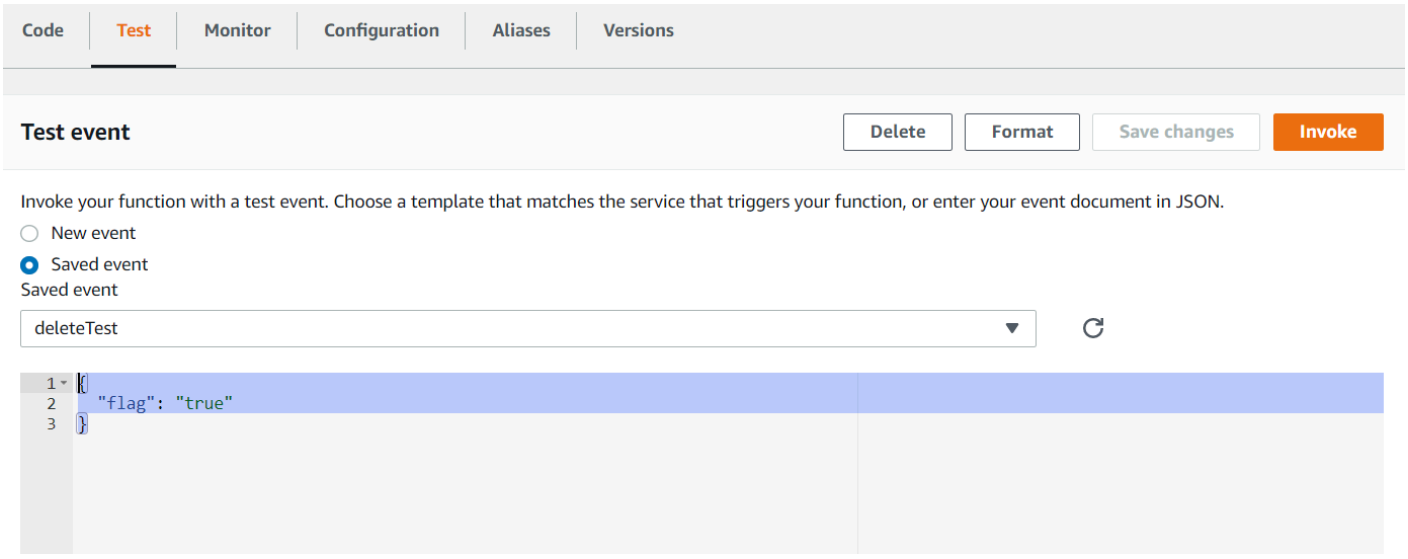
11[Save] を選択します。

Lambda メソッドをテストする

チュートリアルのこの時点で、Lambda 関数をテストできます。

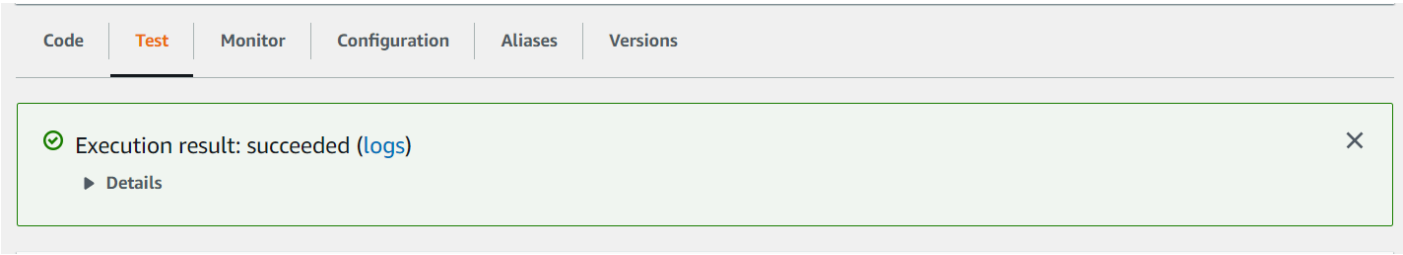
1. Lambda コンソールで、Test タブをクリックし、次の JSON を入力します。

```
                {
"flag": "true"
 }
```



> **Note**
>
> true を送るとデジタルアセットにタグを付け、false を送るとタグを削除します。

2. [呼び出し ボタンを選択します。Lambda 関数が呼び出されると、成功したメッセージが表示されます。



これで、Amazon S3 バケットにあるディフィラルアセットにタグを自動的に適用する AWS Lambda 関数が作成されました。このチュートリアルの冒頭で説明したように、必ずこのチュートリアルの実行中に作成したリソースをすべて終了し、課金されないようにしてください。

その他の AWS マルチサービスの例については、AWS 「Documentation SDK examples GitHub repository」を参照してください。

AWS ビデオアナライザーアプリケーションの作成

AWS SDK for Java バージョン 2 を使用して、ビデオを分析してラベルを検出するための Java ウェブアプリケーションを作成できます。この AWS チュートリアルで作成したアプリケーションでは、Amazon S3 バケットに動画 (MP4 ファイル) をアップロードできます。 Amazon S3 そして、Amazon Rekognition サービスを使用して、ビデオを分析します。結果をもとにデータモデルを作成し、Amazon Simple Email Service を利用してレポートを作成し、特定のユーザーへメール送信します。

下図は、アプリケーションがビデオの分析を完了した後に生成されるレポートを示しています。以下の表には、年齢層、ひげの有無、眼鏡の有無、目の開閉の列があり、属性予測のさまざまな信頼値が表示されています。

	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				

このチュートリアルでは、さまざまな AWS サービスを呼び出す Spring Boot アプリケーションを作成します。Spring Boot API を使用して、モデル、さまざまなビュー、コントローラーを構築します。詳細については、「Spring Boot」を参照してください。

このサービスは、次の AWS サービスを使用します。

- Amazon Rekognition
- Amazon S3
- Amazon SES
- AWS Elastic Beanstalk

このチュートリアルに含まれる AWS サービスは、 AWS 無料利用枠に含まれています。チュートリアルで作成したリソースは、課金されないように、作成が終了したらすべて終了させることをお勧めします。

前提条件

開始する前に、AWS 「SDK for Java のセットアップ」の手順を完了する必要があります。次に、以下があることを確認します。

- Java 1.8 JDK。
- Maven 3.6 以降。
- ビデオ [someValue] という名前の Amazon S3 バケット。Amazon S3 Java コードでは、このバケット名を必ず使用してください。詳細については、「バケットの作成」を参照してください。
- IAM ロール。これから作成する VideoDetectFaces クラスで必要です。詳細については、「Amazon Rekognition Video を設定する」を参照してください。
- 有効な Amazon SNS トピック。これから作成する VideoDetectFaces クラスで必要です。詳細については、「Amazon Rekognition Video を設定する」を参照してください。

手順

チュートリアルのコースで、以下のことを行います。

1. プロジェクトを作成する
2. POM の依存関係をプロジェクトに追加する
3. Java クラスを作成する
4. HTML ファイルを作成する
5. スクリプトファイルを作成する
6. プロジェクトを JAR ファイルにパッケージ化する
7. アプリケーションを にデプロイする AWS Elastic Beanstalk

チュートリアルを進めるには、AWS ドキュメント SDK の例 GitHub リポジトリ の詳細な説明に従ってください。

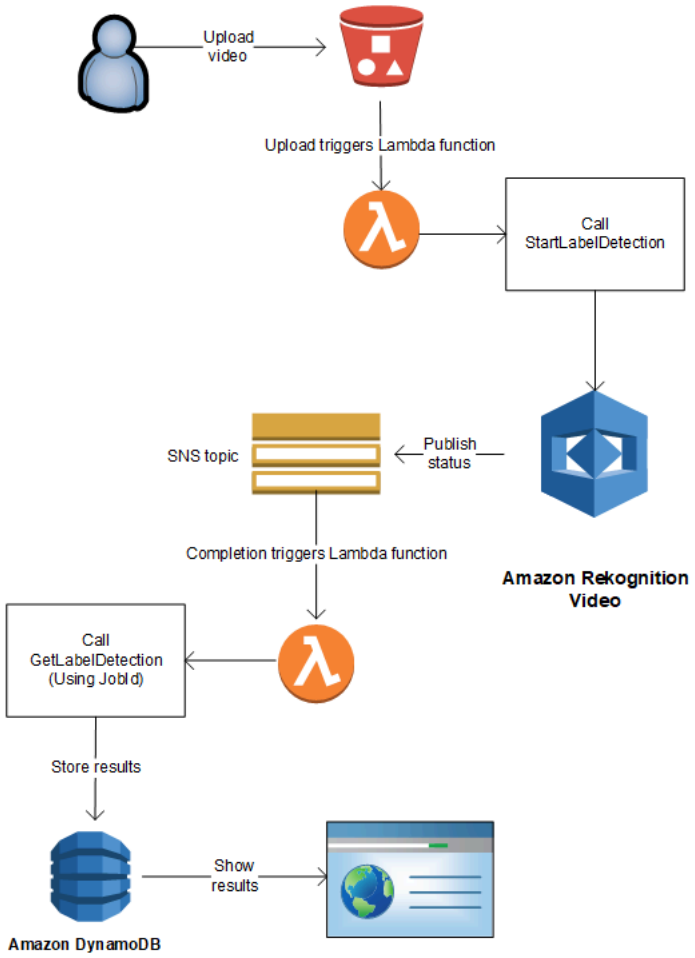
Amazon Rekognition Lambda 関数の作成

このチュートリアルでは、Java Lambda 関数を使用して、ラベルの検出用にビデオ分析オペレーションの結果を取得する方法を示します。

> **Note**
>
> このチュートリアルでは、 AWS SDK for Java 1.x を使用します。Rekognition と AWS SDK for Java バージョン 2 を使用するチュートリアルについては、AWS 「Documentation SDK examples GitHub repository」を参照してください。

Amazon Rekognition Video オペレーションで Lambda 関数を使用できます。たとえば、以下の図に示しているウェブサイトは、Lambda 関数を使用して、Amazon S3 バケットにアップロードされたビデオの分析を自動的に開始します。Lambda 関数がトリガーされると、StartLabelDetection を呼び出して、アップロードされたビデオのラベルの検出を開始します。Lambda を使用して Amazon S3 バケットからのイベント通知を処理する方法については、「Amazon S3 イベント　で AWS Lambda の使用」を参照してください。

分析の完了ステータスが登録済み Amazon SNS トピックに送信されると、2 番目の Lambda 関数がトリガーされます。2 番目の Lambda 関数は GetLabelDetection を呼び出して分析結果を取得します。取得した結果はウェブページに表示するためにデータベースに保存されます。この 2 番目のラムダ関数は、このチュートリアルの焦点です。



このチュートリアルでは、Amazon Rekognition Video によってビデオ分析の完了ステータスが登録済み Amazon SNS トピックに送信されると、 Lambda 関数がトリガーされます。この関数は GetLabelDetection を呼び出して、ビデオの分析結果を収集します。デモの目的で、このチュートリアルでは、ラベル検出結果を CloudWatch ログに書き込みます。アプリケーションの Lambda 関数では、後で使用するために分析結果を保存するようにします。たとえば、Amazon DynamoDB を使用して分析結果を保存できます。詳細については、「DynamoDB の使用」を参照してください。

ここでは、以下の手順を示します。

- Amazon SNS トピックを作成し、アクセス権限を設定する。
- を使用して Lambda 関数を作成し AWS Management Console 、Amazon SNS トピックにサブスクライブします。
- AWS Management Consoleを使用して Lambda 関数を設定します。
- サンプルコードを AWS Toolkit for Eclipse プロジェクトに追加し、Lambda 関数にアップロードします。

- AWS CLIを使用して Lambda 関数をテストする。

> **Note**
>
> チュートリアル全体で同じ AWS リージョンを使用します。

前提条件

このチュートリアルでは、 AWS Toolkit for Eclipseに精通していることを前提としています。詳細については、「AWS Toolkit for Eclipse」を参照してください。

SNS トピックを作成する

Amazon Rekognition Video ビデオ分析オペレーションの完了ステータスは、Amazon SNS トピックに送信されます。この手順では、Amazon SNS トピックと Amazon Rekognition Video にその Amazon SNS トピックへのアクセスを許可する IAM サービスロールを作成します。詳細については、「Amazon Rekognition Video オペレーションを呼び出す」を参照してください。

Amazon SNS トピックを作成するには

1. まだ作成していない場合は、IAM サービスロールを作成して、Amazon Rekognition Video に Amazon SNS トピックへのアクセスを許可します。Amazon リソースネーム (ARN) を記録しておきます。詳細については、「複数の Amazon SNS トピックへのアクセスを許可する」を参照してください。
2. Amazon SNS コンソール を使用して Amazon SNS トピック を作成する。トピック名を指定するだけで済みます。トピック名の先頭に AmazonRekognition を追加します。トピックの ARN を書き留めておきます。

Lambda 関数を作成する

AWS Management Consoleを使用して Lambda 関数を作成できます。次に、 AWS Toolkit for Eclipse プロジェクトを使用して、Lambda 関数パッケージを AWS Lambdaにアップロードします。 AWS Toolkit for Eclipseを使用して Lambda 関数を作成することもできます。詳細については、「チュートリアル: AWS Lambda 関数の作成、アップロード、呼び出し方法」を参照してください。

Lambda 関数を作成するには

1. AWS マネジメントコンソールにサインインして AWS Lambda コンソールを https://console.aws.amazon.com/lambda/ で開きます。
2. [関数を作成] を選択します。
3. [一から作成] を選択します。
4. [Function name] に、関数の名前を入力します。
5. [Runtime] で、[Java 8] を選択します。
6. [Choose or create an execution role] を選択します。
7. [Execution role] で、[Create a new role with basic Lambda permissions] を選択します。
8. [Basic information] セクションの下部に表示される新しいロールの名前をメモします。
9. [関数を作成] を選択します。

Lambda 関数を設定

Lambda 関数を作成したら、「SNS トピックを作成する」で作成した Amazon SNS トピックによってトリガーされるように、その関数を設定します。また、Lambda 関数のメモリ要件とタイムアウト期間も調整します。

Lambda 関数を設定

1. [関数コード] に、[`com.amazonaws.lambda.demo.JobCompletionHandler`] として「ハンドラ」と入力します。
2. [Basic settings] で、[Edit] を選択します。[Edit basic settings] ダイアログが表示されます。
   a. メモリ で 1024 を選択します。
   b. [タイムアウト] で、[10]秒 を選択します。
   c. [保存] を選択します。
3. [Designer] で、[Add trigger] を選択します。[Add trigger] ダイアログが表示されます。
4. [Trigger configuration] で、[SNS] を選択します。

   SNS トピックで、「SNS トピックを作成する」で作成した Amazon SNS トピックを選択します。
5. [トリガーの有効化] を選択します。
6. トリガーを追加するには、[追加] を選択します。

7. [保存] を選択して、更新された Lambda 関数を保存します。

IAM Lambda ロールを設定する

Amazon Rekognition Video オペレーションを呼び出すには、AmazonRekognitionFullAccess AWS マネージドポリシーを IAM Lambda ロールに追加します。StartLabelDetection などの開始オペレーションでは、Amazon Rekognition Video が Amazon SNS トピックへのアクセスに使用する IAM サービスロールに、ロールの引き渡し権限も必要です。

ロールを設定するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. ナビゲーションペインで Roles (ロール) を選択してください。
3. リストで、「Lambda 関数を作成する」で作成した実行ロールの名前を選択します。
4. [Permissions] (許可) タブを選択します。
5. [ポリシーのアタッチ] を選択します。
6. ポリシーのリストから [AmazonRekognitionFullAccess] を選択します。
7. Attach policy] (ポリシーのアタッチ) を選択してください。
8. 再度、実行ロールを選択します。
9. [Add inline policy] (インラインポリシーの追加) を選択します。
10. [JSON] タブを選択します。
11. 既存のポリシーを以下のポリシーに置き換えます。`servicerole` を、「SNS トピックを作成する」で作成した IAM サービスロールに置き換えます。

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

12. [ポリシーの確認] を選択します。
13. [Name*] (名前*) にポリシーの名前を入力します。
14. [Create policy] (ポリシーの作成) を選択します。

AWS Toolkit for Eclipse Lambda プロジェクトを作成

Lambda 関数がトリガーされると、以下のコードが Amazon SNS トピックから完了ステータスを取得し、GetLabelDetection を呼び出して分析結果を取得します。検出されたラベルの数と、検出されたラベルのリストが CloudWatch log に書き込まれます。Lambda 関数では、後で使用するためにビデオ分析結果を保存するようにします。

AWS Toolkit for Eclipse Lambda プロジェクトを作成するには

1. AWS Toolkit for EclipseAWS Lambda プロジェクトを作成します。
   - [プロジェクト名:] に、選択したプロジェクトの名前を入力します。
   - [Class Name: (クラス名:) に、「JobCompletionHandler」と入力します。
   - [Input type:] (入力タイプ:) で、[SNS Event] (SNS イベント) を選択します。
   - 他のフィールドはそのままにしておきます。
2. [Eclipse プロジェクト] エクスプローラーで、生成された Lambda ハンドラメソッド (JobCompletionHandler.java) を開き、その内容を以下のように置き換えます。

```
//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(pagination

       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. Rekognition 名前空間は解決されません。この問題を修正するには:
   - `import com.amazonaws.services.rekognition.AmazonRekognition;` 行の下線部分にマウスを置きます。
   - [Fix project set up...] (プロジェクト設定の修正...) を選択します。
   - 最新バージョンの Amazon Rekognition アーカイブを選択します。
   - [OK] を選択して、アーカイブをプロジェクトに追加します。
4. ファイルを保存します。
5. Eclipse コードウィンドウ内で右クリックし、[AWS Lambda] を選択して、[Upload function to AWS Lambda] を選択します。
6. [Select Target Lambda Function] ページで、使用する AWS リージョンを選択します。
7. [Choose an existing lambda function] (既存の Lambda 関数の選択) を選択してから、「Lambda 関数を作成する」で作成した Lambda 関数を選択します。
8. [次へ] を選択します。[Function Configuration] ダイアログボックスが表示されます。
9. [IAM Role] で、「Lambda 関数を作成する」で作成した IAM ロールを選択します。
10. [完了] を選択すると、Lambda 関数が AWSにアップロードされます。

Lambda 関数をテストする

次の AWS CLI コマンドを使用して、ビデオのラベル検出分析を開始して Lambda 関数をテストします。分析が終了すると、Lambda 関数がトリガーされます。CloudWatch Logs ログを確認して、分析が成功したことを確認します。

Lambda 関数をテストする

1. S3 バケットに MOV あるいは MPEG-4 形式のビデオファイルをアップロードします。テストの場合、長さが 30 秒以内のビデオをアップロードしてください。

   手順については、[Amazon Simple Storage Service ユーザーガイド] の [Amazon S3 へのオブジェクトのアップロード] を参照してください。
2. 次の AWS CLI コマンドを実行して、ビデオ内のラベルの検出を開始します。

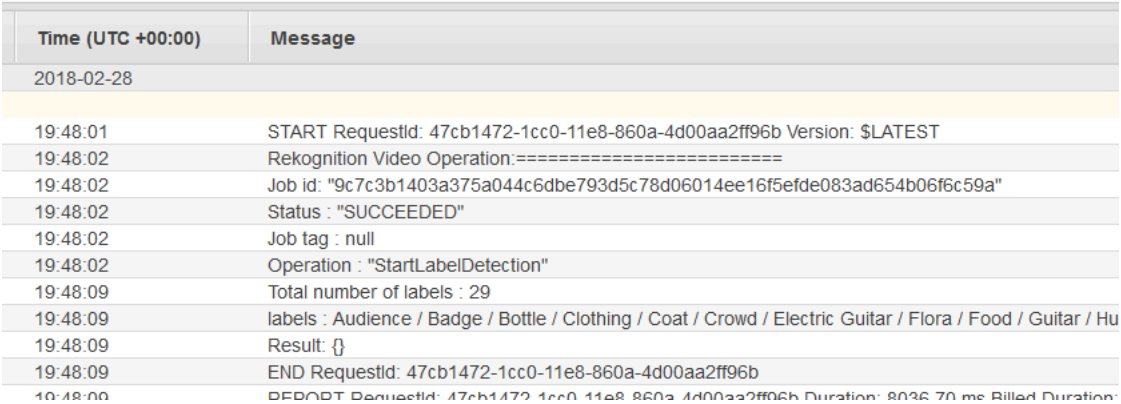
   ```
   aws rekognition start-label-detection --video
    "S3Object={Bucket="bucketname",Name="videofile"}" \
   --notification-channel "SNSTopicArn=TopicARN,RoleArn=RoleARN" \
   --region Region
   ```

   以下の値を更新します。

   - bucketname と videofile を、Amazon S3 バケットの名前と、ラベルを検出する対象のビデオの名前に変更します。
   - TopicARN を、「SNS トピックを作成する」で作成した Amazon SNS トピックの ARN に変更します。
   - RoleARN を、「SNS トピックを作成する」で作成した IAM ロールの ARN に変更します。
   - を、使用している AWS リージョンRegionに変更します。
3. 応答内の JobId の値に注意してください。この応答は次の JSON の例のようになります。

   ```
   {
       "JobId": "547089ce5b9a8a0e7831afa655f42e5d7b5c838553f1a584bf350ennnnnnnnnn"
   }
   ```
4. コンソール https://console.aws.amazon.com/cloudwatch/ を開きます。
5. 分析が完了すると、Lambda 関数のログエントリが [ロググループ]に表示されます。
6. Lambda 関数を選択すると、ログストリームが表示されます。

7. 最新のログストリームを選択すると、Lambda 関数によって作成されたログエントリが表示されます。オペレーションが成功した場合、出力は以下のようになります。ジョブ ID、オペレーションタイプ「StartLabelDetection」、検出されたラベルのカテゴリ (「Bottle」、「Clothing」、「Crowd」、「Food」など) のリストといった、ビデオ認識オペレーションの詳細が示されています。



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
19:48:09	REPORT RequestId: 47cb1472-1cc0-11e8-860a-4d00aa2ff96b Duration: 8036.70 ms Billed Duration:

[Job id] (ジョブ ID) の値は、手順 3 で記録した `JobId` の値と一致しています。

本人確認における Amazon Rekognition の使用

Amazon Rekognition には、本人確認のシステムを簡単に作成できるオペレーションが複数用意されています。Amazon Rekognition を使用すれば、ユーザーはイメージから顔を検出して、検出された顔と他の顔とでデータどうしを比較し、顔を比較できます。この顔データは「コレクション」と呼ばれるサーバー側のコンテナに保存されます。Amazon Rekognition の顔検出、顔比較、コレクション管理の各オペレーションを活用することで、本人確認の機能を備えたアプリケーションを作成できます。

このチュートリアルでは、本人確認が必要なアプリケーションを作成するための、一般的な 2 つのワークフローを紹介します。

1 つ目のワークフローは、新規ユーザーをコレクションに登録する作業を伴います。2 つ目のワークフローは、リピーターをログインさせるために既存のコレクションを検索する作業を伴います。

このチュートリアルでは AWS SDK for Python を使用します。その他の Python チュートリアルについては、 AWS Documentation SDK サンプル GitHub リポジトリも参照してください。

トピック

- 前提条件

- コレクションの作成
- 新規ユーザー登録
- 既存ユーザーのログイン

前提条件

このチュートリアルを開始する前に、Python をインストールし、Python AWS SDK のセットアップに必要な手順を完了する必要があります。これ以外にも、次のことを確認してください:

- AWS アカウントと IAM ロールを作成しました
- Python SDK (Boto3) をインストールしました
- AWS アクセス認証情報を適切に設定する
- Amazon Simple Storage Service バケットを作成し、本人確認で ID として使用するイメージをアップロードしました
- 本人確認のターゲットイメージとして使用する 2 つ目のイメージを選択しました

コレクションの作成

コレクションに新規ユーザーを登録する、またはコレクションでユーザーを検索する前に、作業を行うコレクションを作成しておく必要があります。Amazon Rekognition コレクションとはサーバー側のコンテナであり、検出した顔に関する情報はここに保存されます。

コレクションを作成する

最初に、自分のアプリケーションで使用するコレクションを作成するための、関数を作成します。Amazon Rekognition では、検出した顔に関する情報は、コレクションと呼ばれるサーバー側のコンテナに保存されます。既知の顔に関しては、コレクションに保存された顔情報を検索できます。顔情報を保存するには、まず、`CreateCollection` オペレーションを使ってコレクションを作成する必要があります。

1. 作成するコレクションの名前を選択します。以下のコードで、`collection_id` の値を、作成するコレクションの名前に置き換え、`region` の値を、ユーザー認証情報で定義されているリージョンの名前に置き換えます。`Tags` 引数を使用すると、任意のタグをコレクションに適用できますが、必須ではありません。`CreateCollection` オペレーションは、コレクションの Arn など、作成したコレクションに関する情報を返します。コードを実行した後に受け取った Arn は、書き留めておきます。

```
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. コードを保存して実行します。コレクション Arn を書き留めておきます。

   Rekognition コレクションを作成できました。これで、顔の情報や ID をこのコレクションに保存できます。また、顔を、保存した情報と比較して検証することもできます。

新規ユーザー登録

次に、新規ユーザーをコレクションに登録し、その情報を追加します。新規ユーザーを登録するプロセスは、通常は以下のステップに従います。

DetectFaces オペレーションを呼び出す

DetectFaces オペレーションを使って顔イメージの画質をチェックするための、コードを記述します。DetectFaces オペレーションを使って、カメラで撮影したイメージが SearchFacesByImage オペレーションによる処理に適しているかどうかを判断します。イメージに映る顔は 1 つでなければなりません。DetectFaces オペレーションでローカルの入力イメージファイルを指定し、イメージ内で検出された顔の詳細を受け取ります。次のコード例では、DetectFaces で入力イメージを指定し、イメージ内で検出された顔が 1 つのみであるかどうかを確認します。

1. 以下のコード例で、photo を、検出しようとしている顔のターゲットイメージの名前に置き換えます。また、region の値は、アカウントに関連付けられているリージョンの名前に置き換える必要があります。

```
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. 上のコードを保存して実行します。

CompareFaces オペレーションを呼び出す

アプリケーションでは、コレクションに新規ユーザーを登録し、リピーターが同一人物であることを確認できる必要があります。そのためには、最初に、新規ユーザーの登録に使用する関数を作成します。まず、CompareFaces オペレーションを使って、ユーザーのローカルの入力イメージ/ターゲットイメージと、ID/保存済みイメージを比較します。両方のイメージから検出された顔が一致している場合は、コレクション内を検索し、このユーザーが既に登録されているかどうかを確認できます。

まず、入力イメージを、Amazon S3 バケットに保存した ID イメージと比較する関数を作成します。以下のコード例では、入力イメージを自分で用意する必要があります。このイメージは、何らかの方法でライブネス検出を行った後に、撮影する必要があります。また、Amazon S3 バケットに保存されているイメージの名前も渡す必要があります。

1. bucket の値を、ソースファイルが保存されている Amazon S3 バケットの名前に置き換えます。また、source_file の値は、使用しているソースイメージの名前に置き換える必要があります。target_file の値を、指定したターゲットファイルの名前に置き換えます。region の値を、ユーザー認証情報で定義されている region の名前に置き換えます。

   また、similarityThreshold 引数を使用して、レスポンスで返すマッチレベルの最小信頼度も指定します。検出された顔は、信頼度がこのしきい値を超えた場合にのみ、FaceMatches 配列に返されます。選択した similarityThreshold は、ユーザー固有のユースケースの性質を反映しているものでなければなりません。ユースケースが重大なセキュリティアプリケーションに関係する場合、選択するしきい値は 99 を使用します。

```
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. 上のコードを保存して実行します。

   一致した顔と、信頼度に関する情報を含む、レスポンスオブジェクトが返されます。

SearchFacesByImage オペレーションを呼び出す

CompareFaces オペレーションの信頼度が、選択した SimilarityThreshold を上回った場合、入力イメージに一致する可能性のある顔をコレクション内で検索します。コレクションで一致する顔が見つかった場合、そのユーザーは既にコレクションに登録されている可能性が高いため、新規ユーザーとして登録する必要はありません。一致する顔が見つからなければ新規ユーザーとしてコレクションに登録できます。

1. まず、SearchFacesByImage オペレーションを呼び出すコードを作成します。このオペレーションは、ローカルのイメージファイルを引数として受け取り、提供されたイメージの中で最も多く検出された顔と一致する顔を Collection で検索します。

   以下のコード例で、collectionId の値を、検索するコレクションに変更します。region の値を、アカウントに関連付けられたリージョンの名前に置き換えます。また、photo の値を、入力ファイルの名前に置き換える必要があります。さらに、threshold の値を、選択したパーセンタイルに置き換えて、類似度のしきい値を指定します。

```
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. 上のコードを保存して実行します。一致する顔が見つかった場合、イメージで認識された人物は既にコレクションに含まれているため、次のステップに進む必要はありません。この場合は、ユーザーにアプリケーションへのアクセスを許可するだけで済みます。

IndexFaces オペレーションを呼び出す

検索したコレクションの中に一致する顔が見つからなければ、そのユーザーの顔をコレクションに追加する必要があります。顔を追加するには、IndexFaces オペレーションを呼び出します。IndexFaces を呼び出すと、入力イメージで識別された顔の特徴が Amazon Rekognition によって抽出され、そのデータが指定したコレクションに保存されます。

1. まず、IndexFaces を呼び出すコードを作成します。image の値を、IndexFaces オペレーションの入力イメージとして使用するローカルファイルの名前に置き換えます。また、photo_name の値を、入力イメージの名前に置き換える必要があります。collection_id の値を、以前に作成したコレクションの ID に置き換えます。次に、region の値を、アカウントに関連付けられたリージョンの名前に置き換えます。また、MaxFaces 入力パラメータの値を指定します。このパラメータは、インデックスを作成する必要がある、イメージ内の顔の最大数を定義します。このパラメータのデフォルト値は 1 です。

```
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. 上のコードを保存して実行します。イメージ内の人物に割り当てられた FaceID など、IndexFaces オペレーションによって返されるデータを保存するかどうかを決定します。このデータを保存する方法は、次のセクションで解説します。次に進む前に、返された FaceId、ImageId、Confidence の値を書き留めておきます。

イメージと FaceID のデータを Amazon S3 と Amazon DynamoDB に保存する

入力イメージの Face ID を取得すると、イメージデータを Amazon S3 に保存し、顔データとイメージ URL を DynamoDB などのデータベースに入力できるようになります。

1. 入力イメージを Amazon S3 データベースにアップロードするためのコードを作成します。以下のコード例で、bucket の値を、ファイルのアップロード先となるバケットの名前に置き換え、file_name の値を、Amazon S3 バケットに保存するローカルファイルの名前に置き換えます。key_name の値を、イメージファイルに付ける名前に置き換えることにより、Amazon S3 バケット内のファイルを識別するキー名を指定します。アップロードするファイルは、前にコード例で定義したファイルと同じファイルで、IndexFaces に使用した入力ファイルです。最後に、region の値を、アカウントに関連付けられたリージョンの名前に置き換えます。

```
import boto3
import logging
from botocore.exceptions import ClientError

# store local file in S3 bucket
bucket = "amzn-s3-demo-bucket"
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. 上のコード例を保存して実行し、入力イメージを Amazon S3 にアップロードします。
3. 返された Face ID もデータベースに保存します。保存するには、DynamoDB データベーステーブルを作成し、そのテーブルに Face ID をアップロードします。以下のコード例では、DynamoDB テーブルを作成します。テーブルを作成するコードを実行するのは 1 回のみで

すので、ご注意ください。以下のコードで、region の値を、アカウントに関連付けられたリージョンの値に置き換えます。また、database_name の値を、DynamoDB テーブルに付ける名前に置き換えます。

```
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. 上のコードを保存して実行し、テーブルを作成します。
5. テーブルを作成すると、返された FaceID をそこにアップロードできます。アップロードするには、Table 関数を使用してテーブルへの接続を確立し、put_item 関数を使用してデータをアップロードします。

   以下のコード例で、bucket の値を、Amazon S3 にアップロードした入力イメージを含むバケットの名前に置き換えます。また、 file_name の値を、Amazon S3 バケットにアップロードした入力ファイルの名前に置き換え、key_name の値を、前に入力ファイルの識別に使用したキーに置き換えます。最後に、region の値を、アカウントに関連付けられたリージョンの名前に置き換えます。これらの値は、ステップ 1 で指定した値と一致している必要があります。

AddDBEntry は、顔に割り当てられた FaceID、ImageId、Confidence の値をコレクションに保存します。前の IndexFaces セクションのステップ 2 で保存した値を、以下の関数に入力します。

```
import boto3
from pprint import pprint
from decimal import Decimal
import json

# The local file that was stored in S3 bucket
bucket = "amzn-s3-demo-bucket"
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. 上のコード例を保存して実行し、返された Face ID データをテーブルに保存します。

既存ユーザーのログイン

ユーザーは、一度コレクションに登録されると、SearchFacesByImage オペレーションを使って、再度利用する際に認証を受けることができます。そのためには、入力イメージを取得し、そのイメージの質を DetectFaces を使って確認する必要があります。そうすることで、SearchFacesbyImage オペレーションを実行する前に、適切なイメージが使用されているかどうかを判断します。

DetectFaces オペレーションを呼び出す

1. DetectFaces オペレーションを使って、顔イメージの画質をチェックし、カメラで撮影したイメージが SearchFacesByImage オペレーションによる処理に適しているかどうかを判断します。入力イメージには顔を 1 つしか含むことができません。以下のコード例では、入力イメージを取得して DetectFaces オペレーションに使用します。

   以下のコード例では、photo の値を、ローカルのターゲットイメージの名前に置き換え、region の値を、アカウントに関連付けられたリージョンの名前に置き換えます。

```
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. コードを保存して実行します。

SearchFacesByImage オペレーションを呼び出す

1. SearchFacesByImage を使って検出された顔とコレクション内にある顔とを比較するコードを作成します。上の「新規ユーザー登録」のセクションで紹介したコードを使って、入力イメージを SearchFacesByImage オペレーションに提供します。

   以下のコード例で、collectionId の値を、検索するコレクションに変更します。また、bucket の値を Amazon S3 バケットの名前に変更し、fileName の値を そのバケット内のイメージファイルに変更します。region の値を、アカウントに関連付けられたリージョンの名前に置き換えます。さらに、threshold の値を、選択したパーセンタイルに置き換えて、類似度のしきい値を指定します。

```
import boto3

bucket = 'amzn-s3-demo-bucket'
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. コードを保存して実行します。

返された FaceID と信頼度を確認

FaceID、Similarity、Confidence の各属性など、レスポンスの要素を出力することにより、一致した FaceID の情報を確認できるようになりました。

```
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']))
```

マーケティングアプリケーション用に Amazon Rekognition (REK) を使用してラベルを検出する

このチュートリアルでは、ウェブサイトにアップロードされた画像に基づいてユーザーに E メールを送信する、Python サンプルアプリケーションの構築方法について説明します。このサンプルアプリケーションの目的は、ユーザーをソーシャルメディアのマーケティングキャンペーンに参加させることです。写真内でランドマークが認識された場合に、旅行のお得情報に関するパーソナライズされた E メールがユーザーに送信されます。

このソリューションは、Amazon Rekognition、Amazon S3、DynamoDB DynamoDB 、CloudWatch、Amazon SES などのさまざまな AWS サービスを統合します。DynamoDB データベースに登録されているアカウントから画像がアップロードされると、このアプリケーションは Amazon Rekognition を使用してその画像のラベルを検出します。その後、検出したラベルに基づいて、アカウント所有者にマーケティング E メールを送信します。このソリューションのアーキテクチャの全容は以下のとおりです。

- ユーザーデータを DynamoDB データベースに保存します。
- ユーザーによって画像データとメタデータ (ユーザーアカウント番号) が Amazon S3 にアップロードされます。

- Lambda 関数が DetectLabels を呼び出して、アップロードされた画像内のラベルを識別して記録し、DynamoDB 内のユーザー E メールを検索して、画像をアップロードしたユーザーにマーケティング E メールを送信します。
- Lambda 関数の結果が、後で確認できるように、CloudWatch に記録されます。

このチュートリアルのすべての手順の概要は以下のとおりです。

1. DynamoDB テーブルを作成してデータを入力します。
2. ログと通知の機能を備えた Lambda 関数を作成します。
3. Lambda 関数とアクセス許可を設定します。
4. 画像とメタデータを Amazon S3 にアップロードします。
5. CloudWatch Logs をポーリングします。

トピック

- 前提条件
- DynamoDB テーブルの作成とデータ入力
- ログと通知の機能を備えた Lambda 関数の作成
- アクセス許可と Lambda 関数の設定
- 画像をメタデータとともに Amazon S3 にアップロードする
- クライアントからの CloudWatch Logs のポーリング

前提条件

このチュートリアルを開始する前に、以下が必要です。

- AWS アカウントと適切な IAM アクセス許可。
- 開発環境にインストールされた Python と Boto3。
- Lambda、Amazon S3、DynamoDB、Amazon SES、CloudWatch の基本的な理解。
- Lambda と Amazon SES の基本的な知識。

DynamoDB テーブルの作成とデータ入力

このチュートリアルを開始するにあたり、顧客データ (E メールアドレス、年齢、電話番号、会員ステータスなど) を保存するための DynamoDB テーブルを作成します。「AccountNumber」変数をプライマリキーとして使用します。また、このテーブルにサンプルデータを挿入します。

DynamoDB テーブルの作成

まず、DynamoDB テーブルを作成します。Python と Boto3 を使用して、このテーブルを以下のように設定します。`create_user_table` という関数を作成し、その中で DynamoDB リソースに接続します。以下のコードサンプルで、「region_name」の値をお客様のアカウントが運用されているリージョンの名前に置き換えてから、コードセルを実行してテーブルを作成します。

```
import boto3
def create_user_table(dynamodb=None):
    if not dynamodb:
        dynamodb = boto3.resource('dynamodb', region_name='us-east-1')

    table = dynamodb.create_table(
        TableName='CustomerDataTable',
        KeySchema=[
            {
                'AttributeName': 'AccountNumber',
                'KeyType': 'HASH'  # Partition key
            },
        ],
        AttributeDefinitions=[
            {
                'AttributeName': 'AccountNumber',
                'AttributeType': 'S'
            },
        ],
        ProvisionedThroughput={
            'ReadCapacityUnits': 10,
            'WriteCapacityUnits': 10
        }
    )

    # Wait until the table exists.
    table.wait_until_exists()
    print("Table status:", table.table_status)
```

```
# Create the DynamoDB table.
create_user_table()
```

このスクリプトを実行すると、AccountNumber をプライマリキーとする CustomerDataTable という名前の DynamoDB テーブルが設定されます。

サンプルデータの挿入

次は、テーブルにサンプルデータを挿入します。このサンプルデータを使用して、アプリケーションの機能を一通り確認できます。

先ほど作成した CustomerDataTable にサンプルデータを追加する 2 つ目の関数を作成します。以下のコードによって 3 つのサンプルエントリが作成されます。各エントリには、アカウント番号、E メールアドレス、年齢、電話番号、会員ステータスが含まれています。以下のコードサンプルで、region_name の値をお客様のアカウントが運用されているリージョンの名前に置き換えてから、コードセルを実行してテーブルを作成します。アプリケーションの E メール配信部分をテストする場合は、以下の最初の顧客エントリの EmailAddress の値を、受信可能な E メールアドレスに置き換えます。コードを保存して実行します。

```
import boto3

def insert_sample_data(dynamodb=None):
    if not dynamodb:
        dynamodb = boto3.resource('dynamodb', region_name='us-east-1')

    table = dynamodb.Table('CustomerDataTable')

    # Sample data
    customers = [
        {
            'AccountNumber': 'ACC1000',
            'EmailAddress': 'email-for-delivery-here',
            'Age': 30,
            'PhoneNumber': '123-456-7890',
            'MembershipStatus': 'Active'
        },
        {
            'AccountNumber': 'ACC1001',
            'EmailAddress': 'jane.doe@example.com',
```

```
            'Age': 25,
            'PhoneNumber': '098-765-4321',
            'MembershipStatus': 'Inactive'
        },
        {
            'AccountNumber': 'ACC1002',
            'EmailAddress': 'pat.candella@example.com',
            'Age': 35,
            'PhoneNumber': '555-555-5555',
            'MembershipStatus': 'Active'
        }
    ]
        # Inserting data
    for customer in customers:
            print(f"Adding customer: {customer['AccountNumber']}")
            table.put_item(Item=customer)

# Insert sample data into DynamoDB
insert_sample_data()
```

DynamoDB テーブルの設定とデータの入力が完了したので、次はこのデータの取得処理を Lambda 関数に統合できます。これにより、アプリケーションは、今後の画像処理ワークフローで識別されるアカウント番号に基づいてユーザーの詳細を取得できるようになります。

ログと通知の機能を備えた Lambda 関数の作成

次は、Lambda 関数を作成します。画像のアップロードでトリガーされる Lambda 関数が、画像データとアカウントメタデータの両方を読み取り、それらのデータを使用して DynamoDB テーブル内の関連するユーザーデータを検索できるようにする必要があります。つまり、Amazon Rekognition の DetectLabels 関数を呼び出すだけでなく、AccountNumber を受け取り、そのデータを使用して DynamoDB から該当する E メールアドレスを取得する関数も必要です。さらに、Amazon SES を使用してその E メールアドレスに E メールを送信する関数も必要です。最後に、ロガーを使用してこのプロセスに関する情報を記録します。このデータは CloudWatch によって表示されます。

Lambda 関数の作成

ここでは、前述の要件を処理する Lambda 関数の概要を示します。テーブル名として CustomerDataTable 以外を使用した場合は、以下のコードサンプルで、正しい DynamoDB テーブルを指定していることを確認してください。さらに、「send_marketing_email」関数では、

「Source」引数の値を、実際に送信元として使用できる E メールアドレスに置き換える必要があります。

```
import json
import boto3
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    s3_bucket = event['Records'][0]['s3']['bucket']['name']
    s3_object_key = event['Records'][0]['s3']['object']['key']
    print(s3_bucket)
    print(s3_object_key)

    s3 = boto3.client('s3')
    try:
        s3_response = s3.head_object(Bucket=s3_bucket, Key=s3_object_key)
        account_number = s3_response['Metadata']['account_number']
    except Exception as e:
        logger.error(f"Failed to retrieve object or metadata: {str(e)}")
        raise e  # Optionally re-raise to handle the error upstream or signal a failure

    rekognition = boto3.client('rekognition')
    try:
        labels_response = rekognition.detect_labels(Image={'S3Object': {'Bucket':
 s3_bucket, 'Name': s3_object_key}})
        #logger.info(f"Detected labels: {json.dumps(labels_response['Labels'],
 indent=4)}")
    except Exception as e:
        #logger.info(f"Detected label: {label['Name']}")
        raise e

    #logger.info(f"Detected labels: {json.dumps(labels_response['Labels'], indent=4)}")

    landmark_detected = any(label['Name'] == 'Landmark' and label['Confidence'] > 20
 for label in labels_response['Labels'])
    if landmark_detected:
        result = notify_user_based_on_landmark(account_number)
        print(result)
        #logger.info(f"Detected label: {label['Name']}")
        #logger.info(f"Notification sent: {result}")
```

```
    return {
        'statusCode': 200,
        'body': json.dumps('Process completed successfully!')
    }

def notify_user_based_on_landmark(account_number):
     # Retrieve user data from DynamoDB
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('CustomerDataTable')
    user_info = table.get_item(Key={'AccountNumber': account_number})

    # Send email if user is found
    if 'Item' in user_info:
        send_marketing_email(user_info['Item']['EmailAddress'])
    return "User notified with relevant travel information."

def send_marketing_email(email):
    ses = boto3.client('ses')
    response = ses.send_email(
        Source='your-email@example.com',
        Destination={'ToAddresses': [email]},
        Message={
            'Subject': {'Data': 'Explore New Destinations!'},
            'Body': {
                'Text': {'Data': 'Check out our exclusive travel packages inspired by
 the landmark in your uploaded image!'}
            }
        }
    )
    return f"Email sent to {email} with status {response['ResponseMetadata']
['HTTPStatusCode']}"
    print("succeess")
```

Lambda 関数を作成したので、Lambda のアクセス許可を設定し、 AWS マネジメントコンソールで Lambda 関数のインスタンスを作成する必要があります。

アクセス許可と Lambda 関数の設定

IAM ロールを作成または更新する

ユーザーがアップロードした画像を Lambda 関数で処理する前に、そのためのアクセス許可を設定する必要があります。Lambda 関数には、Amazon S3、DynamoDB、Amazon SES、Amazon Rekognition、CloudWatch とのやり取りを許可するポリシーを設定した IAM ロールが必要です。

Lambda 関数の IAM ロールを設定するには:

1. AWS マネジメントコンソールを開きます。
2. [IAM] > [ロール] > [ロールの作成] に移動します。

   [ロールの作成] を選択します。
3. このロールを使用するサービスとして [Lambda] を選択します。[次へ] をクリックします。
4. 以下のポリシーをアタッチします (これらのポリシーはデモ目的でのみ選択されています。本番環境では、必要なアクセス許可のみに制限する必要があります)。

   - AmazonS3ReadOnlyAccess
   - [AmazonRekognitionReadOnlyAccess] を選択します。
   - AmazonDynamoDBFullAccess
   - AmazonSESFullAccess
   - AWSLambdaExecute
   - AWSLambdaBasicExecutionRole (CloudWatch のログ用)

   [次へ] をクリックします。
5. ロールの名前と説明を入力し、[ロールの作成] を選択してロールを作成します。
6. Lambda に適切なアクセス許可を設定したら、 AWS マネジメントコンソールを使用して Lambda 関数のインスタンスを作成できます。

   AWS マネジメントコンソールで Lambda サービスに移動します。
7. [関数を作成] を選択します。[一から作成] を選択します。
8. 関数名を入力し、Python ランタイムを選択します。[デフォルトの実行ロールの変更] で、[既存のロールを使用する] を選択し、先ほど作成した IAM ロールを選択します。
9. 次は、Lambda 関数のタブでコードを作成し、更新する必要があります。「lambda_function」というタブに移動し、そこにあるコードを次の Lambda コードサンプルに置き換えます。

   変更を保存し、デプロイします。
10. 次は、Lambda 関数のトリガーとして Amazon S3 イベントを設定する必要があります。

Lambda 関数の設定タブ/ページで、[トリガー] に移動し、[トリガーを追加] をクリックします。

11. 使用可能なトリガーのリストから [Amazon S3] を選択します。
12. トリガーを設定するには、関数をトリガーするバケットを選択します。

    イベントタイプとして [PUT] を選択します。特定の名前やタイプのファイルのみを処理する場合は、オプションでプレフィックスまたはサフィックスを指定できます。
13. [追加] をクリックしてトリガーを有効にし、設定を保存します。

Amazon SES E メールアドレスの検証

Amazon SES を使用して E メールを送信する前に、送信者と受取人の両方の E メールアドレスを検証する必要があります。これを実行するには:

1. Amazon SES コンソールに移動します。[ID 管理]、[E メールアドレス] の順に移動します。
2. [新しい E メールアドレスを検証する] をクリックします。検証する E メールアドレスを入力し、[この E メールアドレスを検証する] をクリックします。検証用リンクが記載された E メールが届きます。リンクをクリックして、検証プロセスを完了します。
3. 両方のアカウントの E メールアドレスを検証し、Lambda 関数に正しいアクセス許可を設定し、サンプルの顧客データを作成したら、選択した Amazon S3 バケットにテスト画像をアップロードすることで Lambda 関数をテストできます。

画像をメタデータとともに Amazon S3 にアップロードする

これで、 AWS コンソールまたは前に準備したスクリプトを使用して、選択した Amazon S3 バケットにテストイメージをアップロードできます。先ほど Lambda 関数で指定した Amazon S3 バケットに画像をアップロードするスクリプトを作成します。以下のコードサンプルでは、「upload_image_to_s3」への最初の引数として画像のパスを、2 番目の引数としてバケット名を、最後の引数として画像をアップロードするユーザーのアカウント番号を指定します。

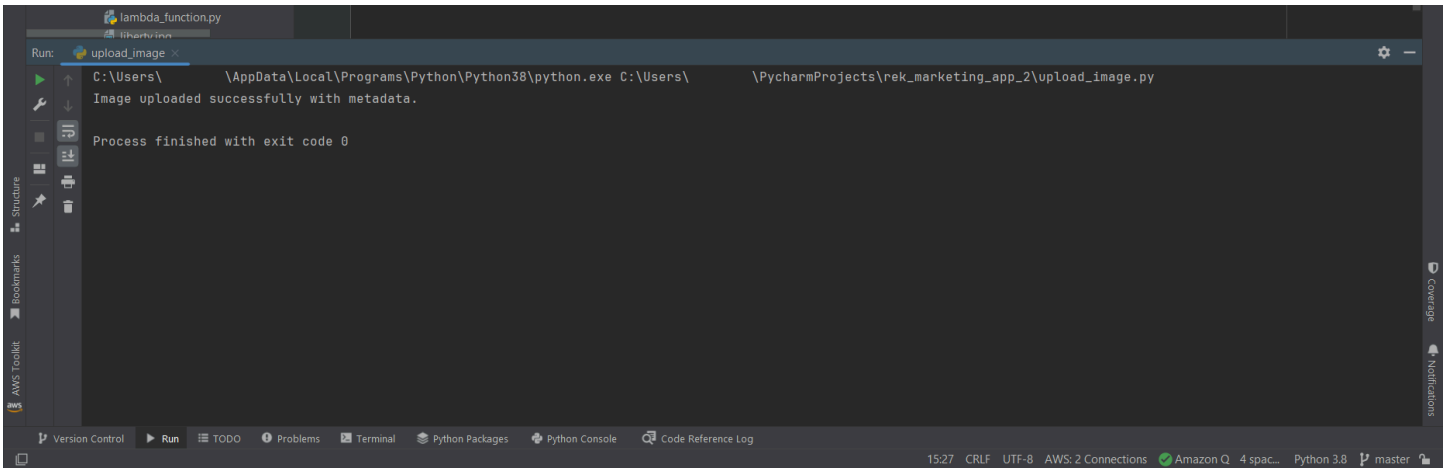
```
import boto3
def upload_image_to_s3(file_name, bucket, account_number):
    s3 = boto3.client('s3')
    try:
        with open(file_name, 'rb') as data:
            s3.upload_fileobj(
                Fileobj=data,
```

```
                Bucket=bucket,
                Key=file_name,
                ExtraArgs={
                    'Metadata': {'account_number': account_number}
                }
            )
        print("Image uploaded successfully with metadata.")
    except Exception as e:
        print("Failed to upload image")
        print(e)

# Usage
upload_image_to_s3('path-to-image-here', 's3-bucket-name-here', 'user-account-number-
here')
```

この関数では、upload_fileobj メソッドの ExtraArgs パラメータを使用して、画像とともにユーザー定義のメタデータ (account_number) を含めます。このメタデータは、後で によって使用 AWS されて、それに応じてイメージを処理できます。

スクリプトを保存し、実行します。これにより、画像がアップロードされます。



画像をアップロードしてから数分後、ここで指定したアカウントに先ほど関連付けたアドレスに、E メールが届きます。

クライアントからの CloudWatch Logs のポーリング

CloudWatch のログで、Lambda 関数が想定どおりにトリガーされ、実行されたかどうかを確認します。ログは [CloudWatch] > [ログ] > [ロググループ] > /AWS/lambda/your_function_name にあります。また、ログにアクセスして表示するスクリプトを作成することもできます。以下のコードサンプ

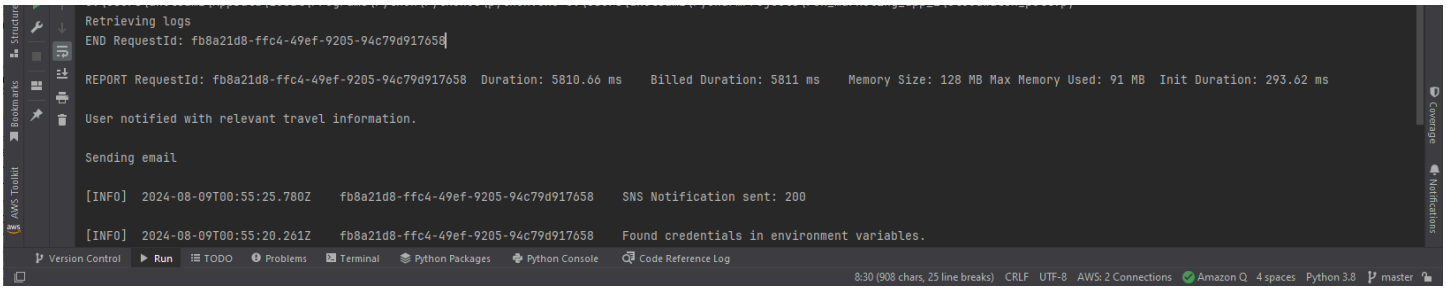
ルは、Lambda グループのログをポーリングし、過去 1 時間に生成されたログを出力します。コードを保存して実行します。

```
import boto3
import time

def fetch_lambda_logs(log_group_name, start_time):
    client = boto3.client('logs')
    query = "fields @timestamp, @message | sort @timestamp desc | limit 20"
    start_query_response = client.start_query(
        logGroupName=log_group_name,
        startTime=int(start_time),
        endTime=int(time.time()),
        queryString=query,
    )
    query_id = start_query_response['queryId']
    response = None
    while response is None or response['status'] == 'Running':
        time.sleep(1)  # Wait for 1 second before checking the query status again
        response = client.get_query_results(queryId=query_id)
    return response['results']

# Usage
log_group = '/aws/lambda/RekMediaFunction'
logs = fetch_lambda_logs(log_group, int(time.time()) - 3600)  # Fetch logs from the
 last hour
print("Retrieving logs")
for log in logs:
    #print(log)
    print(log[1]['value'])
```

このコードを実行するとログが表示され、関連する旅行情報を含む E メールがユーザーに通知されたことがわかります。



これで、Amazon S3 バケットにアップロードされた画像のラベルを検出し、画像をアップロードしたユーザーにプロモーションメッセージを含む E メールを送信するアプリケーションが完成しました。不要な課金が発生しないように、使用しなくなったリソースは必ず削除してください。

Lambda と Python を使用したイメージ内のラベルの検出

AWS Lambda は、サーバーのプロビジョニングや管理を行わずにコードを実行するために使用できるコンピューティングサービスです。Rekognition API オペレーションは、Lambda 関数内から呼び出せます。以下の手順は、Python で `DetectLabels` を呼び出す Lambda 関数を作成する方法を示したものです。

Lambda 関数が `DetectLabels` を呼び出すと、イメージ内で検出されたラベルと、検出時のその信頼度が返されます。

以下の手順には、Lambda 関数の呼び出し方と、Amazon S3 バケットまたはローカルのコンピュータにあるイメージと共にこれを指定する方法を示す、Python コードの例が含まれています。

選択したイメージが Rekognition の制限を満たしていることを確認します。イメージのファイルタイプとサイズの上限に関する詳細は、Rekognition の「Guidelines and quotas」と「DetectLabels API Reference」を参照してください。

Lambda 関数の作成 (コンソール)

このステップでは、空の Lambda 関数と、その関数が `DetectLabels` オペレーションを呼び出すための IAM 実行ロールを作成します。ソースコードとレイヤー (オプション) は、後半のステップで Lambda 関数に追加します。

Amazon S3 バケットに保存されているドキュメントを使用している場合は、このステップで、ドキュメントを保存しているバケットへのアクセスを許可する方法についても説明します。

AWS Lambda 関数を作成するには (コンソール)

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/lambda/ で AWS Lambda コンソールを開きます。
2. [Create function (関数の作成)] を選択します。詳細については、「コンソールで Lambda の関数の作成」を参照してください。
3. 次のオプションを選択します。

   - [最初から作成] を選択します。

- [関数名] に値を入力します。
- [ランタイム] で、最新バージョンの Python を選択します。
- [アーキテクチャ] で [x86_64] を選択します。

4. [関数の作成] を選択し、 AWS Lambda 関数を作成します。
5. 関数ページで、[設定] タブを選択します。
6. [アクセス権限] タブの [実行ロール] でロール名を選択し、このロールを IAM コンソールで開きます。
7. [アクセス権限] タブで、[アクセス権限を追加]、次に [インラインポリシーを作成] をクリックします。
8. [JSON] タブを選択し、ポリシーを次のポリシーに置き換えます。

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

9. [ポリシーの確認] を選択します。
10. ポリシーの名前を入力します (例: DetectLabels-access)。
11. [Create policy] を選択します。
12. 分析用のドキュメントを Amazon S3 バケットに保存する場合は、Amazon S3 アクセスポリシーを追加する必要があります。追加するには、 AWS Lambda コンソールでステップ 7 ~ 11 を繰り返し、以下のように変更を加えます。

    a. ステップ 8 で、以下のポリシーを使用します。*bucket/folder path* を、Amazon S3 バケットと、分析するドキュメントのフォルダパスに置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3Access",
```

```
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::bucket/folder path/*"
        }
    ]
}
```

b. ステップ 10 で、異なるポリシー名 (S3Bucket-access など) を選択します。

(オプション) レイヤーの作成 (コンソール)

Lambda 関数を使用して `DetectLabels` を呼び出すときは、このステップを実行する必要はありません。

`DetectLabels` オペレーションは、 AWS SDK for Python (Boto3) の一部としてデフォルトの Lambda Python 環境に含まれています。

Lambda 関数の他の部分で、デフォルトの Lambda Python 環境にない最新の AWS サービス更新が必要な場合は、このステップを実行して、最新の Boto3 SDK リリースをレイヤーとして関数に追加できます。

SDK をレイヤーとして追加するには、最初に Boto3 SDK を含む .zip ファイルアーカイブを作成します。次に、レイヤーを作成して .zip ファイルアーカイブをそのレイヤーに追加します。詳細については「Using layers with your Lambda function」を参照してください。

レイヤーを作成して追加するには (コンソール)

1. コマンドプロンプトを開き、次のコマンドを入力して、最新バージョンの AWS SDK でデプロイパッケージを作成します。

```
pip install boto3 --target python/.
zip boto3-layer.zip -r python/
```

2. zip ファイルの名前 (boto3-layer.zip) を書き留めます。この名前は、この手順内のステップ 8 で使用します。
3. https://console.aws.amazon.com/lambda/ で AWS Lambda コンソールを開きます。
4. ナビゲーションペインで [レイヤー] を選択します。
5. [レイヤーを作成] を選択します。
6. [名前] と [説明] の値を入力します。

7. [コードエントリタイプ] で、[.zip ファイルをアップロード] を選択し、[アップロード] を選択します。
8. ダイアログボックスで、この手順のステップ 1 で作成した .zip ファイルアーカイブ (boto3-layer.zip) を選択します。
9. [互換性のあるランタイム] で、最新バージョンの Python を選択します。
10. [作成] を選択してレイヤーを作成します。
11. ナビゲーションペインのメニューアイコンを選択します。
12. ナビゲーションペインで、[関数] を選択します。
13. リソースリストの中から、「???」で作成した関数を選択します。
14. [コード] タブを選択します。
15. [レイヤー] セクションで、[レイヤーの追加] を選択します。
16. [カスタムレイヤー] を選択します。
17. [カスタムレイヤー] で、ステップ 6 で入力したレイヤー名を選択します。
18. [バージョン] でレイヤーバージョンを選択します。バージョンは「1」です。
19. [追加] を選択します。

Python コードの追加 (コンソール)

このステップでは、Lambda コンソールのコードエディタから Lambda 関数に Python コードを追加します。このコードは、`DetectLabels` オペレーションを使用してイメージ内のラベルを検出します。それにより、イメージ内で検出されたラベルと、検出されたラベルの信頼度が返されます。

`DetectLabels` オペレーションに提供するドキュメントは、Amazon S3 バケットかローカルコンピュータにあります。

Python コードを追加するには (コンソール)

1. [コード] タブに移動します。
2. コードエディタで、lambda_function.py のコードを以下のコードに置き換えます。

```
import boto3
import logging
from botocore.exceptions import ClientError
import json
import base64
```

```
# Instantiate logger
logger = logging.getLogger(__name__)

# connect to the Rekognition client
rekognition = boto3.client('rekognition')

def lambda_handler(event, context):

    try:
        image = None
        if 'S3Bucket' in event and 'S3Object' in event:
            s3 = boto3.resource('s3')
            s3_object = s3.Object(event['S3Bucket'], event['S3Object'])
            image = s3_object.get()['Body'].read()

        elif 'image' in event:
            image_bytes = event['image'].encode('utf-8')
            img_b64decoded = base64.b64decode(image_bytes)
            image = img_b64decoded


        elif image is None:
            raise ValueError('Missing image, check image or bucket path.')

        else:
            raise ValueError("Only base 64 encoded image bytes or S3Object are
 supported.")

        response = rekognition.detect_labels(Image={'Bytes': image})
        lambda_response = {
            "statusCode": 200,
            "body": json.dumps(response)
        }
        labels = [label['Name'] for label in response['Labels']]
        print("Labels found:")
        print(labels)

    except ClientError as client_err:

       error_message = "Couldn't analyze image: " + client_err.response['Error']
['Message']

       lambda_response = {
           'statusCode': 400,
```

```
            'body': {
                "Error": client_err.response['Error']['Code'],
                "ErrorMessage": error_message
            }
        }
        logger.error("Error function %s: %s",
                     context.invoked_function_arn, error_message)


    except ValueError as val_error:

        lambda_response = {
            'statusCode': 400,
            'body': {
                "Error": "ValueError",
                "ErrorMessage": format(val_error)
            }
        }
        logger.error("Error function %s: %s",
                     context.invoked_function_arn, format(val_error))

    return lambda_response
```

3. [デプロイ] を選択し、Lambda 関数をデプロイします。

Python コードを追加するには (コンソール)

Lambda 関数を作成したので、この関数を呼び出すことでイメージ内のラベルを検出できます。

このステップでは、コンピュータで Python コードを実行し、ローカルのイメージまたは Amazon S3 バケット内のイメージを Lambda 関数に渡します。

Lambda 関数を作成したリージョンと同じ AWS リージョンでコードを実行してください。Lambda 関数の AWS リージョンは、Lambda コンソールの関数の詳細ページのナビゲーションバーで表示できます。

Lambda 関数がタイムアウトエラーを返した場合は、Lambda 関数のタイムアウト期間を延長します。詳細については、「関数のタイムアウトの設定 (コンソール)」を参照してください。

コードから Lambda 関数を呼び出す方法の詳細については、「Lambda 関数を呼び出す」を参照してください。

Lambda 関数をテストするには

1. まだの場合は、 以下を実行してください:

   a. ユーザーに lambda:InvokeFunction のアクセス許可があることを確認します。以下のポリシーを使用できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "InvokeLambda",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "ARN for lambda function"
        }
    ]
}
```

   Lambda 関数の ARN は、Lambda コンソールにある [関数の概要] から取得できます。

   アクセス権限を付与するにはユーザー、グループ、またはロールにアクセス許可を追加します。

   - 以下のユーザーとグループ AWS IAM Identity Center :

     アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」の手順に従ってください。
   - IAM 内で、ID プロバイダーによって管理されているユーザー:

     ID フェデレーションのロールを作成します。詳細については「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールを作成する」を参照してください。
   - IAM ユーザー:
     - ユーザーが担当できるロールを作成します。手順については「IAM ユーザーガイド」の「IAM ユーザーのロールの作成」を参照してください。
     - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加します。詳細については「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス権限の追加」を参照してください。

b. AWS SDK for Python をインストールして設定します。詳細については、「ステップ 2: AWS CLI と AWS SDKsを設定する」を参照してください。

2. 次のコードを client.py という名前のファイルに保存します。

```
import boto3
import json
import base64
import pprint

# Replace with the name of your S3 bucket and image object key
bucket_name = "name of bucket"
object_key = "name of file in s3 bucket"
# If using a local file, supply the file name as the value of image_path below
image_path = ""

# Create session and establish connection to client['
session = boto3.Session(profile_name='developer-role')
s3 = session.client('s3', region_name="us-east-1")
lambda_client = session.client('lambda',  region_name="us-east-1")

# Replace with the name of your Lambda function
function_name = 'RekDetectLabels'

def analyze_image_local(img_path):

    print("Analyzing local image:")

    with open(img_path, 'rb') as image_file:
        image_bytes = image_file.read()
        data = base64.b64encode(image_bytes).decode("utf8")

        lambda_payload = {"image": data}

        # Invoke the Lambda function with the event payload
        response = lambda_client.invoke(
            FunctionName=function_name,
            Payload=(json.dumps(lambda_payload))
        )

        decoded = json.loads(response['Payload'].read().decode())
        pprint.pprint(decoded)

def analyze_image_s3(bucket_name, object_key):
```

```
    print("Analyzing image in S3 bucket:")

    # Load the image data from S3 into memory
    response = s3.get_object(Bucket=bucket_name, Key=object_key)
    image_data = response['Body'].read()
    image_data = base64.b64encode(image_data).decode("utf8")

     # Create the Lambda event payload
    event = {
        'S3Bucket': bucket_name,
        'S3Object': object_key,
        'ImageBytes': image_data
    }

    # Invoke the Lambda function with the event payload
    response = lambda_client.invoke(
            FunctionName=function_name,
            InvocationType='RequestResponse',
            Payload=json.dumps(event),
            )

    decoded = json.loads(response['Payload'].read().decode())
    pprint.pprint(decoded)

def main(path_to_image, name_s3_bucket, obj_key):

    if str(path_to_image) != "":
        analyze_image_local(path_to_image)
    else:
        analyze_image_s3(name_s3_bucket, obj_key)

if __name__ == "__main__":
    main(image_path, bucket_name, object_key)
```

3. コードを実行します。ドキュメントが Amazon S3 バケットにある場合は、「???」のステップ 12 で指定したバケットと同じバケットであることを確認します。

   正常に実行されると、コードはドキュメント内で検出された各ブロックタイプについて JSON レスポンスの一部を返します。

SDK を使用した Amazon Rekognition のコード例 AWS SDKs

次のコード例は、 AWS Software Development Kit (SDK) で Amazon Rekognition を使用する方法を示しています。本章のコード例は、このガイドの残りの部分で見られるコード例を補足することを目的としています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「シナリオ」は、1 つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

開始方法

Hello Amazon Rekognition

以下のコード例では、Amazon Rekognition の使用を開始する方法を示しています。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CMakeLists.txt CMake ファイルのコード。

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
```

```
set(SERVICE_COMPONENTS rekognition)

# Set this project's name.
project("hello_rekognition")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_rekognition.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_rekognition.cpp ソースファイルのコード。

```
#include <aws/core/Aws.h>
#include <aws/rekognition/RekognitionClient.h>
#include <aws/rekognition/model/ListCollectionsRequest.h>
#include <iostream>

/*
 *  A "Hello Rekognition" starter application which initializes an Amazon
 Rekognition client and
 *  lists the Amazon Rekognition collections in the current account and region.
 *
 *  main function
 *
 *  Usage: 'hello_rekognition'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    //  Optional: change the log level for debugging.
    //  options.loggingOptions.logLevel = Aws::Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::Rekognition::RekognitionClient rekognitionClient(clientConfig);
        Aws::Rekognition::Model::ListCollectionsRequest request;
        Aws::Rekognition::Model::ListCollectionsOutcome outcome =
                rekognitionClient.ListCollections(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::String>& collectionsIds =
 outcome.GetResult().GetCollectionIds();
            if (!collectionsIds.empty()) {
                std::cout << "collectionsIds: " << std::endl;
                for (auto &collectionId : collectionsIds) {
                    std::cout << "- " << collectionId << std::endl;
                }
            } else {
                std::cout << "No collections found" << std::endl;
            }
        } else {
```

```
            std::cerr << "Error with ListCollections: " << outcome.GetError()
                      << std::endl;
        }
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「ListCollections」を参照してください。

コードの例

- AWS SDKs を使用した Amazon Rekognition の基本的な例
  - Hello Amazon Rekognition
  - SDK を使用した Amazon Rekognition のアクション AWS SDKs
    - AWS SDK または CLI CompareFacesで を使用する
    - AWS SDK または CLI CreateCollectionで を使用する
    - AWS SDK または CLI DeleteCollectionで を使用する
    - AWS SDK または CLI DeleteFacesで を使用する
    - AWS SDK または CLI DescribeCollectionで を使用する
    - AWS SDK または CLI DetectFacesで を使用する
    - AWS SDK または CLI DetectLabelsで を使用する
    - AWS SDK または CLI DetectModerationLabelsで を使用する
    - AWS SDK または CLI DetectTextで を使用する
    - AWS SDK または CLI DisassociateFacesで を使用する
    - AWS SDK または CLI GetCelebrityInfoで を使用する
    - AWS SDK または CLI IndexFacesで を使用する
    - AWS SDK または CLI ListCollectionsで を使用する
    - AWS SDK または CLI ListFacesで を使用する
    - AWS SDK または CLI RecognizeCelebritiesで を使用する
    - AWS SDK または CLI SearchFacesで を使用する

    - AWS SDK または CLI SearchFacesByImageで を使用する
- SDK を使用した Amazon Rekognition のシナリオ AWS SDKs
    - AWS SDK を使用して Amazon Rekognition コレクションを構築し、その中に顔を見つける
    - ユーザーがラベルを使用して写真を管理できる写真アセット管理アプリケーションの作成
    - AWS SDK を使用して Amazon Rekognition でイメージ内の PPE を検出する
    - AWS SDK を使用して Amazon Rekognition でイメージ内の要素を検出して表示する
    - AWS SDK を使用してイメージ内の顔を検出する
    - Amazon Rekognition と AWS SDK を使用してビデオ内の情報を検出する
    - AWS SDK を使用して Amazon Rekognition でイメージ内のオブジェクトを検出する
    - AWS SDK を使用して Amazon Rekognition でビデオ内の人物とオブジェクトを検出する
    - AWS SDK を使用して EXIF およびその他のイメージ情報を保存する

AWS SDKs を使用した Amazon Rekognition の基本的な例

以下のコード例では、 AWS SDK を利用した Amazon Rekognition の基本的な使用方法を示しています。

例

- Hello Amazon Rekognition
- SDK を使用した Amazon Rekognition のアクション AWS SDKs
    - AWS SDK または CLI CompareFacesで を使用する
    - AWS SDK または CLI CreateCollectionで を使用する
    - AWS SDK または CLI DeleteCollectionで を使用する
    - AWS SDK または CLI DeleteFacesで を使用する
    - AWS SDK または CLI DescribeCollectionで を使用する
    - AWS SDK または CLI DetectFacesで を使用する
    - AWS SDK または CLI DetectLabelsで を使用する
    - AWS SDK または CLI DetectModerationLabelsで を使用する
    - AWS SDK または CLI DetectTextで を使用する
    - AWS SDK または CLI DisassociateFacesで を使用する
    - AWS SDK または CLI GetCelebrityInfoで を使用する

- AWS SDK または CLI IndexFacesで を使用する
- AWS SDK または CLI ListCollectionsで を使用する
- AWS SDK または CLI ListFacesで を使用する
- AWS SDK または CLI RecognizeCelebritiesで を使用する
- AWS SDK または CLI SearchFacesで を使用する
- AWS SDK または CLI SearchFacesByImageで を使用する

Hello Amazon Rekognition

以下のコード例では、Amazon Rekognition の使用を開始する方法を示しています。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

CMakeLists.txt CMake ファイルのコード。

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS rekognition)

# Set this project's name.
project("hello_rekognition")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})
```

```
if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_rekognition.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_rekognition.cpp ソースファイルのコード。

```
#include <aws/core/Aws.h>
#include <aws/rekognition/RekognitionClient.h>
#include <aws/rekognition/model/ListCollectionsRequest.h>
#include <iostream>

/*
 *  A "Hello Rekognition" starter application which initializes an Amazon
 Rekognition client and
 *  lists the Amazon Rekognition collections in the current account and region.
 *
 *  main function
```

```
 *
 *  Usage: 'hello_rekognition'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    //  Optional: change the log level for debugging.
    //  options.loggingOptions.logLevel = Aws::Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::Rekognition::RekognitionClient rekognitionClient(clientConfig);
        Aws::Rekognition::Model::ListCollectionsRequest request;
        Aws::Rekognition::Model::ListCollectionsOutcome outcome =
                rekognitionClient.ListCollections(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::String>& collectionsIds =
 outcome.GetResult().GetCollectionIds();
            if (!collectionsIds.empty()) {
                std::cout << "collectionsIds: " << std::endl;
                for (auto &collectionId : collectionsIds) {
                    std::cout << "- " << collectionId << std::endl;
                }
            } else {
                std::cout << "No collections found" << std::endl;
            }
        } else {
            std::cerr << "Error with ListCollections: " << outcome.GetError()
                      << std::endl;
        }
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「ListCollections」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用した Amazon Rekognition のアクション AWS SDKs

次のコード例は、 AWS SDKs を使用して個々の Amazon Rekognition アクションを実行する方法を示しています。それぞれの例には、GitHub へのリンクがあり、そこにはコードの設定と実行に関する説明が記載されています。

これらの抜粋は Amazon Rekognition API を呼び出すもので、コンテキスト内で実行する必要がある大規模なプログラムからのコードの抜粋です。アクションは SDK を使用した Amazon Rekognition のシナリオ AWS SDKs のコンテキスト内で確認できます。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細な一覧については、「Amazon Rekognition API Reference」を参照してください。

例

- AWS SDK または CLI CompareFacesで を使用する
- AWS SDK または CLI CreateCollectionで を使用する
- AWS SDK または CLI DeleteCollectionで を使用する
- AWS SDK または CLI DeleteFacesで を使用する
- AWS SDK または CLI DescribeCollectionで を使用する
- AWS SDK または CLI DetectFacesで を使用する
- AWS SDK または CLI DetectLabelsで を使用する
- AWS SDK または CLI DetectModerationLabelsで を使用する
- AWS SDK または CLI DetectTextで を使用する
- AWS SDK または CLI DisassociateFacesで を使用する
- AWS SDK または CLI GetCelebrityInfoで を使用する
- AWS SDK または CLI IndexFacesで を使用する
- AWS SDK または CLI ListCollectionsで を使用する
- AWS SDK または CLI ListFacesで を使用する

- AWS SDK または CLI RecognizeCelebritiesで を使用する
- AWS SDK または CLI SearchFacesで を使用する
- AWS SDK または CLI SearchFacesByImageで を使用する

AWS SDK または CLI **CompareFaces**で を使用する

次のサンプルコードは、CompareFaces を使用する方法を説明しています。

詳細については、「イメージ内の顔を比較する」を参照してください。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    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.
    /// </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.");
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「CompareFaces」を参照してください。

CLI

AWS CLI

2 つの画像内の顔を比較するには

次の compare-faces コマンドは、Amazon S3 バケットに保存されている 2 つの画像の顔を比較します。

```
aws rekognition compare-faces \
    --source-image '{"S3Object":
{"Bucket":"MyImageS3Bucket","Name":"source.jpg"}}' \
    --target-image '{"S3Object":
{"Bucket":"MyImageS3Bucket","Name":"target.jpg"}}'
```

出力:

```
{
    "UnmatchedFaces": [],
    "FaceMatches": [
        {
            "Face": {
```

```
                "BoundingBox": {
                    "Width": 0.12368916720151901,
                    "Top": 0.16007372736930847,
                    "Left": 0.5901257991790771,
                    "Height": 0.25140416622161865
                },
                "Confidence": 100.0,
                "Pose": {
                    "Yaw": -3.7351467609405518,
                    "Roll": -0.10309021919965744,
                    "Pitch": 0.8637830018997192
                },
                "Quality": {
                    "Sharpness": 95.51618957519531,
                    "Brightness": 65.29893493652344
                },
                "Landmarks": [
                    {
                        "Y": 0.26721030473709106,
                        "X": 0.6204193830490112,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.26831310987472534,
                        "X": 0.6776827573776245,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.3514654338359833,
                        "X": 0.6241428852081299,
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.35258132219314575,
                        "X": 0.6713621020317078,
                        "Type": "mouthRight"
                    },
                    {
                        "Y": 0.3140771687030792,
                        "X": 0.6428444981575012,
                        "Type": "nose"
                    }
                ]
            },
```

```
            "Similarity": 100.0
        }
    ],
    "SourceImageFace": {
        "BoundingBox": {
            "Width": 0.12368916720151901,
            "Top": 0.16007372736930847,
            "Left": 0.5901257991790771,
            "Height": 0.25140416622161865
        },
        "Confidence": 100.0
    }
}
```

詳細については、「Amazon Rekognition ディベロッパーガイド」の「イメージ間の顔の比較」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「CompareFaces」を参照してください。https://awscli.amazonaws.com/v2/documentation/api/latest/reference/rekognition/compare-faces.html

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;
import software.amazon.awssdk.core.SdkBytes;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
```

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CompareFaces {
    public static void main(String[] args) {
        final String usage = """
            Usage: <bucketName> <sourceKey> <targetKey>

            Where:
                bucketName - The name of the S3 bucket where the images are
 stored.
                sourceKey  - The S3 key (file name) for the source image.
                targetKey  - The S3 key (file name) for the target image.
           """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucketName = args[0];
        String sourceKey = args[1];
        String targetKey = args[2];

        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();
        compareTwoFaces(rekClient, bucketName, sourceKey, targetKey);
    }

    /**
     * Compares two faces from images stored in an Amazon S3 bucket using AWS
 Rekognition.
     *
     * <p>This method takes two image keys from an S3 bucket and compares the
 faces within them.
```

```
    * It prints out the confidence level of matched faces and reports the number
of unmatched faces.</p>
    *
    * @param rekClient   The {@link RekognitionClient} used to call AWS
Rekognition.
    * @param bucketName  The name of the S3 bucket containing the images.
    * @param sourceKey   The object key (file path) for the source image in the
S3 bucket.
    * @param targetKey   The object key (file path) for the target image in the
S3 bucket.
    * @throws RuntimeException If the Rekognition service returns an error.
    */
   public static void compareTwoFaces(RekognitionClient rekClient, String
bucketName, String sourceKey, String targetKey) {
       try {
           Float similarityThreshold = 70F;
           S3Object s3ObjectSource = S3Object.builder()
                   .bucket(bucketName)
                   .name(sourceKey)
                   .build();

           Image sourceImage = Image.builder()
                   .s3Object(s3ObjectSource)
                   .build();

           S3Object s3ObjectTarget = S3Object.builder()
                   .bucket(bucketName)
                   .name(targetKey)
                   .build();

           Image targetImage = Image.builder()
                   .s3Object(s3ObjectTarget)
                   .build();

           CompareFacesRequest facesRequest = CompareFacesRequest.builder()
                   .sourceImage(sourceImage)
                   .targetImage(targetImage)
                   .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> unmatchedFaces =
 compareFacesResult.unmatchedFaces();
            System.out.println("There were " + unmatchedFaces.size() + " face(s)
 that did not match.");

        } catch (RekognitionException e) {
            System.err.println("Error comparing faces: " +
 e.awsErrorDetails().errorMessage());
            throw new RuntimeException(e);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「CompareFaces」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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.fromEnvironment { 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}")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API Reference」の「CompareFaces」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「CompareFaces」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **CreateCollection**で を使用する

次のサンプルコードは、CreateCollection を使用する方法を説明しています。

詳細については、「コレクションを作成する」を参照してください。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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.
/// </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}");
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「CreateCollection」を参照してください。

CLI

AWS CLI

コレクションを作成するには

次の create-collection コマンドは、指定された名前のコレクションを作成します。

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

出力:

```
{
    "CollectionArn": "aws:rekognition:us-west-2:123456789012:collection/
MyCollection",
    "FaceModelVersion": "4.0",
    "StatusCode": 200
}
```

詳細については、「Amazon Rekognition 開発者ガイド」の「コレクションの作成」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「CreateCollection」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.CreateCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.CreateCollectionRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateCollection {
    public static void main(String[] args) {
        final String usage = """

            Usage: <collectionName>\s

            Where:
                collectionName - The name of the collection.\s
            """;

        if (args.length != 1) {
            System.out.println(usage);
```

```
            System.exit(1);
        }

        String collectionId = args[0];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Creating collection: " + collectionId);
        createMyCollection(rekClient, collectionId);
        rekClient.close();
    }

    /**
     * Creates a new Amazon Rekognition collection.
     *
     * @param rekClient    the Amazon Rekognition client used to interact with
 the Rekognition service
     * @param collectionId the unique identifier for the collection to be created
     */
    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);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「CreateCollection」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun createMyCollection(collectionIdVal: String) {
    val request =
        CreateCollectionRequest {
            collectionId = collectionIdVal
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.createCollection(request)
        println("Collection ARN is ${response.collectionArn}")
        println("Status code is ${response.statusCode}")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API Reference」の「CreateCollection」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンス の「CreateCollection」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DeleteCollection**で を使用する

次のサンプルコードは、DeleteCollection を使用する方法を説明しています。

詳細については、「コレクションを削除する」を参照してください。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.Threading.Tasks;
using Amazon.Rekognition;
using Amazon.Rekognition.Model;

/// <summary>
/// Uses the Amazon Rekognition Service to delete an existing collection.
/// </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}");
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DeleteCollection」を参照してください。

CLI

AWS CLI

コレクションを削除するには

次の delete-collection コマンドは、指定されたコレクションを削除します。

```
aws rekognition delete-collection \
    --collection-id MyCollection
```

出力:

```
{
    "StatusCode": 200
}
```

詳細については、「Amazon Rekognition ディベロッパーガイド」の「コレクションの削除」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「DeleteCollection」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DeleteCollectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.DeleteCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteCollection {
    public static void main(String[] args) {
        final String usage = """
            Usage: <collectionId>\s

            Where:
                collectionId - The id of the collection to delete.\s
            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        Region region = Region.US_EAST_1;
```

```
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Deleting collection: " + collectionId);
        deleteMyCollection(rekClient, collectionId);
        rekClient.close();
    }

    /**
     * Deletes an Amazon Rekognition collection.
     *
     * @param rekClient      An instance of the {@link RekognitionClient} class,
 which is used to interact with the Amazon Rekognition service.
     * @param collectionId   The ID of the collection to be deleted.
     */
    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);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DeleteCollection」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun deleteMyCollection(collectionIdVal: String) {
    val request =
        DeleteCollectionRequest {
            collectionId = collectionIdVal
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.deleteCollection(request)
        println("The collectionId status is ${response.statusCode}")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API Reference」の「DeleteCollection」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
```

- API の詳細については、 AWS SDK for Python (Boto3) API リファレンス で「DeleteCollection」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DeleteFaces**で を使用する

次のサンプルコードは、DeleteFaces を使用する方法を説明しています。

詳細については、「コレクションから顔を削除する」を参照してください。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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.
/// </summary>
public class DeleteFaces
```

```
    {
        public static async Task Main()
        {
            string collectionId = "MyCollection";
            var faces = new List<string> { "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxx" };

            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}");
            });
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DeleteFaces」を参照してください。

CLI

AWS CLI

コレクションから顔を削除するには

次の delete-faces コマンドは、コレクションから指定された顔を削除します。

```
aws rekognition delete-faces \
    --collection-id MyCollection
    --face-ids '["0040279c-0178-436e-b70a-e61b074e96b0"]'
```

出力:

```
{
    "DeletedFaces": [
        "0040279c-0178-436e-b70a-e61b074e96b0"
    ]
}
```

詳細については、「Amazon Rekognition ディベロッパーガイド」の「コレクションからの顔の削除」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「DeleteFaces」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DeleteFacesRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteFacesFromCollection {
    public static void main(String[] args) {
        final String usage = """
            Usage: <collectionId> <faceId>\s
```

```
            Where:
                collectionId - The id of the collection from which faces are
deleted.\s
                faceId - The id of the face to delete.\s
           """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        String faceId = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Deleting collection: " + collectionId);
        deleteFacesCollection(rekClient, collectionId, faceId);
        rekClient.close();
    }

    /**
     * Deletes a face from the specified Amazon Rekognition collection.
     *
     * @param rekClient     an instance of the Amazon Rekognition client
     * @param collectionId  the ID of the collection from which the face should
be deleted
     * @param faceId        the ID of the face to be deleted
     * @throws RekognitionException if an error occurs while deleting the face
     */
    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);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DeleteFaces」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun deleteFacesCollection(
    collectionIdVal: String?,
    faceIdVal: String,
) {
    val deleteFacesRequest =
        DeleteFacesRequest {
            collectionId = collectionIdVal
            faceIds = listOf(faceIdVal)
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        rekClient.deleteFaces(deleteFacesRequest)
        println("$faceIdVal was deleted from the collection")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API Reference」の「DeleteFaces」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンス で「DeleteFaces」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DescribeCollection**で を使用する

次のサンプルコードは、DescribeCollection を使用する方法を説明しています。

詳細については、「コレクションを定義する」を参照してください。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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.
/// </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}");
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DescribeCollection」を参照してください。

CLI

AWS CLI

コレクションを記述するには

次の describe-collection の例は、指定されたコレクションの詳細を表示します。

```
aws rekognition describe-collection \
    --collection-id MyCollection
```

出力:

```
{
    "FaceCount": 200,
    "CreationTimestamp": 1569444828.274,
    "CollectionARN": "arn:aws:rekognition:us-west-2:123456789012:collection/
MyCollection",
    "FaceModelVersion": "4.0"
}
```

詳細については、「Amazon Rekognition 開発者ガイド」の「コレクションの定義」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「DescribeCollection」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.DescribeCollectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.DescribeCollectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DescribeCollection {
    public static void main(String[] args) {
        final String usage = """
            Usage:    <collectionName>

            Where:
                collectionName - The name of the Amazon Rekognition collection.\s
            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionName = args[0];
```

```
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        describeColl(rekClient, collectionName);
        rekClient.close();
    }

    /**
     * Describes an Amazon Rekognition collection.
     *
     * @param rekClient         The Amazon Rekognition client used to make the
 request.
     * @param collectionName    The name of the collection to describe.
     *
     * @throws RekognitionException If an error occurs while describing the
 collection.
     */
    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);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DescribeCollection」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun describeColl(collectionName: String) {
    val request =
        DescribeCollectionRequest {
            collectionId = collectionName
        }

    RekognitionClient.fromEnvironment { 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}")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API Reference」の「DescribeCollection」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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()
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DescribeCollection」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DetectFaces**で を使用する

次のサンプルコードは、DetectFaces を使用する方法を説明しています。

詳細については、「イメージ内の顔を検出する」を参照してください。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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.
/// </summary>
public class DetectFaces
{
    public static async Task Main()
    {
        string photo = "input.jpg";
        string bucket = "amzn-s3-demo-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);
            }
        }
    }
```

イメージ内のすべての顔の境界ボックス情報を表示します。

```
    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.
    /// </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}");
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DetectFaces」を参照してください。

CLI

AWS CLI

画像内の顔を検出するには

次の detect-faces コマンドは、Amazon S3 バケットに保存されている指定された画像の顔を検出します。

```
aws rekognition detect-faces \
    --image '{"S3Object":{"Bucket":"MyImageS3Bucket","Name":"MyFriend.jpg"}}' \
    --attributes "ALL"
```

出力:

```
{
    "FaceDetails": [
        {
            "Confidence": 100.0,
            "Eyeglasses": {
                "Confidence": 98.91107940673828,
                "Value": false
            },
            "Sunglasses": {
                "Confidence": 99.7966537475586,
                "Value": false
            },
            "Gender": {
                "Confidence": 99.56611633300781,
                "Value": "Male"
            },
            "Landmarks": [
```

```
                {
                    "Y": 0.26721030473709106,
                    "X": 0.6204193830490112,
                    "Type": "eyeLeft"
                },
                {
                    "Y": 0.26831310987472534,
                    "X": 0.6776827573776245,
                    "Type": "eyeRight"
                },
                {
                    "Y": 0.3514654338359833,
                    "X": 0.6241428852081299,
                    "Type": "mouthLeft"
                },
                {
                    "Y": 0.35258132219314575,
                    "X": 0.6713621020317078,
                    "Type": "mouthRight"
                },
                {
                    "Y": 0.3140771687030792,
                    "X": 0.6428444981575012,
                    "Type": "nose"
                },
                {
                    "Y": 0.24662546813488007,
                    "X": 0.6001564860343933,
                    "Type": "leftEyeBrowLeft"
                },
                {
                    "Y": 0.24326619505882263,
                    "X": 0.6303644776344299,
                    "Type": "leftEyeBrowRight"
                },
                {
                    "Y": 0.23818562924861908,
                    "X": 0.6146903038024902,
                    "Type": "leftEyeBrowUp"
                },
                {
                    "Y": 0.24373626708984375,
                    "X": 0.6640064716339111,
                    "Type": "rightEyeBrowLeft"
```

```
                },
                {
                    "Y": 0.24877218902111053,
                    "X": 0.7025929093360901,
                    "Type": "rightEyeBrowRight"
                },
                {
                    "Y": 0.23938551545143127,
                    "X": 0.6823262572288513,
                    "Type": "rightEyeBrowUp"
                },
                {
                    "Y": 0.265746533870697,
                    "X": 0.6112898588180542,
                    "Type": "leftEyeLeft"
                },
                {
                    "Y": 0.2676128149032593,
                    "X": 0.6317071914672852,
                    "Type": "leftEyeRight"
                },
                {
                    "Y": 0.262735515832901,
                    "X": 0.6201658248901367,
                    "Type": "leftEyeUp"
                },
                {
                    "Y": 0.27025148272514343,
                    "X": 0.6206279993057251,
                    "Type": "leftEyeDown"
                },
                {
                    "Y": 0.268223375082016,
                    "X": 0.6658390760421753,
                    "Type": "rightEyeLeft"
                },
                {
                    "Y": 0.2672517001628876,
                    "X": 0.687832236289978,
                    "Type": "rightEyeRight"
                },
                {
                    "Y": 0.26383838057518005,
                    "X": 0.6769183874130249,
```

```
                    "Type": "rightEyeUp"
                },
                {
                    "Y": 0.27138751745224,
                    "X": 0.676596462726593,
                    "Type": "rightEyeDown"
                },
                {
                    "Y": 0.32283174991607666,
                    "X": 0.6350004076957703,
                    "Type": "noseLeft"
                },
                {
                    "Y": 0.3219289481639862,
                    "X": 0.6567046642303467,
                    "Type": "noseRight"
                },
                {
                    "Y": 0.3420318365097046,
                    "X": 0.6450609564781189,
                    "Type": "mouthUp"
                },
                {
                    "Y": 0.3664324879646301,
                    "X": 0.6455618143081665,
                    "Type": "mouthDown"
                },
                {
                    "Y": 0.26721030473709106,
                    "X": 0.6204193830490112,
                    "Type": "leftPupil"
                },
                {
                    "Y": 0.26831310987472534,
                    "X": 0.6776827573776245,
                    "Type": "rightPupil"
                },
                {
                    "Y": 0.26343393325805664,
                    "X": 0.5946047306060791,
                    "Type": "upperJawlineLeft"
                },
                {
                    "Y": 0.3543180525302887,
```

```
                    "X": 0.6044883728027344,
                    "Type": "midJawlineLeft"
                },
                {
                    "Y": 0.4084877669811249,
                    "X": 0.6477024555206299,
                    "Type": "chinBottom"
                },
                {
                    "Y": 0.3562754988670349,
                    "X": 0.707981526851654,
                    "Type": "midJawlineRight"
                },
                {
                    "Y": 0.26580461859703064,
                    "X": 0.7234612107276917,
                    "Type": "upperJawlineRight"
                }
            ],
            "Pose": {
                "Yaw": -3.7351467609405518,
                "Roll": -0.10309021919965744,
                "Pitch": 0.8637830018997192
            },
            "Emotions": [
                {
                    "Confidence": 8.74203109741211,
                    "Type": "SURPRISED"
                },
                {
                    "Confidence": 2.501944065093994,
                    "Type": "ANGRY"
                },
                {
                    "Confidence": 0.7378743290901184,
                    "Type": "DISGUSTED"
                },
                {
                    "Confidence": 3.5296201705932617,
                    "Type": "HAPPY"
                },
                {
                    "Confidence": 1.7162904739379883,
                    "Type": "SAD"
```

```
                    },
                    {
                        "Confidence": 9.518536567687988,
                        "Type": "CONFUSED"
                    },
                    {
                        "Confidence": 0.45474427938461304,
                        "Type": "FEAR"
                    },
                    {
                        "Confidence": 72.79895782470703,
                        "Type": "CALM"
                    }
                ],
                "AgeRange": {
                    "High": 48,
                    "Low": 32
                },
                "EyesOpen": {
                    "Confidence": 98.93987274169922,
                    "Value": true
                },
                "BoundingBox": {
                    "Width": 0.12368916720151901,
                    "Top": 0.16007372736930847,
                    "Left": 0.5901257991790771,
                    "Height": 0.25140416622161865
                },
                "Smile": {
                    "Confidence": 93.4493179321289,
                    "Value": false
                },
                "MouthOpen": {
                    "Confidence": 90.53053283691406,
                    "Value": false
                },
                "Quality": {
                    "Sharpness": 95.51618957519531,
                    "Brightness": 65.29893493652344
                },
                "Mustache": {
                    "Confidence": 89.85221099853516,
                    "Value": false
                },
```

```
            "Beard": {
                "Confidence": 86.1991195678711,
                "Value": true
            }
        }
    ]
}
```

詳細については、「Amazon Rekognition 開発者ガイド」の「イメージ内の顔の検出」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「DetectFaces」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;

import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectFaces {
    public static void main(String[] args) {
```

```
        final String usage = """

            Usage:   <bucketName> <sourceImage>

            Where:
                bucketName = The name of the Amazon S3 bucket where the source
image is stored.
                sourceImage - The name of the source image file in the Amazon S3
bucket. (for example, pic1.png).\s
            """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucketName = args[0];
        String sourceImage = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        detectFacesinImage(rekClient, bucketName, sourceImage);
        rekClient.close();
    }

    /**
     * Detects faces in an image stored in an Amazon S3 bucket using the Amazon
Rekognition service.
     *
     * @param rekClient    The Amazon Rekognition client used to interact with
the Rekognition service.
     * @param bucketName   The name of the Amazon S3 bucket where the source
image is stored.
     * @param sourceImage  The name of the source image file in the Amazon S3
bucket.
     */
    public static void detectFacesinImage(RekognitionClient rekClient, String
bucketName, String sourceImage) {
        try {
            S3Object s3ObjectTarget = S3Object.builder()
                .bucket(bucketName)
                .name(sourceImage)
```

```
                .build();

            Image targetImage = Image.builder()
                .s3Object(s3ObjectTarget)
                .build();

            DetectFacesRequest facesRequest = DetectFacesRequest.builder()
                .attributes(Attribute.ALL)
                .image(targetImage)
                .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 e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DetectFaces」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun detectFacesinImage(sourceImage: String?) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }

    val request =
        DetectFacesRequest {
            attributes = listOf(Attribute.All)
            image = souImage
        }

    RekognitionClient.fromEnvironment { 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}")
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API Reference」の「DetectFaces」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンス の「DetectFaces」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DetectLabels**で を使用する

次のサンプルコードは、DetectLabels を使用する方法を説明しています。

詳細については、「イメージ内のラベルを検出する」を参照してください。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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.
/// </summary>
public class DetectLabels
```

```
    {
        public static async Task Main()
        {
            string photo = "del_river_02092020_01.jpg"; // "input.jpg";
            string bucket = "amzn-s3-demo-bucket"; // "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);
            }
        }
    }
```

コンピュータに保存されているイメージファイル内のラベルを検出します。

```
    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.
    /// </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);
            }
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DetectLabels」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
//! Detect instances of real-world entities within an image by using Amazon
 Rekognition
/*!
  \param imageBucket: The Amazon Simple Storage Service (Amazon S3) bucket
 containing an image.
  \param imageKey: The Amazon S3 key of an image object.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::Rekognition::detectLabels(const Aws::String &imageBucket,
                                       const Aws::String &imageKey,
```

```
                                        const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::Rekognition::RekognitionClient rekognitionClient(clientConfiguration);

    Aws::Rekognition::Model::DetectLabelsRequest request;
    Aws::Rekognition::Model::S3Object s3Object;
    s3Object.SetBucket(imageBucket);
    s3Object.SetName(imageKey);

    Aws::Rekognition::Model::Image image;
    image.SetS3Object(s3Object);

    request.SetImage(image);

    const Aws::Rekognition::Model::DetectLabelsOutcome outcome =
 rekognitionClient.DetectLabels(request);

    if (outcome.IsSuccess()) {
        const Aws::Vector<Aws::Rekognition::Model::Label> &labels =
 outcome.GetResult().GetLabels();
        if (labels.empty()) {
            std::cout << "No labels detected" << std::endl;
        } else {
            for (const Aws::Rekognition::Model::Label &label: labels) {
                std::cout << label.GetName() << ": " << label.GetConfidence() <<
 std::endl;
            }
        }
    } else {
        std::cerr << "Error while detecting labels: '"
                  << outcome.GetError().GetMessage()
                  << "'" << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DetectLabels」を参照してください。

CLI

AWS CLI

画像内のラベルを検出するには

次の detect-labels の例は、Amazon S3 バケットに保存されている画像内のシーンとオブジェクトを検出します。

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

出力:

```
{
    "Labels": [
        {
            "Instances": [],
            "Confidence": 99.15271759033203,
            "Parents": [
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Automobile"
        },
        {
            "Instances": [],
            "Confidence": 99.15271759033203,
            "Parents": [
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Vehicle"
        },
        {
            "Instances": [],
            "Confidence": 99.15271759033203,
            "Parents": [],
```

```
            "Name": "Transportation"
        },
        {
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.10616336017847061,
                        "Top": 0.5039216876029968,
                        "Left": 0.0037978808395564556,
                        "Height": 0.18528179824352264
                    },
                    "Confidence": 99.15271759033203
                },
                {
                    "BoundingBox": {
                        "Width": 0.2429988533258438,
                        "Top": 0.5251884460449219,
                        "Left": 0.7309805154800415,
                        "Height": 0.21577216684818268
                    },
                    "Confidence": 99.1286392211914
                },
                {
                    "BoundingBox": {
                        "Width": 0.14233611524105072,
                        "Top": 0.5333095788955688,
                        "Left": 0.6494812965393066,
                        "Height": 0.15528248250484467
                    },
                    "Confidence": 98.48368072509766
                },
                {
                    "BoundingBox": {
                        "Width": 0.11086395382881165,
                        "Top": 0.5354844927787781,
                        "Left": 0.10355594009160995,
                        "Height": 0.10271988064050674
                    },
                    "Confidence": 96.45606231689453
                },
                {
                    "BoundingBox": {
                        "Width": 0.06254628300666809,
                        "Top": 0.5573825240135193,
```

```
                "Left": 0.46083059906959534,
                "Height": 0.053911514580249786
            },
            "Confidence": 93.65448760986328
        },
        {
            "BoundingBox": {
                "Width": 0.10105438530445099,
                "Top": 0.534368634223938,
                "Left": 0.5743985772132874,
                "Height": 0.12226245552301407
            },
            "Confidence": 93.06217193603516
        },
        {
            "BoundingBox": {
                "Width": 0.056389667093753815,
                "Top": 0.5235804319381714,
                "Left": 0.9427769780158997,
                "Height": 0.17163699865341187
            },
            "Confidence": 92.6864013671875
        },
        {
            "BoundingBox": {
                "Width": 0.06003860384225845,
                "Top": 0.5441341400146484,
                "Left": 0.22409997880458832,
                "Height": 0.06737709045410156
            },
            "Confidence": 90.4227066040039
        },
        {
            "BoundingBox": {
                "Width": 0.02848697081208229,
                "Top": 0.5107086896896362,
                "Left": 0,
                "Height": 0.19150497019290924
            },
            "Confidence": 86.65286254882812
        },
        {
            "BoundingBox": {
                "Width": 0.04067881405353546,
```

```
                        "Top": 0.5566273927688599,
                        "Left": 0.316415935754776,
                        "Height": 0.03428703173995018
                    },
                    "Confidence": 85.36471557617188
                },
                {
                    "BoundingBox": {
                        "Width": 0.043411049991846085,
                        "Top": 0.5394920110702515,
                        "Left": 0.18293385207653046,
                        "Height": 0.0893595889210701
                    },
                    "Confidence": 82.21705627441406
                },
                {
                    "BoundingBox": {
                        "Width": 0.031183116137981415,
                        "Top": 0.5579366683959961,
                        "Left": 0.2853088080883026,
                        "Height": 0.03989990055561066
                    },
                    "Confidence": 81.0157470703125
                },
                {
                    "BoundingBox": {
                        "Width": 0.031113790348172188,
                        "Top": 0.5504819750785828,
                        "Left": 0.2580395042896271,
                        "Height": 0.056484755128622055
                    },
                    "Confidence": 56.13441467285156
                },
                {
                    "BoundingBox": {
                        "Width": 0.08586374670267105,
                        "Top": 0.5438792705535889,
                        "Left": 0.5128012895584106,
                        "Height": 0.08550430089235306
                    },
                    "Confidence": 52.37760925292969
                }
            ],
            "Confidence": 99.15271759033203,
```

```
            "Parents": [
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Car"
        },
        {
            "Instances": [],
            "Confidence": 98.9914321899414,
            "Parents": [],
            "Name": "Human"
        },
        {
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.19360728561878204,
                        "Top": 0.35072067379951477,
                        "Left": 0.43734854459762573,
                        "Height": 0.2742200493812561
                    },
                    "Confidence": 98.9914321899414
                },
                {
                    "BoundingBox": {
                        "Width": 0.03801717236638069,
                        "Top": 0.5010883808135986,
                        "Left": 0.9155802130699158,
                        "Height": 0.06597328186035156
                    },
                    "Confidence": 85.02790832519531
                }
            ],
            "Confidence": 98.9914321899414,
            "Parents": [],
            "Name": "Person"
        },
        {
            "Instances": [],
            "Confidence": 93.24951934814453,
```

```
            "Parents": [],
            "Name": "Machine"
        },
        {
            "Instances": [
                {
                    "BoundingBox": {
                        "Width": 0.03561960905790329,
                        "Top": 0.6468243598937988,
                        "Left": 0.7850857377052307,
                        "Height": 0.08878646790981293
                    },
                    "Confidence": 93.24951934814453
                },
                {
                    "BoundingBox": {
                        "Width": 0.02217046171426773,
                        "Top": 0.6149078607559204,
                        "Left": 0.04757237061858177,
                        "Height": 0.07136218994855881
                    },
                    "Confidence": 91.5025863647461
                },
                {
                    "BoundingBox": {
                        "Width": 0.016197510063648224,
                        "Top": 0.6274210214614868,
                        "Left": 0.6472989320755005,
                        "Height": 0.04955997318029404
                    },
                    "Confidence": 85.14686584472656
                },
                {
                    "BoundingBox": {
                        "Width": 0.020207518711686134,
                        "Top": 0.6348286867141724,
                        "Left": 0.7295016646385193,
                        "Height": 0.07059963047504425
                    },
                    "Confidence": 83.34547424316406
                },
                {
                    "BoundingBox": {
                        "Width": 0.020280985161662102,
```

```
                "Top": 0.6171894669532776,
                "Left": 0.08744934946298599,
                "Height": 0.05297485366463661
            },
            "Confidence": 79.9981460571289
        },
        {
            "BoundingBox": {
                "Width": 0.018318990245461464,
                "Top": 0.623889148235321,
                "Left": 0.6836880445480347,
                "Height": 0.06730121374130249
            },
            "Confidence": 78.87144470214844
        },
        {
            "BoundingBox": {
                "Width": 0.021310249343514442,
                "Top": 0.6167286038398743,
                "Left": 0.004064912907779217,
                "Height": 0.08317798376083374
            },
            "Confidence": 75.89361572265625
        },
        {
            "BoundingBox": {
                "Width": 0.03604431077837944,
                "Top": 0.7030032277107239,
                "Left": 0.9254803657531738,
                "Height": 0.04569442570209503
            },
            "Confidence": 64.402587890625
        },
        {
            "BoundingBox": {
                "Width": 0.009834849275648594,
                "Top": 0.5821820497512817,
                "Left": 0.28094568848609924,
                "Height": 0.01964157074689865
            },
            "Confidence": 62.79907989501953
        },
        {
            "BoundingBox": {
```

```
                    "Width": 0.01475677452981472,
                    "Top": 0.6137543320655823,
                    "Left": 0.5950819253921509,
                    "Height": 0.039063986390829086
                },
                "Confidence": 59.40483474731445
            }
        ],
        "Confidence": 93.24951934814453,
        "Parents": [
            {
                "Name": "Machine"
            }
        ],
        "Name": "Wheel"
    },
    {
        "Instances": [],
        "Confidence": 92.61514282226562,
        "Parents": [],
        "Name": "Road"
    },
    {
        "Instances": [],
        "Confidence": 92.37877655029297,
        "Parents": [
            {
                "Name": "Person"
            }
        ],
        "Name": "Sport"
    },
    {
        "Instances": [],
        "Confidence": 92.37877655029297,
        "Parents": [
            {
                "Name": "Person"
            }
        ],
        "Name": "Sports"
    },
    {
        "Instances": [
```

```
                {
                    "BoundingBox": {
                        "Width": 0.12326609343290329,
                        "Top": 0.6332163214683533,
                        "Left": 0.44815489649772644,
                        "Height": 0.058117982000112534
                    },
                    "Confidence": 92.37877655029297
                }
            ],
            "Confidence": 92.37877655029297,
            "Parents": [
                {
                    "Name": "Person"
                },
                {
                    "Name": "Sport"
                }
            ],
            "Name": "Skateboard"
        },
        {
            "Instances": [],
            "Confidence": 90.62931060791016,
            "Parents": [
                {
                    "Name": "Person"
                }
            ],
            "Name": "Pedestrian"
        },
        {
            "Instances": [],
            "Confidence": 88.81334686279297,
            "Parents": [],
            "Name": "Asphalt"
        },
        {
            "Instances": [],
            "Confidence": 88.81334686279297,
            "Parents": [],
            "Name": "Tarmac"
        },
        {
```

```
            "Instances": [],
            "Confidence": 88.23201751708984,
            "Parents": [],
            "Name": "Path"
        },
        {
            "Instances": [],
            "Confidence": 80.26520538330078,
            "Parents": [],
            "Name": "Urban"
        },
        {
            "Instances": [],
            "Confidence": 80.26520538330078,
            "Parents": [
                {
                    "Name": "Building"
                },
                {
                    "Name": "Urban"
                }
            ],
            "Name": "Town"
        },
        {
            "Instances": [],
            "Confidence": 80.26520538330078,
            "Parents": [],
            "Name": "Building"
        },
        {
            "Instances": [],
            "Confidence": 80.26520538330078,
            "Parents": [
                {
                    "Name": "Building"
                },
                {
                    "Name": "Urban"
                }
            ],
            "Name": "City"
        },
        {
```

```
            "Instances": [],
            "Confidence": 78.37934875488281,
            "Parents": [
                {
                    "Name": "Car"
                },
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Parking Lot"
        },
        {
            "Instances": [],
            "Confidence": 78.37934875488281,
            "Parents": [
                {
                    "Name": "Car"
                },
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Parking"
        },
        {
            "Instances": [],
            "Confidence": 74.37590026855469,
            "Parents": [
                {
                    "Name": "Building"
                },
                {
                    "Name": "Urban"
                },
                {
                    "Name": "City"
                }
```

```
            ],
            "Name": "Downtown"
        },
        {
            "Instances": [],
            "Confidence": 69.84622955322266,
            "Parents": [
                {
                    "Name": "Road"
                }
            ],
            "Name": "Intersection"
        },
        {
            "Instances": [],
            "Confidence": 57.68518829345703,
            "Parents": [
                {
                    "Name": "Sports Car"
                },
                {
                    "Name": "Car"
                },
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
                }
            ],
            "Name": "Coupe"
        },
        {
            "Instances": [],
            "Confidence": 57.68518829345703,
            "Parents": [
                {
                    "Name": "Car"
                },
                {
                    "Name": "Vehicle"
                },
                {
                    "Name": "Transportation"
```

```
                }
            ],
            "Name": "Sports Car"
        },
        {
            "Instances": [],
            "Confidence": 56.59492111206055,
            "Parents": [
                {
                    "Name": "Path"
                }
            ],
            "Name": "Sidewalk"
        },
        {
            "Instances": [],
            "Confidence": 56.59492111206055,
            "Parents": [
                {
                    "Name": "Path"
                }
            ],
            "Name": "Pavement"
        },
        {
            "Instances": [],
            "Confidence": 55.58770751953125,
            "Parents": [
                {
                    "Name": "Building"
                },
                {
                    "Name": "Urban"
                }
            ],
            "Name": "Neighborhood"
        }
    ],
    "LabelModelVersion": "2.0"
}
```


詳細については、「Amazon Rekognition 開発者ガイド」の「イメージ内のラベルの検出」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「DetectLabels」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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.*;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectLabels {
    public static void main(String[] args) {
        final String usage = """
            Usage: <bucketName> <sourceImage>

            Where:
                bucketName - The name of the Amazon S3 bucket where the image is
 stored
```

```
                sourceImage - The name of the image file (for example, pic1.png).
\s
            """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucketName = args[0] ;
        String sourceImage = args[1] ;
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        detectImageLabels(rekClient, bucketName, sourceImage);
        rekClient.close();
    }

    /**
     * Detects the labels in an image stored in an Amazon S3 bucket using the
 Amazon Rekognition service.
     *
     * @param rekClient     the Amazon Rekognition client used to make the
 detection request
     * @param bucketName    the name of the Amazon S3 bucket where the image is
 stored
     * @param sourceImage   the name of the image file to be analyzed
     */
    public static void detectImageLabels(RekognitionClient rekClient, String
 bucketName, String sourceImage) {
        try {
            S3Object s3ObjectTarget = S3Object.builder()
                    .bucket(bucketName)
                    .name(sourceImage)
                    .build();

            Image souImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .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 e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DetectLabels」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun detectImageLabels(sourceImage: String) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }
    val request =
        DetectLabelsRequest {
```

```
            image = souImage
            maxLabels = 10
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.detectLabels(request)
        response.labels?.forEach { label ->
            println("${label.name} : ${label.confidence}")
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API Reference」の「DetectLabels」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンス の「DetectLabels」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DetectModerationLabels**で を使用する

次のサンプルコードは、DetectModerationLabels を使用する方法を説明しています。

詳細については、「不適切なイメージを検出する」を参照してください。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    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.
    /// </summary>
    public class DetectModerationLabels
    {
        public static async Task Main(string[] args)
        {
            string photo = "input.jpg";
            string bucket = "amzn-s3-demo-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);
            }
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DetectModerationLabels」を参照してください。

CLI

AWS CLI

画像内の安全でないコンテンツを検出するには

次の `detect-moderation-labels` コマンドは、Amazon S3 バケットに保存されている指定された画像の安全でないコンテンツを検出します。

```
aws rekognition detect-moderation-labels \
    --image "S3Object={Bucket=MyImageS3Bucket,Name=gun.jpg}"
```

出力:

```
{
    "ModerationModelVersion": "3.0",
    "ModerationLabels": [
```

```
        {
            "Confidence": 97.29618072509766,
            "ParentName": "Violence",
            "Name": "Weapon Violence"
        },
        {
            "Confidence": 97.29618072509766,
            "ParentName": "",
            "Name": "Violence"
        }
    ]
}
```

詳細については、「Amazon Rekognition 開発者ガイド」の「不適切なイメージの検出」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「DetectModerationLabels」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
```

```
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectModerationLabels {

    public static void main(String[] args) {
        final String usage = """
            Usage:  <bucketName>  <sourceImage>

            Where:
                bucketName - The name of the S3 bucket where the images are
 stored.
                sourceImage - The name of the image (for example, pic1.png).\s
            """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucketName = args[0];
        String sourceImage = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        detectModLabels(rekClient, bucketName, sourceImage);
        rekClient.close();
    }

    /**
     * Detects moderation labels in an image stored in an Amazon S3 bucket.
     *
     * @param rekClient      the Amazon Rekognition client to use for the
 detection
     * @param bucketName     the name of the Amazon S3 bucket where the image is
 stored
     * @param sourceImage    the name of the image file to be analyzed
     *
     * @throws RekognitionException if there is an error during the image
 detection process
```

```
     */
    public static void detectModLabels(RekognitionClient rekClient, String
 bucketName, String sourceImage) {
        try {
            S3Object s3ObjectTarget = S3Object.builder()
                    .bucket(bucketName)
                    .name(sourceImage)
                    .build();

            Image targetImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .build();

            DetectModerationLabelsRequest moderationLabelsRequest =
 DetectModerationLabelsRequest.builder()
                    .image(targetImage)
                    .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 e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DetectModerationLabels」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun detectModLabels(sourceImage: String) {
    val myImage =
        Image {
            this.bytes = (File(sourceImage).readBytes())
        }

    val request =
        DetectModerationLabelsRequest {
            image = myImage
            minConfidence = 60f
        }

    RekognitionClient.fromEnvironment { 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}")
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API Reference」の「DetectModerationLabels」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「DetectModerationLabels」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DetectText**で を使用する

次のサンプルコードは、DetectText を使用する方法を説明しています。

詳細については、「イメージ内のテキストを検出する」を参照してください。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    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 = "amzn-s3-demo-bucket"; // "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);
            }
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DetectText」を参照してください。

CLI

AWS CLI

画像内のテキストを検出するには

次の detect-text コマンドは、指定された画像内のテキストを検出します。

```
aws rekognition detect-text \
    --image '{"S3Object":
{"Bucket":"MyImageS3Bucket","Name":"ExamplePicture.jpg"}}'
```

出力:

```
{
    "TextDetections": [
        {
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.24624845385551453,
                    "Top": 0.28288066387176514,
                    "Left": 0.391388863325119,
                    "Height": 0.022687450051307678
                },
                "Polygon": [
                    {
                        "Y": 0.28288066387176514,
                        "X": 0.391388863325119
                    },
                    {
```

```
                        "Y": 0.2826388478279114,
                        "X": 0.6376373171806335
                    },
                    {
                        "Y": 0.30532628297805786,
                        "X": 0.637677013874054
                    },
                    {
                        "Y": 0.305568128824234,
                        "X": 0.39142853021621704
                    }
                ]
            },
            "Confidence": 94.35709381103516,
            "DetectedText": "ESTD 1882",
            "Type": "LINE",
            "Id": 0
        },
        {
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.33933889865875244,
                    "Top": 0.32603850960731506,
                    "Left": 0.34534579515457153,
                    "Height": 0.07126858830451965
                },
                "Polygon": [
                    {
                        "Y": 0.32603850960731506,
                        "X": 0.34534579515457153
                    },
                    {
                        "Y": 0.32633158564567566,
                        "X": 0.684684693813324
                    },
                    {
                        "Y": 0.3976001739501953,
                        "X": 0.684575080871582
                    },
                    {
                        "Y": 0.3973070979118347,
                        "X": 0.345236212015152
                    }
                ]
```

```
            },
            "Confidence": 99.95779418945312,
            "DetectedText": "BRAINS",
            "Type": "LINE",
            "Id": 1
        },
        {
            "Confidence": 97.22098541259766,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.061079490929841995,
                    "Top": 0.2843210697174072,
                    "Left": 0.391391396522522,
                    "Height": 0.021029088646173477
                },
                "Polygon": [
                    {
                        "Y": 0.2843210697174072,
                        "X": 0.391391396522522
                    },
                    {
                        "Y": 0.2828207015991211,
                        "X": 0.4524524509906769
                    },
                    {
                        "Y": 0.3038259446620941,
                        "X": 0.4534534513950348
                    },
                    {
                        "Y": 0.30532634258270264,
                        "X": 0.3923923969268799
                    }
                ]
            },
            "DetectedText": "ESTD",
            "ParentId": 0,
            "Type": "WORD",
            "Id": 2
        },
        {
            "Confidence": 91.49320983886719,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.07007007300853729,
```

```
                    "Top": 0.2828207015991211,
                    "Left": 0.5675675868988037,
                    "Height": 0.02250562608242035
                },
                "Polygon": [
                    {
                        "Y": 0.2828207015991211,
                        "X": 0.5675675868988037
                    },
                    {
                        "Y": 0.2828207015991211,
                        "X": 0.6376376152038574
                    },
                    {
                        "Y": 0.30532634258270264,
                        "X": 0.6376376152038574
                    },
                    {
                        "Y": 0.30532634258270264,
                        "X": 0.5675675868988037
                    }
                ]
            },
            "DetectedText": "1882",
            "ParentId": 0,
            "Type": "WORD",
            "Id": 3
        },
        {
            "Confidence": 99.95779418945312,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.33933934569358826,
                    "Top": 0.32633158564567566,
                    "Left": 0.3453453481197357,
                    "Height": 0.07127484679222107
                },
                "Polygon": [
                    {
                        "Y": 0.32633158564567566,
                        "X": 0.3453453481197357
                    },
                    {
                        "Y": 0.32633158564567566,
```

```
                        "X": 0.684684693813324
                    },
                    {
                        "Y": 0.39759939908981323,
                        "X": 0.6836836934089661
                    },
                    {
                        "Y": 0.39684921503067017,
                        "X": 0.3453453481197357
                    }
                ]
            },
            "DetectedText": "BRAINS",
            "ParentId": 1,
            "Type": "WORD",
            "Id": 4
        }
    ]
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「DetectText」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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.*;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
```

```
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectText {
    public static void main(String[] args) {
        final String usage = "\n" +
            "Usage:   <bucketName> <sourceImage>\n" +
            "\n" +
            "Where:\n" +
            "   bucketName - The name of the S3 bucket where the image is stored
\n" +
            "   sourceImage - The path to the image that contains text (for
 example, pic1.png). \n";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucketName = args[0];
        String sourceImage = args[1];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        detectTextLabels(rekClient, bucketName, sourceImage);
        rekClient.close();
    }

    /**
     * Detects text labels in an image stored in an S3 bucket using Amazon
 Rekognition.
     *
     * @param rekClient    an instance of the Amazon Rekognition client
     * @param bucketName   the name of the S3 bucket where the image is stored
```

```
     * @param sourceImage  the name of the image file in the S3 bucket
     * @throws RekognitionException if an error occurs while calling the Amazon
 Rekognition API
     */
    public static void detectTextLabels(RekognitionClient rekClient, String
 bucketName, String sourceImage) {
        try {
            S3Object s3ObjectTarget = S3Object.builder()
                    .bucket(bucketName)
                    .name(sourceImage)
                    .build();

            Image souImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .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 e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DetectText」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun detectTextLabels(sourceImage: String?) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }

    val request =
        DetectTextRequest {
            image = souImage
        }

    RekognitionClient.fromEnvironment { 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}")
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API Reference」の「DetectText」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
```

- API の詳細については、https://docs.aws.amazon.com/goto/boto3/rekognition-2016-06-27/DetectText SDK for Python (Boto3) API リファレンス の「AWS DetectText」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DisassociateFaces**で を使用する

次のサンプルコードは、DisassociateFaces を使用する方法を説明しています。

CLI

AWS CLI

```
aws rekognition disassociate-faces --face-ids list-of-face-ids
  --user-id user-id --collection-id collection-name --region region-name
```

- API の詳細については、AWS CLI コマンドリファレンスの「DisassociateFaces」を参照してください。

Python

SDK for Python (Boto3)

```
from botocore.exceptions import ClientError
import boto3
import logging

logger = logging.getLogger(__name__)
session = boto3.Session(profile_name='profile-name')
client = session.client('rekognition')
```

```
def disassociate_faces(collection_id, user_id, face_ids):
    """
    Disassociate stored faces within collection to the given user

    :param collection_id: The ID of the collection where user and faces are
 stored.
    :param user_id: The ID of the user that we want to disassociate faces from
    :param face_ids: The list of face IDs to be disassociated from the given user

    :return: response of AssociateFaces API
    """
    logger.info(f'Disssociating faces from user: {user_id}, {face_ids}')
    try:
        response = client.disassociate_faces(
            CollectionId=collection_id,
            UserId=user_id,
            FaceIds=face_ids
        )
        print(f'- disassociated {len(response["DisassociatedFaces"])} faces')
    except ClientError:
        logger.exception("Failed to disassociate faces from the given user")
        raise
    else:
        print(response)
        return response

def main():
    face_ids = ["faceId1", "faceId2"]
    collection_id = "collection-id"
    user_id = "user-id"
    disassociate_faces(collection_id, user_id, face_ids)

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

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DisassociateFaces」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **GetCelebrityInfo**で を使用する

次のサンプルコードは、GetCelebrityInfo を使用する方法を説明しています。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    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.
    /// </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);
            });
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetCelebrityInfo」を参照してください。

CLI

AWS CLI

有名人に関する情報を取得するには

次の get-celebrity-info コマンドは、指定された有名人に関する詳細情報を表示します。id パラメータは以前に recognize-celebrities を呼び出したときのものです。

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

出力:

```
{
    "Name": "Celeb A",
    "Urls": [
        "www.imdb.com/name/aaaaaaaaa"
    ]
}
```

詳細については、「Amazon Rekognition 開発者ガイド」の「有名人に関する情報の取得」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「GetCelebrityInfo」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **IndexFaces**で を使用する

次のサンプルコードは、IndexFaces を使用する方法を説明しています。

詳細については、「コレクションに顔を追加する」を参照してください。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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.
/// </summary>
public class AddFaces
{
    public static async Task Main()
    {
        string collectionId = "MyCollection2";
        string bucket = "amzn-s3-demo-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}");
            }
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「IndexFaces」を参照してください。

CLI

AWS CLI

コレクションに顔を追加するには

次の index-faces コマンドは、画像内の顔を指定されたコレクションに追加します。

```
aws rekognition index-faces \
    --image '{"S3Object":{"Bucket":"MyVideoS3Bucket","Name":"MyPicture.jpg"}}' \
```

```
    --collection-id MyCollection \
    --max-faces 1 \
    --quality-filter "AUTO" \
    --detection-attributes "ALL" \
    --external-image-id "MyPicture.jpg"
```

出力:

```
{
    "FaceRecords": [
        {
            "FaceDetail": {
                "Confidence": 99.993408203125,
                "Eyeglasses": {
                    "Confidence": 99.11750030517578,
                    "Value": false
                },
                "Sunglasses": {
                    "Confidence": 99.98249053955078,
                    "Value": false
                },
                "Gender": {
                    "Confidence": 99.92769622802734,
                    "Value": "Male"
                },
                "Landmarks": [
                    {
                        "Y": 0.26750367879867554,
                        "X": 0.6202793717384338,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.26642778515815735,
                        "X": 0.6787431836128235,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.31361380219459534,
                        "X": 0.6421601176261902,
                        "Type": "nose"
                    },
                    {
                        "Y": 0.3495299220085144,
```

```
                    "X": 0.6216195225715637,
                    "Type": "mouthLeft"
                },
                {
                    "Y": 0.35194727778434753,
                    "X": 0.669899046421051,
                    "Type": "mouthRight"
                },
                {
                    "Y": 0.26844894886016846,
                    "X": 0.6210268139839172,
                    "Type": "leftPupil"
                },
                {
                    "Y": 0.26707562804222107,
                    "X": 0.6817160844802856,
                    "Type": "rightPupil"
                },
                {
                    "Y": 0.24834522604942322,
                    "X": 0.6018546223640442,
                    "Type": "leftEyeBrowLeft"
                },
                {
                    "Y": 0.24397172033786774,
                    "X": 0.6172008514404297,
                    "Type": "leftEyeBrowUp"
                },
                {
                    "Y": 0.24677404761314392,
                    "X": 0.6339119076728821,
                    "Type": "leftEyeBrowRight"
                },
                {
                    "Y": 0.24582654237747192,
                    "X": 0.6619398593902588,
                    "Type": "rightEyeBrowLeft"
                },
                {
                    "Y": 0.23973053693771362,
                    "X": 0.6804757118225098,
                    "Type": "rightEyeBrowUp"
                },
                {
```

```
                        "Y": 0.24441994726657867,
                        "X": 0.6978968977928162,
                        "Type": "rightEyeBrowRight"
                    },
                    {
                        "Y": 0.2695908546447754,
                        "X": 0.6085202693939209,
                        "Type": "leftEyeLeft"
                    },
                    {
                        "Y": 0.26716896891593933,
                        "X": 0.6315826177597046,
                        "Type": "leftEyeRight"
                    },
                    {
                        "Y": 0.26289820671081543,
                        "X": 0.6202316880226135,
                        "Type": "leftEyeUp"
                    },
                    {
                        "Y": 0.27123287320137024,
                        "X": 0.6205548048019409,
                        "Type": "leftEyeDown"
                    },
                    {
                        "Y": 0.2668408751487732,
                        "X": 0.6663622260093689,
                        "Type": "rightEyeLeft"
                    },
                    {
                        "Y": 0.26741549372673035,
                        "X": 0.6910083889961243,
                        "Type": "rightEyeRight"
                    },
                    {
                        "Y": 0.2614026665687561,
                        "X": 0.6785826086997986,
                        "Type": "rightEyeUp"
                    },
                    {
                        "Y": 0.27075251936912537,
                        "X": 0.6789616942405701,
                        "Type": "rightEyeDown"
                    },
```

```
                    {
                        "Y": 0.3211299479007721,
                        "X": 0.6324167847633362,
                        "Type": "noseLeft"
                    },
                    {
                        "Y": 0.32276326417922974,
                        "X": 0.6558475494384766,
                        "Type": "noseRight"
                    },
                    {
                        "Y": 0.34385165572166443,
                        "X": 0.6444970965385437,
                        "Type": "mouthUp"
                    },
                    {
                        "Y": 0.3671635091304779,
                        "X": 0.6459195017814636,
                        "Type": "mouthDown"
                    }
                ],
                "Pose": {
                    "Yaw": -9.54541015625,
                    "Roll": -0.5709401965141296,
                    "Pitch": 0.6045494675636292
                },
                "Emotions": [
                    {
                        "Confidence": 39.90074157714844,
                        "Type": "HAPPY"
                    },
                    {
                        "Confidence": 23.38753890991211,
                        "Type": "CALM"
                    },
                    {
                        "Confidence": 5.840933322906494,
                        "Type": "CONFUSED"
                    }
                ],
                "AgeRange": {
                    "High": 63,
                    "Low": 45
                },
```

```
                "EyesOpen": {
                    "Confidence": 99.80887603759766,
                    "Value": true
                },
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618015021085739,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770642817020416
                },
                "Smile": {
                    "Confidence": 99.69740295410156,
                    "Value": false
                },
                "MouthOpen": {
                    "Confidence": 99.97393798828125,
                    "Value": false
                },
                "Quality": {
                    "Sharpness": 95.54405975341797,
                    "Brightness": 63.867706298828125
                },
                "Mustache": {
                    "Confidence": 97.05007934570312,
                    "Value": false
                },
                "Beard": {
                    "Confidence": 87.34505462646484,
                    "Value": false
                }
            },
            "Face": {
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618015021085739,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770642817020416
                },
                "FaceId": "ce7ed422-2132-4a11-ab14-06c5c410f29f",
                "ExternalImageId": "example-image.jpg",
                "Confidence": 99.993408203125,
                "ImageId": "8d67061e-90d2-598f-9fbd-29c8497039c0"
            }
        }
```

```
    ],
    "UnindexedFaces": [],
    "FaceModelVersion": "3.0",
    "OrientationCorrection": "ROTATE_0"
}
```

詳細については、「Amazon Rekognition 開発者ガイド」の「コレクションへの顔の追加」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「IndexFaces」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.*;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class AddFacesToCollection {
    public static void main(String[] args) {
        final String usage = """
            Usage: <collectionId> <sourceImage> <bucketName>

            Where:
```

```
                collectionName - The name of the collection.
                sourceImage - The name of the image (for example, pic1.png).
                bucketName - The name of the S3 bucket.
            """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        String sourceImage = args[1];
        String bucketName = args[2];;
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        addToCollection(rekClient, collectionId, bucketName, sourceImage);
        rekClient.close();
    }

    /**
     * Adds a face from an image to an Amazon Rekognition collection.
     *
     * @param rekClient     the Amazon Rekognition client
     * @param collectionId  the ID of the collection to add the face to
     * @param bucketName    the name of the Amazon S3 bucket containing the image
     * @param sourceImage   the name of the image file to add to the collection
     * @throws RekognitionException if there is an error while interacting with
the Amazon Rekognition service
     */
    public static void addToCollection(RekognitionClient rekClient, String
collectionId, String bucketName, String sourceImage) {
        try {
            S3Object s3ObjectTarget = S3Object.builder()
                    .bucket(bucketName)
                    .name(sourceImage)
                    .build();

            Image targetImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .build();
```

```
            IndexFacesRequest facesRequest = IndexFacesRequest.builder()
                    .collectionId(collectionId)
                    .image(targetImage)
                    .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 e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「IndexFaces」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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.fromEnvironment { 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")
            }
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API Reference」の「IndexFaces」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「IndexFaces」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **ListCollections**で を使用する

次のサンプルコードは、ListCollections を使用する方法を説明しています。

コレクションの詳細については、「コレクションを一覧表示する」を参照してください。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.Threading.Tasks;
```

```
    using Amazon.Rekognition;
    using Amazon.Rekognition.Model;

    /// <summary>
    /// Uses Amazon Rekognition to list the collection IDs in the
    /// current account.
    /// </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);
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListCollections」を参照してください。

CLI

AWS CLI

使用可能なコレクションを一覧表示するには

次のlist-collectionsコマンドは、 AWS アカウントで使用可能なコレクションを一覧表示します。

```
aws rekognition list-collections
```

出力:

```
{
    "FaceModelVersions": [
        "2.0",
        "3.0",
        "3.0",
        "3.0",
        "4.0",
        "1.0",
        "3.0",
        "4.0",
        "4.0",
        "4.0"
    ],
    "CollectionIds": [
        "MyCollection1",
        "MyCollection2",
        "MyCollection3",
        "MyCollection4",
        "MyCollection5",
        "MyCollection6",
        "MyCollection7",
        "MyCollection8",
        "MyCollection9",
        "MyCollection10"
```

```
    ]
}
```

詳細については、「Amazon Rekognition 開発者ガイド」の「コレクションの一覧表示」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「ListCollections」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.ListCollectionsRequest;
import software.amazon.awssdk.services.rekognition.model.ListCollectionsResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListCollections {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();
```

```
        System.out.println("Listing collections");
        listAllCollections(rekClient);
        rekClient.close();
    }

    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);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「ListCollections」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun listAllCollections() {
```

```
    val request =
        ListCollectionsRequest {
            maxResults = 10
        }

    RekognitionClient.fromEnvironment { region = "us-east-1" }.use { rekClient ->
        val response = rekClient.listCollections(request)
        response.collectionIds?.forEach { resultId ->
            println(resultId)
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API Reference」の「ListCollections」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「ListCollections」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **ListFaces**で を使用する

次のサンプルコードは、ListFaces を使用する方法を説明しています。

詳細については、「コレクションに顔を保存する」を参照してください。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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.
/// </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));
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListFaces」を参照してください。

CLI

AWS CLI

コレクション内の顔を一覧表示するには

次の `list-faces` コマンドは、指定されたコレクション内の顔を一覧表示します。

```
aws rekognition list-faces \
    --collection-id MyCollection
```

出力:

```
{
    "FaceModelVersion": "3.0",
    "Faces": [
        {
            "BoundingBox": {
                "Width": 0.5216310024261475,
                "Top": 0.3256250023841858,
                "Left": 0.13394300639629364,
                "Height": 0.3918749988079071
            },
            "FaceId": "0040279c-0178-436e-b70a-e61b074e96b0",
            "ExternalImageId": "image1.jpg",
            "Confidence": 100.0,
            "ImageId": "f976e487-3719-5e2d-be8b-ea2724c26991"
        },
```

```
        {
            "BoundingBox": {
                "Width": 0.5074880123138428,
                "Top": 0.3774999976158142,
                "Left": 0.18302799761295319,
                "Height": 0.3812499940395355
            },
            "FaceId": "086261e8-6deb-4bc0-ac73-ab22323cc38d",
            "ExternalImageId": "image2.jpg",
            "Confidence": 99.99930572509766,
            "ImageId": "ae1593b0-a8f6-5e24-a306-abf529e276fa"
        },
        {
            "BoundingBox": {
                "Width": 0.5574039816856384,
                "Top": 0.37187498807907104,
                "Left": 0.14559100568294525,
                "Height": 0.4181250035762787
            },
            "FaceId": "11c4bd3c-19c5-4eb8-aecc-24feb93a26e1",
            "ExternalImageId": "image3.jpg",
            "Confidence": 99.99960327148438,
            "ImageId": "80739b4d-883f-5b78-97cf-5124038e26b9"
        },
        {
            "BoundingBox": {
                "Width": 0.18562500178813934,
                "Top": 0.1618019938468933,
                "Left": 0.5575000047683716,
                "Height": 0.24770599603652954
            },
            "FaceId": "13692fe4-990a-4679-b14a-5ac23d135eab",
            "ExternalImageId": "image4.jpg",
            "Confidence": 99.99340057373047,
            "ImageId": "8df18239-9ad1-5acd-a46a-6581ff98f51b"
        },
        {
            "BoundingBox": {
                "Width": 0.5307819843292236,
                "Top": 0.2862499952316284,
                "Left": 0.1564060002565384,
                "Height": 0.3987500071525574
            },
            "FaceId": "2eb5f3fd-e2a9-4b1c-a89f-afa0a518fe06",
```

```
            "ExternalImageId": "image5.jpg",
            "Confidence": 99.99970245361328,
            "ImageId": "3c314792-197d-528d-bbb6-798ed012c150"
        },
        {
            "BoundingBox": {
                "Width": 0.5773710012435913,
                "Top": 0.34437501430511475,
                "Left": 0.12396000325679779,
                "Height": 0.4337500035762787
            },
            "FaceId": "57189455-42b0-4839-a86c-abda48b13174",
            "ExternalImageId": "image6.jpg",
            "Confidence": 100.0,
            "ImageId": "0aff2f37-e7a2-5dbc-a3a3-4ef6ec18eaa0"
        },
        {
            "BoundingBox": {
                "Width": 0.5349419713020325,
                "Top": 0.29124999046325684,
                "Left": 0.16389399766921997,
                "Height": 0.40187498927116394
            },
            "FaceId": "745f7509-b1fa-44e0-8b95-367b1359638a",
            "ExternalImageId": "image7.jpg",
            "Confidence": 99.99979400634766,
            "ImageId": "67a34327-48d1-5179-b042-01e52ccfeada"
        },
        {
            "BoundingBox": {
                "Width": 0.41499999165534973,
                "Top": 0.09187500178813934,
                "Left": 0.28083300590515137,
                "Height": 0.3112500011920929
            },
            "FaceId": "8d3cfc70-4ba8-4b36-9644-90fba29c2dac",
            "ExternalImageId": "image8.jpg",
            "Confidence": 99.99769592285156,
            "ImageId": "a294da46-2cb1-5cc4-9045-61d7ca567662"
        },
        {
            "BoundingBox": {
                "Width": 0.48166701197624207,
                "Top": 0.20999999344348907,
```

```
                "Left": 0.21250000596046448,
                "Height": 0.36125001311302185
            },
            "FaceId": "bd4ceb4d-9acc-4ab7-8ef8-1c2d2ba0a66a",
            "ExternalImageId": "image9.jpg",
            "Confidence": 99.99949645996094,
            "ImageId": "5e1a7588-e5a0-5ee3-bd00-c642518dfe3a"
        },
        {
            "BoundingBox": {
                "Width": 0.18562500178813934,
                "Top": 0.1618019938468933,
                "Left": 0.5575000047683716,
                "Height": 0.24770599603652954
            },
            "FaceId": "ce7ed422-2132-4a11-ab14-06c5c410f29f",
            "ExternalImageId": "image10.jpg",
            "Confidence": 99.99340057373047,
            "ImageId": "8d67061e-90d2-598f-9fbd-29c8497039c0"
        }
    ]
}
```

詳細については、「Amazon Rekognition 開発者ガイド」の「コレクション内の顔と関連するユーザーを一覧表示します。」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「ListFaces」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.Face;
```

```
import software.amazon.awssdk.services.rekognition.model.ListFacesRequest;
import software.amazon.awssdk.services.rekognition.model.ListFacesResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListFacesInCollection {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <collectionId>

                Where:
                   collectionId - The name of the collection.\s
                """;

        if (args.length < 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Faces in collection " + collectionId);
        listFacesCollection(rekClient, collectionId);
        rekClient.close();
    }

    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);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「ListFaces」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun listFacesCollection(collectionIdVal: String?) {
    val request =
        ListFacesRequest {
            collectionId = collectionIdVal
            maxResults = 10
        }

    RekognitionClient.fromEnvironment { 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}")
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API Reference」の「ListFaces」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
```

- API の詳細については、https://docs.aws.amazon.com/goto/boto3/rekognition-2016-06-27/ListFaces SDK for Python (Boto3) API リファレンス で「AWS ListFaces」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **RecognizeCelebrities**で を使用する

次のサンプルコードは、RecognizeCelebrities を使用する方法を説明しています。

詳細については、「イメージ内で有名人を認識する」を参照してください。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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.
/// </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.");
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「RecognizeCelebrities」を参照してください。

CLI

AWS CLI

画像内の有名人を認識するには

次の recognize-celebrities コマンドは、Amazon S3 バケットに保存されている指定された画像の有名人を識別します。

```
aws rekognition recognize-celebrities \
    --image "S3Object={Bucket=MyImageS3Bucket,Name=moviestars.jpg}"
```

出力:

```
{
    "UnrecognizedFaces": [
        {
            "BoundingBox": {
                "Width": 0.14416666328907013,
                "Top": 0.07777778059244156,
                "Left": 0.625,
                "Height": 0.2746031880378723
            },
            "Confidence": 99.9990234375,
            "Pose": {
                "Yaw": 10.80408763885498,
                "Roll": -12.761146545410156,
                "Pitch": 10.96889877319336
            },
            "Quality": {
                "Sharpness": 94.1185531616211,
```

```
                "Brightness": 79.18367004394531
            },
            "Landmarks": [
                {
                    "Y": 0.18220913410186768,
                    "X": 0.6702951788902283,
                    "Type": "eyeLeft"
                },
                {
                    "Y": 0.16337193548679352,
                    "X": 0.7188183665275574,
                    "Type": "eyeRight"
                },
                {
                    "Y": 0.20739148557186127,
                    "X": 0.7055801749229431,
                    "Type": "nose"
                },
                {
                    "Y": 0.2889308035373688,
                    "X": 0.687512218952179,
                    "Type": "mouthLeft"
                },
                {
                    "Y": 0.2706988751888275,
                    "X": 0.7250053286552429,
                    "Type": "mouthRight"
                }
            ]
        }
    ],
    "CelebrityFaces": [
        {
            "MatchConfidence": 100.0,
            "Face": {
                "BoundingBox": {
                    "Width": 0.14000000059604645,
                    "Top": 0.1190476194024086,
                    "Left": 0.82833331823349,
                    "Height": 0.2666666805744171
                },
                "Confidence": 99.99359130859375,
                "Pose": {
                    "Yaw": -10.509642601013184,
```

```
                    "Roll": -14.51749324798584,
                    "Pitch": 13.799399375915527
                },
                "Quality": {
                    "Sharpness": 78.74752044677734,
                    "Brightness": 42.201324462890625
                },
                "Landmarks": [
                    {
                        "Y": 0.2290833294391632,
                        "X": 0.8709492087364197,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.20639978349208832,
                        "X": 0.9153988361358643,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.25417643785476685,
                        "X": 0.8907724022865295,
                        "Type": "nose"
                    },
                    {
                        "Y": 0.32729196548461914,
                        "X": 0.8876466155052185,
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.3115464746952057,
                        "X": 0.9238573312759399,
                        "Type": "mouthRight"
                    }
                ]
            },
            "Name": "Celeb A",
            "Urls": [
                "www.imdb.com/name/aaaaaaaaa"
            ],
            "Id": "1111111"
        },
        {
            "MatchConfidence": 97.0,
            "Face": {
```

```
                "BoundingBox": {
                    "Width": 0.13333334028720856,
                    "Top": 0.24920634925365448,
                    "Left": 0.4449999928474426,
                    "Height": 0.2539682686328888
                },
                "Confidence": 99.99979400634766,
                "Pose": {
                    "Yaw": 6.557040691375732,
                    "Roll": -7.316643714904785,
                    "Pitch": 9.272967338562012
                },
                "Quality": {
                    "Sharpness": 83.23492431640625,
                    "Brightness": 78.83267974853516
                },
                "Landmarks": [
                    {
                        "Y": 0.3625510632991791,
                        "X": 0.48898839950561523,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.35366007685661316,
                        "X": 0.5313721299171448,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.3894785940647125,
                        "X": 0.5173314809799194,
                        "Type": "nose"
                    },
                    {
                        "Y": 0.44889405369758606,
                        "X": 0.5020005702972412,
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.4408611059188843,
                        "X": 0.5351271629333496,
                        "Type": "mouthRight"
                    }
                ]
            },
```

```
            "Name": "Celeb B",
            "Urls": [
                "www.imdb.com/name/bbbbbbbbb"
            ],
            "Id": "2222222"
        },
        {
            "MatchConfidence": 100.0,
            "Face": {
                "BoundingBox": {
                    "Width": 0.12416666746139526,
                    "Top": 0.2968254089355469,
                    "Left": 0.2150000035762787,
                    "Height": 0.23650793731212616
                },
                "Confidence": 99.99958801269531,
                "Pose": {
                    "Yaw": 7.801797866821289,
                    "Roll": -8.326810836791992,
                    "Pitch": 7.844768047332764
                },
                "Quality": {
                    "Sharpness": 86.93206024169922,
                    "Brightness": 79.81291198730469
                },
                "Landmarks": [
                    {
                        "Y": 0.4027804136276245,
                        "X": 0.2575301229953766,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.3934555947780609,
                        "X": 0.2956969439983368,
                        "Type": "eyeRight"
                    },
                    {
                        "Y": 0.4309830069541931,
                        "X": 0.2837020754814148,
                        "Type": "nose"
                    },
                    {
                        "Y": 0.48186683654785156,
                        "X": 0.26812544465065,
```

```
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.47338807582855225,
                        "X": 0.29905644059181213,
                        "Type": "mouthRight"
                    }
                ]
            },
            "Name": "Celeb C",
            "Urls": [
                "www.imdb.com/name/ccccccccc"
            ],
            "Id": "3333333"
        },
        {
            "MatchConfidence": 97.0,
            "Face": {
                "BoundingBox": {
                    "Width": 0.11916666477918625,
                    "Top": 0.3698412775993347,
                    "Left": 0.008333333767950535,
                    "Height": 0.22698412835597992
                },
                "Confidence": 99.99999237060547,
                "Pose": {
                    "Yaw": 16.38478660583496,
                    "Roll": -1.0260354280471802,
                    "Pitch": 5.975185394287109
                },
                "Quality": {
                    "Sharpness": 83.23492431640625,
                    "Brightness": 61.408443450927734
                },
                "Landmarks": [
                    {
                        "Y": 0.4632347822189331,
                        "X": 0.049406956881284714,
                        "Type": "eyeLeft"
                    },
                    {
                        "Y": 0.46388113498687744,
                        "X": 0.08722897619009018,
                        "Type": "eyeRight"
```

```
                    },
                    {
                        "Y": 0.5020678639411926,
                        "X": 0.0758260041475296,
                        "Type": "nose"
                    },
                    {
                        "Y": 0.544157862663269,
                        "X": 0.054029736667871475,
                        "Type": "mouthLeft"
                    },
                    {
                        "Y": 0.5463630557060242,
                        "X": 0.08464983850717545,
                        "Type": "mouthRight"
                    }
                ]
            },
            "Name": "Celeb D",
            "Urls": [
                "www.imdb.com/name/ddddddddd"
            ],
            "Id": "4444444"
        }
    ]
}
```

詳細については、「Amazon Rekognition 開発者ガイド」の「イメージ内の有名人の認識」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「RecognizeCelebrities」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.core.SdkBytes;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

import software.amazon.awssdk.services.rekognition.model.*;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class RecognizeCelebrities {
    public static void main(String[] args) {
        final String usage = """
                Usage:   <bucketName> <sourceImage>

                Where:
                   bucketName - The name of the S3 bucket where the images are
 stored.
                   sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png).\s
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
       }

        String bucketName = args[0];;
        String sourceImage = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();
```

```
        System.out.println("Locating celebrities in " + sourceImage);
        recognizeAllCelebrities(rekClient, bucketName, sourceImage);
        rekClient.close();
    }

    /**
     * Recognizes all celebrities in an image stored in an Amazon S3 bucket.
     *
     * @param rekClient    the Amazon Rekognition client used to perform the
 celebrity recognition operation
     * @param bucketName   the name of the Amazon S3 bucket where the source
 image is stored
     * @param sourceImage  the name of the source image file stored in the Amazon
 S3 bucket
     */
    public static void recognizeAllCelebrities(RekognitionClient rekClient,
 String bucketName, String sourceImage) {
        try {
            S3Object s3ObjectTarget = S3Object.builder()
                .bucket(bucketName)
                .name(sourceImage)
                .build();

            Image souImage = Image.builder()
                    .s3Object(s3ObjectTarget)
                    .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 e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「RecognizeCelebrities」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun recognizeAllCelebrities(sourceImage: String?) {
    val souImage =
        Image {
            bytes = (File(sourceImage).readBytes())
        }

    val request =
        RecognizeCelebritiesRequest {
            image = souImage
        }

    RekognitionClient.fromEnvironment { 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.")
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API Reference」の「RecognizeCelebrities」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンスの RecognizeCelebrities」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **SearchFaces**で を使用する

次のサンプルコードは、SearchFaces を使用する方法を説明しています。

詳細については、顔 (フェイス ID) を検索する を参照してください。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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.
/// </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}");
            });
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「SearchFaces」を参照してください。

CLI

AWS CLI

顔 ID に一致するコレクション内の顔を検索するには

次の search-faces コマンドは、指定され顔 ID に一致するコレクション内の顔を検索します。

```
aws rekognition search-faces \
    --face-id 8d3cfc70-4ba8-4b36-9644-90fba29c2dac \
    --collection-id MyCollection
```

出力:

```
{
    "SearchedFaceId": "8d3cfc70-4ba8-4b36-9644-90fba29c2dac",
    "FaceModelVersion": "3.0",
```

```
    "FaceMatches": [
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.48166701197624207,
                    "Top": 0.20999999344348907,
                    "Left": 0.21250000596046448,
                    "Height": 0.36125001311302185
                },
                "FaceId": "bd4ceb4d-9acc-4ab7-8ef8-1c2d2ba0a66a",
                "ExternalImageId": "image1.jpg",
                "Confidence": 99.99949645996094,
                "ImageId": "5e1a7588-e5a0-5ee3-bd00-c642518dfe3a"
            },
            "Similarity": 99.30997467041016
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618019938468933,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770599603652954
                },
                "FaceId": "ce7ed422-2132-4a11-ab14-06c5c410f29f",
                "ExternalImageId": "example-image.jpg",
                "Confidence": 99.99340057373047,
                "ImageId": "8d67061e-90d2-598f-9fbd-29c8497039c0"
            },
            "Similarity": 99.24862670898438
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618019938468933,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770599603652954
                },
                "FaceId": "13692fe4-990a-4679-b14a-5ac23d135eab",
                "ExternalImageId": "image3.jpg",
                "Confidence": 99.99340057373047,
                "ImageId": "8df18239-9ad1-5acd-a46a-6581ff98f51b"
            },
```

```
            "Similarity": 99.24862670898438
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5349419713020325,
                    "Top": 0.29124999046325684,
                    "Left": 0.16389399766921997,
                    "Height": 0.40187498927116394
                },
                "FaceId": "745f7509-b1fa-44e0-8b95-367b1359638a",
                "ExternalImageId": "image9.jpg",
                "Confidence": 99.99979400634766,
                "ImageId": "67a34327-48d1-5179-b042-01e52ccfeada"
            },
            "Similarity": 96.73158264160156
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5307819843292236,
                    "Top": 0.2862499952316284,
                    "Left": 0.1564060002565384,
                    "Height": 0.3987500071525574
                },
                "FaceId": "2eb5f3fd-e2a9-4b1c-a89f-afa0a518fe06",
                "ExternalImageId": "image10.jpg",
                "Confidence": 99.99970245361328,
                "ImageId": "3c314792-197d-528d-bbb6-798ed012c150"
            },
            "Similarity": 96.48291015625
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5074880123138428,
                    "Top": 0.3774999976158142,
                    "Left": 0.18302799761295319,
                    "Height": 0.3812499940395355
                },
                "FaceId": "086261e8-6deb-4bc0-ac73-ab22323cc38d",
                "ExternalImageId": "image6.jpg",
                "Confidence": 99.99930572509766,
                "ImageId": "ae1593b0-a8f6-5e24-a306-abf529e276fa"
```

```
            },
            "Similarity": 96.43287658691406
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5574039816856384,
                    "Top": 0.37187498807907104,
                    "Left": 0.14559100568294525,
                    "Height": 0.4181250035762787
                },
                "FaceId": "11c4bd3c-19c5-4eb8-aecc-24feb93a26e1",
                "ExternalImageId": "image5.jpg",
                "Confidence": 99.99960327148438,
                "ImageId": "80739b4d-883f-5b78-97cf-5124038e26b9"
            },
            "Similarity": 95.25305938720703
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5773710012435913,
                    "Top": 0.34437501430511475,
                    "Left": 0.12396000325679779,
                    "Height": 0.4337500035762787
                },
                "FaceId": "57189455-42b0-4839-a86c-abda48b13174",
                "ExternalImageId": "image8.jpg",
                "Confidence": 100.0,
                "ImageId": "0aff2f37-e7a2-5dbc-a3a3-4ef6ec18eaa0"
            },
            "Similarity": 95.22837829589844
        }
    ]
}
```


詳細については、「Amazon Rekognition ディベロッパーガイド」の「Face ID で顔を検索する」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「SearchFaces」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.SearchFacesByImageRequest;
import software.amazon.awssdk.services.rekognition.model.Image;
import
 software.amazon.awssdk.services.rekognition.model.SearchFacesByImageResponse;
import software.amazon.awssdk.services.rekognition.model.FaceMatch;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SearchFaceMatchingImageCollection {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <collectionId> <sourceImage>

                Where:
```

```
                    collectionId - The id of the collection. \s
                    sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png).\s

                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        String sourceImage = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Searching for a face in a collections");
        searchFaceInCollection(rekClient, collectionId, sourceImage);
        rekClient.close();
    }

    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);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「SearchFaces」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンス の「SearchFaces」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **SearchFacesByImage**で を使用する

次のサンプルコードは、SearchFacesByImage を使用する方法を説明しています。

詳細については、「顔を検索する (イメージ)」を参照してください。

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    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.
    /// </summary>
    public class SearchFacesMatchingImage
    {
        public static async Task Main()
        {
            string collectionId = "MyCollection";
            string bucket = "amzn-s3-demo-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}");
            });
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「SearchFacesByImage」を参照してください。

CLI

AWS CLI

画像内の最大の顔と一致するコレクション内の顔を検索するには

次の search-faces-by-image コマンドは、指定された画像内の最大の顔と一致するコレクション内の顔を検索します。

```
aws rekognition search-faces-by-image \
    --image '{"S3Object":
{"Bucket":"MyImageS3Bucket","Name":"ExamplePerson.jpg"}}' \
    --collection-id MyFaceImageCollection

{
    "SearchedFaceBoundingBox": {
        "Width": 0.18562500178813934,
        "Top": 0.1618015021085739,
        "Left": 0.5575000047683716,
        "Height": 0.24770642817020416
    },
    "SearchedFaceConfidence": 99.993408203125,
    "FaceMatches": [
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.18562500178813934,
                    "Top": 0.1618019938468933,
                    "Left": 0.5575000047683716,
                    "Height": 0.24770599603652954
                },
```

```
            "FaceId": "ce7ed422-2132-4a11-ab14-06c5c410f29f",
            "ExternalImageId": "example-image.jpg",
            "Confidence": 99.99340057373047,
            "ImageId": "8d67061e-90d2-598f-9fbd-29c8497039c0"
        },
        "Similarity": 99.97913360595703
    },
    {
        "Face": {
            "BoundingBox": {
                "Width": 0.18562500178813934,
                "Top": 0.1618019938468933,
                "Left": 0.5575000047683716,
                "Height": 0.24770599603652954
            },
            "FaceId": "13692fe4-990a-4679-b14a-5ac23d135eab",
            "ExternalImageId": "image3.jpg",
            "Confidence": 99.99340057373047,
            "ImageId": "8df18239-9ad1-5acd-a46a-6581ff98f51b"
        },
        "Similarity": 99.97913360595703
    },
    {
        "Face": {
            "BoundingBox": {
                "Width": 0.41499999165534973,
                "Top": 0.09187500178813934,
                "Left": 0.28083300590515137,
                "Height": 0.3112500011920929
            },
            "FaceId": "8d3cfc70-4ba8-4b36-9644-90fba29c2dac",
            "ExternalImageId": "image2.jpg",
            "Confidence": 99.99769592285156,
            "ImageId": "a294da46-2cb1-5cc4-9045-61d7ca567662"
        },
        "Similarity": 99.18069458007812
    },
    {
        "Face": {
            "BoundingBox": {
                "Width": 0.48166701197624207,
                "Top": 0.20999999344348907,
                "Left": 0.21250000596046448,
                "Height": 0.36125001311302185
```

```
            },
            "FaceId": "bd4ceb4d-9acc-4ab7-8ef8-1c2d2ba0a66a",
            "ExternalImageId": "image1.jpg",
            "Confidence": 99.99949645996094,
            "ImageId": "5e1a7588-e5a0-5ee3-bd00-c642518dfe3a"
        },
        "Similarity": 98.66607666015625
    },
    {
        "Face": {
            "BoundingBox": {
                "Width": 0.5349419713020325,
                "Top": 0.29124999046325684,
                "Left": 0.16389399766921997,
                "Height": 0.40187498927116394
            },
            "FaceId": "745f7509-b1fa-44e0-8b95-367b1359638a",
            "ExternalImageId": "image9.jpg",
            "Confidence": 99.99979400634766,
            "ImageId": "67a34327-48d1-5179-b042-01e52ccfeada"
        },
        "Similarity": 98.24278259277344
    },
    {
        "Face": {
            "BoundingBox": {
                "Width": 0.5307819843292236,
                "Top": 0.2862499952316284,
                "Left": 0.1564060002565384,
                "Height": 0.3987500071525574
            },
            "FaceId": "2eb5f3fd-e2a9-4b1c-a89f-afa0a518fe06",
            "ExternalImageId": "image10.jpg",
            "Confidence": 99.99970245361328,
            "ImageId": "3c314792-197d-528d-bbb6-798ed012c150"
        },
        "Similarity": 98.10665893554688
    },
    {
        "Face": {
            "BoundingBox": {
                "Width": 0.5074880123138428,
                "Top": 0.3774999976158142,
                "Left": 0.18302799761295319,
```

```
                    "Height": 0.3812499940395355
                },
                "FaceId": "086261e8-6deb-4bc0-ac73-ab22323cc38d",
                "ExternalImageId": "image6.jpg",
                "Confidence": 99.99930572509766,
                "ImageId": "ae1593b0-a8f6-5e24-a306-abf529e276fa"
            },
            "Similarity": 98.10526275634766
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5574039816856384,
                    "Top": 0.37187498807907104,
                    "Left": 0.14559100568294525,
                    "Height": 0.4181250035762787
                },
                "FaceId": "11c4bd3c-19c5-4eb8-aecc-24feb93a26e1",
                "ExternalImageId": "image5.jpg",
                "Confidence": 99.99960327148438,
                "ImageId": "80739b4d-883f-5b78-97cf-5124038e26b9"
            },
            "Similarity": 97.94659423828125
        },
        {
            "Face": {
                "BoundingBox": {
                    "Width": 0.5773710012435913,
                    "Top": 0.34437501430511475,
                    "Left": 0.12396000325679779,
                    "Height": 0.4337500035762787
                },
                "FaceId": "57189455-42b0-4839-a86c-abda48b13174",
                "ExternalImageId": "image8.jpg",
                "Confidence": 100.0,
                "ImageId": "0aff2f37-e7a2-5dbc-a3a3-4ef6ec18eaa0"
            },
            "Similarity": 97.93476867675781
        }
    ],

 "FaceModelVersion": "3.0"
}
```

詳細については、「Amazon Rekognition 開発者ガイド」の「画像付きの顔を検索する」を参照してください。

- API の詳細については、AWS CLI コマンドリファレンスの「SearchFacesByImage」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.SearchFacesRequest;
import software.amazon.awssdk.services.rekognition.model.SearchFacesResponse;
import software.amazon.awssdk.services.rekognition.model.FaceMatch;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class SearchFaceMatchingIdCollection {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <collectionId> <sourceImage>

                Where:
                   collectionId - The id of the collection. \s
```

```
                    sourceImage - The path to the image (for example, C:\\AWS\
\pic1.png).\s
                """";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String collectionId = args[0];
        String faceId = args[1];
        Region region = Region.US_WEST_2;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        System.out.println("Searching for a face in a collections");
        searchFacebyId(rekClient, collectionId, faceId);
        rekClient.close();
    }

    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);
```

```
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「SearchFacesByImage」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンス の「SearchFacesByImage」を参照してください。

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用した Amazon Rekognition のシナリオ AWS SDKs

次のコード例は、 AWS SDKs を使用して Amazon Rekognition で一般的なシナリオを実装する方法を示しています。これらのシナリオでは、Amazon Rekognition 内で複数の関数を呼び出したり、他の AWS のサービスと組み合わせたりすることで、特定のタスクを実行する方法を示しています。各シナリオには、完全なソースコードへのリンクが含まれており、そこからコードの設定方法と実行方法に関する手順を確認できます。

シナリオは、サービスアクションをコンテキストで理解するのに役立つ中級レベルの経験を対象としています。

例

- AWS SDK を使用して Amazon Rekognition コレクションを構築し、その中に顔を見つける
- ユーザーがラベルを使用して写真を管理できる写真アセット管理アプリケーションの作成

- AWS SDK を使用して Amazon Rekognition でイメージ内の PPE を検出する
- AWS SDK を使用して Amazon Rekognition でイメージ内の要素を検出して表示する
- AWS SDK を使用してイメージ内の顔を検出する
- Amazon Rekognition と AWS SDK を使用してビデオ内の情報を検出する
- AWS SDK を使用して Amazon Rekognition でイメージ内のオブジェクトを検出する
- AWS SDK を使用して Amazon Rekognition でビデオ内の人物とオブジェクトを検出する
- AWS SDK を使用して EXIF およびその他のイメージ情報を保存する

AWS SDK を使用して Amazon Rekognition コレクションを構築し、その中に顔を見つける

次のコードサンプルは、以下の操作方法を示しています。

- Amazon Rekognition コレクションを作成します。
- このコレクションにイメージを追加し、その中から顔を検出します。
- 参照イメージに一致する顔をコレクション内で検索します。
- コレクションを削除します。

詳細については、コレクション内の顔を検索するには を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon Rekognition 関数をラップするクラスを作成します。

```
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
```

ラッパークラスを使用して、一連のイメージから顔のコレクションを作成し、コレクション内の顔を検索します。

```
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)
```

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

ユーザーがラベルを使用して写真を管理できる写真アセット管理アプリケーションの作成

以下のコード例は、ユーザーがラベルを使用して写真を管理できるサーバーレスアプリケーションの作成方法を示しています。

.NET

SDK for .NET

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

SDK for C++

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

SDK for Java 2.x

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition

- Amazon S3
- Amazon SNS

Kotlin

SDK for Kotlin

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

SDK for PHP

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway

- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

SDK for Rust

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、GitHub で完全な例を参照してください。

この例のソースについて詳しくは、AWS コミュニティでブログ投稿を参照してください。

この例で使用されているサービス

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用して Amazon Rekognition でイメージ内の PPE を検出する

次のコード例は、Amazon Rekognition を使用してイメージ内の個人用防護具 (PPE) を検出するアプリケーションを構築する方法を示しています。

Java

SDK for Java 2.x

個人用保護具を使用してイメージを検出する AWS Lambda 関数を作成する方法を示します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用して Amazon Rekognition でイメージ内の要素を検出して表示する

次のコードサンプルは、以下の操作方法を示しています。

- Amazon Rekognition を使用してイメージから要素を検出します。
- イメージを表示し、検出された要素の周囲に境界ボックスを描画します。

詳細については、境界ボックスの表示 を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon Rekognition 関数をラップするクラスを作成します。

```
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
```

境界ボックスとポリゴンを描画するヘルパー関数を作成します。

```
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()
```

Amazon Rekognition から返されたオブジェクトを解析するクラスを作成します。

```
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
```

ラッパークラスを使用して、イメージ内の要素を検出し、その境界ボックスを表示します。この例で使用されているイメージは GitHub で、指示やコードとともに確認できます。

```
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)
```

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用してイメージ内の顔を検出する

次のコードサンプルは、以下の操作方法を示しています。

- イメージを Amazon S3 バケットに保存します。
- Amazon Rekognition を使用して、年齢層、性別、感情 (笑顔など) などの顔の詳細を検出します。
- これらの詳細を表示します。

Rust

SDK for Rust

アップロード プレフィックスを付け、Amazon S3 バケット内でイメージを保存し、Amazon Rekognition を使用して、年齢層、性別、感情 (笑顔など) などの顔の詳細を検出し、それらの詳細を表示します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Rekognition と AWS SDK を使用してビデオ内の情報を検出する

次のコード例は、以下を実行する方法を示しています。

- Amazon Rekognition のジョブを開始し、人物、オブジェクト、テキストなどの要素を動画から検出します。
- ジョブが完了するまでジョブのステータスを確認します。
- 検出された要素のリストをジョブごとに出力します。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon S3 バケット内のビデオから有名人の結果を取得します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartCelebrityRecognitionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.CelebrityRecognitionSortBy;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.CelebrityRecognition;
import software.amazon.awssdk.services.rekognition.model.CelebrityDetail;
import
 software.amazon.awssdk.services.rekognition.model.StartCelebrityRecognitionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityRecognitionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetCelebrityRecognitionResponse;
import java.util.List;
```

```
/**
 * To run this code example, ensure that you perform the Prerequisites as stated
 * in the Amazon Rekognition Guide:
 * https://docs.aws.amazon.com/rekognition/latest/dg/video-analyzing-with-
sqs.html
 *
 * Also, ensure that set up your development environment, including your
 * credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class VideoCelebrityDetection {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];
        Region region = Region.US_EAST_1;
```

```
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startCelebrityDetection(rekClient, channel, bucket, video);
        getCelebrityDetectionResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    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);
        }
    }
}
```

ラベル検出オペレーションによって、ビデオ内のラベルを検出します。

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Video;
```

```
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.LabelDetectionSortBy;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.LabelDetection;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.Instance;
import software.amazon.awssdk.services.rekognition.model.Parent;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetect {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <queueUrl> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of the video (for example, people.mp4).\s
                   queueUrl- The URL of a SQS queue.\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
```

```
                    roleArn - The ARN of the AWS Identity and Access Management
(IAM) role to use.\s
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String queueUrl = args[2];
        String topicArn = args[3];
        String roleArn = args[4];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        SqsClient sqs = SqsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startLabels(rekClient, channel, bucket, video);
        getLabelJob(rekClient, sqs, queueUrl);
        System.out.println("This example is done!");
        sqs.close();
        rekClient.close();
    }

    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);
        }
    }
}
```

Amazon S3 バケットに保存されたビデオ内の顔を検出する

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.GetLabelDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.LabelDetectionSortBy;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.LabelDetection;
import software.amazon.awssdk.services.rekognition.model.Label;
import software.amazon.awssdk.services.rekognition.model.Instance;
import software.amazon.awssdk.services.rekognition.model.Parent;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import java.util.List;
```

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetect {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <queueUrl> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of the video (for example, people.mp4).\s
                   queueUrl- The URL of a SQS queue.\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String queueUrl = args[2];
        String topicArn = args[3];
        String roleArn = args[4];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();
```

```
        SqsClient sqs = SqsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startLabels(rekClient, channel, bucket, video);
        getLabelJob(rekClient, sqs, queueUrl);
        System.out.println("This example is done!");
        sqs.close();
        rekClient.close();
    }

    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);
        }
    }
}
```

Amazon S3 バケットに保存されたビデオ内の不適切なコンテンツや攻撃的なコンテンツを検出します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.S3Object;
```

```
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartContentModerationRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartContentModerationResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetContentModerationResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetContentModerationRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import
 software.amazon.awssdk.services.rekognition.model.ContentModerationDetection;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetectInappropriate {
    private static String startJobId = "";

    public static void main(String[] args) {

        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
 (Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
 (IAM) role to use.\s
                """;

        if (args.length != 4) {
```

```
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];
        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startModerationDetection(rekClient, channel, bucket, video);
        getModResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    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);
        }
    }
}
```

Amazon S3 バケットに保存されているビデオ内のテクニカルキューセグメントおよびショット検出セグメントを検出します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
```

```
import
 software.amazon.awssdk.services.rekognition.model.StartShotDetectionFilter;
import
 software.amazon.awssdk.services.rekognition.model.StartTechnicalCueDetectionFilter;
import
 software.amazon.awssdk.services.rekognition.model.StartSegmentDetectionFilters;
import
 software.amazon.awssdk.services.rekognition.model.StartSegmentDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartSegmentDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetSegmentDetectionResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetSegmentDetectionRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.SegmentDetection;
import software.amazon.awssdk.services.rekognition.model.TechnicalCueSegment;
import software.amazon.awssdk.services.rekognition.model.ShotSegment;
import software.amazon.awssdk.services.rekognition.model.SegmentType;
import software.amazon.awssdk.services.sqs.SqsClient;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetectSegment {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
```

```
                    topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic.\s
                    roleArn - The ARN of the AWS Identity and Access Management
(IAM) role to use.\s
                """;

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];

        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        SqsClient sqs = SqsClient.builder()
                .region(Region.US_EAST_1)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startSegmentDetection(rekClient, channel, bucket, video);
        getSegmentResults(rekClient);
        System.out.println("This example is done!");
        sqs.close();
        rekClient.close();
    }

    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);
        }
    }
}
```

Amazon S3 バケットに保存されたビデオに保存されたビデオ内のテキストを検出します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionRequest;
import
 software.amazon.awssdk.services.rekognition.model.StartTextDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetTextDetectionResponse;
import software.amazon.awssdk.services.rekognition.model.GetTextDetectionRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.TextDetectionResult;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoDetectText {
    private static String startJobId = "";

    public static void main(String[] args) {
        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>

                Where:
                   bucket - The name of the bucket in which the video is located
 (for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
```

```
                    topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic.\s
                    roleArn - The ARN of the AWS Identity and Access Management
(IAM) role to use.\s
                """";

        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        String bucket = args[0];
        String video = args[1];
        String topicArn = args[2];
        String roleArn = args[3];

        Region region = Region.US_EAST_1;
        RekognitionClient rekClient = RekognitionClient.builder()
                .region(region)
                .build();

        NotificationChannel channel = NotificationChannel.builder()
                .snsTopicArn(topicArn)
                .roleArn(roleArn)
                .build();

        startTextLabels(rekClient, channel, bucket, video);
        getTextResults(rekClient);
        System.out.println("This example is done!");
        rekClient.close();
    }

    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);
```

```
        }
    }
}
```

Amazon S3 バケットに保存されたビデオに保存されたビデオ内の人物を検出します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.NotificationChannel;
import
 software.amazon.awssdk.services.rekognition.model.StartPersonTrackingRequest;
import software.amazon.awssdk.services.rekognition.model.Video;
import
 software.amazon.awssdk.services.rekognition.model.StartPersonTrackingResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
 software.amazon.awssdk.services.rekognition.model.GetPersonTrackingResponse;
import
 software.amazon.awssdk.services.rekognition.model.GetPersonTrackingRequest;
import software.amazon.awssdk.services.rekognition.model.VideoMetadata;
import software.amazon.awssdk.services.rekognition.model.PersonDetection;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class VideoPersonDetection {
    private static String startJobId = "";

    public static void main(String[] args) {

        final String usage = """

                Usage:    <bucket> <video> <topicArn> <roleArn>
```

```
                Where:
                   bucket - The name of the bucket in which the video is located
(for example, (for example, myBucket).\s
                   video - The name of video (for example, people.mp4).\s
                   topicArn - The ARN of the Amazon Simple Notification Service
(Amazon SNS) topic.\s
                   roleArn - The ARN of the AWS Identity and Access Management
(IAM) role to use.\s
                """;

       if (args.length != 4) {
           System.out.println(usage);
           System.exit(1);
       }

       String bucket = args[0];
       String video = args[1];
       String topicArn = args[2];
       String roleArn = args[3];
       Region region = Region.US_EAST_1;
       RekognitionClient rekClient = RekognitionClient.builder()
               .region(region)
               .build();

       NotificationChannel channel = NotificationChannel.builder()
               .snsTopicArn(topicArn)
               .roleArn(roleArn)
               .build();

       startPersonLabels(rekClient, channel, bucket, video);
       getPersonDetectionResults(rekClient);
       System.out.println("This example is done!");
       rekClient.close();
   }

   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);
        }
```

```
    }
}
```

- API の詳細については、『AWS SDK for Java 2.x API リファレンス』の以下のトピックを参照してください。
  - GetCelebrityRecognition
  - GetContentModeration
  - GetLabelDetection
  - GetPersonTracking
  - GetSegmentDetection
  - GetTextDetection
  - StartCelebrityRecognition
  - StartContentModeration
  - StartLabelDetection
  - StartPersonTracking
  - StartSegmentDetection
  - StartTextDetection

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon S3 バケットに保存されたビデオ内の顔を検出する

```
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.fromEnvironment { 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.fromEnvironment { 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}")
        }
    }
}
```

Amazon S3 バケットに保存されたビデオ内の不適切なコンテンツや攻撃的なコンテンツを検出します。

```
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.fromEnvironment { 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)
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の以下のトピックを参照してください。
  - GetCelebrityRecognition
  - GetContentModeration
  - GetLabelDetection
  - GetPersonTracking
  - GetSegmentDetection
  - GetTextDetection
  - StartCelebrityRecognition
  - StartContentModeration
  - StartLabelDetection
  - StartPersonTracking
  - StartSegmentDetection
  - StartTextDetection

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用して Amazon Rekognition でイメージ内のオブジェクトを検出する

次のコード例は、Amazon Rekognition を使用して画像内からカテゴリ別にオブジェクトを検出するアプリケーションを構築する方法を示しています。

.NET

SDK for .NET

Amazon Simple Storage Service (Amazon S3) バケットにあるイメージ内から、Amazon Rekognition を使用してカテゴリ別にオブジェクトを識別するアプリケーションを、Amazon Rekognition .NET API を使用して作成する方法を示します。アプリケーションは、Amazon Simple Email Service (Amazon SES) を使用して、結果を記載した E メール通知を管理者に送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES

Java

SDK for Java 2.x

Amazon Rekognition Java API を使用して、Amazon Rekognition を使用し、 Amazon Simple Storage Service (Amazon S3) バケットにあるイメージ内でカテゴリ別にオブジェクトを識別するアプリケーションを作成する方法について説明します。アプリケーションは Amazon Simple Email Service (Amazon SES) を使用して、結果を含む E メール通知を管理者に送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES

JavaScript

SDK for JavaScript (v3)

で Amazon Rekognition を使用して、Amazon Rekognition を使用して Amazon Simple Storage Service (Amazon S3) バケットにあるイメージ内のオブジェクトをカテゴリ別に識別するアプリケーション AWS SDK for JavaScript を作成する方法を示します。アプリケーションは Amazon Simple Email Service (Amazon SES) を使用して、結果を含む E メール通知を管理者に送信します。

以下ではその方法を説明しています。

- Amazon Cognito を使用して認証されていないユーザーを作成します。
- Amazon Rekognition を使用して、オブジェクトのイメージを分析します。
- Amazon SES の E メールアドレスを検証します。
- Amazon SES を使用して、E メール通知を送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES

Kotlin

SDK for Kotlin

Amazon Simple Storage Service (Amazon S3) バケットにある画像内からカテゴリ別にオブジェクトを Amazon Rekognition を使用して識別するアプリケーションを、Amazon Rekognition Kotlin API を使用して作成する方法を示します。アプリケーションは、Amazon Simple Email Service (Amazon SES) を使用して、結果を記載した E メール通知を管理者に送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES

Python

SDK for Python (Boto3)

を使用して AWS SDK for Python (Boto3) 、以下を可能にするウェブアプリケーションを作成する方法を示します。

- Amazon Simple Storage Service (Amazon S3) バケットに、写真をアップロードします。
- Amazon Rekognition を使用して、写真を分析およびラベル付けします。
- Amazon Simple Email Service (Amazon SES) を使用して、イメージ分析の E メールレポートを送信します。

この例には、React で構築された JavaScript で記述されたウェブページと、Flask-RESTful で構築された Python で記述された REST サービスの 2 つの主要なコンポーネントが含まれています。

React ウェブページを使用すると、次のことができます。

- S3 バケットに保存されているイメージのリストを表示します。
- イメージを S3 バケットにアップロードします。
- イメージ内で検出された項目を識別するイメージとラベルを表示します。
- S3 バケット内のすべてのイメージのレポートを取得し、レポートの E メールを送信します。

ウェブページが REST サービスを呼び出します。サービスはリクエストを AWS に送信して、以下のアクションを実行します。

- S3 バケット内のイメージのリストを取得し、フィルタリングします。
- Amazon S3 バケットに写真をアップロードします。
- Amazon Rekognition を使用して個々の写真を分析し、写真で検出された項目を識別するラベルのリストを取得します。
- S3 バケット内のすべての写真を分析し、Amazon SES を使用してレポートを E メールで送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用して Amazon Rekognition でビデオ内の人物とオブジェクトを検出する

次のコード例は、Amazon Rekognition で動画内の人やオブジェクトを検出する方法を示します。

Java

SDK for Java 2.x

Amazon Rekognition Java API を使用し、Amazon Simple Storage Service (Amazon S3) バケットにあるビデオ内で顔やオブジェクトを検出するアプリケーションを作成する方法について説明します。アプリケーションは Amazon Simple Email Service (Amazon SES) を使用して、結果を含む E メール通知を管理者に送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

Python

SDK for Python (Boto3)

Amazon Rekognition を使用して、非同期検出ジョブを開始して、動画内の顔、オブジェクト、人を検出します。この例では、ジョブが完了し、Amazon Simple Queue Service (Amazon SQS) キューをトピックにサブスクライブしたときに、Amazon Simple Notification Service (Amazon SNS) トピックに通知するように Amazon Rekognition を設定します。キューがジョブに関するメッセージを受信すると、ジョブが取得され、結果が出力されます。

この例は GitHub で最もよく見られます。完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用して EXIF およびその他のイメージ情報を保存する

次のコードサンプルは、以下の操作方法を示しています。

- JPG、JPEG、または PNG ファイルから EXIF 情報を取得します。
- Amazon S3 バケットにイメージファイルをアップロードします。
- Amazon Rekognition を使用して、ファイル内の 3 つの上位属性 (ラベル) を特定します。
- EXIF およびラベル情報を、リージョンの Amazon DynamoDB テーブルに追加します。

Rust

SDK for Rust

JPG、JPEG、または PNG ファイルから EXIF 情報を取得し、イメージファイルを Amazon S3 バケットにアップロードし、Amazon Rekognition を使用してファイル内の 3 つの上位属性 (Amazon Rekognition のラベル) を特定し、リージョンの Amazon DynamoDB テーブルに EXIF およびラベル情報を追加します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Amazon Rekognition
- Amazon S3

AWS SDK 開発者ガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Rekognition の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

API リファレンス

Amazon Rekognition API リファレンスは、「Amazon Rekognition API Reference」にあります。

Amazon Rekognition セキュリティ

でのクラウドセキュリティが最優先事項 AWS です。 AWS のお客様は、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

以下のトピックでは、Amazon Rekognition のリソースをセキュリティで保護する方法について説明します。

トピック

- Amazon Rekognition の Identity and Access Management
- Amazon Rekognition でのデータ保護
- Amazon VPC エンドポイントで Amazon Rekognition を使用する
- Amazon Rekognition のコンプライアンス検証
- Amazon Rekognition の耐障害性
- Amazon Rekognition での設定と脆弱性の分析
- サービス間の混乱した代理の防止
- Amazon Rekognition のインフラストラクチャセキュリティ

Amazon Rekognition の Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を 認証 (サインイン) し、誰に Amazon Rekognition リソースの使用を承認する (アクセス許可を付与する) かを制御します。IAM は、追加料金なしで AWS のサービス 使用できる です。

トピック

- 対象者
- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理
- Amazon Rekognition が IAM と機能する仕組み
- AWS Amazon Rekognition の マネージドポリシー
- ポリシーに関するベストプラクティス

- Amazon Rekognition コンソールの使用
- Amazon Rekognition のアイデンティティベースのポリシーの使用例
- Amazon Rekognition のリソースベースのポリシーの例
- Amazon Rekognition アイデンティティとアクセスのトラブルシューティング

対象者

AWS Identity and Access Management (IAM) の使用方法は、Amazon Rekognition で行う作業によって異なります。

サービスユーザー – 業務を行うために Amazon Rekognition サービスを使用する場合は、管理者から必要な認証情報と許可が提供されます。さらに多くの Amazon Rekognition 機能を使用して作業を行う場合は、追加のアクセス許可が必要になることがあります。アクセスの管理方法を理解すると、管理者に適切なアクセス許可をリクエストするのに役に立ちます。Amazon Rekognition の機能にアクセスできない場合は、「Amazon Rekognition アイデンティティとアクセスのトラブルシューティング」を参照してください。

サービス管理者 – 社内の Amazon Rekognition リソースを担当している場合は、Amazon Rekognition に対する完全なアクセス権があると思われます。サービスのユーザーがどの Amazon Rekognition 機能やリソースにアクセスすべきかを決定するのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。会社で Amazon Rekognition と IAM を併用する方法の詳細については、「Amazon Rekognition が IAM と機能する仕組み」を参照してください。

IAM 管理者 – IAM 管理者には、Amazon Rekognition へのアクセスを管理するポリシーの作成方法の詳細を理解することが推奨されます。IAM で使用できる Amazon Rekognition のアイデンティティベースポリシーの例を確認するには、「Amazon Rekognition のアイデンティティベースのポリシーの使用例」を参照してください。

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けることで、認証 (にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center （ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook 認証情報は、フェデレーティッド ID の例です。フェデレーティッド

ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して にアクセスすると、間接的 AWS にロールを引き受けます。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 AWS サインイン ユーザーガイド」の「 にサインインする方法 AWS アカウント」を参照してください。

AWS プログラムで にアクセスする場合、 は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストに暗号で署名します。 AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。リクエストに自分で署名する推奨方法の使用については、「IAM ユーザーガイド」の「API リクエストに対するAWS Signature Version 4」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、 AWS では、多要素認証 (MFA) を使用してアカウントのセキュリティを強化することをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証」および「IAM ユーザーガイド」の「IAM のAWS 多要素認証」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、単一のユーザーまたはアプリケーションに特定のアクセス許可 AWS アカウントを持つ 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、「IAM ユーザーガイド」の「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザーに関するユースケース」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。で IAM ロールを一時的に引き受けるには AWS Management Console、ユーザーから IAM ロール (コンソール) に切り替えることができます。ロールを引き受けるには、 または AWS API オペレーションを AWS CLI 呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションのロールについては、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールを作成する」を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center User Guide」の「Permission sets」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、 (ロールをプロキシとして使用する代わりに) リソースに直接ポリシーをアタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。
- クロスサービスアクセス — 一部の では、他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用してアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可と AWS のサービス、ダウンストリームサービス AWS のサー

ビス へのリクエストのリクエストリクエストを組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

- サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除することができます。詳細については、「IAM ユーザーガイド」の「AWS のサービスに許可を委任するロールを作成する」を参照してください。
- サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、 サービスによって所有されます。IAM 管理者は、サービスリンクロールのアクセス許可を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを実行しているアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与する」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは のオブジェクト AWS であり、アイデンティティまたはリソースに関連付けられると、そのアクセス許可を定義します。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うときに、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、IAM ユーザーガイドの JSON ポリシー概要を参照してください。

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

アイデンティティベースのポリシーを使用する

アイデンティティベースポリシーは、IAM ユーザーグループ、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースポリシーの作成方法については、「IAM ユーザーガイド」の「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」を参照してください。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、IAM ユーザーガイドの「管理ポリシーとインラインポリシーのいずれかを選択する」を参照してください。

リソースベースのポリシーを利用する

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、 AWS WAF、および Amazon VPC は、ACLs。ACL の詳細については、「Amazon Simple Storage Service デベロッパーガイド」の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS は、一般的ではない追加のポリシータイプをサポートしています。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可の境界」を参照してください。
- サービスコントロールポリシー (SCPs) – SCPsは、 の組織または組織単位 (OU) の最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、ビジネスが所有する複数の AWS アカウント をグループ化して一元管理するためのサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー (SCP)」を参照してください。
- リソースコントロールポリシー (RCP) – RCP は、所有する各リソースにアタッチされた IAM ポリシーを更新することなく、アカウント内のリソースに利用可能な最大数のアクセス許可を設定するために使用できる JSON ポリシーです。RCP は、メンバーアカウントのリソースのアクセス許可を制限し、組織に属しているかどうかにかかわらず AWS アカウントのルートユーザー、 を含む ID の有効なアクセス許可に影響を与える可能性があります。RCP をサポートする のリストを含む Organizations と RCP の詳細については、AWS Organizations 「 ユーザーガイド AWS のサービス 」の「リソースコントロールポリシー (RCPs」を参照してください。 RCPs

- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関係する場合に がリクエストを許可するかどうか AWS を決定する方法については、「IAM ユーザーガイド」の「ポリシー評価ロジック」を参照してください。

Amazon Rekognition が IAM と機能する仕組み

IAM を使用して Amazon Rekognition へのアクセスを管理する前に、Amazon Rekognition で使用できる IAM 機能について理解しておく必要があります。Amazon Rekognition およびその他の AWS のサービスが IAM と連携する方法の概要を把握するには、「IAM ユーザーガイド」のAWS 「IAM と連携する のサービス」を参照してください。

トピック

- Amazon Rekognition のアイデンティティベースのポリシー
- Amazon Rekognition リソースベースのポリシー
- Amazon Rekognition の IAM ロール

Amazon Rekognition のアイデンティティベースのポリシー

IAM アイデンティティベースのポリシーでは許可または拒否するアクションとリソース、またアクションを許可または拒否する条件を指定できます。Amazon Rekognition は、特定のアクション、リソース、および条件キーをサポートしています。JSON ポリシーで使用するすべての要素については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素のリファレンス」を参照してください。

アクション

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの Action 要素にはポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連する AWS API オペレーションと同じです。一致する API オペレーションのない許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは依存アクションと呼ばれます。

このアクションは関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

Amazon Rekognition のポリシーアクションは、アクションの前にプレフィックス rekognition: を使用します。たとえば、DetectLabels Amazon Rekognition API オペレーションでイメージ内のオブジェクト、シーン、または概念を検出するアクセス許可を付与するには、ポリシーに rekognition:DetectLabels アクションを含めます。ポリシーステートメントにはAction または NotAction 要素を含める必要があります。Amazon Rekognition は、このサービスで実行できるタスクを記述する独自のアクションのセットを定義します。

単一のステートメントに複数のアクションを指定するには次のようにコンマで区切ります。

```
"Action": [
      "rekognition:action1",
      "rekognition:action2"
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、Describe という単語で始まるすべてのアクションを指定するには次のアクションを含めます。

```
"Action": "rekognition:Describe*"
```

Amazon Rekognition アクションのリストを確認するには、IAM ユーザーガイド の「Amazon Rekognition で定義されるアクション」を参照してください。

リソース

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルが、どのリソースに対してどのような条件下でアクションを実行できるかということです。

Resource JSON ポリシー要素はアクションが適用されるオブジェクトを指定します。ステートメントにはResource または NotResource 要素を含める必要があります。ベストプラクティスとし

て、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

ARN の形式の詳細については、「Amazon リソースネーム (ARNs) と AWS サービス名前空間」を参照してください。

たとえば、ステートメントで MyCollection コレクションを指定するには、次の ARN を使用します。

```
"Resource": "arn:aws:rekognition:us-east-1:123456789012:collection/MyCollection"
```

特定のアカウントに属するすべてのインスタンスを指定するには、ワイルドカード (*) を使用します。

```
"Resource": "arn:aws:rekognition:us-east-1:123456789012:collection/*"
```

リソースの作成を含む、一部の Amazon Rekognition アクションは、特定のリソースで実行できません。このような場合はワイルドカード *を使用する必要があります。

```
"Resource": "*"
```

Amazon Rekognition のリソースタイプとそれらの ARN のリストを確認するには、IAM ユーザーガイドの「Amazon Rekognition で定義されるリソースタイプ」を参照してください。どのアクションで各リソースの ARN を指定できるかについては、「Amazon Rekognition で定義されるアクション」を参照してください。

条件キー

Amazon Rekognition にはサービス固有条件キーがありませんが、いくつかのグローバル条件キーの使用をサポートしています。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「グローバル条件コンテキストキー」を参照してください。

Amazon Rekognition リソースベースのポリシー

Amazon Rekognition は、Custom Labels モデルのコピーオペレーションを行うためのリソースベースポリシーをサポートしています。詳細については、「Amazon Rekognition resource-based policy examples」を参照してください。

Simple Storage Service (Amazon S3) などの他のサービスでは、リソースベースの許可ポリシーもサポートされています。例えば、ポリシーを S3 バケットにアタッチして、そのバケットに対するアクセス許可を管理できます。

Amazon S3 バケットに保存されているイメージにアクセスするには、S3 バケット内のオブジェクトにアクセスするための権限が必要です。このアクセス許可により、Amazon Rekognition は S3 バケットからイメージをダウンロードできます。次のポリシー例では、amzn-s3-demo-bucket3 という名前の S3 バケットに対して `s3:GetObject`アクションを実行することをユーザーに許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket3/*"
            ]
        }
    ]
}
```

バージョニングを有効にして S3 バケットを使用するには、次の例に示すように、`s3:GetObjectVersion` アクションを追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
```

```
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket3/*"
            ]
        }
    ]
```

Amazon Rekognition の IAM ロール

IAM ロールは、特定のアクセス許可を持つ AWS アカウント内のエンティティです。

Amazon Rekognition での一時的な認証情報の使用

一時的な認証情報を使用して、フェデレーションでサインインする、IAM 役割を引き受ける、またはクロスアカウント役割を引き受けることができます。一時的なセキュリティ認証情報を取得するには、AssumeRole や GetFederationToken などの AWS STS API オペレーションを呼び出します。

Amazon Rekognition は、一時的な認証情報の使用をサポートします。

サービスにリンクされた役割

サービスにリンクされたロールを使用すると、 AWS サービスは他の サービスのリソースにアクセスして、ユーザーに代わってアクションを実行できます。サービスリンクロールは IAM アカウント内に表示され、サービスによって所有されます。IAM 管理者は、サービスリンクロールの許可を表示できますが、編集することはできません。

Amazon Rekognition ではサービスにリンクされたロールがサポートされていません。

サービス役割

この機能により、ユーザーに代わってサービスがサービス役割を引き受けることが許可されます。この役割により、サービスがお客様に代わって他のサービスのリソースにアクセスし、アクションを完了することが許可されます。サービス役割はIAM アカウントに表示され、アカウントによって所有されます。つまり、IAM 管理者はこの役割の権限を変更できます。ただし、それにより、サービスの機能が損なわれる場合があります。

Amazon Rekognition ではサービスロールがサポートされています。

サービスロールを使用すると、Amazon Rekognition を使用して別のサービスを呼び出し、アクセスすべきではないリソースに対してアクションを実行するというセキュリティ上の問題が発生する可能性があります。アカウントを安全に保つには、Amazon Rekognition のアクセス範囲を、使用しているリソースのみに制限する必要があります。これは、IAM サービスロールに信頼ポリシーをアタッ

チすることで実行できます。これを行う方法については、「サービス間の混乱した代理の防止」を参照してください。

Amazon Rekognition での IAM ロールの選択

保存した動画を分析するために Amazon Rekognition を設定する場合は、Amazon Rekognition が自動的に Amazon SNS にアクセスすることを許可するロールを選択する必要があります。サービスロールあるいはサービスにリンクされたロールを以前に作成している場合、Amazon Rekognition は選択できるロールを一覧表示します。詳細については、「the section called “Amazon Rekognition Video の設定”」を参照してください。

例: Amazon S3 バケット内のイメージにアクセスするための Amazon Rekognition Amazon S3の設定

以下は、Amazon S3 バケット内のイメージを分析するために Amazon Rekognition を設定する方法の例です。 Amazon S3 Amazon Rekognition を使用して Amazon S3 バケット内のイメージを分析する場合は、以下を実行する必要があります。

1. IAM ユーザー/ロール (クライアント) に、関連する Amazon Rekognition API オペレーション (DetectLabels、DetectFaces など) を呼び出すアクセス許可があることを確認します。

   目的の API オペレーションを呼び出すための適切なアクセス許可を付与するアイデンティティベースのポリシーをアタッチします。たとえば、ロールに `DetectLabels`と を呼び出すアクセス許可を付与するには`DetectFaces`、ロールに次のようなポリシーをアタッチします。

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "rekognition:DetectLabels",
                   "rekognition:DetectFaces"
                   // other Rekognition permissions as needed
               ],
               "Resource": "*"
           }
       ]
   }
   ```

2. Amazon Rekognition サービスにはAmazon S3バケットにアクセスするためのアクセス許可が必要です。API コールを行うときに Amazon Rekognition に渡す必要がある IAM サービスロールを作

成します。サービスロール: Amazon Rekognition サービスプリンシパルを信頼し、バケットに対するs3:GetObjectアクセス許可があることを確認します。

信頼ポリシーは次のようになります。

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

サービスロールにアタッチされたアイデンティティベースのポリシーは次のようになります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

AWS Amazon Rekognition の マネージドポリシー

ユーザー、グループ、ロールにアクセス許可を追加するには、自分でポリシーを作成するよりも、AWS 管理ポリシーを使用する方が簡単です。チームに必要な権限のみを提供する IAM カスタマーマネージドポリシーを作成するには時間と専門知識が必要です。すぐに開始するには、 AWS マネー

ジドポリシーを使用できます。これらのポリシーは一般的なユースケースを対象としており、 AWS アカウントで利用できます。 AWS 管理ポリシーの詳細については、「IAM ユーザーガイド」の「AWS 管理ポリシー」を参照してください。

AWS サービスは、 AWS 管理ポリシーを維持および更新します。 AWS 管理ポリシーのアクセス許可は変更できません。サービスでは新しい機能を利用できるようにするために、 AWS マネージドポリシーに権限が追加されることがあります。この種類の更新はポリシーがアタッチされている、すべてのアイデンティティ (ユーザー、グループおよびロール) に影響を与えます。新しい機能が立ち上げられた場合や、新しいオペレーションが使用可能になった場合に、各サービスが AWS マネージドポリシーを更新する可能性が最も高くなります。サービスは AWS 管理ポリシーからアクセス許可を削除しないため、ポリシーの更新によって既存のアクセス許可が破損することはありません。

さらに、 は、複数のサービスにまたがる職務機能の管理ポリシー AWS をサポートします。例えば、ReadOnlyAccess AWS 管理ポリシーは、すべての AWS サービスとリソースへの読み取り専用アクセスを提供します。サービスが新機能を起動すると、 は新しいオペレーションとリソースに読み取り専用アクセス許可 AWS を追加します。ジョブ機能ポリシーのリストと説明については、IAM ユーザーガイドのジョブ機能のAWS 管理ポリシーを参照してください。

AWS 管理ポリシー:AmazonRekognitionFullAccess

`AmazonRekognitionFullAccess` Amazon Rekognition リソースに対して、コレクションの作成や削除なども含めて、フルアクセスを許可します。

`AmazonRekognitionFullAccess` ポリシーを IAM アイデンティティにアタッチできます。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

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

```
    ]
}
```

AWS 管理ポリシー: AmazonRekognitionReadOnlyAccess

AmazonRekognitionReadOnlyAccess Amazon Rekognition リソースへの読み取り専用アクセスを許可します。

AmazonRekognitionReadOnlyAccess ポリシーを IAM アイデンティティにアタッチできます。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonRekognitionReadOnlyAccess",
            "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",
                "rekognition:ListUsers",
                "rekognition:SearchUsers",
                "rekognition:SearchUsersByImage",
                "rekognition:GetMediaAnalysisJob",
                "rekognition:ListMediaAnalysisJobs"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 管理ポリシー:AmazonRekognitionServiceRole

AmazonRekognitionServiceRole Amazon Rekognition は、お客様に代わって Amazon Kinesis Data Streams および Amazon SNS サービスを呼び出すことを許可します。

AmazonRekognitionServiceRole ポリシーを IAM アイデンティティにアタッチできます。

このサービスロールを使用する場合は、Amazon Rekognition のアクセス範囲を、使用しているリソースのみに制限してアカウントを安全に保つ必要があります。これは、IAM サービスロールに信頼ポリシーをアタッチすることで実行できます。これを行う方法については、「サービス間の混乱した代理の防止」を参照してください。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

```
{
    "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 管理ポリシー:Amazon RekognitionCustomLabelsFullAccess

このポリシーは、Amazon Rekognition Custom Labels; users を対象としています。AmazonRekognitionCustomLabelsFullAccess ポリシーを使用して、ユーザーが Amazon Rekognition カスタムラベル API へのフルアクセスと、Amazon Rekognition カスタムラベルコンソールによって作成されたコンソールバケットへのフルアクセスを許可します。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

```
{
    "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": "*"
        }
    ]
}
```

AWS マネージドポリシーに対する Amazon Rekognition の更新

Amazon Rekognition の AWS マネージドポリシーの更新に関する詳細を、このサービスがこれらの変更の追跡を開始した以降の分について表示します。このページへの変更に関する自動アラートを受

け取るには、Amazon Rekognition の ドキュメント履歴 ページで RSS フィードにサブスクライブしてください。

変更	説明	日付
メディア分析ジョブを含むアクションが次の管理ポリシーに追加されました。 • AWS 管理ポリシー: AmazonRekognitionReadOnlyAccess	Amazon Rekognition が、次のアクションを AmazonRekognitionReadOnlyAccess 管理ポリシーに追加しました。 • GetMediaAnalysisJob • ListMediaAnalysisJob	2023 年 10 月 31 日
ユーザーを管理するアクションが以下の管理ポリシーに追加されました。 • AWS 管理ポリシー: AmazonRekognitionReadOnlyAccess	Amazon Rekognition が、次のアクションを AmazonRekognitionReadOnlyAccess 管理ポリシーに追加しました。 • ListUsers • SearchUsers • SearchUsersByImage	2023 年 6 月 12 日
ProjectPolicy と Custom Labels モデルコピーのアクションが以下の管理ポリシーに追加されました。 • AWS 管理ポリシー:AmazonRekognitionFullAccess • AWS 管理ポリシー:Amazon RekognitionCustomLabelsFullAccess	Amazon Rekognition が、次のアクションを AmazonRekognitionCustomLabelsFullAccess および AmazonRekognitionFullAccess 管理ポリシーに追加しました。 • CopyProjectVersion • PutProjectPolicy • ListProjectPolicies	2022 年 7 月 21 日

変更	説明	日付
	• DeleteProjectPolicy	
ProjectPolicy と Custom Labels モデルコピーのアクションが以下の管理ポリシーに追加されました。 • AWS 管理ポリシー: AmazonRekognitionReadOnlyAccess	Amazon Rekognition が、次のアクションを AmazonRekognitionReadOnlyAccess 管理ポリシーに追加しました。 • ListProjectPolicies	2022 年 7 月 21 日
次の管理ポリシーのデータセット管理が更新されます。 • AWS 管理ポリシー: AmazonRekognitionReadOnlyAccess • AWS 管理ポリシー:AmazonRekognitionFullAccess • AWS 管理ポリシー:Amazon RekognitionCustomLabelsFullAccess	Amazon Rekognition は、AmazonRekognitionReadOnlyAccess、AmazonrekognitionfullOnlyAccess、およびAmazonrekognitionCustomLabelsFullAccess 管理ポリシーに以下のアクションを追加しました。 • CreateDataset • ListDatasetEntries • ListDatasetLabels • DescribeDataset • UpdateDatasetEntries • DistributeDatasetEntries • DeleteDataset	2021 年 11 月 1 日

変更	説明	日付
更新のタグ付け AWS 管理ポリシー: AmazonRekognitionReadOnlyAccess および AWS 管理ポリシー:AmazonRekognitionFullAccess	Amazon Rekognition は、AmazonrekognitionFullAccess および AmazonrekognitionReadOnlyAccess ポリシーに新しいタグ付けアクションを追加しました。	2021 年 4 月 2 日
Amazon Rekognition が変更の追跡を開始	Amazon Rekognition が AWS マネージドポリシーの変更の追跡を開始しました。	2021 年 4 月 2 日

ポリシーに関するベストプラクティス

アイデンティティベースのポリシーは、アカウント内で誰かが Amazon Rekognition リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションを実行すると、 AWS アカウントに料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可を付与するには、多くの一般的なユースケースにアクセス許可を付与するAWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義して、アクセス許可をさらに減らすことをお勧めします。詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」または「ジョブ機能のAWS マネージドポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、「IAM ユーザーガイド」の「IAM でのポリシーとアクセス許可」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、サービスアクションが などの特定の を通じて使用される場合に AWS のサービス、サービスアクションへのアクセスを許

可することもできます AWS CloudFormation。詳細については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素:条件」を参照してください。

- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、「IAM ユーザーガイド」の「IAM Access Analyzer でポリシーを検証する」を参照してください。
- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、セキュリティを強化するために MFA を有効にします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、「IAM ユーザーガイド」の「MFA を使用した安全な API アクセス」を参照してください。

IAM でのベストプラクティスの詳細については、IAM ユーザーガイドの IAM でのセキュリティのベストプラクティスを参照してください。

Amazon Rekognition コンソールの使用

Amazon Rekognition カスタムラベル機能を除く、Amazon Rekognition は、Amazon Rekognition コンソールの使用時に追加のアクセス許可を必要としません。Amazon Rekognition カスタムラベルの詳細については、「ステップ 5: Amazon Rekognition カスタムラベルコンソールのアクセス許可を設定する」を参照してください。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

Amazon Rekognition のアイデンティティベースのポリシーの使用例

デフォルトでは、ユーザーおよびロールには Amazon Rekognition リソースを作成または修正するアクセス権限はありません。また、、 AWS Management Console AWS CLI、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、ユーザーとロールに必要な、指定されたリソースで特定の API オペレーションを実行する権限をユーザーとロールに付与する IAM ポリシーを作成する必要があります。続いて、管理者はそれらのアクセス許可が必要なユーザーまたはグループにそのポリシーをアタッチします。

JSON ポリシードキュメントのこれらの例を使用して、IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「JSON タブでのポリシーの作成」を参照してください。

トピック

- Amazon Rekognition カスタムラベル例
- 例 1: リソースへの読み取り専用アクセスをユーザーに許可する
- 例 2: リソースへのフルアクセスをユーザーに許可する
- 自分の権限の表示をユーザーに許可する

Amazon Rekognition カスタムラベル例

Amazon Rekognition Custom Labels のアイデンティティベースのポリシーを作成できます。詳細については、「セキュリティ」を参照してください。

例 1: リソースへの読み取り専用アクセスをユーザーに許可する

次の例では、Amazon Rekognition リソースへのフルアクセスを許可します。

```
{
    "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": "*"
        }
    ]
}
```

例 2: リソースへのフルアクセスをユーザーに許可する

次の例では、Amazon Rekognition リソースへのフルアクセスを許可します。

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

自分の権限の表示をユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、

または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

```
{
    "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 のリソースベースのポリシーの例

Amazon Rekognition Custom Labels は、プロジェクトポリシーと呼ばれるリソースベースポリシーを使用して、モデルバージョンのコピー権限を管理します。

プロジェクトポリシーは、ソースプロジェクトから目的のプロジェクトにモデルバージョンをコピーするアクセス許可を許可または拒否します。送信先プロジェクトが別の AWS アカウントにある場合

や、 AWS アカウント内のアクセスを制限する場合は、プロジェクトポリシーが必要です。例えば、特定の IAM ロールへのコピーアクセス許可を拒否できます。詳細については、「Copying a model」を参照してください。

モデルのバージョンをコピーする権限を付与する

次の例では、プリンシパル `arn:aws:iam::123456789012:role/Admin` がモデルバージョン `arn:aws:rekognition:us-east-1:123456789012:project/my_project/version/test_1/1627045542080` をコピーできます。

```
{
  "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"
    }
  ]
}
```

Amazon Rekognition アイデンティティとアクセスのトラブルシューティング

以下の情報を使用して、Amazon Rekognition と IAM の使用時に発生する可能性がある一般的な問題の診断と修正に役立てます。

トピック

- Amazon Rekognition でアクションを実行する権限がない
- iam:PassRole を実行する権限がない
- 管理者として Amazon Rekognition へのアクセスを他のユーザーに許可したい
- AWS アカウント外のユーザーに Amazon Rekognition リソースへのアクセスを許可したい

Amazon Rekognition でアクションを実行する権限がない

からアクションを実行する権限がないと AWS Management Console 通知された場合は、管理者に連絡してサポートを依頼する必要があります。管理者とは、サインイン認証情報を提供した担当者です。

以下の例のエラーは、mateojackson IAM ユーザーがコンソールを使用して [######] の詳細を表示する際に、rekognition:*GetWidget* 許可がない場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 rekognition:GetWidget on resource: my-example-widget
```

この場合、Mateo は、rekognition:*GetWidget* アクションを使用して *my-example-widget* リソースへのアクセスが許可されるように、管理者にポリシーの更新を依頼します。

iam:PassRole を実行する権限がない

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して Amazon Rekognition にロールを渡せるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、既存のロールをそのサービスに渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

以下の例のエラーは、marymajor という IAM ユーザーがコンソールを使用して Amazon Rekognition でアクションを実行しようする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。メアリーには、ロールをサービスに渡す許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

管理者として Amazon Rekognition へのアクセスを他のユーザーに許可したい

他のユーザーに Amazon Rekognition へのアクセスを許可するには、アクセスが必要なユーザーまたはアプリケーションにアクセス許可を付与する必要があります。 AWS IAM Identity Center を使用してユーザーとアプリケーションを管理する場合は、アクセスレベルを定義するアクセス許可セットをユーザーまたはグループに割り当てます。アクセス許可セットは、ユーザーまたはアプリケーションに関連付けられている IAM ロールに自動的に IAM ポリシーを作成して割り当てます。詳細については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セット」を参照してください。

IAM アイデンティティセンターを使用していない場合は、アクセスを必要としているユーザーまたはアプリケーションの IAM エンティティ (ユーザーまたはロール) を作成する必要があります。次に、 Amazon Rekognition の適切な許可を付与するエンティティにポリシーをアタッチする必要があります。アクセス許可が付与されたら、ユーザーまたはアプリケーション開発者に認証情報を提供します。これらの認証情報を使用して AWSにアクセスします。IAM ユーザー、グループ、ポリシー、アクセス許可の作成の詳細については、「IAM ユーザーガイド」の「IAM アイデンティティ」と「IAM のポリシーとアクセス許可」を参照してください。

AWS アカウント外のユーザーに Amazon Rekognition リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- Amazon Rekognition がこれらの機能をサポートしているかどうかを確認するには、「Amazon Rekognition が IAM と機能する仕組み」を参照してください。
- 所有 AWS アカウント する 全体のリソースへのアクセスを提供する方法については、「IAM ユーザーガイド」の「所有 AWS アカウント する別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、「IAM ユーザーガイド」の「サードパーティーが所有する へのアクセスを提供する AWS アカウント」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、「IAM ユーザーガイド」の「外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可」を参照してください。

- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。

Amazon Rekognition でのデータ保護

Amazon Rekognition でのデータ保護には、 AWS 責任共有モデルが適用されます。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。ユーザーは、このインフラストラクチャでホストされるコンテンツに対する管理を維持する責任があります。また、使用する「 AWS のサービス 」のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、データプライバシーに関するよくある質問を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された AWS 責任共有モデルおよび GDPR のブログ記事を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント 、 AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。CloudTrail 証跡を使用して AWS アクティビティをキャプチャする方法については、「 AWS CloudTrail ユーザーガイド」のCloudTrail 証跡の使用」を参照してください。
- AWS 暗号化ソリューションと、その中のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度な管理されたセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-3 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-3」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報を、タグ、または [名前] フィールドなどの自由形式のテキストフィールドに含めないことを強くお勧めします。これは、コンソール、API、AWS CLIまたは SDK を使用して Rekognition または他の AWS のサービス を使用する場合も同様

です。 AWS SDKs タグ、または名前に使用される自由記述のテキストフィールドに入力したデータは、請求または診断ログに使用される場合があります。外部サーバーに URL を提供する場合、そのサーバーへのリクエストを検証できるように、認証情報を URL に含めないことを強くお勧めします。

データ暗号化

次の情報は、ユーザーのデータを保護するために、 Amazon Rekognition がデータ暗号化を使用する場所について説明しています。

保管中の暗号化

Amazon Rekognition イメージ

イメージ

Amazon Rekognition API オペレーションに渡されたイメージは、AI サービスオプトアウトに関するポリシーのページを訪問し、そこで説明されたプロセスに従ってオプトアウトしない限り、サービスの改善のために保存されたり使用されたりすることがあります。保存されたイメージは、AWS Key Management Service (SSE-KMS) を使用して保管時 (Amazon S3) に暗号化されます。

コレクション

コレクションに情報を保存する顔比較オペレーションの場合、基になる検出アルゴリズムで、最初に入力イメージ内の顔を検出して、顔ごとのベクトルを抽出し、次にその顔ベクトルをコレクションに保存します。Amazon Rekognition は、顔の比較を実行するときに、これらの顔ベクトルを使用します。顔ベクトルは浮動小数点数の配列として保存され、保管中は暗号化されます。

Amazon Rekognition Video

動画

動画を分析するために、Amazon Rekognition は動画を処理用のサービスにコピーします。動画は、AI サービスオプトアウトに関するポリシーのページ を訪問し、そこで説明されたプロセスに従ってオプトアウトしない限り、サービスの改善のために保存されたり使用されたりすることがあります。動画は、AWS Key Management Service (SSE-KMS) を使用して保管時 (Amazon S3) に暗号化されます。

Amazon Rekognition Custom Labels

Amazon Rekognition カスタムラベルでは、保管中のデータが暗号化されます。

イメージ

モデルをトレーニングするために、Amazon Rekognition Custom Labels は出典トレーニング画像とテストイメージのコピーを作成します。コピーされたイメージは、 AWS KMS key 指定した または AWS 所有の KMS キーによるサーバー側の暗号化を使用して、Amazon Simple Storage Service (S3) で保管時に暗号化されます。Amazon Rekognition カスタムラベルでは、対称 KMS キーのみがサポートされます。出典のイメージには影響がありません。詳細については、「Amazon Rekognition カスタムラベルモデルのトレーニング」を参照してください。

モデル

デフォルトでは、Amazon Rekognition カスタムラベルは、 AWS 所有のキーでサーバー側の暗号化を使用し、Amazon S3 バケットに保存されているトレーニング済みのモデルおよびマニフェストファイルを暗号化します。詳細については、「サーバー側暗号化を使用するデータの保護」を参照してください。トレーニング結果は CreateProjectVersion への `OutputConfig` 入力パラメータで指定されたバケットに書き込まれます。トレーニング結果は、バケット (`OutputConfig`) に対して構成された暗号化設定を使用して暗号化されます。

コンソールのバケット

Amazon Rekognition カスタムラベルのコンソールは、プロジェクトを管理するために使用できる Amazon S3 バケット (コンソールのバケット) を作成します。コンソールのバケットは、デフォルトの Amazon S3 暗号化を使用して暗号化されます。詳細については、「S3 バケット用 Amazon Simple Storage Service デフォルト暗号化」を参照してください。独自の KMS キーを使用している場合は、作成後にコンソールバケットを設定します。詳細については、「サーバー側暗号化を使用するデータの保護」を参照してください。Amazon Rekognition のカスタムラベルは、コンソールのバケットへのパブリックアクセスをブロックします。

Rekognition Face Liveness

Rekognition Face Liveness サービスのアカウントに保存されているすべてのセッション関連データは、保管時に完全に暗号化されます。デフォルトでは、参照イメージと監査イメージは、サービスアカウントの AWS 所有キーを使用して暗号化されます。ただし、これらのイメージを暗号化するために独自の AWS KMS キーを指定することもできます。

転送中の暗号化

Amazon Rekognition API エンドポイントでは、HTTPS 経由の安全な接続のみがサポートされます。すべての通信は、Transport Layer Security (TLS) で暗号化されます。

キー管理

AWS Key Management Service (KMS) を使用して、Amazon S3 バケットに保存する入力イメージおよび動画のキーを管理できます。詳細については、「AWS Key Management Service の概念」を参照してください。

Face Liveness のカスタマーマネージドキーによる暗号化

CreateFaceLivenessSession API は、オプションの `KmsKeyId` パラメータを受け入れます。ユーザーは、自分のアカウントで作成した KMS キーの `id` を指定できます。このキーは StartFaceLivenessSession API の実行中に取得した参照イメージと監査イメージを暗号化するために使用され、イメージは GetFaceLivenessSessionResults API の実行中にこのキーを使用して復号され、その後、結果が返されます。CreateFaceLivenessSession リクエストに OutputConfig が含まれている場合、参照イメージと監査イメージは指定された Amazon S3 パスにアップロードされます。Amazon S3 バケットでサーバー側の暗号化 (SSE-S3) を有効にし、データを保管時も暗号化された状態にすることが推奨されます。

独自の AWS KMS キー ID を指定すると、Rekognition Face Liveness サービスは APIs を呼び出すプリンシパルに代わってカスタマーマネージドキーを使用するアクセス許可を取得します。顧客のバックエンド (API `CreateFaceLivenessSession` と `GetFaceLivenessSessionResults`) から API を呼び出すために使用されるプリンシパル (ユーザーまたはロール) には、以下を実行するためのアクセス権が必要です。

- kms:DescribeKey
- kms:GenerateDataKey
- kms:Decrypt

インターネットトラフィックのプライバシー

Amazon Rekognition の Amazon Virtual Private Cloud (Amazon VPC) エンドポイントは、Amazon Rekognition への接続のみを許可する VPC 内の論理エンティティです。Amazon VPC ルートは、Amazon Rekognition にリクエストし、VPC にレスポンスをルーティングします。詳細については、Amazon VPC ユーザーガイド の VPC エンドポイント を参照してください。Amazon Rekognition で Amazon VPC エンドポイントを使用する方法については、Amazon VPC エンドポイントで Amazon Rekognition を使用する を参照してください。

Amazon VPC エンドポイントで Amazon Rekognition を使用する

Amazon Virtual Private Cloud (Amazon VPC) を使用して AWS リソースをホストする場合、VPC と Amazon Rekognition の間のプライベート接続を確立できます。この接続を使用すると、Amazon Rekognition はパブリックインターネットを経由せずに、VPC のリソースと通信できます。

Amazon VPC は、お客様の定義する仮想ネットワークで AWS リソースを起動するために使用できる AWS のサービスです。VPC を使用することで、IP アドレス範囲、サブネット、ルートテーブル、ネットワークゲートウェイなどのネットワーク設定を制御できます。VPC エンドポイントを使用して、AWS ネットワークは VPC と AWS のサービスとの間のルーティングを処理します。

VPC を Amazon Rekognition に接続するには、Amazon Rekognition 用のインターフェイス VPC エンドポイントを定義します。インターフェイスエンドポイントは、サポートされている AWS サービス宛てのトラフィックのエントリポイントとして機能するプライベート IP アドレスを持つ Elastic Network Interface です。このエンドポイントは、インターネットゲートウェイ、ネットワークアドレス変換 (NAT) インスタンス、または VPN 接続を必要とせず、信頼性が高くスケーラブルな Amazon Rekognition への接続を提供します。詳細については、「Amazon VPC ユーザーガイド」の「Amazon VPC とは」を参照してください。

インターフェイス VPC エンドポイントは AWS PrivateLink によって有効になります。この AWS テクノロジーにより、Elastic Network Interface とプライベート IP アドレスを使用して、AWS のサービス間でプライベート通信が可能になります。

> **Note**
>
> すべての Amazon Rekognition 連邦情報処理規格 (FIPS) エンドポイントは AWS PrivateLink によってサポートされています。

Amazon Rekognition 用の Amazon VPC エンドポイントの作成

Amazon Rekognition で使用する 2 種類の Amazon VPC エンドポイントを作成できます。

- Amazon Rekognition オペレーションで使用する VPC エンドポイント。ほとんどのユーザーにとって、これは VPC エンドポイントの最適なタイプです。
- 連邦情報処理標準 (FIPS) 刊行物 140-2 米国政府標準に準拠するエンドポイントを使用した、Amazon Rekognition オペレーションの VPC エンドポイント。

VPC で Amazon Rekognition の使用を開始するには、Amazon VPC コンソールで Amazon Rekognition 用のインターフェイス VPC エンドポイントを作成します。手順については、「インターフェイスエンドポイントの作成」で手順「コンソールを使用して AWS のサービスへのインターフェイスエンドポイントを作成するには」を参照してください。以下の手順のステップに注意してください。

- ステップ 3 [サービスカテゴリ] で、[AWS サービス] を選択します。
- ステップ 4 [サービス名] で、以下のオプションのいずれかを選択します。
  - com.amazonaws.region.rekognition Amazon Rekognition オペレーション用の VPC エンドポイントを作成します。
  - com.amazonaws.region.rekognition-fips 連邦情報処理標準 (FIPS) 刊行物 140-2 米国政府標準に準拠するエンドポイントを使用して、Amazon Rekognition オペレーションの VPC エンドポイントを作成します。

詳細については、『Amazon VPC ユーザーガイド』の開始方法を参照してください。

Amazon Rekognition の VPC エンドポイントポリシーの作成

Amazon Rekognition 用の Amazon VPC エンドポイントに対するポリシーを作成して、以下を指定することができます。

- アクションを実行できるプリンシパル。
- 実行可能なアクション。
- アクションを実行できるリソース。

詳細については、『Amazon VPC ユーザーガイド』の「VPC エンドポイントでサービスへのアクセスを制御する」を参照してください。

以下のポリシー例では、ユーザーは VPC エンドポイントを介して Amazon Rekognition に接続して `DetectFaces` API オペレーションを呼び出すことができます。このポリシーでは、ユーザーは VPC エンドポイントを介して Amazon Rekognition の他の API オペレーションを実行することはできません。

ユーザーは VPC の外部から、他の Amazon Rekognition API オペレーションを呼び出すことができます。VPC の外部にある Amazon Rekognition API オペレーションへのアクセスを拒否する方法については、「Amazon Rekognition のアイデンティティベースのポリシー」を参照してください。

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

Amazon Rekognition の VPC エンドポイントポリシーを変更するには

1. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. Amazon Rekognition のエンドポイントをまだ作成していない場合は、[エンドポイントの作成] を選択します。次に、[com.amazonaws.***Region***.rekognition] を選択し、[エンドポイントの作成] を選択します。
3. ナビゲーションペインで、[エンドポイント] を選択します。
4. [com.amazonaws.***Region***.rekognition] エンドポイントを選択し、画面の下部で [ポリシー] タブを選択します。
5. [ポリシーの編集] を選択してポリシーを変更します。

Amazon Rekognition のコンプライアンス検証

サードパーティーの監査者は、複数の コンプライアンスプログラムの一環として Amazon Rekognition のセキュリティと AWS コンプライアンスを評価します。これらのプログラムには、SOC、PCI、FedRAMP、HIPAA などがあります。

特定のコンプライアンスプログラムの対象となる AWS サービスのリストについては、「コンプライアンスプログラムによる AWS 対象範囲内のサービス」を参照してください。一般的な情報については、「AWS コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「Downloading Reports in AWS Artifact」(AWS Artifact のレポートのダウンロード) を参照してください。

Amazon Rekognition を使用する際のお客様のコンプライアンス責任は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 では、コンプライアンスに役立つ以下のリソース AWS を提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境をデプロイする手順について説明します AWS。
- 「HIPAA のセキュリティとコンプライアンスのための設計」ホワイトペーパー – このホワイトペーパーでは、企業が AWS を使用して HIPAA 準拠のアプリケーションを作成する方法について説明します。
- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や場所に適用される場合があります。
- AWS Config – この AWS サービスは、リソース設定が社内プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。
- AWS Security Hub – この AWS サービスは、 内のセキュリティ状態を包括的に把握 AWS し、セキュリティ業界標準とベストプラクティスへの準拠を確認するのに役立ちます。

Amazon Rekognition の耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョンとアベイラビリティーゾーンを中心に構築されています。 AWS リージョンは、低レイテンシー、高スループット、および高度に冗長なネットワークで接続された、物理的に分離された複数のアベイラビリティーゾーンを提供します。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、スケーラブルです。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

Amazon Rekognition は、 AWS グローバルインフラストラクチャに加えて、データの耐障害性とバックアップのニーズをサポートするのに役立ついくつかの機能を提供します。

Amazon Rekognition での設定と脆弱性の分析

設定および IT 管理は、AWS とお客様の間で共有される責任です。詳細については、 AWS 「責任共有モデル」を参照してください。

サービス間の混乱した代理の防止

では AWS、あるサービス (呼び出し元のサービス) が別のサービス (呼び出し元のサービス) を呼び出すと、サービス間のなりすましが発生する可能性があります。呼び出し側のサービスは、適切なアクセス許可を持たないはずの場合でも、別の顧客のリソースを操作するように操作される可能性があり、その結果、混乱した代理問題が発生します。

これを防ぐために、 は、アカウント内のリソースへのアクセス権が付与されたサービスプリンシパルを使用して、すべてのサービスのデータを保護するのに役立つツール AWS を提供します。

リソースポリシー内では `aws:SourceArn` および `aws:SourceAccount` のグローバル条件コンテキストキーを使用して、Amazon Rekognition が別のサービスに付与する、リソースへのアクセス許可を制限することをお勧めします。

`aws:SourceArn` の値に Amazon S3 バケット ARN などのアカウント ID が含まれていない場合は、両方のキーを使用して、アクセス許可を制限する必要があります。両方のキーを使用し、`aws:SourceArn` 値にアカウント ID が含まれている場合、`aws:SourceAccount` 値と `aws:SourceArn` 値内のアカウントは、同じポリシーステートメントで使用するときに、同じアカウント ID を使用する必要があります。

クロスサービスアクセスにリソースを 1 つだけ関連付けたい場合は、`aws:SourceArn` を使用します。そのアカウント内のリソースをクロスサービスの使用に関連付けることを許可する場合は、`aws:SourceAccount` を使用します。

`aws:SourceArn` の値は Rekognition が使用するリソースの ARN で、`arn:aws:rekognition:region:account:resource` の形式で指定する必要があります。

`arn:User` ARN の値は、ビデオ分析オペレーションを呼び出すユーザー (ロールを引き受けるユーザー) の ARN である必要があります。

代理の混乱が生じないようにするための最も効果的な方法は、リソースの完全な ARN を指定しながら、`aws:SourceArn` グローバル条件コンテキストキーを使用することです。

リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合には、aws:SourceArn キーで、ARN の未知部分を示すワイルドカード文字 (*) を使用します。例えば、arn:aws:*rekognition*:*:111122223333:* と指定します。

代理の混乱が生じないようにするには、次のステップを実行します。

1. IAM コンソールのナビゲーションペインで [ロール] をクリックします。コンソールに、現在のアカウントのロールが表示されます。
2. 修正するロールの名前を選択します。修正するロールには、AmazonRekognitionServiceRole アクセス権限ポリシーがあるはずです。[信頼関係] タブを選択します。
3. [信頼ポリシーを編集] を選択します。
4. [信頼ポリシーを編集] ページで、デフォルトの JSON ポリシーを、aws:SourceArn と aws:SourceAccount のグローバル条件コンテキストキーのいずれか、または両方を使用しているポリシーに置き換えます。以下のポリシー例を参照してください。
5. [ポリシーの更新] を選択してください。

以下は、Amazon Rekognition で aws:SourceArn および aws:SourceAccount グローバル条件コンテキストキーを使用して代理の混乱を防ぐ例です。

ビデオを保存してストリーミング配信する場合は、IAM ロールに次のようなポリシーを使用できます。

```
{
   "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/*"
```

```
            }
         }
      }
   ]
}
```

保存されたビデオのみを使用する場合は、IAM ロールに次のようなポリシーを使用できます (streamprocessor を指定する StringLike 引数を含める必要はないことにご注意ください)。

```
{
   "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"
            }
         }
      }
   ]
}
```

Amazon Rekognition のインフラストラクチャセキュリティ

マネージドサービスである Amazon Rekognition は AWS グローバルネットワークセキュリティで保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「セキュリティの柱 AWS Well-Architected フレームワーク」の「インフラストラクチャの保護」を参照してください。

AWS が公開した API コールを使用して、ネットワーク経由で Amazon Rekognition にアクセスします。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS)。TLS 1.2 が必須で、TLS 1.3 をお勧めします。

- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードはJava 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストにはアクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

Amazon Rekognition のモニタリング

モニタリングは、Amazon Rekognition といった AWS のソリューションの信頼性、可用性、パフォーマンスを維持するための重要な要素です。AWS は、Rekognition を監視して異常が検出された場合に報告を行い、必要に応じて自動アクションを実行できるように、次のモニタリングツールを提供します。

- Amazon CloudWatch は、 AWS リソースと、 で実行しているアプリケーションを AWS リアルタイムでモニタリングします。メトリクスの収集と追跡、カスタマイズしたダッシュボードの作成、および指定したメトリクスが指定したしきい値に達したときに通知またはアクションを実行するアラームの設定を行うことができます。例えば、CloudWatch で Amazon EC2 インスタンスの CPU 使用率などのメトリクスを追跡し、必要に応じて新しいインスタンスを自動的に起動できます。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。
- Amazon CloudWatch Logs では、Amazon EC2 インスタンス、CloudTrail、その他ソースから得たログファイルのモニタリング、保存、およびアクセスが可能です。CloudWatch Logs は、ログファイル内の情報をモニタリングし、特定のしきい値が満たされたときに通知します。高い耐久性を備えたストレージにログデータをアーカイブすることも可能です。詳細については、「Amazon CloudWatch Logs ユーザーガイド」を参照してください。
- Amazon EventBridge を使用すると、 AWS サービスを自動化し、アプリケーションの可用性の問題やリソースの変更などのシステムイベントに自動的に対応できます。 AWS サービスからのイベントは、ほぼリアルタイムで EventBridge に配信されます。簡単なルールを記述して、注目するイベントと、イベントがルールに一致した場合に自動的に実行するアクションを指定できます。詳細については、「Amazon EventBridge ユーザーガイド」を参照してください。
- AWS CloudTrail は、 AWS アカウントによって、またはアカウントに代わって行われた API コールおよび関連イベントをキャプチャし、指定した Amazon S3 バケットにログファイルを配信します。が呼び出したユーザーとアカウント AWS、呼び出し元のソース IP アドレス、および呼び出しの発生日時を特定できます。詳細については、AWS CloudTrail ユーザーガイドをご参照ください。

Amazon CloudWatch による Rekognition のモニタリング

CloudWatch を使用すると、個々の Rekognition オペレーションのメトリクス、またはアカウントのグローバルRekognition メトリクスを取得できます。メトリクスを使用して、Rekognition ベースのソリューションの健全性を追跡し、1つまたは複数のメトリクスが定義したしきい値を下回った場合に通知するアラームを設定することができます。メトリクスでは、サーバーエラーの発生数や顔の

削除数などを確認できます。特定の Rekognition オペレーションが成功した回数のメトリクスを確認することもできます。メトリクスを表示するには、Amazon CloudWatch、Amazon AWS Command Line Interface、または CloudWatch API を使用できます。

また、Rekognition コンソールを使用して、選択した期間の集計メトリクスを表示することもできます。詳細については、「演習 4: 集計メトリクスを参照する (コンソール)」を参照してください。

Rekognition での CloudWatch メトリクスの使用です。

メトリクスを使用するには、以下の情報を指定する必要があります。

- メトリクスのディメンション、またはディメンションなし。ディメンションは、メトリクスを一意に識別するための名前と値のペアです。Rekognition は Operation という 1 つのディメンションを持っています。オペレーション別のメトリクスを提供します。ディメンションを指定しないと、アカウント内のすべての Rekognition オペレーションがメトリクスの対象範囲になります。
- メトリクス名 (`UserErrorCount` など)。

AWS Management Console、 AWS CLI、または CloudWatch API を使用して Rekognition のモニタリングデータを取得することができます。CloudWatch API は、いずれかの Amazon AWS Software Development Kit (SDK) または CloudWatch API ツールでも使用できます。コンソールには、CloudWatch API の raw データに基づいて一連のグラフが表示されます。必要に応じて、コンソールに表示されるグラフまたは API から取得したグラフを使用できます。

以下のリストは、メトリクスの一般的な利用方法をいくつか示しています。ここで紹介するのは開始するための提案事項です。すべてを網羅しているわけではありません。

目的	関連するメトリクス
認識された顔の数を追跡する	`DetectedFaceCount` メトリクスの `Sum` 統計をモニタリングします。
アプリケーションが 1 秒あたりの最大リクエスト数に達したかどうかを確認する	`ThrottledCount` メトリクスの `Sum` 統計をモニタリングします。
リクエストエラーをモニタリングする	`UserErrorCount` メトリクスの `Sum` 統計を使用します。

目的	関連するメトリクス
リクエストの総数を確認する	ResponseTime と、ResponseTime メトリクスの Data Samples 統計を使用します。これには、エラーになったリクエストも含まれます。オペレーションの呼び出しが成功した回数のみを確認する場合は、SuccessfulRequestCount メトリクスを使用します。
Rekognition オペレーションの呼び出しのレイテンシーをモニタリングする	ResponseTime メトリクスを使用します。
IndexFaces が Rekognition コレクションに顔を何回正常に追加したかをモニタリングするにはどうすればよいですか？	SuccessfulRequestCount メトリクスの Sum 統計と IndexFaces オペレーションをモニタリングします。オペレーションとメトリクスを選択するには、Operation ディメンションを使用します。

CloudWatch で Rekognition をモニタリングするには、適切な CloudWatch のアクセス許可が必要です。詳細については、「Amazon CloudWatch に対する認証とアクセスコントロール」を参照してください。

Rekognition メトリクスにアクセスします。

次の例は、CloudWatch コンソール、、 AWS CLIおよび CloudWatch API を使用して Rekognition メトリクスにアクセスする方法を示しています。

メトリクスを表示するには (コンソール)

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. [メトリクス] を選択し、[すべてのメトリクス] タブを選択して、[Rekognition] を選択します。
3. [範囲の定められていないメトリクス] を選択し、メトリクスを選択します。

   たとえば、顔の検出数を確認するには、[DetectedFace] メトリクスを選びます。
4. 日付範囲の値を選択します。メトリクスのカウントがグラフに表示されます。

一定期間に **DetectFaces** オペレーションの呼び出しが成功した回数をメトリクスで確認するには (CLI)

- を開き AWS CLI 、次のコマンドを入力します。

  ```
  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
  ```

  この例では、一定期間に DetectFaces オペレーションの呼び出しが成功した回数を示しています。詳細については、「get-metric-statistics」を参照してください。

メトリクスにアクセスするには (CloudWatch API)

- GetMetricStatistics を呼び出します。詳細については、「Amazon CloudWatch API リファレンス」を参照してください。

アラームの作成

アラームの状態が変化したときに Amazon Simple Notification Service (Amazon SNS) メッセージを送信する CloudWatch のアラームを作成することができます。アラームは、指定期間にわたって単一のメトリクスを監視し、指定したしきい値に対応したメトリクスの値に基づいて、期間数にわたって 1 つ以上のアクションを実行します。アクションは、Amazon SNS トピックまたは Auto Scaling ポリシーに送信される通知です。

アラームは持続している状態変化に対してのみアクションを呼び出します。CloudWatch アラームは、特定の状態にあるという理由だけではアクションを呼び出しません。状態が変って、変わった状態が指定期間にわたって維持される必要があります。

アラームを設定するには (コンソール)

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
2. アラームの作成(アラームの作成) を選択します。これにより、[Create Alarm Wizard] が起動します。

3. [範囲の定められていないメトリクス] メトリクスリストで、[Rekognition メトリクス] を選択し、メトリクスを選択します。

   たとえば、顔の最大検出数のアラームを設定するには、[DetectedFaceCount] を選びます。

4. [時間範囲] 領域で、顔の検出オペレーションを呼び出す期間の値を選択します。[次へ] を選択します。

5. [名前] と [説明] を入力します。[次の時] で、[>=] を選択し、任意の最大値を入力します。

6. アラーム状態になったときに CloudWatch からメールを送信させたい場合は、[このアラームが鳴る時:] で、[アラーム状態] を選択します。既存の Amazon SNS トピックにアラームを送信するには、[通知の送信先:] で既存の SNS トピックを選択します。新しいメールサブスクリプションリストの名前と E メールアドレスを設定するには、[トピックの作成] を選択します。このリストは、 CloudWatch に保存されてフィールドに表示されるため、以降のアラーム設定に利用できます。

   > **Note**
   >
   > [トピックの作成] を使用して新しい Amazon SNS トピックを作成する場合、目的の受信者が通知を受け取る前に、 E メール アドレスを確認する必要があります。Amazon SNS は、アラームがアラーム状態になったときにのみメールを送信します。アラーム状態になったときにメールアドレスの検証がまだ完了していない場合、宛先には通知が届きません。

7. [Alarm Preview] (アラームの確認) セクションでアラームをプレビューします。アラームの作成(アラームの作成) を選択します。

アラームを設定するには (AWS CLI)

- を開き AWS CLI 、次のコマンドを入力します。`alarm-actions` パラメータの値を変更して、作成済みの Amazon SNS トピックを参照します。

```
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
```

この例では、5 分以内にユーザーエラーが 10 回を超えた場合のアラームの作成方法を示しています。詳細については、「put-metric-alarm」を参照してください。

アラームを設定するには (CloudWatch API)

- `PutMetricAlarm` を呼び出します。詳細については、「Amazon CloudWatch API リファレンス」を参照してください。

Rekognition の CloudWatch メトリクスです。

このセクションでは、Amazon Rekognition で利用可能な Amazon CloudWatch メトリクスと Operation ディメンションに関する情報を提供します。

Rekognition コンソールから Rekognition の集計メトリクスを確認することもできます。詳細については、「演習 4: 集計メトリクスを参照する (コンソール)」を参照してください。

Rekognition の CloudWatch メトリクスです。

次の表は、Rekognition メトリクスをまとめたものです。

メトリクス	説明
SuccessfulRequestCount	成功したリクエストの数。成功したリクエストのレスポンスコード範囲は 200〜299 です。 単位: 数 有効な統計: `Sum,Average`
ThrottledCount	スロットルされたリクエストの数。Rekognitionは、アカウントに設定された 1 秒あたりのトランザクションの制限を超えるリクエストを受信すると、リクエストをスロットルします。アカウントに設定された制限を頻繁に超える場合は、制限の引き上げをリクエストできます。引き上げをリクエストするには、「AWS サービス制限」を参照してください。 単位: 数 有効な統計: `Sum,Average`

メトリクス	説明
ResponseTime	Rekognition がレスポンスを計算するための時間 (ミリ秒)。 単位: 1. `Data Samples` 統計のカウント 2. `Average` 統計のミリ秒 有効な統計: `Data Samples,Average` **Note** `ResponseTime` メトリクスは Rekognition メトリクスペインに含まれていません。
DetectedFaceCount	`IndexFaces` または `DetectFaces` オペレーションで検出された顔の数。 単位: 数 有効な統計: `Sum,Average`
DetectedLabelCount	`DetectLabels` オペレーションで検出されたラベルの数。 単位: 数 有効な統計: `Sum,Average`
ServerErrorCount	サーバーエラーの数。サーバーエラーのレスポンスコード範囲は 500〜599 です。 単位: 数 有効な統計: `Sum, Average`

メトリクス	説明
UserErrorCount	ユーザーエラーの数 (無効なパラメータ、無効なイメージ、アクセス権限なしなど)。ユーザーエラーのレスポンスコード範囲は 400〜499 です。 単位: 数 有効な統計: `Sum,Average`
MinInferenceUnits	`StartProjectVersion` リクエスト中に指定された推論単位の最小数 単位: 数 有効な統計: `Average`
MaxInferenceUnits	`StartProjectVersion` リクエスト中に指定された推論単位の最大数 単位: 数 有効な統計: `Average`
DesiredInferenceUnits	Rekognition がスケールアップまたはスケールダウンする推論単位の数 単位: 数 有効な統計: `Average`
InServiceInferenceUnits	モデルで使用されている推論単位の数 単位: 数 有効な統計: `Average` 平均統計を使用して、使用されたインスタンス数の 1 分間の平均値を取得することをお勧めします。

Rekognition ストリーミングの CloudWatch メトリクス

また、Rekognition には、ストリーミングオペレーション「Rekognition ストリーミング」に使用される 2 つ目の名前空間があります。次の表は、Rekognition ストリーミングのメトリクスをまとめたものです。

メトリクス	説明
SuccessfulRequestCount	成功したリクエストの数。成功したリクエストのレスポンスコード範囲は 200〜299 です。 単位: 数 有効な統計: `Sum,Average`
CallCount	アカウントで実行された指定されたオペレーションの数。 有効な統計: `Sum,Average`
ThrottledCount	スロットルされたリクエストの数。Rekognitionは、アカウントに設定された 1 秒あたりのトランザクションの制限を超えるリクエストを受信すると、リクエストをスロットルします。アカウントに設定された制限を頻繁に超える場合は、制限の引き上げをリクエストできます。引き上げをリクエストするには、「AWS サービス制限」を参照してください。 単位: 数 有効な統計: `Sum,Average`
ServerErrorCount	サーバーエラーの数。サーバーエラーのレスポンスコード範囲は 500〜599 です。 単位: 数 有効な統計: `Sum, Average`
UserErrorCount	ユーザーエラーの数 (無効なパラメータ、無効なイメージ、アクセス権限なしなど)。ユーザーエラーのレスポンスコード範囲は 400〜499 です。 単位: 数

メトリクス	説明
	有効な統計: `Sum,Average`
ConcurrentSessions	アカウントが現在使用している同時セッションの数。 単位: 数 有効な統計: `Max,Average`

Rekognition の CloudWatch ディメンション

オペレーション別のメトリクスを取得するには、`Rekognition` 名前空間を使用して、オペレーションディメンションを指定します。

ディメンションの詳細については、「Amazon CloudWatch ユーザーガイド」の「ディメンション」を参照してください。

Rekognition Custom Labels の CloudWatch ディメンション

次の表は、Rekognition Custom Labels で使用できる CloudWatch ディメンションを示しています。

ディメンション	説明
ProjectName	`CreateProject` で作成した Rekognition Custom Labels プロジェクトの名前
VersionName	`CreateProjectVersion` で作成した Rekognition Custom Labels プロジェクトバージョンの名前

ディメンションの詳細については、「Amazon CloudWatch ユーザーガイド」の「ディメンション」を参照してください。

を使用した Amazon Rekognition API コールのログ記録 AWS CloudTrail

Amazon Rekognition は AWS CloudTrail、Amazon Rekognition のユーザー、ロール、または サービスによって実行されたアクションを記録する AWS サービスである と統合されています。 Amazon Rekognition CloudTrail は、Amazon Rekognition へのすべての API コールをイベントとしてキャプチャします。キャプチャされた呼び出しには、Amazon Rekognition コンソールからの呼び出しと、Amazon Rekognition API オペレーションへのコードの呼び出しが含まれます。証跡を作成する場合は、Amazon Rekognition のイベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます。証跡を設定しない場合でも、CloudTrail コンソールの [イベント履歴] で最新のイベントを表示できます。CloudTrail で収集された情報を使用して、Amazon Rekognition に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

CloudTrail の詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。

CloudTrail 内の Amazon Rekognition に関する情報

CloudTrail は、 AWS アカウントの作成時にアカウントで有効になります。Amazon Rekognition でアクティビティが発生すると、そのアクティビティはイベント履歴の他の AWS サービスイベントとともに CloudTrail イベントに記録されます。 AWS アカウントで最近のイベントを表示、検索、ダウンロードできます。詳細については、「CloudTrailイベント履歴でのイベントの表示」を参照してください。

Amazon Rekognition のイベントなど、 AWS アカウントのイベントの継続的な記録については、証跡を作成します。追跡により、CloudTrailはログファイルをSimple Storage Service (Amazon S3) バケットに配信できます。デフォルトでは、コンソールで作成した証跡がすべての AWS リージョンに適用されます。証跡は、 AWS パーティション内のすべてのリージョンからのイベントを口グに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、CloudTrail ログで収集されたイベントデータをより詳細に分析し、それに基づいて行動するように、他の AWS サービスを設定できます。詳細については、次を参照してください:

- 証跡の作成のための概要
- CloudTrail がサポートするサービスと統合
- CloudTrail 用 Amazon SNS 通知の構成
- 「複数のリージョンからCloudTrailログファイルを受け取る」および「複数のアカウントからCloudTrailログファイルを受け取る」

Amazon Rekognition でのアクションは全て CloudTrail に記録され Amazon Rekognition API リファレンス 内に保管されます。例えば、CreateCollection、CreateStreamProcessor、DetectCustomLabels の各アクションを呼び出すと、CloudTrail ログファイルにエントリが生成されます。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます。

- リクエストがルートまたは AWS Identity and Access Management (IAM) ユーザー認証情報を使用して行われたかどうか。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが別の AWS サービスによって行われたかどうか。

詳細については、CloudTrail userIdentity 要素を参照してください。

Amazon Rekognition ログファイルエントリを理解する

証跡は、指定した Amazon S3 バケットにイベントをログファイルとして送ることのできる設定機能です。CloudTrail のログファイルは、単一か複数のログエントリを含みます。イベントは任意ソースからの単一リクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどの情報を含みます。CloudTrail ログファイルは、パブリック API 呼び出しの順序付けられたスタックトレースではないため、特定の順序では表示されません。

以下の例では、API の StartLabelDetection および DetectLabels のアクションに伴う CloudTrail ログエントリを示します。

```
{
    "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"
        }
    ]
}
```

Amazon Rekognition のガイドラインとクォータ

以下のセクションでは、Amazon Rekognition を使用する際のガイドラインとクォータについて説明します。クォータは 2 種類あります。[クォータの設定] たとえば、イメージの最大サイズを変更することはできません。[デフォルトのクォータ] にリストされている AWS Service Quotas ページを変更するには、デフォルトのクォータ セクションで説明されている手順に従います。

トピック

- サポートされるリージョン
- クォータの設定
- デフォルトのクォータ

サポートされるリージョン

Amazon Rekognition が利用可能な AWS リージョンのリストについては、Amazon Web Services 全般のリファレンスの「AWS のリージョンとエンドポイント」を参照してください。

クォータの設定

以下は、変更できない Amazon Rekognition の制限の一覧です。トランザクション/秒 (TPS) など サービスの制限 に関する情報については、「デフォルトのクォータ」を参照してください。

Amazon Rekognition カスタムラベルの制限については、「Amazon Rekognition カスタムラベルのガイドラインとクォータ」を参照してください。

Amazon Rekognition イメージ

- Amazon S3 オブジェクトとして保存できるイメージの最大サイズは 15 MB に制限されています。
- `DetectModerationLabels` に利用できる高さと幅の最大イメージサイズはどちらも 10K ピクセルです。
- `DetectLabels` に利用できる高さと幅の最大イメージサイズはどちらも 10K ピクセルです。
- 1920x1080 ピクセルのイメージでは、顔が 40x40 ピクセル以上でないと検出できません。寸法が 1920X1080 ピクセルよりも大きいイメージでは、必要な最小顔サイズも比例してより大きくなります。

- 高さと幅の最小画像サイズはどちらも 80 ピクセルです。DetectProtectiveEquipment に利用できる高さと幅の最小画像サイズはどちらも 64 ピクセルです。
- DetectProtectiveEquipment に利用できる高さと幅の最大画像サイズはどちらも 4,096 ピクセルです。
- 800x1300 ピクセルのイメージでは、顔が 100x100 ピクセル以上でないと DetectProtectiveEquipment 検出できません。寸法が 800x1300 ピクセルよりも大きいイメージでは、必要な最小顔サイズも比例してより大きくなります。
- API にパラメータとして渡すイメージの最大サイズ (raw バイト) は 5 MB です。制限は 4 MB DetectProtectiveEquipment API。
- Amazon Rekognition は、PNG および JPEG イメージ形式をサポートしています。さまざまな API オペレーション (DetectLabels や IndexFaces など) に入力として渡すイメージは、サポートされている形式であることが必要です。
- 1 つの顔コレクションに保存できる顔ベクトルの最大数は 2,000 万です。
- 1 つの顔コレクションに保存できる顔ベクトルの、デフォルトの最大数は 1,000 万です。
- 検索 API から返される、一致する顔ベクトルの最大数は 4,096 です。
- 検索 API から返される、一致するユーザーベクトルの最大数は 4,096 です。
- DetectText はイメージから最大 100 個の単語を検出できます。
- DetectProtectiveEquipment は、15 人までの個人用保護具を検出できます。

イメージと顔の比較に関するベストプラクティスについては、「センサー、入力イメージ、ビデオのベストプラクティス」を参照してください。

Amazon Rekognition Image 一括分析

- Amazon Rekognition Image 一括分析では、最大 10,000 個の画像を一括で分析できます。
- Amazon Rekognition Image 一括分析では、最大 50 MB の入力マニフェストをサポートしています。

Amazon Rekognition Video 保存ビデオ

- Amazon Rekognition Video では、サイズが 10 GB までの保存済みビデオを分析できます。
- Amazon Rekognition Video では、長さが 6 時間までの保存済みビデオを分析できます。
- Amazon Rekognition Video は、アカウントあたり最大 20 の同時ジョブをサポートしています。

- 保存されたビデオは、H.264 コーデックを使用してエンコードする必要があります。サポートされているファイル形式は MPEG-4 および MOV です。
- オーディオデータを分析する Amazon Rekognition Video API は、AAC オーディオコーデックのみをサポートします。
- ページ分割トークンの有効期限 (TTL) は 24 時間です。ページ分割トークンは、`GetLabeldetection` など、Get オペレーションで返される `NextToken` フィールド内にあります。

Amazon Rekognition Video ストリーミング ビデオ

- Kinesis ビデオ入力ストリームは、最大 1 つの Amazon Rekognition Video ストリームプロセッサと関連付けることができます。
- Kinesis ビデオ入力ストリームは、最大 1 つの Amazon Rekognition Video ストリームプロセッサと関連付けることができます。
- Amazon Rekognition Video ストリームプロセッサに関連付けられた Kinesis ビデオ入力ストリームや Kinesis データ出力ストリームを複数のプロセッサで共有することはできません。
- オーディオデータを分析する Amazon Rekognition Video API は、ACC オーディオコーデックのみをサポートします。

デフォルトのクォータ

デフォルトのクォータのリストについては、「AWS Service Quotas」を参照してください。このデフォルトの制限は変更できます。制限の引き上げをリクエストするには、ケースを作成してください。現在のクォータ制限 (適用されたクォータ値) を確認するには、[Amazon Rekognition Service Quotas] を参照してください。Amazon Rekognition Image API の TPS 使用率履歴を表示するには「Amazon Rekognition Service Quotas ページ」を参照し、特定の API オペレーションを選択して、その操作の履歴を表示します。

トピック

- TPS クォータの変更を計算する
- TPS クォータのベストプラクティス
- TPS クォータを変更するケースを作成する

TPS クォータの変更を計算する

お客様が要求している新しい制限は何ですか？ 1 秒あたりのトランザクション数 (TPS) は、予想されるワークロードのピーク時に最も関連性があります。ワークロードのピーク時およびレスポンスの時間 (5 ~ 15 秒) の最大同時 API 呼び出しを理解することが重要です。最小が 5 秒のためご注意ください。以下に 2 つの例を挙げます。

- 例 1: 最も忙しい時間の開始時に予想される最大同時顔認証 (CompareFaces API) ユーザー数は 1000 です。これらの応答は 10 秒間にわたって広がります。したがって、関連するリージョンの CompareFaces API に必要なTPS は 100 (1000/10) です。
- 例 2: 最も忙しい時間の初めに予想される最大同時オブジェクト検出 (DetectLabels API) 呼び出しは 250 です。これらの応答は 5 秒間にわたって広がります。したがって、関連するリージョンの DetectLabels API に必要な TPS は 50 (250/5) です。

TPS クォータのベストプラクティス

Transactions Per Second (TPS) の推奨されるベストプラクティスには、スパイキートラフィックのスムーズ化、再試行の設定、エクスポネンシャルバックオフとジッタの設定などがあります。

1. スパイキートラフィックのスムーズ化。スパイキートラフィックはスループットに影響します。割り当てられたトランザクション/秒 (TPS) の最大スループットを得るには、キューイングサーバーレスアーキテクチャまたは別のメカニズムを使用してトラフィックを「スムーズ」し、一貫性を保ちます。Rekognition を使用したサーバーレス大規模イメージおよびビデオ処理のコードサンプルおよびリファレンスについては、Amazon Rekognition による大規模な画像や動画処理 を参照ください。
2. 再試行を設定します。the section called “エラー処理” のガイドラインをフォローして、エラーの再試行を構成します。
3. エクスポネンシャルバックオフとジッターの設定 リトライを設定するときにエクスポネンシャルバックオフとジッタを設定すると、達成可能なスループットを向上させることができます。「エラーの再試行とエクスポネンシャルバックオフ AWS」を参照してください。

TPS クォータを変更するケースを作成する

ケースを作成するには、「ケースの作成」を参照し、以下の質問に答えます。

- the section called “TPS クォータのベストプラクティス” トラフィックのスパイクをスムーズにし、再試行、エクスポネンシャルバックオフ、ジッタを設定するために実装しましたか？

- 必要な TPS クォータの変更を計算しましたか？ そうでない場合は、「the section called “TPS クォータの変更を計算する”」を参照してください。
- TPS の使用履歴を確認して、将来のニーズをより正確に予測しましたか？ TPS の使用履歴を表示するには、Amazon Rekognition Service Quotas ページ を参照ください。
- ユースケースは何ですか。
- どの API を使用しますか。
- これらの API を使用するリージョンはどれですか。
- 負荷を複数のリージョンに分散させることはできますか？
- 毎日何枚の画像を処理していますか。
- このボリュームを維持する期間はどれくらいですか (1 回限りのスパイクですか、それとも進行中ですか)。
- デフォルトの制限でどのようにブロックされていますか。次の例外表を参照して、発生したシナリオを確認します。

エラーコード	Exception	メッセージ	これはどういう意味ですか。	再試行できますか。
HTTP ステータスコード 400	ProvisionedThroughputExceededException	プロビジョンドレートを超過しました	スロットリングを示します。制限の引き上げリクエストを再実行または評価できます。	はい
HTTP ステータスコード 400	ThrottlingException	レートを下げてください: リクエストのレートが急激に増加しました。	スパイキートラフィックを送信し、スロットリングに遭遇している可能性があります。トラフィックを形作って、よりスムーズで一貫性のあるものにする必要がありま	はい

エラーコード	Exception	メッセージ	これはどういう意味ですか。	再試行できますか。
			す。次に、追加の再試行を設定します。ベストプラクティスを共有します。	
HTTP ステータスコード 5xx	ThrottlingException (HTTP 500)	Service Unavailable	バックエンドがアクションをサポートするためにスケールアップ中であることを示します。リクエストを再実行する必要があります。	はい

エラーコードの詳細については、「the section called “エラー処理”」を参照してください。

> **Note**
>
> これらの制限は、お住まいの地域によって異なります。制限を変更するケースは、リクエストしたリージョンで、リクエストした API オペレーションに影響します。その他の API オペレーションおよびリージョンは影響を受けません。

Amazon Rekognition のドキュメント履歴

次の表に、Amazon Rekognition デベロッパー向けガイド の各リリースの重要な変更点を示します。このドキュメントの更新に関する通知を受け取るには、RSS フィードにサブスクライブできます。

- 最終更新日: 2023 年 6 月 15 日

変更	説明	日付
Amazon Rekognition で新たなモデレーションラベルがサポートされたほか、画像に対するコンテンツのモデレーションの精度が向上	Amazon Rekognition のコンテンツモデレーションの機能が強化されて、精度が向上したほか、新たなラベルが検出されるようになりました。また、アニメーションやイラストのコンテンツを識別できるようになりました。	2024 年 2 月 1 日
Amazon Rekognition が一括イメージ分析をサポート	Amazon Rekognition が、StartMediaAnalysisJob オペレーションでマニフェストファイルを使用することによる、大量のイメージコレクションの非同期的処理のサポートを開始しました。	2023 年 10 月 23 日
Amazon Rekognition がアダプターを使用したカスタムコンテンツモデレーションをサポート	Amazon Rekognition が、既存の Rekognition 深層学習モデルの機能を拡張するアダプターを使用した、DetectModerationLabels API のさらに高い精度をサポートしました。	2023 年 10 月 12 日
Rekognition がコレクションを使用したユーザーベクトルをサポート	Rekognition の顔コレクションがユーザーベクトルの作成のサポートを開始しました。ユーザーベクトルは、同一	2023 年 6 月 12 日

	ユーザーの複数の顔ベクトルを集約し、ユーザーのより安定した描写で精度を高めます。	
ユーザーの管理に関係するアクションがマネージドポリシー AmazonRekognitionReadOnlyAccess に追加されました。	Amazon Rekognition は、アクション `ListUsers` 、`SearchUsers` 、`SearchUsersByImage` を `AmazonRekognitionReadOnlyAccess` マネージドポリシーに追加しました。	2023 年 6 月 12 日
Amazon Rekognition Image で視線の方向の推測が可能に	Amazon Rekognition Image の顔検出オペレーションの機能が向上し、検出された顔の視線の方向を推測できるようになりました。	2023 年 5 月 31 日
Rekognition コンテンツモデレーション API が改善	Rekognition で、イメージとビデオのモデレーションにおけるコンテンツモデレーションモデルが改善されました。この改善により、露骨な表現や暴力的、性的な表現を含むコンテンツの検出が大幅に拡大されました。露骨なコンテンツや暴力的なコンテンツをこれまでよりも高い精度で検出できるようになったため、エンドユーザーエクスペリエンスを改善し、ブランドアイデンティティを保護して、すべてのコンテンツで業界の規制や方針を確実に順守することが可能になります。	2023 年 5 月 9 日

Amazon Rekognition Image で隠れている顔の検出が可能に	Amazon Rekognition Image で隠れている顔の検出が可能になりました。新しい FaceOccluded 属性は、Amazon Rekognition Image の DetectFaces API と IndexFaces API によって返されます。これが示されるのは、イメージ内の顔が一部しかキャプチャされていないか、物、衣服、体の一部で隠れているために完全には表示されていない場合です。	2023 年 5 月 5 日
Rekognition で顔のライブネスの検出が可能に	Amazon Rekognition Video を使用してビデオ内のライブネスを検出し、カメラの前にいるユーザーが物理的に存在しているかどうかを確認できるようになりました。Face Liveness の検出機能は、カメラに仕掛けられたなりすまし攻撃、またはカメラを迂回しようとする攻撃も検出します。	2023 年 4 月 11 日

Amazon Rekognition Video にアップデート	Amazon Rekognition Video で、より多くのラベルを検出し、イメージやラベルの属性に関する情報をこれまでよりも多く返せるようになりました。GetLabelDetection API がエイリアスとカテゴリに関する情報を返すようになりました。返されたラベル情報は、包含および除外のフィルターオプションでフィルタリングできます。結果はタイムスタンプ別、またはビデオのセグメント別に集計できます。	2022 年 12 月 7 日
Amazon Rekognition Image にアップデート	Amazon Rekognition Image で、より多くのラベルを検出し、イメージやラベルの属性に関する情報をこれまでよりも多く返せるようになりました。DetectLabels API で、エイリアス、カテゴリ、さらに、ドミナントカラーなどのイメージプロパティに関する情報を返せるようになりました。返されたラベル情報は、包含および除外のフィルターオプションでフィルタリングできます。	2022 年 11 月 11 日

ProjectPolicy とカスタムラベルモデルコピーのアクションが、マネージドポリシー AmazonRekognitionReadOnlyAccess に追加されました。	Amazon Rekognition で、アクション ListProjectPolicies が AmazonRekognitionReadOnlyAccess マネージドポリシーに追加されました。	2022 年 7 月 21 日
ProjectPolicy とカスタムラベルモデルコピーのアクションが、マネージドポリシー AmazonRekognitionFullAccess、AmazonRekognitionCustomLabelsFullAccess に追加されました。	Rekognition で、アクション CopyProjectVersion 、PutProjectPolicy 、ListProjectPolicies 、DeleteProjectPolicy が AmazonRekognitionCustomLabelsFullAccess および AmazonRekognitionFullAccess マネージドポリシーに追加されました。	2022 年 7 月 21 日
Amazon Rekognition Video でストリーミングビデオのラベルの検出が可能に	Amazon Rekognition Video で、ペットやパッケージなどのラベルをストリーミングビデオで検出できるようになりました。この検出は、CreateStreamProcessor オペレーションで作成されたストリームプロセッサの ConnectedHome 設定を使用して行われます。	2022 年 4 月 28 日
Amazon Rekognition デベロッパーガイドから API リファレンスが削除されました。	Amazon Rekognition API リファレンスは、今後は「Amazon Rekognition API Reference」でご覧いただけます。	2022 年 2 月 24 日

マネージドポリシー AWS 管理ポリシー: AmazonRekognitionReadOnlyAccess、AWS 管理ポリシー:AmazonRekognitionFullAccess、AWS 管理ポリシー:Amazon RekognitionCustomLabelsFullAccess のデータセット管理が更新されました。	Amazon Rekognition で、アクション `CreateDataset` 、`ListDatasetEntries` 、`ListDatasetLabels` 、`DescribeDataset` 、`UpdateDatasetEntries` 、`DistributeDatasetEntries` 、`DeleteDataset` が `AmazonRekognitionReadOnlyAccess` 、`AmazonRekognitionFullOnlyAccess` 、`AmazonRekognitionCustomLabelsFullAccess` マネージドポリシーに追加されました。	2021 年 11 月 1 日
コンテンツウィンドウの新しいノードには、GitHub でホストされている Amazon Rekognition の例が表示されます	Code AWS Examples Repository から更新されたコード例は、Amazon Rekognition デベロッパーガイドの別のノードに表示され、アクセスが容易になりました。	2021 年 10 月 22 日

Amazon Rekognition は、ビデオセグメント内のブラックフレームとプライマリプログラムのコンテンツを検出できます	Amazon Rekognition は、StartSegmentDetection および GetSegmentDetection オペレーションを使用して、テクニカルキューとしてビデオ内の黒フレーム、カラーバー、オープニングクレジット、エンドクレジット、スタジオロゴ、および主要な番組コンテンツを識別できます。	2021 年 6 月 7 日
次の管理ポリシーのデータセット管理が更新されます。	イメージ内で最大 100 語を検出する Amazon Rekognition の DetectText オペレーションを使用できます。	2021 年 5 月 21 日
AmazonRekognitionReadOnlyAccess と AmazonRekognitionFullAccess でタグ付けがアップデート	Rekognition は、AmazonRekognitionFullAccess と AmazonRekognitionReadOnlyAccess ポリシーに新しいタグ付けアクションを追加しました。	2021 年 4 月 2 日
Amazon Rekognition がタグ付けをサポート	タグを使用して、Amazon Rekognition コレクション、ストリームプロセッサ、カスタムラベルモデルを識別、整理、検索、およびフィルタリングできるようになりました。	2021 年 3 月 25 日
Amazon Rekognition が個人用保護機器を検出できるようになりました	Amazon Rekognition は、イメージ内の人物のハンドカバー、フェイスカバー、ヘッドカバーを検出できるようになりました。	2020 年 10 月 15 日

Amazon Rekognition に新しいコンテンツモデレーションのカテゴリが追加されました	Amazon Rekognition コンテンツモデレーションのカテゴリには、薬物、タバコ、アルコール、ギャンブル、失礼なジェスチャー、ヘイトシンボルの 6 つの新しいカテゴリが追加されました。	2020 年 10 月 12 日
Kinesis Video Streams から Amazon Rekognition Video の結果をローカルに表示するための新しいチュートリアル	ローカルビデオフィードの Kinesis Video Streams 内のストリーミングビデオから Amazon Rekognition Video の出力を表示できます。	2020 年 7 月 20 日
Gstreamer を使用するための新しい Amazon Rekognition チュートリアル	Gstreamer を使用すると、Kinesis Video Streams を通して、デバイスカメラソースから Amazon Rekognition Video にライブストリームビデオを取り込むことができます。	2020 年 7 月 17 日
Amazon Rekognition で保存されたビデオのセグメンテーションをサポート	非同期の Amazon Rekognition Video セグメンテーション API を使用すると、保存されたビデオ内のブラックフレーム、カラーバー、エンドクレジット、およびショットを検出できます。	2020年6月22日
Amazon Rekognition で Amazon VPC エンドポイントのポリシーをサポート	ポリシーを指定することで、Amazon Rekognition Amazon VPC エンドポイントへのアクセスを制限できます。	2020 年 3 月 3 日

Amazon Rekognition では保存された動画のテキストの検出をサポート	Amazon Rekognition Video API を使用して、保存された動画のテキストを非同期的に検出できます。	2020 年 2 月 17 日
Amazon Rekognition は、Augmented AI (プレビュー) および Amazon Rekognition カスタムラベルをサポート	Amazon Rekognition カスタムラベルを使用すると、独自の機械学習モデルを作成することで、イメージ内の専用オブジェクト、シーン、概念を検出できます。DetectModerationLabels は Amazon Augmented AI (プレビュー) をサポートするようになりました。	2019 年 12 月 3 日
Amazon Rekognition が AWS PrivateLink をサポート	AWS PrivateLink を使用すると、VPC と Amazon Rekognition との間にプライベート接続を確立できます。	2019 年 9 月 12 日
Amazon Rekognition の顔フィルタリング	Amazon Rekognition は、IndexFaces API オペレーションに強化された顔フィルタリングのサポートを追加し、CompareFaces および SearchFacesByImage API オペレーションで顔フィルタリングを導入します。	2019 年 9 月 12 日
Amazon Rekognition Video の更新された例	Amazon SNS トピックと Amazon SQS キューを作成したり設定したりするために更新された Amazon Rekognition Video のサンプルコード	2019年9月5日

Ruby と Node.js の例の追加	同期ラベルおよび顔検出用に追加された Amazon Rekognition のイメージの Ruby と Node.js の例。	2019 年 8 月 19 日
安全でないコンテンツの検出の更新	Amazon Rekognition の安全でないコンテンツの検出により、暴力的なコンテンツを検出できるようになりました。	2019 年 8 月 9 日
GetContentModeration オペレーションの更新	GetContentModeration は、安全でないコンテンツを検出するために使用されるモデレーション検出のバージョンを返すようになりました。	2019 年 2 月 13 日
GetLabelDetection および DetectModerationLabels オペレーションが更新されました	GetLabelDetection が、一般的なオブジェクトの境界ボックス情報と、検出されたラベルの階層分類を返すようになりました。ラベル検出に使用されたモデルのバージョンを返すようになりました。DetectModerationLabels が、安全でないコンテンツの検出に使用されたモデルのバージョンを返すようになりました。	2019 年 1 月 17 日

更新された DetectFaces、IndexFaces オペレーション	このリリースでは、DetectFaces および IndexFaces オペレーションが更新されています。Attributes 入力パラメータが ALL に設定されている場合、顔の位置のランドマークに 5 つの新しいランドマークが含まれています: upperJawlineLeft、midJawlineLeft、chinBottom、midJawlineRight、upperJawlineRight。	2018 年 11 月 19 日
更新された DetectLabels オペレーション	特定の被写体に対して境界ボックスが返されるようになりました。階層分類がラベルに使用できるようになりました。検出に使用された検出モデルのバージョンを入手できるようになりました。	2018 年 11 月 1 日
IndexFaces オペレーションの更新	IndexFaces では、QualityFilter 入力パラメーターを使用して低品質で検出された顔を除外できるようになりました。MaxFaces 入力パラメーターを使用し、顔検出の品質と検出された顔のサイズに基づいて返される顔の数を減らすこともできます。	2018 年 9 月 18 日
DescribeCollection オペレーションが追加されました。	DescribeCollection オペレーションを呼び出すことで、既存のコレクションに関する情報を取得できるようになりました。	2018 年 8 月 22 日

Python の新しい例	Python の例が Amazon Rekognition Video のコンテンツに追加され、一部のコンテンツが再編成されました。	2018 年 6 月 26 日
コンテンツのレイアウトの更新	Amazon Rekognition イメージのコンテンツが再構成され、新しい Python および C# の例が追加されました。	2018 年 5 月 29 日
Amazon Rekognition が をサポート AWS CloudTrail	Amazon Rekognition は AWS CloudTrail、Amazon Rekognition のユーザー、ロール、または サービスによって実行されたアクションを記録する AWS サービスである と統合されています。 Amazon Rekognition 詳細については、「AWS CloudTrail を使用した Amazon Rekognition API コールのログ記録」を参照してください。	2018 年 4 月 6 日
保存したビデオとストリーミングビデオを分析する。新しい目次	保存したビデオの分析の詳細については、「保存したビデオの使用」を参照してください。ストリーミングビデオの分析の詳細については、「ストリーミングビデオの使用」を参照してください。Amazon Rekognition ドキュメントの目次は、イメージおよびビデオのオペレーションを反映して再編されています。	2017 年 11 月 29 日

イメージ内のテキストと顔の検出モデル	Amazon Rekognition でイメージ内のテキストを検出できるようになりました。詳細については、「テキストの検出」を参照してください。Amazon Rekognition で顔検出の深層学習モデルのバージョニングが導入されました。詳細については、「モデルのバージョニング」を参照してください。	2017 年 11 月 21 日
有名人の認識	Amazon Rekognition で有名人のイメージを分析できるようになりました。詳細については、「有名人の認識」を参照してください。	2017 年 6 月 8 日
イメージモデレーション	Amazon Rekognition では、イメージに明示的または暗示的なアダルトコンテンツが含まれているかどうかを判定できるようになりました。詳細については、「安全でないコンテンツの検出」を参照してください。	2017 年 4 月 19 日

検出された顔の年齢層。Rekognition の集計メトリクスペイン	Amazon Rekognition は、Rekognition API で検出された顔の推定年齢層を年数で返すようになりました。詳細については、「AgeRange」を参照してください。Rekognition コンソールのメトリクスペインに、指定した期間にわたる Rekognition の Amazon CloudWatch メトリクスの集計がアクティビティグラフとして表示されるようになりました。詳細については、「演習 4: 集計メトリクスを確認する (コンソール)」を参照してください。	2017 年 2 月 9 日
新しいサービスとガイド	これはイメージ分析サービスである Amazon Rekognition および Amazon Rekognition デベロッパー向けガイド の初回リリースです。	2016 年 11 月 30 日

AWS 用語集

最新の用語については、 AWS 「 AWS の用語集 リファレンス」のAWS 「 用語集」を参照してください。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。