



Custom Labels ガイド

# Rekognition



# Rekognition: Custom Labels ガイド

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標とトレードドレスは、Amazon 以外の製品またはサービスとの関連において、顧客に混乱を招いたり、Amazon の名誉または信用を毀損するような方法で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 Custom Labels とは。 .....	1
主な利点 .....	1
Amazon Rekognition Custom Labels の使用を選択する .....	2
Amazon Rekognition Image ラベルの検出 .....	2
Amazon Rekognition Custom Labels .....	3
Amazon Rekognition Custom Labels を初めて使用するユーザー向けの情報。 .....	4
Amazon Rekognition Custom Labels のセットアップ .....	5
ステップ 1: AWS アカウントを作成する .....	5
にサインアップする AWS アカウント .....	6
管理アクセスを持つユーザーを作成する .....	6
プログラマ的なアクセス .....	8
ステップ 2: コンソールのアクセス許可のセットアップ .....	9
コンソールへのアクセスを許可する .....	10
外部の Amazon S3 バケットへのアクセス .....	11
アクセス許可の割り当て .....	12
ステップ 3: コンソールバケットの作成 .....	12
ステップ 4: AWS CLI と AWS SDKsを設定する .....	13
AWS SDK のインストール .....	13
プログラマチックアクセス権を付与する .....	8
SDK アクセス許可の設定 .....	18
オペレーションを呼び出す .....	19
ステップ 5: (オプション) トレーニングファイルを暗号化する .....	23
で暗号化されたファイルの復号化 AWS Key Management Service .....	24
コピーしたトレーニングイメージとテストイメージを暗号化する .....	24
ステップ 6: (オプション) 以前のデータセットを関連付ける .....	25
以前のデータセットをテストデータセットとして使用する .....	26
Amazon Rekognition Custom Labels について .....	27
モデルタイプの決定 .....	27
オブジェクト、シーン、概念を検出する .....	28
オブジェクトの位置の検索 .....	29
ブランドの所在地を探す .....	29
モデルを作成する .....	30
プロジェクトを作成する .....	30
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 .....	31

モデルをトレーニングする .....	32
モデルの改善 .....	33
モデルの評価 .....	33
モデルの改善 .....	34
モデルの開始 .....	34
モデルの開始 (コンソール) .....	35
モデルの開始 .....	35
イメージの分析 .....	35
モデルの停止 .....	37
モデルの停止する (コンソール) .....	37
モデルの停止 (SDK) .....	37
開始 .....	38
チュートリアルビデオ .....	38
プロジェクトの例 .....	39
画像分類 .....	39
マルチラベルイメージ分類 .....	39
ブランド検出 .....	40
オブジェクトのローカリゼーション .....	40
サンプルプロジェクトの使用 .....	41
プロジェクト例の作成 .....	41
モデルのトレーニング .....	42
データモデルの使用 .....	42
次のステップ .....	42
ステップ 1: プロジェクト例を選択する .....	42
ステップ 2: モデルのトレーニング .....	45
ステップ 3: モデルをスタートする .....	50
ステップ 4: モデルを使用してイメージを分析する .....	51
イメージ例の取得 .....	56
ステップ 5: モデルを停止する .....	58
ステップ 6: 次のステップ .....	60
チュートリアル: イメージの分類 .....	62
ステップ 1: イメージを集める .....	62
ステップ 2: クラスを決める .....	63
ステップ 3: プロジェクトを作成する .....	64
ステップ 4: トレーニングデータセットおよびテストデータセットを作成する .....	65
ステップ 5: ラベルをプロジェクトに追加する .....	71



ステップ 6: トレーニングデータセットとテストデータセットに画像レベルのラベルを割り当てる .....	72
ステップ 7: モデルのトレーニング .....	73
ステップ 8: モデルをスタートする .....	78
ステップ 9: モデルを使用してイメージを分析する .....	80
ステップ 10: モデルを停止する .....	83
モデルの作成 .....	86
プロジェクトの作成 .....	86
プロジェクトの作成 (コンソール) .....	86
プロジェクトの作成 (SDK) .....	87
データセットの作成 .....	92
データセットの目的の設定 .....	93
イメージの準備 .....	98
イメージ付きのデータセットの作成 .....	100
イメージにラベルを付ける .....	162
データセットのデバッグ .....	172
モデルのトレーニング .....	179
モデルのトレーニング (コンソール) .....	181
モデルのトレーニング (SDK) .....	185
モデルトレーニングのデバッグ .....	195
ターミナルエラー .....	196
非ターミナル JSON 行検証エラー .....	198
マニフェストの概要について .....	199
トレーニングとテストの検証結果マニフェストを理解する .....	203
検証結果の取得 .....	208
トレーニングエラーの修正 .....	211
ターミナルマニフェストファイルエラー .....	213
ターミナルマニフェストコンテンツエラー .....	215
非ターミナル JSON 行検証エラー .....	225
トレーニング済みモデルの改善 .....	249
モデルを評価するためのメトリクス .....	249
モデルパフォーマンスの評価 .....	250
想定しきい値 .....	251
適合率 .....	251
リコール .....	252
F1 .....	252

メトリクスの使用 .....	253
評価メトリクスへのアクセス (コンソール) .....	253
評価メトリクスへのアクセス (SDK) .....	256
概要ファイル .....	257
評価マニフェストのスナップショット .....	259
概要ファイルと評価マニフェストスナップショット (SDK) へのアクセス .....	263
モデルの混同行列の表示 .....	264
リファレンス: 概要ファイル .....	271
モデルの改善 .....	273
[データ] .....	273
偽陽性の削減 (適合率の向上) .....	274
偽陰性の削減 (再現率の向上) .....	274
トレーニング済みモデルの実行 .....	275
推論単位 .....	275
推論単位によるスループットの管理 .....	276
アベイラビリティゾーン .....	278
モデルの開始 .....	278
モデルの開始または停止 (コンソール) .....	279
モデル (SDK) の開始 .....	280
モデルの停止 .....	290
モデルの停止 (コンソール) .....	290
モデル (SDK) の停止 .....	291
時間と推論単位のレポート .....	300
イメージの分析 .....	303
DetectCustomLabels オペレーションリクエスト .....	329
DetectCustomLabels オペレーションレスポンス .....	330
リソースの管理 .....	331
プロジェクトの管理 .....	331
プロジェクトの削除 .....	331
プロジェクトの記述 (SDK) .....	341
を使用したプロジェクトの作成 AWS CloudFormation .....	348
データセットの管理 .....	349
データセットの追加 .....	349
画像をさらに追加する .....	359
既存のデータセットを使用したデータセットの作成 (SDK) .....	369
データセットの記述 (SDK) .....	378

データセットエントリの一覧表示 (SDK) .....	383
トレーニングデータセットの分散 (SDK) .....	389
データセットの削除 .....	399
モデルの管理 .....	406
モデルの削除 .....	407
モデルのタグ付け .....	416
モデルの記述 (SDK) .....	423
モデルのコピー (SDK) .....	431
例 .....	468
モデルフィードバックソリューション .....	468
Amazon Rekognition Custom Labels のデモンストレーション .....	469
動画分析 .....	469
AWS Lambda 関数によるイメージ分析 .....	472
ステップ 1: AWS Lambda 関数を作成する (コンソール) .....	472
ステップ 2: (オプション) レイヤーを作成する (コンソール) .....	475
ステップ 3: Python コードを追加する (コンソール) .....	476
ステップ 4: Lambda 関数をテストする .....	479
セキュリティ .....	484
Amazon Rekognition Custom Labels プロジェクトの保護 .....	484
DetectCustomLabels の保護 .....	485
AWS マネージドポリシー .....	486
ガイドラインとクォータ .....	487
サポートされるリージョン .....	487
クォータ .....	487
トレーニング .....	487
テスト .....	488
検出 .....	489
モデルのコピー .....	489
API リファレンス .....	490
モデルのトレーニング .....	501
プロジェクト .....	501
プロジェクトポリシー .....	501
データセット .....	501
モデル .....	502
タグ .....	501
モデルの使用 .....	502

---

ドキュメント履歴 .....	503
AWS 用語集 .....	509
.....	dx

# Amazon Rekognition Custom Labels とは。

Amazon Rekognition Custom Labels を使用して、ビジネスニーズに合わせた画像のオブジェクト、ロゴ、シーンを特定できます。例えば、ソーシャルメディアの記事から自社のロゴを検索したり、店頭で商品を特定したり、組立ラインで機械部品を分類したり、健康な植物と病気に感染した植物とを区別したり、画像のアニメーションキャラクターを検出したりできます。

画像を分析するカスタムモデルの開発は、時間、専門知識、リソースを必要とする重要な作業です。多くの場合、完了するまでに数か月かかります。さらに、正確な判断を下すのに十分なデータをモデルに提供するには、手作業によるラベル付けされた画像が数千から数万必要になることもあります。このデータを生成するには収集に数か月かかることがあり、機械学習で使用できるように準備するために大規模なラベラーチームが必要になることもあります。

Amazon Rekognition Custom Labels は、Amazon Rekognition の既存の機能を拡張するものです。これらの機能は、多くのカテゴリにわたる数千万の画像で既にトレーニングされています。何千もの画像の代わりに、ユースケースに特化した少数のトレーニングイメージ (通常は数百枚以下の画像) をアップロードできます。これは、使いやすいコンソールを使用して行うことができます。イメージに既にラベルが付けられている場合、Amazon Rekognition Custom Labels は短時間でモデルのトレーニングを開始できます。そうでない場合は、ラベル付けインターフェイス内で直接イメージにラベルを付けることも、Amazon SageMaker Ground Truth を使用してイメージにラベルを付けることもできます。

Amazon Rekognition Custom Labels がイメージセットからトレーニングを開始すると、わずか数時間でカスタムイメージ分析モデルを作成できます。Amazon Rekognition Custom Labels はバックグラウンドでトレーニングデータを自動的にロードして検査し、適切な機械学習アルゴリズムを選択し、モデルをトレーニングし、モデルパフォーマンスメトリクスを提供します。その後、Amazon Rekognition Custom Labels API を通じてカスタムモデルを使用し、アプリケーションに統合できます。

## トピック

- [主な利点](#)
- [Amazon Rekognition Custom Labels の使用を選択する](#)
- [Amazon Rekognition Custom Labels を初めて使用するユーザー向けの情報。](#)

## 主な利点

データラベリングの簡略化

Amazon Rekognition Custom Labels コンソールには、イメージにすばやく簡単にラベルを付けるための視覚的なインターフェイスがあります。このインターフェイスでは、イメージ全体にラベルを付けることができます。クリックアンドドラッグインターフェイスの境界ボックスを使用して、画像内の特定のオブジェクトを識別してラベルを付けることもできます。また、データセットが大きい場合は、[Amazon SageMaker Ground Truth](#) を使用してイメージに大規模に効率的にラベルを付けることができます。

## 自動機械学習

カスタムモデルを構築するのに、機械学習の専門知識は必要ありません。Amazon Rekognition Custom Labels には、機械学習を自動的に行う自動機械学習 (AutoML) 機能が含まれています。トレーニングイメージが提供されると、Amazon Rekognition Custom Labels は自動的にデータをロードして検査し、適切な機械学習アルゴリズムを選択し、モデルをトレーニングし、モデルパフォーマンスメトリクスを提供できます。

## モデルの評価、推論、フィードバックの簡略化

テストセットでカスタムモデルのパフォーマンスを評価します。テストセット内のすべてのイメージについて、モデルの予測と割り当てられたラベルを並べて比較できます。また、適合率、再現率、F1 スコア、信頼度スコアなどの詳細なパフォーマンスメトリクスを確認することもできます。モデルをすぐに画像分析に使用することも、新しいバージョンを繰り返し処理してイメージを増やして再トレーニングしてパフォーマンスを向上させることもできます。モデルの使用を開始したら、予測を追跡して間違いを修正し、フィードバックデータを使用して新しいモデルバージョンを再トレーニングしてパフォーマンスを向上させます。

# Amazon Rekognition Custom Labels の使用を選択する

Amazon Rekognition には、イメージ内のラベル (オブジェクト、シーン、概念) の検索に使用できる 2 つの機能があります。Amazon Rekognition Custom Labels と [Amazon Rekognition Image ラベル検出](#) です。次の情報を使用して、どの機能を使用すべきかを判断してください。

## Amazon Rekognition Image ラベルの検出

Amazon Rekognition Image のラベル検出機能を使用すると、機械学習モデルを作成しなくても、イメージや動画の一般的なラベルを大規模に識別、分類、検索できます。例えば、車やトラック、トマト、バスケットボール、サッカーボールなど、何千もの一般的なオブジェクトを簡単に検出できます。

アプリケーションで一般的なラベルを検出する必要がある場合は、モデルをトレーニングする必要がないため、Amazon Rekognition Image ラベル検出を使用することをお勧めします。Amazon

Rekognition Image ラベル検出で検出されたラベルのリストを取得するには、「[ラベルの検出](#)」を参照してください。

Amazon Rekognition Image のラベル検出では検出されないラベル (組立ラインのカスタム機械部品など) をアプリケーションで検出する必要がある場合は、Amazon Rekognition Custom Labels を使用することをお勧めします。

## Amazon Rekognition Custom Labels

Amazon Rekognition Custom Labels を使用すると、ビジネスニーズに合ったイメージ内のラベル (オブジェクト、ロゴ、シーン、概念) を検索する機械学習モデルを簡単にトレーニングできます。

Amazon Rekognition Custom Labels では、イメージを分類 (イメージレベルの予測) したり、イメージ内のオブジェクトの位置を検出したりできます (オブジェクト/境界ボックスレベルの予測)。

Amazon Rekognition Custom Labels を使用すると、検出できるオブジェクトやシーンのタイプをより柔軟に設定できます。例えば、Amazon Rekognition Image ラベル検出を使用して、植物や葉を検索できます。健康な植物、損傷した植物、感染した植物を区別するには、Amazon Rekognition Custom Labels を使用する必要があります。

Amazon Rekognition Custom Labels の使用例を次に示します。

- 選手のジャージやヘルメットのチームロゴを識別する
- 特定の機械部品や組立ライン上の製品を区別する
- メディアライブラリ内の漫画のキャラクターを識別する
- 小売り店の棚にある特定のブランドの商品を探す
- 農産物の品質 (腐った、熟した、生のものなど) を分類する

### Note

Amazon Rekognition Custom Labels は、顔の分析、テキストの検出、またはイメージ内の安全でないイメージコンテンツの検出を目的として設計されていません。これらのタスクを実行するには、Amazon Rekognition Image を使用できます。詳細については、「[Amazon Rekognition とは](#)」を参照してください。

# Amazon Rekognition Custom Labels を初めて使用するユーザー向けの情報。

Amazon Rekognition Custom Labels 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 Custom Labels のセットアップ](#) - このセクションでは、お客様のアカウント情報を設定します。
2. [Amazon Rekognition Custom Labels について](#) - このセクションでは、モデルを作成するためのワークフローについて学習します。
3. [Amazon Rekognition Custom Labels の開始方法](#) - このセクションでは、Amazon Rekognition Custom Labels によって作成されたサンプルプロジェクトを使用してモデルをトレーニングします。
4. [チュートリアル: イメージの分類](#) - このセクションでは、作成したデータセットを使用してイメージを分類するモデルをトレーニングする方法を学びます。



# Amazon Rekognition Custom Labels のセットアップ

次の手順は、Amazon Rekognition Custom Labels コンソールと SDK をセットアップする方法を示しています。

Amazon Rekognition Custom Labels コンソールは次のブラウザで使用できることに注意してください。

- Chrome - バージョン 21 以降
- Firefox - バージョン 27 以降
- Microsoft Edge - バージョン 88 以降
- Safari - バージョン 7 以降。また、Safari を使用して、Amazon Rekognition Custom Labels コンソールで境界ボックスを描画することはできません。詳細については、「[境界ボックスによるオブジェクトのラベル付け](#)」を参照してください。

Amazon Rekognition Custom Labels 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してください。

## トピック

- [ステップ 1: AWS アカウントを作成する](#)
- [ステップ 2: Amazon Rekognition Custom Labels コンソールのアクセス許可をセットアップする](#)
- [ステップ 3: コンソールバケットの作成](#)
- [ステップ 4: AWS CLI と AWS SDKsを設定する](#)
- [ステップ 5: \(オプション\) トレーニングファイルを暗号化する](#)
- [ステップ 6: \(オプション\) 以前のデータセットを新しいプロジェクトに関連付ける](#)

## ステップ 1: AWS アカウントを作成する

このステップでは、AWS アカウントを作成し、管理ユーザーを作成し、AWS SDK へのプログラムによるアクセス権の付与について説明します。

## トピック

- [にサインアップする AWS アカウント](#)
- [管理アクセスを持つユーザーを作成する](#)

## • プログラマ的なアクセス

### 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、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

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。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用して ルートユーザーアクセスが必要なタスク を実行してください。

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] を選んで、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、 を保護し AWS アカウントのルートユーザー、 を有効にして AWS IAM Identity Center、日常的なタスクにルートユーザーを使用しないように管理ユーザーを作成します。

のセキュリティ保護 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「[ルートユーザーの仮想 MFA デバイスを有効にする \(コンソール\)](#)」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、「[ユーザーガイド](#)」の「[デフォルトで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](#)」AWS IAM Identity Center」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「[グループの参加](#)」を参照してください。

## プログラムのなアクセス

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、 に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 Identity Center 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 SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。  • <a href="#">については AWS CLI、「ユーザーガイド」の AWS CLI 「を使用するための の設定 AWS IAM Identity CenterAWS Command Line Interface」を参照してください。</a>  • AWS SDKs 、ツール、AWS APIs 「SDK とツールのリファレンスガイド」の <a href="#">「IAM Identity Center 認証」</a> 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 SDKs AWS CLI、または AWS APIs。	<a href="#">「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」</a> の手順に従います。
IAM	(非推奨) 長期認証情報を使用して、AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。

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
		<ul style="list-style-type: none"><li>• については AWS CLI、「<a href="#">AWS Command Line Interface ユーザーガイド</a>」の「<a href="#">IAM ユーザー認証情報を使用した認証</a>」を参照してください。</li><li>• AWS SDKs「<a href="#">SDK とツールのリファレンスガイド</a>」の「<a href="#">長期的な認証情報を使用した認証</a>」を参照してください。AWS SDKs</li><li>• AWS APIs <a href="#">ユーザーガイド</a>」の「<a href="#">IAM ユーザーのアクセスキーの管理</a>」を参照してください。</li></ul>

## ステップ 2: Amazon Rekognition Custom Labels コンソールのアクセス許可をセットアップする

Amazon Rekognition コンソールを使用するには、適切なアクセス許可を追加する必要があります。トレーニングファイルを[コンソールバケット](#)以外のバケットに保存する場合は、追加のアクセス許可が必要です。

### トピック

- [コンソールへのアクセスを許可する](#)
- [外部の Amazon S3 バケットへのアクセス](#)
- [アクセス許可の割り当て](#)

## コンソールへのアクセスを許可する

Amazon Rekognition Custom Labels コンソールを使用するには、Amazon S3、SageMaker Ground Truth、および Amazon Rekognition Custom Labels を対象とする次の IAM ポリシーが必要です。アクセス許可の割り当ての詳細については、「[アクセス許可の割り当て](#)」を参照してください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:ListAllMyBuckets"
      ],
      "Resource": "*"
    },
    {
      "Sid": "s3Policies",
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:CreateBucket",
        "s3:GetBucketAcl",
        "s3:GetBucketLocation",
        "s3:GetObject",
        "s3:GetObjectAcl",
        "s3:GetObjectVersion",
        "s3:GetObjectTagging",
        "s3:GetBucketVersioning",
        "s3:GetObjectVersionTagging",
        "s3:PutBucketCORS",
        "s3:PutLifecycleConfiguration",
        "s3:PutBucketPolicy",
        "s3:PutObject",
        "s3:PutObjectTagging",
        "s3:PutBucketVersioning",
        "s3:PutObjectVersionTagging"
      ],
      "Resource": [
        "arn:aws:s3:::custom-labels-console-*"
      ]
    }
  ]
}
```

```
    },
    {
      "Sid": "rekognitionPolicies",
      "Effect": "Allow",
      "Action": [
        "rekognition:*"
      ],
      "Resource": "*"
    },
    {
      "Sid": "groundTruthPolicies",
      "Effect": "Allow",
      "Action": [
        "groundtruthlabeling:*"
      ],
      "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 Amazon Rekognition Custom Labels コンソールを初めて開くと、Amazon Rekognition Custom Labels 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、前のポリシーに次のポリシーブロックを追加します。my-bucket をバケットの名前に置き換えます。

```
{
  "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:::my-bucket*"  
]  
}
```

## アクセス許可の割り当て

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。

- 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。IAM ユーザーガイドの「[ユーザー \(コンソール\) へのアクセス許可の追加](#)」の指示に従います。

## ステップ 3: コンソールバケットの作成

Amazon Rekognition Custom Labels プロジェクトを使用して、モデルを作成および管理します。新しい AWS リージョンで Amazon Rekognition Custom Labels コンソールを初めて開くと、Amazon Rekognition Custom Labels は Amazon S3 バケット (コンソールバケット) を作成してプロジェクトを保存します。AWS SDK オペレーションやデータセットの作成などのコンソールタスクでバケット名を使用する必要がある可能性があるため、後で参照できる場所にコンソールバケット名を書き留めておく必要があります。

バケット名の形式は次のとおりです。custom-labels-console-*<region>*-*<random value>*。  
ランダム値により、バケット名の中に衝突が発生しないようにします。



コンソールバケットを作成するには

1. ユーザーに正しいアクセス許可があるかどうかを確認します。詳細については、「[コンソールへのアクセスを許可する](#)」を参照してください。
2. 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。
3. [開始する] を選びます。
4. 現在の AWS リージョンでコンソールを初めて開いた場合は、[First Time Set Up] (初回セットアップ) ダイアログボックスで以下を実行します。
  - a. 表示されている Amazon S3 バケットの名前をコピーします。この情報は後で必要になります。
  - b. Amazon Rekognition Custom Labels に Amazon S3 バケット (コンソールバケット) の作成を代行させる場合は、[S3 バケットを作成する] を選択します。
5. ブラウザウィンドウを閉じます。

## ステップ 4: AWS CLI と AWS SDKs を設定する

Amazon Rekognition Custom Labels は、AWS Command Line Interface (AWS CLI) および AWS SDKs で使用できます。ターミナルから Amazon Rekognition Custom Labels オペレーションを実行する必要がある場合は、AWS CLI をインストールします。アプリケーションを作成する場合は、使用しているプログラミング言語の AWS SDK をダウンロードします。

トピック

- [AWS SDK のインストール](#)
- [プログラマチックアクセス権を付与する](#)
- [SDK アクセス許可の設定](#)
- [Amazon Rekognition Custom Labels オペレーションを呼び出す](#)

## AWS SDK のインストール

AWS SDKs をダウンロードして設定するには、次の手順に従います。

AWS CLI と AWS SDKs をセットアップするには

- および使用する AWS SDKs をダウンロードしてインストール [AWS CLI](#) します。このガイドでは、[Java AWS CLI](#)、および [Python の例を示します](#)。AWS SDKs [「Amazon Web Services のツール」](#) を参照してください。

## 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 コード例の実行の詳細については、「」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、 に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 Identity Center 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 SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の の手引きに従ってください。  • については AWS CLI、 「 <a href="#">ユーザーガイド</a> 」の

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
		<p><a href="#">AWS CLI 「を使用するための の設定 AWS IAM Identity CenterAWS Command Line Interface」</a> を参照してください。</p> <ul style="list-style-type: none"> <li>• AWS SDKs、ツール、AWS APIs 「SDK とツールのリファレンスガイド」の <a href="#">「IAM Identity Center 認証」</a> を参照してください。 AWS SDKs</li> </ul>
IAM	一時的な認証情報を使用して、AWS SDKs AWS CLI、または AWS APIs。	<p>「IAM <a href="#">ユーザーガイド</a>」の <a href="#">「AWS リソースでの一時的な認証情報の使用」</a> の手順に従います。</p>

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
IAM	(非推奨) 長期認証情報を使用して、AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 <ul style="list-style-type: none"> <li>• 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。</li> <li>• AWS SDKs 「SDK とツールのリファレンスガイド」の「長期的な認証情報を使用した認証」を参照してください。 AWS SDKs</li> <li>• AWS APIs <a href="#">ユーザーガイド</a> 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。</li> </ul>

### 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 custom-labels-access 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='custom-labels-access')
rekognition_client = session.client("rekognition")
```

プロファイルが表すユーザーには、Amazon Rekognition Custom Labels SDK オペレーションおよび例に必要なその他の 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。詳細については、「[SDK アクセス許可の設定](#)」を参照してください。権限を割り当てるには、「[SDK アクセス許可の設定](#)」を参照してください。

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 `custom-labels-access` 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 (によって管理されるユーザー AWS IAM Identity Center) — [を使用するように AWS CLI を設定する手順 AWS IAM Identity Center](#)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。

## SDK アクセス許可の設定

Amazon Rekognition Custom Labels SDK オペレーションを使用するには、Amazon Rekognition Custom Labels API およびモデルトレーニングに使用される Amazon S3 バケットへのアクセス許可が必要です。

### トピック

- [SDK オペレーションアクセス許可の付与](#)
- [AWS SDK を使用するためのポリシーの更新](#)
- [アクセス許可の割り当て](#)

## SDK オペレーションアクセス許可の付与

タスクの実行に必要な権限 (最小特権) のみを付与することをお勧めします。例えば、を呼び出すには[DetectCustomLabels](#)、を実行するためのアクセス許可が必要ですrekognition:DetectCustomLabels。オペレーションの権限を確認するには、「[API リファレンス](#)」をチェックしてください。

アプリケーションを始めたばかりの場合は、必要な特定のアクセス許可がわからない場合があるため、より広範なアクセス許可から始めることができます。AWS マネージドポリシーによって、作業の開始に役立つアクセス許可が提供されます。AmazonRekognitionCustomLabelsFullAccess AWS 管理ポリシーを使用して、Amazon Rekognition Custom Labels API への完全なアクセスを取得できます。詳細については、「[AWS マネージドポリシー : AmazonRekognitionCustomLabelsFullAccess](#)」を参照してください。アプリケーションに必要な権限がわかったら、ユースケースに応じたカスタマー管理ポリシーを定義することで、権限をさらに減らします。詳細については、「[カスタマー管理ポリシー](#)」を参照してください。

アクセス許可を割り当てるには、「[アクセス許可の割り当て](#)」を参照してください。

## AWS SDK を使用するためのポリシーの更新

Amazon Rekognition Custom Labels の最新リリースで AWS SDK を使用するには、トレーニングイメージとテストイメージを含む Amazon S3 バケットにアクセスするためのアクセス許可を Amazon

Rekognition Custom Labels に付与する必要がなくなりました。Amazon S3 以前にアクセス許可を追加したことがある場合は、アクセス許可を削除する必要はありません。選択する場合は、プリンシパルのサービスが `rekognition.amazonaws.com` であるバケットからポリシーをすべて削除してください。例:

```
"Principal": {  
  "Service": "rekognition.amazonaws.com"  
}
```

詳細については、「[バケットポリシーの使用](#)」を参照してください。

## アクセス許可の割り当て

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。

- 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。IAM ユーザーガイドの「[ユーザー \(コンソール\) へのアクセス許可の追加](#)」の指示に従います。

## Amazon Rekognition Custom Labels オペレーションを呼び出す

次のコードを実行して、Amazon Rekognition Custom Labels API を呼び出せることを確認します。このコードは、現在の AWS リージョンの AWS アカウント内のプロジェクトを一覧表示します。まだプロジェクトを作成していない場合、レスポンスは空ですが、DescribeProjects オペレーションを呼び出せることを確認できます。

一般的に、サンプル関数を呼び出すには、AWS SDK Rekognition クライアントとそのほかの必要なパラメータが必要です。AWS SDK クライアントはメイン関数で宣言されます。

コードが失敗する場合は、使用するユーザーに正しいアクセス許可があるかどうかを確認します。また、Amazon Rekognition Custom Labels がすべての AWS リージョンで利用できるわけではないため、使用している AWS リージョンも確認してください。

Amazon Rekognition Custom Labels オペレーションを呼び出すには

1. 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 4: AWS CLI と AWS SDKsを設定する](#)」を参照してください。
2. 次のサンプルコードを使用して、プロジェクトを表示します。

## CLI

describe-projects コマンドを使用して、アカウントのプロジェクトを一覧表示します。

```
aws rekognition describe-projects \  
--profile custom-labels-access
```

## Python

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
  
"""  
This example shows how to describe your Amazon Rekognition Custom Labels  
projects.  
If you haven't previously created a project in the current AWS Region,  
the response is an empty list, but does confirm that you can call an  
Amazon Rekognition Custom Labels operation.  
"""  
from botocore.exceptions import ClientError  
import boto3  
  
def describe_projects(rekognition_client):  
    """  
    Lists information about the projects that are in in your AWS account  
    and in the current AWS Region.  
    """
```



```
: param rekognition_client: A Boto3 Rekognition client.
"""
try:
    response = rekognition_client.describe_projects()
    for project in response["ProjectDescriptions"]:
        print("Status: " + project["Status"])
        print("ARN: " + project["ProjectArn"])
        print()
    print("Done!")
except ClientError as err:
    print(f"Couldn't describe projects. \n{err}")
    raise

def main():
    """
    Entrypoint for script.
    """

    session = boto3.Session(profile_name='custom-labels-access')
    rekognition_client = session.client("rekognition")

    describe_projects(rekognition_client)

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

## Java V2

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
*/

package com.example.rekognition;

import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
```

```
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.DatasetMetadata;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectsRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectsResponse;
import software.amazon.awssdk.services.rekognition.model.ProjectDescription;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

public class Hello {

    public static final Logger logger = Logger.getLogger(Hello.class.getName());

    public static void describeMyProjects(RekognitionClient rekClient) {

        DescribeProjectsRequest descProjects = null;

        // If a single project name is supplied, build projectNames argument
        List<String> projectNames = new ArrayList<String>();

        descProjects = DescribeProjectsRequest.builder().build();

        // Display useful information for each project.

        DescribeProjectsResponse resp =
            rekClient.describeProjects(descProjects);

        for (ProjectDescription projectDescription : resp.projectDescriptions())
        {

            System.out.println("ARN: " + projectDescription.projectArn());
            System.out.println("Status: " +
                projectDescription.statusAsString());
            if (projectDescription.hasDatasets()) {
                for (DatasetMetadata datasetDescription :
                    projectDescription.datasets()) {
                    System.out.println("\tdataset Type: " +
                        datasetDescription.datasetTypeAsString());
                    System.out.println("\tdataset ARN: " +
                        datasetDescription.datasetArn());
                }
            }
        }
    }
}
```

```
        System.out.println("\tdataset Status: " +
datasetDescription.statusAsString());
    }
    }
    System.out.println();
}

}

public static void main(String[] args) {

    try {

        // Get the Rekognition client
        RekognitionClient rekClient = RekognitionClient.builder()
            .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
            .region(Region.US_WEST_2)
            .build();

        // Describe projects

        describeMyProjects(rekClient);

        rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    }

}

}
```

## ステップ 5: (オプション) トレーニングファイルを暗号化する

以下のオプションのいずれかを選択して、コンソールバケットまたは外部の Amazon S3 バケットにある Amazon Rekognition Custom Labels のマニフェストファイルとイメージファイルを暗号化できます。

- Amazon S3 キー (SSE-S3) を使用します。
- を使用します AWS KMS key。

#### Note

呼び出し元の [IAM プリンシパル](#)には、ファイルを復号化するアクセス許可が必要です。詳細については、「[で暗号化されたファイルの復号化 AWS Key Management Service](#)」を参照してください。

Amazon S3 バケットの暗号化の詳細については、「[Amazon S3 バケットのデフォルトのサーバー側の暗号化動作の設定](#)」を参照してください。

## で暗号化されたファイルの復号化 AWS Key Management Service

AWS Key Management Service (KMS) を使用して Amazon Rekognition Custom Labels マニフェストファイルとイメージファイルを暗号化する場合は、Amazon Rekognition Custom Labels を呼び出す IAM プリンシパルを KMS キーのキーポリシーに追加します。これにより、Amazon Rekognition Custom Labels はトレーニングの前にマニフェストとイメージファイルを復号できます。詳細については、「[Amazon S3 バケットには、カスタム AWS KMS キーを使用したデフォルトの暗号化があります](#)」を参照してください。ユーザーがバケットからダウンロードしたり、バケットにアップロードしたりできるようにする方法を教えてください。

IAM プリンシパルには、KMS キーに対する次のアクセス許可が必要です。

- kms:GenerateDataKey
- kms:Decrypt

詳細については、「[AWS Key Management Service \(SSE-KMS\) に保存された KMS キーを使用したサーバー側暗号化を使用したデータの保護](#)」を参照してください。

## コピーしたトレーニングイメージとテストイメージを暗号化する

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。デフォルトでは、コピーされたイメージは、AWS が所有および管理するキーで保管時に暗号化されます。独自の AWS KMS key の使用を選択することもできます。独自の KMS キーを使用する場合は、KMS キーに対する次のアクセス許可が必要です。

- kms:CreateGrant
- kms:DescribeKey

コンソールでモデルをトレーニングする場合、または `CreateProjectVersion` オペレーションを呼び出す場合に KMS キーを必要に応じて指定できます。使用する KMS キーは、Amazon S3 バケット内のマニフェストファイルとイメージファイルの暗号化に使用する KMS キーと同一である必要はありません。詳細については、「[ステップ 5: \(オプション\) トレーニングファイルを暗号化する](#)」を参照してください。

詳細については、「[AWS Key Management Service の概念](#)」を参照してください。ソース画像には影響がありません。

モデルをトレーニングする方法については、「[Amazon Rekognition Custom Labels モデルをトレーニングする](#)」を参照してください。

## ステップ 6: (オプション) 以前のデータセットを新しいプロジェクトに関連付ける

Amazon Rekognition Custom Labels がプロジェクトでデータセットを管理するようになりました。以前に作成した (以前の) データセットは読み取り専用で、使用する前にプロジェクトに関連付ける必要があります。コンソールでプロジェクトの詳細ページを開くと、プロジェクトのモデルの最新バージョンをトレーニングしたデータセットがプロジェクトと自動的に関連付けられます。AWS SDK を使用している場合、データセットとプロジェクトの自動関連付けは行われません。

関連付けられていない以前のデータセットは、モデルのトレーニングに使用されたことはありませんが、以前のバージョンのモデルのトレーニングに使用されました。以前のデータセットページには、関連するデータセットと関連付けられていないデータセットがすべて表示されます。

関連付けられていない以前のデータセットを使用するには、以前のデータセットページで新しいプロジェクトを作成します。このデータセットは、新しいプロジェクトのトレーニングデータセットになります。以前のデータセットには複数の関連付けがある場合があるため、既に関連付けられているデータセットのプロジェクトを作成することもできます。

以前のデータセットを新しいプロジェクトに関連付けるには

1. Amazon Rekognition コンソールを <https://console.aws.amazon.com/rekognition/> で開きます。
2. 左側のペインで、[カスタムラベルを使用] を選択します。Amazon Rekognition Custom Labels のランディングページが表示されます。

3. 左のナビゲーションペインで、[データセット] を選択します。
4. データセットビューで、プロジェクトに関連付ける以前のデータセットを選択します。
5. [データセットを使用してプロジェクトを作成] を選択します。
6. [プロジェクトを作成する] ウィンドウの [プロジェクト名] に、新しいプロジェクトの名前を入力します。
7. [プロジェクトを作成する] を選択してプロジェクトを作成します。プロジェクトの作成までに時間がかかる場合があります。
8. プロジェクトを使用してください。詳細については、「[Amazon Rekognition Custom Labels について](#)」を参照してください。

## 以前のデータセットをテストデータセットとして使用する

以前のデータセットを既存のプロジェクトのテストデータセットとして使用するには、まず以前のデータセットを新しいプロジェクトに関連付けます。次に、新しいプロジェクトのトレーニングデータセットを既存のプロジェクトのテストデータセットにコピーします。

以前のデータセットをテストデータセットとして使用するには

1. [ステップ 6: \(オプション\) 以前のデータセットを新しいプロジェクトに関連付ける](#) で説明されている手順に従って、以前のデータセットを新しいプロジェクトに関連付けます。
2. 新しいプロジェクトからトレーニングデータセットをコピーして、既存のプロジェクトにテストデータセットを作成します。詳細については、「[既存のデータセット](#)」を参照してください。
3. [Amazon Rekognition Custom Labels プロジェクトの削除 \(コンソール\)](#) に記載されている手順に従って、新しいプロジェクトを削除します。

以前のデータセットのマニフェストファイルを使用してテストデータセットを作成することもできます。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

# Amazon Rekognition Custom Labels について

このセクションでは、コンソールと AWS SDK で Amazon Rekognition Custom Labels モデルをトレーニングして使用するワークフローの概要を説明します。

## Note

Amazon Rekognition Custom Labels がプロジェクト内のデータセットを管理するようになりました。コンソールと AWS SDK を使用して、プロジェクトのデータセットを作成できます。以前に Amazon Rekognition Custom Labels を使用したことがある場合は、古いデータセットを新しいプロジェクトに関連付ける必要がある場合があります。詳細については、「[ステップ 6: \(オプション\) 以前のデータセットを新しいプロジェクトに関連付ける](#)」を参照してください。

## トピック

- [モデルタイプの決定](#)
- [モデルを作成する](#)
- [モデルの改善](#)
- [モデルの開始](#)
- [イメージの分析](#)
- [モデルの停止](#)

## モデルタイプの決定

まず、ビジネス目標に応じて、どのタイプのモデルをトレーニングするかを決定します。例えば、ソーシャルメディアの投稿で自社のロゴを見つけたり、店舗の棚で商品を識別したり、組立ラインで機械部品を分類したりするようにモデルをトレーニングできます。

Amazon Rekognition Custom Labels では、以下のタイプのモデルをトレーニングできます。

- [オブジェクト、シーン、概念を検出する](#)
- [オブジェクトの位置の検索](#)
- [ブランドの所在地を探す](#)

トレーニングするモデルのタイプを決定しやすくするために、Amazon Rekognition Custom Labels には使用できるサンプルプロジェクトが用意されています。詳細については、「[Amazon Rekognition Custom Labels の開始方法](#)」を参照してください。

## オブジェクト、シーン、概念を検出する

このモデルは、イメージ全体に関連するオブジェクト、シーン、概念の分類を予測します。例えば、イメージに観光名所が含まれているかどうかを判断するモデルをトレーニングできます。サンプルプロジェクトについては、「[画像分類](#)」を参照してください。次のレイクのイメージは、オブジェクト、シーン、概念を認識できるイメージの種類の例です。

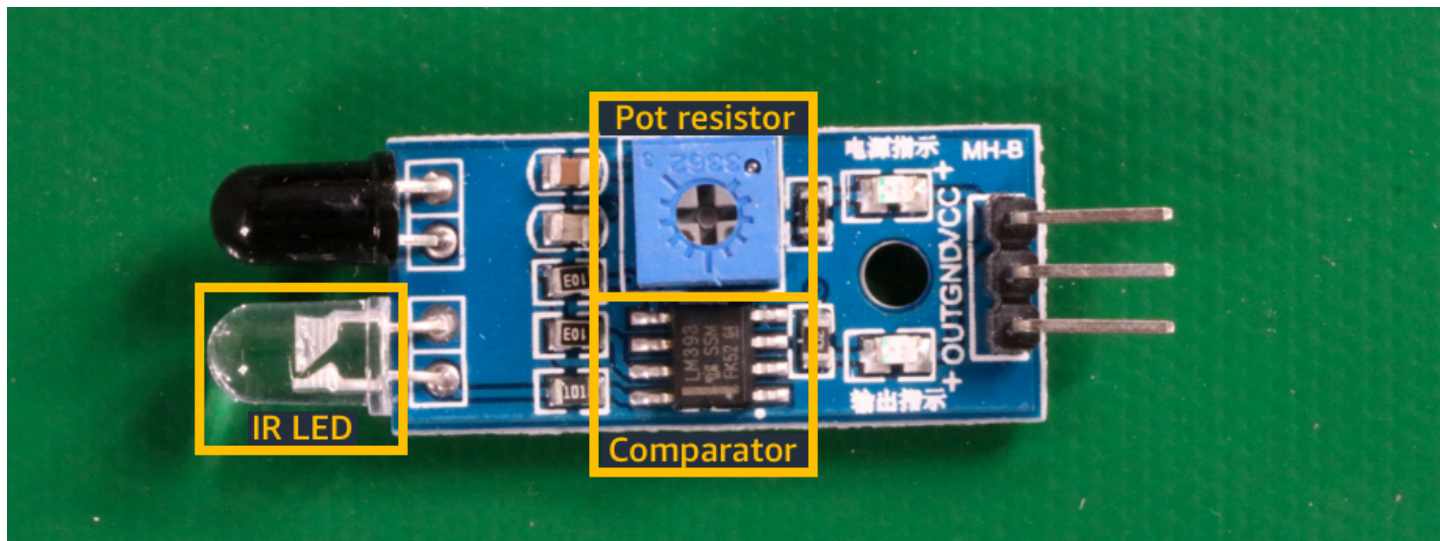


イメージを複数のカテゴリに分類するモデルをトレーニングすることもできます。例えば、前のイメージには、空の色、反射、湖などのカテゴリが含まれている場合があります。サンプルプロジェクトについては、「[マルチラベルイメージ分類](#)」を参照してください。



## オブジェクトの位置の検索

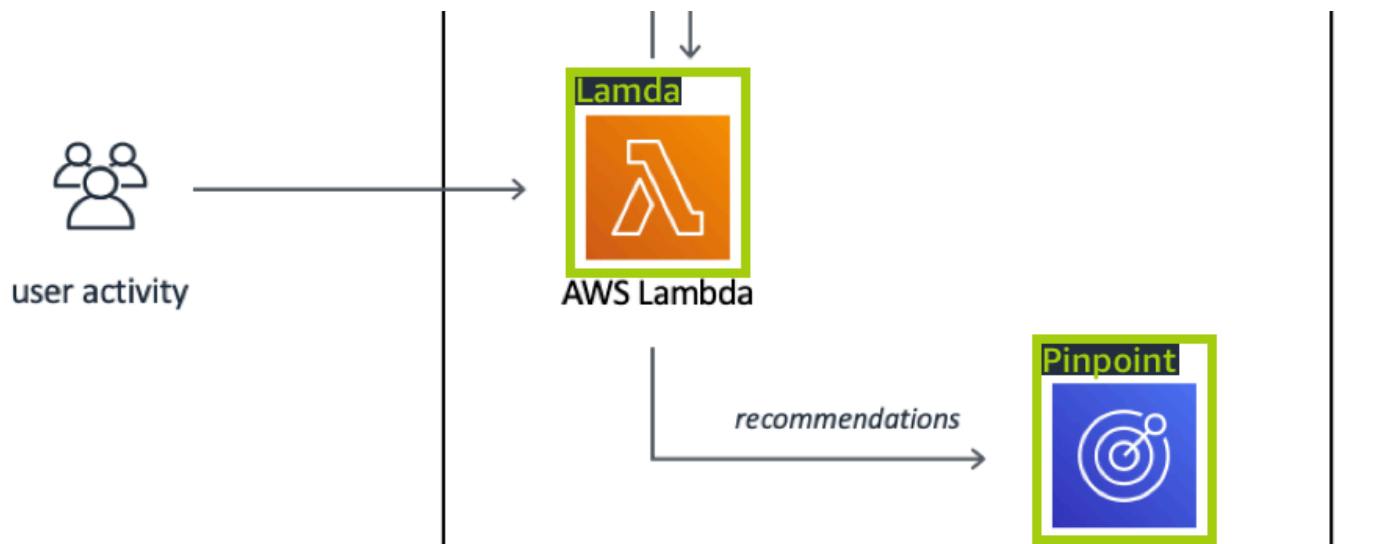
モデルはイメージ上のオブジェクトの位置を予測します。予測には、オブジェクトの位置に関する境界ボックス情報と、境界ボックス内のオブジェクトを識別するラベルが含まれます。例えば、次のイメージは、コンパレータやポット抵抗など、回路基板のさまざまな部分を囲む境界ボックスを示しています。



[オブジェクトのローカリゼーション](#) のサンプルプロジェクトでは、Amazon Rekognition Custom Labels がラベル付き境界ボックスを使用して、オブジェクトの位置を検出するモデルをトレーニングする方法を示しています。

## ブランドの所在地を探す

Amazon Rekognition Custom Labels は、イメージ上のブランドの場所 (ロゴなど) を検索するようにモデルをトレーニングできます。予測には、ブランドローケーションの境界ボックス情報と、境界ボックス内のオブジェクトを識別するラベルが含まれます。サンプルプロジェクトについては、「[ブランド検出](#)」を参照してください。次の画像は、モデルが検出できるいくつかのブランドの例です。



## モデルを作成する

モデルを作成する手順は、プロジェクトの作成、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、モデルのトレーニングです。

### プロジェクトを作成する

プロジェクトとは、Amazon Rekognition Custom Labels モデルのバージョンを作成および管理するために必要なリソースのグループです。プロジェクトでは、次のものが管理されます。

- データセット - モデルのトレーニングに使用されるイメージとイメージラベル。プロジェクトには、トレーニングデータセットとテストデータセットがあります。
- モデル - ビジネス特有の概念、シーン、オブジェクトを検索するためのトレーニングを行うソフトウェアです。プロジェクトには、モデルの複数のバージョンを含めることができます。

回路基板上の回路基板部品を検索するなど、単一のユースケースにプロジェクトを使用することをお勧めします。

Amazon Rekognition Custom Labels コンソールと [CreateProject](#) API を使用してプロジェクトを作成できます。詳細については、「[プロジェクトの作成](#)」を参照してください。

## 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

データセットとは、イメージとそのイメージを説明するラベルの集合のことです。プロジェクト内で、Amazon Rekognition Custom Labels でモデルのトレーニングとテストに使用するトレーニングデータセットとテストデータセットを作成します。

ラベルは、イメージ内のオブジェクトを囲むオブジェクト、シーン、概念、または境界ボックスを識別します。ラベルはイメージ全体 (イメージレベル) に割り当てられるか、イメージ上のオブジェクトを囲む境界ボックスに割り当てられます。

### Important

データセット内のイメージに付けるラベルによって、Amazon Rekognition Custom Labels が作成するモデルのタイプが決まります。例えば、オブジェクト、シーン、概念を検出するモデルをトレーニングするには、トレーニングデータセットとテストデータセットのイメージにイメージレベルのラベルを割り当てます。詳細については、「[データセットの目的の設定](#)」を参照してください。

イメージは PNG 形式および JPEG 形式である必要があり、入力イメージの推奨事項に従う必要があります。詳細については、「[イメージの準備](#)」を参照してください。

## 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 (コンソール)

1 つのデータセットでプロジェクトを開始することも、個別のトレーニングデータセットとテストデータセットを持つプロジェクトから始めることもできます。1 つのデータセットから始めると、Amazon Rekognition Custom Labels はトレーニング中にデータセットを分割して、プロジェクトのトレーニングデータセット (80%) とテストデータセット (20%) を作成します。Amazon Rekognition Custom Labels にトレーニングとテストに使用するイメージを決定させる場合は、1 つのデータセットから始めてください。トレーニング、テスト、パフォーマンスのチューニングを完全に制御するには、トレーニングデータセットとテストデータセットを分けてプロジェクトを開始することをお勧めします。

プロジェクトのデータセットを作成するには、次のいずれかの方法でイメージをインポートします。

- ローカルコンピュータから画像をインポートします。
- S3 バケットから画像をインポートします。Amazon Rekognition Custom Labels では、イメージを含むフォルダ名を使用してイメージにラベル付けすることができます。

- Amazon SageMaker Ground Truth マニフェストファイルをインポートします。
- 既存の Amazon Rekognition Custom Labels データセットをコピーします。

詳細については、「[イメージ付きの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](#)」を参照してください。

イメージのインポート元によっては、イメージにラベルが付いていない場合があります。例えば、ローカルコンピュータからインポートされたイメージにはラベルは付きません。Amazon SageMaker Ground Truth マニフェストファイルからインポートされたイメージにはラベルが付けられます。Amazon Rekognition Custom Labels コンソールを使用して、ラベルの追加、変更、割り当てを行うことができます。詳細については、「[イメージにラベルを付ける](#)」を参照してください。

コンソールでトレーニングデータセットとテストデータセットを作成するには、「[イメージ付きの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](#)」を参照してください。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を含むチュートリアルについては、「[チュートリアル: イメージの分類](#)」を参照してください。

## 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 (SDK)

トレーニングデータセットとテストデータセットを作成するには、CreateDataset API を使用します。Amazon Sagemaker 形式のマニフェストファイルを使用するか、既存の Amazon Rekognition Custom Labels をコピーして、データセットを作成できます。詳細については、「[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 \(SDK\)](#)」を参照してください。必要に応じて、独自のマニフェストファイルを作成できます。詳細については、「[the section called “マニフェストファイルの作成”](#)」を参照してください。

## モデルをトレーニングする

トレーニングデータセットでモデルをトレーニングします。モデルの新しいバージョンは、トレーニングのたびに作成されます。トレーニング中、Amazon Rekognition Custom Labels はトレーニング済みモデルのパフォーマンスをテストします。その結果を使用して、モデルを評価し、改善することができます。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。モデルのトレーニングが成功した場合にのみ課金されます。詳細については、「[Amazon Rekognition Custom Labels モデルをトレーニングする](#)」を参照してください。モデルトレーニングが失敗した場合、Amazon Rekognition Custom Labels は使用できるデバッグ情報を提供します。詳細については、「[失敗したモデルトレーニングのデバッグ](#)」を参照してください。

## モデルのトレーニング (コンソール)

コンソールでモデルをトレーニングする方法については、「[モデルのトレーニング \(コンソール\)](#)」を参照してください。

## モデルのトレーニング (SDK)

Amazon Rekognition Custom Labels モデルをトレーニングするには、[CreateProjectバージョン](#) を呼び出します。詳細については、「[モデルのトレーニング \(SDK\)](#)」を参照してください。

## モデルの改善

テスト中、Amazon Rekognition Custom Labels は、トレーニング済みモデルの改善に使用できる評価メトリクスを作成します。

## モデルの評価

テスト中に作成されたパフォーマンスメトリクスを使用して、モデルのパフォーマンスを評価します。F1、適合率、再現率などのパフォーマンスメトリクスにより、トレーニングしたモデルのパフォーマンスを理解し、本稼働で使用する準備ができたかどうかを判断することができます。詳細については、「[モデルを評価するためのメトリクス](#)」を参照してください。

## モデルを評価する (コンソール)

パフォーマンスメトリクスを表示するには、「[評価メトリクスへのアクセス \(コンソール\)](#)」を参照してください。

## モデルの評価 (SDK)

パフォーマンスメトリクスを取得するには、[DescribeProjectバージョン](#) を呼び出してテスト結果を取得します。詳細については、「[Amazon Rekognition Custom Labels の評価メトリクス \(SDK\) へのアクセス](#)」を参照してください。テスト結果には、分類結果の混同行列など、コンソールにはないメトリクスが含まれます。テスト結果は次の形式で返されます。

- F1 スコア - モデルの全体的な適合率と再現率を表す単一の値。詳細については、「[F1](#)」を参照してください。
- サマリーファイルの場所 - テストサマリーには、テストデータセット全体の集計評価メトリクスと、個々のラベルのメトリクスが含まれます。DescribeProjectVersions は、サマリーファ

イルの S3 バケットとフォルダの場所を返します。詳細については、「[概要ファイル](#)」を参照してください。

- 評価マニフェストのスナップショットの場所 - スナップショットには、信頼度評価や誤検出などの二項分類テストの結果など、テスト結果に関する詳細が含まれています。DescribeProjectVersions は、スナップショットファイルの S3 バケットとフォルダの場所を返します。詳細については、「[評価マニフェストのスナップショット](#)」を参照してください。

## モデルの改善

改善が必要な場合は、トレーニングイメージを追加するか、データセットのラベル付けを改善することができます。詳細については、「[Amazon Rekognition Custom Labels モデルの改善](#)」を参照してください。モデルが作成した予測についてフィードバックを与えて、それを使用してモデルを改善することもできます。詳細については、「[モデルフィードバックソリューション](#)」を参照してください。

### モデルの改善 (コンソール)

データセットにイメージを追加する方法については、「[データセットへのイメージの追加](#)」を参照してください。ラベルを追加または変更するには、「[the section called “イメージにラベルを付ける”](#)」を参照してください。

モデルを再トレーニングするには、「[モデルのトレーニング \(コンソール\)](#)」を参照してください。

### モデルの改善 (SDK)

データセットにイメージを追加したり、イメージのラベルを変更するには、UpdateDatasetEntries API を使用します。UpdateDatasetEntries は JSON 行を更新するか、マニフェストファイルに追加します。JSON の各行には、割り当てられたラベルや境界ボックスの情報など、1つのイメージに関する情報が含まれています。詳細については、「[イメージの追加 \(SDK\)](#)」を参照してください。データセット内のエントリを表示するには、ListDatasetEntries API を使用します。

モデルを再トレーニングするには、「[モデルのトレーニング \(SDK\)](#)」を参照してください。

## モデルの開始

モデルを使用する前に、Amazon Rekognition Custom Labels コンソールまたは StartProjectVersion API を使用してモデルを開始します。モデルの稼働時間に応じて課金されます。詳細については、「[トレーニング済みモデルの実行](#)」を参照してください。



## モデルの開始 (コンソール)

コンソールを使用してモデルを開始する方法については、「[Amazon Rekognition Custom Labels モデルの開始 \(コンソール\)](#)」を参照してください。

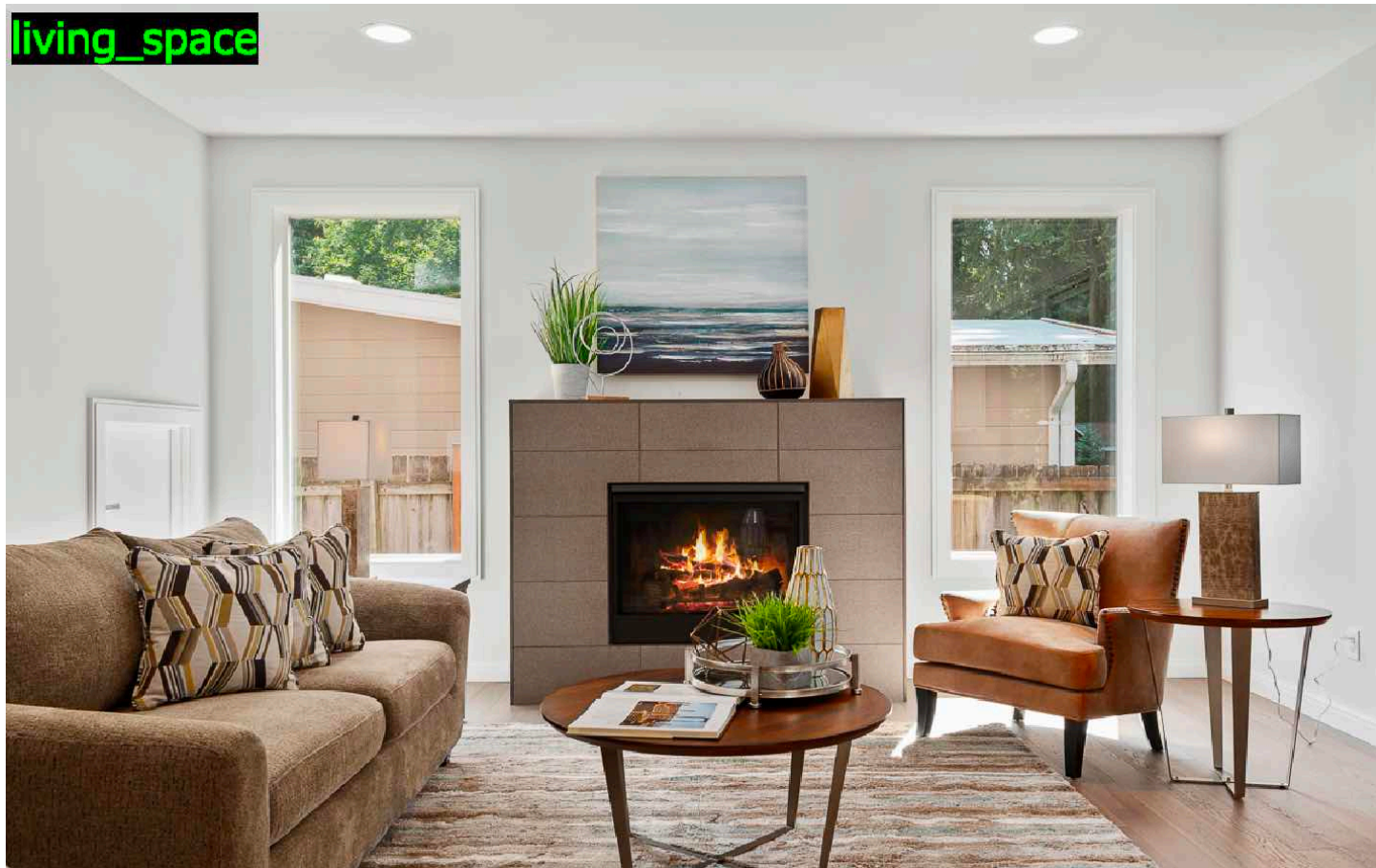
## モデルの開始

`StartProject/バージョン` を呼び出すモデルを開始します。詳細については、「[Amazon Rekognition Custom Labels モデル \(SDK\) を開始します。](#)」を参照してください。

## イメージの分析

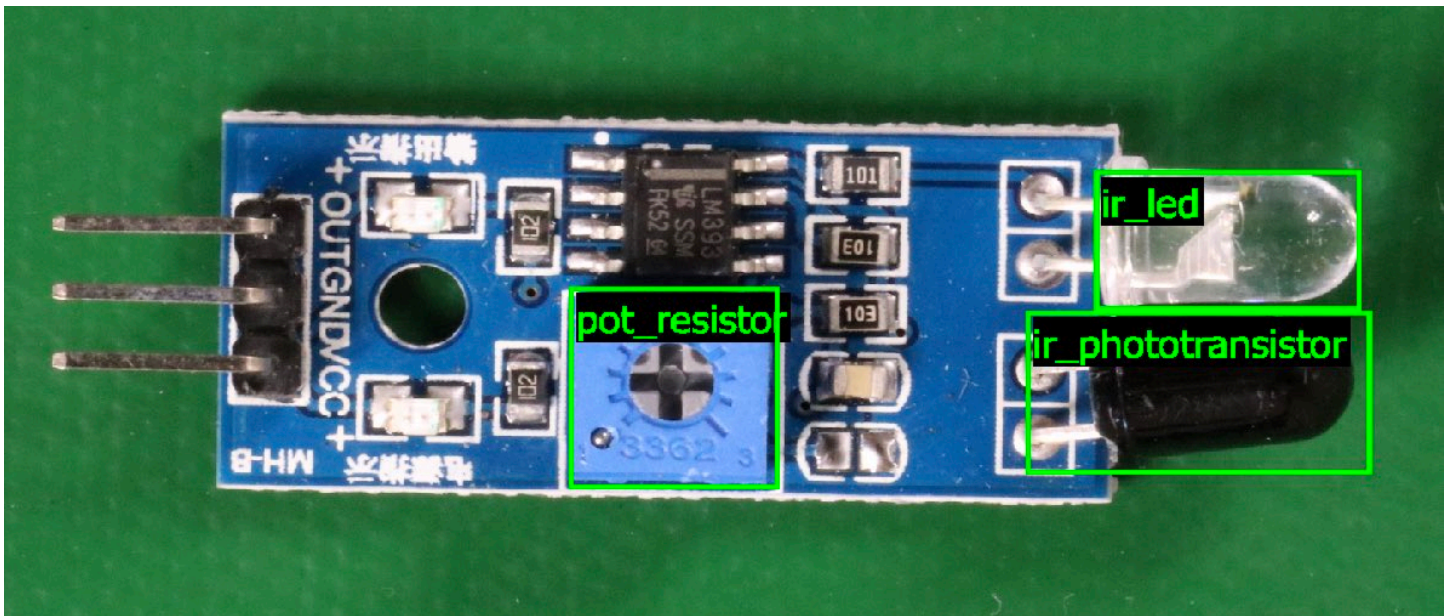
モデルを使用してイメージを分析するには、`DetectCustomLabels` API を使用します。ローカルイメージ、または S3 バケットに保存されているイメージを指定できます。オペレーションには、使用するモデルの Amazon リソースネーム (ARN) も必要です。

モデルでオブジェクト、シーン、概念が見つかった場合、レスポンスにはイメージに含まれるイメージレベルのラベルのリストが含まれます。例えば、次のイメージは、Rooms サンプルプロジェクトで見つかったイメージレベルのラベルを示しています。

**living\_space**

モデルがオブジェクトの位置を検出した場合、レスポンスにはイメージ内のラベル付き境界ボックスのリストが含まれます。境界ボックスは、イメージ上のオブジェクトの位置を表します。境界ボックスの情報を使用して、オブジェクトの周囲に境界ボックスを描画できます。例えば、次のイメージは、回路基板サンプルプロジェクトを使用して見つかった回路基板部品の周囲の境界ボックスを示しています。





詳細については、「[トレーニングされたモデルによるイメージの分析](#)」を参照してください。

## モデルの停止

モデルの稼働時間に応じて課金されます。モデルを使用しなくなった場合は、Amazon Rekognition Custom Labels コンソールまたは StopProjectVersion API を使用してモデルを停止します。詳細については、「[Amazon Rekognition Custom Labels モデルの停止](#)」を参照してください。

### モデルの停止する (コンソール)

コンソールを使用して実行中のモデルを停止するには、「[Amazon Rekognition Custom Labels モデルの停止 \(コンソール\)](#)」を参照してください。

### モデルの停止 (SDK)

実行中のモデルを停止するには、[StopProjectバージョン](#) を呼び出します。詳細については、「[Amazon Rekognition Custom Labels モデル \(SDK\) の停止](#)」を参照してください。

# Amazon Rekognition Custom Labels の開始方法

これらの「はじめに」の手順を開始する前に、「[Amazon Rekognition Custom Labels について](#)」を読むことをお勧めします。

Amazon Rekognition Custom Labels を使用して、機械学習モデルをトレーニングできます。トレーニング済みモデルは、ビジネスニーズ特有のオブジェクト、シーン、コンセプトを検索するためのイメージを分析します。例えば、住宅のイメージを分類したり、プリント基板上の電子部品の位置を見つけたりするようにモデルをトレーニングできます。

Amazon Rekognition Custom Labels の使用を開始していただけるように、チュートリアルビデオとサンプルプロジェクトが含まれています。

## Note

Amazon Rekognition Custom Labels がサポートする AWS リージョンとエンドポイントの詳細については、「[Rekognition エンドポイントとクォータ](#)」を参照してください。

## チュートリアルビデオ

ビデオでは、Amazon Rekognition Custom Labels を使用してモデルをトレーニングし使用する方法を紹介しています。

チュートリアルビデオを見るには

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、[カスタムラベルを使用] を選択します。Amazon Rekognition Custom Labels のランディングページが表示されます。[カスタムラベルを使用] が表示されない場合は、使用している [AWS リージョン](#) が Amazon Rekognition Custom Labels をサポートしていることを確認してください。
3. ナビゲーションペインで、[開始方法] を選択します。
4. [Amazon Rekognition Custom Labels とは] で、ビデオを選択して概要ビデオを視聴します。
5. ナビゲーションペインで、[チュートリアル] を選択します。
6. [チュートリアル] ページで、視聴するチュートリアルビデオを選択します。

## プロジェクトの例

Amazon Rekognition Custom Labels には次のサンプルプロジェクトが用意されています。

### 画像分類

イメージ分類プロジェクト (ルーム) は、家の中の 1 つ以上の場所 (裏庭、キッチン、パティオなど) を検出するモデルをトレーニングします。トレーニングイメージとテストイメージは 1 つの場所を表します。各イメージには、キッチン、パティオ、または living\_space など、1 つのイメージレベルのラベルが付けられます。分析済みのイメージでは、トレーニングされたモデルは、トレーニングに使用されたイメージレベルのラベルセットから 1 つ以上の一致するラベルを返します。例えば、モデルは次のイメージ内で living\_space というラベルを検出する場合があります。詳細については、「[オブジェクト、シーン、概念を検出する](#)」を参照してください。



### マルチラベルイメージ分類

マルチラベルイメージ分類プロジェクト (フラワー) は、花のイメージを 3 つの概念 (花の種類、葉の有無、成長段階) に分類するモデルをトレーニングします。

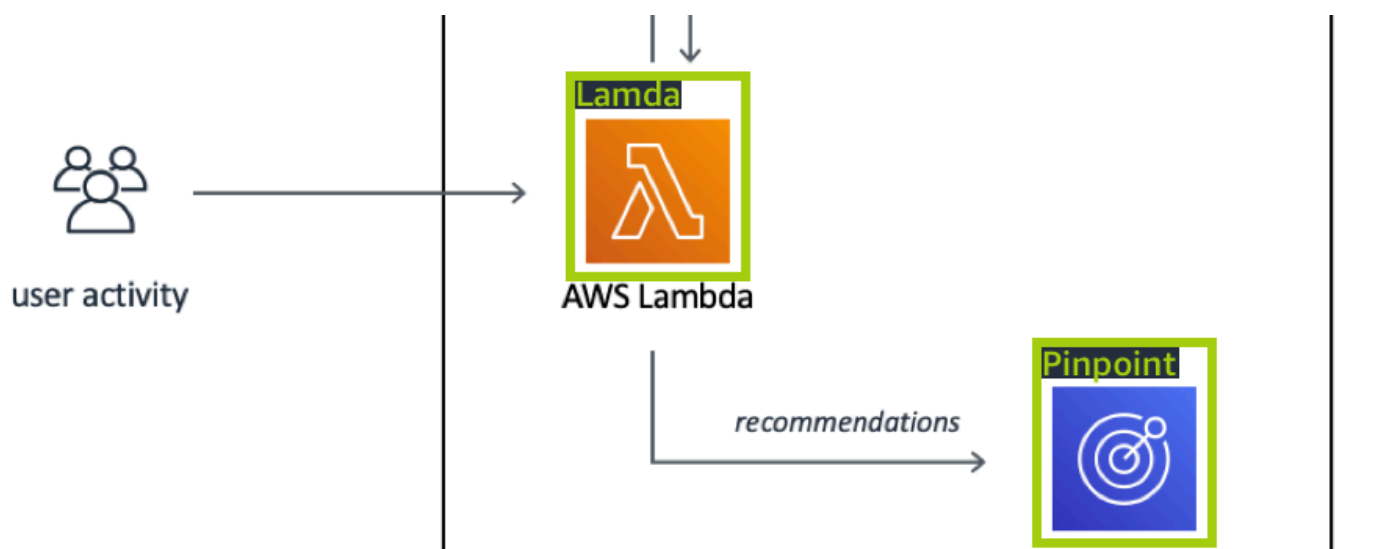
トレーニングイメージとテストイメージには、コンセプトごとにイメージレベルのラベルが付けられています。例えば、花の種類にカメラリア、葉のある花に with\_leaves、完全に成長した花に fully\_grown などのラベルが付けられています。

分析済みのイメージでは、トレーニングされたモデルは、トレーニングに使用されたイメージレベルのラベルセットから一致するラベルを返します。例えば、モデルは次のイメージに対して mediterranean\_spurge と with\_leaves というラベルを返します。詳細については、「[オブジェクト、シーン、概念を検出する](#)」を参照してください。



## ブランド検出

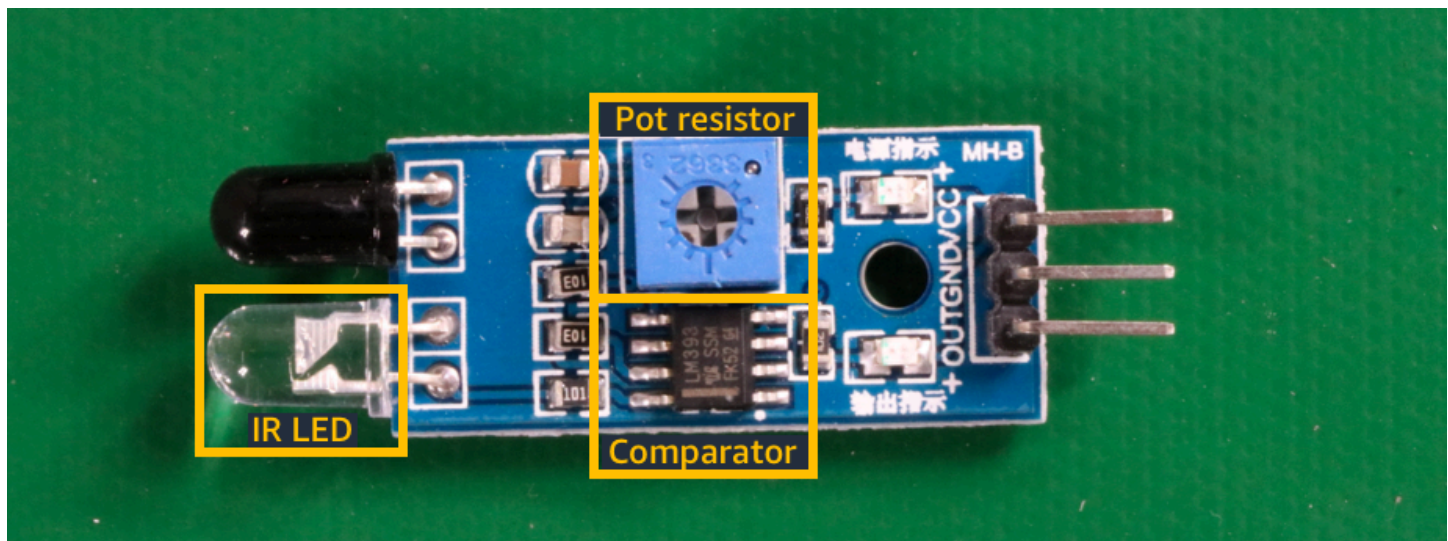
ブランド検出プロジェクト (Logos) は、Amazon Textract や AWS Lambda などの特定の AWS ロゴの位置をモデルが検出するモデルをトレーニングします。トレーニングイメージはロゴのみで、Lambda や textract など、1つのイメージのレベルラベルが付いています。また、ブランドロケーションの境界ボックスが付いたトレーニングイメージを使ってブランド検出モデルをトレーニングすることもできます。テストイメージには、建築図などの自然な場所にあるロゴの位置を表すラベルが付けた境界ボックスがあります。トレーニング済みのモデルはロゴを見つけて、見つかったロゴごとにラベル付きの境界ボックスを返します。詳細については、「[ブランドの位置の検索](#)」を参照してください。



## オブジェクトのローカリゼーション

オブジェクトのローカリゼーションプロジェクト (回路基板) は、コンパレータまたは赤外線赤色発光ダイオードなどの、プリント回路基板上の部品の位置を検索するモデルをトレーニングします。トレーニングイメージとテストイメージには、回路基板の部品を囲む境界ボックスと、境界ボックス内の部品を識別するラベルが含まれます。次の画像例では、ラベル名は ir\_Phototransistor、ir\_led、pot\_resistor、およびコンパレータです。トレーニング済みモデルは回路基板の部品を見つけ、見つかった回路部品ごとにラベル付きの境界を返します。詳細については、「[オブジェクトの位置の検索](#)」を参照してください。





## サンプルプロジェクトの使用

これらのスタートガイドでは、Amazon Rekognition Custom Labels が作成するプロジェクトの例を使用してモデルをトレーニングする方法を示しています。また、モデルを起動し、それを使用してイメージを分析する方法についても説明します。

## プロジェクト例の作成

開始するには、使用するプロジェクトを決めてください。詳細については、「[ステップ 1: プロジェクト例を選択する](#)」を参照してください。

Amazon Rekognition Custom Labels は、データセットを使用してモデルのトレーニングと評価 (テスト) を行います。データセットは、イメージと、イメージの内容を識別するラベルを管理します。サンプルプロジェクトには、トレーニングデータセットと、すべてのイメージにラベルが付けられたテストデータセットが含まれています。モデルをトレーニングする前に変更を加える必要はありません。サンプルプロジェクトでは、Amazon Rekognition Custom Labels がラベルを使用してさまざまなタイプのモデルをトレーニングする 2 つの方法を示しています。

- **イメージレベル** - ラベルはイメージ全体を表す、オブジェクト、シーン、またはコンセプトを識別します。
- **境界ボックス** - ラベルは境界ボックスの内容を識別します。境界ボックスは、イメージ内のオブジェクトを囲むイメージ座標のセットです。

後で独自のイメージを使用してプロジェクトを作成する場合、トレーニングデータセットとテストデータセットを作成して、イメージにラベルを付ける必要があります。詳細については、「[モデルタイプの決定](#)」を参照してください。

## モデルのトレーニング

Amazon Rekognition Custom Labels がプロジェクト例を作成したら、モデルをトレーニングできます。詳細については、「[ステップ 2: モデルのトレーニング](#)」を参照してください。トレーニングが終了したら、通常はモデルのパフォーマンスを評価します。データセット例のイメージは既に高性能モデルを作成済みなので、モデルを実行する前にモデルを評価する必要はありません。詳細については、「[トレーニング済み Amazon Rekognition Custom Labels の改善](#)」を参照してください。

## データモデルの使用

次に、モデルを起動します。詳細については、「[ステップ 3: モデルをスタートする](#)」を参照してください。

モデルの実行を開始したら、そのモデルを使用して新しいイメージを分析することができます。詳細については、「[ステップ 4: モデルを使用してイメージを分析する](#)」を参照してください。

モデルの稼働時間に応じて課金されます。モデル例を使い終わったら、モデルを停止する必要があります。詳細については、「[ステップ 5: モデルを停止する](#)」を参照してください。

## 次のステップ

準備が整ったら、独自のプロジェクトを作成できます。詳細については、「[ステップ 6: 次のステップ](#)」を参照してください。

## ステップ 1: プロジェクト例を選択する

このステップでは、「プロジェクト例を選択する」を使用します。次に、Amazon Rekognition Custom Labels によってプロジェクトとデータセットが作成されます。プロジェクトはモデルのトレーニングに使用されるファイルを管理します。詳細については、「[Amazon Rekognition Custom Labels プロジェクトの管理](#)」を参照してください。データセットには、モデルのトレーニングとテストに使用するイメージと、割り当てられたラベル、境界ボックスが含まれています。詳細については、「[the section called “データセットの管理”](#)」を参照してください。

サンプルプロジェクトの詳細については、「[プロジェクトの例](#)」を参照してください。

## プロジェクト例を選択する

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、[カスタムラベルを使用] を選択します。Amazon Rekognition Custom Labels のランディングページが表示されます。[カスタムラベルを使用] が表示されない場合は、使用している [AWS リージョン](#) が Amazon Rekognition Custom Labels をサポートしていることを確認してください。
3. [開始する] を選択します。

Amazon Rekognition Custom Labels セクションには、開始方法、「サンプルプロジェクト」が強調表示されたチュートリアル、プロジェクト、データセットが表示されます。

## Amazon Rekognition Custom Labels ×

### ▼ Get started

#### Tutorials

#### Example projects

#### Projects

#### Datasets

4. [サンプルプロジェクトを探索する] で [サンプルプロジェクトを試す] を選択します。
5. 使用するプロジェクトを決め、例セクション内の [プロジェクト「#####」を作成] を選択します。次に、Amazon Rekognition Custom Labels によってプロジェクト例が作成されます。

#### i Note

現在の AWS リージョンでコンソールを初めて開いた場合は、初回セットアップダイアログボックスが表示されます。以下の操作を実行します。

1. 表示されている Amazon S3 バケットの名前を記録します。

2. Amazon Rekognition Custom Labels に Amazon S3 バケット (コンソールバケット) の作成を代行させる場合は、[続行] を選択します。以下のコンソールのイメージは、イメージ分類 (部屋)、マルチラベル分類 (花)、ブランド検出 (ロゴ)、オブジェクトローカリゼーション (回路基板) の「プロジェクトの作成」ボタンの例を示しています。

### Image Classification

Recommended for content categorization



Classify images as belonging to a set of predefined labels. For example, real estate companies can use Amazon Rekognition Custom Labels to categorize their images of living rooms, backyards, bedrooms, and other household locations.

Create project "Rooms"

### Multi-label classification

Recommended for inventory management

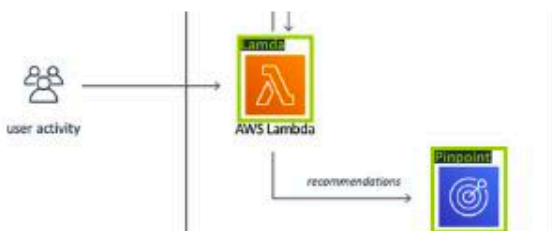


Classify images into multiple categories, such as the color, size, texture, and type of a flower. For example, plant growers can use Amazon Rekognition Custom Labels to distinguish between different types of flowers and if they are healthy, damaged, or infected.

Create project "Flowers"

### Brand detection

Recommended for retail, media networks, and advertising

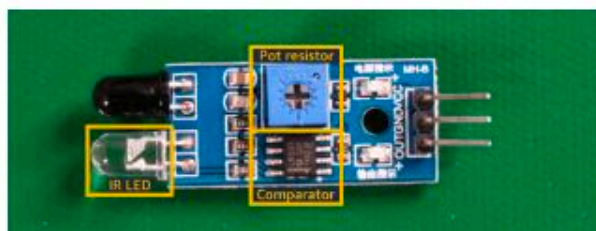


Use brand detection to find the location of commercial brands in images. For example, to report on advertiser coverage, media networks can use Amazon Rekognition Custom Labels to report on the location of sponsor logos in photographs.

Create project "Logos"

### Object localization

Recommended for manufacturing and production chains

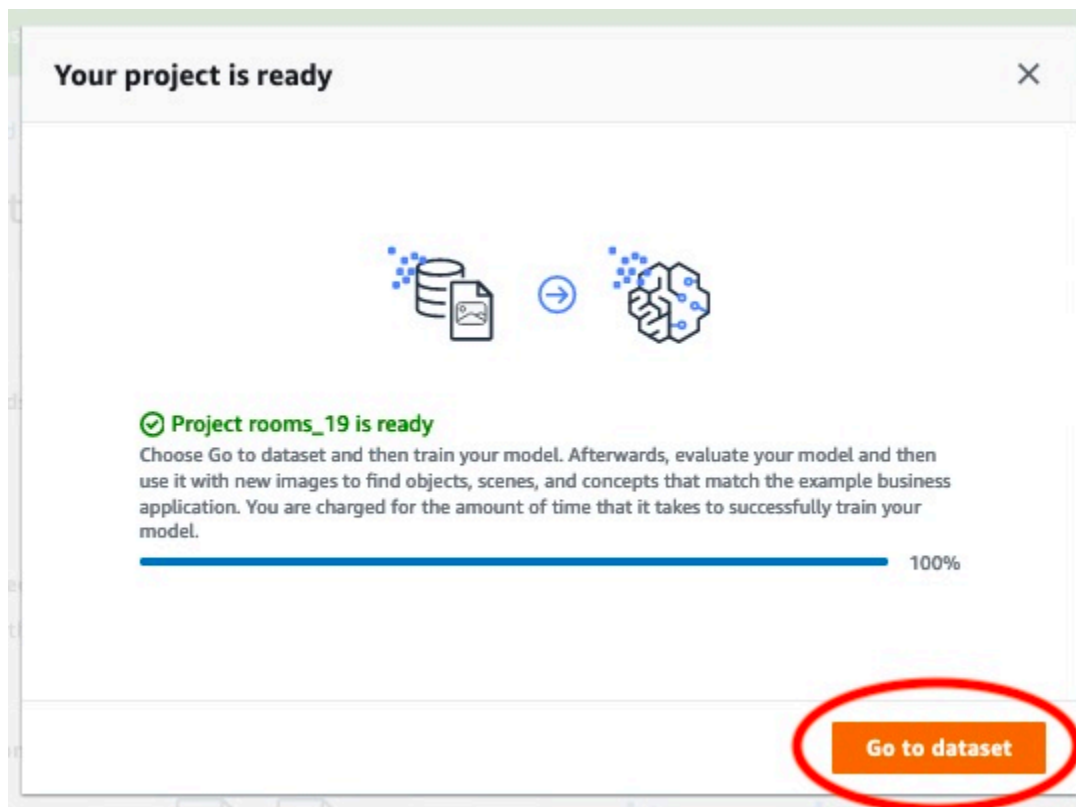


Use object localization to locate parts used in production or manufacturing lines. For example, in the electronics industry, Amazon Rekognition Custom Labels can help count the number of capacitors on a circuit board.

Create project "Circuit boards"



6. プロジェクトの準備ができたら、[データセットに移動] を選択します。次の図は、プロジェクトの準備ができたとときにプロジェクトパネルがどのように表示されるかを示しています。

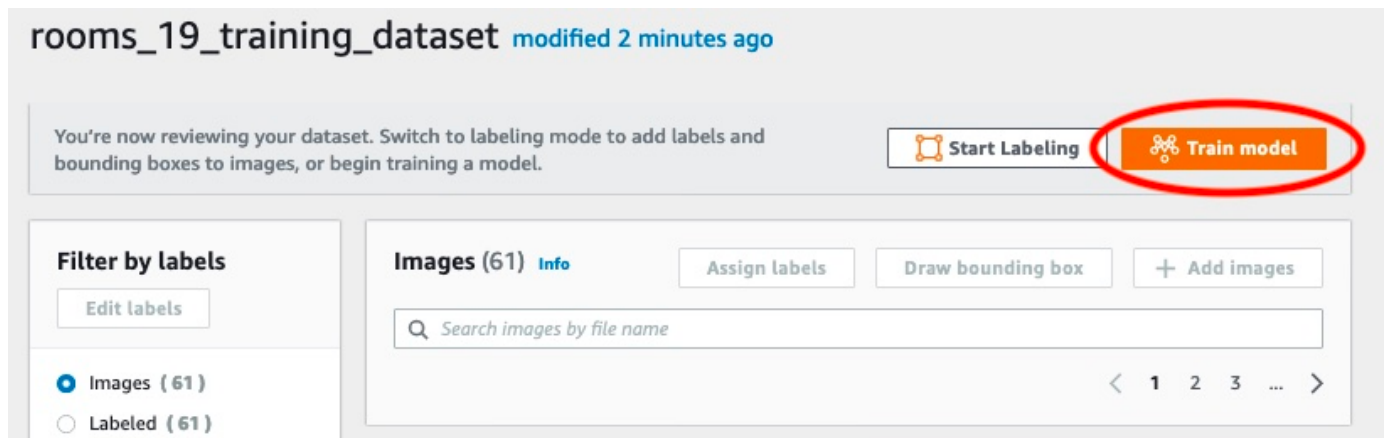


## ステップ 2: モデルのトレーニング

このステップでは自分のモデルをトレーニングします。トレーニングデータセットとテストデータセットは自動的に設定されます。トレーニングが正常に完了すると、全体的な評価結果と個々のテストイメージの評価結果を確認できます。詳細については、「[Amazon Rekognition Custom Labels モデルをトレーニングする](#)」を参照してください。

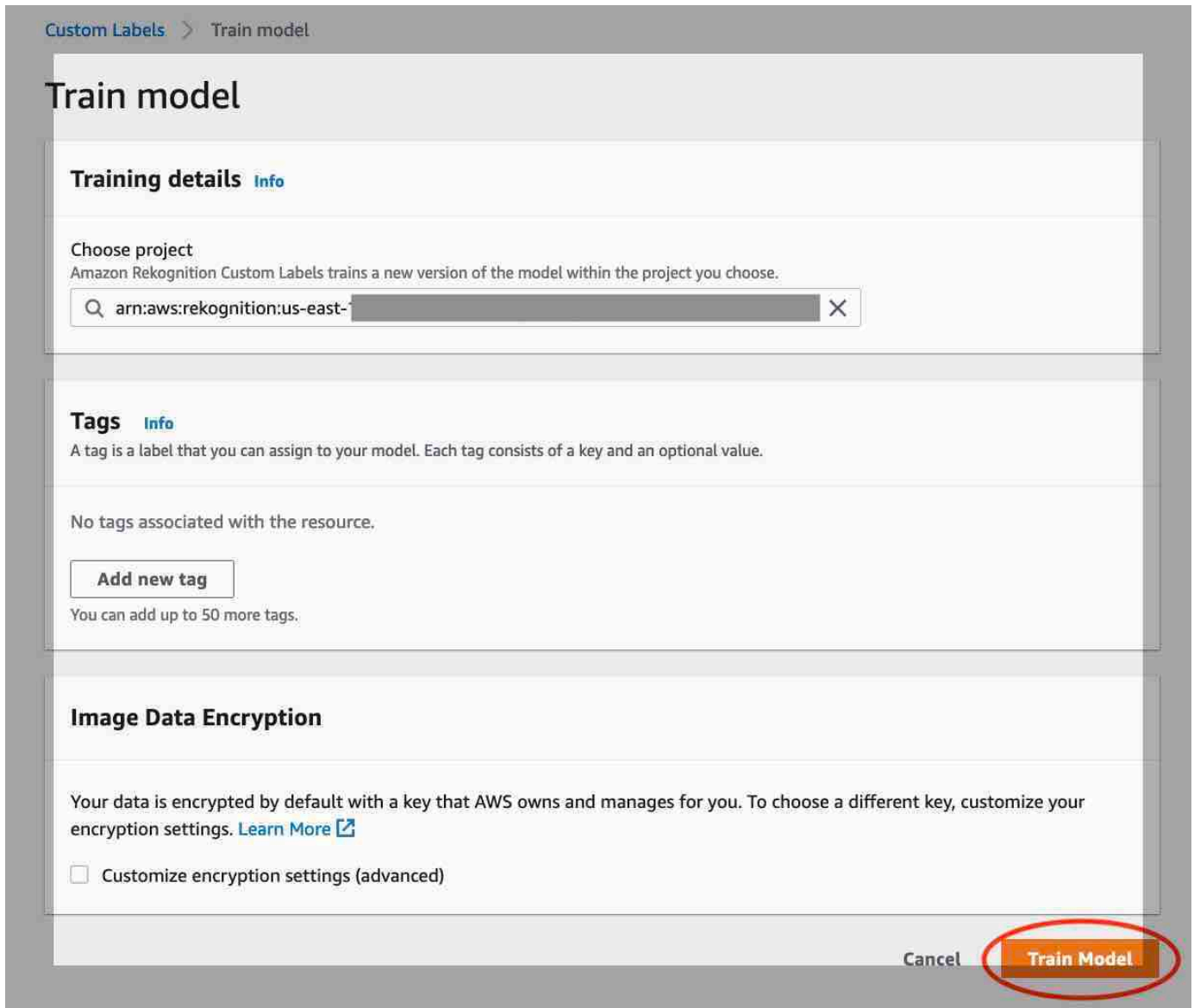
モデルをトレーニングするには

1. データセットページで、モデルのトレーニングを選択します。次の図は、モデルトレーニングボタンが表示されたコンソールを示しています。

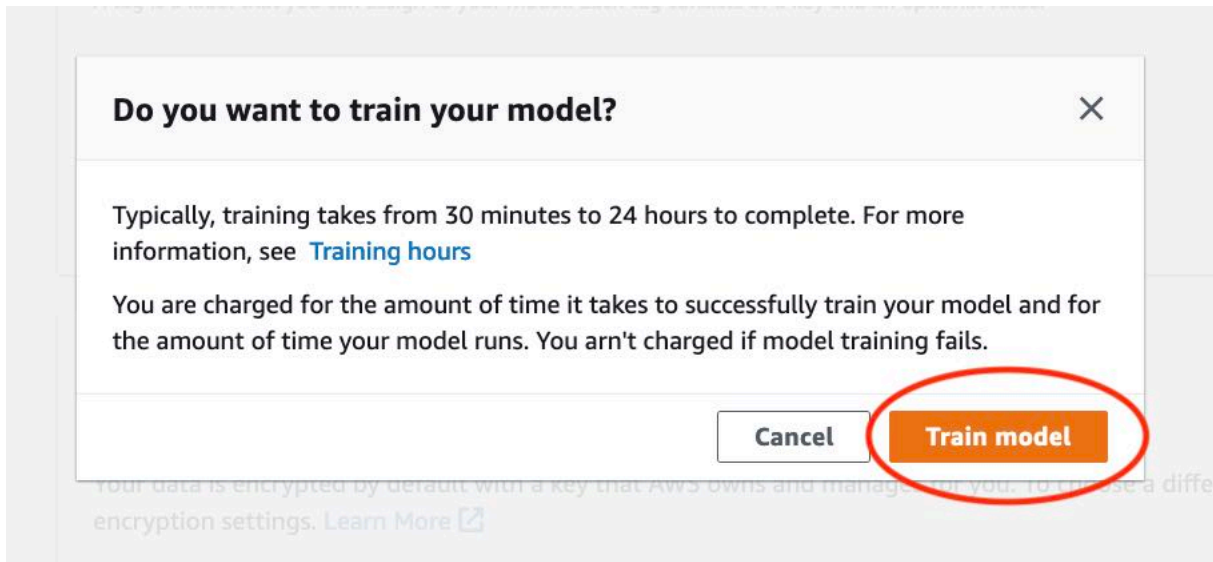


The screenshot shows the Amazon Rekognition console interface for a dataset named 'rooms\_19\_training\_dataset', which was modified 2 minutes ago. At the top, there is a message: 'You're now reviewing your dataset. Switch to labeling mode to add labels and bounding boxes to images, or begin training a model.' To the right of this message are two buttons: 'Start Labeling' and 'Train model'. The 'Train model' button is highlighted with a red circle. Below the message, there is a 'Filter by labels' section with an 'Edit labels' button and two radio button options: 'Images (61)' (selected) and 'Labeled (61)'. To the right of the filter section, there is a main area for 'Images (61)' with an 'Info' link, 'Assign labels', 'Draw bounding box', and '+ Add images' buttons. Below these buttons is a search bar with the placeholder text 'Search images by file name' and a search icon. At the bottom right of the main area, there is a pagination control showing '< 1 2 3 ... >'.

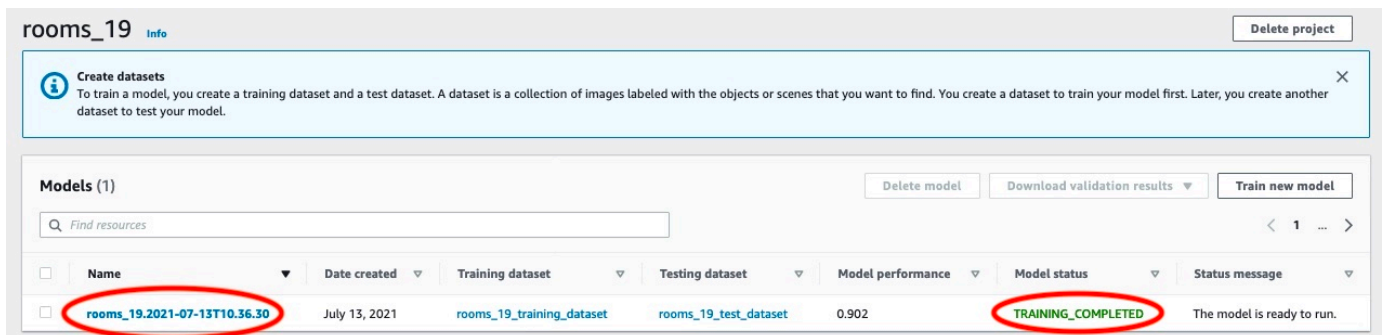
2. [モデルをトレーニング] ページで、[モデルをトレーニング] を選択します。以下の図は、モデルのトレーニングボタンを示しています。プロジェクトの Amazon リソースネーム (ARN) は、プロジェクトの選択編集ボックスにあります。



3. 次の画像に示すように、「モデルをトレーニングしますか？」ダイアログボックスで、「モデルをトレーニングする」を選択します。



4. トレーニングが完了したら、モデル名を選択します。次のコンソールのスクリーンショットに示すように、モデルのステータスが `TRAINING_COMPLETED` になるとトレーニングは終了します。



5. [評価] ボタンを選択すると、評価結果が表示されます。モデルの評価の詳細については、「[トレーニング済み Amazon Rekognition Custom Labels の改善](#)」を参照してください。
6. [テスト結果を表示] を選択すると、個々のテストイメージの結果が表示されます。次のスクリーンショットに示すように、評価ダッシュボードには、各ラベルの F1 スコア、精度、再現率などのメトリクスとテストイメージの数が表示されます。平均、精度、再現率などの全体的なメトリクスも表示されます。

**rooms\_19** [Info](#) Delete model

**Evaluate** | Model details | Use Model | Tags

**Evaluation results** View test results

F1 score [Info](#)  
0.902

Average precision [Info](#)  
0.893

Overall recall [Info](#)  
0.928

Date completed  
July 13, 2021  
Trained in 1.223 hours

Training dataset  
10 labels, 61 images

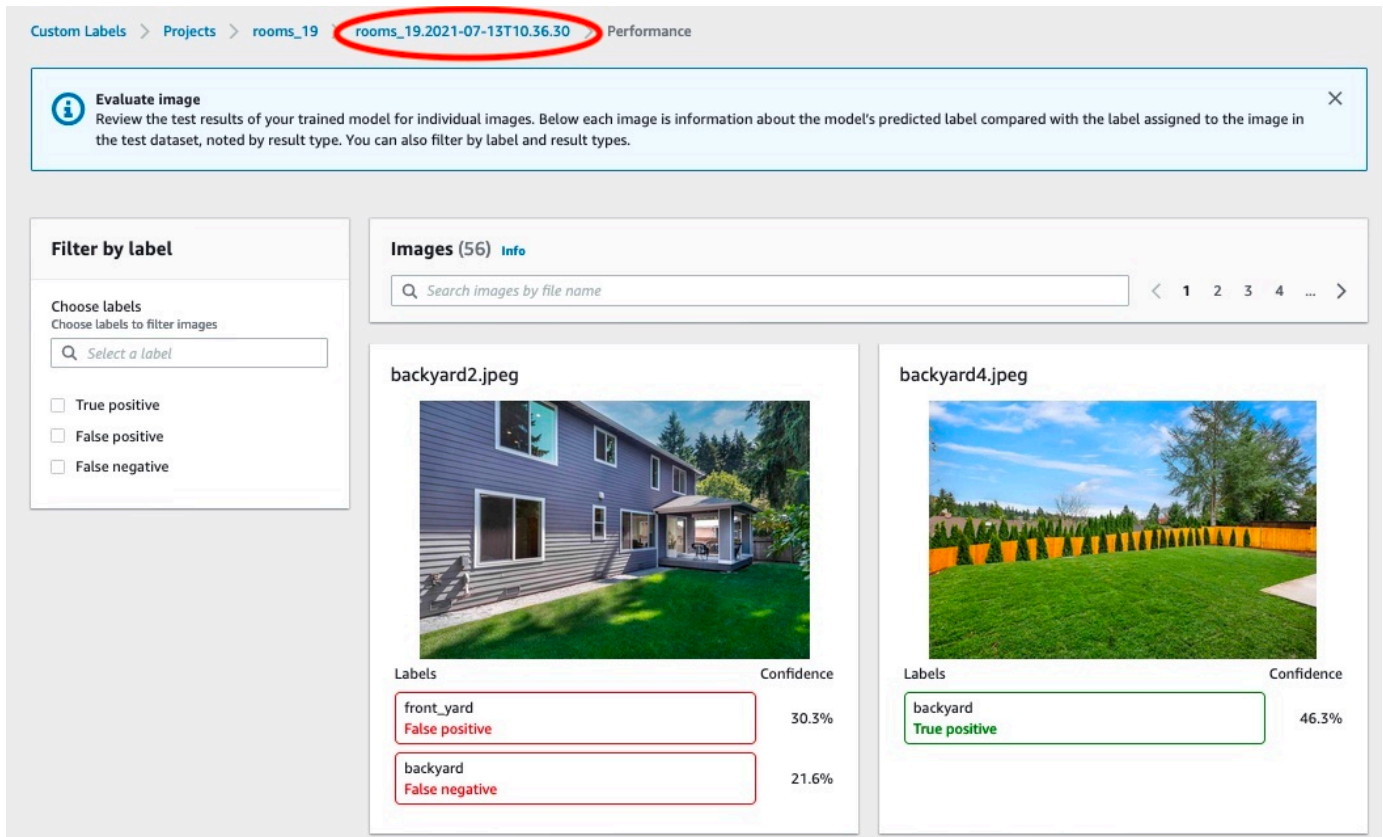
Testing dataset  
10 labels, 56 images

**Per label performance (10)**

Find labels < 1 >

Label name ▲	F1 score ▼	Test images ▼	Precision ▼	Recall ▼	Assumed threshold ▼
backyard	0.857	4	1.000	0.750	0.286
bathroom	0.889	9	0.889	0.889	0.185
bedroom	0.900	11	1.000	0.818	0.262
closet	1.000	2	1.000	1.000	0.169
entry_way	1.000	3	1.000	1.000	0.149
floor_plan	1.000	2	1.000	1.000	0.685

7. テスト結果を確認したら、モデル名を選択してモデルページに戻ります。次のパフォーマンスダッシュボードのスクリーンショットでは、モデルページに戻る をクリックします。



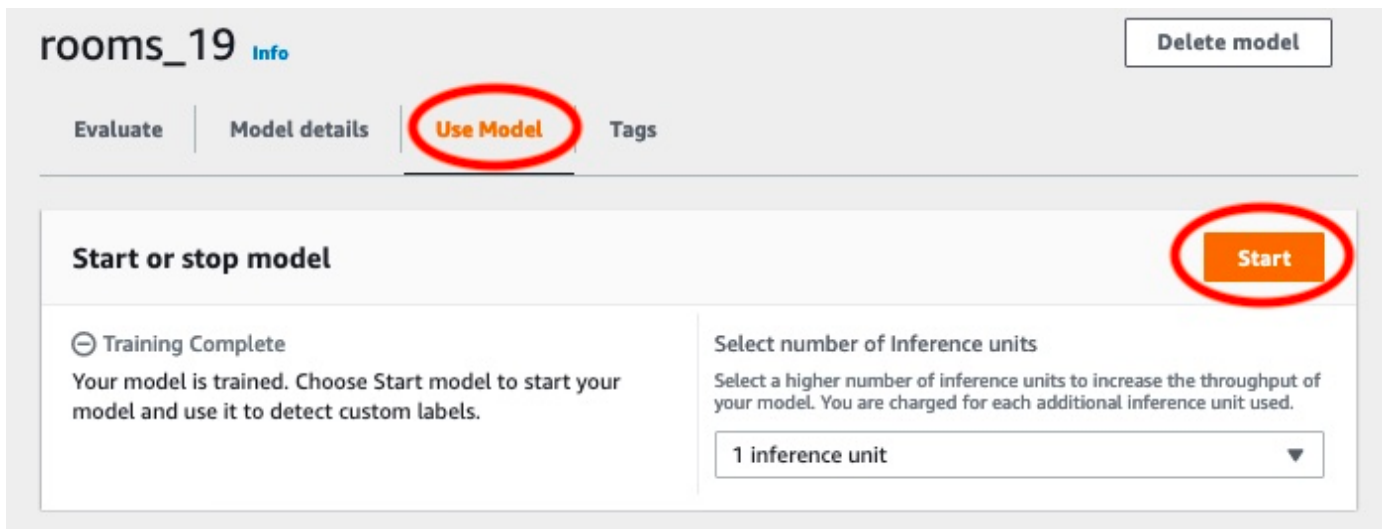
### ステップ 3: モデルをスタートする

このステップではモデルを開始します。モデルの開始後、モデルを使用してイメージを分析できます。

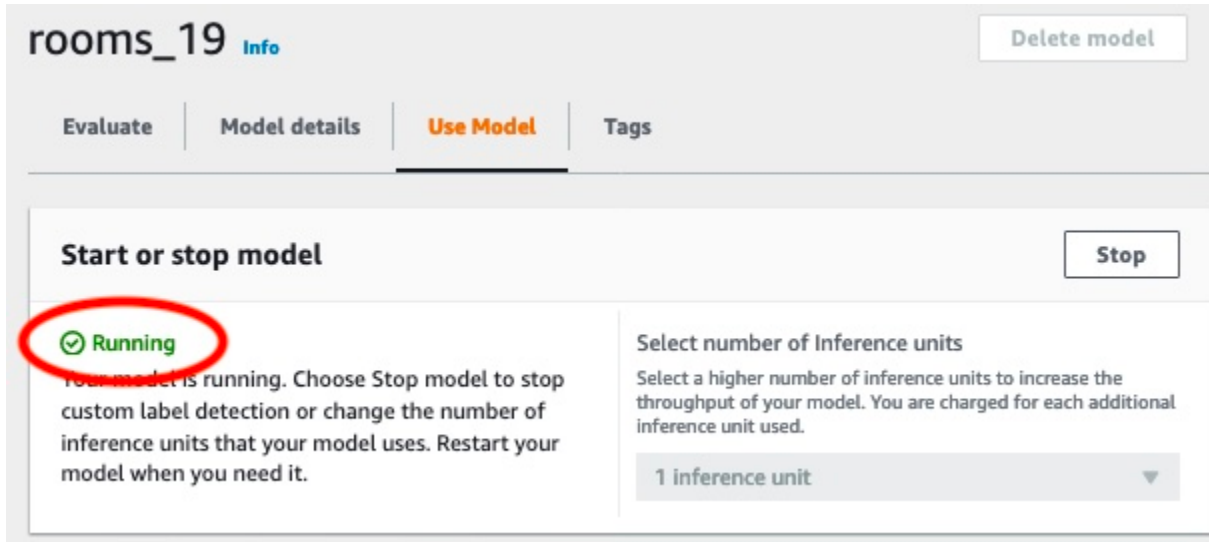
モデルの稼働時間に応じて課金されます。イメージを分析する必要がない場合は、モデルを停止してください。モデルは後で再起動できます。詳細については、「[トレーニング済み Amazon Rekognition Custom Labels の実行](#)」を参照してください。

モデルを開始するには

1. モデルページで [モデルを使用] タブを選択します。
2. [モデルの開始または停止] セクションで、次の操作を行います。
  - a. [開始] を選択します。
  - b. [モデルを開始] ダイアログボックスで、[開始] を選択します。次の図は、モデルコントロールパネルのスタートボタンを示しています。



3. モデルが実行されるまで待ちます。次のスクリーンショットは、モデルの実行中のコンソールを示しています。モデルの開始または停止セクションのステータスは実行中です。



4. モデルを使用してイメージを分類します。詳細については、「[ステップ 4: モデルを使用してイメージを分析する](#)」を参照してください。

## ステップ 4: モデルを使用してイメージを分析する

[DetectCustomLabels API](#) を呼び出してイメージを分析します。このステップでは、`detect-custom-labels` AWS Command Line Interface (AWS CLI) コマンドを使用してサンプルイメージを分析します。AWS CLI コマンドは Amazon Rekognition Custom Labels コンソールから取得します。コンソールは、モデルを使用するように AWS CLI コマンドを設定します。Amazon S3 バケッ



トに保存されているイメージを指定するだけです。このトピックでは、各プロジェクト例に使用できるイメージを提供しています。

#### Note

コンソールには Python サンプルコードも用意されています。

detect-custom-labels からの出力には、イメージ内のラベルのリスト、境界ボックス (モデルがオブジェクトの位置を検出した場合)、予測の精度に対するモデルの信頼度が含まれます。

詳細については、「[トレーニングされたモデルによるイメージの分析](#)」を参照してください。

イメージを分析するには (コンソール)

1. <textobject><phrase>モデルのステータスが Running と表示され、停止ボタンを使用して実行中のモデルを停止します。</phrase></textobject>

まだ設定していない場合は、[AWS CLI をセットアップ](#)します。手順については、「[the section called “ステップ 4: AWS CLI と AWS SDKs を設定する”](#)」を参照してください。

2. まだ実行していない場合は、モデルの実行を開始します。詳細については、「[ステップ 3: モデルをスタートする](#)」を参照してください。
3. [モデルを使用] タブを選択し、[API コード] を選択します。以下のモデルステータスパネルには、実行中のモデルが表示され、実行中のモデルを停止する停止ボタンと API を表示するオプションがあります。



The screenshot shows the AWS Rekognition console for a custom label model named 'rooms\_19'. At the top right, there is a 'Delete model' button. Below the model name, there are four tabs: 'Evaluate', 'Model details', 'Use Model' (which is selected and circled in red), and 'Tags'. The main content area is divided into two sections. The first section, 'Start or stop model', contains a 'Stop' button and a status indicator 'Running' with a green checkmark. Below the status, there is a description: 'Your model is running. Choose Stop model to stop custom label detection or change the number of inference units that your model uses. Restart your model when you need it.' To the right of this text, there is a section titled 'Select number of Inference units' with a dropdown menu currently set to '1 inference unit'. The second section, 'Use your model', contains a text input field for 'Amazon Resource Name (ARN)'. At the bottom of this section, there is a link '▶ API Code' which is circled in red.

4. [AWS CLI コマンド] を選択します。
5. イメージの分析セクションで、 を呼び出す AWS CLI コマンドをコピーします detect-custom-labels。次の Rekognition コンソールのイメージは、機械学習モデルを使用してイメージ上のカスタムラベルを検出する AWS CLI コマンドと、モデルを起動してイメージの詳細を提供する手順を含む「イメージの分析」セクションを示しています。

**Use your model**

Amazon Resource Name (ARN)

▼ API Code

Use your model rooms\_ [ ] by calling the following AWS CLI commands or Python scripts. You can start and stop the model, and analyze custom labels in new images.

AWS CLI command

Python

**Start model**  
Command used to start the rooms\_ [ ] model.

```
1 aws rekognition start-project-version \  
2 --project-version-arn "arn:aws:rekognition:us-east-1:[ ]" \  
3 --min-inference-units 1 \  
4 --region us-east-1
```

**Analyze image**  
Command used to use analyze an image with the rooms\_ [ ] model. Replace MY\_BUCKET and PATH\_TO\_MY\_IMAGE with your S3 bucket name and image path.

```
1 aws rekognition detect-custom-labels \  
2 --project-version-arn "arn:aws:rekognition:us-east-1:[ ]" \  
3 --image '{"S3Object": {"Bucket": "MY_BUCKET", "Name": "PATH_TO_MY_IMAGE"}}' \  
4 --region us-east-1
```

6. 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。手順については、「[イメージ例の取得](#)」を参照してください。
7. コマンドプロンプトで、前のステップでコピーした AWS CLI コマンドを入力します。次の例のようになります。

--project-version-arn の値は、モデルの Amazon リソースネーム (ARN) になるはずですが、--region の値は、モデルを作成した AWS リージョンであるはずですが。

MY\_BUCKET と PATH\_TO\_MY\_IMAGE を前のステップで使用した Amazon S3 バケットとイメージに変更します。

[custom-labels-access](#) プロファイルを使用して認証情報を取得する場合は、--profile custom-labels-access パラメータを追加します。

```
aws rekognition detect-custom-labels \  
  --project-version-arn "model_arn" \  
  --image '{"S3Object": {"Bucket": "MY_BUCKET", "Name": "PATH_TO_MY_IMAGE"}}' \  
  --region us-east-1 \  
  --profile custom-labels-access
```

モデルがオブジェクトおよびコンセプトを見つけた場合、AWS CLI コマンドの JSON 出力は次のようになります。Name はモデルが見つけたイメージレベルのラベル名です。Confidence (0-100) は予測の精度に対するモデルの信頼度です。

```
{  
  "CustomLabels": [  
    {  
      "Name": "living_space",  
      "Confidence": 83.41299819946289  
    }  
  ]  
}
```

モデルがオブジェクトの位置やブランドを見つけたら、ラベル付きの境界ボックスが返されます。BoundingBox にはオブジェクトを囲むボックスの位置が含まれます。Name は境界ボックス内でモデルが見つけたオブジェクトです。Confidence は境界ボックス内にオブジェクトが含まれているモデルの信頼度です。

```
{  
  "CustomLabels": [  
    {  
      "Name": "textextract",  
      "Confidence": 87.7729721069336,  
      "Geometry": {  
        "BoundingBox": {  
          "Width": 0.198987677693367,  
          "Height": 0.31296101212501526,  
          "Left": 0.07924537360668182,  
          "Top": 0.4037395715713501  
        }  
      }  
    }  
  ]  
}
```

- 引き続きモデルを使用して他のイメージを分析してください。使用しなくなったモデルは停止してください。詳細については、「[ステップ 5: モデルを停止する](#)」を参照してください。

## イメージ例の取得

DetectCustomLabels オペレーションでは、次のイメージを使用できます。各プロジェクトには 1 つのイメージがあります。イメージを使用するには、イメージを S3 バケットにアップロードします。

イメージ例を使用するには

1. 使用しているプロジェクト例と一致する、次のイメージを右クリックします。次に [イメージを保存] を選択し、イメージをコンピュータに保存します。メニューオプションは、ご使用のブラウザによって異なる場合があります。
2. AWS アカウントが所有し、Amazon Rekognition Custom Labels を使用しているのと同じリージョンにある Amazon S3 バケットにイメージをアップロードします。AWS Amazon Rekognition

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。

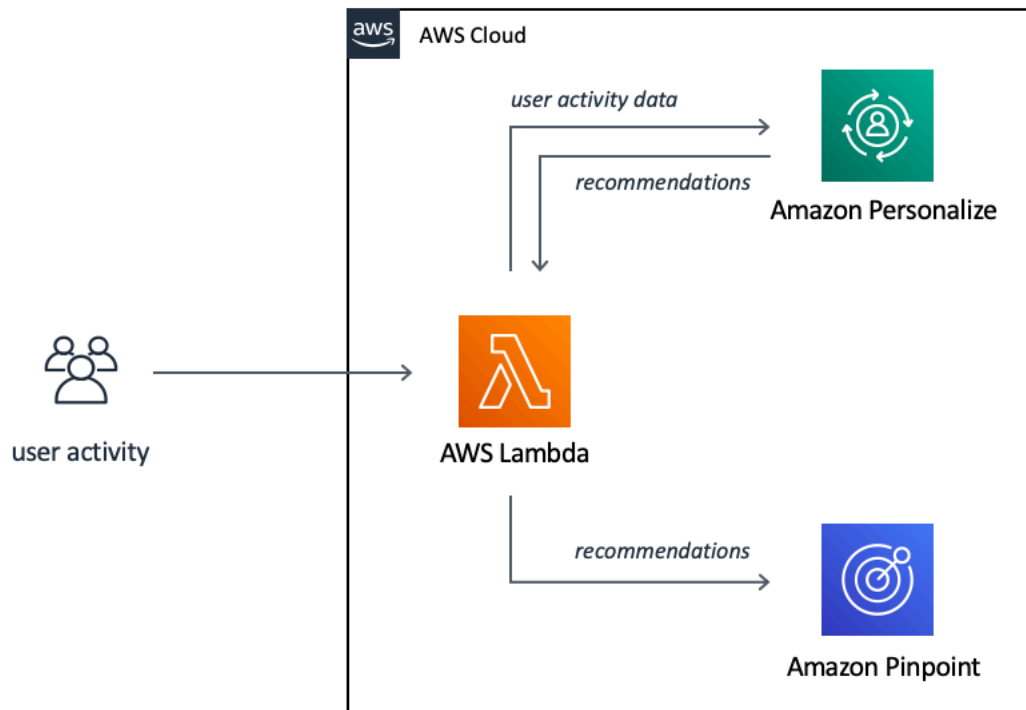
## 画像分類



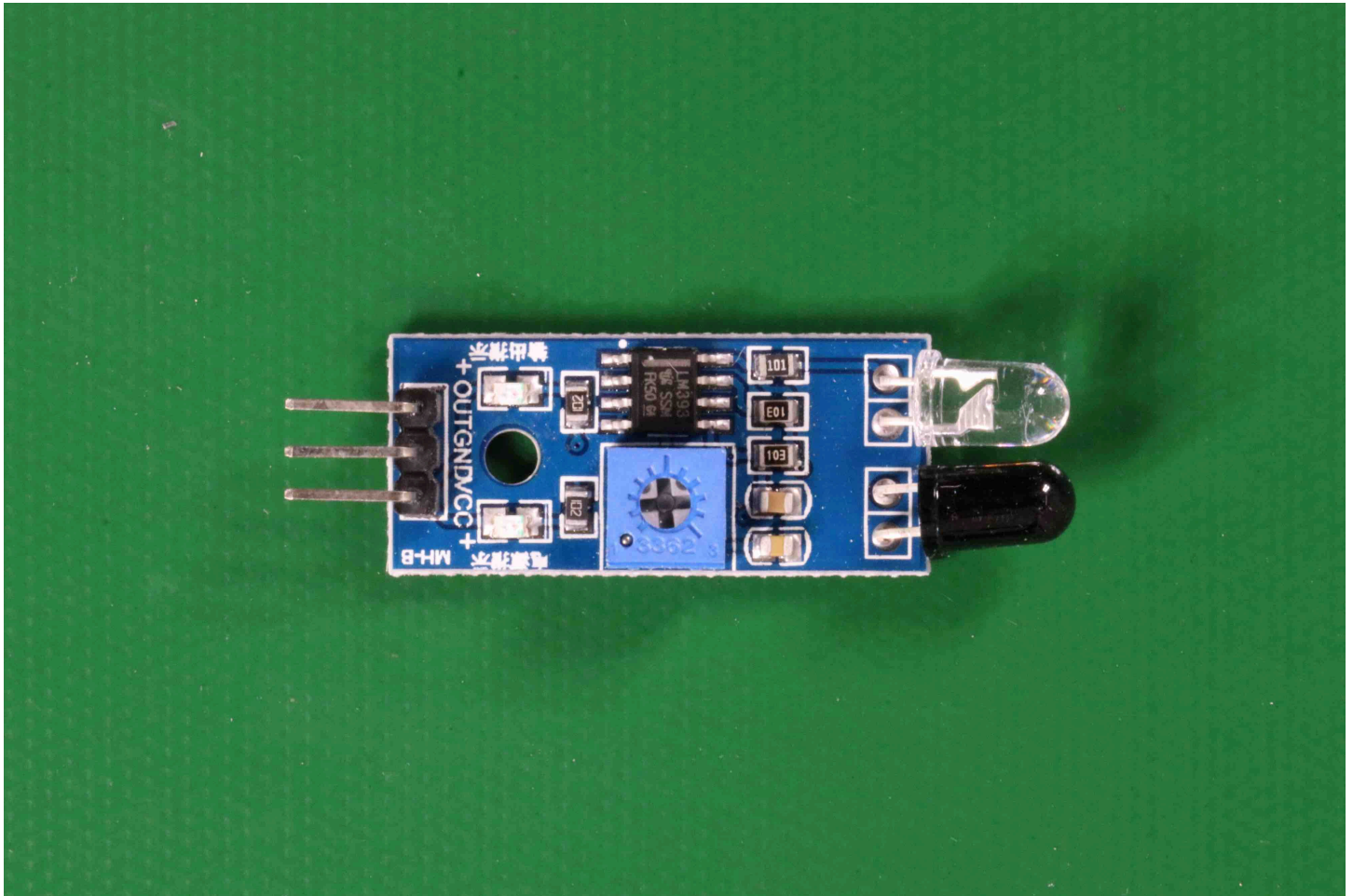
## マルチラベル分類



## ブランド検出



## オブジェクトのローカリゼーション



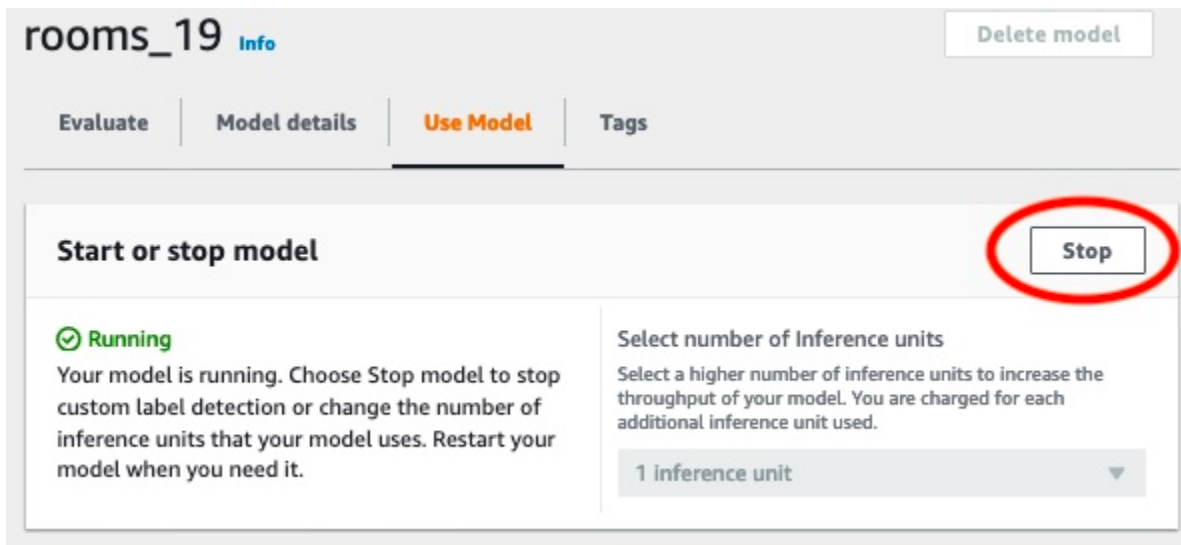
### ステップ 5: モデルを停止する

このステップではモデルの実行を停止します。モデルの稼働時間に応じて課金されます。モデルの使用を終了した場合は、停止する必要があります。

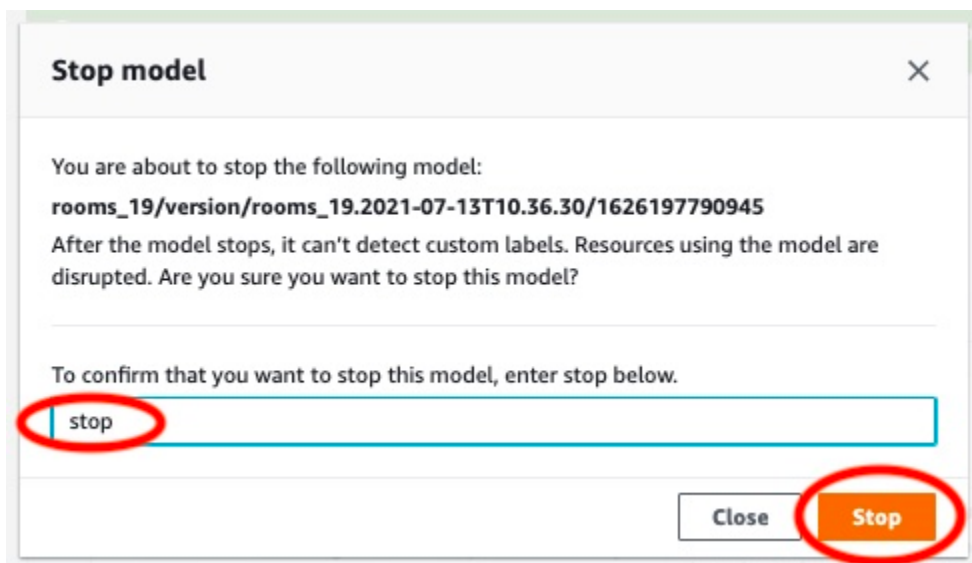
モデルを停止するには

1. [モデルの開始または停止] セクションで、[停止] を選択します。





2. [モデルを停止] ダイアログボックスで、[停止] と入力し、モデルを停止することを確認します。



3. [停止] を選択してモデルを停止します。[モデルの開始または停止] セクションのステータスが [停止済み] になると、モデルは停止します。次のスクリーンショットでは、ユーザーインターフェイスセクションに機械学習モデルを開始または停止するオプションがあります。モデルのステータスは「Stopped」と表示され、「Start」ボタンが表示され、推論単位の数を選択するためのドロップダウンが表示されます。

The screenshot shows the Amazon Rekognition console interface for a custom model named 'rooms\_19'. The model is currently in a 'Stopped' state, indicated by a red circle around the 'Stopped' text and a minus sign icon. The 'Use Model' tab is active, and the 'Start or stop model' section contains a 'Start' button and a dropdown menu set to '1 inference unit'. The console also shows tabs for 'Evaluate', 'Model details', and 'Tags'.

## ステップ 6 : 次のステップ

プロジェクト例を試し終えたら、独自のイメージとデータセットを使用して独自のモデルを作成できます。詳細については、「[Amazon Rekognition Custom Labels について](#)」を参照してください。

以下の表のラベル情報を使用して、サンプルプロジェクトと類似したモデルをトレーニングしてください。

例	トレーニングイメージ	テストイメージ
イメージ分類 (部屋)	1つのイメージにつき 1つのイメージレベルのラベル	1つのイメージにつき 1つのイメージレベルのラベル
マルチラベル分類 (花)	1つのイメージにつき複数のイメージレベルのラベル	1つのイメージにつき複数のイメージレベルのラベル
ブランド検出 (ロゴ)	イメージレベルのラベル (ラベル付き境界ボックスも使用できます)	ラベル付き境界ボックス
イメージローカリゼーション (回路基板)	ラベル付き境界ボックス	ラベル付き境界ボックス



[チュートリアル: イメージの分類](#) は、イメージ分類モデルのプロジェクト、データセット、モデルを作成する方法を示しています。

データセットの作成とモデルのトレーニングの詳細については、「[Amazon Rekognition Custom Labels モデルの作成](#)」を参照してください。

# チュートリアル: イメージの分類

このチュートリアルでは、イメージに含まれるオブジェクト、シーン、概念を分類するモデルのプロジェクトとデータセットを作成する方法を説明します。このモデルはイメージ全体を分類します。例えば、このチュートリアルに従うと、リビングルームやキッチンなどの家庭の場所を認識するようにモデルをトレーニングできます。このチュートリアルでは、モデルを使用してイメージを分析する方法も説明します。

このチュートリアルを開始する前に、「[Amazon Rekognition Custom Labels について](#)」を読むことをお勧めします。

このチュートリアルでは、ローカルコンピュータからイメージをアップロードして、トレーニングデータセットとテストデータセットを作成します。後で、トレーニングおよびテストデータセットのイメージに画像レベルのラベルを割り当てます。

作成したモデルでは、トレーニングデータセットのイメージに割り当てたイメージレベルのラベルセットに属するものとしてイメージを分類します。例えば、トレーニングデータセット内のイメージレベルのラベルのセットが、kitchen、living\_room、patio、および backyard の場合、モデルは 1 つのイメージ内のそれらのイメージレベルのラベルをすべて検出できる可能性があります。

## Note

イメージ上のオブジェクトの位置を調べるなど、さまざまな目的でモデルを作成できます。詳細については、「[モデルタイプの決定](#)」を参照してください。

## ステップ 1: イメージを集める

2 セットのイメージが必要です。1 セットはトレーニングデータセットに追加します。もう 1 つのセットはテストデータセットに追加します。イメージは、モデルが分類するオブジェクト、シーン、概念を表している必要があります。イメージは、PNG または JPEG 形式であることが必要です。詳細については、「[イメージの準備](#)」を参照してください。

トレーニングデータセットには 10 枚以上、テストデータセットには 10 枚以上のイメージが必要です。

まだイメージがない場合は、[ルーム] サンプル分類プロジェクトのイメージを使用してください。プロジェクトを作成すると、トレーニングイメージとテストイメージは次の Amazon S3 バケットの場所に配置されます。

- トレーニングイメージ - `s3://custom-labels-console-region-numbers/assets/rooms_version number_test_dataset/`
- テストイメージ - `s3://custom-labels-console-region-numbers/assets/rooms_version number_test_dataset/`

`region` は、Amazon Rekognition Custom Labels コンソールを使用している AWS リージョンです。`numbers` はコンソールがバケット名に割り当てる値です。`Version number` は 1 から始まるサンプルプロジェクトのバージョン番号です。

以下の手順では、Rooms プロジェクトのイメージを、コンピュータ上の `training` および `test` という名前のローカルフォルダに保存します。

Rooms サンプルプロジェクトのイメージファイルをダウンロードするには

1. Rooms プロジェクトを作成します。詳細については、「[ステップ 1: プロジェクト例を選択する](#)」を参照してください。
2. コマンドプロンプトを開き、次のコマンドを入力してトレーニングイメージをダウンロードします。

```
aws s3 cp s3://custom-labels-console-region-numbers/assets/rooms_version number_training_dataset/ training --recursive
```

3. コマンドプロンプトで、次のコマンドを入力してテストイメージをダウンロードします。

```
aws s3 cp s3://custom-labels-console-region-numbers/assets/rooms_version number_test_dataset/ test --recursive
```

4. 2つのイメージをトレーニングフォルダから任意の別のフォルダに移動します。このイメージを使用して、トレーニングしたモデルを [ステップ 9: モデルを使用してイメージを分析する](#) で試してみます。

## ステップ 2: クラスを決める

モデルで検索するクラスのリストを作成します。例えば、家の部屋を認識するようにモデルをトレーニングする場合、次のイメージを `living_room` として分類できます。



各クラスはイメージレベルのラベルにマップされます。後で、トレーニングおよびテストデータセットのイメージに画像レベルのラベルを割り当てます。

Rooms サンプルプロジェクトのイメージを使用している場合、イメージレベルのラベルは、裏庭、浴室、寝室、クローゼット、entry\_way、floor\_plan、front\_yard、kitchen、living\_space、およびパティオです。

## ステップ 3: プロジェクトを作成する

データセットとモデルを管理するには、プロジェクトを作成します。プロジェクトごとに、家の中の部屋を認識するなど、単一のユースケースに対応する必要があります。

プロジェクトを作成するには (コンソール)

1. まだ追加していない場合には、Amazon Rekognition Custom Labels コンソールを設定します。詳細については、「[Amazon Rekognition Custom Labels のセットアップ](#)」を参照してください。
2. AWS Management Console にサインインし、<https://console.aws.amazon.com/rekognition/> で Amazon Rekognition コンソールを開きます。
3. 左側のペインで、[カスタムラベルを使用] を選択します。Amazon Rekognition Custom Labels のランディングページが表示されます。
4. Amazon Rekognition Custom Labels のランディングページで、[開始方法] を選択します。
5. 左側のナビゲーションペインで、[プロジェクト] を選択します。
6. [プロジェクト] ページで、[プロジェクトを作成] を選択します。

7. [Project name] (プロジェクト名) にプロジェクトの名前を入力します。
8. [プロジェクトを作成] を選択してプロジェクトを作成します。

Custom Labels > Create project

## Create project [Info](#)

**Project details**

Project name

The project name can't be more than 63 characters. It can only contain alphanumeric characters, with no spaces or special characters.

Cancel **Create project**

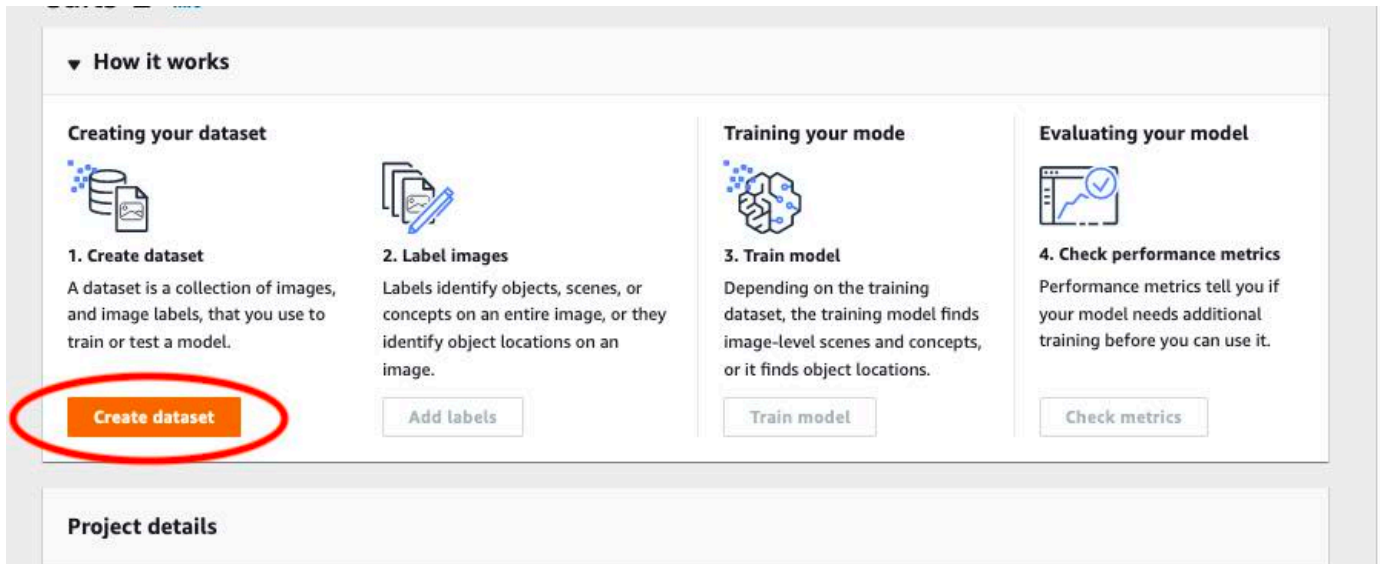
## ステップ 4: トレーニングデータセットおよびテストデータセットを作成する

このステップでは、ローカルコンピュータからイメージをアップロードして、トレーニングデータセットとテストデータセットを作成します。同時にアップロードできるイメージは、30 枚までです。アップロードするイメージがたくさんある場合は、Amazon S3 バケットからイメージをインポートしてデータセットを作成することを検討してください。詳細については、「[Amazon S3 バケット](#)」を参照してください。

データセットの詳細については、「[データセットの管理](#)」を参照してください。

ローカルコンピュータ上の画像を使用してデータセットを作成するには (コンソール)

1. プロジェクトの詳細ページで、[Create dataset] (データセットの作成) を選択します。



2. [設定の開始] セクションで、[トレーニングデータセットとテストデータセットで開始する] を選択します。
3. [Training dataset details] (トレーニングデータセット詳細) セクションで、[Upload images from your computer] (コンピュータから画像をアップロードする) を選択します。
4. [Test dataset details] (テストデータセット詳細) セクションで、[Upload images from your computer] (コンピュータから画像をアップロードする) を選択します。
5. [データセットを作成] を選択します。

### Create dataset Info

**Starting configuration**

Configuration options

- Start with a single dataset  
When you train your model, the dataset is split to create the training dataset (80%) and test dataset (20%) for your project.
- Start with a training dataset and a test dataset  
Recommended for most users. Start with the highest control over training, testing, and performance tuning.

**What are training datasets and test datasets?**

- A training dataset teaches your model to identify scenes or objects in images.
- A test dataset evaluates the performance of your trained model.

**Training dataset details**

Import images Info  
Import images from one of the sources below.

- Import images from S3 bucket  
Use images from an existing S3 bucket by entering the S3 bucket URL. You can automatically add labels based on your S3 bucket folder names.
- Upload images from your computer  
Add images by uploading files from your local computer. You're limited to uploading 50 images at one time.
- Copy an existing Amazon Rekognition Custom Labels dataset  
Use an existing dataset as a starting point for your new dataset. Your original dataset will remain unchanged.
- Import images labeled by SageMaker Ground Truth  
Provide the location of your manifest file. If you have a labeled datasets in a different format, convert them to a manifest format.

**Test dataset details**

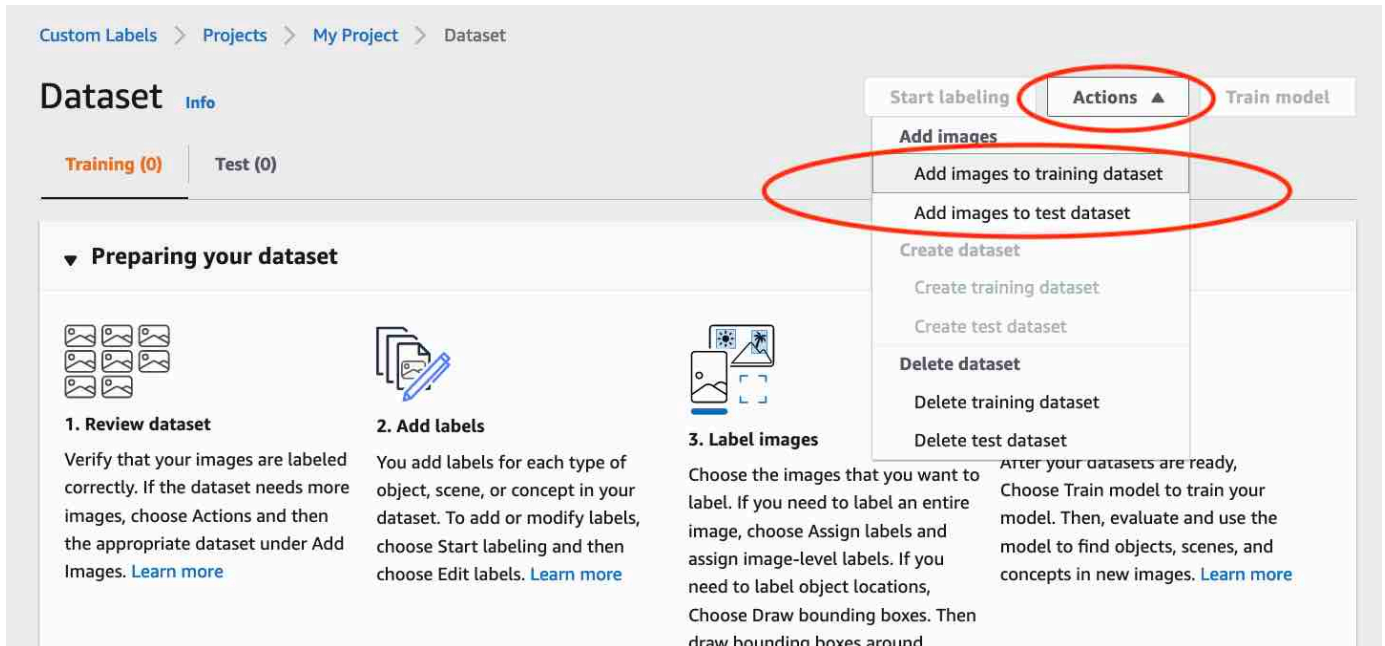
Import images Info  
Import images from one of the sources below.

- Import images from S3 bucket  
Use images from an existing S3 bucket by entering the S3 bucket URL. You can automatically add labels based on your S3 bucket folder names.
- Upload images from your computer  
Add images by uploading files from your local computer. You're limited to uploading 50 images at one time.
- Copy an existing Amazon Rekognition Custom Labels dataset  
Use an existing dataset as a starting point for your new dataset. Your original dataset will remain unchanged.
- Import images labeled by SageMaker Ground Truth  
Provide the location of your manifest file. If you have a labeled datasets in a different format, convert them to a manifest format.

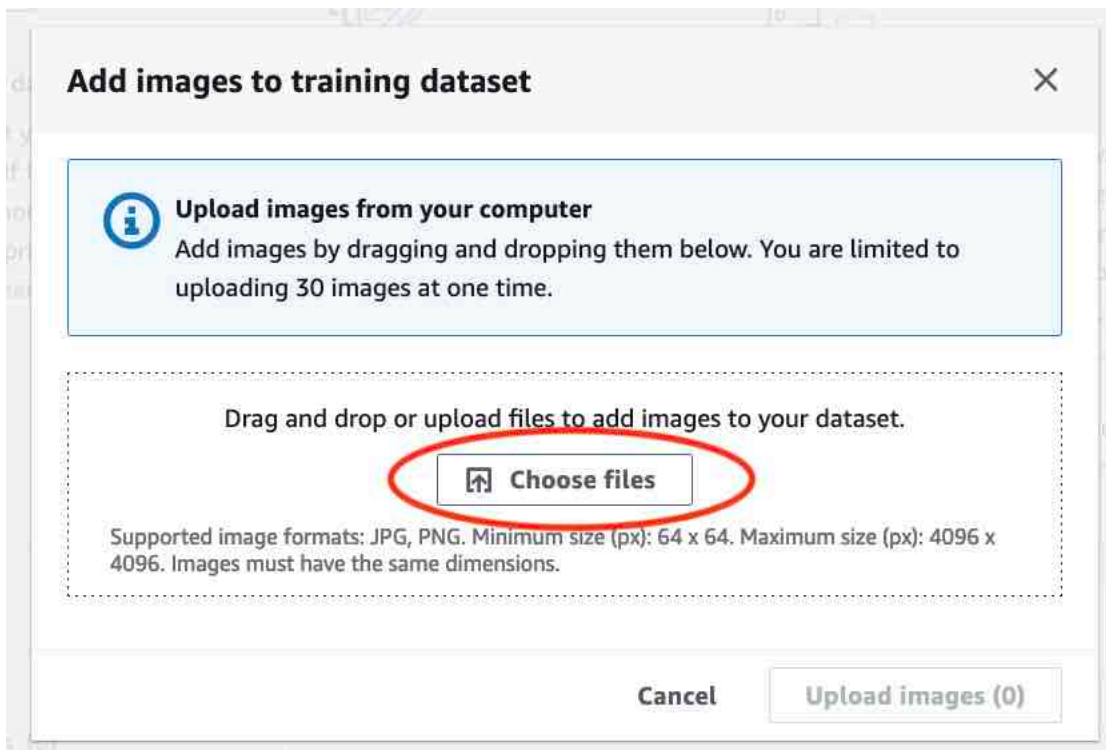
Cancel **Create Datasets**



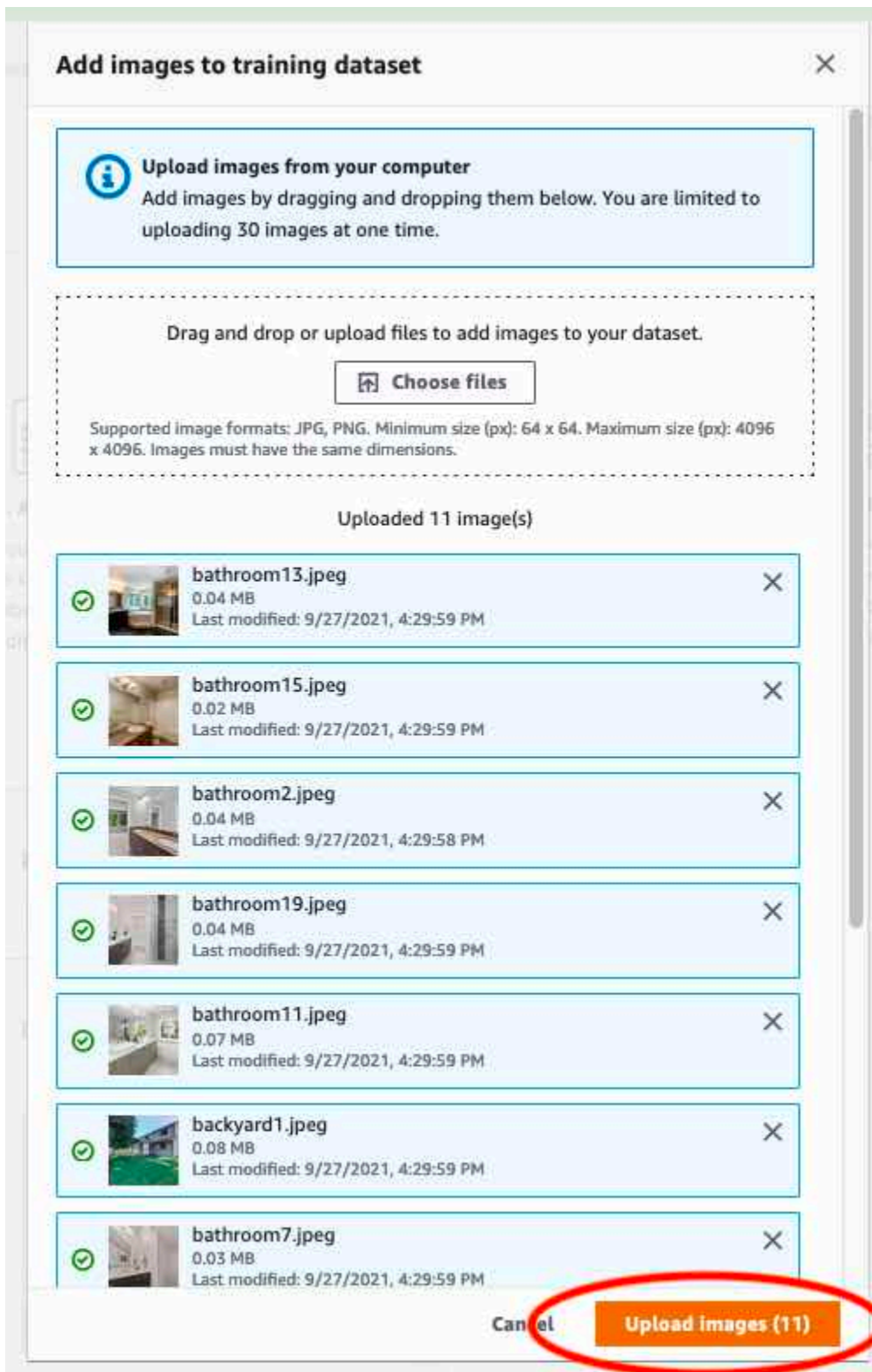
6. データセットページが表示され、それぞれのデータセットの[Training] (トレーニング) タブと[Test] (テスト) タブが表示されます。
7. [データセット] ページで [トレーニング] タブを選択します。
8. [Actions] (アクション) を選択し、[Add images to training dataset] (トレーニングデータセットに画像を追加する) をクリックします。



9. [トレーニングデータセットに画像を追加] ダイアログボックスで、[ファイルを選択] を選択します。



10. データセットにアップロードする画像を選択します。同時にアップロードできるイメージは、30 枚までです。
11. [Upload images] (画像をアップロード) を選択します。Amazon Rekognition Custom Labels がデータセットにイメージを追加するまでに数秒かかることがあります。



12. トレーニングデータセットに追加するイメージが他にもある場合は、ステップ 9~12 を繰り返します。
13. [テスト] タブを選択します。

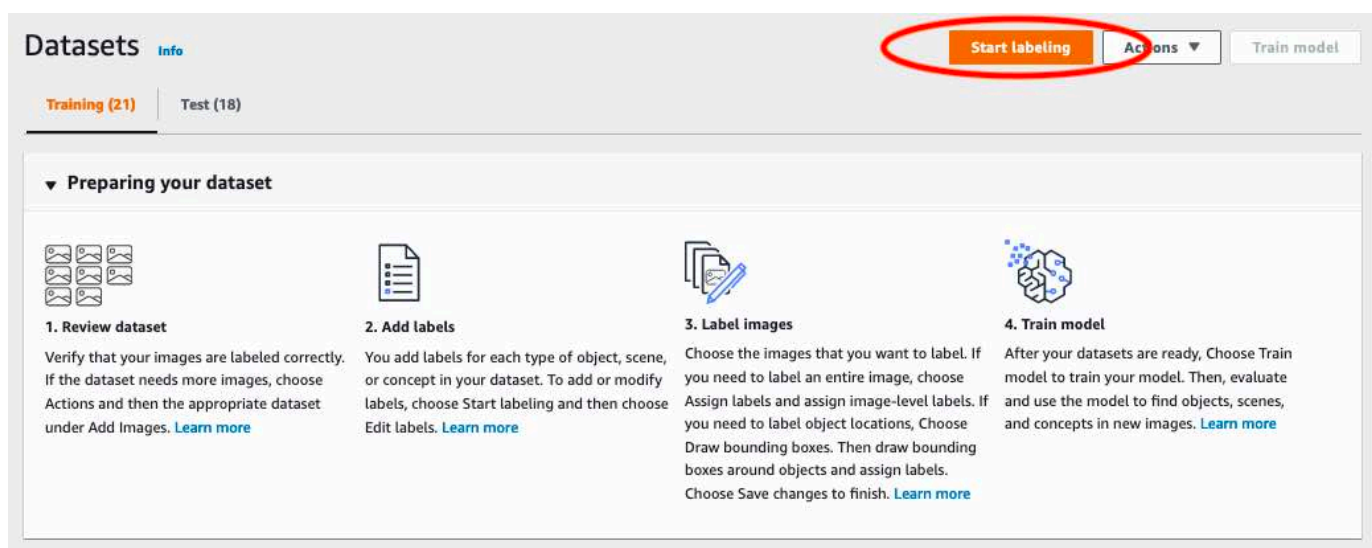
- ステップ 8~12 を繰り返して、テストデータセットにイメージを追加します。ステップ 8 で、[アクション] を選択します。[テストデータセットに画像を追加] をクリックします。

## ステップ 5: ラベルをプロジェクトに追加する

このステップでは、ステップ [ステップ 2: クラスを決める](#) で特定した各クラスのラベルをプロジェクトに追加します。

新しいラベルを追加するには (コンソール)

- データセットギャラリーページで、[ラベル付けを開始] を選択してラベル付けモードに入ります。



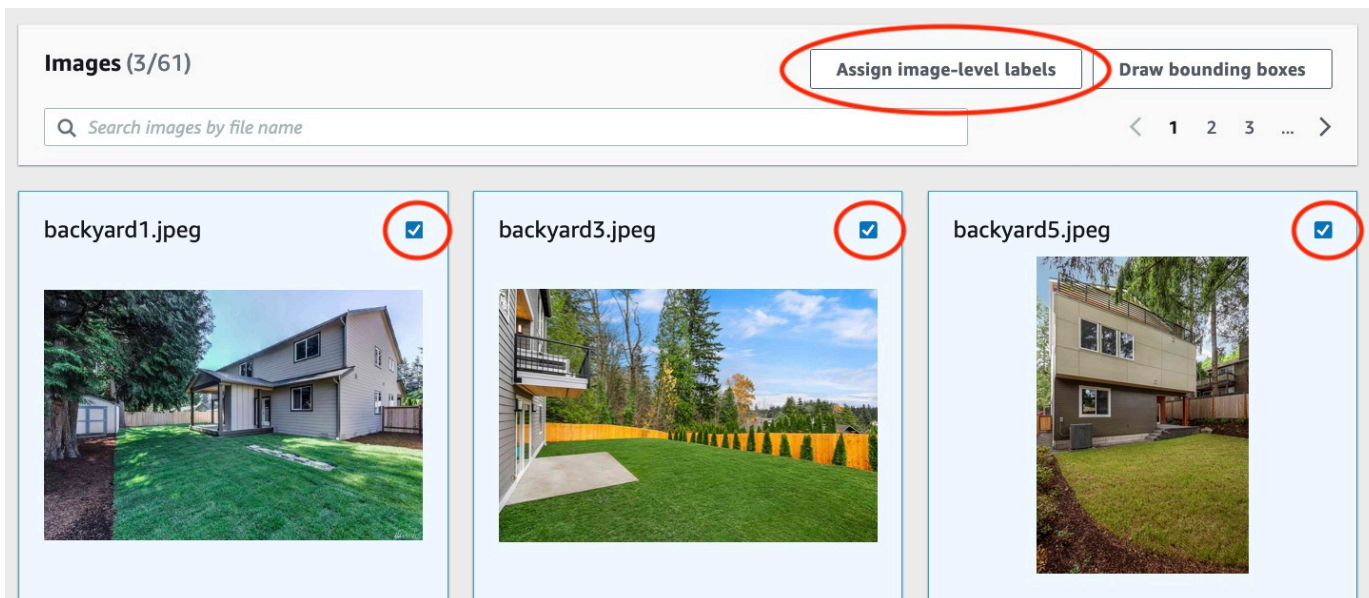
- データセットギャラリーの [ラベル] セクションで、[ラベルを編集] を選択し、[ラベルを管理] ダイアログボックスを開きます。
- 編集ボックスに新しいラベル名を入力します。
- [新しいラベルを追加] を選択します。
- 必要なラベルがすべて作成されるまで、ステップ 3 と 4 を繰り返します。
- [保存] を選択して、追加したラベルを保存します。

## ステップ 6: トレーニングデータセットとテストデータセットに画像レベルのラベルを割り当てる

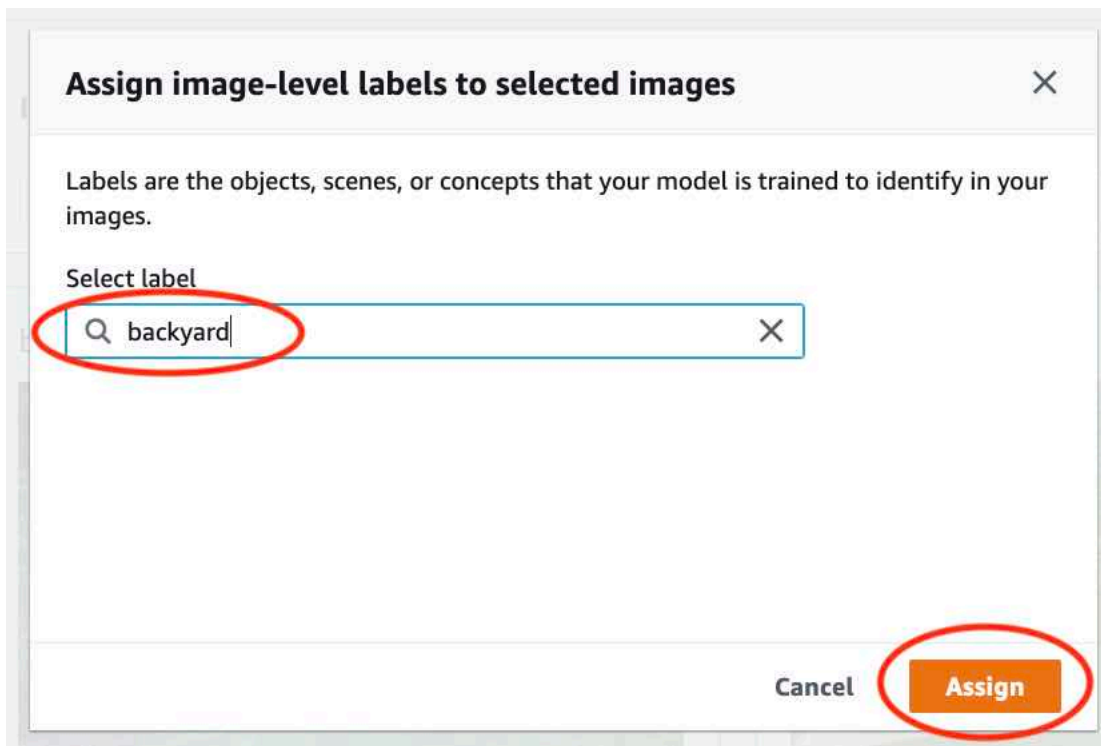
このステップでは、トレーニングデータセットとテストデータセットの各イメージに 1 つのイメージレベルを割り当てます。イメージレベルのラベルは、各イメージが表すクラスです。

画像レベルのラベルを画像に割り当てるには (コンソール)

1. [データセット] ページで [トレーニング] タブを選択します。
2. [ラベル付けを開始] を選択してラベル付けモードに入ります。
3. ラベルを追加するイメージを 1 つ以上選択します。一度に選択できるのは 1 ページのイメージのみです。1 ページの連続した範囲のイメージを選択するには
  - a. 最初のイメージを選択します。
  - b. Shift キーを押したままにします。
  - c. 2 つ目のイメージを選択します。最初のイメージと 2 番目のイメージ間のイメージも選択されます。
  - d. Shift キーを放します。
4. [画像レベルのラベルを割り当てる] を選択します。



5. [画像レベルのラベルを選択した画像に割り当てる] ダイアログボックスで、1 つ以上のイメージに割り当てるラベルを選択します。
6. [割り当てる] を選択して、イメージにラベルを割り当てます。



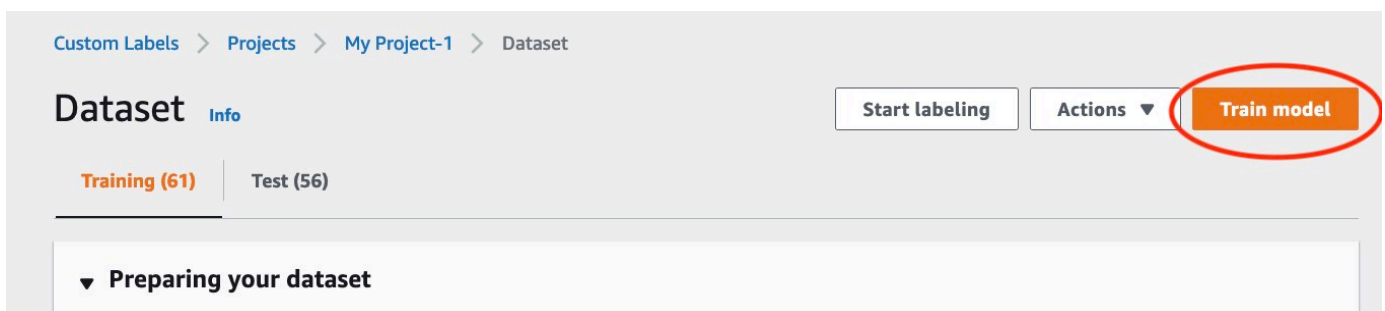
7. すべてのイメージに必要なラベルが付けられるまで、ラベル付けを繰り返します。
8. [テスト] タブを選択します。
9. ステップを繰り返して、テストデータセットのイメージにイメージレベルのラベルを割り当てます。

## ステップ 7: モデルのトレーニング

以下のステップに従って、モデルをトレーニングします。詳細については、「[Amazon Rekognition Custom Labels モデルをトレーニングする](#)」を参照してください。

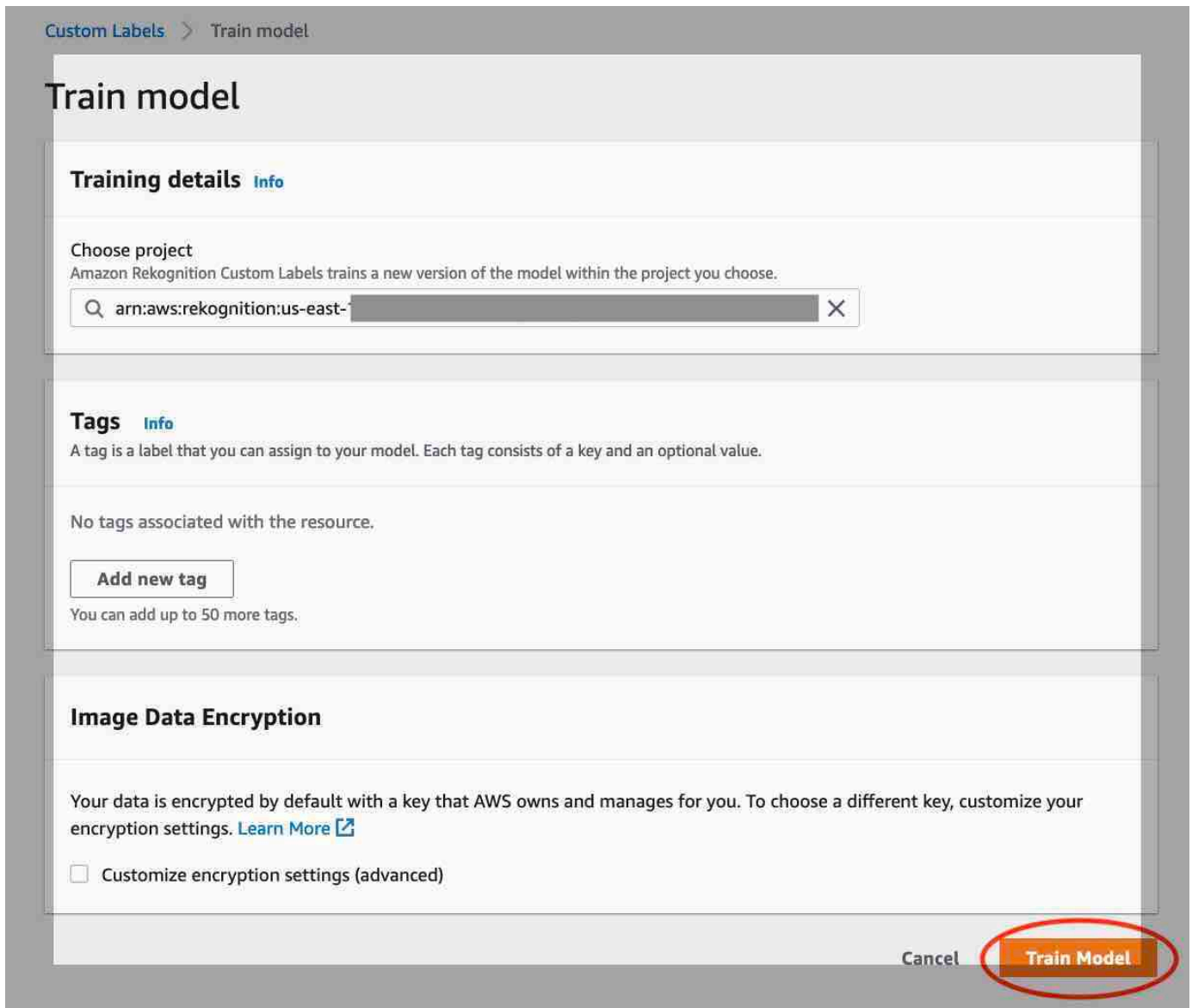
モデルをトレーニングするには (コンソール)

1. [データセット] ページで [モデルをトレーニング] を選択します。



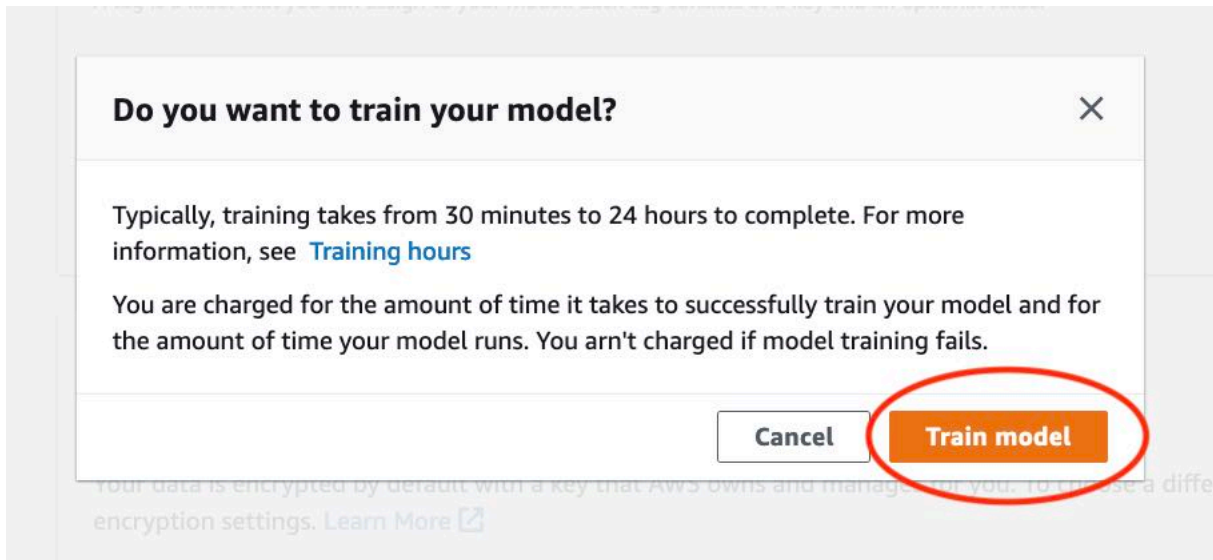


2. [モデルをトレーニング] ページで、[モデルをトレーニング] を選択します。プロジェクトの Amazon リソースネーム (ARN) は、[プロジェクトを選択] 編集ボックスにあります。



3. [モデルをトレーニングしますか?] ダイアログボックスで、[モデルをトレーニング] を選択します。






- プロジェクトページの [モデル] セクションには、トレーニングが進行中であることが表示されます。モデルバージョンの Model Status 列を表示して、現在のステータスを確認できます。モデルのトレーニングは、完了までに時間がかかります。

Custom Labels > Projects > My-Project-1

## My-Project-1 Info

### How it works


#### Creating your dataset



**1. Create dataset**  
A dataset is a collection of images, and image labels, that you use to train or test a model.

Created


#### Label images



**2. Label images**  
Labels identify objects, scenes, or concepts on an entire image, or they identify object locations on an image.

Label images


#### Training your model



**3. Train model**  
Depending on the training dataset, the training model finds image-level scenes and concepts, or it finds object locations.

Train model

#### Evaluating your model



**4. Check performance metrics**  
Performance metrics tell you if your model needs additional training before you can use it.

Check metrics

### Project details

Project name	Created	Dataset	Models
My-Project-1	October 04, 2021 at 13:05:06 (UTC-07:00)	↳	1

### Models (1)

Delete model Download validation results ▾

<input type="checkbox"/>	Name	Date created	Training dataset	Test dataset	Model performance (F1 score)	Model status	Status message
<input type="checkbox"/>	My-Project-1.2021-10-04T13.52.53	October 04, 2021			N/A	<b>TRAINING_IN_PROGRESS</b>	The model is being trained.

5. トレーニングが完了したら、モデル名を選択します。モデルのステータスが [TRAINING\_COMPLETED] になったらトレーニングは終了します。

rooms\_19 Info Delete project

**Create datasets**  
To train a model, you create a training dataset and a test dataset. A dataset is a collection of images labeled with the objects or scenes that you want to find. You create a dataset to train your model first. Later, you create another dataset to test your model.

### Models (1)

Delete model Download validation results ▾ Train new model

<input type="checkbox"/>	Name	Date created	Training dataset	Testing dataset	Model performance	Model status	Status message
<input type="checkbox"/>	<b>rooms_19.2021-07-13T10.36.30</b>	July 13, 2021	rooms_19_training_dataset	rooms_19_test_dataset	0.902	<b>TRAINING_COMPLETED</b>	The model is ready to run.

6. [評価] ボタンを選択すると、評価結果が表示されます。モデルの評価の詳細については、「[トレーニング済み Amazon Rekognition Custom Labels の改善](#)」を参照してください。
7. [テスト結果を表示] を選択すると、個々のテストイメージの結果が表示されます。詳細については、「[モデルを評価するためのメトリクス](#)」を参照してください。

**rooms\_19** Info Delete model

**Evaluate** | Model details | Use Model | Tags

---

**Evaluation results** View test results

<b>F1 score</b> <small>Info</small> 0.902	<b>Average precision</b> <small>Info</small> 0.893	<b>Overall recall</b> <small>Info</small> 0.928
<b>Date completed</b> July 13, 2021 Trained in 1.223 hours	<b>Training dataset</b> 10 labels, 61 images	<b>Testing dataset</b> 10 labels, 56 images

---

**Per label performance (10)**

Find labels < 1 >

Label name ▲	F1 score ▼	Test images ▼	Precision ▼	Recall ▼	Assumed threshold ▼
backyard	0.857	4	1.000	0.750	0.286
bathroom	0.889	9	0.889	0.889	0.185
bedroom	0.900	11	1.000	0.818	0.262
closet	1.000	2	1.000	1.000	0.169
entry_way	1.000	3	1.000	1.000	0.149
floor_plan	1.000	2	1.000	1.000	0.685

8. テスト結果を確認したら、モデル名を選択してモデルページに戻ります。

Custom Labels > Projects > rooms\_19 **rooms\_19.2021-07-13T10.36.30** Performance

**Evaluate image**  
Review the test results of your trained model for individual images. Below each image is information about the model's predicted label compared with the label assigned to the image in the test dataset, noted by result type. You can also filter by label and result types.

**Filter by label**


Choose labels  
Choose labels to filter images

True positive  
 False positive  
 False negative

**Images (56)** Info


< 1 2 3 4 ... >

**backyard2.jpeg**



Labels	Confidence
front_yard False positive	30.3%
backyard False negative	21.6%

**backyard4.jpeg**



Labels	Confidence
backyard True positive	46.3%

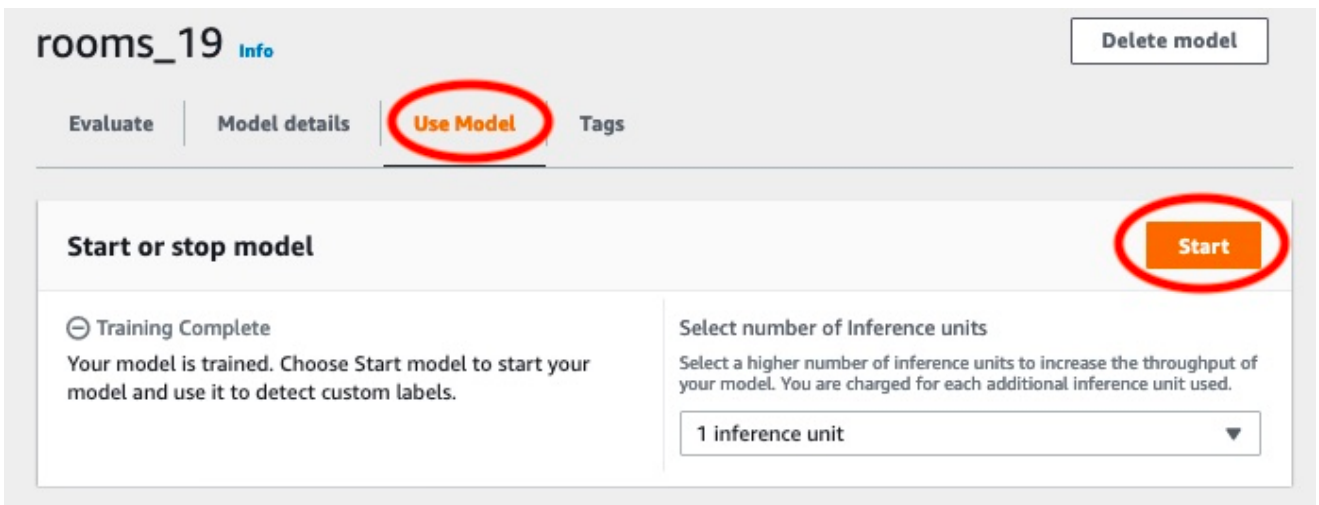
## ステップ 8: モデルをスタートする

このステップではモデルを開始します。モデルの開始後、モデルを使用してイメージを分析できます。

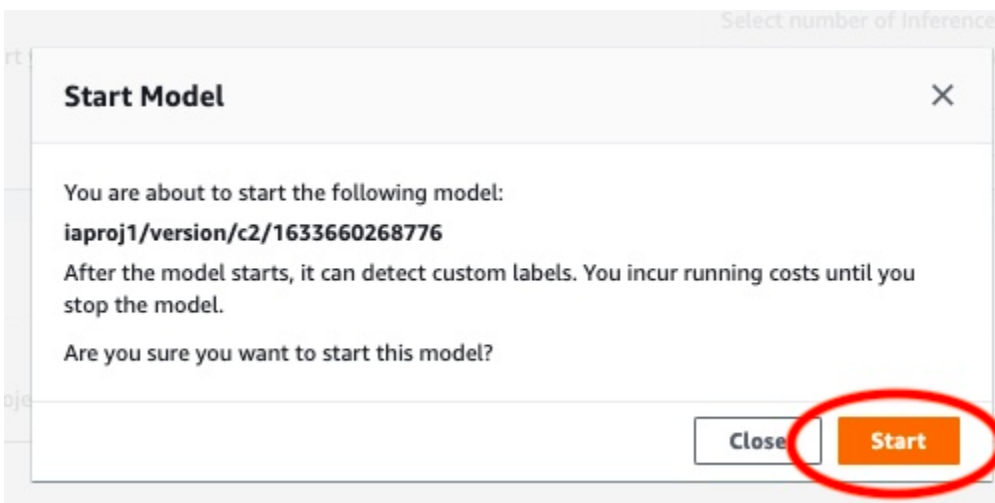
モデルの稼働時間に応じて課金されます。イメージを分析する必要がない場合は、モデルを停止してください。モデルは後で再起動できます。詳細については、「[トレーニング済み Amazon Rekognition Custom Labels の実行](#)」を参照してください。

モデルを開始するには

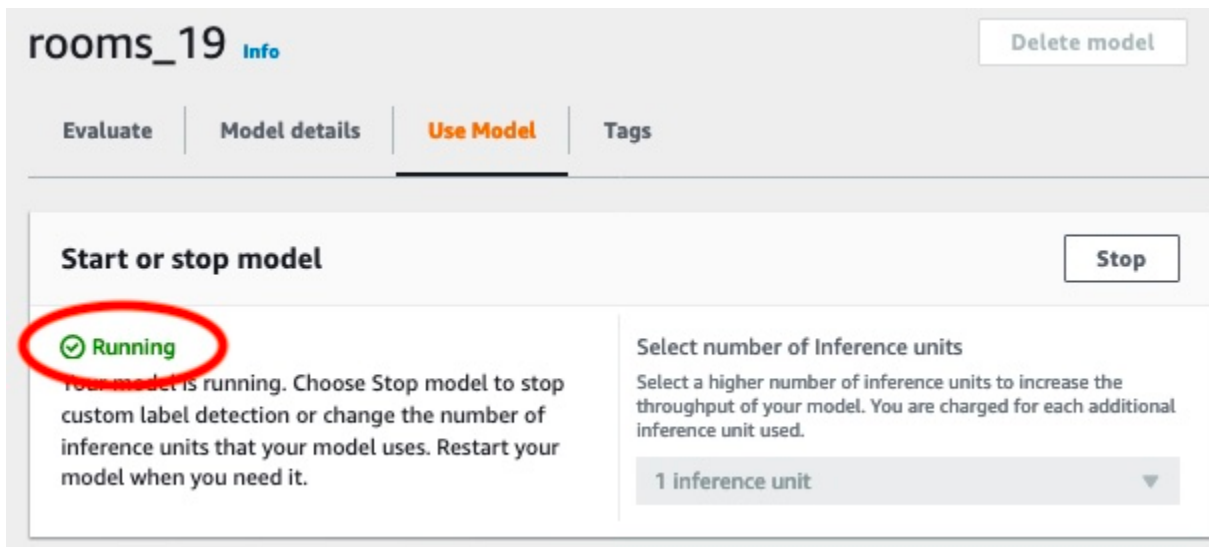
1. モデルページで [モデルを使用] タブを選択します。
2. [モデルの開始または停止] セクションで、次の操作を行います。
  - a. [開始] を選択します。



- b. [モデルを開始] ダイアログボックスで、[開始] を選択します。



3. モデルが実行されるまで待ちます。[モデルの開始または停止] セクションのステータスが [実行中] の場合、モデルは実行中です。



## ステップ 9: モデルを使用してイメージを分析する

イメージを分析するには、[DetectCustomLabels](#) API を呼び出します。このステップでは、`detect-custom-labels` AWS Command Line Interface (AWS CLI) コマンドを使用してサンプルイメージを分析します。AWS CLI コマンドは Amazon Rekognition Custom Labels コンソールから取得します。コンソールはモデルを使用するように AWS CLI コマンドを設定します。Amazon S3 バケットに保存されているイメージを指定するだけです。

### Note

コンソールには Python サンプルコードも用意されています。

`detect-custom-labels` からの出力には、イメージ内のラベルのリスト、境界ボックス (モデルがオブジェクトの位置を検出した場合)、予測の精度に対するモデルの信頼度が含まれます。

詳細については、「[トレーニングされたモデルによるイメージの分析](#)」を参照してください。

イメージを分析するには (コンソール)

1. まだセットアップしていない場合は、AWS CLI をセットアップしてください。手順については、「[the section called “ステップ 4: AWS CLI と AWS SDKsを設定する”](#)」を参照してください。
2. [モデルを使用] タブを選択し、[API コード] を選択します。

The screenshot shows the AWS Rekognition console interface for a custom label model named 'rooms\_19'. At the top, there are tabs for 'Evaluate', 'Model details', 'Use Model', and 'Tags'. The 'Use Model' tab is circled in red. To the right of the tabs is a 'Delete model' button. Below the tabs, there is a 'Start or stop model' section with a 'Stop' button. The status is 'Running', and there is a section for 'Select number of Inference units' with a dropdown menu set to '1 inference unit'. Below this is a 'Use your model' section with an input field for 'Amazon Resource Name (ARN)'. At the bottom of this section, the 'API Code' link is circled in red.

3. [AWS CLI コマンド] を選択します。
4. [画像を分析する] セクションで、`detect-custom-labels` を呼び出す AWS CLI コマンドをコピーします。



**Use your model**

Amazon Resource Name (ARN)

▼ API Code

Use your model rooms\_ [ ] by calling the following AWS CLI commands or Python scripts. You can start and stop the model, and analyze custom labels in new images.

AWS CLI command

Python

**Start model**  
Command used to start the rooms\_ [ ] model.

```
1 aws rekognition start-project-version \  
2 --project-version-arn "arn:aws:rekognition:us-east-1:[ ]" \  
3 --min-inference-units 1 \  
4 --region us-east-1
```

**Analyze image**  
Command used to use analyze an image with the rooms\_ [ ] model. Replace MY\_BUCKET and PATH\_TO\_MY\_IMAGE with your S3 bucket name and image path.

```
1 aws rekognition detect-custom-labels \  
2 --project-version-arn "arn:aws:rekognition:us-east-1:[ ]" \  
3 --image '{"S3Object": {"Bucket": "MY_BUCKET", "Name": "PATH_TO_MY_IMAGE"}}' \  
4 --region us-east-1
```

5. 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。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。Rooms プロジェクトのイメージを使用している場合は、[ステップ 1: イメージを集める](#) 内の別のフォルダに移動したイメージの 1 つを使用してください。
6. コマンドプロンプトで、前のステップでコピーした AWS CLI コマンドを入力します。次の例のようになります。

--project-version-arn の値は、モデルの Amazon リソースネーム (ARN) になるはずですが、--region の値は、モデルを作成した AWS リージョンであるはずですが。

MY\_BUCKET と PATH\_TO\_MY\_IMAGE を前のステップで使用した Amazon S3 バケットとイメージに変更します。

[custom-labels-access](#) プロファイルを使用して認証情報を取得する場合は、`--profile custom-labels-access` パラメータを追加します。

```
aws rekognition detect-custom-labels \  
  --project-version-arn "model_arn" \  
  --image '{"S3Object": {"Bucket": "MY_BUCKET", "Name": "PATH_TO_MY_IMAGE"}}' \  
  --region us-east-1 \  
  --profile custom-labels-access
```

AWS CLI コマンドの JSON 出力は次のようになります。Name はモデルが見つけたイメージレベルのラベルの名前です。Confidence (0-100) は予測の精度に対するモデルの信頼度です。

```
{  
  "CustomLabels": [  
    {  
      "Name": "living_space",  
      "Confidence": 83.41299819946289  
    }  
  ]  
}
```

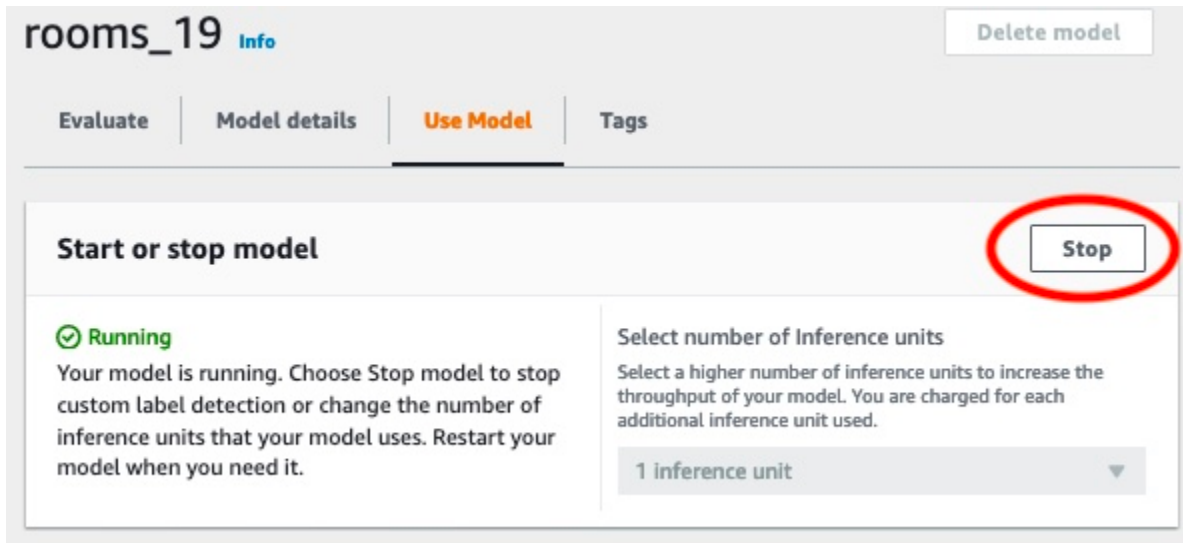
7. 引き続きモデルを使用して他のイメージを分析してください。使用しなくなったモデルは停止してください。

## ステップ 10: モデルを停止する

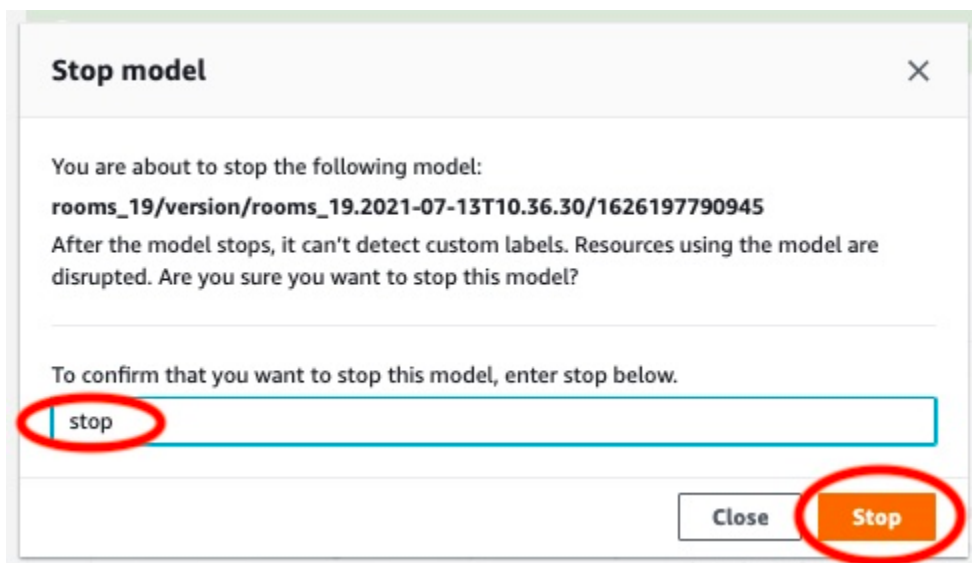
このステップではモデルの実行を停止します。モデルの稼働時間に応じて課金されます。モデルの使用を終了した場合は、停止する必要があります。

モデルを停止するには

1. [モデルの開始または停止] セクションで、[停止] を選択します。



2. [モデルを停止] ダイアログボックスで、[停止] と入力し、モデルを停止することを確認します。



3. [停止] を選択してモデルを停止します。[モデルの開始または停止] セクションのステータスが [停止済み] になると、モデルは停止します。

**rooms\_19** Info
Delete model

Evaluate
Model details
Use Model
Tags

---

### Start or stop model

⊖ Stopped

Your model isn't running. To start running your model, choose Start model or use the example code in Use your model. You can then use your model to find custom labels in images.

#### Select number of Inference units

Select a higher number of inference units to increase the throughput of your model. You are charged for each additional inference unit used.

1 inference unit
▼

Start

# Amazon Rekognition Custom Labels モデルの作成

モデルとは、ビジネス特有の概念、シーン、オブジェクトを検索するためにトレーニングするソフトウェアのことです。Amazon Rekognition Custom Labels コンソールまたは AWS SDK を使用してモデルを作成できます。Amazon Rekognition Custom Labels のモデルを作成する前に、「[Amazon Rekognition Custom Labels について](#)」を参照することを推奨します。

このセクションでは、プロジェクトの作成、さまざまなモデルタイプの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、モデルのトレーニングに関するコンソールと SDK の情報を提供します。この後のセクションでは、モデルを改善して使用方法について説明します。コンソールで特定のタイプのモデルを作成し、使用方法のチュートリアルについては、「[チュートリアル: イメージの分類](#)」を参照してください。

## トピック

- [プロジェクトの作成](#)
- [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](#)
- [Amazon Rekognition Custom Labels モデルをトレーニングする](#)
- [失敗したモデルトレーニングのデバッグ](#)

## プロジェクトの作成

プロジェクトでは、モデルのモデルバージョン、トレーニングデータセット、テストデータセットを管理します。Amazon Rekognition Custom Labels コンソールまたは API を使用してプロジェクトを作成できます。プロジェクトの削除など、そのほかのプロジェクトタスクについては、「[Amazon Rekognition Custom Labels プロジェクトの管理](#)」を参照してください。

## Amazon Rekognition Custom Labels プロジェクトの作成 (コンソール)

Amazon Rekognition Custom Labels コンソールを使用して、プロジェクトを作成できます。新しい AWS リージョンで初めてコンソールを使用する場合、Amazon Rekognition Custom Labels は AWS アカウントに Amazon S3 バケット (コンソールバケット) を作成するように求めます。このバケットは、プロジェクトファイルを保存するために使用されます。コンソールバケットが作成されない限り、Amazon Rekognition Custom Labels コンソールは使用できません。

Amazon Rekognition Custom Labels コンソールを使用して、プロジェクトを作成できます。

## プロジェクトを作成するには (コンソール)

1. AWS Management Console にサインインし、<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、[カスタムラベルを使用] を選択します。Amazon Rekognition Custom Labels のランディングページが表示されます。
3. Amazon Rekognition Custom Labels のランディングページで、[開始方法] を選択します。
4. 左側のペインで、[プロジェクト] を選択します。
5. [Create Project (プロジェクトの作成)] を選択します。
6. [Project name] (プロジェクト名) にプロジェクトの名前を入力します。
7. [プロジェクトを作成] を選択してプロジェクトを作成します。
8. 「[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](#)」のステップに従って、プロジェクト用のトレーニングデータセットとテストデータセットを作成します。

## Amazon Rekognition Custom Labels プロジェクトの作成 (SDK)

Amazon Rekognition Custom Labels のプロジェクトを作成するには、[CreateProject](#) を呼び出します。このレスポンスは、プロジェクト識別する Amazon リソースネーム (ARN) です。プロジェクトを作成したら、モデルのトレーニング用とテスト用のデータセットを作成します。詳細については、「[イメージ付きの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](#)」を参照してください。

### プロジェクトを作成するには (SDK)

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. プロジェクトを作成するには、次のコードを使用します。

#### AWS CLI

次の例では、プロジェクトを作成し、その ARN を表示しています。

project-name の値を作成するプロジェクトの名前に変更します。

```
aws rekognition create-project --project-name my_project \  
--profile custom-labels-access
```

## Python

次の例では、プロジェクトを作成し、その ARN を表示しています。次のコマンドライン引数を指定します。

- `project_name` - 作成するプロジェクトの名前。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

import argparse
import logging
import boto3

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def create_project(rek_client, project_name):
    """
    Creates an Amazon Rekognition Custom Labels project
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_name: A name for the new project.
    """

    try:
        #Create the project.
        logger.info("Creating project: %s",project_name)

        response=rek_client.create_project(ProjectName=project_name)

        logger.info("project ARN: %s",response['ProjectArn'])

        return response['ProjectArn']

    except ClientError as err:
        logger.exception("Couldn't create project - %s: %s", project_name,
err.response['Error']['Message'])
        raise

def add_arguments(parser):
```



```
"""
Adds command line arguments to the parser.
:param parser: The command line parser.
"""

parser.add_argument(
    "project_name", help="A name for the new project."
)

def main():

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Creating project: {args.project_name}")

        # Create the project.
        session = boto3.Session(profile_name='custom-labels-access')
        rekognition_client = session.client("rekognition")

        project_arn=create_project(rekognition_client,
                                   args.project_name)

        print(f"Finished creating project: {args.project_name}")
        print(f"ARN: {project_arn}")

    except ClientError as err:
        logger.exception("Problem creating project: %s", err)
        print(f"Problem creating project: {err}")

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

## Java V2

次の例では、プロジェクトを作成し、その ARN を表示しています。

次のコマンドライン引数を指定します。

- `project_name` - 作成するプロジェクトの名前。

```
/*
 Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 SPDX-License-Identifier: Apache-2.0
*/
package com.example.rekognition;

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.CreateProjectRequest;
import software.amazon.awssdk.services.rekognition.model.CreateProjectResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

import java.util.logging.Level;
import java.util.logging.Logger;

public class CreateProject {

    public static final Logger logger =
        Logger.getLogger(CreateProject.class.getName());

    public static String createMyProject(RekognitionClient rekClient, String
        projectName) {

        try {

            logger.log(Level.INFO, "Creating project: {0}", projectName);
            CreateProjectRequest createProjectRequest =
                CreateProjectRequest.builder().projectName(projectName).build();

            CreateProjectResponse response =
                rekClient.createProject(createProjectRequest);

            logger.log(Level.INFO, "Project ARN: {0} ", response.projectArn());
```

```
        return response.projectArn();

    } catch (RekognitionException e) {
        logger.log(Level.SEVERE, "Could not create project: {0}",
e.getMessage());
        throw e;
    }

}

public static void main(String[] args) {

    final String USAGE = "\n" + "Usage: " + "<project_name> <bucket> <image>
\n\n" + "Where:\n"
        + "    project_name - A name for the new project\n\n";

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

    String projectName = args[0];
    String projectArn = null;
    ;

    try {

        // Get the Rekognition client.
        RekognitionClient rekClient = RekognitionClient.builder()
            .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
            .region(Region.US_WEST_2)
            .build();

        // Create the project
        projectArn = createMyProject(rekClient, projectName);

        System.out.println(String.format("Created project: %s \nProject ARN:
%s", projectName, projectArn));

        rekClient.close();

    } catch (RekognitionException rekError) {
```

```
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
            rekError.getMessage());
        System.exit(1);
    }

}

}
```

- レスポンスに表示されるプロジェクト ARN の名前を記録しておきます。モデルを作成するために必要となります。
- 「[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 \(SDK\)](#)」のステップに従って、プロジェクト用のトレーニングデータセットとテストデータセットを作成します。

## 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

データセットとは、イメージとそのイメージを説明するラベルの集合のことです。プロジェクトには、トレーニングデータセットとテストデータセットが必要です。Amazon Rekognition Custom Labels は、トレーニングデータセットを使用してモデルをトレーニングします。Amazon Rekognition Custom Labels は、トレーニング後にテストデータセットを使用して、トレーニングされたモデルがどの程度正しいラベルを予測できるかを検証します。

Amazon Rekognition Custom Labels コンソールまたは AWS SDK を使用してデータセットを作成できます。データセットを作成する前に、「[Amazon Rekognition Custom Labels について](#)」を参照することを推奨します。そのほかのデータセットタスクについては、「[データセットの管理](#)」を参照してください。

プロジェクトのトレーニングデータセットとテストデータセットを作成するステップは次のとおりです。

プロジェクト用のトレーニングデータセットとテストデータセットを作成するには

- トレーニングデータセットとテストデータセットに対し、どのようなラベル付けが必要かを判断します。詳細については、「[データセットの目的の設定](#)」を参照してください。
- トレーニングデータセットとテストデータセットのイメージを収集します。詳細については、「[the section called “イメージの準備”](#)」を参照してください。
- トレーニングデータセットとテストデータセットを作成します。詳細については、「[イメージ付きの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](#)」を参照してください。AWS SDK

を使用している場合は、「」を参照してください[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 \(SDK\)](#)。

- 必要に応じて、データセットのイメージにイメージレベルのラベルや境界ボックスを追加します。詳細については、「[イメージにラベルを付ける](#)」を参照してください。

データセットを作成すると、モデルを[トレーニング](#)できます。

## トピック

- [データセットの目的の設定](#)
- [イメージの準備](#)
- [イメージ付きの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](#)
- [イメージにラベルを付ける](#)
- [データセットのデバッグ](#)

## データセットの目的の設定

プロジェクト内のトレーニングデータセットとテストデータセットにどのようにラベルを付けるかによって、作成するモデルのタイプが決まります。Amazon Rekognition Custom Labels を使用すると、次のことが実行するモデルを作成できます。

- [オブジェクト、シーン、概念を検出する](#)
- [オブジェクトの位置の検索](#)
- [ブランドの位置の検索](#)

### オブジェクト、シーン、概念を検出する

モデルによって、イメージ全体に関連付けられているオブジェクト、シーン、概念を分類します。

イメージ分類とマルチラベル分類の2つのタイプの分類モデルを作成できます。どちらのタイプの分類モデルでも、モデルはトレーニングに使用されたラベルのセット全体から一致するラベルを検索します。トレーニングデータセットとテストデータセットで、どちらも最低2つのラベルが必要です。

## 画像分類

このモデルは、イメージを事前定義済みのラベルのセットに帰属するものとして分類します。例えば、イメージに居住スペースが含まれているかどうかを判断するモデルが必要な場合があります。次のイメージには living\_space のイメージレベルのラベルが付いている場合があります。



このタイプのモデルでは、トレーニングデータセットとテストデータセットのイメージに、イメージレベルのラベルをそれぞれ1つ追加します。サンプルプロジェクトについては、「[画像分類](#)」を参照してください。

### マルチラベル分類

このモデルは、花の種類や葉の有無など、イメージを複数のカテゴリに分類します。例えば、次のイメージには mediterranean\_spurge と no\_leaves のイメージレベルのラベルが付いている可能性があります。



このタイプのモデルでは、トレーニングデータセットとテストデータセットのイメージに、各カテゴリの画像レベルのラベルを割り当てます。サンプルプロジェクトについては、「[マルチラベルイメージ分類](#)」を参照してください。

### イメージレベルのラベルの割り当て

イメージが Amazon S3 バケットに保存されている場合は、[フォルダ名](#)を使用すればイメージレベルのラベルを自動的に追加できます。詳細については、「[Amazon S3 バケット](#)」を参照してください。また、データセットを作成した後で、イメージにイメージレベルのラベルを追加できます。詳細については、「[the section called “イメージにイメージレベルのラベルを割り当てる”](#)」を参照してください。必要に応じて、新しいラベルを追加できます。詳細については、「[ラベルの管理](#)」を参照してください。

### オブジェクトの位置の検索

イメージ内のオブジェクトの位置を予測するモデルを作成するには、トレーニングデータセットとテストデータセットのイメージにオブジェクト位置の境界ボックスとラベルを定義します。境界ボックスとは、オブジェクトをぴったりと囲んだボックスのことです。次のイメージは Amazon Echo と Amazon Echo Dot を囲む境界ボックスの例を示しています。各境界ボックスにはラベル (Amazon Echo または Amazon Echo Dot) が割り当てられています。





オブジェクトの位置を検索するには、データセットに少なくとも1つのラベルが必要です。モデルのトレーニング中に、イメージの境界ボックスの外側の領域を表すラベルが自動的に作成されます。

### 境界ボックスの割り当て

データセットを作成する際に、イメージの境界ボックス情報を含めることができます。例えば、境界ボックスを含む SageMaker Ground Truth 形式の [マニフェストファイル](#) をインポートできます。また、データセットの作成後に境界ボックスを追加できます。詳細については、「[境界ボックスによるオブジェクトのラベル付け](#)」を参照してください。必要に応じて、新しいラベルを追加できます。詳細については、「[ラベルの管理](#)」を参照してください。

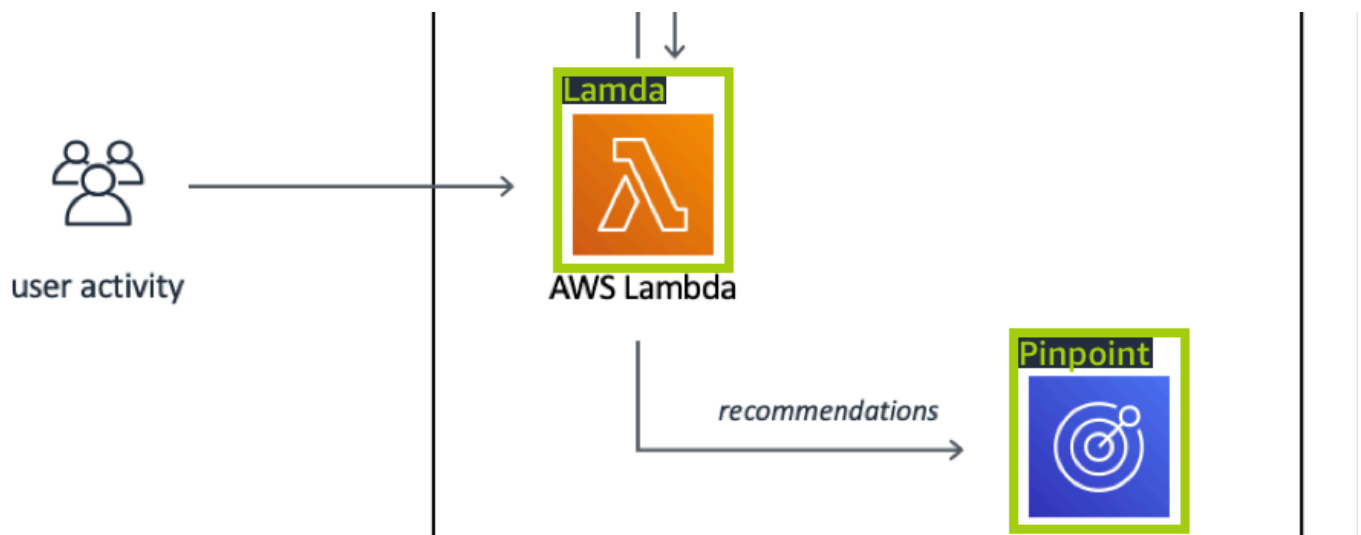
### ブランドの位置の検索

ロゴやアニメーション化されたキャラクターなどのブランドの位置を検索する場合は、トレーニングデータセットのイメージに2つのタイプのイメージを使用できます。

- ロゴのみのイメージ。各イメージには、ロゴ名を表すイメージレベルのラベルが 1 つ必要です。例えば、次のイメージのイメージレベルのラベルは Lambda になります。



- フットボールの試合や建築図など、自然な場所にロゴが入っているイメージ。各トレーニングイメージには、ロゴの各インスタンスを囲む境界ボックスが必要です。例えば、次の画像は、AWS Lambda ロゴと Amazon Pinpoint ロゴを囲むラベル付き境界ボックスを含むアーキテクチャ図を示しています。



トレーニングイメージには、イメージレベルのラベルと境界ボックスを混在させないことを推奨します。

テストイメージには、検索するブランドのインスタンスの周囲に境界ボックスが必要です。トレーニングイメージにラベル付き境界ボックスが含まれている場合のみ、トレーニングデータセットを分割してテストデータセットを作成できます。トレーニングイメージにイメージレベルのラベルしかない場合は、ラベル付き境界ボックスが付いたイメージを含むテストデータセットを作成する必要があります。

まず、ブランドの位置を検知するようにモデルをトレーニングする場合は、[イメージのラベル付け方法に従って 境界ボックスによるオブジェクトのラベル付け](#) と [イメージにイメージレベルのラベルを割り当てる](#) を行います。

[ブランド検出](#) のサンプルプロジェクトでは、Amazon Rekognition Custom Labels がラベル付き境界ボックスを使用して、オブジェクトの位置を検出するモデルをトレーニングする方法を示しています。

## モデルタイプのラベル要件

次の表を使用して、イメージにラベルを付ける方法を決定します。

イメージレベルのラベルと境界ボックスでラベル付きイメージを 1 つのデータセットにまとめることができます。この場合、Amazon Rekognition Custom Labels は、イメージレベルのモデルを作成するか、オブジェクト位置のモデルを作成するかを選択します。

例	トレーニングイメージ	テストイメージ
<a href="#">画像分類</a>	1 つのイメージにつき 1 つのイメージレベルのラベル	1 つのイメージにつき 1 つのイメージレベルのラベル
<a href="#">マルチラベル分類</a>	1 つのイメージにつき複数のイメージレベルのラベル	1 つのイメージにつき複数のイメージレベルのラベル
<a href="#">ブランドの位置の検索</a>	イメージレベルのラベル (ラベル付き境界ボックスも使用できます)	ラベル付き境界ボックス
<a href="#">オブジェクトの位置の検索</a>	ラベル付き境界ボックス	ラベル付き境界ボックス

## イメージの準備

トレーニングデータセットとテストデータセットのイメージには、モデルが検知するオブジェクト、シーン、または概念が含まれています。

イメージのコンテンツには、トレーニングを受けたモデルが識別するイメージを表すさまざまな背景と照明が必要です。

このセクションでは、トレーニングデータセットとテストデータセットのイメージに関する情報を提供します。

## イメージ形式

Amazon Rekognition Custom Labels では、PNG および JPEG 形式のイメージを使用してトレーニングできます。同様に、DetectCustomLabels を使用してカスタムラベルを検知するには、PNG および JPEG 形式のイメージが必要です。

## 入力するイメージの推奨事項

Amazon Rekognition Custom Labels では、モデルのトレーニングとテストにイメージが必要です。イメージを準備するには、以下の事項を考慮してください。

- 作成するモデルの特定のドメインを選択します。例えば、景観用のモデルや機械部品などのオブジェクト用のモデルを選択できます。Amazon Rekognition Custom Labels は、選択したドメインにイメージが存在する場合に最適です。
- モデルをトレーニングするには、少なくとも 10 枚のイメージを使用します。
- イメージは PNG または JPEG 形式である必要があります。
- さまざまな照明、背景、解像度でオブジェクトを撮影したイメージを使用します。
- トレーニングイメージとテストイメージは、モデルで使用するイメージに似ている必要があります。
- イメージに割り当てるラベルを決定します。
- 画像の解像度が十分に大きいことを確認します。詳細については、「[Amazon Rekognition Custom Labels のガイドラインとクォータ](#)」を参照してください。
- オクルージョンによって、検知対象のオブジェクトが不鮮明にならないようにします。
- 背景とのコントラストが十分である画像を使用します。
- 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イメージは、できるだけ使用しないでください。
- イメージの大部分をオブジェクトが占めるイメージを使用します。
- テストデータセット内のイメージがトレーニングデータセット内のイメージに存在しない必要があります。これらには、モデルが分析するように訓練されたオブジェクト、シーン、概念が含まれている必要があります。

## イメージセットのサイズ

Amazon Rekognition Custom Labels は、一連のイメージを使用してモデルをトレーニングします。トレーニングには、少なくとも 10 枚のイメージを使用する必要があります。Amazon Rekognition Custom Labels は、トレーニングイメージとテストイメージをデータセットに保存します。詳細につ

いては、「[イメージ付きの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](#)」を参照してください。

## イメージ付きの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

1つのデータセットを使用するプロジェクトから始めることも、個別のトレーニングデータセットとテストデータセットを持つプロジェクトから始めることもできます。1つのデータセットから始めると、Amazon Rekognition Custom Labels はトレーニング中にデータセットを分割して、プロジェクトのトレーニングデータセット (80%) とテストデータセット (20%) を作成します。Amazon Rekognition Custom Labels にトレーニングとテストに使用するイメージを決定させる場合は、1つのデータセットから始めてください。トレーニング、テスト、パフォーマンスのチューニングを完全に制御するには、トレーニングデータセットとテストデータセットを分けてプロジェクトを開始することをお勧めします。

以下のいずれかの場所からイメージをインポートすることにより、プロジェクトのトレーニングデータセットとテストデータセットを作成できます。

- [Amazon S3 バケット](#)
- [ローカルコンピュータ](#)
- [マニフェストファイル](#)
- [既存のデータセット](#)

トレーニングデータセットとテストデータセットを分けてプロジェクトを開始する場合は、データセットごとに異なるソースの場所を使用できます。

イメージのインポート元によっては、イメージにラベルが付いていない場合があります。例えば、ローカルコンピュータからインポートされたイメージにはラベルは付きません。Amazon SageMaker Ground Truth マニフェストファイルからインポートされたイメージにはラベルが付けられます。Amazon Rekognition Custom Labels コンソールを使用して、ラベルの追加、変更、割り当てを行うことができます。詳細については、「[イメージにラベルを付ける](#)」を参照してください。

イメージのアップロード時にエラーが発生する、イメージが見つからない、イメージにラベルが付いていない場合は、「[失敗したモデルトレーニングのデバッグ](#)」を参照してください。

データセットの詳細については、「[データセットの管理](#)」を参照してください。

### 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 (SDK)

AWS SDK を使用して、トレーニングデータセットとテストデータセットを作成できます。

## トレーニングデータセット

AWS SDK を使用して、次の方法でトレーニングデータセットを作成できます。

- 指定した Amazon Sagemaker 形式のマニフェストファイル [CreateDataset](#) で を使用します。詳細については、「[the section called “マニフェストファイルの作成”](#)」を参照してください。サンプルコードについては、「[SageMaker Ground Truth マニフェストファイル \(SDK\) を使用したデータセットの作成](#)」を参照してください。
- CreateDataset を使用して、既存の Amazon Rekognition Custom Labels データセットをコピーします。サンプルコードについては、「[既存のデータセットを使用したデータセットの作成 \(SDK\)](#)」を参照してください。
- 空のデータセット CreateDataset を作成し、後で データセットエントリを追加します [UpdateDatasetEntries](#)。空のデータセットを作成する方法については、「[データセットをプロジェクトに追加する](#)」を参照してください。データセットにイメージを追加する方法については、「[イメージの追加 \(SDK\)](#)」を参照してください。モデルをトレーニングする前に、データセットエントリを追加する必要があります。

## テストデータセット

AWS SDK を使用して、次の方法でテストデータセットを作成できます。

- 指定した Amazon Sagemaker 形式のマニフェストファイル [CreateDataset](#) で を使用します。詳細については、「[the section called “マニフェストファイルの作成”](#)」を参照してください。サンプルコードについては、「[SageMaker Ground Truth マニフェストファイル \(SDK\) を使用したデータセットの作成](#)」を参照してください。
- CreateDataset を使用して、既存の Amazon Rekognition Custom Labels データセットをコピーします。サンプルコードについては、「[既存のデータセットを使用したデータセットの作成 \(SDK\)](#)」を参照してください。
- CreateDataset で空のデータセットを作成し、後で UpdateDatasetEntries でデータセットエントリを追加します。空のデータセットを作成する方法については、「[データセットをプロジェクトに追加する](#)」を参照してください。データセットにイメージを追加する方法については、「[イメージの追加 \(SDK\)](#)」を参照してください。モデルをトレーニングする前に、データセットエントリを追加する必要があります。
- トレーニングデータセットとテストデータセットを分割します。まず、CreateDataset で空のテストデータセットを作成します。次に、 を呼び出して、トレーニングデータセットエントリの 20% をテストデータセットに移動します [DistributeDatasetEntries](#)。空のデータセットを作成する方法については、「[データセットをプロジェクトに追加する \(SDK\)](#)」を参照してください。トレー



ニングデータセットを分割する方法については、「[トレーニングデータセットの分散 \(SDK\)](#)」を参照してください。

## Amazon S3 バケット

イメージは Amazon S3 バケットからインポートされます。コンソールバケット、または AWS アカウント内の別の Amazon S3 バケットを使用できます。コンソールバケットを使用している場合、必要な権限は既に設定されています。コンソールバケットを使用していない場合は、「[外部の Amazon S3 バケットへのアクセス](#)」を参照してください。

### Note

AWS SDK を使用して、Amazon S3 バケット内の画像から直接データセットを作成することはできません。代わりに、イメージのソースロケーションを参照するマニフェストファイルを作成してください。詳細については、「[マニフェストファイル](#)」を参照してください。

データセットの作成中に、画像を含むフォルダの名前に基づいて画像にラベル名を割り当てることを選択できます。フォルダは、データセット作成時に S3 フォルダの場所で指定した Amazon S3 フォルダパスの子である必要があります。データセットの作成については、「[S3 バケットのイメージをインポートしてデータセットを作成する](#)」を参照してください。

例えば、Amazon S3 バケットに次のようなフォルダ構造があるとします。Amazon S3 フォルダの場所を S3-bucket/alexa-devices として指定した場合、echo フォルダ内のイメージには、echo というラベルが割り当てられます。同様に、echo-dot フォルダ内のイメージには echo-dot というラベルが割り当てられます。より深い子フォルダの名前は、画像のラベル付けには使用されません。代わりに、Amazon S3 フォルダの場所の適切な子フォルダが使用されます。例えば、フォルダ内のイメージ white-echo-dots には、ラベル echo-dot が割り当てられます。S3 フォルダの場所 (Alexa-devices) のレベルにあるイメージにはラベルは割り当てられていません。

フォルダ構造がより深いフォルダであれば、S3 フォルダのより深い場所を指定してイメージにラベルを付けることができます。例えば、S3-bucket/alexa-devices/echo-dot を指定すると、フォルダ内のイメージには というラベル white-echo-dot が付けられます white-echo-dot。echo など、指定した S3 フォルダの場所以外のイメージはインポートされません。

```
S3-bucket
### alexa-devices
    ### echo
        #   ### echo-image-1.png
```



```
#   ### echo-image-2.png
#   ### .
#   ### .
### echo-dot
    ### white-echo-dot
    #   ### white-echo-dot-image-1.png
    #   ### white-echo-dot-image-2.png
    #
    ### echo-dot-image-1.png
    ### echo-dot-image-2.png
    ### .
    ### .
```

現在の AWS リージョンでコンソールを初めて開いたときに、Amazon Rekognition によって作成された Amazon S3 バケット (コンソールバケット) を使用することをお勧めします。Amazon Rekognition 使用している Amazon S3 バケットがコンソールバケットと異なる (外部) 場合、データセットの作成中に、コンソールから適切な権限を設定するように求められます。詳細については、「[the section called “ステップ 2: コンソールのアクセス許可のセットアップ”](#)」を参照してください。

### S3 バケットのイメージをインポートしてデータセットを作成する

次の手順では、コンソール S3 バケットに保存されているイメージを使用してデータセットを作成する方法を示しています。イメージには、保存されているフォルダの名前で自動的にラベルが付けられます。

イメージをインポートした後は、データセットのギャラリーページからのイメージの追加、ラベルの割り当て、境界ボックスの追加を行うことができます。詳細については、「[イメージにラベルを付ける](#)」を参照してください。

Amazon Simple Storage Service バケットにイメージをアップロードします。

1. ローカルファイルシステムにフォルダを作成します。Alexa-devices などのフォルダ名を使用してください。
2. 作成したフォルダ内に、使用する各ラベルの名前を付けたフォルダを作成します。例えば、echo や echo-dot などです。フォルダの構成は次のようにします。

```
alex-devices
### echo
#   ### echo-image-1.png
```

```
# ### echo-image-2.png
# ### .
# ### .
### echo-dot
### echo-dot-image-1.png
### echo-dot-image-2.png
### .
### .
```

- ラベルに対応するイメージを、同じラベル名のフォルダに配置します。
- にサインイン AWS Management Console し、<https://console.aws.amazon.com/s3/> で Amazon S3 コンソールを開きます。
- 初回セットアップ時に Amazon Rekognition Custom Labels が作成した Amazon S3 バケット (コンソールバケット) に、ステップ 1 で作成した [フォルダを追加](#) します。詳細については、「[Amazon Rekognition Custom Labels プロジェクトの管理](#)」を参照してください。
- Amazon Rekognition コンソールを <https://console.aws.amazon.com/rekognition/> で開きます。
- [カスタムラベルを使用] を選択します。
- [開始方法] を選択します。
- 左側のナビゲーションペインで、[プロジェクト] を選択します。
- 「プロジェクト」ページで、データセットを追加したいプロジェクトを選択します。プロジェクトの詳細ページが表示されます。
- [データセットを作成] を選択します。「データセットを作成」ページが表示されます。
- [設定の開始] で、[1 つのデータセットで開始] または [トレーニングデータセットで開始] を選択します。より高品質のモデルを作成するには、トレーニングデータセットとテストデータセットを別々に始めることを推奨します。

### Single dataset

- [トレーニングデータセットの詳細] セクションで、[S3 バケットからのイメージのインポート] を選択します。
- [トレーニングデータセットの詳細] セクションで、[イメージソース設定] セクションにステップ 13~15 の情報を入力します。

### Separate training and test datasets

- [トレーニングデータセットの詳細] セクションで、[S3 バケットからのイメージのインポート] を選択します。

- b. [トレーニングデータセットの詳細]セクションで、[イメージソース設定] セクションにステップ 13~15 の情報を入力します。
  - c. [テストデータセットの詳細] セクションで、[S3 バケットからのイメージのインポート] を選択します。
  - d. [テストデータセットの詳細]セクションで、[イメージソース設定] セクションにステップ 13~15 の情報を入力します。
13. [Amazon S3 バケットから画像をインポート] を選択します。
  14. S3 URI には、Amazon S3 バケットの場所とフォルダのパスを入力します。
  15. [フォルダに基づいて画像にラベルを自動的にアタッチ] を選択します。
  16. [データセットを作成] を選択します。プロジェクトのデータセットページが開きます。
  17. ラベルの追加または変更の必要がある場合は、[イメージにラベルを付ける](#) を実行してください。
  18. 「[モデルのトレーニング \(コンソール\)](#)」の手順に従って、モデルをトレーニングします。

## ローカルコンピュータ

イメージは、コンピュータから直接ロードされます。同時にアップロードできる画像は、30 枚までです。

アップロードしたイメージにはラベルは付いていません。詳細については、「[イメージにラベルを付ける](#)」を参照してください。アップロードするイメージが多い場合は、Amazon S3 バケットの使用を検討してください。詳細については、「[Amazon S3 バケット](#)」を参照してください。

### Note

AWS SDK を使用してローカルイメージを含むデータセットを作成することはできません。代わりに、マニフェストファイルを作成し、Amazon S3 バケットにイメージをアップロードします。詳細については、「[マニフェストファイル](#)」を参照してください。

ローカルコンピュータ上の画像を使用してデータセットを作成するには (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. [カスタムラベルを使用] を選択します。
3. [開始方法] を選択します。

4. 左側のナビゲーションペインで、[プロジェクト] を選択します。
5. 「プロジェクト」ページで、データセットを追加したいプロジェクトを選択します。プロジェクトの詳細ページが表示されます。
6. [データセットを作成] を選択します。「データセットを作成」ページが表示されます。
7. [設定の開始] で、[1 つのデータセットで開始] または [トレーニングデータセットで開始] を選択します。より高品質のモデルを作成するには、トレーニングデータセットとテストデータセットを別々に始めることを推奨します。

### Single dataset

- a. [トレーニングデータセットの詳細] セクションで、[コンピュータから画像をアップロード] を選択します。
- b. [データセットを作成] を選択します。
- c. データセットページで、[画像を追加] を選択します。
- d. コンピュータファイルからデータセットにアップロードする画像を選択します。画像をドラッグするか、ローカルコンピュータからアップロードする画像を選択できます。
- e. [画像をアップロード] を選択します。

### Separate training and test datasets

- a. [トレーニングデータセット詳細] セクションで、[コンピュータから画像をアップロード] を選択します。
- b. [テストデータセット詳細] セクションで、[コンピュータから画像をアップロード] を選択します。

#### Note

トレーニングデータセットとテストデータセットは、異なる画像ソースを持つことができます。

- c. [データセットを作成] を選択します。プロジェクトのデータセットページが表示され、それぞれのデータセットの[トレーニング] タブと[テスト] タブが表示されます。
- d. [アクション] を選択し、[トレーニングデータセットに画像を追加] をクリックします。
- e. データセットにアップロードする画像を選択します。画像をドラッグするか、ローカルコンピュータからアップロードする画像を選択できます。

- f. [画像をアップロード] を選択します。
  - g. ステップ 5e ~ 5g を繰り返します。ステップ 5e で、[アクション] を選択します。[テストデータセットに画像を追加] をクリックします。
8. 「[イメージにラベルを付ける](#)」の手順に従って、画像にラベルを付けます。
  9. 「[モデルのトレーニング \(コンソール\)](#)」の手順に従って、モデルをトレーニングします。

## マニフェストファイル

Amazon SageMaker Ground Truth 形式のマニフェストファイルを使用してデータセットを作成できます。Amazon SageMaker Ground Truth ジョブからマニフェストファイルを使用できます。イメージとラベルが SageMaker Ground Truth マニフェストファイルの形式でない場合は、SageMaker 形式マニフェストファイルを作成し、それを使用してラベル付きイメージをインポートできます。

### トピック

- [SageMaker Ground Truth マニフェストファイルを使用したデータセットの作成 \(コンソール\)](#)
- [SageMaker Ground Truth マニフェストファイル \(SDK\) を使用したデータセットの作成](#)
- [Amazon SageMaker Ground Truth ジョブ](#)
- [マニフェストファイルの作成](#)
- [他のデータセット形式をマニフェストファイルに変換する](#)

### SageMaker Ground Truth マニフェストファイルを使用したデータセットの作成 (コンソール)

次の手順は、SageMaker Ground Truth 形式のマニフェストファイルを使用してデータセットを作成する方法を示しています。

1. 次のいずれかの方法で、トレーニングデータセットのマニフェストファイルを作成します。
  - 「」の手順に従って、SageMaker GroundTruth ジョブを含むマニフェストファイルを作成します [Amazon SageMaker Ground Truth ジョブ](#)。
  - 「[マニフェストファイルの作成](#)」の手順に従って、独自のマニフェストファイルを作成します。

テストデータセットを作成する場合は、ステップ 1 を繰り返してテストデータセットを作成します。

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

3. [カスタムラベルを使用] を選択します。
4. [開始方法] を選択します。
5. 左側のナビゲーションペインで、[プロジェクト] を選択します。
6. 「プロジェクト」ページで、データセットを追加したいプロジェクトを選択します。プロジェクトの詳細ページが表示されます。
7. [データセットを作成] を選択します。「データセットを作成」ページが表示されます。
8. [設定の開始] で、[1 つのデータセットで開始] または [トレーニングデータセットで開始] を選択します。より高品質のモデルを作成するには、トレーニングデータセットとテストデータセットを別々に始めることを推奨します。

### Single dataset

- a. 「トレーニングデータセットの詳細」セクションで、SageMaker 「Ground Truth でラベル付けされたイメージのインポート」を選択します。
- b. [マニフェストファイルの場所] には、ステップ 1 で作成したマニフェストファイルの場所を入力します。
- c. [データセットを作成] を選択します。プロジェクトのデータセットページが開きます。

### Separate training and test datasets

- a. 「トレーニングデータセットの詳細」セクションで、SageMaker 「Ground Truth でラベル付けされたイメージのインポート」を選択します。
- b. [マニフェストファイルの場所] には、ステップ 1 で作成したトレーニングデータセットのマニフェストファイルの場所を入力します。
- c. 「データセットの詳細をテストする」セクションで、SageMaker 「Ground Truth でラベル付けされた画像のインポート」を選択します。

#### Note

トレーニングデータセットとテストデータセットは、異なる画像ソースを持つことができます。

- d. [マニフェストファイルの場所] には、ステップ 1 で作成したテストデータセットのマニフェストファイルの場所を入力します。
- e. [データセットを作成] を選択します。プロジェクトのデータセットページが開きます。

9. ラベルの追加または変更の必要がある場合は、[イメージにラベルを付ける](#) を実行してください。
10. 「[モデルのトレーニング \(コンソール\)](#)」の手順に従って、モデルをトレーニングします。

## SageMaker Ground Truth マニフェストファイル (SDK) を使用したデータセットの作成

次の手順では、[CreateDataset](#) API を使用してマニフェストファイルからトレーニングデータセットまたはテストデータセットを作成する方法を示します。

[SageMaker Ground Truth ジョブ](#) からの出力などの既存のマニフェストファイルを使用するか、独自の[マニフェストファイル](#)を作成できます。

1. 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 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. 次のいずれかの方法で、トレーニングデータセットのマニフェストファイルを作成します。
  - 「」の手順に従って、SageMaker GroundTruth ジョブを含むマニフェストファイルを作成します [Amazon SageMaker Ground Truth ジョブ](#)。
  - 「[マニフェストファイルの作成](#)」の手順に従って、独自のマニフェストファイルを作成します。

テストデータセットを作成する場合は、ステップ 2 を繰り返してテストデータセットを作成します。

3. 次のサンプルコードを使用して、トレーニングデータセットとテストデータセットを作成します。

### AWS CLI

次のコードを使用してデータセットを作成します。以下に置き換えます:

- `project_arn` - テストデータセットを追加するプロジェクトの ARN。
- `type` - 作成するデータセットのタイプ (トレーニングまたはテスト)
- `bucket` - データセットのマニフェストファイルを含むバケット。
- `manifest_file` - マニフェストファイルのパスとファイル名。



```
aws rekognition create-dataset --project-arn project_arn \  
  --dataset-type type \  
  --dataset-source '{ "GroundTruthManifest": { "S3Object": { "Bucket": "bucket",  
"Name": "manifest_file" } } }' \  
  --profile custom-labels-access
```

## Python

次の値を使用してデータセットを作成します。次のコマンドラインパラメータを指定します。

- `project_arn` - テストデータセットを追加するプロジェクトの ARN。
- `dataset_type` - 作成するデータセットのタイプ (train または test)。
- `bucket` - データセットのマニフェストファイルを含むバケット。
- `manifest_file` - マニフェストファイルのパスとファイル名。

```
#Copyright 2023 Amazon.com, Inc. or its affiliates. All Rights Reserved.  
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/  
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-  
SAMPLECODE.)
```

```
import argparse  
import logging  
import time  
import json  
import boto3  
from botocore.exceptions import ClientError
```

```
logger = logging.getLogger(__name__)
```

```
def create_dataset(rek_client, project_arn, dataset_type, bucket,  
manifest_file):
```

```
    """
```

```
    Creates an Amazon Rekognition Custom Labels dataset.
```

```
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
```

```
    :param project_arn: The ARN of the project in which you want to create a  
dataset.
```

```
:param dataset_type: The type of the dataset that you want to create (train
or test).
:param bucket: The S3 bucket that contains the manifest file.
:param manifest_file: The path and filename of the manifest file.
"""

try:
    #Create the project
    logger.info("Creating %s dataset for project %s",dataset_type,
project_arn)

    dataset_type = dataset_type.upper()

    dataset_source = json.loads(
        '{ "GroundTruthManifest": { "S3Object": { "Bucket": "'
        + bucket
        + '", "Name": "'
        + manifest_file
        + '" } } }'
    )

    response = rek_client.create_dataset(
        ProjectArn=project_arn, DatasetType=dataset_type,
DatasetSource=dataset_source
    )

    dataset_arn=response['DatasetArn']

    logger.info("dataset ARN: %s",dataset_arn)

    finished=False
    while finished is False:

        dataset=rek_client.describe_dataset(DatasetArn=dataset_arn)

        status=dataset['DatasetDescription']['Status']

        if status == "CREATE_IN_PROGRESS":
            logger.info("Creating dataset: %s ",dataset_arn)
            time.sleep(5)
            continue

        if status == "CREATE_COMPLETE":
            logger.info("Dataset created: %s", dataset_arn)
```

```
        finished=True
        continue

        if status == "CREATE_FAILED":
            error_message = f"Dataset creation failed: {status} :
{dataset_arn}"
            logger.exception(error_message)
            raise Exception (error_message)

            error_message = f"Failed. Unexpected state for dataset creation:
{status} : {dataset_arn}"
            logger.exception(error_message)
            raise Exception(error_message)

        return dataset_arn

    except ClientError as err:
        logger.exception("Couldn't create dataset: %s",err.response['Error']
['Message'])
        raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The ARN of the project in which you want to create
the dataset."
    )

    parser.add_argument(
        "dataset_type", help="The type of the dataset that you want to create
(train or test).")
    )

    parser.add_argument(
        "bucket", help="The S3 bucket that contains the manifest file."
    )

    parser.add_argument(
        "manifest_file", help="The path and filename of the manifest file."
    )
```

```
)

def main():

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    try:

        #Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Creating {args.dataset_type} dataset for project
{args.project_arn}")

        #Create the dataset.
        session = boto3.Session(profile_name='custom-labels-access')
        rekognition_client = session.client("rekognition")

        dataset_arn=create_dataset(rekognition_client,
            args.project_arn,
            args.dataset_type,
            args.bucket,
            args.manifest_file)

        print(f"Finished creating dataset: {dataset_arn}")

    except ClientError as err:
        logger.exception("Problem creating dataset: %s", err)
        print(f"Problem creating dataset: {err}")

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

## Java V2

次の値を使用してデータセットを作成します。次のコマンドラインパラメータを指定します。

- `project_arn` - テストデータセットを追加するプロジェクトの ARN。
- `dataset_type` - 作成するデータセットのタイプ (train または test)。
- `bucket` - データセットのマニフェストファイルを含むバケット。
- `manifest_file` - マニフェストファイルのパスとファイル名。

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
 */

package com.example.rekognition;

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.CreateDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.CreateDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.DatasetDescription;
import software.amazon.awssdk.services.rekognition.model.DatasetSource;
import software.amazon.awssdk.services.rekognition.model.DatasetStatus;
import software.amazon.awssdk.services.rekognition.model.DatasetType;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.GroundTruthManifest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.S3Object;

import java.util.logging.Level;
import java.util.logging.Logger;

public class CreateDatasetManifestFiles {

    public static final Logger logger =
        Logger.getLogger(CreateDatasetManifestFiles.class.getName());
```

```
public static String createMyDataset(RekognitionClient rekClient, String
projectArn, String datasetType,
    String bucket, String name) throws Exception, RekognitionException {

    try {

        logger.log(Level.INFO, "Creating {0} dataset for project : {1} from
s3://{2}/{3} ",
            new Object[] { datasetType, projectArn, bucket, name });

        DatasetType requestDatasetType = null;

        switch (datasetType) {
        case "train":
            requestDatasetType = DatasetType.TRAIN;
            break;
        case "test":
            requestDatasetType = DatasetType.TEST;
            break;
        default:
            logger.log(Level.SEVERE, "Could not create dataset. Unrecognized
dataset type: {0}", datasetType);
            throw new Exception("Could not create dataset. Unrecognized
dataset type: " + datasetType);
        }

        GroundTruthManifest groundTruthManifest =
GroundTruthManifest.builder()

.s3Object(S3Object.builder().bucket(bucket).name(name).build()).build();

        DatasetSource datasetSource =
DatasetSource.builder().groundTruthManifest(groundTruthManifest).build();

        CreateDatasetRequest createDatasetRequest =
CreateDatasetRequest.builder().projectArn(projectArn)

.datasetType(requestDatasetType).datasetSource(datasetSource).build();

        CreateDatasetResponse response =
rekClient.createDataset(createDatasetRequest);
```

```
        boolean created = false;

        do {

            DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder()
                .datasetArn(response.datasetArn()).build();
            DescribeDatasetResponse describeDatasetResponse =
rekClient.describeDataset(describeDatasetRequest);

            DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();

            DatasetStatus status = datasetDescription.status();

            logger.log(Level.INFO, "Creating dataset ARN: {0} ",
response.datasetArn());

            switch (status) {

                case CREATE_COMPLETE:
                    logger.log(Level.INFO, "Dataset created");
                    created = true;
                    break;

                case CREATE_IN_PROGRESS:
                    Thread.sleep(5000);
                    break;

                case CREATE_FAILED:
                    String error = "Dataset creation failed: " +
datasetDescription.statusAsString() + " "
                        + datasetDescription.statusMessage() + " " +
response.datasetArn();
                    logger.log(Level.SEVERE, error);
                    throw new Exception(error);

                default:
                    String unexpectedError = "Unexpected creation state: " +
datasetDescription.statusAsString() + " "
                        + datasetDescription.statusMessage() + " " +
response.datasetArn();
                    logger.log(Level.SEVERE, unexpectedError);
                    throw new Exception(unexpectedError);
            }
        }
    }
}
```



```
    }

    } while (created == false);

    return response.datasetArn();

} catch (RekognitionException e) {
    logger.log(Level.SEVERE, "Could not create dataset: {0}",
e.getMessage());
    throw e;
}

}

public static void main(String[] args) {

    String datasetType = null;
    String bucket = null;
    String name = null;
    String projectArn = null;
    String datasetArn = null;

    final String USAGE = "\n" + "Usage: " + "<project_arn> <dataset_type>
<dataset_arn>\n\n" + "Where:\n"
        + "    project_arn - the ARN of the project that you want to add
copy the data to.\n\n"
        + "    dataset_type - the type of the dataset that you want to
create (train or test).\n\n"
        + "    bucket - the S3 bucket that contains the manifest file.\n
\n"
        + "    name - the location and name of the manifest file within
the bucket.\n\n";

    if (args.length != 4) {
        System.out.println(USAGE);
        System.exit(1);
    }

    projectArn = args[0];
    datasetType = args[1];
    bucket = args[2];
    name = args[3];

    try {
```

```
        // Get the Rekognition client
        RekognitionClient rekClient = RekognitionClient.builder()
            .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
            .region(Region.US_WEST_2)
            .build();

        // Create the dataset
        datasetArn = createMyDataset(rekClient, projectArn, datasetType,
bucket, name);

        System.out.println(String.format("Created dataset: %s",
datasetArn));

        rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    } catch (Exception rekError) {
        logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());
        System.exit(1);
    }
}
}
```

4. ラベルの追加または変更の必要がある場合は、「[ラベルの管理 \(SDK\)](#)」を参照してください。
5. 「[モデルのトレーニング \(SDK\)](#)」の手順に従って、モデルをトレーニングします。

## Amazon SageMaker Ground Truth ジョブ

Amazon SageMaker Ground Truth では、選択したベンダー企業である Amazon Mechanical Turk のワーカー、またはラベル付き画像セットを作成できる機械学習を備えた社内のプライベートワークフォースを使用できます。Amazon Rekognition Custom Labels は、指定した Amazon S3 バケットから SageMaker Ground Truth マニフェストファイルをインポートします。Amazon S3

Amazon Rekognition Custom Labels は、以下の SageMaker Ground Truth タスクをサポートしています。

- [イメージ分類](#)
- [境界ボックス](#)

インポートするファイルは、イメージとマニフェストファイルです。マニフェストファイルには、インポートするイメージのラベルと境界ボックス情報が含まれています。

Amazon Rekognition には、イメージが保存されている 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。Amazon Rekognition Custom Labels によって設定されたコンソールバケットを使用している場合、必要なアクセス許可は既に設定されています。コンソールバケットを使用していない場合は、「[外部の Amazon S3 バケットへのアクセス](#)」を参照してください。

SageMaker Ground Truth ジョブを使用したマニフェストファイルの作成 (コンソール)

次の手順は、SageMaker Ground Truth ジョブでラベル付けされた画像を使用してデータセットを作成する方法を示しています。ジョブ出力ファイルは Amazon Rekognition Custom Labels コンソールバケットに保存されます。

SageMaker Ground Truth ジョブでラベル付けされた画像を使用してデータセットを作成するには (コンソール)

1. にサインイン AWS Management Console し、<https://console.aws.amazon.com/s3/> で Amazon S3 コンソールを開きます。
2. コンソールバケットに、トレーニングイメージを保存する [フォルダを作成](#)します。

**Note**

コンソールバケットは、AWS リージョンで Amazon Rekognition Custom Labels コンソールを初めて開いたときに作成されます。詳細については、「[Amazon Rekognition Custom Labels プロジェクトの管理](#)」を参照してください。

3. 作成したフォルダに [イメージをアップロード](#)します。
4. コンソールバケットに、Ground Truth ジョブの出力を保存するフォルダを作成します。
5. <https://console.aws.amazon.com/sagemaker/> で SageMaker コンソールを開きます。

6. Ground Truth ラベリングジョブを作成します。ステップ 2 とステップ 4 で作成したフォルダの Amazon S3 URL が必要になります。詳細については、「[データラベリングに Amazon SageMaker Ground Truth を使用する](#)」を参照してください。
7. ステップ 4 で作成したフォルダ内の output.manifest ファイルの場所を記録しておきます。このファイルは、サブフォルダ *Ground-Truth-Job-Name*/manifests/output に保存されている必要があります。
8. 「[SageMaker Ground Truth マニフェストファイルを使用したデータセットの作成 \(コンソール\)](#)」の手順に従って、アップロードしたマニフェストファイルを使用してデータセットを作成します。ステップ 8 では、[マニフェストファイルの場所] に、前のステップで記録しておいた場所の Amazon S3 URL を入力します。AWS SDK を使用している場合は、[SageMaker Ground Truth マニフェストファイル \(SDK\) を使用したデータセットの作成](#) を実行します。
9. ステップ 1~6 を繰り返して、テストデータセットの SageMaker Ground Truth ジョブを作成します。

## マニフェストファイルの作成

SageMaker Ground Truth 形式のマニフェストファイルをインポートすることで、テストデータセットまたはトレーニングデータセットを作成できます。イメージが SageMaker Ground Truth マニフェストファイルではない形式でラベル付けされている場合は、次の情報を使用して SageMaker Ground Truth 形式のマニフェストファイルを作成します。

マニフェストファイルは [JSON Lines](#) 形式で、各行は画像のラベリング情報を表す完全な JSON オブジェクトです。Amazon Rekognition Custom Labels は、次の形式で JSON 行を含む SageMaker Ground Truth マニフェストをサポートしています。

- [\[分類ジョブ出力\]](#) - イメージにイメージレベルのラベルを追加する場合に使用します。イメージレベルのラベルは、イメージ上にあるシーン、概念、オブジェクト (オブジェクトの位置情報が不要な場合) のクラスを定義します。1 つのイメージには、複数のイメージレベルのラベルを付けることができます。詳細については、「[マニフェストファイル内のイメージレベルのラベル](#)」を参照してください。
- [\[境界ボックスジョブ出力\]](#) - イメージ上の 1 つ以上のオブジェクトのクラスと位置にラベルを付けるために使用します。詳細については、「[マニフェストファイル内のオブジェクトのローカリゼーション](#)」を参照してください。

イメージレベルとローカリゼーション (境界ボックス) の JSON 行を同じマニフェストファイルに連結できます。

**Note**

このセクションの JSON Lines の例は、読みやすいようにフォーマットされています。

マニフェストファイルをインポートすると、Amazon Rekognition Custom Labels は制限、構文、セマンティクスの検証ルールを適用します。詳細については、「[マニフェストファイルの検証ルール](#)」を参照してください。

マニフェストファイルで参照される画像は、同じ Amazon S3 バケットに配置する必要があります。マニフェストファイルは、イメージを保存する Amazon S3 バケットとは異なる Amazon S3 バケットに配置できます。JSON Lines の source-ref フィールドに画像の場所を指定します。

Amazon Rekognition には、イメージが保存されている 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。Amazon Rekognition Custom Labels によって設定されたコンソールバケットを使用している場合、必要なアクセス許可は既に設定されています。コンソールバケットを使用していない場合は、「[外部の Amazon S3 バケットへのアクセス](#)」を参照してください。

## トピック

- [マニフェストファイルの作成](#)
- [マニフェストファイル内のイメージレベルのラベル](#)
- [マニフェストファイル内のオブジェクトのローカリゼーション](#)
- [マニフェストファイルの検証ルール](#)

## マニフェストファイルの作成

次の手順で、トレーニングデータセットとテストデータセットを持つプロジェクトを作成します。データセットは、作成した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から作成されます。

SageMaker Ground Truth 形式のマニフェストファイルを使用してデータセットを作成するには (コンソール)

1. コンソールバケットに、マニフェストファイルを保存する [フォルダを作成](#) します。
2. コンソールバケットに、イメージを保存するフォルダを作成します。
3. 作成したフォルダに画像をアップロードします。

4. トレーニングデータセットの SageMaker Ground Truth 形式のマニフェストファイルを作成します。詳細については、「[マニフェストファイル内のイメージレベルのラベル](#)」および「[マニフェストファイル内のオブジェクトのローカリゼーション](#)」を参照してください。

#### Important

各 JSON 行の source-ref フィールド値は、アップロードしたイメージにマッピングする必要があります。

5. テストデータセットの SageMaker Ground Truth 形式のマニフェストファイルを作成します。
6. 先ほど作成したフォルダに、[マニフェストファイルをアップロード](#)します。
7. マニフェストファイルの場所を記録しておきます。
8. 「[SageMaker Ground Truth マニフェストファイルを使用したデータセットの作成 \(コンソール\)](#)」の手順に従って、アップロードしたマニフェストファイルを使用してデータセットを作成します。ステップ 8 では、[マニフェストファイルの場所] に、前のステップで記録しておいた場所の Amazon S3 URL を入力します。AWS SDK を使用している場合は、[を実行します SageMaker Ground Truth マニフェストファイル \(SDK\) を使用したデータセットの作成](#)。

## マニフェストファイル内のイメージレベルのラベル

イメージレベルのラベル (ローカリゼーション情報を必要としないシーン、概念、またはオブジェクトでラベル付けされたイメージ) をインポートするには、SageMaker Ground Truth [分類ジョブ出力](#)形式の JSON 行をマニフェストファイルに追加します。マニフェストファイルは、インポートする画像ごとに1行以上の JSON Lines で構成されます。

#### Tip

マニフェストファイルの作成を簡易化するために、CSV ファイルからマニフェストファイルを作成する Python スクリプトを提供しています。詳細については、「[CSV ファイルからのマニフェストファイルの作成](#)」を参照してください。

## 画像レベルラベルのマニフェストファイルを作成するには

1. 空のテキストファイルを作成します。
2. インポートする画像ごとに JSON Lines を追加します。各 JSON 行は次のようになります。

```
{"source-ref":"s3://custom-labels-console-us-east-1-nnnnnnnnnn/gt-job/manifest/IMG_1133.png","TestCLConsoleBucket":0,"TestCLConsoleBucket-metadata":{"confidence":0.95,"job-name":"labeling-job/testclconsolebucket","class-name":"Echo Dot","human-annotated":"yes","creation-date":"2020-04-15T20:17:23.433061","type":"groundtruth/image-classification"}}
```

3. ファイルを保存します。 .manifest 拡張機能を使用できますが、必須ではありません。
4. 作成したマニフェストファイルを使用してデータセットを作成します。詳細については、「[SageMaker Ground Truth 形式のマニフェストファイルを使用してデータセットを作成するには \(コンソール\)](#)」を参照してください。

## イメージレベルの JSON 行

このセクションでは、1つのイメージの JSON 行を作成する方法について説明します。次のイメージについて考えます。次のイメージは、日の出のようなシーンです。





上のイメージの日の出というシーンの JSON 行は次のようになります。

```
{
  "source-ref": "s3://bucket/images/sunrise.png",
  "testdataset-classification_Sunrise": 1,
  "testdataset-classification_Sunrise-metadata": {
    "confidence": 1,
    "job-name": "labeling-job/testdataset-classification_Sunrise",
    "class-name": "Sunrise",
    "human-annotated": "yes",
    "creation-date": "2020-03-06T17:46:39.176",
    "type": "groundtruth/image-classification"
  }
}
```

以下の情報に注意してください。

source-ref

(必須) 画像の Amazon S3 の場所。形式は "`s3://BUCKET/OBJECT_PATH`" です。インポートされたデータセット内の画像は、同じ Amazon S3 バケットに格納する必要があります。

### ***testdataset-classification\_Sunrise***

(必須) ラベル属性。フィールド名を選択します。フィールド値 (前の例では 1) はラベル属性識別子です。Amazon Rekognition Custom Labels では使用されないため、整数値などのような値でも構いません。末尾に `-metadata` が付いたフィールド名により特定される該当メタデータが必要です。例えば、"`testdataset-classification_Sunrise-metadata`"。

### ***testdataset-classification\_Sunrise-metadata***

(必須) ラベル属性に関するメタデータ。フィールド名は、`-metadata` を付加したラベル属性と同じでなければなりません。

confidence

(必須) 現在 Amazon Rekognition Custom Labels では使用されていませんが、0 から 1 の間の値を指定する必要があります。

job-name

(オプション) 画像を処理するジョブに対して選択した名前。

class-name

(必須) イメージに適用されるシーンまたは概念に対して選択したクラス名。例えば "`Sunrise`" です。

human-annotated

(必須) アノテーションが人によって完成されている場合、"`yes`" を指定します。それ以外の場合は、"`no`" です。

creation-date

(必須) ラベルが作成された協定世界時 (UTC) の日時。

type

(必須) 画像に適用する処理のタイプ。イメージレベルのラベルの場合、値は "`groundtruth/image-classification`" です。

## 複数のイメージレベルのラベルをイメージに追加する

イメージには複数のラベルを追加できます。例えば、次の JSON では 1 つのイメージに football と ball の 2 つのラベルを追加します。

```
{
  "source-ref": "S3 bucket location",
  "sport0":0, # FIRST label
  "sport0-metadata": {
    "class-name": "football",
    "confidence": 0.8,
    "type":"groundtruth/image-classification",
    "job-name": "identify-sport",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:18:13.527256"
  },
  "sport1":1, # SECOND label
  "sport1-metadata": {
    "class-name": "ball",
    "confidence": 0.8,
    "type":"groundtruth/image-classification",
    "job-name": "identify-sport",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:18:13.527256"
  }
} # end of annotations for 1 image
```

## マニフェストファイル内のオブジェクトのローカリゼーション

Ground Truth [境界ボックスジョブ出力](#)形式の JSON 行をマニフェストファイルに追加 SageMaker することで、オブジェクトローカリゼーション情報でラベル付けされたイメージをインポートできます。

ローカリゼーション情報は、イメージ上のオブジェクトの位置を表します。位置は、オブジェクトを囲む境界ボックスで表されます。境界ボックスの構造には、境界ボックスの左上の座標と、境界ボックスの幅と高さが含まれます。境界ボックス形式の JSON 行には、イメージ上の 1 つ以上のオブジェクトの位置と、イメージ上の各オブジェクトのクラスの境界ボックスが含まれます。

マニフェストファイルは 1 行以上の JSON 行で構成され、各行には 1 つのイメージの情報が含まれます。

## オブジェクトローカリゼーションのマニフェストファイルを作成するには

1. 空のテキストファイルを作成します。
2. インポートする画像ごとに JSON Lines を追加します。各 JSON 行は次のようになります。

```
{"source-ref": "s3://bucket/images/IMG_1186.png", "bounding-box": {"image_size": [{"width": 640, "height": 480, "depth": 3}], "annotations": [{"class_id": 1, "top": 251, "left": 399, "width": 155, "height": 101}, {"class_id": 0, "top": 65, "left": 86, "width": 220, "height": 334}]}, "bounding-box-metadata": {"objects": [{"confidence": 1}, {"confidence": 1}], "class-map": {"0": "Echo", "1": "Echo Dot"}, "type": "groundtruth/object-detection", "human-annotated": "yes", "creation-date": "2013-11-18T02:53:27", "job-name": "my job"}}
```

3. ファイルを保存します。 .manifest 拡張機能を使用できますが、必須ではありません。
4. 作成したファイルを使用してデータセットを作成します。詳細については、「 [SageMaker Ground Truth 形式のマニフェストファイルを使用してデータセットを作成するには \(コンソール\)](#) 」を参照してください。

### オブジェクト境界ボックスの JSON 行

このセクションでは、1つのイメージの JSON 行を作成する方法について説明します。次のイメージは、Amazon Echo デバイスと Amazon Echo Dot デバイスの周囲の境界ボックスを示しています。



以下は、前のイメージの境界ボックスの JSON 行です。

```
{
  "source-ref": "s3://custom-labels-bucket/images/IMG_1186.png",
  "bounding-box": {
    "image_size": [{
      "width": 640,
      "height": 480,
      "depth": 3
    }],
    "annotations": [{
      "class_id": 1,
      "top": 251,
      "left": 399,
      "width": 155,
```

```
    "height": 101
  }, {
    "class_id": 0,
    "top": 65,
    "left": 86,
    "width": 220,
    "height": 334
  ]
},
"bounding-box-metadata": {
  "objects": [{
    "confidence": 1
  }, {
    "confidence": 1
  }],
  "class-map": {
    "0": "Echo",
    "1": "Echo Dot"
  },
  "type": "groundtruth/object-detection",
  "human-annotated": "yes",
  "creation-date": "2013-11-18T02:53:27",
  "job-name": "my job"
}
}
```

以下の情報に注意してください。

#### source-ref

(必須) 画像の Amazon S3 の場所。形式は `s3://BUCKET/OBJECT_PATH` です。インポートされたデータセット内の画像は、同じ Amazon S3 バケットに格納する必要があります。

#### **bounding-box**

(必須) ラベル属性。フィールド名を選択します。イメージサイズと、イメージ内で検知された各オブジェクトの境界ボックスが含まれます。末尾に `-metadata` が付いたフィールド名により特定される該当メタデータが必要です。例えば、`"bounding-box-metadata"`。

#### image\_size

(必須) イメージのサイズ (ピクセル単位) を含む単一要素の配列。

- `height` - (必須) イメージの高さ (ピクセル単位)。



- width - (必須) イメージの奥行き (ピクセル単位)。
- depth - (必須) イメージ内のチャンネル数。RGB イメージの最大値は 3 です。Amazon Rekognition Custom Labels では現在使用されていませんが、値が必要です。

#### annotations

(必須) イメージ内で検知された各オブジェクトの境界ボックス情報の配列。

- class\_id - (必須) class-map のラベルにマッピングします。前の例では、class\_id が 1 のオブジェクトは、イメージ内の Echo Dot です。
- top - (必須) イメージの上部から境界ボックスの上部までの距離 (ピクセル単位)。
- left - (必須) イメージの左から境界ボックスの左までの距離 (ピクセル単位)。
- width - (必須) 境界ボックスの幅 (ピクセル単位)。
- height - (必須) 境界ボックスの高さ (ピクセル単位)。

#### **bounding-box**-metadata

(必須) ラベル属性に関するメタデータ。フィールド名は、-metadata を付加したラベル属性と同じでなければなりません。イメージ内で検知された各オブジェクトの境界ボックス情報の配列。

#### オブジェクト

(必須) イメージ内のオブジェクトの配列。インデックスによって annotations 配列にマッピングされます。Amazon Rekognition Custom Labels では、信頼性属性は使用されません。

#### class-map

(必須) イメージ内で検知されたオブジェクトに適用されるクラスのマップ。

#### type

(必須) 分類ジョブのタイプ。"groundtruth/object-detection" はジョブをオブジェクト検知として識別します。

#### creation-date

(必須) ラベルが作成された協定世界時 (UTC) の日時。

#### human-annotated

(必須) アノテーションが人によって完成されている場合、"yes" を指定します。それ以外の場合は、"no" です。



## job-name

(オプション) イメージを処理するジョブの名前。

## マニフェストファイルの検証ルール

マニフェストファイルをインポートすると、Amazon Rekognition Custom Labels は制限、構文、セマンティクスの検証ルールを適用します。 SageMaker Ground Truth スキーマは構文検証を適用しません。詳細については、「[出力](#)」を参照してください。制限とセマンティクスの検証規則は次のとおりです。

### Note

- 20% の無効ルールは、すべての検証ルールに累積で適用されます。JSON が 15% 無効、イメージが 15% など、何らかの組み合わせでインポートが 20% の制限を超えた場合、インポートは失敗します。
- 各データセットオブジェクトはマニフェスト内の 1 行です。空白行や無効行もデータセットオブジェクトとしてカウントされます。
- オーバーラップは (テストとトレーニングの共通ラベル)/(トレーニングラベル) です。

## トピック

- [制限](#)
- [セマンティクス](#)

## 制限

検証	制限	エラー発生
マニフェストファイルのサイズ	最大 1 GB	エラー
マニフェストファイルの最大行数	マニフェストの 1 行あたりのデータセットオブジェクトの最大数は 250,000 です。	エラー

検証	制限	エラー発生
ラベルごとの有効なデータセットオブジェクトの合計の下限	$\geq 1$	エラー
ラベルの下限	$\geq 2$	エラー
ラベルの上限	$\leq 250$	エラー
イメージごとの最小境界ボックス	0	なし
イメージごとの最大境界ボックス	50	なし

## セマンティクス

検証	制限	エラー発生
マニフェストが空		エラー
source-ref オブジェクトが見つからない、またはアクセスできない	20% 未満のオブジェクト数	警告
source-ref オブジェクトが見つからない、またはアクセスできない	プロジェクト数 > 20%	エラー
トレーニングデータセットにテストラベルがない	ラベルのオーバーラップが 50% 以上	エラー
データセット内の同じラベルのラベルサンプルとオブジェクトサンプルが混合している。データセットオブジェク		エラー/警告なし

検証	制限	エラー発生
ト内の同じクラスの分類と検知。		
テスト用とトレーニング用のアセットの重複	テストデータセットとトレーニングデータセットの間の重複がないこと。	
データセット内のイメージが同じバケットにある	オブジェクトが別のバケットにある場合はエラーになる。	エラー

### 他のデータセット形式をマニフェストファイルに変換する

以下の情報を使用して、さまざまなソースデータセット形式から Amazon SageMaker 形式のマニフェストファイルを作成できます。マニフェストファイルを作成したら、それを使用してデータセットを作成します。詳細については、「[マニフェストファイル](#)」を参照してください。

#### トピック

- [COCO データセットの変換](#)
- [マルチラベルの SageMaker Ground Truth マニフェストファイルの変換](#)
- [CSV ファイルからのマニフェストファイルの作成](#)

### COCO データセットの変換

[COCO](#) は、大規模なオブジェクト検知、セグメンテーション、キャプションデータセットを指定するための形式です。この Python の[例](#)では、COCO オブジェクト検知形式のデータセットを Amazon Rekognition Custom Labels [境界ボックス形式のマニフェストファイル](#)に変換する方法を示しています。このセクションには、独自のコードを記述するために使用できる情報も含まれています。

COCO 形式の JSON ファイルは、データセット全体の情報を提供する 5 つのセクションで構成されています。詳細については、「[COCO 形式](#)」を参照してください。

- `info` - データセットに関する一般情報。
- `licenses` - データセット内のイメージのライセンス情報。
- `images` - データセット内のイメージのリスト。

- [annotations](#) - データセット内のすべてのイメージに含まれる注釈 (境界ボックスを含む) のリスト。
- [categories](#) - ラベルカテゴリのリスト。

Amazon Rekognition Custom Labels マニフェストファイルを作成するには images、annotations、および categories のリストの情報が必要です。

Amazon Rekognition Custom Labels マニフェストファイルは JSON 行形式で、各行にはイメージの 1 つ以上のオブジェクトの境界ボックスとラベル情報が含まれます。詳細については、「[マニフェストファイル内のオブジェクトのローカリゼーション](#)」を参照してください。

COCO オブジェクトをカスタムラベルの JSON 行にマッピングする

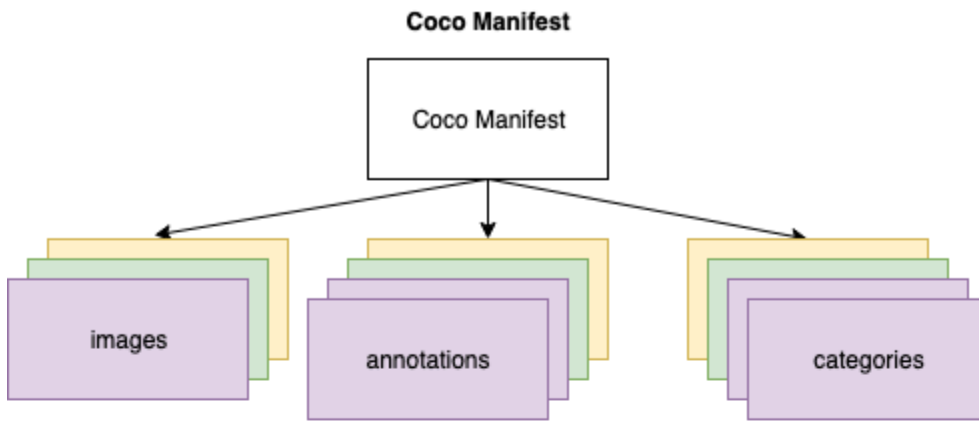
COCO 形式のデータセットを変換するには、COCO データセットを Amazon Rekognition Custom Labels マニフェストファイルにマッピングして、オブジェクトのローカリゼーションを行います。詳細については、「[マニフェストファイル内のオブジェクトのローカリゼーション](#)」を参照してください。イメージごとに JSON 行を作成するには、マニフェストファイルに COCO データセット image、annotation、および category オブジェクトフィールド ID をマッピングする必要があります。

次は、COCO マニフェストファイルの例です。詳細については、「[COCO 形式](#)」を参照してください。

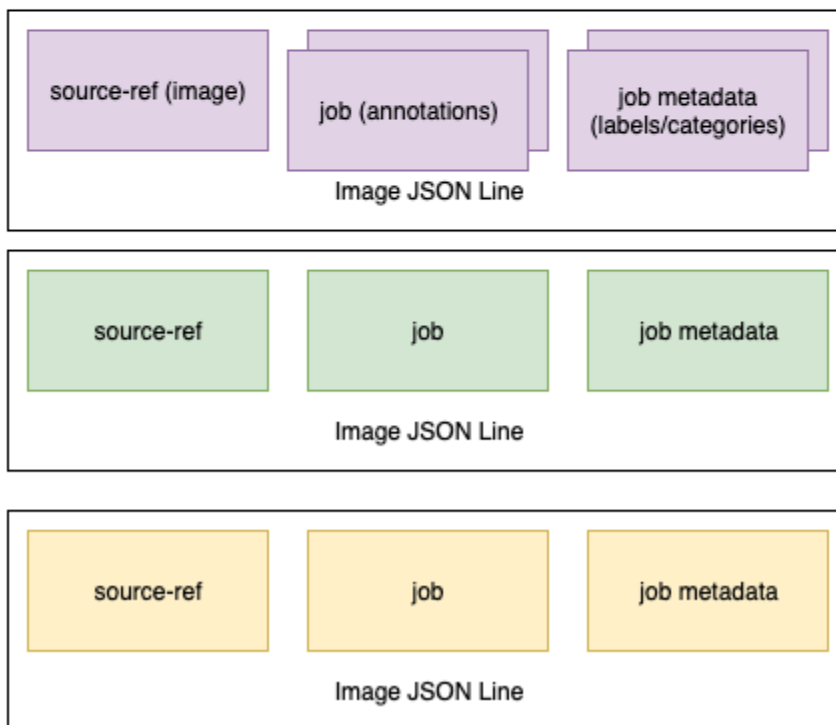
```
{
  "info": {
    "description": "COCO 2017 Dataset","url": "http://cocodataset.org","version":
"1.0","year": 2017,"contributor": "COCO Consortium","date_created": "2017/09/01"
  },
  "licenses": [
    {"url": "http://creativecommons.org/licenses/by/2.0/","id": 4,"name":
"Attribution License"}
  ],
  "images": [
    {"id": 242287, "license": 4, "coco_url": "http://images.cocodataset.org/
val2017/xxxxxxxxxxxx.jpg", "flickr_url": "http://farm3.staticflickr.com/2626/
xxxxxxxxxxxx.jpg", "width": 426, "height": 640, "file_name": "xxxxxxxx.jpg",
"date_captured": "2013-11-15 02:41:42"},
    {"id": 245915, "license": 4, "coco_url": "http://images.cocodataset.org/
val2017/nnnnnnnnnnn.jpg", "flickr_url": "http://farm1.staticflickr.com/88/
xxxxxxxxxxxx.jpg", "width": 640, "height": 480, "file_name": "nnnnnnnnnn.jpg",
"date_captured": "2013-11-18 02:53:27"}
```

```
    ],
    "annotations": [
      {"id": 125686, "category_id": 0, "iscrowd": 0, "segmentation": [[164.81,
417.51,.....167.55, 410.64]], "image_id": 242287, "area": 42061.803400000001, "bbox":
[19.23, 383.18, 314.5, 244.46]},
      {"id": 1409619, "category_id": 0, "iscrowd": 0, "segmentation": [[376.81,
238.8,.....382.74, 241.17]], "image_id": 245915, "area": 3556.2197000000015,
"bbox": [399, 251, 155, 101]},
      {"id": 1410165, "category_id": 1, "iscrowd": 0, "segmentation": [[486.34,
239.01,.....495.95, 244.39]], "image_id": 245915, "area": 1775.8932499999994,
"bbox": [86, 65, 220, 334]}
    ],
    "categories": [
      {"supercategory": "speaker","id": 0,"name": "echo"},
      {"supercategory": "speaker","id": 1,"name": "echo dot"}
    ]
  }
}
```

次の図では、データセットの COCO データセットのリストが、イメージの Amazon Rekognition Custom Labels JSON 行にどのようにマッピングされるかを示しています。一致する色は、1 つのイメージの情報を示しています。



**Custom Labels JSON Lines**



単一の JSON 行の COCO オブジェクトを取得するには

1. イメージリスト内の各イメージについて、注釈リストから、注釈フィールド `image_id` の値がイメージ `id` フィールドと一致する注釈を取得します。
2. ステップ 1 で一致した注釈ごとに、`categories` リストを読み込み、`category` フィールド `id` の値が `annotation` オブジェクトの `category_id` フィールドと一致する各 `category` を取得します。

3. 一致した image、annotation、および category オブジェクトを使用して、イメージの JSON 行を作成します。フィールドをマッピングするには、「[COCO オブジェクトフィールドをカスタムラベル JSON 行オブジェクトフィールドにマッピングする](#)」を参照してください。
4. images リスト内の各 image オブジェクトに JSON 行が作成されるまで、ステップ 1~3 を繰り返します。

サンプルコードについては、「[COCO データセットの変換](#)」を参照してください。

## COCO オブジェクトフィールドをカスタムラベル JSON 行オブジェクトフィールドにマッピングする

Amazon Rekognition Custom Labels JSON 行の COCO オブジェクトを特定したら、COCO オブジェクトフィールドをそれぞれの Amazon Rekognition Custom Labels JSON 行オブジェクトフィールドにマッピングする必要があります。次の Amazon Rekognition Custom Labels JSON 行の例では、1つのイメージ (id=000000245915) を前述の COCO JSON の例にマッピングしています。以下の情報に注意してください。

- source-ref は Amazon S3 バケット内のイメージの場所です。COCO のイメージが Amazon S3 バケットに保存されていない場合は、それらを Amazon S3 バケットに移動する必要があります。
- annotations リストには、イメージのオブジェクトごとに 1つの annotation オブジェクトが含まれています。annotation オブジェクトには、境界ボックス情報 (top、left、width、height) とラベル識別子 (class\_id) が含まれます。
- ラベル識別子 (class\_id) はメタデータ内の class-map リストにマッピングされます。イメージに使用されているラベルが一覧表示されます。

```
{
  "source-ref": "s3://custom-labels-bucket/images/000000245915.jpg",
  "bounding-box": {
    "image_size": {
      "width": 640,
      "height": 480,
      "depth": 3
    },
    "annotations": [{
      "class_id": 0,
      "top": 251,
      "left": 399,
      "width": 155,
```



```
    "height": 101
  }, {
    "class_id": 1,
    "top": 65,
    "left": 86,
    "width": 220,
    "height": 334
  ]
},
"bounding-box-metadata": {
  "objects": [{
    "confidence": 1
  }, {
    "confidence": 1
  }],
  "class-map": {
    "0": "Echo",
    "1": "Echo Dot"
  },
  "type": "groundtruth/object-detection",
  "human-annotated": "yes",
  "creation-date": "2018-10-18T22:18:13.527256",
  "job-name": "my job"
}
}
```

以下の情報を使用して、Amazon Rekognition Custom Labels マニフェストファイルフィールドを COCO データセット JSON フィールドにマッピングします。

#### source-ref

イメージの場所の S3 形式の URL。イメージは S3 バケットに保存されている必要があります。詳細については、「[source-ref](#)」を参照してください。coco\_url COCO フィールドが S3 バケットの場所を指している場合、coco\_url の値には source-ref の値を使用できます。または、source-ref を file\_name (COCO) フィールドにマッピングし、イメージが保存されている場所に必要な S3 パスを変換コードに追加することもできます。

#### **bounding-box**

選択したラベルの属性名。詳細については、「[bounding-box](#)」を参照してください。

## image\_size

イメージサイズ (ピクセル単位) [イメージ](#) リストの image オブジェクトにマッピングします。

- height-> [image](#).height
- width-> [image](#).width
- depth-> Amazon Rekognition Custom Labels では使用されませんが、値を指定する必要があります。

## annotations

annotation オブジェクトのリスト。イメージのオブジェクトごとに 1 つの annotation があります。

### 注釈

イメージのオブジェクトの 1 つのインスタンスの境界ボックス情報が含まれます。

- class\_id-> カスタムラベルの class-map リストへの数値 ID マッピング。
- top -> [bbox](#)[1]
- left -> [bbox](#)[0]
- width -> [bbox](#)[2]
- height -> [bbox](#)[3]

## **bounding-box**-metadata

ラベル属性のメタデータ。ラベルとラベル ID が含まれています。詳細については、「[bounding-box-metadata](#)」を参照してください。

### オブジェクト

イメージ内のオブジェクトの配列。インデックスによって annotations リストにマッピングされます。

### オブジェクト

- confidence->Amazon Rekognition Custom Labels では使用されませんが、値 (1) が必要です。

## class-map

イメージ内で検知されたオブジェクトに適用されるラベル (クラス) のマップ。[カテゴリ](#)リスト内のカテゴリオブジェクトにマッピングします。

- id -> [category](#).id
- id value -> [category](#).name

## type

groundtruth/object-detection を指定してください

## human-annotated

yes または no を指定します。詳細については、「[bounding-box-metadata](#)」を参照してください。

creation-date -> [image](#).date\_captured

イメージの作成日時。COCO イメージリストの [image](#).date\_captured フィールドにマッピングされます。Amazon Rekognition Custom Labels では、creation-date の形式が Y-M-DTH:M:S であることを想定しています。

## job-name

ユーザーが選択したジョブ名。

## COCO 形式

COCO データセットは、データセット全体の情報を提供する 5 つの情報セクションで構成されています。COCO オブジェクト検知データセットの形式は「[COCO Data Format](#)」に記載されています。

- info - データセットに関する一般情報。
- licenses - データセット内のイメージのライセンス情報。
- [images](#) - データセット内のイメージのリスト。
- [annotations](#) - データセット内のすべてのイメージに含まれる注釈 (境界ボックスを含む) のリスト。
- [categories](#) - ラベルカテゴリのリスト。

カスタムラベルマニフェストを作成するには、COCO マニフェストファイルの images、annotations、および categories リストを使用します。他のセクション (info、licences) は必須ではありません。次は、COCO マニフェストファイルの例です。

```
{
  "info": {
    "description": "COCO 2017 Dataset","url": "http://cocodataset.org","version":
    "1.0","year": 2017,"contributor": "COCO Consortium","date_created": "2017/09/01"
  },
  "licenses": [
    {"url": "http://creativecommons.org/licenses/by/2.0/","id": 4,"name":
    "Attribution License"}
  ],
  "images": [
    {"id": 242287, "license": 4, "coco_url": "http://images.cocodataset.org/
    val2017/xxxxxxxxxxxxx.jpg", "flickr_url": "http://farm3.staticflickr.com/2626/
    xxxxxxxxxxxxxx.jpg", "width": 426, "height": 640, "file_name": "xxxxxxxxx.jpg",
    "date_captured": "2013-11-15 02:41:42"},
    {"id": 245915, "license": 4, "coco_url": "http://images.cocodataset.org/
    val2017/nnnnnnnnnnnnn.jpg", "flickr_url": "http://farm1.staticflickr.com/88/
    xxxxxxxxxxxxxx.jpg", "width": 640, "height": 480, "file_name": "nnnnnnnnnnn.jpg",
    "date_captured": "2013-11-18 02:53:27"}
  ],
  "annotations": [
    {"id": 125686, "category_id": 0, "iscrowd": 0, "segmentation": [[164.81,
    417.51,.....167.55, 410.64]], "image_id": 242287, "area": 42061.80340000001, "bbox":
    [19.23, 383.18, 314.5, 244.46]},
    {"id": 1409619, "category_id": 0, "iscrowd": 0, "segmentation": [[376.81,
    238.8,.....382.74, 241.17]], "image_id": 245915, "area": 3556.2197000000015,
    "bbox": [399, 251, 155, 101]},
    {"id": 1410165, "category_id": 1, "iscrowd": 0, "segmentation": [[486.34,
    239.01,.....495.95, 244.39]], "image_id": 245915, "area": 1775.8932499999994,
    "bbox": [86, 65, 220, 334]}
  ],
  "categories": [
    {"supercategory": "speaker","id": 0,"name": "echo"},
    {"supercategory": "speaker","id": 1,"name": "echo dot"}
  ]
}
```

## イメージリスト

COCO データセットが参照するイメージは、イメージ配列に一覧表示されます。各イメージオブジェクトには、イメージファイル名などのイメージに関する情報が含まれています。次のイメージオブジェクトの例では、次の情報と、Amazon Rekognition Custom Labels マニフェストファイルを作成するために必要なフィールドを記録しておきます。

- `id` - (必須) イメージの一意の識別子。`id` フィールドは、`annotations` 配列 (境界ボックス情報が格納されている) の `id` フィールドにマッピングされます。
- `license` - (不要) ライセンス配列にマッピングします。
- `coco_url` - (オプション) スキーマの場所。
- `flickr_url` - (不要) Flickr のイメージの場所。
- `width` - (必須) イメージの幅。
- `height` - (必須) イメージの高さ。
- `file_name` - (必須) イメージのファイル名。この例では `file_name` と `id` が一致していますが、これは COCO データセットの要件ではありません。
- `date_captured` - (必須) イメージがキャプチャされた日時。

```
{
  "id": 245915,
  "license": 4,
  "coco_url": "http://images.cocodataset.org/val2017/nnnnnnnnnnnnn.jpg",
  "flickr_url": "http://farm1.staticflickr.com/88/nnnnnnnnnnnnnnnnnnnn.jpg",
  "width": 640,
  "height": 480,
  "file_name": "000000245915.jpg",
  "date_captured": "2013-11-18 02:53:27"
}
```

## 注釈 (境界ボックス) リスト

全イメージのすべてのオブジェクトの境界ボックス情報は、注釈リストに保存されます。1つの注釈オブジェクトには、1つのオブジェクトの境界ボックス情報と、イメージのオブジェクトのラベルが含まれています。イメージのオブジェクトのインスタンスごとに1つの注釈オブジェクトがあります。

次の例では、次の情報と、Amazon Rekognition Custom Labels マニフェストファイルを作成するために必要なフィールドを記録しておきます。

- `id` - (不要) 注釈の識別子。
- `image_id` - (必須) イメージ配列内のイメージ `id` に対応します。
- `category_id` - (必須) 境界ボックス内のオブジェクトを識別するラベルの識別子。カテゴリ配列の `id` フィールドにマッピングされます。
- `iscrowd` - (不要) イメージに多数のオブジェクトが含まれているかどうかを指定します。
- `segmentation` - (不要) イメージのオブジェクトのセグメンテーション情報。Amazon Rekognition Custom Labels はセグメンテーションをサポートしていません。
- `area` - (不要) 注釈の領域。
- `bbox` - (必須) イメージのオブジェクトを囲む境界ボックスの座標 (ピクセル単位) が含まれます。

```
{
  "id": 1409619,
  "category_id": 1,
  "iscrowd": 0,
  "segmentation": [
    [86.0, 238.8, .....382.74, 241.17]
  ],
  "image_id": 245915,
  "area": 3556.21970000000015,
  "bbox": [86, 65, 220, 334]
}
```

## カテゴリリスト

ラベル情報は、カテゴリ配列に保存されます。次のカテゴリオブジェクトの例では、次の情報と、Amazon Rekognition Custom Labels マニフェストファイルを作成するために必要なフィールドを記録しておきます。

- `supercategory` - (不要) ラベルの親カテゴリ。
- `id` - (必須) ラベル識別子。`id` フィールドは `annotation` オブジェクト内の `category_id` フィールドにマッピングされます。次の例では、Echo Dot の識別子は 2 です。
- `name` - (必須) ラベル名。

```
{"supercategory": "speaker", "id": 2, "name": "echo dot"}
```

## COCO データセットの変換

次の Python の例を使用して、境界ボックス情報を、COCO 形式のデータセットから Amazon Rekognition Custom Labels マニフェストファイルに変換します。このコードは、作成したマニフェストファイルを Amazon S3 バケットにアップロードします。このコードには、イメージのアップロードに使用できる AWS CLI コマンドも用意されています。

COCO データセットを変換するには (SDK)

1. まだ実行していない場合:
  - a. AmazonS3FullAccess の権限があることを確認します。詳細については、「[SDK アクセス許可の設定](#)」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 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. 次の Python コードを使用して COCO データセットを変換します。次の値を設定します。
  - s3\_bucket - イメージと Amazon Rekognition Custom Labels マニフェストファイルを保存する S3 バケットの名前。
  - s3\_key\_path\_images - S3 バケット (s3\_bucket) 内のイメージを保存する場所へのパス。
  - s3\_key\_path\_manifest\_file - S3 バケット (s3\_bucket) 内の Custom Labels マニフェストファイルを保存する場所へのパス。
  - local\_path - サンプルが入力 COCO データセットを開き、新しいカスタムラベルマニフェストファイルを保存するローカルパス。
  - local\_images\_path - トレーニングに使用するイメージへのローカルパス。
  - coco\_manifest - 入力 COCO データセットのファイル名。
  - cl\_manifest\_file - このサンプルで作成されたマニフェストファイルの名前。ファイルは local\_path で指定された場所に保存されます。慣例により、ファイルには拡張子 .manifest が付いていますが、必須ではありません。
  - job\_name - カスタムラベルジョブの名前。

```
import json
import os
import random
import shutil
import datetime
import boto3
```



```
import boto3
import PIL.Image as Image
import io

#S3 location for images
s3_bucket = 'bucket'
s3_key_path_manifest_file = 'path to custom labels manifest file/'
s3_key_path_images = 'path to images/'
s3_path='s3://' + s3_bucket + '/' + s3_key_path_images
s3 = boto3.resource('s3')

#Local file information
local_path='path to input COCO dataset and output Custom Labels manifest/'
local_images_path='path to COCO images/'
coco_manifest = 'COCO dataset JSON file name'
coco_json_file = local_path + coco_manifest
job_name='Custom Labels job name'
cl_manifest_file = 'custom_labels.manifest'

label_attribute = 'bounding-box'

open(local_path + cl_manifest_file, 'w').close()

# class representing a Custom Label JSON line for an image
class cl_json_line:
    def __init__(self, job, img):

        #Get image info. Annotations are dealt with seperately
        sizes=[]
        image_size={}
        image_size["width"] = img["width"]
        image_size["depth"] = 3
        image_size["height"] = img["height"]
        sizes.append(image_size)

        bounding_box={}
        bounding_box["annotations"] = []
        bounding_box["image_size"] = sizes

        self.__dict__["source-ref"] = s3_path + img['file_name']
        self.__dict__[job] = bounding_box

        #get metadata
        metadata = {}
```

```
    metadata['job-name'] = job_name
    metadata['class-map'] = {}
    metadata['human-annotated']='yes'
    metadata['objects'] = []
    date_time_obj = datetime.datetime.strptime(img['date_captured'], '%Y-%m-%d
%H:%M:%S')
    metadata['creation-date']= date_time_obj.strftime('%Y-%m-%dT%H:%M:%S')
    metadata['type']='groundtruth/object-detection'

    self.__dict__[job + '-metadata'] = metadata

print("Getting image, annotations, and categories from COCO file...")

with open(coco_json_file) as f:

    #Get custom label compatible info
    js = json.load(f)
    images = js['images']
    categories = js['categories']
    annotations = js['annotations']

    print('Images: ' + str(len(images)))
    print('annotations: ' + str(len(annotations)))
    print('categories: ' + str(len(categories)))

print("Creating CL JSON lines...")

images_dict = {image['id']: cl_json_line(label_attribute, image) for image in
images}

print('Parsing annotations...')
for annotation in annotations:

    image=images_dict[annotation['image_id']]

    cl_annotation = {}
    cl_class_map={}

    # get bounding box information
    cl_bounding_box={}
    cl_bounding_box['left'] = annotation['bbox'][0]
    cl_bounding_box['top'] = annotation['bbox'][1]
```

```
cl_bounding_box['width'] = annotation['bbox'][2]
cl_bounding_box['height'] = annotation['bbox'][3]
cl_bounding_box['class_id'] = annotation['category_id']

getattr(image, label_attribute)['annotations'].append(cl_bounding_box)

for category in categories:
    if annotation['category_id'] == category['id']:
        getattr(image, label_attribute + '-metadata')['class-map']
[category['id']] = category['name']

cl_object={}
cl_object['confidence'] = int(1) #not currently used by Custom Labels
getattr(image, label_attribute + '-metadata')['objects'].append(cl_object)

print('Done parsing annotations')

# Create manifest file.
print('Writing Custom Labels manifest...')

for im in images_dict.values():

    with open(local_path+cl_manifest_file, 'a+') as outfile:
        json.dump(im.__dict__,outfile)
        outfile.write('\n')
        outfile.close()

# Upload manifest file to S3 bucket.
print ('Uploading Custom Labels manifest file to S3 bucket')
print('Uploading' + local_path + cl_manifest_file + ' to ' +
s3_key_path_manifest_file)
print(s3_bucket)
s3 = boto3.resource('s3')
s3.Bucket(s3_bucket).upload_file(local_path + cl_manifest_file,
s3_key_path_manifest_file + cl_manifest_file)

# Print S3 URL to manifest file,
print ('S3 URL Path to manifest file. ')
print('\033[1m s3://' + s3_bucket + '/' + s3_key_path_manifest_file +
cl_manifest_file + '\033[0m')
```

```
# Display aws s3 sync command.
print ('\nAWS CLI s3 sync command to upload your images to S3 bucket. ')
print ('\033[1m aws s3 sync ' + local_images_path + ' ' + s3_path + '\033[0m')
```

3. コードを実行します。
4. プログラムの出力で、s3 sync コマンドを記録しておきます。それは次の手順で必要となります。
5. コマンドラインプロンプトで、s3 sync コマンドを実行します。イメージが S3 バケットにアップロードされます。アップロード中にコマンドが失敗した場合は、ローカルイメージ S3 バケットと同期されるまでコマンドを再度実行してください。
6. プログラムの出力で、マニフェストファイルへの S3 URL パスを記録しておきます。それは次の手順で必要となります。
7. 「[SageMaker Ground Truth マニフェストファイルを使用したデータセットの作成 \(コンソール\)](#)」の手順に従って、アップロードしたマニフェストファイルを使用してデータセットを作成します。ステップ 8 では、[マニフェストファイルの場所] に、前のステップで記録しておいた Amazon S3 URL を入力します。AWS SDK を使用している場合は、[SageMaker Ground Truth マニフェストファイル \(SDK\) を使用したデータセットの作成](#) を実行します。

## マルチラベルの SageMaker Ground Truth マニフェストファイルの変換

このトピックでは、マルチラベルの Amazon SageMaker Ground Truth マニフェストファイルを Amazon Rekognition Custom Labels 形式のマニフェストファイルに変換する方法について説明します。

SageMaker マルチラベルジョブの Ground Truth マニフェストファイルは、Amazon Rekognition Custom Labels 形式のマニフェストファイルとは異なる形式になっています。マルチラベル分類は、イメージを複数のクラスに分類しても同時に複数のクラスに属する場合があります。この場合、イメージには football と ball などの複数のラベル (マルチラベル) が付けられる可能性があります。

マルチラベル SageMaker Ground Truth ジョブの詳細については、「[イメージ分類 \(マルチラベル\)](#)」を参照してください。マルチラベル形式の Amazon Rekognition Custom Labels マニフェストファイルの詳細については、「[the section called “複数のイメージレベルのラベルをイメージに追加する”](#)」を参照してください。

## SageMaker Ground Truth ジョブのマニフェストファイルの取得

次の手順は、Amazon SageMaker Ground Truth ジョブの出カマニフェストファイル (output.manifest) を取得する方法を示しています。次の手順の入力として output.manifest を使用します。

SageMaker Ground Truth ジョブマニフェストファイルをダウンロードするには

1. <https://console.aws.amazon.com/sagemaker/> を開きます。
2. ナビゲーションペインで [Ground Truth] を選択し、次に [ラベリングジョブ] を選択します。
3. 使用するマニフェストファイルを含むラベリングジョブを選択します。
4. 詳細ページで、[出力データセットの場所] の下のリンクを選択します。そのデータセットの場所で Amazon S3 コンソールが開きます。
5. [Manifests]、[output]、[output.manifest] の順に選択します。
6. オブジェクトをダウンロードするには、[オブジェクトアクション]、[ダウンロード] の順に選択します。

## マルチラベルマ SageMaker マニフェストファイルの変換

次の手順では、既存のマルチラベル形式のマニフェストファイルからマルチラベル形式の Amazon Rekognition Custom Labels SageMaker GroundTruth マニフェストファイルを作成します。

### Note

コードを実行するには、Python バージョン 3 以降が必要です。

マルチラベルマ SageMaker マニフェストファイルを変換するには

1. 次の Python コードを実行します。 [SageMaker Ground Truth ジョブのマニフェストファイルの取得](#) で作成したマニフェストファイルの名前をコマンドライン引数として指定します。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Purpose
Shows how to create and Amazon Rekognition Custom Labels format
manifest file from an Amazon SageMaker Ground Truth Image
Classification (Multi-label) format manifest file.
```

```
"""
import json
import logging
import argparse
import os.path

logger = logging.getLogger(__name__)

def create_manifest_file(ground_truth_manifest_file):
    """
    Creates an Amazon Rekognition Custom Labels format manifest file from
    an Amazon SageMaker Ground Truth Image Classification (Multi-label) format
    manifest file.
    :param: ground_truth_manifest_file: The name of the Ground Truth manifest file,
    including the relative path.
    :return: The name of the new Custom Labels manifest file.
    """

    logger.info('Creating manifest file from %s', ground_truth_manifest_file)
    new_manifest_file =
    f'custom_labels_{os.path.basename(ground_truth_manifest_file)}'

    # Read the SageMaker Ground Truth manifest file into memory.
    with open(ground_truth_manifest_file) as gt_file:
        lines = gt_file.readlines()

    # Iterate through the lines one at a time to generate the
    # new lines for the Custom Labels manifest file.
    with open(new_manifest_file, 'w') as the_new_file:
        for line in lines:
            # job_name - The of the Amazon Sagemaker Ground Truth job.
            job_name = ''
            # Load in the old json item from the Ground Truth manifest file
            old_json = json.loads(line)

            # Get the job name
            keys = old_json.keys()
            for key in keys:
                if 'source-ref' not in key and '-metadata' not in key:
                    job_name = key

            new_json = {}
            # Set the location of the image
            new_json['source-ref'] = old_json['source-ref']
```

```
# Temporarily store the list of labels
labels = old_json[job_name]

# Iterate through the labels and reformat to Custom Labels format
for index, label in enumerate(labels):
    new_json[f'{job_name}{index}'] = index
    metadata = {}
    metadata['class-name'] = old_json[f'{job_name}-metadata']['class-
map'][str(label)]
    metadata['confidence'] = old_json[f'{job_name}-metadata']
['confidence-map'][str(label)]
    metadata['type'] = 'groundtruth/image-classification'
    metadata['job-name'] = old_json[f'{job_name}-metadata']['job-name']
    metadata['human-annotated'] = old_json[f'{job_name}-metadata']
['human-annotated']
    metadata['creation-date'] = old_json[f'{job_name}-metadata']
['creation-date']
    # Add the metadata to new json line
    new_json[f'{job_name}{index}-metadata'] = metadata
    # Write the current line to the json file
    the_new_file.write(json.dumps(new_json))
    the_new_file.write('\n')

logger.info('Created %s', new_manifest_file)
return new_manifest_file

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "manifest_file", help="The Amazon SageMaker Ground Truth manifest file"
        "that you want to use."
    )

def main():
    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")
    try:
        # get command line arguments
```



```
parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
add_arguments(parser)
args = parser.parse_args()
# Create the manifest file
manifest_file = create_manifest_file(args.manifest_file)
print(f'Manifest file created: {manifest_file}')
except FileNotFoundError as err:
    logger.exception('File not found: %s', err)
    print(f'File not found: {err}. Check your manifest file.')

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

2. スクリプトに表示される新しいマニフェストファイルの名前を記録しておきます。これは次のステップで使います。
3. マニフェストファイルを保存するために使用する Amazon S3 バケットに[マニフェストファイルをアップロード](#)します。

#### Note

Amazon Rekognition Custom Labels が、マニフェストファイルの JSON 行の source-ref フィールドで参照されている Amazon S3 バケットにアクセスできることを確認してください。詳細については、「[外部の Amazon S3 バケットへのアクセス](#)」を参照してください。Ground Truth ジョブが Amazon Rekognition Custom Labels コンソールバケットにイメージを保存する場合、アクセス許可を追加する必要はありません。

4. 「[SageMaker Ground Truth マニフェストファイルを使用したデータセットの作成 \(コンソール\)](#)」の手順に従って、アップロードしたマニフェストファイルを使用してデータセットを作成します。ステップ 8 では、[マニフェストファイルの場所] に、マニフェストファイルの場所として Amazon S3 URL を入力します。AWS SDK を使用している場合は、[を実行します SageMaker Ground Truth マニフェストファイル \(SDK\) を使用したデータセットの作成。](#)

## CSV ファイルからのマニフェストファイルの作成

この Python スクリプト例では、カンマ区切り値 (CSV) ファイルを使用してイメージにラベルを付けることにより、マニフェストファイルの作成を簡素化しています。ユーザーによって CSV ファイルを作成します。マニフェストファイルは[マルチラベルイメージ分類](#)または[マルチラベルイメージ分類](#)に適しています。詳細については、「[オブジェクト、シーン、概念を検出する](#)」を参照してください。

**Note**

このスクリプトでは、[オブジェクトの位置](#)や[ブランドの位置](#)の検索に適したマニフェストファイルは作成されません。

マニフェストファイルには、モデルのトレーニングに使用されるイメージが記述されています。例えば、イメージの位置やイメージに割り当てられたラベルなどです。マニフェストファイルは、1行以上の JSON 行で構成されます。各 JSON Lines では 1 つの画像について記述されます。詳細については、「[the section called “マニフェストファイル内のイメージレベルのラベル”](#)」を参照してください。

CSV ファイルは、テキストファイル内の複数行にわたる表形式のデータを表します。行内のフィールドはカンマで区切られます。詳細については、「[カンマ区切り値](#)」を参照してください。このスクリプトでは、CSV ファイルの各行が 1 つのイメージを表し、マニフェストファイルの JSON 行にマッピングされます。[マルチラベルイメージ分類](#)をサポートするマニフェストファイル用の CSV ファイルを作成するには、各行に 1 つ以上のイメージレベルのラベルを追加します。[画像分類](#)に適したマニフェストファイルを作成するには、各行に 1 つのイメージレベルのラベルを追加します。

例えば、次の CSV ファイルには [マルチラベルイメージ分類](#) (花) 入門プロジェクトのイメージが記述されています。

```
camellia1.jpg,camellia,with_leaves
camellia2.jpg,camellia,with_leaves
camellia3.jpg,camellia,without_leaves
helleborus1.jpg,helleborus,without_leaves,not_fully_grown
helleborus2.jpg,helleborus,with_leaves,fully_grown
helleborus3.jpg,helleborus,with_leaves,fully_grown
jonquil1.jpg,jonquil,with_leaves
jonquil2.jpg,jonquil,with_leaves
jonquil3.jpg,jonquil,with_leaves
jonquil4.jpg,jonquil,without_leaves
mauve_honey_myrtle1.jpg,mauve_honey_myrtle,without_leaves
mauve_honey_myrtle2.jpg,mauve_honey_myrtle,with_leaves
mauve_honey_myrtle3.jpg,mauve_honey_myrtle,with_leaves
mediterranean_spurge1.jpg,mediterranean_spurge,with_leaves
mediterranean_spurge2.jpg,mediterranean_spurge,without_leaves
```

このスクリプトは、各行に JSON 行を生成します。例えば、以下は最初の行の JSON 行 (camellia1.jpg,camellia,with\_leaves) です。

```

{"source-ref": "s3://bucket/flowers/train/camellia1.jpg","camellia": 1,"camellia-
metadata":{"confidence": 1,"job-name": "labeling-job/camellia","class-name":
"camellia","human-annotated": "yes","creation-date": "2022-01-21T14:21:05","type":
"groundtruth/image-classification"},"with_leaves": 1,"with_leaves-metadata":
{"confidence": 1,"job-name": "labeling-job/with_leaves","class-name":
"with_leaves","human-annotated": "yes","creation-date": "2022-01-21T14:21:05","type":
"groundtruth/image-classification"}}

```

CSV の例では、イメージへの Amazon S3 パスは存在しません。CSV ファイルにイメージの Amazon S3 パスが含まれていない場合は、`--s3_path` コマンドライン引数を使用してイメージへの Amazon S3 パスを指定します。

このスクリプトは、各イメージの最初のエントリを重複排除されたイメージ CSV ファイルに記録します。重複排除されたイメージ CSV ファイルには、入力 CSV ファイルで見つかった各イメージのインスタンスが含まれています。入力 CSV ファイル内でイメージがさらに見つかり、重複イメージ CSV ファイルに記録されます。スクリプトによって重複イメージが検知された場合は、重複イメージ CSV ファイルを確認し、必要に応じて重複排除されたイメージ CSV ファイルを更新します。重複排除されたファイルを使用してスクリプトを再度実行します。入力 CSV ファイルに重複が見つからなかった場合、スクリプトは重複排除されたイメージ CSV ファイルと重複イメージ CSV ファイルが空として削除します。

この手順では、CSV ファイルを作成し、Python スクリプトを実行してマニフェストファイルを作成します。

CSV ファイルからマニフェストファイルを作成するには

1. 各行に以下のフィールドを含む CSV ファイルを作成します (1 画像ごとに 1 行)。CSV ファイルにはヘッダー行を追加しないでください。

フィールド 1	フィールド 2	フィールド n
イメージ名または Amazon S3 へのイメージのパス。 例えば <code>s3://my-bucket/flowers/train/camellia1.jpg</code> です。Amazon S3 パスのイメージと、パスがないイメー	イメージの最初のイメージレベルのラベル。	カンマで区切られた 1 つ以上の追加のイメージレベルのラベル。 <a href="#">マルチラベルイメージ分類</a> をサポートするマニフェストファイルを作成する場合にのみ追加してください。

フィールド 1	フィールド 2	フィールド n
ジを混在させることはできません。		

例えば、`camellia1.jpg,camellia,with_leaves`、`s3://my-bucket/flowers/train/camellia1.jpg,camellia,with_leaves` などです。

2. CSV ファイルを保存します。

3. 以下の Python スクリプトを実行します。以下の情報を提供します:

- `csv_file` - ステップ 1 で作成した CSV ファイル。
- `manifest_file` - 作成するマニフェストファイルの名前。
- (オプション) `--s3_path s3://path_to_folder/` - イメージファイル名に追加する Amazon S3 パス (フィールド 1)。この段階でフィールド 1 のイメージに S3 パスが含まれていない場合は、`--s3_path` を使用します。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

from datetime import datetime, timezone
import argparse
import logging
import csv
import os
import json

"""
Purpose
Amazon Rekognition Custom Labels model example used in the service documentation.
Shows how to create an image-level (classification) manifest file from a CSV file.
You can specify multiple image level labels per image.
CSV file format is
image,label,label,..
If necessary, use the bucket argument to specify the S3 bucket folder for the
images.
https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/md-gt-cl-transform.html
"""
```

```
logger = logging.getLogger(__name__)

def check_duplicates(csv_file, deduplicated_file, duplicates_file):
    """
    Checks for duplicate images in a CSV file. If duplicate images
    are found, deduplicated_file is the deduplicated CSV file - only the first
    occurrence of a duplicate is recorded. Other duplicates are recorded in
    duplicates_file.
    :param csv_file: The source CSV file.
    :param deduplicated_file: The deduplicated CSV file to create. If no duplicates
    are found
    this file is removed.
    :param duplicates_file: The duplicate images CSV file to create. If no
    duplicates are found
    this file is removed.
    :return: True if duplicates are found, otherwise false.
    """

    logger.info("Deduplicating %s", csv_file)

    duplicates_found = False

    # Find duplicates.
    with open(csv_file, 'r', newline='', encoding="UTF-8") as f,\
        open(deduplicated_file, 'w', encoding="UTF-8") as dedup,\
        open(duplicates_file, 'w', encoding="UTF-8") as duplicates:

        reader = csv.reader(f, delimiter=',')
        dedup_writer = csv.writer(dedup)
        duplicates_writer = csv.writer(duplicates)

        entries = set()
        for row in reader:
            # Skip empty lines.
            if not ''.join(row).strip():
                continue

            key = row[0]
            if key not in entries:
                dedup_writer.writerow(row)
                entries.add(key)
            else:
                duplicates_writer.writerow(row)
```

```
        duplicates_found = True

    if duplicates_found:
        logger.info("Duplicates found check %s", duplicates_file)

    else:
        os.remove(duplicates_file)
        os.remove(deduplicated_file)

    return duplicates_found

def create_manifest_file(csv_file, manifest_file, s3_path):
    """
    Reads a CSV file and creates a Custom Labels classification manifest file.
    :param csv_file: The source CSV file.
    :param manifest_file: The name of the manifest file to create.
    :param s3_path: The S3 path to the folder that contains the images.
    """
    logger.info("Processing CSV file %s", csv_file)

    image_count = 0
    label_count = 0

    with open(csv_file, newline='', encoding="UTF-8") as csvfile, \
        open(manifest_file, "w", encoding="UTF-8") as output_file:

        image_classifications = csv.reader(
            csvfile, delimiter=',', quotechar='|')

        # Process each row (image) in CSV file.
        for row in image_classifications:
            source_ref = str(s3_path)+row[0]

            image_count += 1

            # Create JSON for image source ref.
            json_line = {}
            json_line['source-ref'] = source_ref

            # Process each image level label.
            for index in range(1, len(row)):
                image_level_label = row[index]
```

```
        # Skip empty columns.
        if image_level_label == '':
            continue
        label_count += 1

    # Create the JSON line metadata.
    json_line[image_level_label] = 1
    metadata = {}
    metadata['confidence'] = 1
    metadata['job-name'] = 'labeling-job/' + image_level_label
    metadata['class-name'] = image_level_label
    metadata['human-annotated'] = "yes"
    metadata['creation-date'] = \
        datetime.now(timezone.utc).strftime('%Y-%m-%dT%H:%M:%S.%f')
    metadata['type'] = "groundtruth/image-classification"

    json_line[f'{image_level_label}-metadata'] = metadata

    # Write the image JSON Line.
    output_file.write(json.dumps(json_line))
    output_file.write('\n')

output_file.close()
logger.info("Finished creating manifest file %s\nImages: %s\nLabels: %s",
            manifest_file, image_count, label_count)

return image_count, label_count

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "csv_file", help="The CSV file that you want to process."
    )

    parser.add_argument(
        "--s3_path", help="The S3 bucket and folder path for the images."
        " If not supplied, column 1 is assumed to include the S3 path.",
        required=False
    )
```

```
def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        s3_path = args.s3_path
        if s3_path is None:
            s3_path = ''

        # Create file names.
        csv_file = args.csv_file
        file_name = os.path.splitext(csv_file)[0]
        manifest_file = f'{file_name}.manifest'
        duplicates_file = f'{file_name}-duplicates.csv'
        deduplicated_file = f'{file_name}-deduplicated.csv'

        # Create manifest file, if there are no duplicate images.
        if check_duplicates(csv_file, deduplicated_file, duplicates_file):
            print(f"Duplicates found. Use {duplicates_file} to view duplicates "
                  f"and then update {deduplicated_file}. ")
            print(f"{deduplicated_file} contains the first occurrence of a
duplicate. "
                  "Update as necessary with the correct label information.")
            print(f"Re-run the script with {deduplicated_file}")
        else:
            print("No duplicates found. Creating manifest file.")

            image_count, label_count = create_manifest_file(csv_file,
                                                            manifest_file,
                                                            s3_path)

            print(f"Finished creating manifest file: {manifest_file} \n"
                  f"Images: {image_count}\nLabels: {label_count}")

    except FileNotFoundError as err:
```



```
logger.exception("File not found: %s", err)
print(f"File not found: {err}. Check your input CSV file.")

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

4. テストデータセットを使用する場合は、ステップ 1~3 を繰り返してテストデータセットのマニフェストファイルを作成します。
5. 必要に応じて、CSV ファイルの列 1 で指定した (--s3\_path または コマンドラインで指定した) Amazon S3 バケットパスにイメージをコピーします。次の AWS S3 コマンドを使用できます。

```
aws s3 cp --recursive your-local-folder s3://your-target-S3-location
```

6. マニフェストファイルを保存するために使用する Amazon S3 バケットに [マニフェストファイルをアップロード](#) します。

#### Note

Amazon Rekognition Custom Labels が、マニフェストファイルの JSON 行の source-ref フィールドで参照されている Amazon S3 バケットにアクセスできることを確認してください。詳細については、「[外部の Amazon S3 バケットへのアクセス](#)」を参照してください。Ground Truth ジョブが Amazon Rekognition Custom Labels コンソールバケットにイメージを保存する場合、アクセス許可を追加する必要はありません。

7. 「[SageMaker Ground Truth マニフェストファイルを使用したデータセットの作成 \(コンソール\)](#)」の手順に従って、アップロードしたマニフェストファイルを使用してデータセットを作成します。ステップ 8 では、[マニフェストファイルの場所] に、マニフェストファイルの場所として Amazon S3 URL を入力します。AWS SDK を使用している場合は、[を実行します SageMaker Ground Truth マニフェストファイル \(SDK\) を使用したデータセットの作成](#)。

## 既存のデータセット

以前にデータセットを作成した場合は、その内容を新しいデータセットにコピーできます。AWS SDK を使用して既存のデータセットからデータセットを作成するには、「[」を参照してください](#) [既存のデータセットを使用したデータセットの作成 \(SDK\)](#)。

既存の Amazon Rekognition Custom Labels データセットを使用してデータセットを作成するには (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. [カスタムラベルを使用] を選択します。
3. [開始方法] を選択します。
4. 左側のナビゲーションペインで、[プロジェクト] を選択します。
5. 「プロジェクト」ページで、データセットを追加したいプロジェクトを選択します。プロジェクトの詳細ページが表示されます。
6. [データセットを作成] を選択します。「データセットを作成」ページが表示されます。
7. [設定の開始] で、[1 つのデータセットで開始] または [トレーニングデータセットで開始] を選択します。より高品質のモデルを作成するには、トレーニングデータセットとテストデータセットを別々に始めることを推奨します。

#### Single dataset

- a. [トレーニングデータセットの詳細] セクションで、[既存の Amazon Rekognition Custom Labels データセットをコピー] を選択します。
- b. [トレーニングデータセットの詳細] セクションの[データセット] 編集ボックスで、コピーするデータセットの名前を入力または選択します。
- c. [データセットを作成] を選択します。プロジェクトのデータセットページが開きます。

#### Separate training and test datasets

- a. [トレーニングデータセットの詳細] セクションで、[既存の Amazon Rekognition Custom Labels データセットをコピー] を選択します。
- b. [トレーニングデータセットの詳細] セクションの[データセット] 編集ボックスで、コピーするデータセットの名前を入力または選択します。
- c. [テストデータセットの詳細] セクションで、[既存の Amazon Rekognition Custom Labels データセットをコピー] を選択します。
- d. [テストデータセットの詳細] セクションの[データセット] 編集ボックスで、コピーするデータセットの名前を入力または選択します。

**Note**

トレーニングデータセットとテストデータセットは、異なる画像ソースを持つことができます。

- e. [データセットを作成] を選択します。プロジェクトのデータセットページが開きます。
8. ラベルの追加または変更の必要がある場合は、[イメージにラベルを付ける](#) を実行してください。
9. 「[モデルのトレーニング \(コンソール\)](#)」の手順に従って、モデルをトレーニングします。

## イメージにラベルを付ける

ラベルは、イメージ内のオブジェクトを囲むオブジェクト、シーン、概念、または境界ボックスを識別します。例えば、データセットに犬のイメージが含まれている場合は、犬種のラベルを追加できます。

データセットにイメージをインポートした後で、イメージへのラベルの追加や、誤ったラベルが付いたイメージの修正が必要になる場合があります。例えば、ローカルコンピュータからインポートされたイメージにはラベルは付けられません。データセットギャラリーを使用して、データセットに新しいラベルを追加し、データセット内のイメージにラベルと境界ボックスを割り当てます。

データセット内のイメージに付けるラベルによって、Amazon Rekognition Custom Labels がトレーニングするモデルのタイプが決まります。詳細については、「[データセットの目的の設定](#)」を参照してください。

### トピック

- [ラベルの管理](#)
- [イメージにイメージレベルのラベルを割り当てる](#)
- [境界ボックスによるオブジェクトのラベル付け](#)

## ラベルの管理

Amazon Rekognition Custom Labels コンソールを使用してラベルを管理できます。ラベルを管理するための特定の API はありません。ラベルは、CreateDataset でデータセットを作成したとき、または UpdateDatasetEntries でデータセットにイメージを追加したときにデータセットに追加されます。

## トピック

- [ラベルの管理 \(コンソール\)](#)
- [ラベルの管理 \(SDK\)](#)

### ラベルの管理 (コンソール)

Amazon Rekognition Custom Labels コンソールを使用して、データセットのラベルを追加、変更、または削除できます。データセットにラベルを追加するには、自分で作成した新しいラベルを追加するか、Rekognition の既存のデータセットからラベルをインポートします。

## トピック

- [新しいラベルの追加 \(コンソール\)](#)
- [ラベルの変更と削除 \(コンソール\)](#)

### 新しいラベルの追加 (コンソール)

データセットに追加する新しいラベルを指定できます。

編集ウィンドウを使用してラベルを追加します。

### 新しいラベルを追加するには (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. [カスタムラベルを使用] を選択します。
3. [開始方法] を選択します。
4. 左ナビゲーションペインで、[プロジェクト] を選択します。
5. 「プロジェクト」 ページで、削除するプロジェクトを選択します。プロジェクトの詳細ページが表示されます。
6. トレーニングデータセットにラベルを追加する場合は、[トレーニング] タブを選択します。それ以外の場合は、[テスト] タブを選択してテストデータセットにラベルを追加します。
7. [ラベル付けを開始] を選択してラベル付けモードに入ります。
8. データセットギャラリーの [ラベル] セクションで [ラベルを管理] を選択し、[ラベルを管理] ダイアログボックスを開きます。
9. 編集ボックスに新しいラベル名を入力します。
10. [新しいラベルを追加] を選択します。

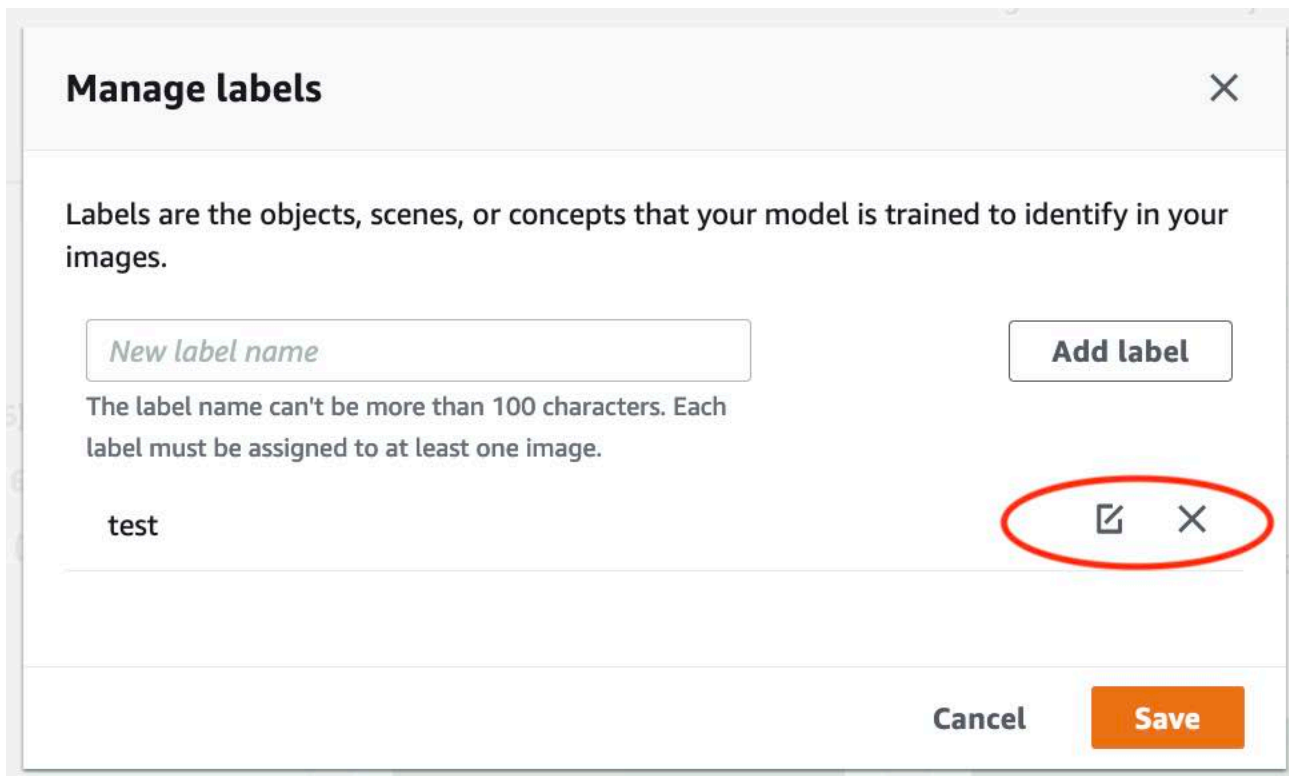
11. 必要なラベルがすべて作成されるまで、ステップ 9 と 10 を繰り返します。
12. [保存] を選択して、追加したラベルを保存します。

### ラベルの変更と削除 (コンソール)

データセットに追加した後に、ラベルの名前を変更したり、削除したりできます。削除できるのは、どのイメージにも割り当てられていないラベルのみです。

### 既存のラベルの名前を変更または削除するには (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. [カスタムラベルを使用] を選択します。
3. [開始方法] を選択します。
4. 左ナビゲーションペインで、[プロジェクト] を選択します。
5. 「プロジェクト」 ページで、削除するプロジェクトを選択します。プロジェクトの詳細ページが表示されます。
6. トレーニングデータセットのラベルを変更または削除する場合は、[トレーニング] タブを選択します。それ以外の場合は、[テスト] タブを選択してテストデータセットのラベルを変更または削除します。
7. [ラベル付けを開始] を選択してラベル付けモードに入ります。
8. データセットギャラリーの [ラベル] セクションで [ラベルを管理] を選択し、[ラベルを管理] ダイアログボックスを開きます。
9. 編集または削除するラベルを選択します。



- a. 削除アイコン (X) を選択すると、ラベルがリストから削除されます。
- b. ラベルを変更する場合は、編集アイコン (鉛筆と紙パッド) を選択し、編集ボックスに新しいラベル名を入力します。

10. [保存] を選択して変更を保存します。

## ラベルの管理 (SDK)

データセットのラベルを管理する独自の API はありません。CreateDataset でデータセットを作成する場合は、マニフェストファイルまたはコピーされたデータセットにあるラベルで、ラベルの初期セットを作成します。UpdateDatasetEntries API を使用してイメージをさらに追加すると、エントリにある新しいラベルがデータセットに追加されます。詳細については、「[イメージの追加 \(SDK\)](#)」を参照してください。データセットからラベルを削除するには、データセット内のすべてのラベルの注釈を削除する必要があります。

データセットからラベルを削除するには

1. ListDatasetEntries を呼び出してデータセットのエントリを取得します。サンプルコードについては、「[データセットエントリの一覧表示 \(SDK\)](#)」を参照してください。

2. ファイル内のラベルの注釈をすべて削除します。詳細については、「[マニフェストファイル内のイメージレベルのラベル](#)」および「[the section called “マニフェストファイル内のオブジェクトのローカリゼーション”](#)」を参照してください。
3. このファイルを使用して UpdateDatasetEntries API でデータセットを更新します。詳細については、「[イメージの追加 \(SDK\)](#)」を参照してください。

## イメージにイメージレベルのラベルを割り当てる

イメージレベルのラベルを使用して、イメージをカテゴリに分類するモデルをトレーニングします。イメージレベルのラベルは、イメージにオブジェクト、シーン、または概念が含まれていることを示しています。例えば、次のイメージは川を示しています。モデルがイメージに川を含むものとして分類した場合、river のイメージレベルのラベルを追加することになります。詳細については、「[データセットの目的の設定](#)」を参照してください。





イメージレベルのラベルを含むデータセットには、少なくとも 2 つのラベルを定義する必要があります。各イメージには、イメージ内のオブジェクト、シーン、または概念を識別するラベルが少なくとも 1 つ割り当てられている必要があります。

画像レベルのラベルを画像に割り当てるには (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. [カスタムラベルを使用] を選択します。
3. [開始方法] を選択します。
4. 左ナビゲーションペインで、[プロジェクト] を選択します。
5. 「プロジェクト」 ページで、削除するプロジェクトを選択します。プロジェクトの詳細ページが表示されます。
6. 左のナビゲーションペインの [データセット] を選択します。
7. トレーニングデータセットにラベルを追加する場合は、[トレーニング] タブを選択します。それ以外の場合は、[テスト] タブを選択してテストデータセットにラベルを追加します。
8. [ラベル付けを開始] を選択してラベル付けモードに入ります。
9. イメージギャラリーで、ラベルを追加するイメージを 1 つ以上選択します。一度に選択できるのは 1 ページのイメージのみです。1 ページの連続した範囲のイメージを選択するには
  - a. 範囲内にある最初のイメージを選択します。
  - b. Shift キーを押したままにします。
  - c. 最後のイメージ範囲を選択します。最初のイメージと 2 番目のイメージ間のイメージも選択されます。
  - d. Shift キーを放します。
10. [画像レベルのラベルを割り当てる] を選択します。
11. [画像レベルのラベルを選択した画像に割り当てる] ダイアログボックスで、1 つ以上のイメージに割り当てるラベルを選択します。
12. [割り当てる] を選択して、イメージにラベルを割り当てます。
13. すべてのイメージに必要なラベルが付けられるまで、ラベル付けを繰り返します。
14. [変更を保存] を選択して、変更を保存します。



## 画像レベルのラベルを割り当てる (SDK)

UpdateDatasetEntries API を使用して、イメージに割り当てられているイメージレベルのラベルを追加または更新できます。UpdateDatasetEntries は 1 行以上の JSON 行を使用します。各 JSON 行は 1 つのイメージを表します。イメージレベルのラベルが付いたイメージの場合、JSON 行は次のようになります。

```
{"source-ref":"s3://custom-labels-console-us-east-1-nnnnnnnnnn/gt-job/manifest/IMG_1133.png","TestCLConsoleBucket":0,"TestCLConsoleBucket-metadata":{"confidence":0.95,"job-name":"labeling-job/testclconsolebucket","class-name":"Echo Dot","human-annotated":"yes","creation-date":"2020-04-15T20:17:23.433061","type":"groundtruth/image-classification"}}
```

source-ref フィールドはイメージの場所を示します。JSON 行には、イメージに割り当てられたイメージレベルのラベルも含まれています。詳細については、「[the section called “マニフェストファイル内のイメージレベルのラベル”](#)」を参照してください。

画像に画像レベルのラベルを割り当てるには

1. ListDatasetEntries を使用して既存のイメージの JSON 行を取得します。source-ref フィールドには、ラベルを割り当てるイメージの場所を指定します。詳細については、「[データセットエントリの一覧表示 \(SDK\)](#)」を参照してください。
2. [マニフェストファイル内のイメージレベルのラベル](#) の情報を使用して、前のステップで返された JSON 行を更新します。
3. UpdateDatasetEntries を呼び出してイメージを更新します。詳細については、「[データセットへのイメージの追加](#)」を参照してください。

## 境界ボックスによるオブジェクトのラベル付け

モデルにイメージ内のオブジェクトの位置を検知させる場合は、そのオブジェクトが何で、イメージ内のどこにあるかを特定する必要があります。境界ボックスは、イメージ内のオブジェクトを分離するボックスです。境界ボックスを使用して、同じイメージ内のさまざまなオブジェクトを検知するようにモデルをトレーニングします。オブジェクトを識別するには、境界ボックスにラベルを割り当てます。

**Note**

イメージレベルのラベルでオブジェクト、シーン、概念を検索するようにモデルをトレーニングする場合は、このステップを実行する必要はありません。

例えば、Amazon Echo Dot デバイスを検知するモデルをトレーニングする場合、イメージ内の各 Echo Dot の周囲に境界ボックスを描画し、その境界ボックスに Echo Dot という名前のラベルを割り当てます。次のイメージは、Echo Dot デバイスを囲む境界ボックスを示しています。イメージには、境界ボックスのない Amazon Echo も含まれています。

**境界ボックスでオブジェクトを検索する (コンソール)**

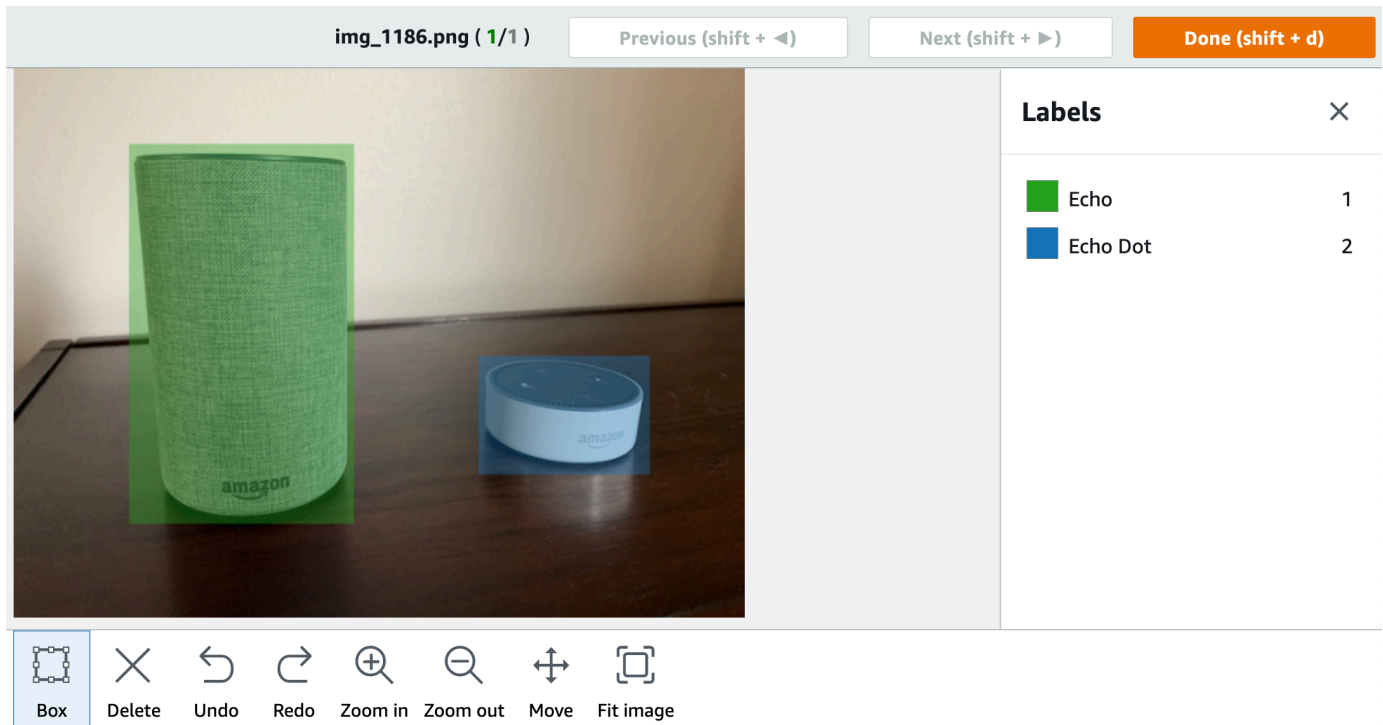
この手順では、コンソールを使用してイメージ内のオブジェクトの周囲に境界ボックスを描画します。また、境界ボックスにラベルを割り当てることにより、イメージ内のオブジェクトを識別できます。

**Note**

Safari ブラウザを使用してイメージに境界ボックスを追加することはできません。サポートされるブラウザについては、「[Amazon Rekognition Custom Labels のセットアップ](#)」を参照してください。

境界ボックスを追加する前に、データセットに少なくとも 1 つのラベルを追加する必要があります。詳細については、「[新しいラベルの追加 \(コンソール\)](#)」を参照してください。

1. Amazon Rekognition コンソールを <https://console.aws.amazon.com/rekognition/> で開きます。
2. [カスタムラベルを使用] を選択します。
3. [開始方法] を選択します。
4. 左ナビゲーションペインで、[プロジェクト] を選択します。
5. 「プロジェクト」 ページで、削除するプロジェクトを選択します。プロジェクトの詳細ページが表示されます。
6. プロジェクトの詳細ページで [画像にラベルを付ける] を選択します。
7. トレーニングデータセットのイメージに境界ボックスを追加する場合は、[トレーニング] タブを選択します。それ以外の場合は、[テスト] タブを選択して、テストデータセットのイメージに境界ボックスを追加します。
8. [ラベル付けを開始] を選択してラベル付けモードに入ります。
9. イメージギャラリーで、境界ボックスを追加するイメージを選択します。
10. [境界ボックスを描画] を選択します。境界ボックスエディタが表示される前に、一連のヒントが表示されます。
11. 右側の [ラベル] ペインで、境界ボックスに割り当てるラベルを選択します。
12. 描画ツールで、目的のオブジェクトの左上の領域にポインタを置きます。
13. マウスの左ボタンを押しながら、オブジェクトの周囲にボックスを描画します。オブジェクトのできるだけ近くに境界ボックスを描くようにしてください。
14. マウスボタンを放します。境界ボックスが強調表示されます。
15. ラベル付けするイメージが他にもある場合は、[次へ] を選択します。それ以外の場合は、[完了] を選択してラベル付けを終了します。



16. オブジェクトを含む境界ボックスを各イメージに作成するまで、ステップ 1~7 を繰り返します。
17. [変更の保存] を選択して、変更を保存します。
18. [終了] を選択してラベリングモードを終了します。

### 境界ボックス (SDK) を使用してオブジェクトを検索する

UpdateDatasetEntries API を使用して、イメージのオブジェクトの位置情報を追加または更新できます。UpdateDatasetEntries では 1 行以上の JSON 行を使用します。各 JSON 行は 1 つのイメージを表します。オブジェクトのローカリゼーションの場合、JSON 行では次のように表示されます。

```
{
  "source-ref": "s3://bucket/images/IMG_1186.png",
  "bounding-box": {
    "image_size": [
      {"width": 640, "height": 480, "depth": 3}
    ],
    "annotations": [
      {
        "class_id": 1,
        "top": 251,
        "left": 399,
        "width": 155,
        "height": 101
      },
      {
        "class_id": 0,
        "top": 65,
        "left": 86,
        "width": 220,
        "height": 334
      }
    ],
    "bounding-box-metadata": {
      "objects": [
        {"confidence": 1},
        {"confidence": 1}
      ],
      "class-map": {
        "0": "Echo",
        "1": "Echo Dot"
      },
      "type": "groundtruth/object-detection",
      "human-annotated": "yes",
      "creation-date": "2013-11-18T02:53:27",
      "job-name": "my job"
    }
  }
}
```

source-ref フィールドはイメージの場所を示します。JSON 行には、イメージの各オブジェクトのラベル付き境界ボックスも含まれています。詳細については、「[the section called “マニフェストファイル内のオブジェクトのローカリゼーション”](#)」を参照してください。

イメージに境界ボックスを割り当てるには

1. ListDatasetEntries を使用して既存のイメージの JSON 行を取得します。source-ref フィールドには、イメージレベルのラベルを割り当てるイメージの場所を指定します。詳細については、「[データセットエントリの一覧表示 \(SDK\)](#)」を参照してください。
2. [マニフェストファイル内のオブジェクトのローカリゼーション](#) の情報を使用して、前のステップで返された JSON 行を更新します。
3. UpdateDatasetEntries を呼び出してイメージを更新します。詳細については、「[データセットへのイメージの追加](#)」を参照してください。

## データセットのデバッグ

データセットの作成中に発生する可能性のあるエラーには、ターミナルエラーと非ターミナルエラーの 2 つのタイプがあります。ターミナルエラーにより、データセットの作成や更新が中止されることがあります。非ターミナルエラーは、データセットの作成や更新は中止されません。

トピック

- [ターミナルエラー](#)
- [非ターミナルエラー](#)

### ターミナルエラー

ターミナルエラーには、データセットの作成に失敗するファイルエラーと、Amazon Rekognition Custom Labels がデータセットから削除するコンテンツエラーの 2 つのタイプがあります。コンテンツエラーが多すぎると、データセットの作成は失敗します。

トピック

- [ターミナルファイルエラー](#)
- [ターミナルコンテンツエラー](#)

## ターミナルファイルエラー

次のものはファイルエラーです。ファイルエラーに関する情報は、DescribeDataset を呼び出して、Status および StatusMessage フィールドを確認することにより得ることができます。サンプルコードについては、「[データセットの記述 \(SDK\)](#)」を参照してください。

- [ERROR\\_MANIFEST\\_INACCESSIBLE\\_OR\\_UNSUPPORTED\\_FORMAT](#)
- [ERROR\\_MANIFEST\\_SIZE\\_TOO\\_LARGE](#).
- [ERROR\\_MANIFEST\\_ROWS\\_EXCEEDS\\_MAXIMUM](#)
- [ERROR\\_INVALID\\_PERMISSIONS\\_MANIFEST\\_S3\\_BUCKET](#)
- [ERROR\\_TOO\\_MANY\\_RECORDS\\_IN\\_ERROR](#)
- [ERROR\\_MANIFEST\\_TOO\\_MANY\\_LABELS](#)
- [ERROR\\_INSUFFICIENT\\_IMAGES\\_PER\\_LABEL\\_FOR\\_DISTRIBUTE](#)

### ERROR\_MANIFEST\_INACCESSIBLE\_OR\_UNSUPPORTED\_FORMAT

#### エラーメッセージ

マニフェストファイルの拡張子または内容が無効です。

トレーニングマニフェストファイルまたはテストマニフェストファイルにファイル拡張子がないか、内容が無効です。

エラー ERROR\_MANIFEST\_INACCESSIBLE\_OR\_UNSUPPORTED\_FORMAT を修正するには

- 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で、以下のような可能性のある原因を確認してください。
  - マニフェストファイルにファイル拡張子がない。慣例により、ファイル拡張子は .manifest です。
  - マニフェストファイルの Amazon S3 バケットまたはキーが見つからない。

### ERROR\_MANIFEST\_SIZE\_TOO\_LARGE

#### エラーメッセージ

マニフェストファイルサイズがサポートされている最大サイズを超えています。

トレーニングマニフェストファイルまたはテストマニフェストファイルのサイズ (バイト単位) が大きすぎます。詳細については、「[Amazon Rekognition Custom Labels のガイドラインとクォータ](#)」を参照してください。マニフェストファイルは、JSON 行数は最大数未満でも、最大ファイルサイズを超えることがあります。

Amazon Rekognition Custom Labels コンソールを使用してエラーを修正することはできません。マニフェストファイルのサイズがサポートされている最大サイズを超えています。

エラー `ERROR_MANIFEST_SIZE_TOO_LARGE` を修正するには

1. トレーニングマニフェストとテストマニフェストのどちらかが最大ファイルサイズを超えているかを確認します。
2. マニフェストファイル内で、超過している JSON 行の数を減らします。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

`ERROR_MANIFEST_ROWS_EXCEEDS_MAXIMUM`

エラーメッセージ

マニフェストファイルの行数が多すぎます。

詳細情報

マニフェストファイル内の JSON 行数 (イメージ数) が上限を超えています。この制限は、イメージレベルモデルとオブジェクト位置モデルでは異なります。詳細については、「[Amazon Rekognition Custom Labels のガイドラインとクォータ](#)」を参照してください。

JSON 行エラーは、JSON 行数が `ERROR_MANIFEST_ROWS_EXCEEDS_MAXIMUM` の上限に達するまで検証されます。

Amazon Rekognition Custom Labels コンソールを使用して `ERROR_MANIFEST_ROWS_EXCEEDS_MAXIMUM` エラーを修正することはできません。

`ERROR_MANIFEST_ROWS_EXCEEDS_MAXIMUM` を修正するには

- マニフェスト内で、JSON 行の数を減らします。詳細については、「[マニフェストファイルの作成](#)」を参照してください。



## ERROR\_INVALID\_PERMISSIONS\_MANIFEST\_S3\_BUCKET

### エラーメッセージ

S3 バケットファイルのアクセス許可が正しくない。

Amazon Rekognition Custom Labels には、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を含むバケットへのアクセス許可がありません。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

エラー ERROR\_INVALID\_PERMISSIONS\_MANIFEST\_S3\_BUCKET を修正するには

- トレーニングマニフェストとテストマニフェストを含むバケットのアクセス許可を確認してください。詳細については、「[ステップ 2: Amazon Rekognition Custom Labels コンソールのアクセス許可をセットアップする](#)」を参照してください。

## ERROR\_TOO\_MANY\_RECORDS\_IN\_ERROR

### エラーメッセージ

マニフェストファイルにターミナルエラーが多すぎます。

ERROR\_TOO\_MANY\_RECORDS\_IN\_ERROR を修正するには

- ターミナルコンテンツエラーがある JSON 行 (イメージ) の数を減らしてください。詳細については、「[ターミナルマニフェストコンテンツエラー](#)」を参照してください。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

## ERROR\_MANIFEST\_TOO\_MANY\_LABELS

### エラーメッセージ

マニフェストファイルのラベルが多すぎます。

### 詳細情報

マニフェスト (データセット) 内の固有ラベルの数が上限を超えています。トレーニングデータセットを分割してテストデータセットを作成する場合、ラベルの数は分割後に決定されます。



## ERROR\_MANIFEST\_TOO\_MANY\_LABELS を修正するには (コンソール)

- データセットからラベルを削除します。詳細については、「[ラベルの管理](#)」を参照してください。ラベルはデータセット内のイメージと境界ボックスから自動的に削除されます。

## ERROR\_MANIFEST\_TOO\_MANY\_LABELS を修正するには (JSON 行)

- イメージレベルの JSON 行を含むマニフェスト - イメージにラベルが 1 つしかない場合は、目的のラベルを使用するイメージの JSON 行を削除します。JSON 行に複数のラベルが含まれている場合は、目的のラベルの JSON オブジェクトのみを削除します。詳細については、「[複数のイメージレベルのラベルをイメージに追加する](#)」を参照してください。

オブジェクトの場所が JSON 行を含むマニフェスト - 削除するラベルの境界ボックスと関連付けられたラベル情報を削除します。目的のラベルを含む JSON 行ごとにこれを実行します。class-map 配列と、それに対応する objects および annotations 配列のオブジェクトからラベルを削除する必要があります。詳細については、「[マニフェストファイル内のオブジェクトのローカリゼーション](#)」を参照してください。

## ERROR\_INSUFFICIENT\_IMAGES\_PER\_LABEL\_FOR\_DISTRIBUTE

### エラーメッセージ

マニフェストファイルには、データセットを分散するのに十分な数のラベル付きイメージがありません。

データセットの分散は、Amazon Rekognition Custom Labels がトレーニングデータセットを分割してテストデータセットを作成するときに発生します。DistributeDatasetEntries API を呼び出してデータセットを分割することもできます。

### エラー ERROR\_MANIFEST\_TOO\_MANY\_LABELS を修正するには

- トレーニングデータセットにラベル付きイメージを追加します

### ターミナルコンテンツエラー

ターミナルコンテンツエラーには次のようなものがあります。データセットの作成時に、ターミナルコンテンツエラーのあるイメージはデータセットから削除されます。データセットは、引き続きトレーニングに使用できます。コンテンツエラーが多すぎると、データセットの作成/更新は失敗しま

す。データセットの操作に関連付けられたターミナルコンテンツエラーは、コンソールには表示されず、DescribeDataset や他の API から返されません。データセットにイメージや注釈がないことに気付いた場合は、データセットのマニフェストファイルに次のような問題がないか確認してください。

- JSON 行の長さが長すぎる。最大長は 100,000 文字です。
- JSON 行に source-ref 値がない。
- JSON 行の source-ref 値の形式が無効。
- JSON 行の内容が無効。
- source-ref フィールドの値が複数回出現する。イメージは 1 つのデータセットで 1 回しか参照できません。

各 source-ref フィールドの詳細については、「[マニフェストファイルの作成](#)」を参照してください。

## 非ターミナルエラー

次のものは、データセットの作成または更新中に発生する可能性のある非ターミナルエラーです。これらのエラーは、JSON 行全体を無効化したり、JSON 行内の注釈を無効にしたりする可能性があります。JSON 行にエラーがある場合、その行はトレーニングには使用されません。JSON 行内の注釈にエラーがある場合でも、JSON 行はトレーニングに使用されますが、その注釈は削除されます。JSON 行の詳細については、「[マニフェストファイルの作成](#)」を参照してください。

非ターミナルエラーには、コンソールや ListDatasetEntries API を呼び出すことによりアクセスできます。詳細については、「[データセットエントリの一覧表示 \(SDK\)](#)」を参照してください。

トレーニング中には、次のエラーも返されます。モデルをトレーニングする前に、これらのエラーを修正することを推奨します。詳細については、「[非ターミナル JSON 行検証エラー](#)」を参照してください。

- [「ラベル属性がない」というエラー](#)
- [エラー:ラベル属性形式が無効です](#)
- [ERROR\\_INVALID\\_LABEL\\_ATTRIBUTE\\_METADATA\\_FORMAT](#)
- [ERROR\\_NO\\_VALID\\_LABEL\\_ATTRIBUTES](#)
- [ERROR\\_INVALID\\_BOUNDING\\_BOX](#)
- [ERROR\\_INVALID\\_IMAGE\\_DIMENSION](#)

- [ERROR\\_BOUNDING\\_BOX\\_TOO\\_SMALL](#)
- [ERROR\\_NO\\_VALID\\_ANNOTATIONS](#)
- [ERROR\\_MISSING\\_BOUNDING\\_BOX\\_CONFIDENCE](#)
- [ERROR\\_MISSING\\_CLASS\\_MAP\\_ID](#)
- [ERROR\\_TOO\\_MANY\\_BOUNDING\\_BOXES](#)
- [ERROR\\_UNSUPPORTED\\_USE\\_CASE\\_TYPE](#)
- [ERROR\\_INVALID\\_LABEL\\_NAME\\_LENGTH](#)

## 非ターミナルエラーへのアクセス

コンソールを使用すると、データセット内のどのイメージに非ターミナルエラーがあるかを調査できます。ListDatasetEntries API を呼び出して、エラーメッセージを取得することもできます。詳細については、「[データセットエントリの一覧表示 \(SDK\)](#)」を参照してください。

## 非ターミナルエラーにアクセスするには (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. [カスタムラベルを使用] を選択します。
3. [開始方法] を選択します。
4. 左ナビゲーションペインで、[プロジェクト] を選択します。
5. 「プロジェクト」 ページで、削除するプロジェクトを選択します。プロジェクトの詳細ページが表示されます。
6. トレーニングデータセット内の非ターミナルエラーを表示する場合は、[トレーニング] タブを選択します。それ以外の場合は、[テスト] タブを選択すると、テストデータセット内の非ターミナルエラーが表示されます。
7. データセットギャラリーの [ラベル] セクションで [エラー] を選択します。データセットギャラリーは、エラーのあるイメージのみが表示されるようにフィルタリングされます。
8. イメージの下にある [エラー] を選択すると、エラーコードが表示されます。「[非ターミナル JSON 行検証エラー](#)」の情報をを使用してエラーを修正してください。

The screenshot displays the Amazon Rekognition Custom Labels interface. On the left, the 'Labels' panel shows a list of labels: 'Images (13)', 'Labeled (13)', 'Untagged (0)', 'Errors (1)', 'pot\_resistor (15)', 'comparator (15)', 'ir\_led (14)', and 'ir\_phototransistor (14)'. The 'Errors (1)' option is selected and circled in red. On the right, the 'Images (13)' panel shows a search bar and a preview of an image named 'test\_normal\_8.jpg'. The image shows a blue printed circuit board (PCB) on a green background, with a red bounding box around it. Below the image, a 'Dataset record errors' dialog box is open, also circled in red. The dialog box contains the following text: 'Dataset record errors', 'To fix errors on an image, choose Start labelling button. Select the image to reassign the labels or redraw the bounding boxes.', 'ERROR\_UNSUPPORTED\_USE\_CASE\_TYPE', and 'ERROR\_NO\_VALID\_LABEL\_ATTRIBUTES'. At the bottom of the dialog, there is a red 'Error' icon and text.

## Amazon Rekognition Custom Labels モデルをトレーニングする

Amazon Rekognition Custom Labels コンソールを使用するか、Amazon Rekognition Custom Labels API を使用してモデルをトレーニングできます。モデルトレーニングが失敗した場合は、[失敗したモデルトレーニングのデバッグ](#) の情報を使用して失敗の原因を突き止めてください。

**Note**

モデルのトレーニングに成功するまでの時間に対して課金されます。通常、トレーニングが完了するまでに 30 分から 24 時間かかります。詳細については、「[トレーニング時間](#)」を参照してください。

モデルをトレーニングするたびに、モデルの新しいバージョンが作成されます。Amazon Rekognition Custom Labels は、プロジェクト名とモデル作成時のタイムスタンプを組み合わせたモデル名を作成します。

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画像のコピーを作成します。デフォルトでは、コピーされたイメージは、AWS が所有および管理するキーで保管時に暗号化されます。独自の AWS KMS key の使用を選択することもできます。独自の KMS キーを使用する場合は、KMS キーに対する次のアクセス許可が必要です。

- kms:CreateGrant
- kms:DescribeKey

詳細については、「[AWS Key Management Service の概念](#)」を参照してください。ソース画像には影響がありません。

KMS サーバー側暗号化 (SSE-KMS) を使用して、Amazon S3 バケット内のトレーニングイメージとテストイメージを Amazon Rekognition Custom Labels にコピーする前に暗号化できます。Amazon Rekognition Custom Labels がイメージにアクセスできるようにするには、AWS アカウントに KMS キーに対する次のアクセス許可が必要です。

- kms:GenerateDataKey
- kms:Decrypt

詳細については、「[AWS Key Management Service \(SSE-KMS\) に保存された KMS キーを使用したサーバー側暗号化を使用したデータの保護](#)」を参照してください。

モデルにトレーニングを行うと、そのモデルのパフォーマンスを評価し、モデルを改善できます。詳細については、「[トレーニング済み Amazon Rekognition Custom Labels の改善](#)」を参照してください。

モデルのタグ付けなど、そのほかのモデルタスクについては、「[Amazon Rekognition Custom Labels モデルの管理](#)」を参照してください。

トピック

- [モデルのトレーニング \(コンソール\)](#)
- [モデルのトレーニング \(SDK\)](#)

## モデルのトレーニング (コンソール)

Amazon Rekognition Custom Labels コンソールを使用して、モデルをトレーニングできます。

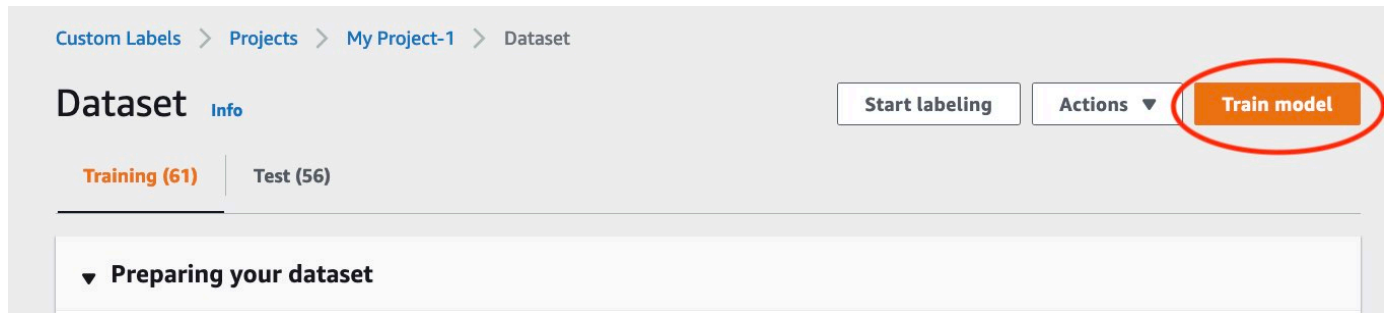
トレーニングには、トレーニングデータセットとテストデータセットを含むプロジェクトが必要です。プロジェクトにテストデータセットがない場合、Amazon Rekognition Custom Labels コンソールはトレーニング中にトレーニングデータセットを分割してプロジェクト用のデータセットを作成します。選択したイメージは代表的なサンプルであり、トレーニングデータセットには使用されません。使用できる代替テストデータセットがない場合にのみ、トレーニングデータセットを分割することをお勧めします。トレーニングデータセットを分割すると、トレーニングに使用できるイメージの数が減ります。

### Note

モデルのトレーニングにかかる時間に対して課金されます。詳細については、「[トレーニング時間](#)」を参照してください。

モデルをトレーニングするには (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. [カスタムラベルを使用] を選択します。
3. 左側のナビゲーションペインで、[プロジェクト] を選択します。
4. [プロジェクト] ページで、トレーニングしたいモデルを含むプロジェクトを選択します。
5. [プロジェクト] ページで、[モデルをトレーニング] を選択します。



6. (オプション) 独自の AWS KMS 暗号化キーを使用する場合は、次の手順を実行します。
  - a. [画像データの暗号化] で [暗号化設定のカスタマイズ (アドバンスド)] を選択します。
  - b. encryption.aws\_kms\_key に、キーの Amazon リソースネーム (ARN) を入力するか、既存の AWS KMS キーを選択します。新しいキーを作成するには、[Create an AWS IMS key] (AWS IMS キーを作成する) を選択します。
7. (オプション) モデルにタグを追加する場合は、次の操作を行います。
  - a. [Tags] (タグ) セクションで [Add new tag] (新しいタグを追加する) を選択します。
  - b. 次のように入力します。
    - i. [Key] (キー) 中のキーの名前。
    - ii. [Value] (値) 中のキーの値。
  - c. タグをさらに追加するには、手順 6a と 6b を繰り返します。
  - d. (オプション) タグを削除したい場合は、削除したいタグの横にある [Remove] (削除) を選びます。以前に保存したタグを削除する場合、変更を保存するとそのタグが削除されます。
8. [モデルをトレーニング] ページで、[モデルをトレーニング] を選択します。プロジェクトの Amazon リソースネーム (ARN) が [プロジェクトを選択] 編集ボックスに入力されているはずで、そうでない場合は、プロジェクトの ARN を入力します。

Custom Labels > Train model

## Train model

### Training details [Info](#)

Choose project  
Amazon Rekognition Custom Labels trains a new version of the model within the project you choose.

arn:aws:rekognition:us-east-

### Tags [Info](#)

A tag is a label that you can assign to your model. Each tag consists of a key and an optional value.

No tags associated with the resource.

[Add new tag](#)

You can add up to 50 more tags.

### Image Data Encryption

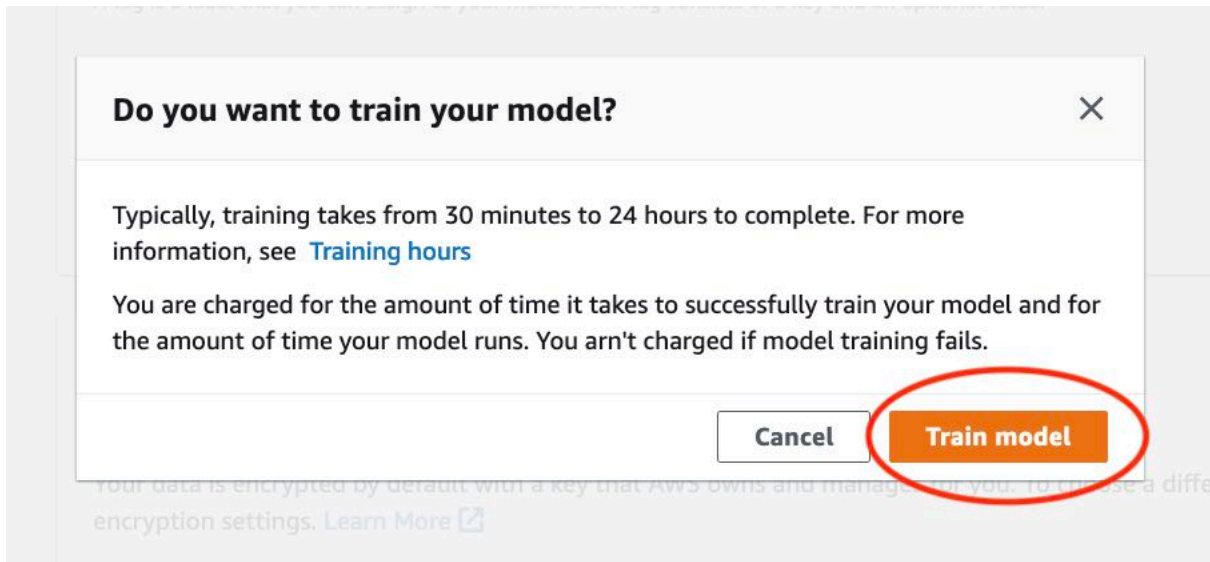
Your data is encrypted by default with a key that AWS owns and manages for you. To choose a different key, customize your encryption settings. [Learn More](#)

Customize encryption settings (advanced)

Cancel **Train Model**

9. [モデルをトレーニングしますか?] ダイアログボックスで、[モデルをトレーニング] を選択します。





10. プロジェクトページの [モデル] セクションでは、トレーニングが進行中の Model Status 列で現在のステータスを確認できます。モデルのトレーニングは、完了までに時間がかかります。

Custom Labels > Projects > My-Project-1

### My-Project-1 Info

**How it works**

**Creating your dataset**

**1. Create dataset**  
A dataset is a collection of images, and image labels, that you use to train or test a model.

Created

**2. Label images**  
Labels identify objects, scenes, or concepts on an entire image, or they identify object locations on an image.

Label images

**Training your model**

**3. Train model**  
Depending on the training dataset, the training model finds image-level scenes and concepts, or it finds object locations.

Train model

**Evaluating your model**

**4. Check performance metrics**  
Performance metrics tell you if your model needs additional training before you can use it.

Check metrics

**Project details**

Project name	Created	Dataset	Models
My-Project-1	October 04, 2021 at 13:05:06 (UTC-07:00)	↳	1

**Models (1)** Delete model Download validation results ▾

🔍 Find resources < 1 ... >

<input type="checkbox"/>	Name	Date created	Training dataset	Test dataset	Model performance (F1 score)	Model status	Status message
<input type="checkbox"/>	My-Project-1.2021-10-04T13.52.53	October 04, 2021			N/A	<b>TRAINING_IN_PROGRESS</b>	The model is being trained.

11. トレーニングが完了したら、モデル名を選択します。モデルのステータスが [TRAINING\_COMPLETED] になったらトレーニングは終了します。トレーニングが失敗した場合は、「[失敗したモデルトレーニングのデバッグ](#)」を読んでください。

The screenshot shows the 'rooms\_19' project page. At the top, there is a 'Delete project' button. Below it is a 'Create datasets' section with an information icon and a close button. The main area is titled 'Models (1)' and contains a search bar and buttons for 'Delete model', 'Download validation results', and 'Train new model'. A table lists the model with the following columns: Name, Date created, Training dataset, Testing dataset, Model performance, Model status, and Status message. The model name 'rooms\_19.2021-07-13T10.36.30' and the status 'TRAINING\_COMPLETED' are circled in red.

Name	Date created	Training dataset	Testing dataset	Model performance	Model status	Status message
rooms_19.2021-07-13T10.36.30	July 13, 2021	rooms_19_training_dataset	rooms_19_test_dataset	0.902	TRAINING_COMPLETED	The model is ready to run.

12. 次のステップ: モデルを評価する。詳細については、「[トレーニング済み Amazon Rekognition Custom Labels の改善](#)」を参照してください。

## モデルのトレーニング (SDK)

モデルをトレーニングするには、[CreateProjectVersion](#) を呼び出します。モデルをトレーニングするには、以下の情報が必要です。

- 名前 - モデルバージョンの一意の名前。
- プロジェクト ARN - モデルを管理するプロジェクトの Amazon リソースネーム (ARN)。
- トレーニング結果の場所 - 結果が置かれている Amazon S3 の場所。コンソールの Amazon S3 バケットと同じ場所を使用することも、別の場所を選択することもできます。別の場所を選択することをお勧めします。これにより、アクセス許可を設定でき、Amazon Rekognition Custom Labels コンソールを使用した場合のトレーニング出力と名前が競合する可能性を回避できます。

トレーニングは、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。詳細については、「[データセットの管理](#)」を参照してください。

### Note

オプションで、プロジェクトの外部にあるトレーニングデータセットとテストデータセットのマニフェストファイルを指定できます。外部のマニフェストファイルを使用してモデルをトレーニングした後にコンソールを開くと、Amazon Rekognition Custom Labels は、トレーニングに使用したマニフェストファイルの最後のセットを使用してデータセットを作成します。外部のマニフェストファイルを指定して、プロジェクトのモデルバージョンをトレーニ

ングすることはできなくなりました。詳細については、「[CreateProjectVersion](#)」を参照してください。

CreateProjectVersion からの応答は、後続のリクエストでモデルバージョンを識別するために使用する ARN です。ARN を使用してモデルバージョンを保護することもできます。詳細については、「[Amazon Rekognition Custom Labels プロジェクトの保護](#)」を参照してください。

モデルバージョンのトレーニングは、完了までに時間がかかります。このトピックの Python と Java の例では、ウェーターを使用してトレーニングが完了するのを待ちます。ウェーターは、発生する特定の状態をポーリングするユーティリティメソッドです。または、DescribeProjectVersions を呼び出すことにより、トレーニングの現在のステータスを取得できます。Status フィールドの値が TRAINING\_COMPLETED になると、トレーニングは完了です。トレーニングが完了したら、評価結果を確認してモデルの品質を評価できます。

## モデルのトレーニング (SDK)

次の例は、プロジェクトに関連する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てモデルをトレーニングする方法を示しています。

モデル (SDK) をトレーニングするには

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. 次のサンプルコードを使用して、プロジェクトをトレーニングします。

### AWS CLI

次の例では、モデルを作成しています。トレーニングデータセットを分割してテストデータセットを作成します。以下に置き換えます:

- プロジェクトの Amazon リソースネーム (ARN) を持つ my\_project\_arn。
- 選択した固有のバージョン名を持つ version\_name。
- Amazon Rekognition Custom Labels がトレーニング結果を保存する Amazon S3 バケットの名前を持つ output\_bucket。
- トレーニング結果を保存するフォルダの名前を持つ output\_folder。

- (オプションパラメータ) AWS Key Management Service カスタマーマスターキーの識別子を持つ `--kms-key-id`。

```
aws rekognition create-project-version \  
  --project-arn project_arn \  
  --version-name version_name \  
  --output-config '{"S3Bucket": "output_bucket", "S3KeyPrefix": "output_folder"}' \  
  \  
  --profile custom-labels-access
```

## Python

次の例では、モデルを作成しています。次のコマンドライン引数を指定します。

- `project_arn` - プロジェクトの Amazon リソースネーム (ARN)。
- `version_name` - 選択したモデルに固有のバージョン名。
- `output_bucket` - Amazon Rekognition Custom Labels がトレーニング結果を保存する Amazon S3 バケットの名前。
- `output_folder` - トレーニング結果を保存するフォルダの名前。

オプションで、以下のコマンドラインパラメータを指定してモデルにタグをアタッチします。

- `tag` - モデルにアタッチする選択したタグ名。
- `tag_value` タグ値。

```
#Copyright 2023 Amazon.com, Inc. or its affiliates. All Rights Reserved.  
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/  
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-  
SAMPLECODE.)
```

```
import argparse  
import logging  
import json  
import boto3
```

```
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def train_model(rek_client, project_arn, version_name, output_bucket,
               output_folder, tag_key, tag_key_value):
    """
    Trains an Amazon Rekognition Custom Labels model.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_arn: The ARN of the project in which you want to train a
    model.
    :param version_name: A version for the model.
    :param output_bucket: The S3 bucket that hosts training output.
    :param output_folder: The path for the training output within output_bucket
    :param tag_key: The name of a tag to attach to the model. Pass None to
    exclude
    :param tag_key_value: The value of the tag. Pass None to exclude
    """

    try:
        #Train the model

        status=""
        logger.info("training model version %s for project %s",
                    version_name, project_arn)

        output_config = json.loads(
            '{"S3Bucket": "'
            + output_bucket
            + '", "S3KeyPrefix": "'
            + output_folder
            + '" } '
        )

        tags={}

        if tag_key is not None and tag_key_value is not None:
            tags = json.loads(
                '{"' + tag_key + '":"' + tag_key_value + '"}'
            )
```

```
response=rek_client.create_project_version(
    ProjectArn=project_arn,
    VersionName=version_name,
    OutputConfig=output_config,
    Tags=tags
)

logger.info("Started training: %s", response['ProjectVersionArn'])

# Wait for the project version training to complete.

project_version_training_completed_waiter =
rek_client.get_waiter('project_version_training_completed')
project_version_training_completed_waiter.wait(ProjectArn=project_arn,
VersionNames=[version_name])

# Get the completion status.

describe_response=rek_client.describe_project_versions(ProjectArn=project_arn,
VersionNames=[version_name])
for model in describe_response['ProjectVersionDescriptions']:
    logger.info("Status: %s", model['Status'])
    logger.info("Message: %s", model['StatusMessage'])
    status=model['Status']

logger.info("finished training")

return response['ProjectVersionArn'], status

except ClientError as err:
    logger.exception("Couldn't create model: %s", err.response['Error']
['Message'] )
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
```

```
        "project_arn", help="The ARN of the project in which you want to train a
model"
    )

    parser.add_argument(
        "version_name", help="A version name of your choosing."
    )

    parser.add_argument(
        "output_bucket", help="The S3 bucket that receives the training
results."
    )

    parser.add_argument(
        "output_folder", help="The folder in the S3 bucket where training
results are stored."
    )

    parser.add_argument(
        "--tag_name", help="The name of a tag to attach to the model",
required=False
    )

    parser.add_argument(
        "--tag_value", help="The value for the tag.", required=False
    )

def main():

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Training model version {args.version_name} for project
{args.project_arn}")
```

```
# Train the model.
session = boto3.Session(profile_name='custom-labels-access')
rekognition_client = session.client("rekognition")

model_arn, status=train_model(rekognition_client,
                              args.project_arn,
                              args.version_name,
                              args.output_bucket,
                              args.output_folder,
                              args.tag_name,
                              args.tag_value)

print(f"Finished training model: {model_arn}")
print(f"Status: {status}")

except ClientError as err:
    logger.exception("Problem training model: %s", err)
    print(f"Problem training model: {err}")
except Exception as err:
    logger.exception("Problem training model: %s", err)
    print(f"Problem training model: {err}")

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

## Java V2

次の例では、モデルをトレーニングしています。次のコマンドライン引数を指定します。

- `project_arn` - プロジェクトの Amazon リソースネーム (ARN)。
- `version_name` - 選択したモデルに固有のバージョン名。
- `output_bucket` - Amazon Rekognition Custom Labels がトレーニング結果を保存する Amazon S3 バケットの名前。
- `output_folder` - トレーニング結果を保存するフォルダの名前。

```
/*
```

```
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```



```
SPDX-License-Identifier: Apache-2.0
*/
package com.example.rekognition;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
    software.amazon.awssdk.services.rekognition.model.CreateProjectVersionRequest;
import
    software.amazon.awssdk.services.rekognition.model.CreateProjectVersionResponse;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsResponse;
import software.amazon.awssdk.services.rekognition.model.OutputConfig;
import
    software.amazon.awssdk.services.rekognition.model.ProjectVersionDescription;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.waiters.RekognitionWaiter;

import java.util.Optional;
import java.util.logging.Level;
import java.util.logging.Logger;

public class TrainModel {

    public static final Logger logger =
        Logger.getLogger(TrainModel.class.getName());

    public static String trainMyModel(RekognitionClient rekClient, String
        projectArn, String versionName,
        String outputBucket, String outputFolder) {

        try {

            OutputConfig outputConfig =
                OutputConfig.builder().s3Bucket(outputBucket).s3KeyPrefix(outputFolder).build();

            logger.log(Level.INFO, "Training Model for project {0}",
                projectArn);
            CreateProjectVersionRequest createProjectVersionRequest =
                CreateProjectVersionRequest.builder()
```

```
.projectArn(projectArn).versionName(versionName).outputConfig(outputConfig).build());

    CreateProjectVersionResponse response =
rekClient.createProjectVersion(createProjectVersionRequest);

    logger.log(Level.INFO, "Model ARN: {0}",
response.projectVersionArn());
    logger.log(Level.INFO, "Training model...");

    // wait until training completes

    DescribeProjectVersionsRequest describeProjectVersionsRequest =
DescribeProjectVersionsRequest.builder()
        .versionNames(versionName)
        .projectArn(projectArn)
        .build();

    RekognitionWaiter waiter = rekClient.waiter();

    WaiterResponse<DescribeProjectVersionsResponse> waiterResponse =
waiter

.waitUntilProjectVersionTrainingCompleted(describeProjectVersionsRequest);

    Optional<DescribeProjectVersionsResponse> optionalResponse =
waiterResponse.matched().response();

    DescribeProjectVersionsResponse describeProjectVersionsResponse =
optionalResponse.get();

    for (ProjectVersionDescription projectVersionDescription :
describeProjectVersionsResponse
        .projectVersionDescriptions()) {
        System.out.println("ARN: " +
projectVersionDescription.projectVersionArn());
        System.out.println("Status: " +
projectVersionDescription.statusAsString());
        System.out.println("Message: " +
projectVersionDescription.statusMessage());
    }

    return response.projectVersionArn();
```

```
    } catch (RekognitionException e) {
        logger.log(Level.SEVERE, "Could not train model: {0}",
e.getMessage());
        throw e;
    }

}

public static void main(String args[]) {

    String versionName = null;
    String projectArn = null;
    String projectVersionArn = null;
    String bucket = null;
    String location = null;

    final String USAGE = "\n" + "Usage: " + "<project_name> <version_name>
<output_bucket> <output_folder>\n\n" + "Where:\n"
        + "    project_arn - The ARN of the project that you want to use.
\n\n"
        + "    version_name - A version name for the model.\n\n"
        + "    output_bucket - The S3 bucket in which to place the
training output. \n\n"
        + "    output_folder - The folder within the bucket that the
training output is stored in. \n\n";

    if (args.length != 4) {
        System.out.println(USAGE);
        System.exit(1);
    }

    projectArn = args[0];
    versionName = args[1];
    bucket = args[2];
    location = args[3];

    try {

        // Get the Rekognition client.
        RekognitionClient rekClient = RekognitionClient.builder()
            .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
            .region(Region.US_WEST_2)
            .build();
```

```
        // Train model
        projectVersionArn = trainMyModel(rekClient, projectArn, versionName,
bucket, location);

        System.out.println(String.format("Created model: %s for Project ARN:
%s", projectVersionArn, projectArn));

        rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    }

}

}
```

3. トレーニングが失敗した場合は、「[失敗したモデルトレーニングのデバッグ](#)」を読んでください。

## 失敗したモデルトレーニングのデバッグ

モデルトレーニング中にエラーが発生する場合があります。Amazon Rekognition Custom Labels は、コンソールと [DescribeProjectVersions](#) からのレスポンスでトレーニングエラーを報告します。

エラーは、ターミナル (トレーニングを継続できない) か、非ターミナル (トレーニングを継続できる) かのいずれかです。トレーニングデータセットとテストデータセットのコンテンツに関連するエラーについては、検証結果 ([マニフェストの概要](#)と[トレーニングとテストの検証マニフェスト](#)) をダウンロードできます。検証結果のエラーコードを使用して、このセクションの詳細情報を確認してください。このセクションには、マニフェストファイルエラー (マニフェストファイルの内容が検証される前に発生するターミナルエラー) に関する情報も記載されています。

### Note

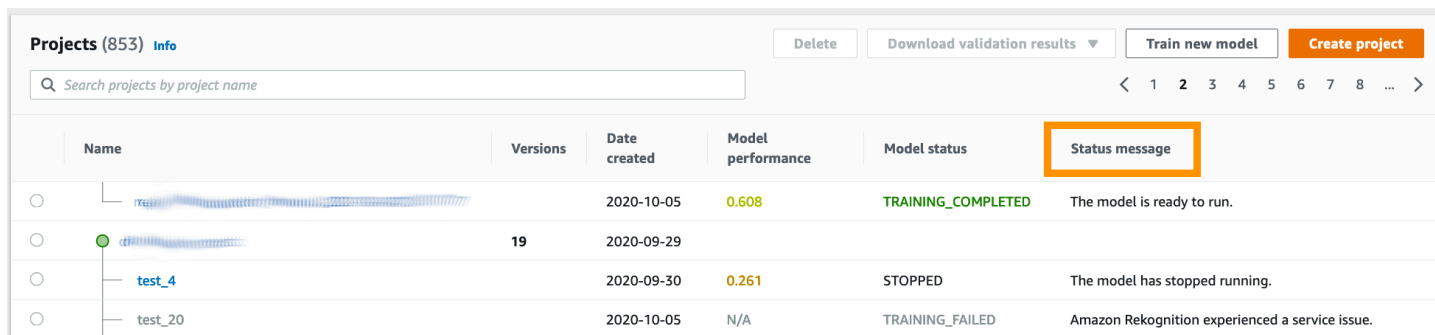
マニフェストは、データセットのコンテンツを保存するために使用されるファイルです。

一部のエラーは、Amazon Rekognition Custom Labels コンソールを使用して修正できます。そのほかのエラーでは、トレーニングマニフェストファイルまたはテストマニフェストファイルの更新が必要となる場合があります。IAM のアクセス許可など、他の変更が必要になる場合があります。詳細については、個々のエラーのドキュメントを参照してください。

## ターミナルエラー

ターミナルエラーが発生すると、モデルのトレーニングが停止します。ターミナルトレーニングエラーには、サービスエラー、マニフェストファイルエラー、マニフェストコンテンツエラーの3つのカテゴリがあります。

コンソールの Amazon Rekognition Custom Labels のプロジェクトページの [ステータスメッセージ] 列には、モデルのターミナルエラーが表示されます。



Name	Versions	Date created	Model performance	Model status	Status message
test_1		2020-10-05	0.608	TRAINING_COMPLETED	The model is ready to run.
test_2	19	2020-09-29			
test_4		2020-09-30	0.261	STOPPED	The model has stopped running.
test_20		2020-10-05	N/A	TRAINING_FAILED	Amazon Rekognition experienced a service issue.

AWS SDK を使用している場合は、[DescribeProjectVersions](#) からの応答を確認して、ターミナルマニフェストファイルエラーまたはターミナルマニフェストコンテンツエラーが発生したかどうかを確認できます。この場合、Status 値は TRAINING\_FAILED で、StatusMessage フィールドにはエラーが含まれています。

## サービスエラー

ターミナルサービスエラーは、Amazon Rekognition でサービスの問題が発生し、トレーニングを継続できない場合に発生します。例えば、Amazon Rekognition Custom Labels が依存する別のサービスの障害などです。Amazon Rekognition でサービスの問題が発生したため、Amazon Rekognition Custom Labels は、コンソールにサービスエラーを報告します。AWS SDK を使用する場合、トレーニング中に発生したサービスエラーは [CreateProjectVersion](#) と [DescribeProjectVersions](#) によって `InternalServerError` 例外として報告されます。

サービスエラーが発生した場合は、モデルのトレーニングを再試行してください。トレーニングが引き続き失敗する場合は、[AWS Support](#) に連絡し、サービスエラーで報告されたエラー情報を記載してください。

## ターミナルマニフェストファイルエラー

マニフェストファイルエラーは、トレーニングデータセットとテストデータセット内の、ファイルレベルまたは複数のファイルにまたがって発生するターミナルエラーです。マニフェストファイルエラーは、トレーニングデータセットとテストデータセットの内容が検証される前に検出されます。マニフェストファイルエラーが発生すると、[非ターミナルの検証エラー](#)は報告されません。例えば、トレーニングマニフェストファイルが空の場合、マニフェストファイルは空ですというエラーが生成されます。ファイルが空であるため、非ターミナルの JSON 行検証エラーは報告されません。マニフェストの概要も作成されません。

モデルをトレーニングする前に、マニフェストファイルエラーを修正する必要があります。

マニフェストファイルエラーを以下に示します。

- [マニフェストファイルの拡張子または内容が無効です。](#)
- [マニフェストファイルが空です。](#)
- [マニフェストファイルサイズがサポートされている最大サイズを超えています。](#)
- [出力 S3 バケットに書き込みできません。](#)
- [S3 バケットファイルのアクセス許可が正しくない。](#)

## ターミナルマニフェストコンテンツエラー

マニフェストコンテンツエラーは、マニフェスト内のコンテンツに関連するターミナルエラーです。例えば、[このマニフェストファイルには、自動分割を実行するには不十分なラベル付きイメージが含まれています](#)というエラーが表示された場合は、トレーニングデータセットにはテストデータセットを作成するのに十分な数のラベル付きイメージがないため、トレーニングを終了できません。

このエラーは、コンソールおよび DescribeProjectVersions からの応答で報告されるだけでなく、他のターミナルマニフェストコンテンツエラーとともにマニフェストの概要でも報告されます。詳細については、「[マニフェストの概要について](#)」を参照してください。

非ターミナル JSON 行エラーは、トレーニングとテストの別の検証結果マニフェストでも報告されます。Amazon Rekognition Custom Labels によって検出される非ターミナル JSON 行エラーは、必ずしもトレーニングを停止するマニフェストコンテンツエラーに関連しているわけではありません。詳細については、「[トレーニングとテストの検証結果マニフェストを理解する](#)」を参照してください。

モデルをトレーニングする前に、マニフェストコンテンツエラーを修正する必要があります。

マニフェストコンテンツエラーのエラーメッセージは次のとおりです。

- [マニフェストファイルに無効な行が多すぎます。](#)
- [マニフェストファイルに複数の S3 バケットのイメージが含まれています。](#)
- [イメージ S3 バケットの所有者 ID が無効です。](#)
- [マニフェストファイルには、ラベルごとのラベル付きイメージが不足しているため、自動分割を実行できません。](#)
- [マニフェストファイルのラベルが少なすぎます。](#)
- [マニフェストファイルのラベルが多すぎます。](#)
- [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間のラベルの重複が {}% 未満です。](#)
- [マニフェストファイルに使用可能なラベルが少なすぎます。](#)
- [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間で重複しているラベルが {}% 未満です。](#)
- [S3 バケットからイメージをコピーできませんでした。](#)

## 非ターミナル JSON 行検証エラー

JSON 行検証エラーは、Amazon Rekognition Custom Labels がモデルのトレーニングを停止する必要がない非ターミナルエラーです。

JSON 行検証エラーはコンソールには表示されません。

トレーニングデータセットとテストデータセットでは、JSON 行は 1 つのイメージのトレーニング情報またはテスト情報を表します。JSON 行の検証エラー (無効なイメージなど) は、トレーニングとテストの検証マニフェストで報告されます。Amazon Rekognition Custom Labels は、マニフェストに含まれる他の有効な JSON 行を使用してトレーニングを完了します。詳細については、「[トレーニングとテストの検証結果マニフェストを理解する](#)」を参照してください。検証ルールの詳細については、「[マニフェストファイルの検証ルール](#)」を参照してください。

### Note

JSON 行エラーが多すぎるとトレーニングは失敗します。

非ターミナル JSON 行エラーも修正することをお勧めします。将来的にエラーが発生したり、モデルのトレーニングに影響を与えたりする可能性があるためです。

Amazon Rekognition Custom Labels は、次のような非ターミナル JSON 行検証エラーを生成する場合があります。

- [ソース参照キーがありません。](#)
- [ソース参照値の形式が無効です。](#)
- [ラベル属性が見つかりません。](#)
- [ラベル属性 {} の形式が無効です。](#)
- [ラベル attributemetadata の形式が無効です。](#)
- [有効なラベル属性が見つかりません。](#)
- [1 つ以上の境界ボックスの信頼値がありません。](#)
- [1 つ以上のクラス ids がクラスマップにありません。](#)
- [JSON 行の形式が無効です。](#)
- [イメージが無効です。S3 パスやイメージのプロパティを確認してください。](#)
- [境界ボックスにオフフレーム値があります。](#)
- [境界ボックスの高さと幅が小さすぎます。](#)
- [境界ボックスが許容最大数を超過しています。](#)
- [有効な注釈が見つかりません。](#)

## マニフェストの概要について

マニフェストの概要には次の情報が含まれています。

- 検証中に発生した [ターミナルマニフェストコンテンツエラー](#) に関するエラー情報。
- トレーニングデータセットとテストデータセット内の [非ターミナル JSON 行検証エラー](#) のエラー位置情報。
- トレーニングデータセットとテストデータセットで見つかった無効な JSON 行の総数などのエラー統計。

[ターミナルマニフェストファイルエラー](#) がない場合、マニフェストの概要はトレーニング中に作成されます。マニフェストの概要ファイル (manifest\_summary.json) の場所を取得するには、「[検証結果の取得](#)」を参照してください。



**Note**

[サービスエラー](#)と[マニフェストファイルエラー](#)は、マニフェストの概要には報告されません。詳細については、「[ターミナルエラー](#)」を参照してください。

特定のマニフェストコンテンツエラーについては、「[ターミナルマニフェストコンテンツエラー](#)」を参照してください。

## マニフェストの概要ファイル形式

マニフェストファイルには、statisticsとerrorsの2つのセクションがあります。

### 統計

statisticsには、トレーニングデータセットとテストデータセットのエラーに関する情報が含まれています。

- training - トレーニングデータセットで見つかった統計とエラー。
- testing - テストデータセットで見つかった統計とエラー。

errors 配列内のオブジェクトには、マニフェストコンテンツエラーのエラーコードとメッセージが含まれます。

error\_line\_indices 配列には、トレーニングマニフェストまたはテストマニフェスト内のエラーのある各 JSON 行の行番号が含まれます。詳細については、「[トレーニングエラーの修正](#)」を参照してください。

### エラー

トレーニングデータセットとテストデータセットの両方にまたがるエラー。例えば、[ERROR\\_INSUFFICIENT\\_USABLE\\_LABEL\\_OVERLAP](#) は、トレーニングデータセットとテストデータセットと重複する使用可能なラベルが十分でない場合に発生します。

```
{
  "statistics": {
    "training": {
      "use_case": String, # Possible values are IMAGE_LEVEL_LABELS,
OBJECT_LOCALIZATION and NOT_DETERMINED
```

```

        "total_json_lines": Number, # Total number json lines (images) in the
training manifest.
        "valid_json_lines": Number, # Total number of JSON Lines (images)
that can be used for training.
        "invalid_json_lines": Number, # Total number of invalid JSON Lines.
They are not used for training.
        "ignored_json_lines": Number, # JSON Lines that have a valid schema but
have no annotations. The aren't used for training and aren't counted as invalid.
        "error_json_line_indices": List[int], # Contains a list of line numbers
for JSON line errors in the training dataset.
        "errors": [
            {
                "code": String, # Error code for a training manifest content
error.
                "message": String # Description for a training manifest content
error.
            }
        ]
    },
    "testing":
    {
        "use_case": String, # Possible values are IMAGE_LEVEL_LABELS,
OBJECT_LOCALIZATION and NOT_DETERMINED
        "total_json_lines": Number, # Total number json lines (images) in the
manifest.
        "valid_json_lines": Number, # Total number of JSON Lines (images) that
can be used for testing.
        "invalid_json_lines": Number, # Total number of invalid JSON Lines.
They are not used for testing.
        "ignored_json_lines": Number, # JSON Lines that have a valid schema but
have no annotations. They aren't used for testing and aren't counted as invalid.
        "error_json_line_indices": List[int], # contains a list of error record
line numbers in testing dataset.
        "errors": [
            {
                "code": String, # # Error code for a testing manifest content
error.
                "message": String # Description for a testing manifest content
error.
            }
        ]
    }
},
"errors": [

```

```
{
  "code": String, # # Error code for errors that span the training and
testing datasets.
  "message": String # Description of the error.
}
]
```

## マニフェストの概要の例

以下の例は、ターミナルマニフェストコンテンツエラー

([ERROR\\_TOO\\_MANY\\_INVALID\\_ROWS\\_IN\\_MANIFEST](#)) を示すマニフェストの概要の一部です。error\_json\_line\_indices 配列には、対応するトレーニングのまたはテストの検証マニフェストに含まれる非ターミナル JSON 行エラーの行番号が含まれています。

```
{
  "errors": [],
  "statistics": {
    "training": {
      "use_case": "NOT_DETERMINED",
      "total_json_lines": 301,
      "valid_json_lines": 146,
      "invalid_json_lines": 155,
      "ignored_json_lines": 0,
      "errors": [
        {
          "code": "ERROR_TOO_MANY_INVALID_ROWS_IN_MANIFEST",
          "message": "The manifest file contains too many invalid rows."
        }
      ],
      "error_json_line_indices": [
        15,
        16,
        17,
        22,
        23,
        24,
        .
        .
        .
        .
        300
      ]
    }
  }
}
```

```
    },
    "testing": {
      "use_case": "NOT_DETERMINED",
      "total_json_lines": 15,
      "valid_json_lines": 13,
      "invalid_json_lines": 2,
      "ignored_json_lines": 0,
      "errors": [],
      "error_json_line_indices": [
        13,
        15
      ]
    }
  }
}
```

## トレーニングとテストの検証結果マニフェストを理解する

トレーニング中、Amazon Rekognition Custom Labels は、非ターミナル JSON 行エラーを含む検証結果マニフェストを作成します。検証結果マニフェストは、エラー情報が追加されたトレーニングデータセットとテストデータセットのコピーです。検証マニフェストには、トレーニングの完了後にアクセスできます。詳細については、「[検証結果の取得](#)」を参照してください。Amazon Rekognition Custom Labels では、エラーの場所や JSON 行エラー数などの JSON 行エラーの概要情報を含むマニフェストの概要も作成されます。詳細については、「[マニフェストの概要について](#)」を参照してください。

### Note

検証結果 (トレーニングとテストの検証結果のマニフェストとマニフェストの概要) は、[ターミナルマニフェストファイルエラー](#) がない場合にのみ作成されます。

マニフェストには、データセット内の各イメージの JSON 行が含まれます。検証結果マニフェスト内では、エラーが発生した JSON 行に JSON 行エラー情報が追加されます。

JSON 行エラーは、1つのイメージに関連する非ターミナルエラーです。非ターミナル検証エラーでは、JSON 行全体または一部のみが無効になることがあります。例えば、JSON 行で参照されているイメージが PNG または JPG 形式でない場合、[ERROR\\_INVALID\\_IMAGE](#) エラーが発生し、JSON 行全体がトレーニングから除外されます。トレーニングは、他の有効な JSON 行で継続されます。

JSON 行内でエラーが発生しても、JSON 行は引き続きトレーニングに使用できる場合があります。例えば、ラベルに関連付けられている 4 つの境界ボックスのうちの 1 つの左側の値が負の場合でも、モデルは他の有効な境界ボックスを使用してトレーニングされます。無効な境界ボックス ([ERROR\\_INVALID\\_BOUNDING\\_BOX](#)) の JSON 行エラー情報が返されます。この例では、エラーが発生した annotation オブジェクトにエラー情報が追加されます。

[WARNING\\_NO\\_ANNOTATIONS](#) などの警告エラーはトレーニングには使用されず、マニフェストの概要では無視された JSON 行 (`ignored_json_lines`) としてカウントされます。詳細については、「[マニフェストの概要について](#)」を参照してください。また、無視された JSON 行は、トレーニングとテストの 20% のエラーしきい値にはカウントされません。

特定の非ターミナルデータ検証エラーについては、「[非ターミナル JSON 行検証エラー](#)」を参照してください。

#### Note

データ検証エラーが多すぎる場合、トレーニングは停止され、マニフェストの概要に [ERROR\\_TOO\\_MANY\\_INVALID\\_ROWS\\_IN\\_MANIFEST](#) ターミナルエラーが報告されます。

JSON 行エラーの修正方法については、「[トレーニングエラーの修正](#)」を参照してください。

## JSON 行エラー形式

Amazon Rekognition Custom Labels は、非ターミナル検証エラー情報をイメージレベルとオブジェクトローカリゼーション形式 JSON 行に追加します。詳細については、「[the section called “マニフェストファイルの作成”](#)」を参照してください。

### イメージレベルエラー

次の例は、イメージレベル JSON 行の Error 配列を示しています。エラーには 2 つのセットがあります。ラベル属性メタデータ (この例では `sport-metadata`) に関連するエラーとイメージに関連するエラー。エラーには、エラーコード (コード)、エラーメッセージ (メッセージ) が含まれます。詳細については、「[マニフェストファイル内のイメージレベルのラベル](#)」を参照してください。

```
{
  "source-ref": String,
  "sport": Number,
  "sport-metadata": {
    "class-name": String,
```

```
    "confidence": Float,  
    "type": String,  
    "job-name": String,  
    "human-annotated": String,  
    "creation-date": String,  
    "errors": [  
      {  
        "code": String, # error codes for label  
        "message": String # Description and additional contextual details of  
the error  
      }  
    ],  
    "errors": [  
      {  
        "code": String, # error codes for image  
        "message": String # Description and additional contextual details of the  
error  
      }  
    ]  
  }  
}
```

## オブジェクトローカリゼーションエラー

次の例は、オブジェクトローカリゼーション JSON 行のエラー配列を示しています。JSON 行には、次の JSON 行セクションのフィールドの Errors 配列情報が含まれています。各 Error オブジェクトにはエラーコードとエラーメッセージが含まれます。

- ラベル属性 - ラベル属性フィールドのエラー。例の「bounding-box」を参照してください。
- 注釈 - 注釈エラー (境界ボックス) はラベル属性内の annotations 配列に保存されます。
- ラベル属性メタデータ - ラベル属性メタデータのエラー。例の「bounding-box-metadata」を参照してください。
- イメージ - ラベル属性、注釈、ラベル属性のメタデータフィールドに関係のないエラー。

詳細については、「[マニフェストファイル内のオブジェクトのローカリゼーション](#)」を参照してください。

```
{  
  "source-ref": String,  
  "bounding-box": {  
    "image_size": [  

```

```
    {
      "width": Int,
      "height": Int,
      "depth": Int,
    }
  ],
  "annotations": [
    {
      "class_id": Int,
      "left": Int,
      "top": Int,
      "width": Int,
      "height": Int,
      "errors": [ # annotation field errors
        {
          "code": String, # annotation field error code
          "message": String # Description and additional contextual
details of the error
        }
      ]
    }
  ],
  "errors": [ #label attribute field errors
    {
      "code": String, # error code
      "message": String # Description and additional contextual details of
the error
    }
  ]
},
"bounding-box-metadata": {
  "objects": [
    {
      "confidence": Float
    }
  ],
  "class-map": {
    String: String
  },
  "type": String,
  "human-annotated": String,
  "creation-date": String,
  "job-name": String,
  "errors": [ #metadata field errors
```

```
    {
      "code": String, # error code
      "message": String # Description and additional contextual details of
the error
    }
  ],
},
"errors": [ # image errors
  {
    "code": String, # error code
    "message": String # Description and additional contextual details of the
error
  }
]
```

## JSON 行エラーの例

次のオブジェクトローカリゼーション JSON 行 (読みやすいようにフォーマットされています) は [ERROR\\_BOUNDING\\_BOX\\_TOO\\_SMALL](#) エラーを示しています。この例では、境界ボックスの寸法 (高さ と 幅) は 1 x 1 以下です。

```
{
  "source-ref": "s3://bucket/Manifests/images/199940-1791.jpg",
  "bounding-box": {
    "image_size": [
      {
        "width": 3000,
        "height": 3000,
        "depth": 3
      }
    ],
    "annotations": [
      {
        "class_id": 1,
        "top": 0,
        "left": 0,
        "width": 1,
        "height": 1,
        "errors": [
          {
            "code": "ERROR_BOUNDING_BOX_TOO_SMALL",
```



```
        "message": "The height and width of the bounding box is too
small."
    }
  ]
},
{
  "class_id": 0,
  "top": 65,
  "left": 86,
  "width": 220,
  "height": 334
}
]
},
"bounding-box-metadata": {
  "objects": [
    {
      "confidence": 1
    },
    {
      "confidence": 1
    }
  ],
  "class-map": {
    "0": "Echo",
    "1": "Echo Dot"
  },
  "type": "groundtruth/object-detection",
  "human-annotated": "yes",
  "creation-date": "2019-11-20T02:57:28.288286",
  "job-name": "my job"
}
}
```

## 検証結果の取得

検証結果には、[ターミナルマニフェストコンテンツエラー](#) および [非ターミナル JSON 行検証エラー](#) のエラー情報が含まれます。検証結果ファイルは 3 つあります。

- training\_manifest\_with\_validation.json - JSON 行のエラー情報が追加されたトレーニングデータセットマニフェストファイルのコピー。

- testing\_manifest\_with\_validation.json - JSON 行エラーエラー情報が追加されたテストデータセットマニフェストファイルのコピー。
- manifest\_summary.json - トレーニングデータセットとテストデータセットで見つかったマニフェストコンテンツエラーと JSON 行エラーの概要です。詳細については、「[マニフェストの概要について](#)」を参照してください。

トレーニングとテストの検証マニフェストのコンテンツについては、「[失敗したモデルトレーニングのデバッグ](#)」を参照してください。

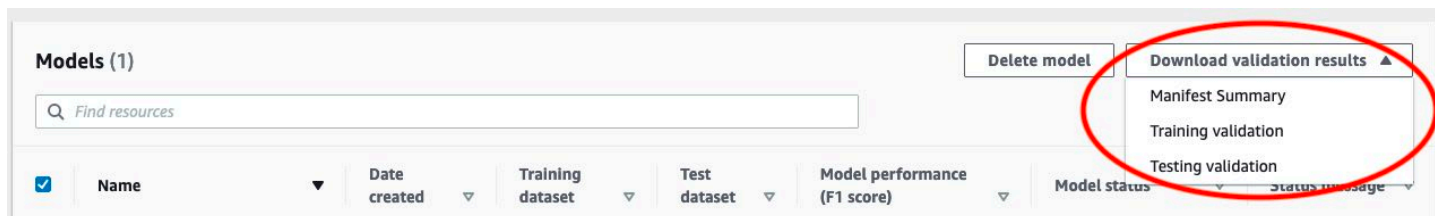
#### Note

- 検証結果は、トレーニング中に [ターミナルマニフェストファイルエラー](#) が生成されなかった場合にのみ作成されます。
- トレーニングマニフェストとテストマニフェストの検証後に [サービスエラー](#) が発生した場合、検証結果は作成されますが、[DescribeProjectVersions](#) からの応答には検証結果ファイルの場所は含まれません。

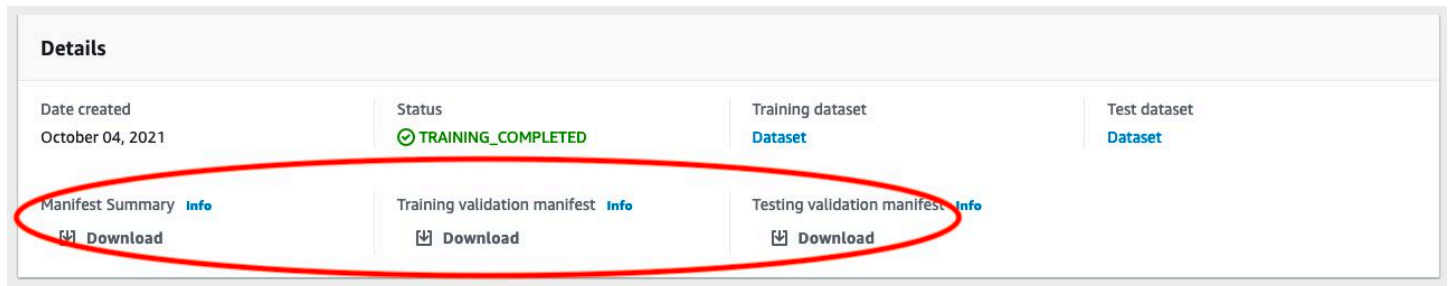
トレーニングが完了または失敗した後は、Amazon Rekognition Custom Labels コンソールを使用して検証結果をダウンロードするか、[DescribeProjectVersions](#) API を呼び出して Amazon S3 バケットの場所を取得できます。

## 検証結果の取得 (コンソール)

コンソールを使用してモデルをトレーニングする場合、次の図に示すように、プロジェクトのモデルリストから検証結果をダウンロードできます。



モデルの詳細ページから検証結果のダウンロードにアクセスすることもできます。



Details			
Date created October 04, 2021	Status 🟢 TRAINING_COMPLETED	Training dataset Dataset	Test dataset Dataset
Manifest Summary <a href="#">Info</a> 📄 Download	Training validation manifest <a href="#">Info</a> 📄 Download	Testing validation manifest <a href="#">Info</a> 📄 Download	

詳細については、「[モデルのトレーニング \(コンソール\)](#)」を参照してください。

## 検証結果 (SDK) の取得

モデルトレーニングが完了すると、Amazon Rekognition Custom Labels はトレーニング中に指定された Amazon S3 バケットに検証結果を保存します。S3 バケットの場所は、トレーニングの完了後に [DescribeProjectVersions](#) API を呼び出して取得できます。モデルをトレーニングするには、「[モデルのトレーニング \(SDK\)](#)」を参照してください。

[ValidationData](#) オブジェクトは、トレーニングデータセット ([TrainingDataResult](#)) とテストデータセット ([TestingDataResult](#)) に対して返されます。マニフェストの概要は、ManifestSummary で返されます。

Amazon S3 バケットの場所を取得したら、検証結果をダウンロードできます。詳細については、「[S3 バケットからオブジェクトをダウンロードする方法](#)」を参照してください。また、[GetObject](#) オペレーションを使用することもできます。

検証データ (SDK) を取得するには

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. 次の例を使用して、検証結果の場所を取得します。

Python

project\_arn を、モデルを含むプロジェクトの Amazon リソースネーム (ARN) に置き換えます。詳細については、「[Amazon Rekognition Custom Labels プロジェクトの管理](#)」を参照してください。version\_name をモデルバージョンの名前に置き換えます。詳細については、「[モデルのトレーニング \(SDK\)](#)」を参照してください。

```
import boto3
```

```
import io
from io import BytesIO
import sys
import json

def describe_model(project_arn, version_name):

    client=boto3.client('rekognition')

    response=client.describe_project_versions(ProjectArn=project_arn,
        VersionNames=[version_name])

    for model in response['ProjectVersionDescriptions']:
        print(json.dumps(model,indent=4,default=str))

def main():

    project_arn='project_arn'
    version_name='version_name'

    describe_model(project_arn, version_name)

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

3. プログラム出力で、TestingDataResult および TrainingDataResult オブジェクト内の Validation フィールドを書き留めてください。マニフェストの概要は、ManifestSummary にあります。

## トレーニングエラーの修正

マニフェストの概要を使用して、トレーニング中に発生した [ターミナルマニフェストコンテンツエラー](#) と [非ターミナル JSON 行検証エラー](#) を特定します。マニフェストコンテンツエラーは修正する必要があります。非ターミナル JSON 行エラーも修正することをお勧めします。特定のエラーについては、「[非ターミナル JSON 行検証エラー](#)」および「[ターミナルマニフェストコンテンツエラー](#)」を参照してください。

トレーニングに使用したトレーニングデータセットまたはテストデータセットを修正できます。または、トレーニングとテストの検証マニフェストファイルに修正を加え、それを使用してモデルをトレーニングすることもできます。

修正を加えたら、更新したマニフェストをインポートしてモデルを再トレーニングする必要があります。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

次の手順は、マニフェストの概要を使用して、ターミナルマニフェストコンテンツエラーを修正する方法です。また、この手順では、トレーニングとテスト検証マニフェスト内の JSON 行エラーを見つけて修正する方法も示しています。

Amazon Rekognition Custom Labels のトレーニングエラーを修正するには

1. 検証結果ファイルをダウンロードします。ファイル名は、`training_manifest_with_validation.json`、`testing_manifest_with_validation.json`、`manifest_summary.json` です。詳細については、「[検証結果の取得](#)」を参照してください。
2. マニフェストサマリーファイル (`manifest_summary.json`) を開きます。
3. マニフェストの概要のエラーを修正します。詳細については、「[マニフェストの概要について](#)」を参照してください。
4. マニフェストの概要で、`training` の `error_line_indices` 配列を反復処理し、対応する JSON 行番号で `training_manifest_with_validation.json` のエラーを修正します。詳細については、「[the section called “トレーニングとテストの検証結果マニフェストを理解する”](#)」を参照してください。
5. `testing` の `error_line_indices` 配列を反復処理し、対応する JSON 行番号で `testing_manifest_with_validation.json` のエラーを修正します。
6. 検証マニフェストファイルをトレーニングデータセットとテストデータセットとして使用して、モデルを再トレーニングします。詳細については、「[the section called “モデルのトレーニング”](#)」を参照してください。

AWS SDK を使用していて、トレーニング検証データまたはテスト検証データのマニフェストファイルエラーを修正することを選択した場合は、[TrainingData](#) および [TestingData](#) 入力パラメータ内の検証データマニフェストファイルの場所を使用して [CreateProjectVersion](#) に渡します。詳細については、「[モデルのトレーニング \(SDK\)](#)」を参照してください。

## JSON 行のエラー優先順位

次の JSON 行エラーが最初に検出されます。これらのエラーのいずれかが発生すると、JSON 行エラーの検証は停止します。他の JSON 行エラーを修正する前に、これらのエラーを修正する必要があります。

- MISSING\_SOURCE\_REF

- ERROR\_INVALID\_SOURCE\_REF\_FORMAT
- 「ラベル属性がない」というエラー
- エラー:ラベル属性形式が無効です
- ERROR\_INVALID\_LABEL\_ATTRIBUTE\_METADATA\_FORMAT
- ERROR\_MISSING\_BOUNDING\_BOX\_CONFIDENCE
- ERROR\_MISSING\_CLASS\_MAP\_ID
- ERROR\_INVALID\_JSON\_LINE

## ターミナルマニフェストファイルエラー

このトピックでは、[ターミナルマニフェストファイルエラー](#) について説明します。マニフェストファイルエラーには、関連するエラーコードはありません。ターミナルマニフェストファイルエラーが発生しても、検証結果マニフェストは作成されません。詳細については、「[マニフェストの概要について](#)」を参照してください。ターミナルマニフェストエラーが発生すると、[非ターミナル JSON 行検証エラー](#) は報告されません。

マニフェストファイルの拡張子または内容が無効です。

トレーニングマニフェストファイルまたはテストマニフェストファイルにファイル拡張子がないか、内容が無効です。

エラーを修正するには、マニフェストファイルの拡張子または内容が無効です。

- 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で、以下のような可能性のある原因を確認してください。
  - マニフェストファイルにファイル拡張子がない。慣例により、ファイル拡張子は `.manifest` です。
  - マニフェストファイルの Amazon S3 バケットまたはキーが見つからない。

マニフェストファイルが空です。

トレーニングに使用したトレーニングマニフェストファイルまたはテストマニフェストファイルは存在しますが、空です。マニフェストファイルには、トレーニングやテストに使用するイメージごとに JSON 行が必要です。

エラーを修正するには、マニフェストファイルが空です。

1. トレーニングマニフェストとテストマニフェストのどちらが空かを確認してください。
2. 空のマニフェストファイルに JSON 行を追加します。詳細については、「[マニフェストファイルの作成](#)」を参照してください。または、コンソールで新しいデータセットを作成します。詳細については、「[the section called “イメージ付きのデータセットの作成”](#)」を参照してください。

マニフェストファイルサイズがサポートされている最大サイズを超えています。

トレーニングマニフェストファイルまたはテストマニフェストファイルのサイズ (バイト単位) が大きすぎます。詳細については、「[Amazon Rekognition Custom Labels のガイドラインとクォータ](#)」を参照してください。マニフェストファイルは、JSON 行数は最大数未満でも、最大ファイルサイズを超えることがあります。

Amazon Rekognition Custom Labels コンソールを使用してエラーを修正することはできません。マニフェストファイルのサイズがサポートされている最大サイズを超えています。

エラーを修正するには、マニフェストファイルのサイズがサポートされている最大サイズを超えています。

1. トレーニングマニフェストとテストマニフェストのどちらが最大ファイルサイズを超えているかを確認します。
2. マニフェストファイル内で、超過している JSON 行の数を減らします。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

S3 バケットファイルのアクセス許可が正しくない。

Amazon Rekognition Custom Labels には、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を含むバケットへのアクセス許可がありません。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

エラーを修正するには S3 バケットのアクセス許可が正しくありません。

- トレーニングマニフェストとテストマニフェストを含むバケットのアクセス許可を確認してください。詳細については、「[ステップ 2: Amazon Rekognition Custom Labels コンソールのアクセス許可をセットアップする](#)」を参照してください。

出力 S3 バケットに書き込みできません。

サービスでは、トレーニング出力ファイルを生成できません。

エラーを修正するには、出力 S3 バケットに書き込めません。

- [CreateProjectVersion](#) への [OutputConfig](#) 入力パラメータの Amazon S3 バケット情報が正しいことを確認してください。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

## ターミナルマニフェストコンテンツエラー

このトピックでは、マニフェストの概要で報告される [ターミナルマニフェストコンテンツエラー](#) について説明します。マニフェストの概要には、検出された各エラーのエラーコードとメッセージが含まれます。詳細については、「[マニフェストの概要について](#)」を参照してください。ターミナルマニフェストコンテンツにエラーがあっても、[非ターミナル JSON 行検証エラー](#) の報告は停止されません。

### ERROR\_TOO\_MANY\_INVALID\_ROWS\_IN\_MANIFEST

#### エラーメッセージ

マニフェストファイルに無効な行が多すぎます。

#### 詳細情報

無効なコンテンツを含む JSON 行が多すぎる

と、ERROR\_TOO\_MANY\_INVALID\_ROWS\_IN\_MANIFEST エラーが発生します。

Amazon Rekognition Custom Labels コンソールを使用し

て、ERROR\_TOO\_MANY\_INVALID\_ROWS\_IN\_MANIFEST エラーを修正することはできません。

ERROR\_TOO\_MANY\_INVALID\_ROWS\_IN\_MANIFEST を修正するには

1. マニフェストに JSON 行エラーがないか確認してください。詳細については、「[トレーニングとテストの検証結果マニフェストを理解する](#)」を参照してください。
2. エラーを含む JSON 行の修正の詳細については、「[非ターミナル JSON 行検証エラー](#)」を参照してください。



## ERROR\_IMAGES\_IN\_MULTIPLE\_S3\_BUCKETS

### エラーメッセージ

マニフェストファイルに複数の S3 バケットのイメージが含まれています。

### 詳細情報

マニフェストは 1 つのバケットに保存されているイメージのみを参照できます。各 JSON 行には、イメージの場所の Amazon S3 の場所が source-ref の値に保存されます。次の例では、バケット名は my-bucket です。

```
"source-ref": "s3://my-bucket/images/sunrise.png"
```

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

### ERROR\_IMAGES\_IN\_MULTIPLE\_S3\_BUCKETS を修正するには

- すべてのイメージが同じ Amazon S3 バケットにあり、すべての JSON 行の source-ref の値がイメージの保存先のバケットを参照していることを確認してください。または、優先する Amazon S3 バケットを選択し、source-ref が優先するバケットを参照しない JSON 行を削除します。

## ERROR\_INVALID\_PERMISSIONS\_IMAGES\_S3\_BUCKET

### エラーメッセージ

イメージ S3 バケットのアクセス許可が無効です。

### 詳細情報

イメージを含む Amazon S3 バケットのアクセス許可が正しくありません。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

## ERROR\_INVALID\_PERMISSIONS\_IMAGES\_S3\_BUCKET を修正するには

- イメージを含むバケットのアクセス許可を確認してください。イメージの source-ref の値にはバケットの場所が含まれます。

## ERROR\_INVALID\_IMAGES\_S3\_BUCKET\_OWNER

### エラーメッセージ

イメージ S3 バケットの所有者 ID が無効です。

### 詳細情報

トレーニングイメージまたはテストイメージを含むバケットの所有者は、トレーニングマニフェストまたはテストマニフェストを含むバケットの所有者と異なります。次のコマンドを使用して、バケットの所有者を検索できます。

```
aws s3api get-bucket-acl --bucket bucket name
```

OWNER ID は、イメージとマニフェストファイルを保存するバケットと一致する必要があります。

## ERROR\_INVALID\_IMAGES\_S3\_BUCKET\_OWNER を修正するには

1. トレーニング、テスト、出力、およびイメージバケットの目的の所有者を選択します。所有者には Amazon Rekognition Custom Labels を使用するアクセス許可が必要です。
2. 目的の所有者が現在所有していない各バケットについて、優先所有者が所有する新しい Amazon S3 バケットを作成します。
3. 古いバケットのコンテンツを新しいバケットにコピーします。詳細については、「[Amazon S3 バケット間でオブジェクトをコピーするには、どうしたらよいですか。](#)」を参照してください。。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

## ERROR\_INSUFFICIENT\_IMAGES\_PER\_LABEL\_FOR\_AUTOSPLIT

### エラーメッセージ

マニフェストファイルには、ラベルごとのラベル付きイメージが不足しているため、自動分割を実行できません。

### 詳細情報

モデルトレーニング中に、トレーニングデータセットの 20% のイメージを使用してテストデータセットを作成できます。ERROR\_INSUFFICIENT\_IMAGES\_PER\_LABEL\_FOR\_AUTOSPLIT は、許容できるテストデータセットを作成するのに十分なイメージがない場合に発生します。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

ERROR\_INSUFFICIENT\_IMAGES\_PER\_LABEL\_FOR\_AUTOSPLIT を修正するには

- トレーニングデータセットとテストデータセットにラベル付きイメージを追加します。Amazon Rekognition Custom Labels コンソールにイメージを追加するには、トレーニングデータセットにイメージを追加するか、トレーニングマニフェストに JSON 行を追加します。詳細については、「[データセットの管理](#)」を参照してください。

## ERROR\_MANIFEST\_TOO\_FEW\_LABELS

### エラーメッセージ

マニフェストファイルのラベルが少なすぎます。

### 詳細情報

トレーニングデータセットとテストデータセットには、必要最小限のラベル数があります。最小値は、データセットがイメージレベルのラベル (分類) を検出するようにモデルをトレーニング/テストするのか、モデルがオブジェクトの位置を検出するのかによって異なります。トレーニングデータセットを分割してテストデータセットを作成する場合、データセット内のラベルの数はトレーニングデータセットの分割後に決定されます。詳細については、「[Amazon Rekognition Custom Labels のガイドラインとクォータ](#)」を参照してください。

## ERROR\_MANIFEST\_TOO\_FEW\_LABELS を修正するには (コンソール)

1. データセットに新しいラベルをさらに追加します。詳細については、「[ラベルの管理](#)」を参照してください。
2. データセット内のイメージに新しいラベルを追加します。モデルがイメージレベルのラベルを検出した場合は、「[イメージにイメージレベルのラベルを割り当てる](#)」を参照してください。モデルがオブジェクトの場所を検出した場合は、「[the section called “境界ボックスによるオブジェクトのラベル付け”](#)」を参照してください。

## ERROR\_MANIFEST\_TOO\_FEW\_LABELS を修正するには (JSON 行)

- 新しいラベルが付いた新しいイメージには JSON 行を追加します。詳細については、「[マニフェストファイルの作成](#)」を参照してください。モデルがイメージレベルのラベルを検出した場合は、class-name フィールドに新しいラベル名を追加します。例えば、次のイメージのラベルは、日の出です。

```
{
  "source-ref": "s3://bucket/images/sunrise.png",
  "testdataset-classification_Sunrise": 1,
  "testdataset-classification_Sunrise-metadata": {
    "confidence": 1,
    "job-name": "labeling-job/testdataset-classification_Sunrise",
    "class-name": "Sunrise",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:18:13.527256",
    "type": "groundtruth/image-classification"
  }
}
```

モデルがオブジェクトの場所を検出した場合は、次の例に示すように、class-map に新しいラベルを追加します。

```
{
  "source-ref": "s3://custom-labels-bucket/images/IMG_1186.png",
  "bounding-box": {
    "image_size": [{
      "width": 640,
      "height": 480,
      "depth": 3
    }
  ]
}
```

```
    ]],  
    "annotations": [{  
      "class_id": 1,  
      "top": 251,  
      "left": 399,  
      "width": 155,  
      "height": 101  
    }, {  
      "class_id": 0,  
      "top": 65,  
      "left": 86,  
      "width": 220,  
      "height": 334  
    }]  
  },  
  "bounding-box-metadata": {  
    "objects": [{  
      "confidence": 1  
    }, {  
      "confidence": 1  
    }],  
    "class-map": {  
      "0": "Echo",  
      "1": "Echo Dot"  
    },  
    "type": "groundtruth/object-detection",  
    "human-annotated": "yes",  
    "creation-date": "2018-10-18T22:18:13.527256",  
    "job-name": "my job"  
  }  
}
```

クラスマップテーブルを境界ボックスの注釈にマッピングする必要があります。詳細については、「[マニフェストファイル内のオブジェクトのローカリゼーション](#)」を参照してください。

## ERROR\_MANIFEST\_TOO\_MANY\_LABELS

### エラーメッセージ

マニフェストファイルのラベルが多すぎます。

## 詳細情報

マニフェスト (データセット) 内の固有ラベルの数が上限を超えています。トレーニングデータセットを分割してテストデータセットを作成する場合、ラベルの数は分割後に決定されます。

ERROR\_MANIFEST\_TOO\_MANY\_LABELS を修正するには (コンソール)

- データセットからラベルを削除します。詳細については、「[ラベルの管理](#)」を参照してください。ラベルはデータセット内のイメージと境界ボックスから自動的に削除されます。

ERROR\_MANIFEST\_TOO\_MANY\_LABELS を修正するには (JSON 行)

- イメージレベルの JSON 行を含むマニフェスト - イメージにラベルが 1 つしかない場合は、目的のラベルを使用するイメージの JSON 行を削除します。JSON 行に複数のラベルが含まれている場合は、目的のラベルの JSON オブジェクトのみを削除します。詳細については、「[複数のイメージレベルのラベルをイメージに追加する](#)」を参照してください。

オブジェクトの場所が JSON 行を含むマニフェスト - 削除するラベルの境界ボックスと関連付けられたラベル情報を削除します。目的のラベルを含む JSON 行ごとにこれを実行します。class-map 配列と、それに対応する objects および annotations 配列のオブジェクトからラベルを削除する必要があります。詳細については、「[マニフェストファイル内のオブジェクトのローカリゼーション](#)」を参照してください。

## ERROR\_INSUFFICIENT\_LABEL\_OVERLAP

### エラーメッセージ

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間のラベルの重複が {}% 未満です。

### 詳細情報

テストデータセットのラベル名とトレーニングデータセットのラベル名の重複は 50% 未満です。

ERROR\_INSUFFICIENT\_LABEL\_OVERLAP を修正するには (コンソール)

- トレーニングデータセットからラベルを削除します。あるいは、より一般的なラベルをテストデータセットに追加することもできます。詳細については、「[ラベルの管理](#)」を参照してください。ラベルはデータセット内のイメージと境界ボックスから自動的に削除されます。

トレーニングデータセットからラベルを削除して `ERROR_INSUFFICIENT_LABEL_OVERLAP` を修正するには (JSON 行)

- イメージレベルの JSON 行を含むマニフェスト - イメージにラベルが 1 つしかない場合は、目的のラベルを使用するイメージの JSON 行を削除します。JSON 行に複数のラベルが含まれている場合は、目的のラベルの JSON オブジェクトのみを削除します。詳細については、「[複数のイメージレベルのラベルをイメージに追加する](#)」を参照してください。削除するラベルを含むマニフェスト内の JSON 行ごとにこれを行います。

オブジェクトの場所が JSON 行を含むマニフェスト - 削除するラベルの境界ボックスと関連付けられたラベル情報を削除します。目的のラベルを含む JSON 行ごとにこれを実行します。class-map 配列と、それに対応する objects および annotations 配列のオブジェクトからラベルを削除する必要があります。詳細については、「[マニフェストファイル内のオブジェクトのローカリゼーション](#)」を参照してください。

テストデータセットに共通のラベルを追加して `ERROR_INSUFFICIENT_LABEL_OVERLAP` を修正するには (JSON 行)

- トレーニングデータセットに既にラベルが付けられたイメージを含む JSON 行をテストデータセットに追加します。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

## ERROR\_MANIFEST\_TOO\_FEW\_USABLE\_LABELS

### エラーメッセージ

マニフェストファイルに使用可能なラベルが少なすぎます。

### 詳細情報

トレーニングマニフェストには、イメージレベルのラベル形式とオブジェクトロケーション形式の JSON 行を含めることができます。トレーニングマニフェストにある JSON 行のタイプに応じて、Amazon Rekognition Custom Labels はイメージレベルのラベルを検出するモデルを作成するか、オブジェクトの場所を検出するモデルを作成します。Amazon Rekognition Custom Labels は、選択した形式ではない JSON 行の有効な JSON レコードを除外します。ERROR\_MANIFEST\_TOO\_FEW\_USABLE\_LABELS は、選択したモデルタイプマニフェストに含まれるラベルの数がモデルをトレーニングするのに十分でない場合に発生します。

イメージレベルのラベルを検出するモデルをトレーニングするには、少なくとも 1 つのラベルが必要です。位置をオブジェクト化するモデルをトレーニングするには、最低 2 つのラベルが必要です。

ERROR\_MANIFEST\_TOO\_FEW\_USABLE\_LABELS を修正するには (コンソール)

1. マニフェスト概要の use\_case フィールドを確認してください。
2. トレーニングデータセットに、use\_case の値と一致するユースケース (イメージレベルまたはオブジェクトのローカリゼーション) 用のラベルをさらに追加します。詳細については、「[ラベルの管理](#)」を参照してください。ラベルはデータセット内のイメージと境界ボックスから自動的に削除されます。

ERROR\_MANIFEST\_TOO\_FEW\_USABLE\_LABELS を修正するには (JSON Line)

1. マニフェスト概要の use\_case フィールドを確認してください。
2. トレーニングデータセットに、use\_case の値と一致するユースケース (イメージレベルまたはオブジェクトのローカリゼーション) 用のラベルをさらに追加します。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

## ERROR\_INSUFFICIENT\_USABLE\_LABEL\_OVERLAP

### エラーメッセージ

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間で重複しているラベルが {}% 未満です。

### 詳細情報

トレーニングマニフェストには、イメージレベルのラベル形式とオブジェクトローケーション形式の JSON 行を含めることができます。トレーニングマニフェストにある形式に応じて、Amazon Rekognition Custom Labels はイメージレベルのラベルを検出するモデルを作成するか、オブジェクトの場所を検出するモデルを作成します。Amazon Rekognition Custom Labels は、選択したモデル形式ではない JSON 行には有効な JSON レコードを使用しません。ERROR\_INSUFFICIENT\_USABLE\_LABEL\_OVERLAP は、使用されているテストラベルとトレーニングラベルの重複が 50% 未満の場合に発生します。



## ERROR\_INSUFFICIENT\_USABLE\_LABEL\_OVERLAP を修正するには (コンソール)

- トレーニングデータセットからラベルを削除します。あるいは、より一般的なラベルをテストデータセットに追加することもできます。詳細については、「[ラベルの管理](#)」を参照してください。ラベルはデータセット内のイメージと境界ボックスから自動的に削除されます。

### トレーニングデータセットからラベルを削除して

## ERROR\_INSUFFICIENT\_USABLE\_LABEL\_OVERLAP を修正するには (JSON 行)

- イメージレベルのラベルの検出に使用されるデータセット - イメージにラベルが 1 つしかない場合は、目的のラベルを使用するイメージの JSON 行を削除します。JSON 行に複数のラベルが含まれている場合は、目的のラベルの JSON オブジェクトのみを削除します。詳細については、「[複数のイメージレベルのラベルをイメージに追加する](#)」を参照してください。削除するラベルを含むマニフェスト内の JSON 行ごとにこれを行います。

オブジェクトの場所の検出に使用されるデータセット - 削除するラベルの境界ボックスと関連するラベル情報を削除します。目的のラベルを含む JSON 行ごとにこれを実行します。class-map 配列と、それに対応する objects および annotations 配列のオブジェクトからラベルを削除する必要があります。詳細については、「[マニフェストファイル内のオブジェクトのローカライゼーション](#)」を参照してください。

### テストデータセットに共通のラベルを追加して

## ERROR\_INSUFFICIENT\_USABLE\_LABEL\_OVERLAP を修正するには (JSON 行)

- トレーニングデータセットに既にラベルが付けられたイメージを含む JSON 行をテストデータセットに追加します。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

## ERROR\_FAILED\_IMAGES\_S3\_COPY

### エラーメッセージ

S3 バケットからイメージをコピーできませんでした。

### 詳細情報

サービスはデータセット内のどのイメージもコピーできませんでした。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

ERROR\_FAILED\_IMAGES\_S3\_COPY を修正するには

1. イメージのアクセス許可を確認してください。
2. AWS KMS を使用している場合は、バケットポリシーを確認してください。詳細については、「[で暗号化されたファイルの復号化 AWS Key Management Service](#)」を参照してください。

マニフェストファイルにターミナルエラーが多すぎます。

ターミナルコンテンツエラーの JSON 行が多すぎます。

ERROR\_TOO\_MANY\_RECORDS\_IN\_ERROR を修正するには

- ターミナルコンテンツエラーがある JSON 行 (イメージ) の数を減らしてください。詳細については、「[ターミナルマニフェストコンテンツエラー](#)」を参照してください。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

## 非ターミナル JSON 行検証エラー

このトピックでは、トレーニング中に Amazon Rekognition Custom Labels によって報告された非ターミナル JSON 回線検証エラーを一覧表示します。このエラーはトレーニングとテストの検証マニフェストで報告されます。詳細については、「[トレーニングとテストの検証結果マニフェストを理解する](#)」を参照してください。非ターミナル JSON 行エラーは、トレーニングマニフェストファイルまたはテストマニフェストファイルの JSON 行を更新することによって修正できます。JSON 行をマニフェストから削除することもできますが、そうするとモデルの品質が低下する場合があります。非ターミナル検証エラーが多い場合は、マニフェストファイルを再作成する方が簡単かもしれません。検証エラーは通常、手動で作成したマニフェストファイルで発生します。詳細については、「[マニフェストファイルの作成](#)」を参照してください。検証エラーを修正する方法については、「[トレーニングエラーの修正](#)」を参照してください。一部のエラーは、Amazon Rekognition Custom Labels コンソールを使用して修正できます。

## ERROR\_MISSING\_SOURCE\_REF

### エラーメッセージ

ソース参照キーがありません。

### 詳細情報

JSON 行 source-ref フィールドには、イメージの Amazon S3 の場所が表示されます。このエラーは、source-ref キーがないか、スペルが間違っているときに発生します。このエラーは通常、手動で作成したマニフェストファイルで発生します。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

### ERROR\_MISSING\_SOURCE\_REF を修正するには

1. source-ref キーが存在し、スペルが正しいことを確認してください。完全な source-ref キーと値は次のようなものです。"source-ref": "s3://bucket/path/image"
2. JSON 行の source-ref キーを更新します。または、マニフェストファイルから JSON 行を削除してください。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

## ERROR\_INVALID\_SOURCE\_REF\_FORMAT

### エラーメッセージ

ソース参照値の形式が無効です。

### 詳細情報

source-ref キーは JSON 行にあります。Amazon S3 パスのスキーマが正しくありません。例えば、パスは S3://... ではなく https://... です。ERROR\_INVALID\_SOURCE\_REF\_FORMAT エラーは通常、手動で作成されたマニフェストファイルで発生します。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

### ERROR\_INVALID\_SOURCE\_REF\_FORMAT を修正するには

1. スキーマが "source-ref": "s3://bucket/path/image" であることを確認してください。例えば、"source-ref": "s3://custom-labels-console-us-east-1-1111111111/images/000000242287.jpg" です。

## 2. マニフェストファイルの JSON 行を更新または削除します。

Amazon Rekognition Custom Labels コンソールを使用して、この `ERROR_INVALID_SOURCE_REF_FORMAT` を解決することはできません。

### 「ラベル属性がない」というエラー

#### エラーメッセージ

ラベル属性が見つかりません。

#### 詳細情報

ラベル属性またはラベル属性の `-metadata` キー名 (あるいはその両方) が無効または存在しません。次の例では、`ERROR_NO_LABEL_ATTRIBUTES` は、`bounding-box` または `bounding-box-metadata` キー (または両方) が欠落している場合に発生します。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

```
{
  "source-ref": "s3://custom-labels-bucket/images/IMG_1186.png",
  "bounding-box": {
    "image_size": [{
      "width": 640,
      "height": 480,
      "depth": 3
    }],
    "annotations": [{
      "class_id": 1,
      "top": 251,
      "left": 399,
      "width": 155,
      "height": 101
    }, {
      "class_id": 0,
      "top": 65,
      "left": 86,
      "width": 220,
      "height": 334
    }
  ],
  "bounding-box-metadata": {
    "objects": [{
```

```
  "confidence": 1
}, {
  "confidence": 1
}],
"class-map": {
  "0": "Echo",
  "1": "Echo Dot"
},
"type": "groundtruth/object-detection",
"human-annotated": "yes",
"creation-date": "2018-10-18T22:18:13.527256",
"job-name": "my job"
}
}
```

ERROR\_NO\_LABEL\_ATTRIBUTES エラーは通常、手動で作成したマニフェストファイルで発生します。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

**ERROR\_NO\_LABEL\_ATTRIBUTES** を修正するには

1. ラベル属性識別子とラベル属性識別子の `-metadata` キーが存在し、キー名のスペルが正しいことを確認してください。
2. マニフェストファイルの JSON 行を更新または削除します。

Amazon Rekognition Custom Labels コンソールを使用して、ERROR\_NO\_LABEL\_ATTRIBUTES を解決することはできません。

エラー:ラベル属性形式が無効です

エラーメッセージ

ラベル属性 `{}` の形式が無効です。

詳細情報

ラベル属性キーのスキーマがないか、無効です。ERROR\_INVALID\_LABEL\_ATTRIBUTE\_FORMAT エラーは通常、手動で作成されたマニフェストファイルで発生します。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

## ERROR\_INVALID\_LABEL\_ATTRIBUTE\_FORMAT を修正するには

1. ラベル属性キーの JSON 行セクションが正しいことを確認してください。次のオブジェクト位置の例では、image\_size および annotations オブジェクトが正しい必要があります。ラベル属性キーには bounding-box という名前が付いています。

```
"bounding-box": {
  "image_size": [{
    "width": 640,
    "height": 480,
    "depth": 3
  }],
  "annotations": [{
    "class_id": 1,
    "top": 251,
    "left": 399,
    "width": 155,
    "height": 101
  }, {
    "class_id": 0,
    "top": 65,
    "left": 86,
    "width": 220,
    "height": 334
  }]
},
```

2. マニフェストファイルの JSON 行を更新または削除します。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

## ERROR\_INVALID\_LABEL\_ATTRIBUTE\_METADATA\_FORMAT

### エラーメッセージ

ラベル属性メタデータの形式が無効です。

### 詳細情報

ラベル属性メタデータキーのスキーマがないか、または無効です。ERROR\_INVALID\_LABEL\_ATTRIBUTE\_METADATA\_FORMAT エラーは通常、手動で作成され

たマニフェストファイルで発生します。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

### ERROR\_INVALID\_LABEL\_ATTRIBUTE\_FORMAT を修正するには

1. ラベル属性メタデータキーの JSON 行スキーマが次の例のようになっていることを確認します。ラベル属性メタデータキーには bounding-box-metadata という名前が付いています。

```
"bounding-box-metadata": {
  "objects": [{
    "confidence": 1
  }, {
    "confidence": 1
  }],
  "class-map": {
    "0": "Echo",
    "1": "Echo Dot"
  },
  "type": "groundtruth/object-detection",
  "human-annotated": "yes",
  "creation-date": "2018-10-18T22:18:13.527256",
  "job-name": "my job"
}
```

2. マニフェストファイルの JSON 行を更新または削除します。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

### ERROR\_NO\_VALID\_LABEL\_ATTRIBUTES

#### エラーメッセージ

有効なラベル属性が見つかりません。

#### 詳細情報

JSON 行に有効なラベル属性が見つかりませんでした。Amazon Rekognition Custom Labels は、ラベル属性とラベル属性識別子の両方をチェックします。ERROR\_INVALID\_LABEL\_ATTRIBUTE\_FORMAT エラーは通常、手動で作成されたマニフェストファイルで発生します。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

JSON 行がサポートされている SageMaker マニフェスト形式でない場合、Amazon Rekognition Custom Labels は JSON 行を無効としてマークし、`ERROR_NO_VALID_LABEL_ATTRIBUTES` エラーが報告されます。現在、Amazon Rekognition Custom Labels は分類ジョブと境界ボックスの形式をサポートしています。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

`ERROR_NO_VALID_LABEL_ATTRIBUTES` を修正するには

1. ラベル属性キーとラベル属性メタデータの JSON が正しいことを確認してください。
2. マニフェストファイルの JSON 行を更新または削除します。詳細については、「[the section called “マニフェストファイルの作成”](#)」を参照してください。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

## `ERROR_MISSING_BOUNDING_BOX_CONFIDENCE`

エラーメッセージ

1 つ以上の境界ボックスの信頼値がありません。

詳細情報

1 つまたは複数のオブジェクト位置境界ボックスの信頼キーがありません。次の例に示すように、境界ボックスの信頼キーはラベル属性メタデータにあります。ERROR\_MISSING\_BOUNDING\_BOX\_CONFIDENCE エラーは通常、手動で作成されたマニフェストファイルで発生します。詳細については、「[the section called “マニフェストファイル内のオブジェクトのローカリゼーション”](#)」を参照してください。

```
"bounding-box-metadata": {
  "objects": [{
    "confidence": 1
  }, {
    "confidence": 1
  }],
}
```

`ERROR_MISSING_BOUNDING_BOX_CONFIDENCE` を修正するには

1. ラベル属性の `objects` 配列に、ラベル属性 `annotations` 配列のオブジェクトと同じ数の信頼キーが含まれていることを確認してください。



## 2. マニフェストファイルの JSON 行を更新または削除します。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

### ERROR\_MISSING\_CLASS\_MAP\_ID

#### エラーメッセージ

1 つ以上のクラス ids がクラスマップにありません。

#### 詳細情報

注釈 (境界ボックス) `class_id` 内のオブジェクトに、ラベル属性メタデータのクラスマップ (`class-map`) に一致するエントリがありません。詳細については、「[マニフェストファイル内のオブジェクトのローカリゼーション](#)」を参照してください。ERROR\_MISSING\_CLASS\_MAP\_ID エラーは通常、手動で作成されたマニフェストファイルで発生します。

ERROR\_MISSING\_CLASS\_MAP\_ID を修正するには

1. 次の例のように、各注釈 (境界ボックス) オブジェクトの `class_id` 値が `class-map` 配列内で対応する値を持っていることを確認します。 `annotations` 配列と `class_map` 配列の要素数は同じでなければなりません。

```
{
  "source-ref": "s3://custom-labels-bucket/images/IMG_1186.png",
  "bounding-box": {
    "image_size": [{
      "width": 640,
      "height": 480,
      "depth": 3
    }],
    "annotations": [{
      "class_id": 1,
      "top": 251,
      "left": 399,
      "width": 155,
      "height": 101
    }, {
      "class_id": 0,
      "top": 65,
```

```
    "left": 86,
    "width": 220,
    "height": 334
  ]],
},
"bounding-box-metadata": {
  "objects": [{
    "confidence": 1
  }, {
    "confidence": 1
  }],
  "class-map": {
    "0": "Echo",
    "1": "Echo Dot"
  },
  "type": "groundtruth/object-detection",
  "human-annotated": "yes",
  "creation-date": "2018-10-18T22:18:13.527256",
  "job-name": "my job"
}
}
```

2. マニフェストファイルの JSON 行を更新または削除します。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

## ERROR\_INVALID\_JSON\_LINE

### エラーメッセージ

JSON 行の形式が無効です。

### 詳細情報

JSON 行に予期しない文字が見つかりました。JSON 行は、エラー情報のみを含む新しい JSON 行に置き換えられます。ERROR\_INVALID\_JSON\_LINE エラーは通常、手動で作成されたマニフェストファイルで発生します。詳細については、「[the section called “マニフェストファイル内のオブジェクトのローカリゼーション”](#)」を参照してください。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

## ERROR\_INVALID\_JSON\_LINE を修正するには

1. マニフェストファイルを開き、ERROR\_INVALID\_JSON\_LINE エラーが発生した JSON 行に移動します。
2. JSON 行に無効な文字が含まれていないことと、必須の ; または , 文字が含まれていることを確認してください。
3. マニフェストファイルの JSON 行を更新または削除します。

## ERROR\_INVALID\_IMAGE

### エラーメッセージ

イメージが無効です。S3 パスやイメージのプロパティを確認してください。

### 詳細情報

source-ref によって参照されるファイルが、有効なイメージではありません。考えられる原因には、イメージのアスペクト比、イメージのサイズ、イメージ形式などがあります。

詳細については、「[ガイドラインとクォータ](#)」を参照してください。

## ERROR\_INVALID\_IMAGE を修正するには

1. 以下をチェックしてください。
  - イメージのアスペクト比が 20:1 未満である。
  - イメージのサイズが 15 MB を超えている。
  - イメージは、PNG または JPEG 形式である。
  - source-ref のイメージへのパスが正しい。
  - イメージの最小イメージ寸法が 64 ピクセル x 64 ピクセルよりも大きい。
  - イメージの最大イメージ寸法が 4096 ピクセル x 4096 ピクセルよりも小さい。
2. マニフェストファイルの JSON 行を更新または削除します。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

## ERROR\_INVALID\_IMAGE\_DIMENSION

### エラーメッセージ

イメージの寸法が許容される寸法に適合していません。

### 詳細情報

source-ref によって参照されるイメージが、許容されるイメージ寸法に適合していません。最小寸法は 64 ピクセルです。最大寸法は 4096 ピクセルです。境界ボックスのあるイメージについて ERROR\_INVALID\_IMAGE\_DIMENSION が報告されます。

詳細については、「[ガイドラインとクォータ](#)」を参照してください。

### ERROR\_INVALID\_IMAGE\_DIMENSION (コンソール) を修正するには

1. Amazon S3 バケット内のイメージを、Amazon Rekognition Custom Labels が処理できる寸法に更新します。
2. Amazon Rekognition Custom Labels コンソールで、次の操作を行います。
  - a. 既存の境界ボックスをイメージから削除します。
  - b. 境界ボックスをイメージに再度追加します。
  - c. 変更を保存します。

詳細については、「[境界ボックスによるオブジェクトのラベル付け](#)」。

### ERROR\_INVALID\_IMAGE\_DIMENSION (SDK) を修正するには

1. Amazon S3 バケット内のイメージを、Amazon Rekognition Custom Labels が処理できる寸法に更新します。
2. ListDataSetEntries を呼び出すことにより、イメージ用の既存の JSON 行を取得します。SourceRefContains 入力パラメータには、Amazon S3 の場所とイメージのファイル名を指定します。
3. [UpdateDataSetEntries](#) を呼び出して、イメージの JSON 行を指定します。source-ref の値が Amazon S3 バケットのイメージの場所と一致することを確認します。更新されたイメージで必要な境界ボックス寸法に合わせて、境界ボックスの注釈を更新します。

```
{  
  "source-ref": "s3://custom-labels-bucket/images/IMG_1186.png",
```

```
"bounding-box": {
  "image_size": [{
    "width": 640,
    "height": 480,
    "depth": 3
  }],
  "annotations": [{
    "class_id": 1,
    "top": 251,
    "left": 399,
    "width": 155,
    "height": 101
  }, {
    "class_id": 0,
    "top": 65,
    "left": 86,
    "width": 220,
    "height": 334
  }]
},
"bounding-box-metadata": {
  "objects": [{
    "confidence": 1
  }, {
    "confidence": 1
  }],
  "class-map": {
    "0": "Echo",
    "1": "Echo Dot"
  },
  "type": "groundtruth/object-detection",
  "human-annotated": "yes",
  "creation-date": "2013-11-18T02:53:27",
  "job-name": "my job"
}
}
```

## ERROR\_INVALID\_BOUNDING\_BOX

### エラーメッセージ

境界ボックスにオフフレーム値があります。

## 詳細情報

境界ボックス情報は、イメージフレームから外れているか負の値を含むイメージを指定します。

詳細については、「[ガイドラインとクォータ](#)」を参照してください。

### **ERROR\_INVALID\_BOUNDING\_BOX** を修正するには

1. annotations 配列内の境界ボックスの値を確認してください。

```
"bounding-box": {
  "image_size": [{
    "width": 640,
    "height": 480,
    "depth": 3
  }],
  "annotations": [{
    "class_id": 1,
    "top": 251,
    "left": 399,
    "width": 155,
    "height": 101
  }]
},
```

2. マニフェストファイルの JSON 行を更新するか、マニフェストファイルから削除します。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

## ERROR\_NO\_VALID\_ANNOTATIONS

### エラーメッセージ

有効な注釈が見つかりません。

### 詳細情報

JSON 行のどの注釈オブジェクトにも有効な境界ボックス情報が含まれていません。

## ERROR\_NO\_VALID\_ANNOTATIONS を修正するには

1. 有効な境界ボックスオブジェクトを含むよう annotations 配列を更新してください。また、ラベル属性メタデータ内の対応する境界ボックス情報 (confidence と class\_map) が正しいことを確認してください。詳細については、「[マニフェストファイル内のオブジェクトのローカライゼーション](#)」を参照してください。

```
{
  "source-ref": "s3://custom-labels-bucket/images/IMG_1186.png",
  "bounding-box": {
    "image_size": [{
      "width": 640,
      "height": 480,
      "depth": 3
    }],
    "annotations": [
      {
        "class_id": 1,      #annotation object
        "top": 251,
        "left": 399,
        "width": 155,
        "height": 101
      }, {
        "class_id": 0,
        "top": 65,
        "left": 86,
        "width": 220,
        "height": 334
      }
    ]
  },
  "bounding-box-metadata": {
    "objects": [
      >{
        "confidence": 1      #confidence object
      },
      {
        "confidence": 1
      }
    ],
    "class-map": {
      "0": "Echo",      #label
      "1": "Echo Dot"
    }
  },
  "type": "groundtruth/object-detection",
}
```

```
"human-annotated": "yes",
"creation-date": "2018-10-18T22:18:13.527256",
"job-name": "my job"
}
}
```

2. マニフェストファイルの JSON 行を更新するか、マニフェストファイルから削除します。

Amazon Rekognition Custom Labels コンソールを使用してこのエラーを修正することはできません。

## ERROR\_BOUNDING\_BOX\_TOO\_SMALL

### エラーメッセージ

境界ボックスの高さと幅が小さすぎます。

### 詳細情報

境界ボックスの寸法 (高さ と 幅) は 1 x 1 ピクセルよりも大きくなければなりません。

イメージの寸法のどちらかが 1280 ピクセルを超える場合、Amazon Rekognition Custom Labels はトレーニング中にイメージのサイズを変更します (ソースイメージには影響しません)。変更後の境界ボックスの高さと幅は 1 x 1 ピクセルよりも大きくなければなりません。境界ボックスの位置は、オブジェクトロケーション JSON 行の annotations 配列に保存されます。詳細については、「[マニフェストファイル内のオブジェクトのローカリゼーション](#)」を参照してください。

```
"bounding-box": {
  "image_size": [{
    "width": 640,
    "height": 480,
    "depth": 3
  }],
  "annotations": [{
    "class_id": 1,
    "top": 251,
    "left": 399,
    "width": 155,
    "height": 101
  }]
},
```



エラー情報は注釈オブジェクトに追加されます。

`ERROR_BOUNDING_BOX_TOO_SMALL` を修正するには

- 次のいずれかのオプションを選択します。
  - 小さすぎる境界ボックスのサイズを大きくします。
  - 小さすぎる境界ボックスを削除します。境界ボックスの削除については、「[ERROR\\_TOO\\_MANY\\_BOUNDING\\_BOXES](#)」を参照してください。
  - マニフェストからイメージ (JSON 行) を削除します。

## ERROR\_TOO\_MANY\_BOUNDING\_BOXES

### エラーメッセージ

境界ボックスが許容最大数を超過しています。

### 詳細情報

境界ボックスが許容最大数 (50) を超過しています。Amazon Rekognition Custom Labels コンソールで余分な境界ボックスを削除することも、JSON 行から削除することもできます。

**ERROR\_TOO\_MANY\_BOUNDING\_BOXES** (コンソール) を修正するには。

1. 削除する境界ボックスを決めます。
2. Amazon Rekognition コンソールを <https://console.aws.amazon.com/rekognition/> で開きます。
3. [カスタムラベルを使用] を選択します。
4. [開始する] を選択します。
5. 左側のナビゲーションペインで、使用するデータセットを含むプロジェクトを選択します。
6. [データセット] ページで、使用するデータセットを選択します。
7. データセットギャラリーページで、[ラベル付けを開始] を選択してラベル付けモードに入ります。
8. 境界ボックスを削除する元のイメージを選択します。
9. [境界ボックスを描画] を選択します。
10. 描画ツールで、削除する境界ボックスを選択します。
11. キーボードの Delete キーを押して、境界ボックスを削除します。

12. 必要な数の境界ボックスが削除されるまで、前の 2 つの手順を繰り返します。
13. [完了] を選択します。
14. [変更の保存] を選択して、変更を保存します。
15. [終了] を選択してラベリングモードを終了します。

ERROR\_TOO\_MANY\_BOUNDING\_BOXES (JSON 行) を修正するには。

1. マニフェストファイルを開き、ERROR\_TOO\_MANY\_BOUNDING\_BOXES エラーが発生した JSON 行に移動します。
2. 削除する境界ボックスごとに、次のものを削除します。
  - 必要な annotation オブジェクトを annotations 配列から削除します。
  - ラベル属性メタデータの objects 配列から対応する confidence オブジェクトを削除します。
  - ラベルが他の境界ボックスで使用されなくなった場合は、class-map からそのラベルを削除します。

次の例を参考にして、削除する項目を特定してください。

```
{
  "source-ref": "s3://custom-labels-bucket/images/IMG_1186.png",
  "bounding-box": {
    "image_size": [{
      "width": 640,
      "height": 480,
      "depth": 3
    }],
    "annotations": [
      {
        "class_id": 1,      #annotation object
        "top": 251,
        "left": 399,
        "width": 155,
        "height": 101
      }, {
        "class_id": 0,
        "top": 65,
        "left": 86,
```

```
    "width": 220,
    "height": 334
  ]],
},
"bounding-box-metadata": {
  "objects": [
    >{
      "confidence": 1          #confidence object
    },
    {
      "confidence": 1
    }
  ]],
"class-map": {
  "0": "Echo",    #label
  "1": "Echo Dot"
},
"type": "groundtruth/object-detection",
"human-annotated": "yes",
"creation-date": "2018-10-18T22:18:13.527256",
"job-name": "my job"
}
}
```

## WARNING\_UNANNOTATED\_RECORD

### 警告メッセージ

レコードに注釈が付いていません。

### 詳細情報

Amazon Rekognition Custom Labels コンソールを使用してデータセットに追加されたイメージにラベルが付けられていませんでした。そのイメージの JSON 行はトレーニングには使用されません。

```
{
  "source-ref": "s3://bucket/images/IMG_1186.png",
  "warnings": [
    {
      "code": "WARNING_UNANNOTATED_RECORD",
      "message": "Record is unannotated."
    }
  ]
}
```

```
]
}
```

WARNING\_UNANNOTATED\_RECORD を修正するには

- Amazon Rekognition Custom Labels コンソールを使用してイメージにラベルを付けます。手順については、[「イメージにイメージレベルのラベルを割り当てる」](#)を参照してください。

## WARNING\_NO\_ANNOTATIONS

### 警告メッセージ

注釈が提供されていません。

### 詳細情報

人間 (human-annotated = yes) によって付けられた注釈があるにもかかわらず、オブジェクトローカリゼーション形式の JSON 行には境界ボックス情報が含まれません。JSON 行は有効ですが、トレーニングには使用されません。詳細については、[「トレーニングとテストの検証結果マニフェストを理解する」](#)を参照してください。

```
{
  "source-ref": "s3://bucket/images/IMG_1186.png",
  "bounding-box": {
    "image_size": [
      {
        "width": 640,
        "height": 480,
        "depth": 3
      }
    ],
    "annotations": [
    ],
    "warnings": [
      {
        "code": "WARNING_NO_ATTRIBUTE_ANNOTATIONS",
        "message": "No attribute annotations were found."
      }
    ]
  }
}
```

```
    ],
  },
  "bounding-box-metadata": {
    "objects": [

    ],
    "class-map": {

    },
    "type": "groundtruth/object-detection",
    "human-annotated": "yes",
    "creation-date": "2013-11-18 02:53:27",
    "job-name": "my job"
  },
  "warnings": [
    {
      "code": "WARNING_NO_ANNOTATIONS",
      "message": "No annotations were found."
    }
  ]
}
```

WARNING\_NO\_ANNOTATIONS を修正するには

- 次のいずれかのオプションを選択します。
  - JSON 行に境界ボックス (annotations) 情報を追加します。詳細については、「[マニフェストファイル内のオブジェクトのローカリゼーション](#)」を参照してください。
  - マニフェストからイメージ (JSON 行) を削除します。

## WARNING\_NO\_ATTRIBUTE\_ANNOTATIONS

### 警告メッセージ

属性の注釈が提供されていません。

### 詳細情報

人間 (human-annotated = yes) によって付けられた注釈があるにもかかわらず、オブジェクトローカリゼーション形式の JSON 行には境界ボックス注釈情報が含まれません。annotations 配

列が存在しないか、入力されていません。JSON 行は有効ですが、トレーニングには使用されません。詳細については、「[トレーニングとテストの検証結果マニフェストを理解する](#)」を参照してください。

```
{
  "source-ref": "s3://bucket/images/IMG_1186.png",
  "bounding-box": {
    "image_size": [
      {
        "width": 640,
        "height": 480,
        "depth": 3
      }
    ],
    "annotations": [
    ],
    "warnings": [
      {
        "code": "WARNING_NO_ATTRIBUTE_ANNOTATIONS",
        "message": "No attribute annotations were found."
      }
    ]
  },
  "bounding-box-metadata": {
    "objects": [
    ],
    "class-map": {
    },
    "type": "groundtruth/object-detection",
    "human-annotated": "yes",
    "creation-date": "2013-11-18 02:53:27",
    "job-name": "my job"
  },
  "warnings": [
    {
      "code": "WARNING_NO_ANNOTATIONS",
      "message": "No annotations were found."
    }
  ]
}
```

## WARNING\_NO\_ATTRIBUTE\_ANNOTATIONS を修正するには

- 次のいずれかのオプションを選択します。
  - JSON 行に 1 つ以上の境界ボックス annotation オブジェクトを追加します。詳細については、「[マニフェストファイル内のオブジェクトのローカリゼーション](#)」を参照してください。
  - 境界ボックス属性を削除します。
  - マニフェストからイメージ (JSON 行) を削除します。JSON 行に他の有効な境界ボックス属性がある場合は、代わりに JSON 行から無効な境界ボックス属性だけを削除できます。

## ERROR\_UNSUPPORTED\_USE\_CASE\_TYPE

### 警告メッセージ

### 詳細情報

type フィールドの値が groundtruth/image-classification または groundtruth/object-detection ではありません。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

```
{
  "source-ref": "s3://bucket/test_normal_8.jpg",
  "BB": {
    "annotations": [
      {
        "left": 1768,
        "top": 1007,
        "width": 448,
        "height": 295,
        "class_id": 0
      },
      {
        "left": 1794,
        "top": 1306,
        "width": 432,
        "height": 411,
        "class_id": 1
      },
      {
        "left": 2568,
        "top": 1346,
```

```
        "width": 710,
        "height": 305,
        "class_id": 2
    },
    {
        "left": 2571,
        "top": 1020,
        "width": 644,
        "height": 312,
        "class_id": 3
    }
],
"image_size": [
    {
        "width": 4000,
        "height": 2667,
        "depth": 3
    }
]
},
"BB-metadata": {
    "job-name": "labeling-job/BB",
    "class-map": {
        "0": "comparator",
        "1": "pot_resistor",
        "2": "ir_phototransistor",
        "3": "ir_led"
    },
    "human-annotated": "yes",
    "objects": [
        {
            "confidence": 1
        },
        {
            "confidence": 1
        },
        {
            "confidence": 1
        },
        {
            "confidence": 1
        }
    ],
    "creation-date": "2021-06-22T09:58:34.811Z",
```



```
    "type": "groundtruth/wrongtype",
    "cl-errors": [
      {
        "code": "ERROR_UNSUPPORTED_USE_CASE_TYPE",
        "message": "The use case type of the BB-metadata label attribute
metadata is unsupported. Check the type field."
      }
    ],
    "cl-metadata": {
      "is_labeled": true
    },
    "cl-errors": [
      {
        "code": "ERROR_NO_VALID_LABEL_ATTRIBUTES",
        "message": "No valid label attributes found."
      }
    ]
  }
}
```

ERROR\_UNSUPPORTED\_USE\_CASE\_TYPE を修正するには

- 以下のオプションのいずれかを選択します。
  - 作成するモデルのタイプに応じて、type フィールドの値を groundtruth/image-classification または groundtruth/object-detection に変更します。詳細については、「[マニフェストファイルの作成](#)」を参照してください。
  - マニフェストからイメージ (JSON 行) を削除します。

## ERROR\_INVALID\_LABEL\_NAME\_LENGTH

### 詳細情報

ラベル名が長すぎます。最大長は 256 文字です。

ERROR\_INVALID\_LABEL\_NAME\_LENGTH を修正するには

- 以下のオプションのいずれかを選択します。
  - ラベル名の長さを 256 文字以下に減らします。
  - マニフェストからイメージ (JSON 行) を削除します。

# トレーニング済み Amazon Rekognition Custom Labels の改善

トレーニングが完了したら、モデルのパフォーマンスを評価します。Amazon Rekognition Custom Labels には、各ラベルには便利なサマリーメトリクスと評価メトリクスが用意されています。使用可能なメトリクスの詳細については、「[モデルを評価するためのメトリクス](#)」を参照してください。メトリクスを使用してモデルを改善する方法については、「[Amazon Rekognition Custom Labels モデルの改善](#)」を参照してください。

モデルの正確性に満足している場合は、使用を開始できます。詳細については、「[トレーニング済み Amazon Rekognition Custom Labels の実行](#)」を参照してください。

## トピック

- [モデルを評価するためのメトリクス](#)
- [評価メトリクスへのアクセス \(コンソール\)](#)
- [Amazon Rekognition Custom Labels の評価メトリクス \(SDK\) へのアクセス](#)
- [Amazon Rekognition Custom Labels モデルの改善](#)

## モデルを評価するためのメトリクス

モデルをトレーニングすると、Amazon Rekognition Custom Labels はモデルテストのメトリクスを返し、モデルのパフォーマンスを評価するために使用できます。このトピックでは、利用可能なメトリクスと、トレーニング済みのモデルが適切に機能しているかどうかを確認する方法について説明します。

Amazon Rekognition Custom Labels コンソールには、トレーニング結果の概要として、また各ラベルのメトリクスとして、以下のメトリクスが用意されています。

- [適合率](#)
- [リコール](#)
- [F1](#)

各メトリクスは、機械学習モデルのパフォーマンスを評価するために一般的に使用されるメトリクスです。Amazon Rekognition Custom Labels は、テストデータセット全体におけるテスト結果のメト

リクスと、各カスタムラベルのメトリクスを返します。また、テストデータセット内の各イメージについて、トレーニング済みのカスタムモデルのパフォーマンスを確認することもできます。詳細については、「[評価メトリクスへのアクセス \(コンソール\)](#)」を参照してください。

## モデルパフォーマンスの評価

テスト中、Amazon Rekognition Custom Labels はテストイメージにカスタムラベルが含まれているかどうかを予測します。信頼スコアは、モデルの予測の確実性を定量化する値です。

カスタムラベルの信頼スコアがしきい値を超えると、モデル出力にこのラベルが含まれます。予測は以下の方法で分類できます。

- **真陽性** - Amazon Rekognition Custom Labels モデルは、テストイメージ内のカスタムラベルの存在を正しく予測します。予測されたラベルは、そのイメージの「グラウンドトゥルース」ラベルでもあります。例えば、Amazon Rekognition Custom Labels は、イメージにサッカーボールが含まれている場合、サッカーボールラベルを正しく返します。
- **偽陽性** - Amazon Rekognition Custom Labels モデルは、テストイメージ内のカスタムラベルの存在を正しく予測します。予測されたラベルは、そのイメージの「グラウンドトゥルース」ラベルではありません。例えば、Amazon Rekognition Custom Labels はサッカーボールラベルを返しますが、そのイメージのグラウンドトゥルースにはサッカーボールラベルがありません。
- **偽陰性** - Amazon Rekognition Custom Labels モデルは、イメージにカスタムラベルが存在することを予測しませんが、そのイメージの「グラウンドトゥルース」にはこのラベルが含まれています。例えば、Amazon Rekognition Custom Labels では、サッカーボールを含むイメージの「サッカーボール」カスタムラベルは返されません。
- **真陰性** - Amazon Rekognition Custom Labels モデルは、テストイメージ内にカスタムラベルがないことを正しく予測します。例えば、Amazon Rekognition Custom Labels では、サッカーボールを含まないイメージの「サッカーボール」ラベルは返されません。

コンソールでは、テストデータセット内の各イメージの真陽性、偽陽性、偽陰性の値にアクセスできます。詳細については、「[評価メトリクスへのアクセス \(コンソール\)](#)」を参照してください。

これらの予測結果を使用して、各ラベルの以下のメトリクスと、テストセット全体の集計値が計算されます。境界ボックスレベルでのモデルによる予測にも同じ定義が当てはまりますが、すべてのメトリクスは各テストイメージの各境界ボックス (予測またはグラウンドトゥルース) で計算されるという違いがあります。

## インターセクションオーバーユニオン (IoU) とオブジェクト検出

インターセクションオーバーユニオン (IoU) は、2つのオブジェクト境界ボックスが結合された領域で重なり合う割合を測定します。範囲は 0 (最小のオーバーラップ) から 1 (完全なオーバーラップ) です。テスト中、グラウンドトゥールズ境界ボックスと予測境界ボックスの IoU が 0.5 以上であれば、予測境界ボックスは正確です。

## 想定しきい値

Amazon Rekognition Custom Labels は、各カスタムラベルの想定しきい値 (0~1) を自動的に計算します。カスタムラベルに想定しきい値を設定することはできません。各ラベルの想定しきい値は、それを超えると予測が真陽性または偽陽性としてカウントされる値です。テストデータセットに基づいて設定されます。想定しきい値は、モデルトレーニング中にテストデータセットで達成された最高の F1 スコアに基づいて計算されます。

ラベルの想定しきい値は、モデルのトレーニング結果から取得できます。詳細については、「[評価メトリクスへのアクセス \(コンソール\)](#)」を参照してください。

通常、想定しきい値は、モデルの適合率と再現率を向上させるために変更します。詳細については、「[Amazon Rekognition Custom Labels モデルの改善](#)」を参照してください。モデルの想定しきい値をラベルに設定することはできないため、DetectCustomLabels のイメージを分析し MinConfidence 入力パラメータを指定しても、同じ結果が得られます。詳細については、「[トレーニングされたモデルによるイメージの分析](#)」を参照してください。

## 適合率

Amazon Rekognition Custom Labels には、各ラベルの適合率メトリクスとテストデータセット全体の平均適合率メトリクスが用意されています。

適合率は、それぞれのラベルの推定しきい値における、すべてのモデル予測 (真陽性および偽陽性) に対する正しい予測 (真陽性) の割合です。しきい値が上がると、モデルによる予測の数が少なくなる可能性があります。ただし、一般的には、しきい値が高い場合はしきい値が低い場合よりも、偽陽性に対する真陽性の比率が高くなります。適合率で指定できる値の範囲は 0~1 で、値が大きいくほど精度は高くなります。

例えば、あるイメージにサッカーボールがあるとモデルが予測したとき、その予測が正しい頻度はどれほどでしょうか。サッカーボールが 8 個、岩が 5 個あるイメージがあるとします。モデルが 9 個のサッカーボールを予測する場合、8 個は正しく予測されて 1 個は偽陽性であり、この例の適合率は

0.89 です。ただし、モデルがイメージ内で 13 個のサッカーボールがあると予測すると、8 個の予測が正しく 5 個が不正確となり、結果として適合率は低くなります。

詳細については、「[適合率と再現率](#)」を参照してください。

## リコール

Amazon Rekognition Custom Labels には、各ラベルの平均リコールメトリクスとテストデータセット全体の平均再現率メトリクスが表示されます。

再現率は、テストセットラベルのうち、想定しきい値を上回ると正しく予測されたものの割合です。これは、テストセットのイメージに実際にカスタムラベルが存在する場合、モデルが正しく予測できる頻度を示す指標です。再現率の範囲は 0~1 です。値が大きいほど、再現率は高くなります。

例えば、イメージ内にサッカーボールが 8 個含まれている場合、正しく検出されるのは何個ですか。この例では、イメージ内にサッカーボールが 8 個と岩が 5 個あり、モデルが 5 個のサッカーボールを検出した場合、再現率値は 0.62 になります。もう一度トレーニングした後、新しいモデルがイメージ内に含まれる 8 個すべてを含む 9 個のサッカーボールを検出する場合、再現率値は 1.0 になります。

詳細については、「[適合率と再現率](#)」を参照してください。

## F1

Amazon Rekognition Custom Labels は、F1 スコアメトリクスを使用して、各ラベルの平均モデルパフォーマンスとテストデータセット全体の平均モデルパフォーマンスを測定します。

モデルパフォーマンスは、すべてのラベルの適合率と再現率の両方を考慮した総合的な指標です (F1 スコアや平均精度など)。モデルのパフォーマンススコアは 0~1 の値をとります。値が大きいほど、再現率と適合率の両方でモデルのパフォーマンスが高い事を意味します。具体的には、分類タスクのモデルパフォーマンスは通常 F1 スコアで測定されます。そのスコアでは、仮定されたしきい値での適合率スコアと再現率スコアを組み合わせた手法を用います。例えば、適合率が 0.9、再現率が 1.0 のモデルの場合、F1 スコアは 0.947 になります。

F1 スコアの値が高いということは、適合率と再現率の両面でモデルのパフォーマンスが良好であることを示しています。モデルのパフォーマンスが良好でない場合、例えば、適合率が 0.30 と低いと、再現率が 1.0 と高くても F1 スコアは 0.46 になります。同様に、適合率が高くても (0.95) 再現率が低い場合 (0.20)、F1 スコアは 0.33 になります。どちらの場合も、F1 スコアは低く、モデルに問題があることを示しています。

詳細については、「[F1 スコア](#)」を参照してください。

## メトリクスの使用

トレーニングした特定のモデルや用途によっては、DetectCustomLabels のために MinConfidence 入力パラメータを使用すると、適合率と再現率のバランスを取ることができます。MinConfidence 値を大きくすると、一般的に適合率は高くなり (サッカーボールの予測がより正確になり)、再現率は低くなります (実際のサッカーボールを見逃す回数が多くなります)。MinConfidence の値が小さいほど再現率は高くなりますが (実際のサッカーボールがより正確に予測される)、適合率は低くなります (予測の多くが誤っていることになります)。詳細については、「[トレーニングされたモデルによるイメージの分析](#)」を参照してください。

この指標は、必要に応じてモデルのパフォーマンスを改善するために取るべきステップについても教えてくれます。詳細については、「[Amazon Rekognition Custom Labels モデルの改善](#)」を参照してください。

### Note

DetectCustomLabels は 0~100 の範囲の予測を返します。これらは 0~1 のメトリクス範囲に対応します。

## 評価メトリクスへのアクセス (コンソール)

テスト中、テストデータセットに対するモデルのパフォーマンスが評価されます。テストデータセット内のラベルは、実際のイメージが表す内容を表すため、「グラウンドトゥールース」と見なされます。テスト中、モデルはテストデータセットを使用して予測を行います。予測されたラベルはグラウンドトゥールースラベルと比較され、結果がコンソールの評価ページに表示されます。

Amazon Rekognition Custom Labels コンソールには、モデル全体のサマリーメトリクスと個々のラベルのメトリクスが表示されます。コンソールで使用できるメトリクスは、適合率再現率、F1 スコア、信頼度、および信頼度しきい値です。詳細については、「[トレーニング済み Amazon Rekognition Custom Labels の改善](#)」を参照してください。

コンソールを使用して、個々のメトリクスに注目できます。例えば、ラベルの適合率に関する問題を調査するために、トレーニング結果をラベル別または偽陽性結果別にフィルターできます。詳細については、「[モデルを評価するためのメトリクス](#)」を参照してください。

トレーニング後、トレーニングデータセットは読み取り専用になります。モデルを改善する場合は、トレーニングデータセットを新しいデータセットにコピーできます。データセットのコピーを使用して、モデルの新しいバージョンをトレーニングします。

このステップでは、コンソールを使用して、コンソールのトレーニング結果にアクセスします。

### 評価メトリクスへのアクセス (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. [カスタムラベルを使用] を選択します。
3. [開始方法] を選択します。
4. 左側のナビゲーションペインで、[プロジェクト] を選択します。
5. [プロジェクト] ページで、評価するトレーニング済みモデルが含まれるプロジェクトを選択します。
6. [モデル] で、評価するモデルを選択します。
7. [評価] タブを選択すると、評価結果が表示されます。モデルの評価の詳細については、「[トレーニング済み Amazon Rekognition Custom Labels の改善](#)」を参照してください。
8. [テスト結果を表示] を選択すると、個々のテストイメージの結果が表示されます。詳細については、「[モデルを評価するためのメトリクス](#)」を参照してください。次のモデル評価の概要のスクリンショットは、テスト結果とパフォーマンスメトリクスを含む 6 つのラベルの F1 スコア、平均精度、および全体的な再現率を示しています。トレーニング済みモデルの使用に関する詳細も提供されます。



**rooms\_19** [Info](#) Delete model

**Evaluate** | Model details | Use Model | Tags

**Evaluation results** View test results

F1 score [Info](#)  
0.902

Average precision [Info](#)  
0.893

Overall recall [Info](#)  
0.928

Date completed  
July 13, 2021  
Trained in 1.223 hours

Training dataset  
10 labels, 61 images

Testing dataset  
10 labels, 56 images

**Per label performance (10)**

Find labels < 1 >

Label name ▲	F1 score ▼	Test images ▼	Precision ▼	Recall ▼	Assumed threshold ▼
backyard	0.857	4	1.000	0.750	0.286
bathroom	0.889	9	0.889	0.889	0.185
bedroom	0.900	11	1.000	0.818	0.262
closet	1.000	2	1.000	1.000	0.169
entry_way	1.000	3	1.000	1.000	0.149
floor_plan	1.000	2	1.000	1.000	0.685

9. テスト結果を確認したら、プロジェクト名を選択してモデルページに戻ります。テスト結果ページには、Backyard と front yard のイメージカテゴリでトレーニングされた機械学習モデルの予測ラベルと信頼スコアを持つイメージが表示されます。2つのサンプル画像が表示されます。



The screenshot shows the Amazon Rekognition Custom Labels console interface. At the top, the breadcrumb navigation includes 'rooms\_19' and 'rooms\_19.2021-07-13T10.36.30', which is circled in red. Below the navigation is an 'Evaluate image' section with a close button. On the left, there is a 'Filter by label' section with a search box and three checkboxes: 'True positive', 'False positive', and 'False negative'. The main area displays 'Images (56)' with a search bar and pagination. Two image cards are shown:

- backyard2.jpeg**: Shows a house with a porch. The evaluation table below the image is:

Labels	Confidence
front_yard False positive	30.3%
backyard False negative	21.6%
- backyard4.jpeg**: Shows a backyard with a lawn and trees. The evaluation table below the image is:

Labels	Confidence
backyard True positive	46.3%

10. メトリクスを使用してモデルのパフォーマンスを評価します。詳細については、「[Amazon Rekognition Custom Labels モデルの改善](#)」を参照してください。

## Amazon Rekognition Custom Labels の評価メトリクス (SDK) へのアクセス

[DescribeProjectVersions](#) オペレーションは、コンソールで提供されるメトリクス以外のメトリクスへのアクセスを提供します。

コンソールと同様に、[DescribeProjectVersions](#) では、テスト結果の概要情報、また各ラベルのテスト結果である次のメトリクスにアクセスできます。

- [適合率](#)
- [リコール](#)
- [F1](#)

すべてのラベルの平均しきい値と個々のラベルのしきい値が返されます。

DescribeProjectVersions では、分類とイメージ検出 (イメージ上のオブジェクトの位置) に関する以下のメトリクスにもアクセスできます。

- イメージ分類用の混同行列。詳細については、「[モデルの混同行列の表示](#)」を参照してください。
- イメージ検出の平均適合率の平均 (mAP)。
- イメージ検出の平均再現率の平均 (mAP)。

DescribeProjectVersions では、真陽性、偽陽性、偽陰性、真陰性の値にもアクセスできます。詳細については、「[モデルを評価するためのメトリクス](#)」を参照してください。

集約された F1 スコアメトリクスは DescribeProjectVersions によって直接返されます。他のメトリクスは、Amazon S3 バケットに格納されている [概要ファイル](#) および [評価マニフェストのスナップショット](#) ファイルからアクセスできます。詳細については、「[概要ファイルと評価マニフェストスナップショット \(SDK\) へのアクセス](#)」を参照してください。

トピック

- [概要ファイル](#)
- [評価マニフェストのスナップショット](#)
- [概要ファイルと評価マニフェストスナップショット \(SDK\) へのアクセス](#)
- [モデルの混同行列の表示](#)
- [参照: トレーニング結果概要ファイル](#)

## 概要ファイル

概要ファイルには、モデル全体に関する評価結果情報、および各ラベルのメトリクスが含まれます。メトリクスは適合率、再現率、F1 スコアです。モデルのしきい値も提供されます。概要ファイルの場所には、EvaluationResult によって返された DescribeProjectVersions オブジェクトからアクセスできます。詳細については、「[参照: トレーニング結果概要ファイル](#)」を参照してください。

次は、概要ファイルの例です。

```
{
  "Version": 1,
  "AggregatedEvaluationResults": {
    "ConfusionMatrix": [
```

```
{
  "GroundTruthLabel": "CAP",
  "PredictedLabel": "CAP",
  "Value": 0.9948717948717949
},
{
  "GroundTruthLabel": "CAP",
  "PredictedLabel": "WATCH",
  "Value": 0.008547008547008548
},
{
  "GroundTruthLabel": "WATCH",
  "PredictedLabel": "CAP",
  "Value": 0.1794871794871795
},
{
  "GroundTruthLabel": "WATCH",
  "PredictedLabel": "WATCH",
  "Value": 0.7008547008547008
}
],
"F1Score": 0.9726959470546408,
"Precision": 0.9719115848331294,
"Recall": 0.9735042735042735
},
"EvaluationDetails": {
  "EvaluationEndTimestamp": "2019-11-21T07:30:23.910943",
  "Labels": [
    "CAP",
    "WATCH"
  ],
  "NumberOfTestingImages": 624,
  "NumberOfTrainingImages": 5216,
  "ProjectVersionArn": "arn:aws:rekognition:us-east-1:nnnnnnnn:project/my-project/
version/v0/1574317227432"
},
"LabelEvaluationResults": [
  {
    "Label": "CAP",
    "Metrics": {
      "F1Score": 0.9794344473007711,
      "Precision": 0.9819587628865979,
      "Recall": 0.9769230769230769,
      "Threshold": 0.9879502058029175
    }
  }
]
```

```
    },
    "NumberOfTestingImages": 390
  },
  {
    "Label": "WATCH",
    "Metrics": {
      "F1Score": 0.9659574468085106,
      "Precision": 0.961864406779661,
      "Recall": 0.9700854700854701,
      "Threshold": 0.014450683258473873
    },
    "NumberOfTestingImages": 234
  }
]
}
```

## 評価マニフェストのスナップショット

評価マニフェストのスナップショットには、テスト結果に関する詳細情報が含まれています。スナップショットには各予測の信頼度が含まれます。また、イメージの実際の分類 (真陽性、真陰性、偽陽性、偽陰性) と比較した予測の分類も含まれます。

テストやトレーニングに使用できるイメージのみが含まれているため、ファイルはスナップショットです。間違った形式のイメージなど、検証できないイメージはマニフェストに含まれません。テスト用スナップショットの場所には、DescribeProjectVersions によって返された TestingDataResult オブジェクトからアクセスできます。トレーニング用スナップショットの場所には、DescribeProjectVersions によって返された TrainingDataResult オブジェクトからアクセスできます。

スナップショットは SageMaker Ground Truth マニフェスト出力形式で、検出の二項分類の結果などの追加情報を提供するためにフィールドが追加されています。次のスニペットは、追加フィールドを示しています。

```
"rekognition-custom-labels-evaluation-details": {
  "version": 1,
  "is-true-positive": true,
  "is-true-negative": false,
  "is-false-positive": false,
  "is-false-negative": false,
  "is-present-in-ground-truth": true
  "ground-truth-labelling-jobs": ["rekognition-custom-labels-training-job"]
}
```

```
}
```

- version - マニフェストスナップショット内の rekognition-custom-labels-evaluation-details フィールド形式のバージョン。
- is-true-positive... - 信頼度スコアとラベルの最小しきい値の比較に基づく予測の二項分類。
- is-present-in-ground-truth - モデルが実行した予測がトレーニングに使用されるグラウンドトゥールース情報に含まれている場合は true、そうでない場合は false。この値は、モデルが計算した最小しきい値を信頼度スコアが超えているかどうかに基づいていません。
- ground-truth-labeling-jobs - テストに使用するマニフェストライン内のグラウンドトゥールースフィールドのリスト。

SageMaker Ground Truth マニフェスト形式の詳細については、[「出力」](#)を参照してください。

以下は、イメージ分類とオブジェクト検出のメトリクスを示すテスト用マニフェストスナップショットの例です。

```
// For image classification
{
  "source-ref": "s3://test-bucket/dataset/beckham.jpeg",
  "rekognition-custom-labels-training-0": 1,
  "rekognition-custom-labels-training-0-metadata": {
    "confidence": 1.0,
    "job-name": "rekognition-custom-labels-training-job",
    "class-name": "Football",
    "human-annotated": "yes",
    "creation-date": "2019-09-06T00:07:25.488243",
    "type": "groundtruth/image-classification"
  },
  "rekognition-custom-labels-evaluation-0": 1,
  "rekognition-custom-labels-evaluation-0-metadata": {
    "confidence": 0.95,
    "job-name": "rekognition-custom-labels-evaluation-job",
    "class-name": "Football",
    "human-annotated": "no",
    "creation-date": "2019-09-06T00:07:25.488243",
    "type": "groundtruth/image-classification",
    "rekognition-custom-labels-evaluation-details": {
      "version": 1,
      "ground-truth-labelling-jobs": ["rekognition-custom-labels-training-job"],
      "is-true-positive": true,

```

```
    "is-true-negative": false,
    "is-false-positive": false,
    "is-false-negative": false,
    "is-present-in-ground-truth": true
  }
}
}

// For object detection
{
  "source-ref": "s3://test-bucket/dataset/beckham.jpeg",
  "rekognition-custom-labels-training-0": {
    "annotations": [
      {
        "class_id": 0,
        "width": 39,
        "top": 409,
        "height": 63,
        "left": 712
      },
      ...
    ],
    "image_size": [
      {
        "width": 1024,
        "depth": 3,
        "height": 768
      }
    ]
  },
  "rekognition-custom-labels-training-0-metadata": {
    "job-name": "rekognition-custom-labels-training-job",
    "class-map": {
      "0": "Cap",
      ...
    },
    "human-annotated": "yes",
    "objects": [
      {
        "confidence": 1.0
      },
      ...
    ],
  },
}
```

```
"creation-date": "2019-10-21T22:02:18.432644",
"type": "groundtruth/object-detection"
},
"rekognition-custom-labels-evaluation": {
  "annotations": [
    {
      "class_id": 0,
      "width": 39,
      "top": 409,
      "height": 63,
      "left": 712
    },
    ...
  ],
  "image_size": [
    {
      "width": 1024,
      "depth": 3,
      "height": 768
    }
  ]
},
"rekognition-custom-labels-evaluation-metadata": {
  "confidence": 0.95,
  "job-name": "rekognition-custom-labels-evaluation-job",
  "class-map": {
    "0": "Cap",
    ...
  },
  "human-annotated": "no",
  "objects": [
    {
      "confidence": 0.95,
      "rekognition-custom-labels-evaluation-details": {
        "version": 1,
        "ground-truth-labelling-jobs": ["rekognition-custom-labels-training-job"],
        "is-true-positive": true,
        "is-true-negative": false,
        "is-false-positive": false,
        "is-false-negative": false,
        "is-present-in-ground-truth": true
      }
    },
    ...
  ]
}
```

```
    ],  
    "creation-date": "2019-10-21T22:02:18.432644",  
    "type": "groundtruth/object-detection"  
  }  
}
```

## 概要ファイルと評価マニフェストスナップショット (SDK) へのアクセス

トレーニング結果を取得するには、[DescribeProjectバージョン](#) を呼び出します。サンプルコードについては、「[モデルの記述 \(SDK\)](#)」を参照してください。

DescribeProjectVersions からの ProjectVersionDescription レスポンスでメトリクスの場所が返されます。

- EvaluationResult - 概要ファイルの場所。
- TestingDataResult - テストに使用する評価マニフェストスナップショットの場所。

F1 スコアと概要ファイルの場所が EvaluationResult で返されます。例:

```
"EvaluationResult": {  
  "F1Score": 1.0,  
  "Summary": {  
    "S3Object": {  
      "Bucket": "echo-dot-scans",  
      "Name": "test-output/EvaluationResultSummary-my-echo-dots-  
project-v2.json"  
    }  
  }  
}
```

評価マニフェストのスナップショットは、[モデルのトレーニング \(SDK\)](#) で指定した `--output-config` 入力パラメータで指定された場所に保存されます。

### Note

トレーニングの料金の請求時間 (秒) が `BillableTrainingTimeInSeconds` で返されま  
す。



Amazon Rekognition Custom Labels によって返されるメトリクスの詳細については、「[Amazon Rekognition Custom Labels の評価メトリクス \(SDK\) へのアクセス](#)」を参照してください。

## モデルの混同行列の表示

混同行列を使用すると、モデル内の他のラベルと混同されているラベルを確認できます。混同行列を使用すれば、モデルの改善点に集中することができます。

モデルの評価中、Amazon Rekognition Custom Labels はテストイメージを使用して誤認された (混同した) ラベルを識別することにより、混同行列を作成します。Amazon Rekognition Custom Labels は分類モデルの混同行列のみを作成します。分類行列には、Amazon Rekognition Custom Labels がモデルトレーニング中に作成する概要ファイルからアクセスできます。Amazon Rekognition Custom Labels コンソールでは混同行列を表示できません。

### トピック

- [混同行列の使用](#)
- [モデルの混同行列の取得](#)

## 混同行列の使用

次の表は、[ルームイメージ分類](#)プロジェクト例の混同行列です。列見出しは、テストイメージに割り当てられたラベル (グラウンドトゥールースラベル) です。行見出しは、テストイメージについてモデルが予測するラベルです。各セルは、グラウンドトゥールースラベル (列) にすべき、ラベル (行) に対する予測のパーセンテージです。例えば、バスルームの予測の 67% はバスルームとして正しくラベル付けされていました。バスルームの 33% はキッチンと誤ってラベル付けされていました。予測ラベルがグラウンドトゥールースラベルと一致する場合、パフォーマンスの高いモデルではセル値が高くなります。これらは、最初から最後の予測ラベルとグラウンドトゥールースラベルを結ぶ対角線として見ることができます。セル値が 0 の場合、セルのグラウンドトゥールースラベルとなる、セルの予測ラベルに対する予測は行われていません。

### Note

モデルは確定的ではないため、ルームプロジェクトのトレーニングから得られる混同行列のセル値は、次の表と異なる場合があります。

混同行列は焦点を当てる領域を特定します。例えば、混同行列では、50% の期間にクローゼットがベッドルームと混同されていることが示されています。このような場合は、クローゼットとベッド

ルームのイメージをトレーニングデータセットにさらに追加する必要があります。また、既存のクローゼットとベッドルームのイメージに正しいラベルが付けられていることも確認してください。これにより、2つのラベルをモデルが区別しやすくなるはずですが、データセットにイメージを追加する方法については、「[データセットへのイメージの追加](#)」を参照してください。

混同行列も役立ちますが、他のメトリクスも考慮することも重要です。例えば、予測の100%が floor\_plan ラベルを正しく見つけており、これはパフォーマンスが優れていることを示しています。ただし、テストデータセットには floor\_plan ラベルの付いたイメージが2つしかありません。また、living\_space というラベルの付いたイメージが11個あります。この不均衡はトレーニングデータセット (living\_space イメージ13枚とクローゼットイメージ2枚) にもあります。より正確な評価を行うには、過小評価されているラベル (この例ではフロアプラン) のイメージをさらに追加して、トレーニングデータセットとテストデータセットのバランスを取ります。ラベルごとのテストイメージの数を取得するには、「[評価メトリクスへのアクセス \(コンソール\)](#)」を参照してください。

グラウンドトゥールスラベル

予測ラベル	裏庭	バスルーム	ベッドルーム	クローゼット	entry_way	floor_plan	front_yard	キッチン	living_space	パティオ
裏庭	75%	0%	0%	0%	0%	0%	33%	0%	0%	0%
バスルーム	0%	67%	0%	0%	0%	0%	0%	0%	0%	0%
ベッドルーム	0%	0%	82%	50%	0%	0%	0%	0%	9%	0%
クローゼット	0%	0%	0%	50%	0%	0%	0%	0%	0%	0%
entry_way	0%	0%	0%	0%	33%	0%	0%	0%	0%	0%

グラウンドトゥールースラベル

予測ラベル	裏庭	バスルーム	ベッドルーム	クローゼット	entry_way	floor_plan	front_yard	キッチン	living_space	パティオ
floor_plan	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%
front_yard	25%	0%	0%	0%	0%	0%	67%	0%	0%	0%
キッチン	0%	33%	0%	0%	0%	0%	0%	88%	0%	0%
living_space	0%	0%	18%	0%	67%	0%	0%	12%	91%	33%
パティオ	0%	0%	0%	0%	0%	0%	0%	0%	0%	67%

モデルの混同行列の取得

次のコードでは、[DescribeProjects](#)および [DescribeProject/バージョン](#) オペレーションを使用して、[モデルの概要ファイル](#)を取得します。次に、概要ファイルを使用してモデルの混同行列を表示します。

モデル (SDK) の混同行列を表示する方法

1. 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 4: AWS CLI と AWS SDKsを設定する](#)」を参照してください。
2. 次のコードを使用してモデルの混同行列を表示します。次のコマンドライン引数を指定します。
  - project\_name - 使用するプロジェクトの名前。プロジェクト名は、Amazon Rekognition Custom Labels コンソールのプロジェクトページから取得できます。

- `version_name` - 使用するモデルのバージョン。バージョン名は、Amazon Rekognition Custom Labels コンソールのプロジェクト詳細ページから取得できます。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

"""
Purpose

Shows how to display the confusion matrix for an Amazon Rekognition Custom labels
image
classification model.
"""

import json
import argparse
import logging
import boto3
import pandas as pd
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def get_model_summary_location(rek_client, project_name, version_name):
    """
    Get the summary file location for a model.

    :param rek_client: A Boto3 Rekognition client.
    :param project_arn: The Amazon Resource Name (ARN) of the project that contains
    the model.
    :param model_arn: The Amazon Resource Name (ARN) of the model.
    :return: The location of the model summary file.
    """

    try:
        logger.info(
            "Getting summary file for model %s in project %s.", version_name,
            project_name)
    
```

```
summary_location = ""

# Get the project ARN from the project name.
response = rek_client.describe_projects(ProjectNames=[project_name])

assert len(response['ProjectDescriptions']) > 0, \
    f"Project {project_name} not found."

project_arn = response['ProjectDescriptions'][0]['ProjectArn']

# Get the summary file location for the model.
describe_response =
rek_client.describe_project_versions(ProjectArn=project_arn,
VersionNames=[version_name])
assert len(describe_response['ProjectVersionDescriptions']) > 0, \
    f"Model {version_name} not found."

model=describe_response['ProjectVersionDescriptions'][0]

evaluation_results=model['EvaluationResult']

summary_location=(f"s3://{evaluation_results['Summary']['S3Object']
['Bucket']}"
                  f"/{evaluation_results['Summary']['S3Object']
['Name']}")

return summary_location

except ClientError as err:
    logger.exception(
        "Couldn't get summary file location: %s", err.response['Error']
['Message'])
    raise

def show_confusion_matrix(summary):
    """
    Shows the confusion matrix for an Amazon Rekognition Custom Labels
    image classification model.
    :param summary: The summary file JSON object.
    """
    pd.options.display.float_format = '{:.0%}'.format
```

```
# Load the model summary JSON into a DataFrame.

summary_df = pd.DataFrame(
    summary['AggregatedEvaluationResults']['ConfusionMatrix'])

# Get the confusion matrix.
confusion_matrix = summary_df.pivot_table(index='PredictedLabel',
                                           columns='GroundTruthLabel',
                                           fill_value=0.0).astype(float)

# Display the confusion matrix.
print(confusion_matrix)

def get_summary(s3_resource, summary):
    """
    Gets the summary file.
    : return: The summary file in bytes.
    """
    try:
        summary_bucket, summary_key = summary.replace(
            "s3://", "").split("/", 1)

        bucket = s3_resource.Bucket(summary_bucket)
        obj = bucket.Object(summary_key)
        body = obj.get()['Body'].read()
        logger.info(
            "Got summary file '%s' from bucket '%s'.",
            obj.key, obj.bucket_name)
    except ClientError:
        logger.exception(
            "Couldn't get summary file '%s' from bucket '%s'.",
            obj.key, obj.bucket_name)
        raise
    else:
        return body

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    : param parser: The command line parser.
    """
```

```
parser.add_argument(
    "project_name", help="The ARN of the project in which the model resides."
)
parser.add_argument(
    "version_name", help="The version of the model that you want to describe."
)

def main():
    """
    Entry point for script.
    """

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # Get the command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(
            f"Showing confusion matrix for: {args.version_name} for project
{args.project_name}.")

        session = boto3.Session(profile_name='custom-labels-access')
        rekognition_client = session.client("rekognition")
        s3_resource = session.resource('s3')

        # Get the summary file for the model.
        summary_location = get_model_summary_location(rekognition_client,
args.project_name,
                                                    args.version_name
                                                    )
        summary = json.loads(get_summary(s3_resource, summary_location))

        # Check that the confusion matrix is available.
        assert 'ConfusionMatrix' in summary['AggregatedEvaluationResults'], \
            "Confusion matrix not found in summary. Is the model a classification
model?"

        # Show the confusion matrix.
```

```
show_confusion_matrix(summary)
print("Done")

except ClientError as err:
    logger.exception("Problem showing confusion matrix: %s", err)
    print(f"Problem describing model: {err}")

except AssertionError as err:
    logger.exception(
        "Error: %s.\n", err)
    print(
        f"Error: {err}\n")

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

## 参照: トレーニング結果概要ファイル

トレーニング結果の概要には、モデルの評価に使用できるメトリクスが含まれています。概要ファイルは、コンソールのトレーニング結果ページにメトリクスを表示するためにも使用されます。概要ファイルは、トレーニング後に Amazon S3 バケットに保存されます。概要ファイルを取得するには、DescribeProjectVersion を呼び出します。サンプルコードについては、「[概要ファイルと評価マニフェストスナップショット \(SDK\) へのアクセス](#)」を参照してください。

### 概要ファイル

次の JSON が概要ファイルの形式です。

#### EvaluationDetails ( セクション 3)

トレーニングタスクに関する概要情報。これには、モデルが属するプロジェクトの ARN (ProjectVersionArn)、トレーニングが終了した日付と時刻、評価されたモデルのバージョン (EvaluationEndTimestamp)、トレーニング中に検出されたラベルのリスト (Labels) が含まれます。また、トレーニング (NumberOfTrainingImages) と評価 (NumberOfTestingImages) に使用されるイメージの数も含まれます。

#### AggregatedEvaluationResults ( セクション 1)

AggregatedEvaluationResults をテストデータセットと併用すると、トレーニング済みモデルの全体的なパフォーマンスを評価できます。Precision、Recall、F1Score メトリク



スには集計メトリクスが含まれます。オブジェクト検出 (イメージ上のオブジェクトの位置) では、AverageRecall (mAR) と AveragePrecision (mAP) メトリックが返されます。分類 (イメージ内のオブジェクトのタイプ) の場合、混同行列メトリクスが返されます。

## LabelEvaluationResults (セクション 2)

labelEvaluationResults を使用して個々のラベルのパフォーマンスを評価できます。ラベルは各ラベルの F1 スコアでソートされます。含まれるメトリクスは Precision、Recall、F1Score、Threshold (分類に使用) です。

ファイル形式は EvaluationSummary-ProjectName-VersionName.json となります。

```
{
  "Version": "integer",
  // section-3
  "EvaluationDetails": {
    "ProjectVersionArn": "string",
    "EvaluationEndTimestamp": "string",
    "Labels": "[string]",
    "NumberOfTrainingImages": "int",
    "NumberOfTestingImages": "int"
  },
  // section-1
  "AggregatedEvaluationResults": {
    "Metrics": {
      "Precision": "float",
      "Recall": "float",
      "F1Score": "float",
      // The following 2 fields are only applicable to object detection
      "AveragePrecision": "float",
      "AverageRecall": "float",
      // The following field is only applicable to classification
      "ConfusionMatrix": [
        {
          "GroundTruthLabel": "string",
          "PredictedLabel": "string",
          "Value": "float"
        },
        ...
      ],
    }
  },
}
```

```
// section-2
"LabelEvaluationResults": [
  {
    "Label": "string",
    "NumberOfTestingImages": "int",
    "Metrics": {
      "Threshold": "float",
      "Precision": "float",
      "Recall": "float",
      "F1Score": "float"
    },
  },
  ...
]
```

## Amazon Rekognition Custom Labels モデルの改善

機械学習モデルのパフォーマンスは、カスタムラベル (関心のある特定のオブジェクトやシーン) の複雑さや変動性、提供するトレーニングデータセットの品質と代表力、モデルのトレーニングに使用されるモデルフレームワークと機械学習手法などの要因に大きく依存します。

Amazon Rekognition Custom Labels を使用すると、このプロセスが簡単になり、機械学習の専門知識は必要ありません。ただし、優れたモデルを構築するプロセスでは、多くの場合望ましいパフォーマンスを実現するためにデータとモデル改善の反復が必要です。以下は、モデルを改善する方法に関する情報です。

### [データ]

一般に、より質の高いデータを大量に収集することにより、モデルの品質を向上させることができます。オブジェクトまたはシーンをはっきりと表示し、不要なアイテムが散らかっていないトレーニングイメージを使用してください。オブジェクトを囲む境界ボックスには、オブジェクトが完全に見え、他のオブジェクトに遮られないトレーニングイメージを使用してください。

トレーニングデータセットとテストデータセットが、最終的に推定対象となるイメージの種類と一致していることを確認してください。ロゴのようにトレーニング例がわずかしかないオブジェクトでは、テストイメージのロゴの周囲に境界ボックスを配置する必要があります。これらのイメージは、オブジェクトをローカライズするシナリオを表したり、描写したりします。

トレーニングデータセットまたはテストデータセットにイメージを追加する方法については、「[データセットへのイメージの追加](#)」を参照してください。

## 偽陽性の削減 (適合率の向上)

- まず、想定しきい値を増やすことによって、偽陽性を減らしながら正しい予測を維持できるかどうかを確認します。ある時点では、特定のモデルの適合率と再現率のトレードオフが原因で、この効果が低下します。ラベルに想定しきい値を設定することはできませんが、MinConfidence 入力パラメータに DetectCustomLabels の高い値を指定しても同じ結果が得られます。詳細については、「[トレーニングされたモデルによるイメージの分析](#)」を参照してください。
- 対象の 1 つ以上のカスタムラベル (A) が、同じクラスのオブジェクト (関心のあるラベルではない) (B) と頻繁に混同されることがあります。わかりやすいように、B をオブジェクトクラスラベルとして (偽陽性のあるイメージと一緒に) トレーニングデータセットに追加してください。つまり、新しいトレーニングイメージを通じて、モデルが A ではなく B を予測できるように学習させます。トレーニングデータセットにイメージを追加する方法については、「[データセットへのイメージの追加](#)」を参照してください。
- 2 つのカスタムラベル (A と B) によってモデルが混同されていることに気付くかもしれません。ラベル A のテストイメージはラベル B であると予測され、その逆も同様です。その場合は、まずトレーニングセットとテストセットに誤ったラベルの付いたイメージがないか確認します。データセットギャラリーを使用して、データセットに割り当てられたラベルを管理します。詳細については、「[ラベルの管理](#)」を参照してください。また、この種の混同に関連してトレーニングイメージを追加すると、再トレーニングされたモデルが A と B をより適切に区別しやすくなります。トレーニングデータセットにイメージを追加する方法については、「[データセットへのイメージの追加](#)」を参照してください。

## 偽陰性の削減 (再現率の向上)

- 想定しきい値には低い値を使用してください。ラベルに想定しきい値を設定することはできませんが、MinConfidence 入力パラメータに DetectCustomLabels の低い値を指定しても同じ結果が得られます。詳細については、「[トレーニングされたモデルによるイメージの分析](#)」を参照してください。
- より適切な例を使用して、オブジェクトとそれが表示されるイメージの両方の多様性をモデル化してください。
- ラベルを覚えやすい 2 つのクラスに分けてください。例えば、良いクッキーと悪いクッキーの代わりに、良いクッキー、焦げたクッキー、欠けたクッキーを用意して、モデルがそれぞれのユニークな概念をよりよく学習できるようにします。

# トレーニング済み Amazon Rekognition Custom Labels の実行

モデルのパフォーマンスに満足している場合は、使用を開始できます。コンソールまたは AWS SDK を使用してモデルを開始および停止できます。コンソールには、使用可能な SDK オペレーションの例も含まれています。

## トピック

- [推論単位](#)
- [アベイラビリティゾーン](#)
- [Amazon Rekognition Custom Labels モデルを開始する](#)
- [Amazon Rekognition Custom Labels モデルの停止](#)
- [実行時間と使用された推論単位のレポート](#)

## 推論単位

モデルの開始時に、モデルが使用する推論単位と呼ばれるコンピューティングリソースの数を指定します。

### Important

モデルの実行の設定方法に基づいて、モデルの実行時間数と、モデルの実行中にモデルが使用する推論単位の数に対して課金されます。例えば、モデルを 2 推論単位で開始し、モデルを 8 時間使用すると、16 推論時間 (8 時間実行時間 × 2 推論単位) に対して課金されます。詳細については、「[推論時間](#)」を参照してください。明示的に[モデルを停止](#)しないと、モデルでイメージをアクティブに分析していない場合でも課金されます。

1 つの推論単位がサポートする 1 秒あたりのトランザクション (TPS) は、次の影響を受けます。

- 一般に、イメージレベルのラベルを検出するモデル (分類) は、境界ボックスでオブジェクトを検出してローカライズするモデル (オブジェクト検出) よりも TPS が高くなります。
- モデルの複雑さ。
- 高解像度のイメージでは、分析に時間がかかります。

- イメージ内のオブジェクトが多いほど、分析に時間がかかります。
- 小さなイメージは、大きなイメージよりも速く分析されます。
- イメージバイトとして渡されたイメージは、最初に 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参照するよりも速く分析されます。イメージバイトとして渡されるイメージは 4.0 MB 未満である必要があります。イメージをほぼリアルタイムで処理する場合や、イメージサイズが 4.0 MB 未満の場合は、イメージバイトを使用することをお勧めします。例えば、IP カメラからキャプチャされたイメージなど。
- Amazon S3 バケットに保存されたイメージの処理は、イメージをダウンロードしてイメージバイトに変換し、分析用にイメージバイトを渡すよりも高速です。
- Amazon S3 バケットに既に保存されたイメージの分析は、イメージバイトとして渡されたイメージを分析するよりも高速です。これは特にイメージサイズが大きい場合に当てはまります。

DetectCustomLabels への呼び出し数が、モデルが使用する推論単位の合計でサポートされる最大 TPS を超えると、Amazon Rekognition Custom Labels は ProvisionedThroughputExceededException 例外を返します。

## 推論単位によるスループットの管理

アプリケーションの要求に応じて、モデルのスループットを増やしたり減らしたりすることができます。スループットを増やすには、追加の推論ユニットを使用します。推論ユニットを追加するたびに、処理速度が 1 推論ユニット増加します。必要な推論ユニット数の計算方法については、「[Amazon Rekognition Custom Labels と Amazon Lookout for Vision モデルの推論単を計算する](#)」を参照してください。モデルのサポート対象スループットを変更する場合は、次の 2 つのオプションがあります：

### 手動で推論ユニットを追加または削除する

モデルを**停止**し、必要な推論ユニット数で**再開**します。この方法の欠点は、再開中はモデルがリクエストを受け取ることができず、需要の急増に対処できないことです。モデルのスループットが安定していて、ユースケースが 10 ~ 20 分のダウンタイムを許容できる場合は、このアプローチを使用してください。例としては、週ユニットのスケジュールを使用してモデルへの呼び出しをバッチ処理する場合などが挙げられます。

## 推論単位の自動スケーリング

モデルの需要の急増に対応する必要がある場合、Amazon Rekognition Custom Labels はモデルが使用する推論単位の数を自動的にスケールできます。需要が増加すると、Amazon Rekognition Custom Labels はモデルに追加の推論単位を追加し、需要が減少するとそれらを削除します。

Amazon Rekognition Custom Labels がモデルの推論単位を自動的にスケールできるようにするには、モデルを[開始し](#)、MaxInferenceUnits パラメータを使用して使用できる推論単位の最大数を設定します。推論単位の最大数を設定すると、使用可能な推論単位の数を制限することにより、モデルの実行コストを管理できます。最大単位数を指定しない場合、Amazon Rekognition Custom Labels はモデルを自動的にスケールせず、最初に使用した推論単位数のみを使用します。推論単位の最大数については、「[Service Quotas](#)」を参照してください。

MinInferenceUnits パラメータを使用して推論ユニットの最小数も指定できます。これにより、モデルの最小スループットを指定できます。1つの推論単位は1時間の処理時間を表します。

### Note

Amazon Rekognition Custom Labels では、推論単位の最大数を設定することはできません。代わりに、StartProjectVersion オペレーションに MaxInferenceUnits 入力パラメータを指定してください。

Amazon Rekognition Custom Labels には、モデルの現在の自動スケーリングステータスを判断するために使用できる以下の Amazon CloudWatch Logs メトリクスが用意されています。

メトリクス	説明
DesiredInferenceUnits	Amazon Rekognition Custom Labels がスケールアップまたはスケールダウンの対象となる推論単位の数。
InServiceInferenceUnits	モデルが使用している推論単位の数。

DesiredInferenceUnits = InServiceInferenceUnits の場合、Amazon Rekognition Custom Labels は現在、推論単位の数をスケールしていません。

`DesiredInferenceUnits > InServiceInferenceUnits` の場合、Amazon Rekognition Custom Labels は `DesiredInferenceUnits` の値までスケールアップされます。

`DesiredInferenceUnits < InServiceInferenceUnits` の場合、Amazon Rekognition Custom Labels は `DesiredInferenceUnits` の値にスケールダウンされます。

Amazon Rekognition Custom Labels によって返されるメトリクスとフィルタリングディメンションの詳細については、「[CloudWatch Rekognition のメトリクス](#)」を参照してください。

モデルにリクエストした推論単位の最大数を確認するには、`DescribeProjectsVersion` を呼び出して応答の `MaxInferenceUnits` フィールドを確認します。サンプルコードについては、「[モデルの記述 \(SDK\)](#)」を参照してください。

## アベイラビリティゾーン

Amazon Rekognition Custom Labels は、AWS リージョン内の複数のアベイラビリティゾーンに推論単位を分散させ、可用性を高めています。詳細については、「[アベイラビリティゾーン](#)」を参照してください。アベイラビリティゾーンの停止や推論ユニットの障害から量産モデルを保護するために、少なくとも2つの推論ユニットを使用して量産モデルを開始することを強くお勧めします。

アベイラビリティゾーンの停止が発生すると、アベイラビリティゾーン内のすべての推論ユニットが使用できなくなり、モデル容量が削減されます。[DetectCustomラベル](#)への呼び出しは、残りの推論ユニットに再分散されます。このような呼び出しは、残りの推論単位のサポートされる TPS (Transactions Per Seconds) を超えていなければ成功します。AWS がアベイラビリティゾーンを修復した後、推論単位が再起動され、フルキャパシティが復元されます。

1つの推論単位に障害が発生すると、Amazon Rekognition Custom Labels は同じアベイラビリティゾーンで新しい推論単位を自動的に開始します。新しい推論単位が始まるまで、モデル容量は減少します。

## Amazon Rekognition Custom Labels モデルを開始する

Amazon Rekognition Custom Labels モデルの実行を開始するには、コンソールを使用するか、[StartProject/バージョン](#)オペレーションを使用します。



### ⚠ Important

モデルを停止するモデルの稼働時間数と、モデルの実行中にモデルが使用する推論単位の数に対して課金されます。詳細については、「[トレーニング済み Amazon Rekognition Custom Labels の実行](#)」を参照してください。

モデルの開始には数分かかることがあります。モデルの準備状況の現在のステータスを確認するには、プロジェクトの詳細ページを確認するか、[DescribeProject/バージョン](#) を使用します。

モデルが開始されたら、[DetectCustomラベル](#) を使用してモデルを使用してイメージを分析します。詳細については、「[トレーニングされたモデルによるイメージの分析](#)」を参照してください。コンソールには DetectCustomLabels を呼び出すサンプルコードも用意されています。

### トピック

- [Amazon Rekognition Custom Labels モデルの開始 \(コンソール\)](#)
- [Amazon Rekognition Custom Labels モデル \(SDK\) を開始します。](#)

## Amazon Rekognition Custom Labels モデルの開始 (コンソール)

以下の手順を使用して、コンソールで Amazon Rekognition Custom Labels モデルの実行を開始します。モデルをコンソールから直接起動することも、コンソールが提供する AWS SDK コードを使用することもできます。

モデル (コンソール) を開始するには

1. Amazon Rekognition コンソールを <https://console.aws.amazon.com/rekognition/> で開きます。
2. [カスタムラベルを使用] を選択します。
3. [開始方法] を選択します。
4. 左側のナビゲーションペインで、[プロジェクト] を選択します。
5. [プロジェクト] リソースページで、開始するモデルを含むプロジェクトを選択します。
6. [モデル] セクションで、開始するモデルを選択します。
7. [モデルを使用] タブを選択します。
8. 次のいずれかを行います。



## Start model using the console

[モデルの開始または停止] セクションで、次の操作を行います。

1. 使用する推論単位の数を選択します。詳細については、「[トレーニング済み Amazon Rekognition Custom Labels の実行](#)」を参照してください。
2. [開始] を選択します。
3. [モデルを開始] ダイアログボックスで、[開始] を選択します。

## Start model using the AWS SDK

[自分のモデルを使用] セクションで、次の操作を行います。

1. [API コード] を選択します。
2. [AWS CLI] または [Python] のいずれかを選択します。
3. [モデルを開始] で、サンプルコードをコピーします。
4. このサンプルコードを使用して、モデルを開始します。詳細については、「[Amazon Rekognition Custom Labels モデル \(SDK\) を開始します。](#)」を参照してください。
9. プロジェクトの概要ページに戻るには、ページの上でプロジェクト名を選択します。
10. [モデル] セクションで、モデルのステータスを確認します。モデルのステータスが RUNNING の場合、モデルを使用してイメージを分析できます。詳細については、「[トレーニングされたモデルによるイメージの分析](#)」を参照してください。

## Amazon Rekognition Custom Labels モデル (SDK) を開始します。

モデルを開始するには、[StartProject/バージョン](#) API を呼び出し、ProjectVersionArn 入力パラメータでモデルの Amazon リソースネーム (ARN) を渡します。使用する推論単位の数も指定します。詳細については、「[トレーニング済み Amazon Rekognition Custom Labels の実行](#)」を参照してください。

モデルの開始までに時間がかかる場合があります。このトピックの Python と Java の例では、ウェーターを使用してモデルの開始を待ちます。ウェーターは、発生する特定の状態をポーリングするユーティリティメソッドです。または、[DescribeProject/バージョン](#) を呼び出して現在のステータスを確認することもできます。

## モデル (SDK) を開始するには

1. 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 4: AWS CLI と AWS SDKsを設定する](#)」を参照してください。
2. 次のサンプルコードを使用して、モデルを開始します。

### CLI

`project-version-arn` の値を開始するモデルの ARN に変更します。 `--min-inference-units` の値を使用する推論単位の数に変更します。必要に応じて、 `--max-inference-units` を Amazon Rekognition Custom Labels がモデルを自動的にスケールするために使用できる推論単位の最大数に変更します。

```
aws rekognition start-project-version --project-version-arn model_arn \  
  --min-inference-units minimum number of units \  
  --max-inference-units maximum number of units \  
  --profile custom-labels-access
```

### Python

次のコマンドラインパラメータを指定します。

- `project_arn` - 開始するモデルを含むプロジェクトの ARN。
- `model_arn` - 開始するモデルの ARN。
- `min_inference_units` - 使用する推論単位の数。
- (オプション) `--max_inference_units` Amazon Rekognition Custom Labels がモデルの自動スケーリングに使用できる推論単位の最大数。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
  
"""  
Purpose  
Shows how to start running an Amazon Lookout for Vision model.  
"""  
  
import argparse
```

```
import logging
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def get_model_status(rek_client, project_arn, model_arn):
    """
    Gets the current status of an Amazon Rekognition Custom Labels model
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_name: The name of the project that you want to use.
    :param model_arn: The name of the model that you want the status for.
    :return: The model status
    """

    logger.info("Getting status for %s.", model_arn)

    # Extract the model version from the model arn.
    version_name = (model_arn.split("version/", 1)[1]).rpartition('/')[0]

    models = rek_client.describe_project_versions(ProjectArn=project_arn,
                                                  VersionNames=[version_name])

    for model in models['ProjectVersionDescriptions']:

        logger.info("Status: %s", model['StatusMessage'])
        return model["Status"]

    error_message = f"Model {model_arn} not found."
    logger.exception(error_message)
    raise Exception(error_message)

def start_model(rek_client, project_arn, model_arn, min_inference_units,
               max_inference_units=None):
    """
    Starts the hosting of an Amazon Rekognition Custom Labels model.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_name: The name of the project that contains the
    model that you want to start hosting.
    :param min_inference_units: The number of inference units to use for
    hosting.
    """
```

```
    :param max_inference_units: The number of inference units to use for auto-
scaling
    the model. If not supplied, auto-scaling does not happen.
    """

    try:
        # Start the model
        logger.info(f"Starting model: {model_arn}. Please wait....")

        if max_inference_units is None:
            rek_client.start_project_version(ProjectVersionArn=model_arn,
MinInferenceUnits=int(min_inference_units))
        else:
            rek_client.start_project_version(ProjectVersionArn=model_arn,
                                           MinInferenceUnits=int(
                                               min_inference_units),
                                           MaxInferenceUnits=int(max_inference_units))

        # Wait for the model to be in the running state
        version_name = (model_arn.split("version/", 1)[1]).rpartition('/')[0]
        project_version_running_waiter = rek_client.get_waiter(
            'project_version_running')
        project_version_running_waiter.wait(
            ProjectArn=project_arn, VersionNames=[version_name])

        # Get the running status
        return get_model_status(rek_client, project_arn, model_arn)

    except ClientError as err:
        logger.exception("Client error: Problem starting model: %s", err)
        raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The ARN of the project that contains that the model
you want to start."
```

```
)
parser.add_argument(
    "model_arn", help="The ARN of the model that you want to start."
)
parser.add_argument(
    "min_inference_units", help="The minimum number of inference units to
use."
)
parser.add_argument(
    "--max_inference_units", help="The maximum number of inference units to
use for auto-scaling the model.", required=False
)

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        # Start the model.
        session = boto3.Session(profile_name='custom-labels-access')
        rekognition_client = session.client("rekognition")

        status = start_model(rekognition_client,
                             args.project_arn, args.model_arn,
                             args.min_inference_units,
                             args.max_inference_units)

        print(f"Finished starting model: {args.model_arn}")
        print(f"Status: {status}")

    except ClientError as err:
        error_message = f"Client error: Problem starting model: {err}"
        logger.exception(error_message)
        print(error_message)

    except Exception as err:
```

```
        error_message = f"Problem starting model:{err}"
        logger.exception(error_message)
        print(error_message)

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

## Java V2

次のコマンドラインパラメータを指定します。

- `project_arn` - 開始するモデルを含むプロジェクトの ARN。
- `model_arn` - 開始するモデルの ARN。
- `min_inference_units` - 使用する推論単位の数。
- (オプション) `max_inference_units` - Amazon Rekognition Custom Labels がモデルを自動的にスケールするために使用できる推論単位の最大数。値を指定しない場合、自動スケーリングは行われません。

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
*/
package com.example.rekognition;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsResponse;
import
    software.amazon.awssdk.services.rekognition.model.ProjectVersionDescription;
import software.amazon.awssdk.services.rekognition.model.ProjectVersionStatus;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
    software.amazon.awssdk.services.rekognition.model.StartProjectVersionRequest;
import
    software.amazon.awssdk.services.rekognition.model.StartProjectVersionResponse;
```

```
import software.amazon.awssdk.services.rekognition.waiters.RekognitionWaiter;

import java.util.Optional;
import java.util.logging.Level;
import java.util.logging.Logger;

public class StartModel {

    public static final Logger logger =
        Logger.getLogger(StartModel.class.getName());

    public static int findForwardSlash(String modelArn, int n) {

        int start = modelArn.indexOf('/');
        while (start >= 0 && n > 1) {
            start = modelArn.indexOf('/', start + 1);
            n -= 1;
        }
        return start;
    }

    public static void startMyModel(RekognitionClient rekClient, String
projectArn, String modelArn,
        Integer minInferenceUnits, Integer maxInferenceUnits
        ) throws Exception, RekognitionException {

        try {

            logger.log(Level.INFO, "Starting model: {0}", modelArn);

            StartProjectVersionRequest startProjectVersionRequest = null;

            if (maxInferenceUnits == null) {
                startProjectVersionRequest =
                StartProjectVersionRequest.builder()
                    .projectVersionArn(modelArn)
                    .minInferenceUnits(minInferenceUnits)
                    .build();
            }
            else {
```

```
        startProjectVersionRequest =
StartProjectVersionRequest.builder()
        .projectVersionArn(modelArn)
        .minInferenceUnits(minInferenceUnits)
        .maxInferenceUnits(maxInferenceUnits)
        .build();

    }

    StartProjectVersionResponse response =
rekClient.startProjectVersion(startProjectVersionRequest);

    logger.log(Level.INFO, "Status: {0}", response.statusAsString() );

    // Get the model version

    int start = findForwardSlash(modelArn, 3) + 1;
    int end = findForwardSlash(modelArn, 4);

    String versionName = modelArn.substring(start, end);

    // wait until model starts

    DescribeProjectVersionsRequest describeProjectVersionsRequest =
DescribeProjectVersionsRequest.builder()
        .versionNames(versionName)
        .projectArn(projectArn)
        .build();

    RekognitionWaiter waiter = rekClient.waiter();

    WaiterResponse<DescribeProjectVersionsResponse> waiterResponse =
waiter

    .waitUntilProjectVersionRunning(describeProjectVersionsRequest);

    Optional<DescribeProjectVersionsResponse> optionalResponse =
waiterResponse.matched().response();

    DescribeProjectVersionsResponse describeProjectVersionsResponse =
optionalResponse.get();
```



```
        for (ProjectVersionDescription projectVersionDescription :
describeProjectVersionsResponse
            .projectVersionDescriptions()) {
            if(projectVersionDescription.status() ==
ProjectVersionStatus.RUNNING) {
                logger.log(Level.INFO, "Model is running" );

            }
            else {
                String error = "Model training failed: " +
projectVersionDescription.statusAsString() + " "
                    + projectVersionDescription.statusMessage() + " " +
modelArn;

                logger.log(Level.SEVERE, error);
                throw new Exception(error);
            }

        }

    } catch (RekognitionException e) {
        logger.log(Level.SEVERE, "Could not start model: {0}",
e.getMessage());
        throw e;
    }

}

public static void main(String[] args) {

    String modelArn = null;
    String projectArn = null;
    Integer minInferenceUnits = null;
    Integer maxInferenceUnits = null;

    final String USAGE = "\n" + "Usage: " + "<project_name> <version_name>
<min_inference_units> <max_inference_units>\n\n" + "Where:\n"
        + "    project_arn - The ARN of the project that contains the
model that you want to start. \n\n"
        + "    model_arn - The ARN of the model version that you want to
start.\n\n"
```

```
        + "    min_inference_units - The number of inference units to
start the model with.\n\n"
        + "    max_inference_units - The maximum number of inference
units that Custom Labels can use to "
        + "    automatically scale the model. If the value is null,
automatic scaling doesn't happen.\n\n";

    if (args.length < 3 || args.length >4) {
        System.out.println(USAGE);
        System.exit(1);
    }

    projectArn = args[0];
    modelArn = args[1];
    minInferenceUnits=Integer.parseInt(args[2]);

    if (args.length == 4) {
        maxInferenceUnits = Integer.parseInt(args[3]);
    }

    try {

        // Get the Rekognition client.
        RekognitionClient rekClient = RekognitionClient.builder()
            .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
            .region(Region.US_WEST_2)
            .build();

        // Start the model.
        startMyModel(rekClient, projectArn, modelArn, minInferenceUnits,
maxInferenceUnits);

        System.out.println(String.format("Model started: %s", modelArn));

        rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    } catch (Exception rekError) {
```

```
        logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());
        System.exit(1);
    }

}

}
```

## Amazon Rekognition Custom Labels モデルの停止

Amazon Rekognition Custom Labels モデルの実行を停止するには、コンソールを使用するか、[StopProject/バージョン](#) オペレーションを使用します。

トピック

- [Amazon Rekognition Custom Labels モデルの停止 \(コンソール\)](#)
- [Amazon Rekognition Custom Labels モデル \(SDK\) の停止](#)

### Amazon Rekognition Custom Labels モデルの停止 (コンソール)

次の手順を使用して、実行中の Amazon Rekognition Custom Labels モデルをコンソールで停止します。モデルをコンソールから直接停止することも、コンソールが提供する AWS SDK コードを使用することもできます。

モデルを停止するには (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. [カスタムラベルを使用] を選択します。
3. [開始方法] を選択します。
4. 左側のナビゲーションペインで、[プロジェクト] を選択します。
5. [プロジェクト] リソースページで、停止するモデルを含むプロジェクトを選択します。
6. [モデル] セクションで、停止するモデルを選択します。
7. [モデルを使用] タブを選択します。

## 8. Stop model using the console

1. [モデルの開始または停止] セクションで [停止] を選択します。
2. [モデルを停止] ダイアログボックスで、[停止] と入力し、モデルを停止することを確認します。
3. [停止] を選択してモデルを停止します。

## Stop model using the AWS SDK

[自分のモデルを使用] セクションで、次の操作を行います。

1. [API コード] を選択します。
  2. [AWS CLI] または [Python] のいずれかを選択します。
  3. [モデルを停止] で、サンプルコードをコピーします。
  4. サンプルコードを使用してモデルを停止します。詳細については、「[Amazon Rekognition Custom Labels モデル \(SDK\) の停止](#)」を参照してください。
9. ページ上部でプロジェクト名を選択すると、プロジェクトの概要ページに戻ります。
10. [モデル] セクションで、モデルのステータスを確認します。モデルのステータスが [停止しました] の場合、モデルは停止しています。

## Amazon Rekognition Custom Labels モデル (SDK) の停止

モデルを停止するには、[StopProjectバージョン](#) API を呼び出し、ProjectVersionArn 入力パラメータでモデルの Amazon リソースネーム (ARN) を渡します。

モデルの停止までに時間がかかる場合があります。現在のステータスを確認するには、DescribeProjectVersions を使用します。

モデル (SDK) を停止するには

1. 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 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. 次のサンプルコードを使用して、実行中のモデルを停止します。

## CLI

`project-version-arn` の値を、停止するモデルバージョンの ARN に変更します。

```
aws rekognition stop-project-version --project-version-arn "model arn" \  
--profile custom-labels-access
```

## Python

次の例では、既に実行されているモデルを停止します。

次のコマンドラインパラメータを指定します。

- `project_arn` - 停止するモデルが含まれているプロジェクトの ARN。
- `model_arn` - 停止するモデルの ARN。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
  
"""  
Purpose  
Shows how to stop a running Amazon Lookout for Vision model.  
"""  
  
import argparse  
import logging  
import time  
import boto3  
  
from botocore.exceptions import ClientError  
  
logger = logging.getLogger(__name__)  
  
def get_model_status(rek_client, project_arn, model_arn):  
    """  
    Gets the current status of an Amazon Rekognition Custom Labels model  
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.  
    :param project_name: The name of the project that you want to use.  
    :param model_arn: The name of the model that you want the status for.  
    """
```

```
"""

logger.info ("Getting status for %s.", model_arn)

# Extract the model version from the model arn.
version_name=(model_arn.split("version/",1)[1]).rpartition('/')[0]

# Get the model status.
models=rek_client.describe_project_versions(ProjectArn=project_arn,
VersionNames=[version_name])

for model in models['ProjectVersionDescriptions']:
    logger.info("Status: %s",model['StatusMessage'])
    return model["Status"]

# No model found.
logger.exception("Model %s not found.", model_arn)
raise Exception("Model %s not found.", model_arn)

def stop_model(rek_client, project_arn, model_arn):
    """
    Stops a running Amazon Rekognition Custom Labels Model.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_arn: The ARN of the project that you want to stop running.
    :param model_arn: The ARN of the model (ProjectVersion) that you want to
    stop running.
    """

    logger.info("Stopping model: %s", model_arn)

    try:
        # Stop the model.
        response=rek_client.stop_project_version(ProjectVersionArn=model_arn)

        logger.info("Status: %s", response['Status'])

        # stops when hosting has stopped or failure.
        status = ""
        finished = False

        while finished is False:

            status=get_model_status(rek_client, project_arn, model_arn)
```

```
        if status == "STOPPING":
            logger.info("Model stopping in progress...")
            time.sleep(10)
            continue
        if status == "STOPPED":
            logger.info("Model is not running.")
            finished = True
            continue

        error_message = f"Error stopping model. Unexpected state: {status}"
        logger.exception(error_message)
        raise Exception(error_message)

    logger.info("finished. Status %s", status)
    return status

except ClientError as err:
    logger.exception("Couldn't stop model - %s: %s",
                    model_arn, err.response['Error']['Message'])
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The ARN of the project that contains the model that
you want to stop."
    )
    parser.add_argument(
        "model_arn", help="The ARN of the model that you want to stop."
    )

def main():

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments.
```

```
parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
add_arguments(parser)
args = parser.parse_args()

# Stop the model.
session = boto3.Session(profile_name='custom-labels-access')
rekognition_client = session.client("rekognition")

status=stop_model(rekognition_client, args.project_arn, args.model_arn)

print(f"Finished stopping model: {args.model_arn}")
print(f"Status: {status}")

except ClientError as err:
    logger.exception("Problem stopping model:%s",err)
    print(f"Failed to stop model: {err}")

except Exception as err:
    logger.exception("Problem stopping model:%s", err)
    print(f"Failed to stop model: {err}")

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

## Java V2

次のコマンドラインパラメータを指定します。

- `project_arn` - 停止するモデルが含まれているプロジェクトの ARN。
- `model_arn` - 停止するモデルの ARN。

```
/*
 Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 SPDX-License-Identifier: Apache-2.0
*/

package com.example.rekognition;

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.DescribeProjectVersionsRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsResponse;
import
    software.amazon.awssdk.services.rekognition.model.ProjectVersionDescription;
import software.amazon.awssdk.services.rekognition.model.ProjectVersionStatus;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
    software.amazon.awssdk.services.rekognition.model.StopProjectVersionRequest;
import
    software.amazon.awssdk.services.rekognition.model.StopProjectVersionResponse;

import java.util.logging.Level;
import java.util.logging.Logger;

public class StopModel {

    public static final Logger logger =
        Logger.getLogger(StopModel.class.getName());

    public static int findForwardSlash(String modelArn, int n) {

        int start = modelArn.indexOf('/');
        while (start >= 0 && n > 1) {
            start = modelArn.indexOf('/', start + 1);
            n -= 1;
        }
        return start;
    }

    public static void stopMyModel(RekognitionClient rekClient, String
projectArn, String modelArn)
        throws Exception, RekognitionException {

        try {

            logger.log(Level.INFO, "Stopping {0}", modelArn);

            StopProjectVersionRequest stopProjectVersionRequest =
                StopProjectVersionRequest.builder()
                    .projectVersionArn(modelArn).build();
```

```
        StopProjectVersionResponse response =
rekClient.stopProjectVersion(stopProjectVersionRequest);

        logger.log(Level.INFO, "Status: {0}", response.statusAsString());

        // Get the model version

        int start = findForwardSlash(modelArn, 3) + 1;
        int end = findForwardSlash(modelArn, 4);

        String versionName = modelArn.substring(start, end);

        // wait until model stops

        DescribeProjectVersionsRequest describeProjectVersionsRequest =
DescribeProjectVersionsRequest.builder()
            .projectArn(projectArn).versionNames(versionName).build();

        boolean stopped = false;

        // Wait until create finishes

        do {

            DescribeProjectVersionsResponse describeProjectVersionsResponse
= rekClient

            .describeProjectVersions(describeProjectVersionsRequest);

            for (ProjectVersionDescription projectVersionDescription :
describeProjectVersionsResponse
                .projectVersionDescriptions()) {

                ProjectVersionStatus status =
projectVersionDescription.status();

                logger.log(Level.INFO, "stopping model: {0} ", modelArn);

                switch (status) {

                    case STOPPED:
                        logger.log(Level.INFO, "Model stopped");
                        stopped = true;
```

```
        break;

        case STOPPING:
            Thread.sleep(5000);
            break;

        case FAILED:
            String error = "Model stopping failed: " +
projectVersionDescription.statusAsString() + " "
                + projectVersionDescription.statusMessage() + "
" + modelArn;

            logger.log(Level.SEVERE, error);
            throw new Exception(error);

        default:
            String unexpectedError = "Unexpected stopping state: "
                + projectVersionDescription.statusAsString() + "
"
                + projectVersionDescription.statusMessage() + "
" + modelArn;

            logger.log(Level.SEVERE, unexpectedError);
            throw new Exception(unexpectedError);
    }
}

} while (stopped == false);

} catch (RekognitionException e) {
    logger.log(Level.SEVERE, "Could not stop model: {0}",
e.getMessage());
    throw e;
}

}

public static void main(String[] args) {

    String modelArn = null;
    String projectArn = null;

    final String USAGE = "\n" + "Usage: " + "<project_name> <version_name>\n
\n" + "Where:\n"
```

```
        + "    project_arn - The ARN of the project that contains the
model that you want to stop. \n\n"
        + "    model_arn - The ARN of the model version that you want to
stop.\n\n";

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

    projectArn = args[0];
    modelArn = args[1];

    try {

        // Get the Rekognition client.
        RekognitionClient rekClient = RekognitionClient.builder()
            .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
            .region(Region.US_WEST_2)
            .build();

        // Stop model
        stopMyModel(rekClient, projectArn, modelArn);

        System.out.println(String.format("Model stopped: %s", modelArn));

        rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    } catch (Exception rekError) {
        logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());
        System.exit(1);
    }

}

}
```

## 実行時間と使用された推論単位のレポート

2022 年 8 月以降にモデルをトレーニングして開始した場合、InServiceInferenceUnitsAmazon CloudWatch メトリクスを使用して、モデルが実行された時間とその時間内に使用された[推論単位](#)の数を判断できます。

### Note

AWS リージョンにモデルが 1 つしかない場合は、StartProjectVersion と StopProjectVersion への正常な呼び出しを追跡することで、モデルの実行時間を取得することもできます CloudWatch。このアプローチは、メトリクスにモデルに関する情報が含まれていないため、AWS リージョンで複数のモデルを実行した場合は使用できません。または、AWS CloudTrail を使用して StartProjectVersion および StopProjectVersion ([イベント履歴](#) の requestParameters フィールドにモデル ARN を含む) への呼び出しを追跡することもできます。CloudTrail イベントは 90 日に制限されますが、最大 7 年間 [CloudTrail Lake](#) に保存できます。

以下の手順では、以下のグラフを作成します。

- モデルが実行された時間数。
- モデルが使用した推論単位の数。

過去 15 か月までの期間を選択できます。メトリクスの保持の詳細については、「[メトリクスの保持](#)」を参照してください。

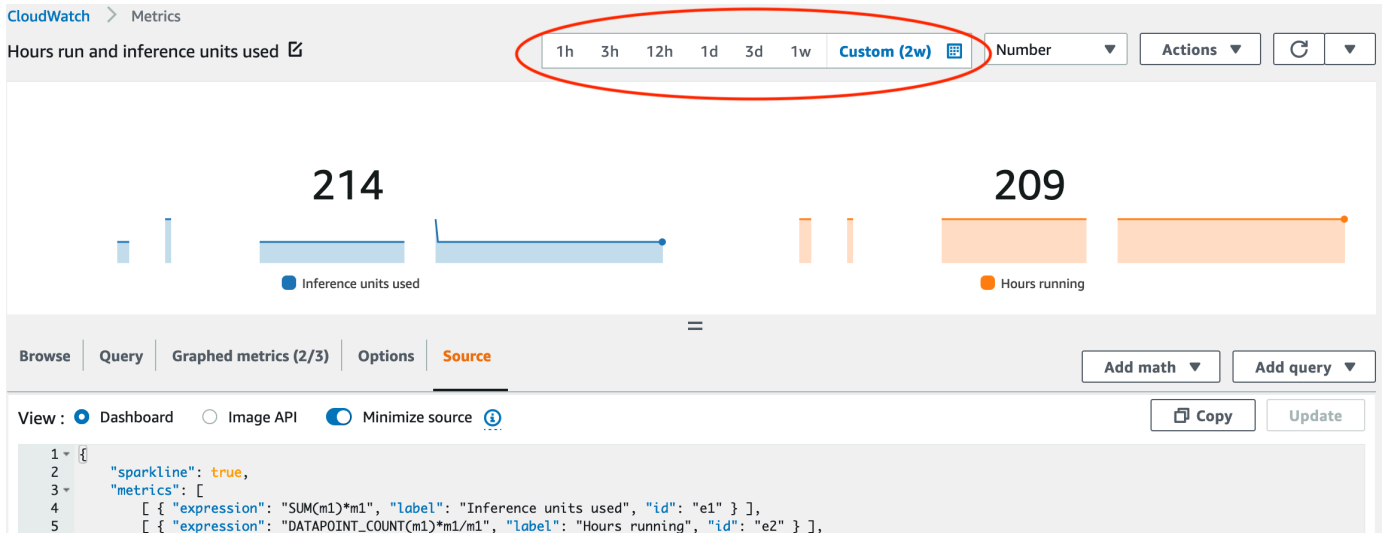
モデル期間とモデルに使用される推論単位を決定するには

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. ナビゲーションペインで、[メトリクス] から [すべてのメトリクス] を選択します。
3. コンテンツペインで、[ソース] タブを選択します。
4. [ダッシュボード] ボタンが選択されていることを確認します。
5. エディタボックス内で既存の JSON を以下の JSON と置き換えます。以下の値を変更します:
  - Project\_Name - グラフにするモデルが含まれているプロジェクト。
  - Version\_Name - グラフにするモデルのバージョン。

- **AWS\_Region** — モデルを含む AWS リージョン。ページ上部のナビゲーションバーの AWS リージョンセレクターをチェックして、CloudWatch コンソールが同じリージョンにあることを確認します。必要に応じて更新します。

```
{
  "sparkline": true,
  "metrics": [
    [
      {
        "expression": "SUM(m1)*m1",
        "label": "Inference units used",
        "id": "e1"
      }
    ],
    [
      {
        "expression": "DATAPoint_COUNT(m1)*m1/m1",
        "label": "Hours running",
        "id": "e2"
      }
    ],
    [
      "AWS/Rekognition",
      "InServiceInferenceUnits",
      "ProjectName",
      "Project_Name",
      "VersionName",
      "Version_Name",
      {
        "id": "m1",
        "visible": false
      }
    ]
  ],
  "view": "singleValue",
  "stacked": false,
  "region": "AWS_Region",
  "stat": "Average",
  "period": 3600,
  "title": "Hours run and inference units used"
}
```

6. [更新] を選択します。
7. ページの上部で、[タイムライン] を選択します。タイムライン中に使用した推論単位と実行時間の数値が表示されるはずですが、グラフのギャップは、モデルが実行されていなかった時間を示しています。以下のコンソールのスクリーンショットは、使用された推論単位と時間、2 週間のカスタム時間が設定されており、最大値は 214 推論単位、209 時間は実行されています。



8. (オプション) グラフをダッシュボードに追加するには、[アクション]、[ダッシュボードに追加 - 改善] の順に選択します。

## トレーニングされたモデルによるイメージの分析

トレーニング済みの Amazon Rekognition Custom Labels モデルを使用してイメージを分析するには、Labels API [DetectCustom](#) を呼び出します。DetectCustomLabels からの結果は、イメージに特定のオブジェクト、シーン、または概念が含まれているという予測になります。

DetectCustomLabels を呼び出すには、以下を指定します。

- 使用する Amazon Rekognition Custom Labels モデルの Amazon リソースネーム (ARN)。
- モデルで予測を行う際に使用するイメージ。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。詳細については、「[イメージ](#)」を参照してください。

カスタムラベルは [Custom Label](#) オブジェクトの配列で返されます。各カスタムラベルは、イメージ内の 1 つのオブジェクト、シーン、または概念を表します。カスタムラベルには以下が含まれます。

- イメージ内のオブジェクト、シーン、または概念のラベル。
- イメージ内のオブジェクトの境界。境界ボックスの座標は、ソースイメージ上のオブジェク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、「」を参照してください [BoundingBox](#)。は、モデルがオブジェクトの位置を検出するようにトレーニングされている場合にのみ境界ボックス DetectCustomLabels を返します。
- Amazon Rekognition Custom Labels がラベルと境界ボックスの精度を示す信頼度。

検出の信頼度に基づいてラベルをフィルタリングするには、必要とされる信頼度と一致する MinConfidence の値を指定します。例えば、予測に高い信頼度を持たせる必要がある場合は、MinConfidence に高い値を指定します。信頼度に関係なくすべてのラベルを取得するには、MinConfidence の値を 0 に指定します。

モデルのパフォーマンスは、モデルトレーニング中に計算されたリコールと精度のメトリクスによって部分的に測定されます。詳細については、「[モデルを評価するためのメトリクス](#)」を参照してください。

モデルの精度を上げるには、MinConfidence に高い値を設定します。詳細については、「[偽陽性の削減 \(適合率の向上\)](#)」を参照してください。



モデルのリコールを高めるには、MinConfidence に低い値を使用してください。詳細については、「[偽陰性の削減 \(再現率の向上\)](#)」を参照してください。

MinConfidence の値を指定しない場合、Amazon Rekognition Custom Labels はラベルの想定しきい値に基づいてそのラベルを返します。詳細については、「[想定しきい値](#)」を参照してください。ラベルの想定しきい値は、モデルのトレーニング結果から取得できます。詳細については、「[モデルのトレーニング \(コンソール\)](#)」を参照してください。

MinConfidence の入力パラメータを使用すると、呼び出しに必要なしきい値を指定できません。MinConfidence の値を下回る信頼度で検出されたラベルは、レスポンスでは返されません。また、ラベルの想定しきい値は、そのラベルがレスポンスに含まれるかどうかに影響しません。

#### Note

Amazon Rekognition Custom Labels メトリクスは、想定されるしきい値を 0~1 の浮動小数点値で表します。MinConfidence の範囲は、しきい値をパーセンテージ値 (0~100) に正規化します。からの信頼レスポンス DetectCustomLabels もパーセンテージとして返されます。

特定のラベルのしきい値を指定することもできます。例えば、精度メトリクスがラベル A では許容されてラベル B では許容されない場合、別のしきい値 (MinConfidence) を指定するときは次の点を考慮してください。

- 1 つのラベル (A) だけを対象とする場合は、MinConfidence の値を目的のしきい値に設定します。レスポンスでは、信頼度が MinConfidence よりも大きい場合にのみ、ラベル A の予測が (他のラベルと共に) 返されます。返された他のラベルはすべて除外する必要があります。
- 複数のラベルに異なるしきい値を適用する場合は、以下に従ってください。
  1. MinConfidence の値を 0 にします。値が 0 の場合、検出の信頼度に関係なく、すべてのラベルが返されます。
  2. 返されるラベルごとに、ラベルの信頼度がラベルで必要なしきい値よりも大きいことを確認して、必要なしきい値を適用します。

詳細については、「[トレーニング済み Amazon Rekognition Custom Labels の改善](#)」を参照してください。

DetectCustomLabels で返される信頼値が低すぎると感じる場合は、モデルの再トレーニングを検討してください。詳細については、「[Amazon Rekognition Custom Labels モデルをト](#)

[レーニングする](#)」を参照してください。MaxResults 入力パラメータを指定することによって、DetectCustomLabels から返されるカスタムラベルの数を制限できます。結果は、信頼度が最も高いものから順に、最も低いものまで返されます。

DetectCustomLabels を呼び出す他の例については、「[例](#)」を参照してください。

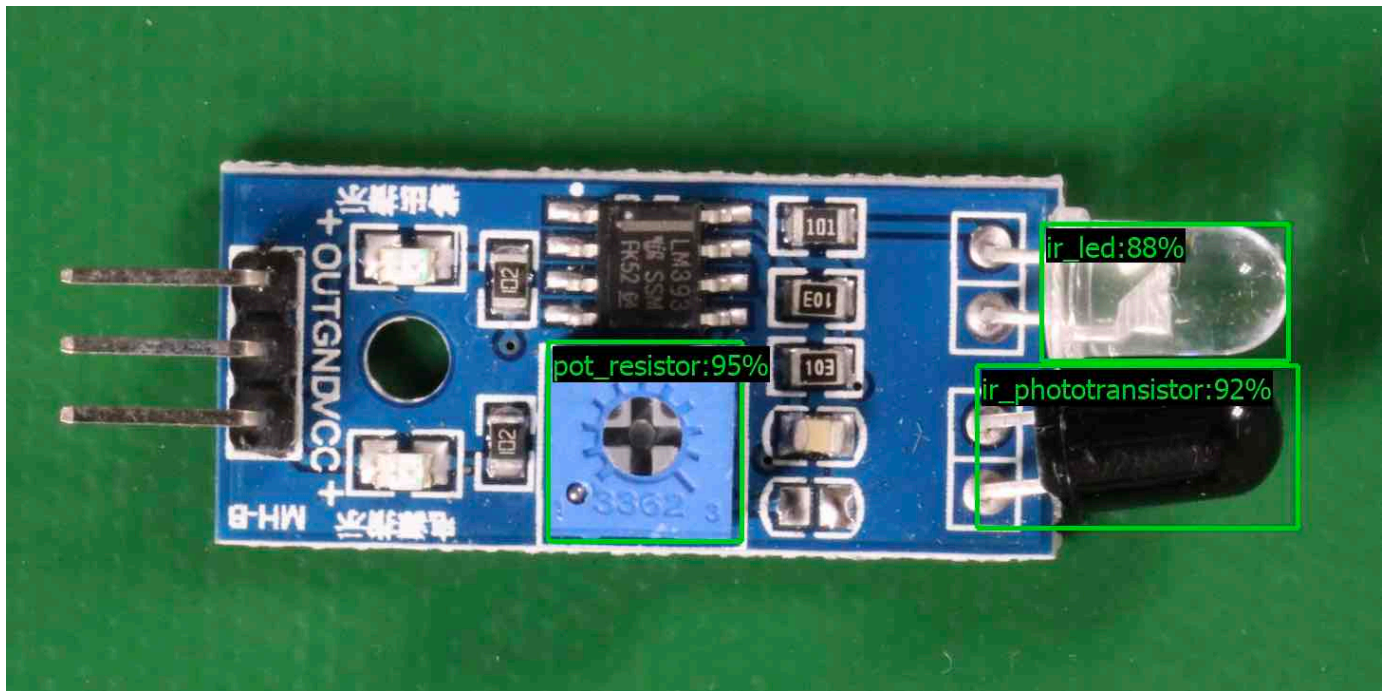
DetectCustomLabels の保護については、「[DetectCustomLabels の保護](#)」を参照してください。

カスタムラベル (API) を検出するには

1. まだ実行していない場合:
  - a. DetectCustomLabels および AmazonS3ReadOnlyAccess のアクセス許可があることを確認します。詳細については、「[SDK アクセス許可の設定](#)」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 4: AWS CLI と AWS SDKsを設定する](#)」を参照してください。
2. モデルをトレーニングしてデプロイします。詳細については、「[Amazon Rekognition Custom Labels モデルの作成](#)」を参照してください。
3. DetectCustomLabels を呼び出すユーザーがステップ 2 で使用されたモデルにアクセスできることを確認します。詳細については、「[DetectCustomLabels の保護](#)」を参照してください。
4. 分析するイメージを 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。Python、Java、Java 2 の例には、RAW バイトを使用してイメージを渡すための、ローカルのイメージファイルの使用方法も示されています。ファイルは 4 MB よりも小さくなければなりません。

5. 以下の例を使用して、DetectCustomLabels オペレーションを呼び出します。Python と Java の例では、次のイメージのように、解析結果を重ね合わせたイメージを表示します。次の画像には、境界線のボックスとラベルが回路基板に含まれており、境界線の境界線、境界線の光変換器、LED コンポーネントが含まれています。



## AWS CLI

この AWS CLI コマンドは、CLI オペレーションの JSON DetectCustomLabels 出力を表示します。次の入力パラメータの値を変更します。

- bucket を、ステップ 4 で使用した Amazon S3 バケットの名前に。
- image を、ステップ 4 でアップロードした入力イメージファイルの名前に。
- projectVersionArn を、使用するモデルの ARN に。

```
aws rekognition detect-custom-labels --project-version-arn model_arn \  
--image '{"S3Object":{"Bucket":"bucket","Name":"image"}}' \  
--min-confidence 70 \  
--profile custom-labels-access
```

## Python

次のコード例では、イメージ内の境界ボックスとイメージレベルラベルを表示します。

ローカルイメージを分析するには、次のコマンドライン引数を指定します。

- イメージの分析に使用するモデルの ARN。
- ローカルイメージファイルの名前と場所。

Amazon S3 バケットに保存されているイメージを分析するには、プログラムを実行し、次のコマンドライン引数を指定します。

- イメージの分析に使用するモデルの ARN。
- ステップ 4 で使用した Amazon S3 バケット内のイメージの名前と場所。
- `--bucket #####` - ステップ 4 で使用した Amazon S3 バケット。

この例では、Pillow のバージョンが `>= 8.0.0` であることを前提としていることに注意してください。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Purpose
Amazon Rekognition Custom Labels detection example used in the service
documentation:
https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/detecting-custom-
labels.html
Shows how to detect custom labels by using an Amazon Rekognition Custom Labels
model.
The image can be stored on your local computer or in an Amazon S3 bucket.
"""

import io
import logging
import argparse
import boto3
from PIL import Image, ImageDraw, ImageFont

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def analyze_local_image(rek_client, model, photo, min_confidence):
    """
    Analyzes an image stored as a local file.
    :param rek_client: The Amazon Rekognition Boto3 client.
    :param s3_connection: The Amazon S3 Boto3 S3 connection object.
```

```
    :param model: The ARN of the Amazon Rekognition Custom Labels model that you
want to use.
    :param photo: The name and file path of the photo that you want to analyze.
    :param min_confidence: The desired threshold/confidence for the call.
    """

    try:
        logger.info("Analyzing local file: %s", photo)
        image = Image.open(photo)
        image_type = Image.MIME[image.format]

        if (image_type == "image/jpeg" or image_type == "image/png") is False:
            logger.error("Invalid image type for %s", photo)
            raise ValueError(
                f"Invalid file format. Supply a jpeg or png format file:
{photo}"
            )

        # get images bytes for call to detect_anomalies
        image_bytes = io.BytesIO()
        image.save(image_bytes, format=image.format)
        image_bytes = image_bytes.getvalue()

        response = rek_client.detect_custom_labels(Image={'Bytes': image_bytes},
                                                    MinConfidence=min_confidence,
                                                    ProjectVersionArn=model)

        show_image(image, response)
        return len(response['CustomLabels'])

    except ClientError as client_err:
        logger.error(format(client_err))
        raise
    except FileNotFoundError as file_error:
        logger.error(format(file_error))
        raise

def analyze_s3_image(rek_client, s3_connection, model, bucket, photo,
                    min_confidence):
    """
    Analyzes an image stored in the specified S3 bucket.
    :param rek_client: The Amazon Rekognition Boto3 client.
    :param s3_connection: The Amazon S3 Boto3 S3 connection object.
```

```
    :param model: The ARN of the Amazon Rekognition Custom Labels model that you
    want to use.
    :param bucket: The name of the S3 bucket that contains the image that you
    want to analyze.
    :param photo: The name of the photo that you want to analyze.
    :param min_confidence: The desired threshold/confidence for the call.
    """

    try:
        # Get image from S3 bucket.

        logger.info("analyzing bucket: %s image: %s", bucket, photo)
        s3_object = s3_connection.Object(bucket, photo)
        s3_response = s3_object.get()

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

        image_type = Image.MIME[image.format]

        if (image_type == "image/jpeg" or image_type == "image/png") is False:
            logger.error("Invalid image type for %s", photo)
            raise ValueError(
                f"Invalid file format. Supply a jpeg or png format file:
{photo}")

        ImageDraw.Draw(image)

        # Call DetectCustomLabels.
        response = rek_client.detect_custom_labels(
            Image={'S3Object': {'Bucket': bucket, 'Name': photo}},
            MinConfidence=min_confidence,
            ProjectVersionArn=model)

        show_image(image, response)
        return len(response['CustomLabels'])

    except ClientError as err:
        logger.error(format(err))
        raise

def show_image(image, response):
    """
```

```
Displays the analyzed image and overlays analysis results
:param image: The analyzed image
:param response: the response from DetectCustomLabels
"""
try:
    font_size = 40
    line_width = 5

    img_width, img_height = image.size
    draw = ImageDraw.Draw(image)

    # Calculate and display bounding boxes for each detected custom label.
    image_level_label_height = 0

    for custom_label in response['CustomLabels']:
        confidence = int(round(custom_label['Confidence'], 0))
        label_text = f"{custom_label['Name']}:{confidence}%"
        fnt = ImageFont.truetype('Tahoma.ttf', font_size)
        text_left, text_top, text_right, text_bottom = draw.textbbox((0, 0),
label_text, fnt)
        text_width, text_height = text_right - text_left, text_bottom -
text_top

        logger.info("Label: %s", custom_label['Name'])
        logger.info("Confidence: %s", confidence)

        # Draw bounding boxes, if present
        if 'Geometry' in custom_label:
            box = custom_label['Geometry']['BoundingBox']
            left = img_width * box['Left']
            top = img_height * box['Top']
            width = img_width * box['Width']
            height = img_height * box['Height']

            logger.info("Bounding box")
            logger.info("\tLeft: {0:.0f}".format(left))
            logger.info("\tTop: {0:.0f}".format(top))
            logger.info("\tLabel Width: {0:.0f}".format(width))
            logger.info("\tLabel Height: {0:.0f}".format(height))

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

```
        (left, top + height),
        (left, top))
    # Draw bounding box and label text
    draw.line(points, fill="limegreen", width=line_width)
    draw.rectangle([(left + line_width, top+line_width),
                    (left + text_width + line_width, top +
line_width + text_height)], fill="black")
    draw.text((left + line_width, top + line_width),
              label_text, fill="limegreen", font=fnt)

    # draw image-level label text.
    else:
        draw.rectangle([(10, image_level_label_height),
                        (text_width + 10, image_level_label_height
+text_height)], fill="black")
        draw.text((10, image_level_label_height),
                  label_text, fill="limegreen", font=fnt)

        image_level_label_height += text_height

    image.show()

except Exception as err:
    logger.error(format(err))
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "model_arn", help="The ARN of the model that you want to use."
    )

    parser.add_argument(
        "image", help="The path and file name of the image that you want to
analyze"
    )

    parser.add_argument(
        "--bucket", help="The bucket that contains the image. If not supplied,
image is assumed to be a local file.", required=False
```



```
)

def main():

    try:
        logging.basicConfig(level=logging.INFO,
                            format="%(levelname)s: %(message)s")

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        label_count = 0
        min_confidence = 50

        session = boto3.Session(profile_name='custom-labels-access')
        rekognition_client = session.client("rekognition")

        if args.bucket is None:
            # Analyze local image.
            label_count = analyze_local_image(rekognition_client,
                                              args.model_arn,
                                              args.image,
                                              min_confidence)
        else:
            # Analyze image in S3 bucket.
            s3_connection = session.resource('s3')
            label_count = analyze_s3_image(rekognition_client,
                                          s3_connection,
                                          args.model_arn,
                                          args.bucket,
                                          args.image,
                                          min_confidence)

        print(f"Custom labels detected: {label_count}")

    except ClientError as client_err:
        print("A service client error occurred: " +
              format(client_err.response["Error"]["Message"]))

    except ValueError as value_err:
        print("A value error occurred: " + format(value_err))
```

```
except FileNotFoundError as file_error:
    print("File not found error: " + format(file_error))

except Exception as err:
    print("An error occurred: " + format(err))

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

## Java

次のコード例では、イメージ内の境界ボックスとイメージレベルラベルを表示します。

ローカルイメージを分析するには、次のコマンドライン引数を指定します。

- イメージの分析に使用するモデルの ARN。
- ローカルイメージファイルの名前と場所。

Amazon S3 バケットに保存されているイメージを分析するには、プログラムを実行し、次のコマンドライン引数を指定します。

- イメージの分析に使用するモデルの ARN。
- ステップ 4 で使用した Amazon S3 バケット内のイメージの名前と場所。
- ステップ 4 で使用したイメージを含む Amazon S3 バケット。

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
*/

package com.amazonaws.samples;

import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.*;
```

```
import java.io.FileNotFoundException;
import java.awt.font.FontRenderContext;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;

import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;

import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.model.CustomLabel;
import com.amazonaws.services.rekognition.model.DetectCustomLabelsRequest;
import com.amazonaws.services.rekognition.model.DetectCustomLabelsResult;
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;

import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.util.IOUtils;

// Calls DetectCustomLabels and displays a bounding box around each detected
// image.
public class DetectCustomLabels extends JPanel {

    private transient DetectCustomLabelsResult response;
    private transient Dimension dimension;
    private transient BufferedImage image;

    public static final Logger logger =
    Logger.getLogger(DetectCustomLabels.class.getName());

    // Finds custom labels in an image stored in an S3 bucket.
    public DetectCustomLabels(AmazonRekognition rekClient,
```

```
        AmazonS3 s3client,
        String projectVersionArn,
        String bucket,
        String key,
        Float minConfidence) throws AmazonRekognitionException,
AmazonS3Exception, IOException {

    logger.log(Level.INFO, "Processing S3 bucket: {0} image {1}", new
Object[] { bucket, key });

    // Get image from S3 bucket and create BufferedImage
    com.amazonaws.services.s3.model.S3Object s3object =
s3client.getObject(bucket, key);
    S3ObjectInputStream inputStream = s3object.getObjectContent();
    image = ImageIO.read(inputStream);

    // Set image size
    setWindowDimensions();

    DetectCustomLabelsRequest request = new DetectCustomLabelsRequest()
        .withProjectVersionArn(projectVersionArn)
        .withImage(new Image().withS3Object(new
S3Object().withName(key).withBucket(bucket)))
        .withMinConfidence(minConfidence);

    // Call DetectCustomLabels

    response = rekClient.detectCustomLabels(request);
    logFoundLabels(response.getCustomLabels());
    drawLabels();

}

// Finds custom label in a local image file.
public DetectCustomLabels(AmazonRekognition rekClient,
    String projectVersionArn,
    String photo,
    Float minConfidence)
    throws IOException, AmazonRekognitionException {

    logger.log(Level.INFO, "Processing local file: {0}", photo);

    // Get image bytes and buffered image
    ByteBuffer imageBytes;
```

```
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);

// Set image size
setWindowDimensions();

// Analyze image
DetectCustomLabelsRequest request = new DetectCustomLabelsRequest()
    .withProjectVersionArn(projectVersionArn)
    .withImage(new Image()
        .withBytes(imageBytes))
    .withMinConfidence(minConfidence);

response = rekClient.detectCustomLabels(request);

logFoundLabels(response.getCustomLabels());

drawLabels();
}

// Log the labels found by DetectCustomLabels
private void logFoundLabels(List<CustomLabel> customLabels) {
    logger.info("Custom labels found");
    if (customLabels.isEmpty()) {
        logger.log(Level.INFO, "No Custom Labels found. Consider lowering
min confidence.");
    } else {
        for (CustomLabel customLabel : customLabels) {
            logger.log(Level.INFO, " Label: {0} Confidence: {1}",
                new Object[] { customLabel.getName(),
customLabel.getConfidence() });
        }
    }
}
```

```
}

// Sets window dimensions to 1/2 screen size, unless image is smaller
public void setWindowDimensions() {
    dimension = java.awt.Toolkit.getDefaultToolkit().getScreenSize();

    dimension.width = (int) dimension.getWidth() / 2;
    if (image.getWidth() < dimension.width) {
        dimension.width = image.getWidth();
    }
    dimension.height = (int) dimension.getHeight() / 2;

    if (image.getHeight() < dimension.height) {
        dimension.height = image.getHeight();
    }

    setPreferredSize(dimension);
}

// Draws the image containing the bounding boxes and labels.
@Override
public void paintComponent(Graphics g) {

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

    // Draw the image.
    g2d.drawImage(image, 0, 0, dimension.width, dimension.height, this);
}

public void drawLabels() {
    // Draws bounding boxes (if present) and label text.

    int boundingBoxBorderWidth = 5;
    int imageHeight = image.getHeight(this);
    int imageWidth = image.getWidth(this);

    // Set up drawing
    Graphics2D g2d = image.createGraphics();
    g2d.setColor(Color.GREEN);
    g2d.setFont(new Font("Tahoma", Font.PLAIN, 50));
    Font font = g2d.getFont();
    FontRenderContext frc = g2d.getFontRenderContext();
```

```
g2d.setStroke(new BasicStroke(boundingBoxBorderWidth));

List<CustomLabel> customLabels = response.getCustomLabels();

int imageLevelLabelHeight = 0;
for (CustomLabel customLabel : customLabels) {

    String label = customLabel.getName();

    int textWidth = (int) (font.getStringBounds(label, frc).getWidth());
    int textHeight = (int) (font.getStringBounds(label,
frc).getHeight());

    // Draw bounding box, if present
    if (customLabel.getGeometry() != null) {

        BoundingBox box = customLabel.getGeometry().getBoundingBox();
        float left = imageWidth * box.getLeft();
        float top = imageHeight * box.getTop();

        // Draw black rectangle
        g2d.setColor(Color.BLACK);
        g2d.fillRect(Math.round(left + (boundingBoxBorderWidth)),
Math.round(top + (boundingBoxBorderWidth)),
                    textWidth + boundingBoxBorderWidth, textHeight +
boundingBoxBorderWidth);

        // Write label onto black rectangle
        g2d.setColor(Color.GREEN);
        g2d.drawString(label, left + boundingBoxBorderWidth, (top +
textHeight));

        // Draw bounding box around label location
        g2d.drawRect(Math.round(left), Math.round(top),
Math.round((imageWidth * box.getWidth())),
                    Math.round((imageHeight * box.getHeight())));
    }
    // Draw image level labels.
    else {
        // Draw black rectangle
        g2d.setColor(Color.BLACK);
        g2d.fillRect(10, 10 + imageLevelLabelHeight, textWidth,
textHeight);
        g2d.setColor(Color.GREEN);
```

```
        g2d.drawString(label, 10, textHeight + imageLevelLabelHeight);

        imageLevelLabelHeight += textHeight;
    }

}

g2d.dispose();

}

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

    String photo = null;
    String bucket = null;
    String projectVersionArn = null;
    float minConfidence = 50;

    final String USAGE = "\n" + "Usage: " + "<model_arn> <image> <bucket>\n"
        + "\n" + "Where:\n"
        + "    model_arn - The ARN of the model that you want to use. \n"
        + "\n"
        + "    image - The location of the image on your local file
        system or within an S3 bucket.\n\n"
        + "    bucket - The S3 bucket that contains the image. Don't
        specify if image is local.\n\n";

    // Collect the arguments. If 3 arguments are present, the image is
    assumed to be
    // in an S3 bucket.

    if (args.length < 2 || args.length > 3) {
        System.out.println(USAGE);
        System.exit(1);
    }

    projectVersionArn = args[0];
    photo = args[1];

    if (args.length == 3) {
        bucket = args[2];
    }

    DetectCustomLabels panel = null;
```



```
try {

    AWSCredentialsProvider provider =new
ProfileCredentialsProvider("custom-labels-access");

    AmazonRekognition rekClient =
AmazonRekognitionClientBuilder.standard()
        .withCredentials(provider)
        .withRegion(Regions.US_WEST_2)
        .build();

    AmazonS3 s3client = AmazonS3ClientBuilder.standard()
        .withCredentials(provider)
        .withRegion(Regions.US_WEST_2)
        .build();

    // Create frame and panel.
    JFrame frame = new JFrame("Custom Labels");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    if (args.length == 2) {
        // Analyze local image
        panel = new DetectCustomLabels(rekClient, projectVersionArn,
photo, minConfidence);
    } else {
        // Analyze image in S3 bucket
        panel = new DetectCustomLabels(rekClient, s3client,
projectVersionArn, bucket, photo, minConfidence);
    }

    frame.setContentPane(panel);
    frame.pack();
    frame.setVisible(true);

} catch (AmazonRekognitionException rekError) {
    String errorMessage = "Rekognition client error: " +
rekError.getMessage();
    logger.log(Level.SEVERE, errorMessage);
    System.out.println(errorMessage);
    System.exit(1);
} catch (FileNotFoundException fileError) {
    String errorMessage = "File not found: " + photo;
    logger.log(Level.SEVERE, errorMessage);
```

```
        System.out.println(errorMessage);
        System.exit(1);
    } catch (IOException fileError) {
        String errorMessage = "Input output exception: " +
fileError.getMessage();
        logger.log(Level.SEVERE, errorMessage);
        System.out.println(errorMessage);
        System.exit(1);
    } catch (AmazonS3Exception s3Error) {
        String errorMessage = "S3 error: " + s3Error.getErrorMessage();
        logger.log(Level.SEVERE, errorMessage);
        System.out.println(errorMessage);
        System.exit(1);
    }
}
}
```

## Java V2

次のコード例では、イメージ内の境界ボックスとイメージレベルラベルを表示します。

ローカルイメージを分析するには、次のコマンドライン引数を指定します。

- projectVersionArn - イメージの分析に使用するモデルの ARN。
- photo - ローカルイメージファイルの名前と場所。

S3 バケットに保存されているイメージを分析するには、プログラムを実行し、次のコマンドライン引数を指定します。

- イメージの分析に使用するモデルの ARN。
- ステップ 4 で使用した S3 バケット内のイメージの名前と場所。
- ステップ 4 で使用したイメージを含む Amazon S3 バケット。

```
/*
    Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
    SPDX-License-Identifier: Apache-2.0
*/

package com.example.rekognition;
```

```
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.core.sync.ResponseTransformer;
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.Image;
import
    software.amazon.awssdk.services.rekognition.model.DetectCustomLabelsRequest;
import
    software.amazon.awssdk.services.rekognition.model.DetectCustomLabelsResponse;
import software.amazon.awssdk.services.rekognition.model.CustomLabel;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;

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.NoSuchBucketException;
import software.amazon.awssdk.services.s3.model.NoSuchKeyException;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;

import java.awt.*;
import java.awt.font.FontRenderContext;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import javax.swing.*;

import java.util.logging.Level;
import java.util.logging.Logger;

// Calls DetectCustomLabels on an image. Displays bounding boxes or
// image level labels found in the image.
public class ShowCustomLabels extends JPanel {

    private transient BufferedImage image;
```

```
private transient DetectCustomLabelsResponse response;
private transient Dimension dimension;
public static final Logger logger =
Logger.getLogger>ShowCustomLabels.class.getName());

// Finds custom labels in an image stored in an S3 bucket.
public ShowCustomLabels(RekognitionClient rekClient,
    S3Client s3client,
    String projectVersionArn,
    String bucket,
    String key,
    Float minConfidence) throws RekognitionException,
NoSuchBucketException, NoSuchKeyException, IOException {

    logger.log(Level.INFO, "Processing S3 bucket: {0} image {1}", new
Object[] { bucket, key });
    // Get image from S3 bucket and create BufferedImage
    GetObjectRequest requestObject =
GetObjectRequest.builder().bucket(bucket).key(key).build();
    ResponseBytes<GetObjectResponse> result =
s3client.getObject(requestObject, ResponseTransformer.toBytes());
    ByteArrayInputStream bis = new
ByteArrayInputStream(result.asByteArray());
    image = ImageIO.read(bis);

    // Set image size
    setWindowDimensions();

    // Construct request parameter for DetectCustomLabels
    S3Object s3object = S3Object.builder().bucket(bucket).name(key).build();

    Image s3Image = Image.builder().s3object(s3object).build();

    DetectCustomLabelsRequest request =
DetectCustomLabelsRequest.builder().image(s3Image)

.projectVersionArn(projectVersionArn).minConfidence(minConfidence).build();

    response = rekClient.detectCustomLabels(request);
    logFoundLabels(response.customLabels());
    drawLabels();
}
```

```
// Finds custom label in a local image file.
public ShowCustomLabels(RekognitionClient rekClient,
    String projectVersionArn,
    String photo,
    Float minConfidence)
    throws IOException, RekognitionException {

    logger.log(Level.INFO, "Processing local file: {0}", photo);
    // Get image bytes and buffered image
    InputStream sourceStream = new FileInputStream(new File(photo));
    SdkBytes imageBytes = SdkBytes.fromInputStream(sourceStream);
    ByteArrayInputStream inputStream = new
ByteArrayInputStream(imageBytes.asByteArray());
    image = ImageIO.read(inputStream);

    setWindowDimensions();

    // Construct request parameter for DetectCustomLabels
    Image localImageBytes = Image.builder().bytes(imageBytes).build();

    DetectCustomLabelsRequest request =
DetectCustomLabelsRequest.builder().image(localImageBytes)
.projectVersionArn(projectVersionArn).minConfidence(minConfidence).build();

    response = rekClient.detectCustomLabels(request);

    logFoundLabels(response.customLabels());
    drawLabels();
}

// Sets window dimensions to 1/2 screen size, unless image is smaller
public void setWindowDimensions() {
    dimension = java.awt.Toolkit.getDefaultToolkit().getScreenSize();

    dimension.width = (int) dimension.getWidth() / 2;
    if (image.getWidth() < dimension.width) {
        dimension.width = image.getWidth();
    }
    dimension.height = (int) dimension.getHeight() / 2;

    if (image.getHeight() < dimension.height) {
        dimension.height = image.getHeight();
    }
}
```

```
    }

    setPreferredSize(dimension);

}

// Draws bounding boxes (if present) and label text.
public void drawLabels() {

    int boundingBoxBorderWidth = 5;
    int imageHeight = image.getHeight(this);
    int imageWidth = image.getWidth(this);

    // Set up drawing
    Graphics2D g2d = image.createGraphics();
    g2d.setColor(Color.GREEN);
    g2d.setFont(new Font("Tahoma", Font.PLAIN, 50));
    Font font = g2d.getFont();
    FontRenderContext frc = g2d.getFontRenderContext();
    g2d.setStroke(new BasicStroke(boundingBoxBorderWidth));

    List<CustomLabel> customLabels = response.customLabels();

    int imageLevelLabelHeight = 0;
    for (CustomLabel customLabel : customLabels) {

        String label = customLabel.name();

        int textWidth = (int) (font.getStringBounds(label, frc).getWidth());
        int textHeight = (int) (font.getStringBounds(label,
frc).getHeight());

        // Draw bounding box, if present
        if (customLabel.geometry() != null) {

            BoundingBox box = customLabel.geometry().boundingBox();
            float left = imageWidth * box.left();
            float top = imageHeight * box.top();

            // Draw black rectangle
            g2d.setColor(Color.BLACK);
            g2d.fillRect(Math.round(left + (boundingBoxBorderWidth)),
Math.round(top + (boundingBoxBorderWidth)),
```

```
        textWidth + boundingBoxBorderWidth, textHeight +
boundingBoxBorderWidth);

        // Write label onto black rectangle
        g2d.setColor(Color.GREEN);
        g2d.drawString(label, left + boundingBoxBorderWidth, (top +
textHeight));

        // Draw bounding box around label location
        g2d.drawRect(Math.round(left), Math.round(top),
Math.round((imageWidth * box.width())),
        Math.round((imageHeight * box.height())));
    }
    // Draw image level labels.
    else {
        // Draw black rectangle
        g2d.setColor(Color.BLACK);
        g2d.fillRect(10, 10 + imageLevelLabelHeight, textWidth,
textHeight);

        g2d.setColor(Color.GREEN);
        g2d.drawString(label, 10, textHeight + imageLevelLabelHeight);

        imageLevelLabelHeight += textHeight;
    }

}
g2d.dispose();

}

// Log the labels found by DetectCustomLabels
private void logFoundLabels(List<CustomLabel> customLabels) {
    logger.info("Custom labels found:");
    if (customLabels.isEmpty()) {
        logger.log(Level.INFO, "No Custom Labels found. Consider lowering
min confidence.");
    }
    else {
        for (CustomLabel customLabel : customLabels) {
            logger.log(Level.INFO, " Label: {0} Confidence: {1}",
                new Object[] { customLabel.name(),
customLabel.confidence() } );
        }
    }
}
```

```
    }  
  }  
  
  // Draws the image containing the bounding boxes and labels.  
  @Override  
  public void paintComponent(Graphics g) {  
  
    Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.  
  
    // Draw the image.  
    g2d.drawImage(image, 0, 0, dimension.width, dimension.height, this);  
  
  }  
  
  public static void main(String args[]) throws Exception {  
  
    String photo = null;  
    String bucket = null;  
    String projectVersionArn = null;  
  
    final String USAGE = "\n" + "Usage: " + "<model_arn> <image> <bucket>\n  
\n" + "Where:\n"  
        + "  model_arn - The ARN of the model that you want to use. \n  
\n"  
        + "  image - The location of the image on your local file  
system or within an S3 bucket.\n\n"  
        + "  bucket - The S3 bucket that contains the image. Don't  
specify if image is local.\n\n";  
  
    // Collect the arguments. If 3 arguments are present, the image is  
assumed to be  
    // in an S3 bucket.  
  
    if (args.length < 2 || args.length > 3) {  
      System.out.println(USAGE);  
      System.exit(1);  
    }  
  
    projectVersionArn = args[0];  
    photo = args[1];  
  
    if (args.length == 3) {  
      bucket = args[2];  
    }  
  }  
}
```



```
float minConfidence = 50;

ShowCustomLabels panel = null;

try {
    // Get the Rekognition client

    // Get the Rekognition client.
    RekognitionClient rekClient = RekognitionClient.builder()
        .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
        .region(Region.US_WEST_2)
        .build();

    S3Client s3Client = S3Client.builder()
        .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
        .region(Region.US_WEST_2)
        .build();

    // Create frame and panel.
    JFrame frame = new JFrame("Custom Labels");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    if (args.length == 2) {
        // Analyze local image
        panel = new ShowCustomLabels(rekClient, projectVersionArn,
photo, minConfidence);
    } else {
        // Analyze image in S3 bucket
        panel = new ShowCustomLabels(rekClient, s3Client,
projectVersionArn, bucket, photo, minConfidence);
    }

    frame.setContentPane(panel);
    frame.pack();
    frame.setVisible(true);

} catch (RekognitionException rekError) {
```

```
        String errorMessage = "Rekognition client error: " +
rekError.getMessage();
        logger.log(Level.SEVERE, errorMessage);
        System.out.println(errorMessage);
        System.exit(1);
    } catch (FileNotFoundException fileError) {
        String errorMessage = "File not found: " + photo;
        logger.log(Level.SEVERE, errorMessage);
        System.out.println(errorMessage);
        System.exit(1);
    } catch (IOException fileError) {
        String errorMessage = "Input output exception: " +
fileError.getMessage();
        logger.log(Level.SEVERE, errorMessage);
        System.out.println(errorMessage);
        System.exit(1);
    } catch (NoSuchKeyException bucketError) {
        String errorMessage = String.format("Image not found: %s in bucket
%s.", photo, bucket);
        logger.log(Level.SEVERE, errorMessage);
        System.out.println(errorMessage);
        System.exit(1);
    } catch (NoSuchBucketException bucketError) {
        String errorMessage = "Bucket not found: " + bucket;
        logger.log(Level.SEVERE, errorMessage);
        System.out.println(errorMessage);
        System.exit(1);
    }
}
}
```

## DetectCustomLabels オペレーションリクエスト

DetectCustomLabels 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。

```
{
  "ProjectVersionArn": "string",
  "Image": {
    "S3object": {
```

```
        "Bucket": "string",
        "Name": "string",
        "Version": "string"
    }
},
"MinConfidence": 90,
"MaxLabels": 10,
}
```

## DetectCustomLabels オペレーションレスポンス

次の DetectCustomLabels 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カスタムラベルを示しています。

```
{
  "CustomLabels": [
    {
      "Name": "MyLogo",
      "Confidence": 77.7729721069336,
      "Geometry": {
        "BoundingBox": {
          "Width": 0.198987677693367,
          "Height": 0.31296101212501526,
          "Left": 0.07924537360668182,
          "Top": 0.4037395715713501
        }
      }
    }
  ]
}
```

# Amazon Rekognition Custom Labels リソースの管理

このセクションでは、Amazon Rekognition Custom Labels モデルのトレーニングと使用に使用するワークフローの概要を説明します。また、AWS SDK を使用してモデルをトレーニングし、モデルを使用するための概要情報も含まれています。

## Amazon Rekognition Custom Labels プロジェクトの管理

Amazon Rekognition Custom Labels 内では、プロジェクトを使用することにより、特定のユースケースに作成したモデルを管理します。プロジェクトでは、データセット、モデルトレーニング、モデルバージョン、モデル評価、およびプロジェクトのモデルの実行を管理します。

### トピック

- [Amazon Rekognition Custom Labels プロジェクトの削除](#)
- [プロジェクトの記述 \(SDK\)](#)
- [AWS CloudFormationを使用したプロジェクトの作成](#)

## Amazon Rekognition Custom Labels プロジェクトの削除

プロジェクトを削除するには、Amazon Rekognition コンソールを使用するか、[DeleteProject](#) API を呼び出します。プロジェクトを削除するには、最初に関連付けられたモデルをすべて削除する必要があります。削除したプロジェクトを元に戻すことはできません。

### トピック

- [Amazon Rekognition Custom Labels プロジェクトの削除 \(コンソール\)](#)
- [Amazon Rekognition Custom Labels プロジェクトの削除 \(SDK\)](#)

## Amazon Rekognition Custom Labels プロジェクトの削除 (コンソール)

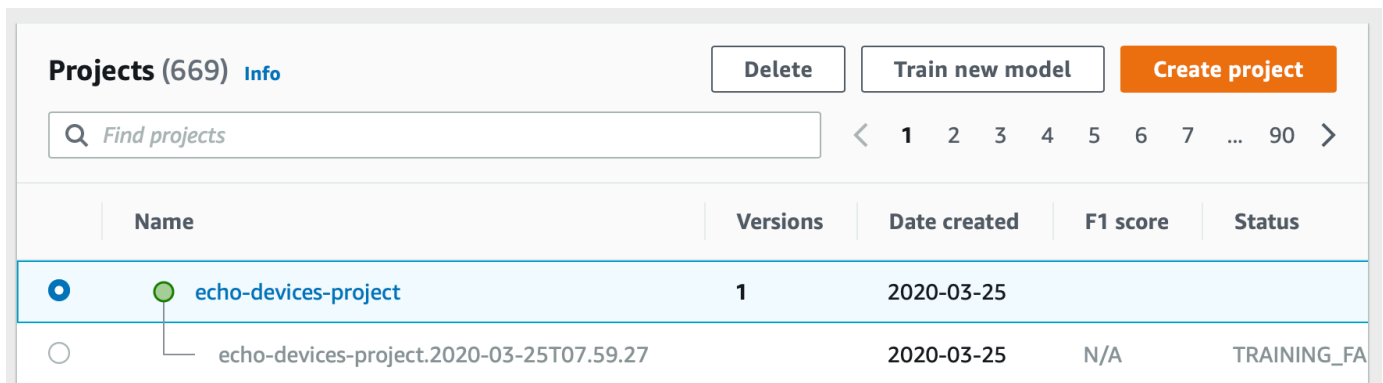
プロジェクトページから、またはプロジェクトの詳細ページからプロジェクトを削除できます。次の手順では、コンソールを使ってプロジェクトを作成する方法を示しています。

Amazon Rekognition Custom Labels コンソールは、プロジェクトの削除中に関連付けられたモデルとデータセットを削除します。プロジェクトが実行中またはトレーニング中の場合は、プロジェクトを削除できません。実行中のモデルを停止するには、「[Amazon Rekognition Custom Labels モデル](#)

[\(SDK\) の停止](#)」を参照してください。プロジェクトがトレーニング中の場合は、終了してからプロジェクトを削除してください。

プロジェクトを削除するには (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. [カスタムラベルを使用] を選択します。
3. [開始方法] を選択します。
4. 左側のナビゲーションペインで、[プロジェクト] を選択します。
5. [プロジェクト] ページで、削除するプロジェクトのラジオボタンを選択します。2020-03-25 に作成されたバージョンが 1 つで、削除 echo-devices-project、新しいモデルのトレーニング、またはプロジェクトの作成のオプションを含む を示すプロジェクトリスト。



The screenshot shows the 'Projects (669)' page in the Amazon Rekognition console. At the top, there are buttons for 'Delete', 'Train new model', and 'Create project'. Below these is a search bar labeled 'Find projects' and a pagination control showing '1 2 3 4 5 6 7 ... 90 >'. The main content is a table with columns: Name, Versions, Date created, F1 score, and Status. The first row is highlighted and shows 'echo-devices-project' with 1 version, created on 2020-03-25. The second row shows a sub-project 'echo-devices-project.2020-03-25T07.59.27' with no F1 score and a status of 'TRAINING\_FA'.

Name	Versions	Date created	F1 score	Status
echo-devices-project	1	2020-03-25		
echo-devices-project.2020-03-25T07.59.27		2020-03-25	N/A	TRAINING_FA

6. ページの上部で、[削除] を選択します。[プロジェクトを削除] ダイアログボックスが表示されます。
7. モデルに関連付けられたモデルがない場合:
  - a. [delete] と入力して、プロジェクトを削除します。
  - b. [削除] を選択して、プロジェクトを削除します。
8. モデルに関連付けられたモデルがある場合:
  - a. [delete] と入力して、削除するモデルとモデルを確認します。
  - b. モデルにデータセット、モデル、またはその両方が含まれているかどうかに応じて、[関連付けられたデータセットを削除]、[関連付けられたモデルを削除]、または [関連付けられたデータセットとモデルを削除] を選択します。モデルの削除が完了するまでに時間がかかる場合があります。

**Note**

コンソールでは、トレーニング中または実行中のモデルを削除することはできません。一覧表示されている実行中のモデルを停止してから再試行するか、トレーニングが終了するまで待ちます。

モデルの削除中にダイアログボックスを閉じた場合でも、モデルは削除されます。後で、この手順を繰り返すことにより、プロジェクトを削除できます。

モデルを削除するためのパネルには、関連するモデルを削除する明示的な手順が表示されません。

### Delete project

✕

Are you sure you want to delete:  
**echo-devices-project** ?

All models in the project must be deleted before the project can be deleted. You cannot delete models which are running or being trained. [Learn more](#)

---

**Delete models**

To delete this project, all of its models must be deleted. Model deletion can take up to 5 minutes.

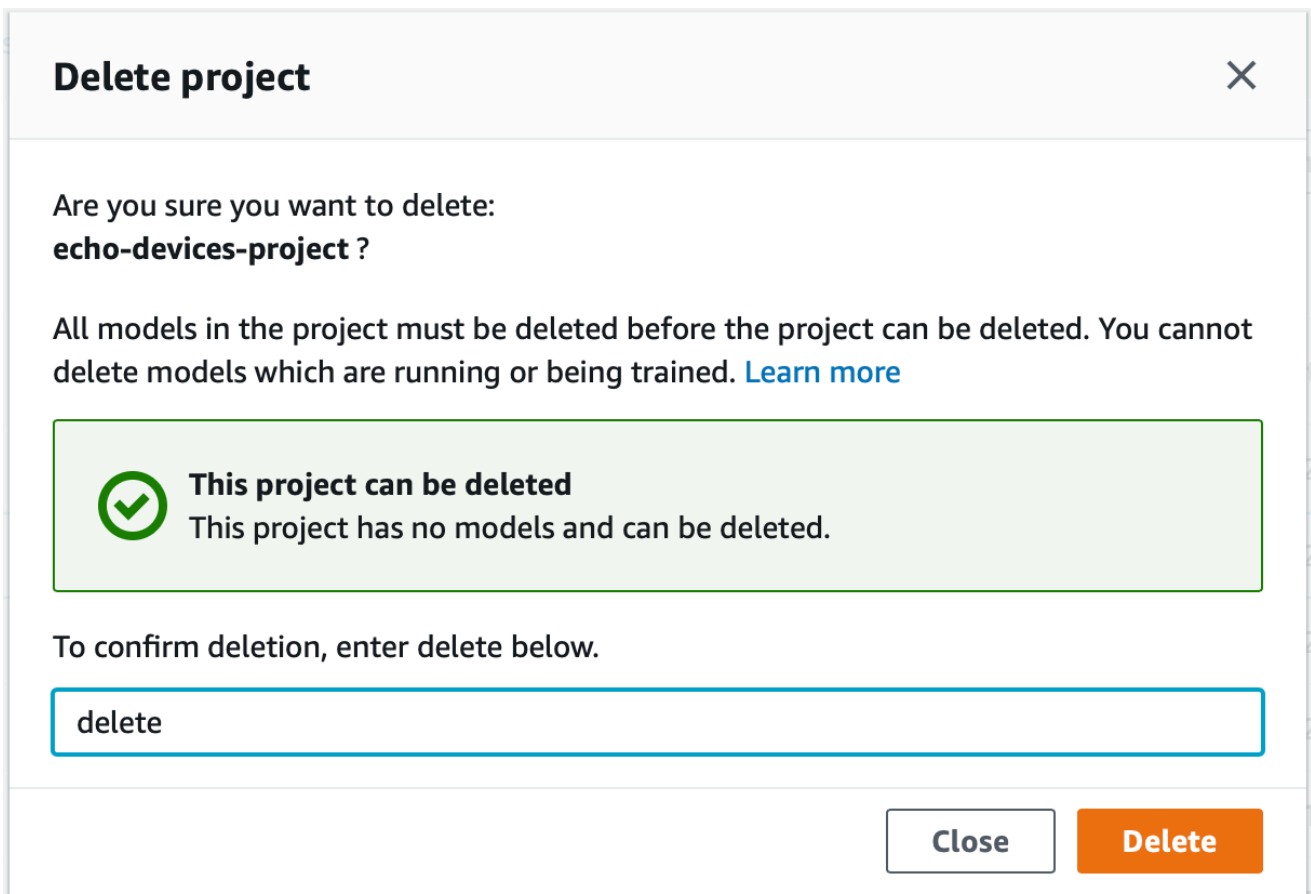
**echo-devices-project.2020-03-30T09.28.17**
**TRAINING\_COMPLETED**

To confirm deletion, enter delete below.

Close

Delete associated models

- c. [delete] と入力して、削除するプロジェクトを確認します。
- d. [削除] を選択して、プロジェクトを削除します。



## Amazon Rekognition Custom Labels プロジェクトの削除 (SDK)

Amazon Rekognition Custom Labels プロジェクトを削除するには、[DeleteProject](#) を呼び出し、削除するプロジェクトの Amazon リソースネーム (ARN) を指定します。AWS アカウント内のプロジェクトの ARNs を取得するには、[DescribeProjects](#) を呼び出します。レスポンスには オブジェクトの配列が含まれます [ProjectDescription](#)。プロジェクト ARN は ProjectArn フィールドです。プロジェクト名を使用すると、プロジェクトの ARN を識別できます。例えば `arn:aws:rekognition:us-east-1:123456789010:project/project name/1234567890123` です。

プロジェクトを削除するには、まず、プロジェクト内のすべてのモデルとデータセットを削除する必要があります。詳細については、「[Amazon Rekognition Custom Labels モデルの削除 \(SDK\)](#)」および「[データセットの削除](#)」を参照してください。

プロジェクトの削除にはしばらくかかることがあります。その間は、プロジェクトのステータスは DELETING です。への後続の呼び出しに、削除したプロジェクトが含まれ [DescribeProjects](#) していない場合、プロジェクトは削除されます。

プロジェクトを削除するには (SDK)

1. 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 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. プロジェクトを削除するには、次のコードを使用します。

AWS CLI

project-arn の値を削除するプロジェクトの名前に変更します。

```
aws rekognition delete-project --project-arn project_arn \  
--profile custom-labels-access
```

Python

次のコードを使用します。次のコマンドラインパラメータを指定します。

- project\_arn - 削除するプロジェクトの ARN。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
  
"""  
Purpose  
Amazon Rekognition Custom Labels project example used in the service  
documentation:  
https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/mp-delete-  
project.html  
Shows how to delete an existing Amazon Rekognition Custom Labels project.  
You must first delete any models and datasets that belong to the project.  
""">  
  
import argparse  
import logging  
import time
```



```
import boto3

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def find_forward_slash(input_string, n):
    """
    Returns the location of '/' after n number of occurrences.
    :param input_string: The string you want to search
    : n: the occurrence that you want to find.
    """
    position = input_string.find('/')
    while position >= 0 and n > 1:
        position = input_string.find('/', position + 1)
        n -= 1
    return position

def delete_project(rek_client, project_arn):
    """
    Deletes an Amazon Rekognition Custom Labels project.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_arn: The ARN of the project that you want to delete.
    """

    try:
        # Delete the project
        logger.info("Deleting project: %s", project_arn)

        response = rek_client.delete_project(ProjectArn=project_arn)

        logger.info("project status: %s", response['Status'])

        deleted = False

        logger.info("waiting for project deletion: %s", project_arn)

        # Get the project name
        start = find_forward_slash(project_arn, 1) + 1
        end = find_forward_slash(project_arn, 2)
        project_name = project_arn[start:end]
```

```
project_names = [project_name]

while deleted is False:

    project_descriptions = rek_client.describe_projects(
        ProjectNames=project_names)['ProjectDescriptions']

    if len(project_descriptions) == 0:
        deleted = True

    else:
        time.sleep(5)

logger.info("project deleted: %s",project_arn)

return True

except ClientError as err:
    logger.exception(
        "Couldn't delete project - %s: %s",
        project_arn, err.response['Error']['Message'])
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The ARN of the project that you want to delete."
    )

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # get command line arguments
```

```
parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
add_arguments(parser)
args = parser.parse_args()

print(f"Deleting project: {args.project_arn}")

# Delete the project.
session = boto3.Session(profile_name='custom-labels-access')
rekognition_client = session.client("rekognition")

delete_project(rekognition_client,
               args.project_arn)

print(f"Finished deleting project: {args.project_arn}")

except ClientError as err:
    error_message = f"Problem deleting project: {err}"
    logger.exception(error_message)
    print(error_message)

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

## Java V2

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `project_arn` - 削除するプロジェクトの ARN。

```
/*
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
SPDX-License-Identifier: Apache-2.0
*/

package com.example.rekognition;

import java.util.List;
import java.util.Objects;
import java.util.logging.Level;
import java.util.logging.Logger;
```

```
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.DeleteProjectRequest;
import software.amazon.awssdk.services.rekognition.model.DeleteProjectResponse;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectsRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectsResponse;
import software.amazon.awssdk.services.rekognition.model.ProjectDescription;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

public class DeleteProject {

    public static final Logger logger =
        Logger.getLogger(DeleteProject.class.getName());

    public static void deleteMyProject(RekognitionClient rekClient, String
        projectArn) throws InterruptedException {

        try {

            logger.log(Level.INFO, "Deleting project: {0}", projectArn);

            // Delete the project

            DeleteProjectRequest deleteProjectRequest =
                DeleteProjectRequest.builder().projectArn(projectArn).build();
            DeleteProjectResponse response =
                rekClient.deleteProject(deleteProjectRequest);

            logger.log(Level.INFO, "Status: {0}", response.status());

            // Wait until deletion finishes

            Boolean deleted = false;

            do {

                DescribeProjectsRequest describeProjectsRequest =
                    DescribeProjectsRequest.builder().build();
                DescribeProjectsResponse describeResponse =
                    rekClient.describeProjects(describeProjectsRequest);
```

```
        List<ProjectDescription> projectDescriptions =
describeResponse.projectDescriptions();

        deleted = true;

        for (ProjectDescription projectDescription :
projectDescriptions) {

            if (Objects.equals(projectDescription.projectArn(),
projectArn)) {

                deleted = false;
                logger.log(Level.INFO, "Not deleted: {0}",
projectDescription.projectArn());
                Thread.sleep(5000);
                break;
            }
        }

        } while (Boolean.FALSE.equals(deleted));

        logger.log(Level.INFO, "Project deleted: {0} ", projectArn);

    } catch (

        RekognitionException e) {
        logger.log(Level.SEVERE, "Client error occurred: {0}",
e.getMessage());
        throw e;
    }

}

public static void main(String[] args) {

    final String USAGE = "\n" + "Usage: " + "<project_arn>\n\n" + "Where:\n"
+ "    project_arn - The ARN of the project that you want to delete.
\n\n";

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

    String projectArn = args[0];
```

```
try {

    RekognitionClient rekClient = RekognitionClient.builder()
        .region(Region.US_WEST_2)
        .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
        .build();

    // Delete the project.
    deleteMyProject(rekClient, projectArn);

    System.out.println(String.format("Project deleted: %s",
projectArn));

    rekClient.close();

} catch (RekognitionException rekError) {
    logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
    System.exit(1);
}

catch (InterruptedException intError) {
    logger.log(Level.SEVERE, "Exception while sleeping: {0}",
intError.getMessage());
    System.exit(1);
}
}
}
```

## プロジェクトの記述 (SDK)

DescribeProjects API を使用すると、プロジェクトに関する情報を取得できます。

プロジェクトを記述するには (SDK)

1. 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 4: AWS CLI と AWS SDKsを設定する](#)」を参照してください。

2. 次のサンプルコードを使用して、プロジェクトを記述します。project\_name を、記述するプロジェクトの名前に置き換えます。--project-names を指定しない場合、すべてのプロジェクトの記述が返されます。

## AWS CLI

```
aws rekognition describe-projects --project-names project_name \  
  --profile custom-labels-access
```

## Python

次のコードを使用します。次のコマンドラインパラメータを指定します。

- project\_name - 記述するプロジェクトの名前。名前を指定しない場合、すべてのプロジェクトの記述が返されます。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
  
"""  
Purpose  
Shows how to describe an Amazon Rekognition Custom Labels project.  
"""  
  
import argparse  
import logging  
import json  
import boto3  
from botocore.exceptions import ClientError  
  
logger = logging.getLogger(__name__)  
  
def display_project_info(project):  
    """  
    Displays information about a Custom Labels project.  
    :param project: The project that you want to display information about.  
    """  
    print(f"Arn: {project['ProjectArn']}")  
    print(f"Status: {project['Status']}")  
  
    if len(project['Datasets']) == 0:
```

```
    print("Datasets: None")
else:
    print("Datasets:")

for dataset in project['Datasets']:
    print(f"\tCreated: {str(dataset['CreationTimestamp'])}")
    print(f"\tType: {dataset['DatasetType']}")
    print(f"\tARN: {dataset['DatasetArn']}")
    print(f"\tStatus: {dataset['Status']}")
    print(f"\tStatus message: {dataset['StatusMessage']}")
    print(f"\tStatus code: {dataset['StatusMessageCode']}")
    print()
print()

def describe_projects(rek_client, project_name):
    """
    Describes an Amazon Rekognition Custom Labels project, or all projects.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_name: The project you want to describe. Pass None to describe
    all projects.
    """

    try:
        # Describe the project
        if project_name is None:
            logger.info("Describing all projects.")
        else:
            logger.info("Describing project: %s.", project_name)

        if project_name is None:
            response = rek_client.describe_projects()
        else:
            project_names = json.loads('["' + project_name + "]')
            response = rek_client.describe_projects(ProjectNames=project_names)

        print('Projects\n-----')
        if len(response['ProjectDescriptions']) == 0:
            print("Project(s) not found.")
        else:
            for project in response['ProjectDescriptions']:
                display_project_info(project)

        logger.info("Finished project description.")
```



```
except ClientError as err:
    logger.exception(
        "Couldn't describe project - %s: %s",
        project_name, err.response['Error']['Message'] )
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "--project_name", help="The name of the project that you want to
describe.", required=False
    )

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)

        args = parser.parse_args()

        print(f"Describing projects: {args.project_name}")

        # Describe the project.
        session = boto3.Session(profile_name='custom-labels-access')
        rekognition_client = session.client("rekognition")

        describe_projects(rekognition_client,
                          args.project_name)

        if args.project_name is None:
            print("Finished describing all projects.")
```

```
        else:
            print("Finished describing project %s.", args.project_name)

    except ClientError as err:
        error_message = f"Problem describing project: {err}"
        logger.exception(error_message)
        print(error_message)

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

## Java V2

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `project_name` - 記述するプロジェクトの ARN。名前を指定しない場合、すべてのプロジェクトの記述が返されます。

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
*/

package com.example.rekognition;

import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DatasetMetadata;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectsRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectsResponse;
import software.amazon.awssdk.services.rekognition.model.ProjectDescription;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

public class DescribeProjects {
```

```
public static final Logger logger =
Logger.getLogger(DescribeProjects.class.getName());

public static void describeMyProjects(RekognitionClient rekClient, String
projectName) {

    DescribeProjectsRequest descProjects = null;

    // If a single project name is supplied, build projectNames argument

    List<String> projectNames = new ArrayList<String>();

    if (projectName == null) {
        descProjects = DescribeProjectsRequest.builder().build();
    } else {
        projectNames.add(projectName);
        descProjects =
DescribeProjectsRequest.builder().projectNames(projectNames).build();
    }

    // Display useful information for each project.

    DescribeProjectsResponse resp =
rekClient.describeProjects(descProjects);

    for (ProjectDescription projectDescription : resp.projectDescriptions())
    {

        System.out.println("ARN: " + projectDescription.projectArn());
        System.out.println("Status: " +
projectDescription.statusAsString());
        if (projectDescription.hasDatasets()) {
            for (DatasetMetadata datasetDescription :
projectDescription.datasets()) {
                System.out.println("\tdataset Type: " +
datasetDescription.datasetTypeAsString());
                System.out.println("\tdataset ARN: " +
datasetDescription.datasetArn());
                System.out.println("\tdataset Status: " +
datasetDescription.statusAsString());
            }
        }
        System.out.println();
    }
}
```

```
    }  
  
    }  
  
    public static void main(String[] args) {  
  
        String projectArn = null;  
  
        // Get command line arguments  
  
        final String USAGE = "\n" + "Usage: " + "<project_name>\n\n" + "Where:  
\n"  
            + "    project_name - (Optional) The name of the project that you  
want to describe. If not specified, all projects "  
            + "are described.\n\n";  
  
        if (args.length > 1) {  
            System.out.println(USAGE);  
            System.exit(1);  
        }  
  
        if (args.length == 1) {  
            projectArn = args[0];  
        }  
  
        try {  
  
            // Get the Rekognition client  
            RekognitionClient rekClient = RekognitionClient.builder()  
                .credentialsProvider(ProfileCredentialsProvider.create("custom-  
labels-access"))  
                .region(Region.US_WEST_2)  
                .build();  
  
            // Describe projects  
  
            describeMyProjects(rekClient, projectArn);  
  
            rekClient.close();  
  
        } catch (RekognitionException rekError) {  
            logger.log(Level.SEVERE, "Rekognition client error: {0}",  
rekError.getMessage());  
            System.exit(1);  
        }  
    }  
}
```

```
    }  
  }  
}
```

## AWS CloudFormationを使用したプロジェクトの作成

Amazon Rekognition Custom Labels は AWS CloudFormation、AWS リソースとインフラストラクチャの作成と管理に費やす時間を短縮できるように、リソースのモデル化とセットアップを支援するサービスであると統合されています。必要なすべての AWS リソースを記述するテンプレートを作成し、AWS CloudFormation がそれらのリソースのプロビジョニングと設定を行います。

を使用して AWS CloudFormation、Amazon Rekognition Custom Labels プロジェクトのプロビジョニングと設定を行うことができます。

を使用すると AWS CloudFormation、テンプレートを再利用して Amazon Rekognition Custom Labels プロジェクトを一貫して繰り返しセットアップできます。プロジェクトを一度記述するだけで、複数の AWS アカウントとリージョンで同じプロジェクトを何度もプロビジョニングできます。

## Amazon Rekognition カスタムラベルと AWS CloudFormation テンプレート

Amazon Rekognition Custom Labels と関連サービスのためのプロジェクトをプロビジョニングし、設定するためには、[AWS CloudFormation テンプレート](#)を理解する必要があります。テンプレートは、JSON や YAML でフォーマットされたテキストファイルです。これらのテンプレートは、AWS CloudFormation スタックでプロビジョニングするリソースを記述します。JSON または YAML に慣れていない場合は、[デザイナー](#)を使用して AWS CloudFormation AWS CloudFormation テンプレートの使用を開始できます。詳細については、「AWS CloudFormation ユーザーガイド」の「[AWS CloudFormation Designer とは](#)」を参照してください。

JSON テンプレートと YAML テンプレートの例を含む Amazon Rekognition Custom Labels プロジェクトのリファレンス情報については、「[Rekognition リソースタイプのリファレンス](#)」を参照してください。

## の詳細 AWS CloudFormation

の詳細については AWS CloudFormation、以下のリソースを参照してください。

- [AWS CloudFormation](#)

- [AWS CloudFormation ユーザーガイド](#)
- [AWS CloudFormation API リファレンス](#)
- [AWS CloudFormation コマンドラインインターフェイスユーザーガイド](#)

## データセットの管理

データセットには、モデルのトレーニングやテストに使用するイメージと、割り当てられたラベルが含まれています。このセクションのトピックでは、Amazon Rekognition Custom Labels コンソールと AWS SDK を使用してデータセットを管理する方法を示しています。

### トピック

- [データセットをプロジェクトに追加する](#)
- [データセットへのイメージの追加](#)
- [既存のデータセットを使用したデータセットの作成 \(SDK\)](#)
- [データセットの記述 \(SDK\)](#)
- [データセットエントリの一覧表示 \(SDK\)](#)
- [トレーニングデータセットの分散 \(SDK\)](#)
- [データセットの削除](#)

## データセットをプロジェクトに追加する

トレーニングデータセットまたはテストデータセットは、既存のデータセットプロジェクトに追加できます。既存のデータセットを置き換える場合は、最初に既存のデータセットを削除します。詳細については、「[データセットの削除](#)」を参照してください。次に、新しいデータセットを追加します。

### トピック

- [データセットをプロジェクトに追加する \(コンソール\)](#)
- [データセットをプロジェクトに追加する \(SDK\)](#)

## データセットをプロジェクトに追加する (コンソール)

Amazon Rekognition Custom Labels コンソールを使用して、トレーニングデータセットまたはテストデータセットをプロジェクトに追加できます。

## データセットをプロジェクトに追加するには

1. Amazon Rekognition コンソールを <https://console.aws.amazon.com/rekognition/> で開きます。
2. 左側のペインで、[カスタムラベルを使用] を選択します。Amazon Rekognition Custom Labels のランディングページが表示されます。
3. 左側のナビゲーションペインで、[Projects] (プロジェクト) を選択します。プロジェクトビューが表示されます。
4. データセットを追加するプロジェクトを選択します。
5. 左側のナビゲーションペインで、プロジェクト名の下にある [データセット] を選択します。
6. プロジェクトに既存のデータセットがない場合は、[データセットを作成] ページが表示されます。次のコマンドを実行します
  - a. [データセットを作成] ページで、イメージソース情報を入力します。詳細については、[「the section called “イメージ付きのデータセットの作成”](#)」を参照してください。
  - b. [データセットを作成] を選択します。
7. プロジェクトに既存のデータセット (トレーニングまたはテスト) がある場合は、プロジェクトの詳細ページが表示されます。次のコマンドを実行します
  - a. プロジェクトの詳細ページで [アクション] を選択します。
  - b. トレーニングデータセットを追加する場合は、[トレーニングデータセットを作成] を選択します。
  - c. テストデータセットを追加する場合は、[テストデータセットを作成] を選択します。
  - d. [データセットを作成] ページで、イメージソース情報を入力します。詳細については、[「the section called “イメージ付きのデータセットの作成”](#)」を参照してください。
  - e. [データセットを作成] を選択します。
8. データセットにイメージを追加します。詳細については、[「イメージの追加 \(コンソール\)」](#)を参照してください。
9. データセットにラベルを追加します。詳細については、[「新しいラベルの追加 \(コンソール\)」](#)を参照してください。
10. イメージにラベルを追加します。イメージレベルのラベルを追加する場合は、[「the section called “イメージにイメージレベルのラベルを割り当てる”](#)」を参照してください。境界ボックスを追加する場合は、[「境界ボックスによるオブジェクトのラベル付け」](#)を参照してください。詳細については、[「データセットの目的の設定」](#)を参照してください。

## データセットをプロジェクトに追加する (SDK)

次の方法で、トレーニングデータセットまたはテストデータセットを既存のデータセットプロジェクトに追加できます。

- マニフェストファイルを使用したデータセットの作成 詳細については、「[SageMaker Ground Truth マニフェストファイル \(SDK\) を使用したデータセットの作成](#)」を参照してください。
- 空のデータセットを作成し、そのデータセットにデータを入力します。次の例は、空のデータセットを作成する方法を示しています。空のデータセットを作成した後でエントリを追加する方法については、「[データセットへのイメージの追加](#)」を参照してください。

### データセットをプロジェクトに追加するには (SDK)

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. 次の例を使用して JSON 行をデータセットに追加します。

#### CLI

`project_arn` を、データセットを追加するプロジェクトに置き換えます。`dataset_type` を TRAIN に置き換えてトレーニングデータセットを作成するか、TEST に置き換えてテストデータセットを作成します。

```
aws rekognition create-dataset --project-arn project_arn \  
  --dataset-type dataset_type \  
  --profile custom-labels-access
```

#### Python

次のコードを使用してデータセットを作成します。次のコマンドラインオプションを指定します。

- `project_arn` - テストデータセットを追加するプロジェクトの ARN。
- `type` - 作成するデータセットのタイプ (トレーニングまたはテスト)

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0
```



```
import argparse
import logging
import time
import boto3

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def create_empty_dataset(rek_client, project_arn, dataset_type):
    """
    Creates an empty Amazon Rekognition Custom Labels dataset.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_arn: The ARN of the project in which you want to create a
    dataset.
    :param dataset_type: The type of the dataset that you want to create (train
    or test).
    """

    try:
        #Create the dataset.
        logger.info("Creating empty %s dataset for project %s",
                    dataset_type, project_arn)

        dataset_type=dataset_type.upper()

        response = rek_client.create_dataset(
            ProjectArn=project_arn, DatasetType=dataset_type
        )

        dataset_arn=response['DatasetArn']

        logger.info("dataset ARN: %s", dataset_arn)

        finished=False
        while finished is False:

            dataset=rek_client.describe_dataset(DatasetArn=dataset_arn)

            status=dataset['DatasetDescription']['Status']

            if status == "CREATE_IN_PROGRESS":
```

```
        logger.info(("Creating dataset: %s ", dataset_arn))
        time.sleep(5)
        continue

    if status == "CREATE_COMPLETE":
        logger.info("Dataset created: %s", dataset_arn)
        finished=True
        continue

    if status == "CREATE_FAILED":
        error_message = f"Dataset creation failed: {status} :
{dataset_arn}"
        logger.exception(error_message)
        raise Exception(error_message)

    error_message = f"Failed. Unexpected state for dataset creation:
{status} : {dataset_arn}"
    logger.exception(error_message)
    raise Exception(error_message)

    return dataset_arn

except ClientError as err:
    logger.exception("Couldn't create dataset: %s", err.response['Error']
['Message'])
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The ARN of the project in which you want to create
the empty dataset."
    )

    parser.add_argument(
        "dataset_type", help="The type of the empty dataset that you want to
create (train or test).")
    )
```

```
def main():

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Creating empty {args.dataset_type} dataset for project
{args.project_arn}")

        # Create the empty dataset.
        session = boto3.Session(profile_name='custom-labels-access')
        rekognition_client = session.client("rekognition")

        dataset_arn=create_empty_dataset(rekognition_client,
            args.project_arn,
            args.dataset_type.lower())

        print(f"Finished creating empty dataset: {dataset_arn}")

    except ClientError as err:
        logger.exception("Problem creating empty dataset: %s", err)
        print(f"Problem creating empty dataset: {err}")
    except Exception as err:
        logger.exception("Problem creating empty dataset: %s", err)
        print(f"Problem creating empty dataset: {err}")

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

## Java V2

次のコードを使用してデータセットを作成します。次のコマンドラインオプションを指定します。

- `project_arn` - テストデータセットを追加するプロジェクトの ARN。

- type - 作成するデータセットのタイプ (トレーニングまたはテスト)

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
*/
package com.example.rekognition;

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.CreateDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.CreateDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.DatasetDescription;
import software.amazon.awssdk.services.rekognition.model.DatasetStatus;
import software.amazon.awssdk.services.rekognition.model.DatasetType;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;
import
software.amazon.awssdk.services.rekognition.model.DescribeDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

import java.net.URI;
import java.util.logging.Level;
import java.util.logging.Logger;

public class CreateEmptyDataset {

    public static final Logger logger =
Logger.getLogger(CreateEmptyDataset.class.getName());

    public static String createMyEmptyDataset(RekognitionClient rekClient,
String projectArn, String datasetType)
        throws Exception, RekognitionException {

        try {

            logger.log(Level.INFO, "Creating empty {0} dataset for project :
{1}",
                new Object[] { datasetType.toString(), projectArn });

            DatasetType requestDatasetType = null;
```

```
switch (datasetType) {
case "train":
    requestDatasetType = DatasetType.TRAIN;
    break;
case "test":
    requestDatasetType = DatasetType.TEST;
    break;
default:
    logger.log(Level.SEVERE, "Unrecognized dataset type: {0}",
datasetType);
    throw new Exception("Unrecognized dataset type: " +
datasetType);
}

CreateDatasetRequest createDatasetRequest =
CreateDatasetRequest.builder().projectArn(projectArn)
    .datasetType(requestDatasetType).build();

CreateDatasetResponse response =
rekClient.createDataset(createDatasetRequest);

boolean created = false;

//Wait until updates finishes

do {

    DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder()
        .datasetArn(response.datasetArn()).build();
    DescribeDatasetResponse describeDatasetResponse =
rekClient.describeDataset(describeDatasetRequest);

    DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();

    DatasetStatus status = datasetDescription.status();

    logger.log(Level.INFO, "Creating dataset ARN: {0} ",
response.datasetArn());

    switch (status) {
```

```
        case CREATE_COMPLETE:
            logger.log(Level.INFO, "Dataset created");
            created = true;
            break;

        case CREATE_IN_PROGRESS:
            Thread.sleep(5000);
            break;

        case CREATE_FAILED:
            String error = "Dataset creation failed: " +
datasetDescription.statusAsString() + " "
                + datasetDescription.statusMessage() + " " +
response.datasetArn();
            logger.log(Level.SEVERE, error);
            throw new Exception(error);

        default:
            String unexpectedError = "Unexpected creation state: " +
datasetDescription.statusAsString() + " "
                + datasetDescription.statusMessage() + " " +
response.datasetArn();
            logger.log(Level.SEVERE, unexpectedError);
            throw new Exception(unexpectedError);
    }

    } while (created == false);

    return response.datasetArn();

} catch (RekognitionException e) {
    logger.log(Level.SEVERE, "Could not create dataset: {0}",
e.getMessage());
    throw e;
}

}

public static void main(String args[]) {

    String datasetType = null;
    String datasetArn = null;
    String projectArn = null;
```

```
        final String USAGE = "\n" + "Usage: " + "<project_arn> <dataset_type>\n\n" + "Where:\n"
            + "    project_arn - the ARN of the project that you want to add\n" + "copy the data to.\n\n"
            + "    dataset_type - the type of the empty dataset that you want\n" + "to create (train or test).\n\n";

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

        projectArn = args[0];
        datasetType = args[1];

        try {

            // Get the Rekognition client
            RekognitionClient rekClient = RekognitionClient.builder()
                .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
                .region(Region.US_WEST_2)
                .build();

            // Create the dataset
            datasetArn = createMyEmptyDataset(rekClient, projectArn,
datasetType);

            System.out.println(String.format("Created dataset: %s",
datasetArn));

            rekClient.close();

        } catch (RekognitionException rekError) {
            logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
            System.exit(1);
        } catch (Exception rekError) {
            logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());
            System.exit(1);
        }
    }
}
```

```
}  
  
}
```

3. データセットにイメージを追加します。詳細については、「[イメージの追加 \(SDK\)](#)」を参照してください。

## データセットへのイメージの追加

データセットにイメージを追加するには、Amazon Rekognition Custom Labels コンソールを使用するか、UpdateDatasetEntries API を呼び出します。

トピック

- [イメージの追加 \(コンソール\)](#)
- [イメージの追加 \(SDK\)](#)

### イメージの追加 (コンソール)

Amazon Rekognition Custom Labels コンソールを使用する場合、ローカルコンピュータからイメージをアップロードします。イメージは、データセットの作成に使用されたイメージが保存されている Amazon S3 バケットの場所 (コンソールまたは外部) に追加されます。

データセットに画像を追加するには (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. 左側のペインで、[カスタムラベルを使用] を選択します。Amazon Rekognition Custom Labels のランディングページが表示されます。
3. 左側のナビゲーションペインで、[Projects] (プロジェクト) を選択します。プロジェクトビューが表示されます。
4. 使用するプロジェクトを選択します。
5. 左側のナビゲーションペインで、プロジェクト名の下にある [データセット] を選択します。
6. [Actions] (アクション) をクリックし、画像を追加するデータセットを選択します。
7. データセットにアップロードする画像を選択します。画像をドラッグするか、ローカルコンピュータからアップロードする画像を選択できます。同時にアップロードできる画像は、30 枚までです。



8. [Upload images] (画像をアップロード) を選択します。
9. [Save changes] (変更の保存) をクリックします。
10. イメージにラベルを付けます。詳細については、「[イメージにラベルを付ける](#)」を参照してください。

## イメージの追加 (SDK)

UpdateDatasetEntries によってマニフェストファイルに JSON 行を更新または追加します。JSON 行を byte64 でエンコードされたデータオブジェクトとして GroundTruth フィールドに渡します。AWS SDK を使用して UpdateDatasetEntries を呼び出す場合、SDK によってデータをエンコードします。JSON の各行には、割り当てられたラベルや境界ボックスの情報など、1 つのイメージに関する情報が含まれています。例:

```
{"source-ref":"s3://bucket/image","BB":{"annotations":
[{"left":1849,"top":1039,"width":422,"height":283,"class_id":0},
{"left":1849,"top":1340,"width":443,"height":415,"class_id":1},
{"left":2637,"top":1380,"width":676,"height":338,"class_id":2},
{"left":2634,"top":1051,"width":673,"height":338,"class_id":3}],"image_size":
[{"width":4000,"height":2667,"depth":3}]},"BB-metadata":{"job-name":"labeling-job/
BB"},"class-map":
{"0":"comparator","1":"pot_resistor","2":"ir_phototransistor","3":"ir_led"},"human-
annotated":"yes","objects":[{"confidence":1},{ "confidence":1},{ "confidence":1},
{"confidence":1}],"creation-date":"2021-06-22T10:11:18.006Z","type":"groundtruth/
object-detection"}}
```

詳細については、「[マニフェストファイルの作成](#)」を参照してください。

source-ref フィールドをキーとして使用して、更新するイメージを識別します。データセットに一致する source-ref フィールド値が含まれていない場合は、新しいイメージとして JSON 行が追加されます。

データセットへのイメージを追加するには (SDK)

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. 次の例を使用して JSON 行をデータセットに追加します。

## CLI

GroundTruth の値を、使用する JSON 行に置き換えます。JSON 行内の特殊文字はすべてエスケープする必要があります。

```
aws rekognition update-dataset-entries\  
  --dataset-arn dataset_arn \  
  --changes '{"GroundTruth" : "{\\"source-ref\\":\\"s3://your_bucket/your_image  
\\",\\"BB\\":{\\"annotations\\":[{\\"left\\":1776,\\"top\\":1017,\\"width\\":458,\\"height  
\\":317,\\"class_id\\":0},{\\"left\\":1797,\\"top\\":1334,\\"width\\":418,\\"height  
\\":415,\\"class_id\\":1},{\\"left\\":2597,\\"top\\":1361,\\"width\\":655,\\"height  
\\":329,\\"class_id\\":2},{\\"left\\":2581,\\"top\\":1020,\\"width\\":689,\\"height  
\\":338,\\"class_id\\":3}],\\"image_size\\":[{\\"width\\":4000,\\"height\\":2667,  
\\\"depth\\":3}]}],\\"BB-metadata\\":{\\"job-name\\":\\"labeling-job/BB\\",\\"class-map  
\\":{\\"0\\":\\"comparator\\",\\"1\\":\\"pot_resistor\\",\\"2\\":\\"ir_phototransistor\\",  
\\\"3\\":\\"ir_led\\"},\\"human-annotated\\":\\"yes\\",\\"objects\\":[{\\"confidence\\":1},  
{\\"confidence\\":1},{\\"confidence\\":1}],\\"creation-date\\":  
\\\"2021-06-22T10:10:48.492Z\\",\\"type\\":\\"groundtruth/object-detection\\"}]"}' \  
  --cli-binary-format raw-in-base64-out \  
  --profile custom-labels-access
```

## Python

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `dataset_arn` - 更新するデータセットの ARN。
- `updates_file` - JSON 行の更新を含むファイル。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
  
""  
Purpose  
Shows how to add entries to an Amazon Rekognition Custom Labels dataset.  
""  
  
import argparse  
import logging  
import time  
import json
```

```
import boto3

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def update_dataset_entries(rek_client, dataset_arn, updates_file):
    """
    Adds dataset entries to an Amazon Rekognition Custom Labels dataset.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param dataset_arn: The ARN of the dataset that you want to update.
    :param updates_file: The manifest file of JSON Lines that contains the
    updates.
    """

    try:
        status=""
        status_message=""

        # Update dataset entries.
        logger.info("Updating dataset %s", dataset_arn)

        with open(updates_file) as f:
            manifest_file = f.read()

        changes=json.loads('{ "GroundTruth" : ' +
            json.dumps(manifest_file) +
            '}')

        rek_client.update_dataset_entries(
            Changes=changes, DatasetArn=dataset_arn
        )

        finished=False
        while finished is False:

            dataset=rek_client.describe_dataset(DatasetArn=dataset_arn)

            status=dataset['DatasetDescription']['Status']
            status_message=dataset['DatasetDescription']['StatusMessage']

            if status == "UPDATE_IN_PROGRESS":
```

```
        logger.info("Updating dataset: %s ", dataset_arn)
        time.sleep(5)
        continue

    if status == "UPDATE_COMPLETE":
        logger.info("Dataset updated: %s : %s : %s",
                    status, status_message, dataset_arn)
        finished=True
        continue

    if status == "UPDATE_FAILED":
        error_message = f"Dataset update failed: {status} :
{status_message} : {dataset_arn}"
        logger.exception(error_message)
        raise Exception (error_message)

        error_message = f"Failed. Unexpected state for dataset update:
{status} : {status_message} : {dataset_arn}"
        logger.exception(error_message)
        raise Exception(error_message)

    logger.info("Added entries to dataset")

    return status, status_message

except ClientError as err:
    logger.exception("Couldn't update dataset: %s", err.response['Error']
['Message'])
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "dataset_arn", help="The ARN of the dataset that you want to update."
    )

    parser.add_argument(
```

```
        "updates_file", help="The manifest file of JSON Lines that contains the
updates."
    )

def main():

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    try:

        #get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Updating dataset {args.dataset_arn} with entries from
{args.updates_file}.")

        # Update the dataset.
        session = boto3.Session(profile_name='custom-labels-access')
        rekognition_client = session.client("rekognition")

        status, status_message=update_dataset_entries(rekognition_client,
            args.dataset_arn,
            args.updates_file)

        print(f"Finished updates dataset: {status} : {status_message}")

    except ClientError as err:
        logger.exception("Problem updating dataset: %s", err)
        print(f"Problem updating dataset: {err}")

    except Exception as err:
        logger.exception("Problem updating dataset: %s", err)
        print(f"Problem updating dataset: {err}")

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

## Java V2

- `dataset_arn` - 更新するデータセットの ARN。
- `update_file` - JSON 行の更新を含むファイル。

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
 */
package com.example.rekognition;

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.DatasetChanges;
import software.amazon.awssdk.services.rekognition.model.DatasetDescription;
import software.amazon.awssdk.services.rekognition.model.DatasetStatus;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
    software.amazon.awssdk.services.rekognition.model.UpdateDatasetEntriesRequest;
import
    software.amazon.awssdk.services.rekognition.model.UpdateDatasetEntriesResponse;

import java.io.FileInputStream;
import java.io.InputStream;
import java.util.logging.Level;
import java.util.logging.Logger;

public class UpdateDatasetEntries {

    public static final Logger logger =
        Logger.getLogger(UpdateDatasetEntries.class.getName());

    public static String updateMyDataset(RekognitionClient rekClient, String
datasetArn,
        String updateFile
        ) throws Exception, RekognitionException {
```

```
try {

    logger.log(Level.INFO, "Updating dataset {0}",
        new Object[] { datasetArn});

    InputStream sourceStream = new FileInputStream(updateFile);
    SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);

    DatasetChanges datasetChanges = DatasetChanges.builder()
        .groundTruth(sourceBytes).build();

    UpdateDatasetEntriesRequest updateDatasetEntriesRequest =
UpdateDatasetEntriesRequest.builder()
        .changes(datasetChanges)
        .datasetArn(datasetArn)
        .build();

    UpdateDatasetEntriesResponse response =
rekClient.updateDatasetEntries(updateDatasetEntriesRequest);

    boolean updated = false;

    //Wait until update completes

    do {

        DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder()
            .datasetArn(datasetArn).build();
        DescribeDatasetResponse describeDatasetResponse =
rekClient.describeDataset(describeDatasetRequest);

        DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();

        DatasetStatus status = datasetDescription.status();

        logger.log(Level.INFO, " dataset ARN: {0} ", datasetArn);

        switch (status) {

            case UPDATE_COMPLETE:
```

```
        logger.log(Level.INFO, "Dataset updated");
        updated = true;
        break;

    case UPDATE_IN_PROGRESS:
        Thread.sleep(5000);
        break;

    case UPDATE_FAILED:
        String error = "Dataset update failed: " +
datasetDescription.statusAsString() + " "
            + datasetDescription.statusMessage() + " " +
datasetArn;

        logger.log(Level.SEVERE, error);
        throw new Exception(error);

    default:
        String unexpectedError = "Unexpected update state: " +
datasetDescription.statusAsString() + " "
            + datasetDescription.statusMessage() + " " +
datasetArn;

        logger.log(Level.SEVERE, unexpectedError);
        throw new Exception(unexpectedError);
    }

    } while (updated == false);

    return datasetArn;

} catch (RekognitionException e) {
    logger.log(Level.SEVERE, "Could not update dataset: {0}",
e.getMessage());
    throw e;
}

}

public static void main(String args[]) {

    String updatesFile = null;
    String datasetArn = null;
```



```
        final String USAGE = "\n" + "Usage: " + "<project_arn> <dataset_arn>  
<updates_file>\n\n" + "Where:\n"  
            + "    dataset_arn - the ARN of the dataset that you want to  
update.\n\n"  
            + "    update_file - The file that includes in JSON Line updates.  
\n\n";  
  
        if (args.length != 2) {  
            System.out.println(USAGE);  
            System.exit(1);  
        }  
  
        datasetArn = args[0];  
        updatesFile = args[1];  
  
        try {  
  
            // Get the Rekognition client.  
            RekognitionClient rekClient = RekognitionClient.builder()  
                .credentialsProvider(ProfileCredentialsProvider.create("custom-  
labels-access"))  
                .region(Region.US_WEST_2)  
                .build();  
  
            // Update the dataset  
            datasetArn = updateMyDataset(rekClient, datasetArn, updatesFile);  
  
            System.out.println(String.format("Dataset updated: %s",  
datasetArn));  
  
            rekClient.close();  
  
        } catch (RekognitionException rekError) {  
            logger.log(Level.SEVERE, "Rekognition client error: {0}",  
rekError.getMessage());  
            System.exit(1);  
        } catch (Exception rekError) {  
            logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());  
            System.exit(1);  
        }  
  
    }  
}
```

```
}
```

## 既存のデータセットを使用したデータセットの作成 (SDK)

次の手順では、[CreateDataset](#) オペレーションを使用して、既存のデータセットからデータセットを作成する方法を示しています。

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. 次のサンプルコードを使用し、別のデータセットをコピーしてデータセットを作成します。

### AWS CLI

次のコードを使用してデータセットを作成します。以下に置き換えます:

- `project_arn` - データセットを追加するプロジェクトの ARN。
- `dataset_type` - プロジェクトで作成するデータセットのタイプ (TRAIN または TEST) を指定します。
- `dataset_arn` - コピーするデータセットの ARN を指定します。

```
aws rekognition create-dataset --project-arn project_arn \  
  --dataset-type dataset_type \  
  --dataset-source '{ "DatasetArn" : "dataset_arn" }' \  
  --profile custom-labels-access
```

### Python

次の例では、既存のデータセットを使用してデータセットを作成し、その ARN を表示します。

プログラムを実行するには、次のコマンドライン引数を指定します。

- `project_arn` - 使用するプロジェクトの ARN。
- `dataset_type` - 作成するプロジェクトのデータセットのタイプ (train または test)。
- `dataset_arn` - データセットを作成するデータセットの ARN。

```
# Copyright 2023 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/
awsdocs/amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-
SAMPLECODE.)

import argparse
import logging
import time
import json
import boto3

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def create_dataset_from_existing_dataset(rek_client, project_arn, dataset_type,
dataset_arn):
    """
    Creates an Amazon Rekognition Custom Labels dataset using an existing
    dataset.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_arn: The ARN of the project in which you want to create a
    dataset.
    :param dataset_type: The type of the dataset that you want to create (train
    or test).
    :param dataset_arn: The ARN of the existing dataset that you want to use.
    """

    try:
        # Create the dataset

        dataset_type=dataset_type.upper()

        logger.info(
            "Creating %s dataset for project %s from dataset %s.",
            dataset_type,project_arn, dataset_arn)

        dataset_source = json.loads(
            '{ "DatasetArn": "' + dataset_arn + '"}'
        )
```

```
    response = rek_client.create_dataset(
        ProjectArn=project_arn, DatasetType=dataset_type,
DatasetSource=dataset_source
    )

    dataset_arn = response['DatasetArn']

    logger.info("New dataset ARN: %s", dataset_arn)

    finished = False
    while finished is False:

        dataset = rek_client.describe_dataset(DatasetArn=dataset_arn)

        status = dataset['DatasetDescription']['Status']

        if status == "CREATE_IN_PROGRESS":

            logger.info(("Creating dataset: %s ", dataset_arn))
            time.sleep(5)
            continue

        if status == "CREATE_COMPLETE":
            logger.info("Dataset created: %s", dataset_arn)
            finished = True
            continue

        if status == "CREATE_FAILED":
            error_message = f"Dataset creation failed: {status} :
{dataset_arn}"
            logger.exception(error_message)
            raise Exception(error_message)

            error_message = f"Failed. Unexpected state for dataset creation:
{status} : {dataset_arn}"
            logger.exception(error_message)
            raise Exception(error_message)

    return dataset_arn

except ClientError as err:
    logger.exception(
        "Couldn't create dataset: %s",err.response['Error']['Message'] )
    raise
```

```
def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The ARN of the project in which you want to create
the dataset."
    )

    parser.add_argument(
        "dataset_type", help="The type of the dataset that you want to create
(train or test)."
    )

    parser.add_argument(
        "dataset_arn", help="The ARN of the dataset that you want to copy from."
    )

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(
            f"Creating {args.dataset_type} dataset for project
{args.project_arn}")

        # Create the dataset.
        session = boto3.Session(profile_name='custom-labels-access')
        rekognition_client = session.client("rekognition")

        dataset_arn = create_dataset_from_existing_dataset(rekognition_client,
```

```
        args.project_arn,
        args.dataset_type,
        args.dataset_arn)

    print(f"Finished creating dataset: {dataset_arn}")

except ClientError as err:
    logger.exception("Problem creating dataset: %s", err)
    print(f"Problem creating dataset: {err}")
except Exception as err:
    logger.exception("Problem creating dataset: %s", err)
    print(f"Problem creating dataset: {err}")

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

## Java V2

次の例では、既存のデータセットを使用してデータセットを作成し、その ARN を表示します。

プログラムを実行するには、次のコマンドライン引数を指定します。

- `project_arn` - 使用するプロジェクトの ARN。
- `dataset_type` - 作成するプロジェクトのデータセットのタイプ (`train` または `test`)。
- `dataset_arn` - データセットを作成するデータセットの ARN。

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
*/

package com.example.rekognition;

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.CreateDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.CreateDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.DatasetDescription;
```

```
import software.amazon.awssdk.services.rekognition.model.DatasetSource;
import software.amazon.awssdk.services.rekognition.model.DatasetStatus;
import software.amazon.awssdk.services.rekognition.model.DatasetType;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

import java.util.logging.Level;
import java.util.logging.Logger;

public class CreateDatasetExisting {

    public static final Logger logger =
        Logger.getLogger(CreateDatasetExisting.class.getName());

    public static String createMyDataset(RekognitionClient rekClient, String
        projectArn, String datasetType,
        String existingDatasetArn) throws Exception, RekognitionException {

        try {

            logger.log(Level.INFO, "Creating {0} dataset for project : {1} from
                dataset {2} ",
                new Object[] { datasetType.toString(), projectArn,
                    existingDatasetArn });

            DatasetType requestDatasetType = null;

            switch (datasetType) {
            case "train":
                requestDatasetType = DatasetType.TRAIN;
                break;
            case "test":
                requestDatasetType = DatasetType.TEST;
                break;
            default:
                logger.log(Level.SEVERE, "Unrecognized dataset type: {0}",
                    datasetType);
                throw new Exception("Unrecognized dataset type: " +
                    datasetType);
            }

        }

    }

}
```

```
DatasetSource datasetSource =
DatasetSource.builder().datasetArn(existingDatasetArn).build();

CreateDatasetRequest createDatasetRequest =
CreateDatasetRequest.builder().projectArn(projectArn)

.datasetType(requestDatasetType).datasetSource(datasetSource).build();

CreateDatasetResponse response =
rekClient.createDataset(createDatasetRequest);

boolean created = false;

//Wait until create finishes

do {

    DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder()
        .datasetArn(response.datasetArn()).build();
    DescribeDatasetResponse describeDatasetResponse =
rekClient.describeDataset(describeDatasetRequest);

    DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();

    DatasetStatus status = datasetDescription.status();

    logger.log(Level.INFO, "Creating dataset ARN: {0} ",
response.datasetArn());

    switch (status) {

        case CREATE_COMPLETE:
            logger.log(Level.INFO, "Dataset created");
            created = true;
            break;

        case CREATE_IN_PROGRESS:
            Thread.sleep(5000);
            break;

        case CREATE_FAILED:
```



```
        String error = "Dataset creation failed: " +
datasetDescription.statusAsString() + " "
                + datasetDescription.statusMessage() + " " +
response.datasetArn();
        logger.log(Level.SEVERE, error);
        throw new Exception(error);

        default:
            String unexpectedError = "Unexpected creation state: " +
datasetDescription.statusAsString() + " "
                + datasetDescription.statusMessage() + " " +
response.datasetArn();
            logger.log(Level.SEVERE, unexpectedError);
            throw new Exception(unexpectedError);
        }

    } while (created == false);

    return response.datasetArn();

} catch (RekognitionException e) {
    logger.log(Level.SEVERE, "Could not create dataset: {0}",
e.getMessage());
    throw e;
}

}

public static void main(String[] args) {

    String datasetType = null;
    String datasetArn = null;
    String projectArn = null;
    String datasetSourceArn = null;

    final String USAGE = "\n" + "Usage: " + "<project_arn> <dataset_type>
<dataset_arn>\n\n" + "Where:\n"
        + "    project_arn - the ARN of the project that you want to add
copy the data to.\n\n"
        + "    dataset_type - the type of the dataset that you want to
create (train or test).\n\n"
        + "    dataset_arn - the ARN of the dataset that you want to copy
from.\n\n";
```

```
    if (args.length != 3) {
        System.out.println(USAGE);
        System.exit(1);
    }

    projectArn = args[0];
    datasetType = args[1];
    datasetSourceArn = args[2];

    try {

        // Get the Rekognition client
        RekognitionClient rekClient = RekognitionClient.builder()
            .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
            .region(Region.US_WEST_2)
            .build();

        // Create the dataset
        datasetArn = createMyDataset(rekClient, projectArn, datasetType,
datasetSourceArn);

        System.out.println(String.format("Created dataset: %s",
datasetArn));

        rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    } catch (Exception rekError) {
        logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());
        System.exit(1);
    }

}

}
```

## データセットの記述 (SDK)

DescribeDataset API を使用して、データセットに関する情報を取得できます。

データセットを記述するには (SDK)

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. 次のサンプルコードを使用して、データセットを記述します。

### AWS CLI

dataset-arn の値を記述するデータセットの ARN に変更します。

```
aws rekognition describe-dataset --dataset-arn dataset_arn \  
--profile custom-labels-access
```

### Python

次のコードを使用します。次のコマンドラインパラメータを指定します。

- dataset\_arn - 記述するデータセットの ARN。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
  
"""  
Purpose  
Shows how to describe an Amazon Rekognition Custom Labels dataset.  
"""  
  
import argparse  
import logging  
import boto3  
  
from botocore.exceptions import ClientError  
  
logger = logging.getLogger(__name__)
```

```
def describe_dataset(rek_client, dataset_arn):
    """
    Describes an Amazon Rekognition Custom Labels dataset.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param dataset_arn: The ARN of the dataset that you want to describe.
    """

    try:
        # Describe the dataset
        logger.info("Describing dataset %s", dataset_arn)

        dataset = rek_client.describe_dataset(DatasetArn=dataset_arn)

        description = dataset['DatasetDescription']

        print(f"Created: {str(description['CreationTimestamp'])}")
        print(f"Updated: {str(description['LastUpdatedTimestamp'])}")
        print(f"Status: {description['Status']}")
        print(f"Status message: {description['StatusMessage']}")
        print(f"Status code: {description['StatusMessageCode']}")
        print("Stats:")
        print(
            f"\tLabeled entries: {description['DatasetStats']
            ['LabeledEntries']}")
        print(
            f"\tTotal entries: {description['DatasetStats']['TotalEntries']}")
        print(f"\tTotal labels: {description['DatasetStats']['TotalLabels']}")

    except ClientError as err:
        logger.exception("Couldn't describe dataset: %s",
                        err.response['Error']['Message'])
        raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "dataset_arn", help="The ARN of the dataset that you want to describe."
    )
```

```
def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Describing dataset {args.dataset_arn}")

        # Describe the dataset.
        session = boto3.Session(profile_name='custom-labels-access')
        rekognition_client = session.client("rekognition")

        describe_dataset(rekognition_client, args.dataset_arn)

        print(f"Finished describing dataset: {args.dataset_arn}")

    except ClientError as err:
        error_message=f"Problem describing dataset: {err}"
        logger.exception(error_message)
        print(error_message)
    except Exception as err:
        error_message = f"Problem describing dataset: {err}"
        logger.exception(error_message)
        print(error_message)

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

## Java V2

- `dataset_arn` - 記述するデータセットの ARN。

```
/*
```

```
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
SPDX-License-Identifier: Apache-2.0
*/

package com.example.rekognition;

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.DatasetDescription;
import software.amazon.awssdk.services.rekognition.model.DatasetStats;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

import java.util.logging.Level;
import java.util.logging.Logger;

public class DescribeDataset {

    public static final Logger logger =
        Logger.getLogger(DescribeDataset.class.getName());

    public static void describeMyDataset(RekognitionClient rekClient, String
datasetArn) {

        try {

            DescribeDatasetRequest describeDatasetRequest =
                DescribeDatasetRequest.builder().datasetArn(datasetArn)
                    .build();
            DescribeDatasetResponse describeDatasetResponse =
                rekClient.describeDataset(describeDatasetRequest);

            DatasetDescription datasetDescription =
                describeDatasetResponse.datasetDescription();
            DatasetStats datasetStats = datasetDescription.datasetStats();

            System.out.println("ARN: " + datasetArn);
            System.out.println("Created: " +
                datasetDescription.creationTimestamp().toString());
            System.out.println("Updated: " +
                datasetDescription.lastUpdatedTimestamp().toString());
        }
    }
}
```

```
        System.out.println("Status: " +
datasetDescription.statusAsString());
        System.out.println("Message: " +
datasetDescription.statusMessage());
        System.out.println("Total Labels: " +
datasetStats.totalLabels().toString());
        System.out.println("Total entries: " +
datasetStats.totalEntries().toString());
        System.out.println("Entries with labels: " +
datasetStats.labeledEntries().toString());
        System.out.println("Entries with at least 1 error: " +
datasetStats.errorEntries().toString());

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        throw rekError;
    }
}

public static void main(String[] args) {

    final String USAGE = "\n" + "Usage: " + "<dataset_arn>\n\n" + "Where:\n"
        + "    dataset_arn - The ARN of the dataset that you want to
describe.\n\n";

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

    String datasetArn = args[0];

    try {

        // Get the Rekognition client.
        RekognitionClient rekClient = RekognitionClient.builder()
            .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
            .region(Region.US_WEST_2)
            .build();

        // Describe the dataset.
```

```
        describeMyDataset(rekClient, datasetArn);

        rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    }

}

}
```

## データセットエントリの一覧表示 (SDK)

ListDatasetEntries API を使用して、データセット内の各イメージの JSON 行を一覧表示できます。詳細については、「[マニフェストファイルの作成](#)」を参照してください。

データセットエントリを一覧表示するには (SDK)

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. 次のサンプルコードを使用して、データセット内のエントリを一覧表示します。

### AWS CLI

dataset-arn の値を一覧表示するデータセットの ARN に変更します。

```
aws rekognition list-dataset-entries --dataset-arn dataset_arn \  
--profile custom-labels-access
```

エラーがある JSON 行のみを一覧表示するには、has-errors を指定します。

```
aws rekognition list-dataset-entries --dataset-arn dataset_arn \  
--has-errors \  
--profile custom-labels-access
```



## Python

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `dataset_arn` - 一覧表示するデータセットの ARN。
- `show_errors_only` - エラーのみを表示する場合は `true` を指定します。それ以外は `false` を指定します。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

"""
Purpose
Shows how to list the entries in an Amazon Rekognition Custom Labels dataset.
"""

import argparse
import logging
import boto3

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def list_dataset_entries(rek_client, dataset_arn, show_errors):
    """
    Lists the entries in an Amazon Rekognition Custom Labels dataset.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param dataset_arn: The ARN of the dataet that you want to use.
    """

    try:
        # List the entries.
        logger.info("Listing dataset entries for the dataset %s.", dataset_arn)

        finished = False
        count = 0
        next_token = ""
        show_errors_only = False
```

```
    if show_errors.lower() == "true":
        show_errors_only = True

    while finished is False:

        response = rek_client.list_dataset_entries(
            DatasetArn=dataset_arn,
            HasErrors=show_errors_only,
            MaxResults=100,
            NextToken=next_token)

        count += len(response['DatasetEntries'])

        for entry in response['DatasetEntries']:
            print(entry)

        if 'NextToken' not in response:
            finished = True
            logger.info("No more entries. Total:%s", count)
        else:
            next_token = response['NextToken']
            logger.info("Getting more entries. Total so far :%s", count)

    except ClientError as err:
        logger.exception(
            "Couldn't list dataset: %s",
            err.response['Error']['Message'])
        raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "dataset_arn", help="The ARN of the dataset that you want to list."
    )

    parser.add_argument(
        "show_errors_only", help="true if you want to see errors only. false
otherwise."
    )
```

```
)

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Listing entries for dataset {args.dataset_arn}")

        # List the dataset entries.
        session = boto3.Session(profile_name='custom-labels-access')
        rekognition_client = session.client("rekognition")

        list_dataset_entries(rekognition_client,
                             args.dataset_arn,
                             args.show_errors_only)

        print(f"Finished listing entries for dataset: {args.dataset_arn}")

    except ClientError as err:
        error_message = f"Problem listing dataset: {err}"
        logger.exception(error_message)
        print(error_message)
    except Exception as err:
        error_message = f"Problem listing dataset: {err}"
        logger.exception(error_message)
        print(error_message)

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

## Java V2

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `dataset_arn` - 一覧表示するデータセットの ARN。
- `show_errors_only` - エラーのみを表示する場合は `true` を指定します。それ以外は `false` を指定します。

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
 */

package com.example.rekognition;

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.ListDatasetEntriesRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
    software.amazon.awssdk.services.rekognition.paginators.ListDatasetEntriesIterable;

import java.net.URI;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ListDatasetEntries {

    public static final Logger logger =
        Logger.getLogger(ListDatasetEntries.class.getName());

    public static void listMyDatasetEntries(RekognitionClient rekClient, String
datasetArn, boolean showErrorsOnly)
        throws Exception, RekognitionException {

        try {

            logger.log(Level.INFO, "Listing dataset {0}", new Object[]
{ datasetArn });

            ListDatasetEntriesRequest listDatasetEntriesRequest =
                ListDatasetEntriesRequest.builder()
```

```
.hasErrors(showErrorsOnly).datasetArn(datasetArn).maxResults(1).build();

        ListDatasetEntriesIterable datasetEntriesList = rekClient
            .listDatasetEntriesPaginator(listDatasetEntriesRequest);

        datasetEntriesList.stream().flatMap(r ->
r.datasetEntries().stream())
            .forEach(datasetEntry ->
System.out.println(datasetEntry.toString()));

    } catch (RekognitionException e) {
        logger.log(Level.SEVERE, "Could not update dataset: {0}",
e.getMessage());
        throw e;
    }

}

public static void main(String args[]) {

    boolean showErrorsOnly = false;
    String datasetArn = null;

    final String USAGE = "\n" + "Usage: " + "<project_arn> <dataset_arn>
<updates_file>\n\n" + "Where:\n"
        + "    dataset_arn - the ARN of the dataset that you want to
update.\n\n"
        + "    show_errors_only - true to show only errors. false
otherwise.\n\n";

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

    datasetArn = args[0];
    if (args[1].toLowerCase().equals("true")) {

        showErrorsOnly = true;
    }

    try {
```

```
// Get the Rekognition client.
RekognitionClient rekClient = RekognitionClient.builder()
    .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
    .region(Region.US_WEST_2)
    .build();

// list the dataset entries.

listMyDatasetEntries(rekClient, datasetArn, showErrorsOnly);

System.out.println(String.format("Finished listing entries for :
%s", datasetArn));

rekClient.close();

} catch (RekognitionException rekError) {
    logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
    System.exit(1);
} catch (Exception rekError) {
    logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());
    System.exit(1);
}

}

}
```

## トレーニングデータセットの分散 (SDK)

Amazon Rekognition Custom Labels には、モデルをトレーニングするためのトレーニングデータセットとテストデータセットが必要です。

API を使用している場合は、[DistributeDatasetEntries API](#) を使用して、トレーニングデータセットの 20% を空のテストデータセットに分散できます。利用可能なマニフェストファイルが 1 つしかない場合は、トレーニングデータセットを分散すると便利です。1 つのマニフェストファイルを使用して、トレーニングデータセットを作成します。次に、空のテストデータセットを作成し、DistributeDatasetEntries を使用してテストデータセットを作成します。

**Note**

Amazon Rekognition Custom Labels コンソールを使用し、1つのデータセットプロジェクトから開始した場合、Amazon Rekognition Custom Labels はトレーニング中にトレーニングデータセットを分割 (分散) してテストデータセットを作成します。トレーニングデータセットのエントリの 20% をテストデータセットに移動します。

トレーニングデータセットを分散するには (SDK)

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. プロジェクトを作成します。詳細については、「[Amazon Rekognition Custom Labels プロジェクトの作成 \(SDK\)](#)」を参照してください。
3. トレーニングデータセットを作成します。データセットについては、「[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](#)」を参照してください。
4. 空のテストデータセットを作成します。
5. 次のサンプルコードを使用して、トレーニングデータセットエントリの 20% をテストデータセットに分散します。プロジェクトのデータセットの Amazon リソースネーム (ARN) は、[DescribeProjects](#) を呼び出すことにより取得できます。サンプルコードについては、「[プロジェクトの記述 \(SDK\)](#)」を参照してください。

## AWS CLI

使用するデータセットの ARN の値で `training_dataset-arn` と `test_dataset-arn` の値を変更します。

```
aws rekognition distribute-dataset-entries --datasets ['{"Arn":  
  "training_dataset_arn"}, {"Arn": "test_dataset_arn"}'] \  
  --profile custom-labels-access
```

## Python

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `training_dataset_arn` - エントリを分散するトレーニングデータセットの ARN。
- `test_dataset_arn` - エントリを分散する先のテストデータセットの ARN。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

import argparse
import logging
import time
import json
import boto3

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def check_dataset_status(rek_client, dataset_arn):
    """
    Checks the current status of a dataset.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param dataset_arn: The dataset that you want to check.
    :return: The dataset status and status message.
    """
    finished = False
    status = ""
    status_message = ""

    while finished is False:

        dataset = rek_client.describe_dataset(DatasetArn=dataset_arn)

        status = dataset['DatasetDescription']['Status']
        status_message = dataset['DatasetDescription']['StatusMessage']

        if status == "UPDATE_IN_PROGRESS":

            logger.info("Distributing dataset: %s ", dataset_arn)
            time.sleep(5)
            continue

        if status == "UPDATE_COMPLETE":
            logger.info(
                "Dataset distribution complete: %s : %s : %s",
                status, status_message, dataset_arn)

```



```
        finished = True
        continue

    if status == "UPDATE_FAILED":
        logger.exception(
            "Dataset distribution failed: %s : %s : %s",
            status, status_message, dataset_arn)
        finished = True
        break

    logger.exception(
        "Failed. Unexpected state for dataset distribution: %s : %s : %s",
        status, status_message, dataset_arn)
    finished = True
    status_message = "An unexpected error occurred while distributing the
dataset"
    break

    return status, status_message

def distribute_dataset_entries(rek_client, training_dataset_arn,
test_dataset_arn):
    """
    Distributes 20% of the supplied training dataset into the supplied test
dataset.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param training_dataset_arn: The ARN of the training dataset that you
distribute entries from.
    :param test_dataset_arn: The ARN of the test dataset that you distribute
entries to.
    """

    try:
        # List dataset labels.
        logger.info("Distributing training dataset entries (%s) into test
dataset (%s).",
                    training_dataset_arn, test_dataset_arn)

        datasets = json.loads(
            '[{"Arn" : "' + str(training_dataset_arn) + '"}, {"Arn" : "' +
str(test_dataset_arn) + '"}]')
```

```
    rek_client.distribute_dataset_entries(
        Datasets=datasets
    )

    training_dataset_status, training_dataset_status_message =
check_dataset_status(
    rek_client, training_dataset_arn)
    test_dataset_status, test_dataset_status_message = check_dataset_status(
        rek_client, test_dataset_arn)

    if training_dataset_status == 'UPDATE_COMPLETE' and test_dataset_status
== "UPDATE_COMPLETE":
        print("Distribution complete")
    else:
        print("Distribution failed:")
        print(
            f"\t\ttraining dataset: {training_dataset_status} :
{training_dataset_status_message}")
        print(
            f"\t\ttest dataset: {test_dataset_status} :
{test_dataset_status_message}")

    except ClientError as err:
        logger.exception(
            "Couldn't distribute dataset: %s",err.response['Error']['Message'] )
        raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "training_dataset_arn", help="The ARN of the training dataset that you
want to distribute from."
    )

    parser.add_argument(
        "test_dataset_arn", help="The ARN of the test dataset that you want to
distribute to."
    )
```

```
def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(
            f"Distributing training dataset entries
            ({args.training_dataset_arn}) "\
            f"into test dataset ({args.test_dataset_arn}).")

        # Distribute the datasets.

        session = boto3.Session(profile_name='custom-labels-access')
        rekognition_client = session.client("rekognition")

        distribute_dataset_entries(rekognition_client,
                                   args.training_dataset_arn,
                                   args.test_dataset_arn)

        print("Finished distributing datasets.")

    except ClientError as err:
        logger.exception("Problem distributing datasets: %s", err)
        print(f"Problem listing dataset labels: {err}")
    except Exception as err:
        logger.exception("Problem distributing datasets: %s", err)
        print(f"Problem distributing datasets: {err}")

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

## Java V2

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `training_dataset_arn` - エントリを分散するトレーニングデータセットの ARN。
- `test_dataset_arn` - エントリを分散する先のテストデータセットの ARN。

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
*/
package com.example.rekognition;

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.DatasetDescription;
import software.amazon.awssdk.services.rekognition.model.DatasetStatus;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.DistributeDataset;
import
    software.amazon.awssdk.services.rekognition.model.DistributeDatasetEntriesRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

public class DistributeDatasetEntries {

    public static final Logger logger =
        Logger.getLogger(DistributeDatasetEntries.class.getName());

    public static DatasetStatus checkDatasetStatus(RekognitionClient rekClient,
        String datasetArn)
        throws Exception, RekognitionException {

        boolean distributed = false;
        DatasetStatus status = null;

        // Wait until distribution completes

        do {
```

```
        DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder().datasetArn(datasetArn)
        .build();
        DescribeDatasetResponse describeDatasetResponse =
rekClient.describeDataset(describeDatasetRequest);

        DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();

        status = datasetDescription.status();

        logger.log(Level.INFO, " dataset ARN: {0} ", datasetArn);

        switch (status) {

        case UPDATE_COMPLETE:
            logger.log(Level.INFO, "Dataset updated");
            distributed = true;
            break;

        case UPDATE_IN_PROGRESS:
            Thread.sleep(5000);
            break;

        case UPDATE_FAILED:
            String error = "Dataset distribution failed: " +
datasetDescription.statusAsString() + " "
                + datasetDescription.statusMessage() + " " + datasetArn;
            logger.log(Level.SEVERE, error);
            break;

        default:
            String unexpectedError = "Unexpected distribution state: " +
datasetDescription.statusAsString() + " "
                + datasetDescription.statusMessage() + " " + datasetArn;
            logger.log(Level.SEVERE, unexpectedError);

        }

    } while (distributed == false);

    return status;
}
```

```
    }

    public static void distributeMyDatasetEntries(RekognitionClient rekClient,
String trainingDatasetArn,
        String testDatasetArn) throws Exception, RekognitionException {

    try {

        logger.log(Level.INFO, "Distributing {0} dataset to {1} ",
            new Object[] { trainingDatasetArn, testDatasetArn });

        DistributeDataset distributeTrainingDataset =
DistributeDataset.builder().arn(trainingDatasetArn).build();

        DistributeDataset distributeTestDataset =
DistributeDataset.builder().arn(testDatasetArn).build();

        ArrayList<DistributeDataset> datasets = new ArrayList();

        datasets.add(distributeTrainingDataset);
        datasets.add(distributeTestDataset);

        DistributeDatasetEntriesRequest distributeDatasetEntriesRequest =
DistributeDatasetEntriesRequest.builder()
            .datasets(datasets).build();

        rekClient.distributeDatasetEntries(distributeDatasetEntriesRequest);

        DatasetStatus trainingStatus = checkDatasetStatus(rekClient,
trainingDatasetArn);
        DatasetStatus testStatus = checkDatasetStatus(rekClient,
testDatasetArn);

        if (trainingStatus == DatasetStatus.UPDATE_COMPLETE && testStatus ==
DatasetStatus.UPDATE_COMPLETE) {
            logger.log(Level.INFO, "Successfully distributed dataset: {0}",
trainingDatasetArn);
        } else {

            throw new Exception("Failed to distribute dataset: " +
trainingDatasetArn);
        }
    }
}
```

```
        } catch (RekognitionException e) {
            logger.log(Level.SEVERE, "Could not distribute dataset: {0}",
e.getMessage());
            throw e;
        }
    }

    public static void main(String[] args) {

        String trainingDatasetArn = null;
        String testDatasetArn = null;

        final String USAGE = "\n" + "Usage: " + "<training_dataset_arn>
<test_dataset_arn>\n\n" + "Where:\n"
            + "    training_dataset_arn - the ARN of the dataset that you
want to distribute from.\n\n"
            + "    test_dataset_arn - the ARN of the dataset that you want to
distribute to.\n\n";

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

        trainingDatasetArn = args[0];
        testDatasetArn = args[1];

        try {

            // Get the Rekognition client.
            RekognitionClient rekClient = RekognitionClient.builder()
                .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
                .region(Region.US_WEST_2)
                .build();

            // Distribute the dataset
            distributeMyDatasetEntries(rekClient, trainingDatasetArn,
testDatasetArn);

            System.out.println("Datasets distributed.");

            rekClient.close();
        }
    }
}
```

```
    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    } catch (Exception rekError) {
        logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());
        System.exit(1);
    }
}
}
```

## データセットの削除

プロジェクトからトレーニングデータセットとテストデータセットを削除できます。

トピック

- [データセットの削除 \(コンソール\)](#)
- [Amazon Rekognition Custom Labels データセットの削除 \(SDK\)](#)

### データセットの削除 (コンソール)

次の手順に従って、データセットを削除します。その後、プロジェクトにデータセット (トレーニングまたはテスト) がある場合は、プロジェクトの詳細ページが表示されます。プロジェクトに残っているデータセットがない場合は、[データセットを作成] ページが表示されます。

トレーニングデータセットを削除した場合、モデルをトレーニングする前に、プロジェクト用に新しいトレーニングデータセットを作成する必要があります。詳細については、「[イメージ付きの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](#)」を参照してください。

テストデータセットを削除すれば、新しいテストデータセットを作成しなくてもモデルをトレーニングできます。トレーニング中、トレーニングデータセットは分割され、プロジェクト用の新しいテストデータセットが作成されます。トレーニングデータセットを分割すると、トレーニングに使用できるイメージの数が減ります。品質を維持するために、モデルをトレーニングする前に新しいテストデータセットを作成することを推奨します。詳細については、「[データセットをプロジェクトに追加する](#)」を参照してください。



## データセットを削除するには

1. Amazon Rekognition コンソールを <https://console.aws.amazon.com/rekognition/> で開きます。
2. 左側のペインで、[カスタムラベルを使用] を選択します。Amazon Rekognition Custom Labels のランディングページが表示されます。
3. 左側のナビゲーションペインで、[Projects] (プロジェクト) を選択します。プロジェクトビューが表示されます。
4. 削除するデータセットが含まれるプロジェクトを選択します。
5. 左側のナビゲーションペインで、プロジェクト名の下にある [データセット] を選択します。
6. [アクション] を選択します。
7. トレーニングデータセットを削除するには、[トレーニングデータセットを削除] を選択します。
8. テストデータセットを削除するには、[テストデータセットを削除] を選択します。
9. [トレーニングまたはテストデータセットを削除] ダイアログボックスで、[削除] と入力し、データセットを削除することを確認します。
10. [トレーニングまたはテストデータセットを削除] を選択して、データセットを削除します。

## Amazon Rekognition Custom Labels データセットの削除 (SDK)

Amazon Rekognition Custom Labels データセットを削除するには、[DeleteDataset](#) を呼び出し、削除するデータセットの Amazon リソースネーム (ARN) を指定します。プロジェクト内のトレーニングデータセットとテストデータセットの ARN を取得するには、[DescribeProjects](#) を呼び出します。レスポンスには [ProjectDescription](#) オブジェクトの配列が含まれます。データセット ARN (DatasetArn) とデータセットタイプ (DatasetType) は、Datasets の一覧にあります。

トレーニングデータセットを削除した場合、モデルをトレーニングする前に、プロジェクト用に新しいトレーニングデータセットを作成する必要があります。テストデータセットを削除した場合、モデルをトレーニングする前に、新しいトレーニングデータセットを作成する必要があります。詳細については、「[データセットをプロジェクトに追加する \(SDK\)](#)」を参照してください。

### データセットを削除するには (SDK)

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. 次のコードを使用してデータセットを削除します。

## AWS CLI

`dataset-arn` の値を削除するデータセットの ARN に変更します。

```
aws rekognition delete-dataset --dataset-arn dataset-arn \  
--profile custom-labels-access
```

## Python

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `dataset_arn` - 削除するデータセットの ARN。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
  
"""  
Purpose  
Shows how to delete an Amazon Rekognition Custom Labels dataset.  
"""  
  
import argparse  
import logging  
import time  
import boto3  
  
from botocore.exceptions import ClientError  
  
logger = logging.getLogger(__name__)  
  
def delete_dataset(rek_client, dataset_arn):  
    """  
    Deletes an Amazon Rekognition Custom Labels dataset.  
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.  
    :param dataset_arn: The ARN of the dataset that you want to delete.  
    """  
  
    try:  
        # Delete the dataset,  
        logger.info("Deleting dataset: %s", dataset_arn)
```

```
rek_client.delete_dataset(DatasetArn=dataset_arn)

deleted = False

logger.info("waiting for dataset deletion %s", dataset_arn)

# Dataset might not be deleted yet, so wait.
while deleted is False:
    try:
        rek_client.describe_dataset(DatasetArn=dataset_arn)
        time.sleep(5)
    except ClientError as err:
        if err.response['Error']['Code'] == 'ResourceNotFoundException':
            logger.info("dataset deleted: %s", dataset_arn)
            deleted = True
        else:
            raise

logger.info("dataset deleted: %s", dataset_arn)

return True

except ClientError as err:
    logger.exception("Couldn't delete dataset - %s: %s",
                    dataset_arn, err.response['Error']['Message'])
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "dataset_arn", help="The ARN of the dataset that you want to delete."
    )

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")
```

```
try:

    # Get command line arguments.
    parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
    add_arguments(parser)
    args = parser.parse_args()

    print(f"Deleting dataset: {args.dataset_arn}")

    # Delete the dataset.
    session = boto3.Session(profile_name='custom-labels-access')
    rekognition_client = session.client("rekognition")

    delete_dataset(rekognition_client,
                   args.dataset_arn)

    print(f"Finished deleting dataset: {args.dataset_arn}")

except ClientError as err:
    error_message = f"Problem deleting dataset: {err}"
    logger.exception(error_message)
    print(error_message)

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

## Java V2

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `dataset_arn` - 削除するデータセットの ARN。

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
*/
package com.example.rekognition;

import java.util.logging.Level;
import java.util.logging.Logger;
```

```
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.DeleteDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.DeleteDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

public class DeleteDataset {

    public static final Logger logger =
        Logger.getLogger(DeleteDataset.class.getName());

    public static void deleteMyDataset(RekognitionClient rekClient, String
datasetArn) throws InterruptedException {

        try {

            logger.log(Level.INFO, "Deleting dataset: {0}", datasetArn);

            // Delete the dataset

            DeleteDatasetRequest deleteDatasetRequest =
DeleteDatasetRequest.builder().datasetArn(datasetArn).build();

            DeleteDatasetResponse response =
rekClient.deleteDataset(deleteDatasetRequest);

            // Wait until deletion finishes

            DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder().datasetArn(datasetArn)
                .build();

            Boolean deleted = false;

            do {

                try {

                    rekClient.describeDataset(describeDatasetRequest);
                    Thread.sleep(5000);
                } catch (RekognitionException e) {
                    String errorCode = e.awsErrorDetails().errorCode();
```

```
        if (errorCode.equals("ResourceNotFoundException")) {
            logger.log(Level.INFO, "Dataset deleted: {0}",
datasetArn);
            deleted = true;
        } else {
            logger.log(Level.SEVERE, "Client error occurred: {0}",
e.getMessage());
            throw e;
        }
    }

    } while (Boolean.FALSE.equals(deleted));

    logger.log(Level.INFO, "Dataset deleted: {0} ", datasetArn);

} catch (
    RekognitionException e) {
    logger.log(Level.SEVERE, "Client error occurred: {0}",
e.getMessage());
    throw e;
}

}

public static void main(String args[]) {

    final String USAGE = "\n" + "Usage: " + "<dataset_arn>\n\n" + "Where:\n"
        + "    dataset_arn - The ARN of the dataset that you want to
delete.\n\n";

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

    String datasetArn = args[0];

    try {

        // Get the Rekognition client.
        RekognitionClient rekClient = RekognitionClient.builder()
```

```
        .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
        .region(Region.US_WEST_2)
        .build();

    // Delete the dataset
    deleteMyDataset(rekClient, datasetArn);

    System.out.println(String.format("Dataset deleted: %s",
datasetArn));

    rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    }

    catch (InterruptedException intError) {
        logger.log(Level.SEVERE, "Exception while sleeping: {0}",
intError.getMessage());
        System.exit(1);
    }

    }

}
```

## Amazon Rekognition Custom Labels モデルの管理

Amazon Rekognition Custom Labels モデルは、新しいイメージに含まれるオブジェクト、シーン、概念の存在を予測する数学モデルです。これは、モデルのトレーニングに使用したイメージのパターンを検知することにより行われます。このセクションでは、モデルをトレーニングし、そのパフォーマンスを評価して改善する方法を説明します。また、モデルを使用可能にする方法と、不要になったモデルを削除する方法について説明します。

### トピック

- [Amazon Rekognition Custom Labels モデルの削除](#)

- [モデルのタグ付け](#)
- [モデルの記述 \(SDK\)](#)
- [Amazon Rekognition Custom Labels モデルのコピー \(SDK\)](#)

## Amazon Rekognition Custom Labels モデルの削除

Amazon Rekognition Custom Labels コンソールを使用するか、[DeleteProjectVersion API](#) を使用してモデルを削除できます。実行中またはトレーニング中の場合は、モデルを削除できません。実行中のモデルを停止するには、[StopProjectVersion API](#) を使用します。詳細については、「[Amazon Rekognition Custom Labels モデル \(SDK\) の停止](#)」を参照してください。モデルがトレーニング中の場合は、モデルを削除する前に終了するまで待ちます。

削除したモデルを元に戻すことはできません。

### トピック

- [Amazon Rekognition Custom Labels モデルの削除 \(コンソール\)](#)
- [Amazon Rekognition Custom Labels モデルの削除 \(SDK\)](#)

## Amazon Rekognition Custom Labels モデルの削除 (コンソール)

次の手順では、プロジェクトの詳細ページからモデルを削除する方法を示しています。モデルの詳細ページからモデルを削除することもできます。

### モデルを削除するには (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. [カスタムラベルを使用] を選択します。
3. [Get started] (開始方法) を選択します。
4. 左側のナビゲーションペインで、[Projects] (プロジェクト) を選択します。
5. 削除するモデルを含むプロジェクトを選択します。プロジェクトの詳細ページが開きます。
6. [モデル] セクションで、削除するモデルを選択します。



**Note**

モデルを選択できない場合、モデルは実行中またはトレーニング中であるため削除できません。[ステータス] フィールドを確認して、実行中のモデルを停止してから再試行するか、トレーニングが終了するまで待ちます。

7. [モデルを削除] を選択すると、[モデルを削除] ダイアログボックスが表示されます。
8. [delete] と入力して、削除を確定します。
9. [削除] を選択してモデルを削除します。モデルの削除が完了するまでに時間がかかる場合があります。

**Note**

モデルの削除中にダイアログボックスを閉じた場合でも、モデルは削除されます。

## Amazon Rekognition Custom Labels モデルの削除 (SDK)

Amazon Rekognition Custom Labels モデルを削除するには、[deleteProjectVersion](#) を呼び出し、削除するモデルの Amazon リソースネーム (ARN) を指定します。モデル ARN は、Amazon Rekognition Custom Labels コンソールのモデルの詳細ページの [モデルを使用する] セクションから取得できます。または、[DescribeProjectVersions](#) を呼び出して、次のものを指定します。

- ジョブが関連付けられているプロジェクト (ProjectArn) の ARN。
- モデルのバージョン名 (VersionNames)。

モデル ARN は、[DescribeProjectVersions](#) からのレスポンスで取得した [ProjectVersionDescription](#) オブジェクト内の ProjectVersionArn フィールドです。

実行中またはトレーニング中の場合は、モデルを削除できません。モデルが実行中またはトレーニング中かを判断するには、[DescribeProjectVersions](#) を呼び出し、モデルの [ProjectVersionDescription](#) オブジェクトの Status フィールドを確認します。実行中のモデルを停止するには、[StopProjectVersion](#) API を使用します。詳細については、「[Amazon Rekognition Custom Labels モデル \(SDK\) の停止](#)」を参照してください。モデルを削除するには、トレーニングが終了するまで待つ必要があります。

## モデルを削除するには (SDK)

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. モデルを削除するには、次のコードを使用します。

### AWS CLI

`project-version-arn` の値を削除するプロジェクトの名前に変更します。

```
aws rekognition delete-project-version --project-version-arn model_arn \  
--profile custom-labels-access
```

### Python

次のコマンドラインパラメータを指定します

- `project_arn` - 削除するモデルを含むプロジェクトの ARN。
- `model_arn` - 削除するモデルのバージョンの ARN。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
  
"""  
Purpose  
Shows how to delete an existing Amazon Rekognition Custom Labels model.  
"""  
  
import argparse  
import logging  
import time  
import boto3  
from botocore.exceptions import ClientError  
  
logger = logging.getLogger(__name__)  
  
def find_forward_slash(input_string, n):  
    """
```

```
Returns the location of '/' after n number of occurrences.
:param input_string: The string you want to search
: n: the occurrence that you want to find.
"""
position = input_string.find('/')
while position >= 0 and n > 1:
    position = input_string.find('/', position + 1)
    n -= 1
return position

def delete_model(rek_client, project_arn, model_arn):
    """
    Deletes an Amazon Rekognition Custom Labels model.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param model_arn: The ARN of the model version that you want to delete.
    """

    try:
        # Delete the model
        logger.info("Deleting dataset: {%s}", model_arn)

        rek_client.delete_project_version(ProjectVersionArn=model_arn)

        # Get the model version name
        start = find_forward_slash(model_arn, 3) + 1
        end = find_forward_slash(model_arn, 4)
        version_name = model_arn[start:end]

        deleted = False

        # model might not be deleted yet, so wait deletion finishes.
        while deleted is False:
            describe_response =
rek_client.describe_project_versions(ProjectArn=project_arn,
VersionNames=[version_name])
            if len(describe_response['ProjectVersionDescriptions']) == 0:
                deleted = True
            else:
                logger.info("Waiting for model deletion %s", model_arn)
                time.sleep(5)

        logger.info("model deleted: %s", model_arn)
```

```
        return True

    except ClientError as err:
        logger.exception("Couldn't delete model - %s: %s",
                        model_arn, err.response['Error']['Message'])
        raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The ARN of the project that contains the model that
you want to delete."
    )

    parser.add_argument(
        "model_arn", help="The ARN of the model version that you want to
delete."
    )

def confirm_model_deletion(model_arn):
    """
    Confirms deletion of the model. Returns True if delete entered.
    :param model_arn: The ARN of the model that you want to delete.
    """
    print(f"Are you sure you wany to delete model {model_arn} ?\n", model_arn)

    start = input("Enter delete to delete your model: ")
    if start == "delete":
        return True
    else:
        return False

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")
```

```
try:

    # Get command line arguments.
    parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
    add_arguments(parser)
    args = parser.parse_args()

    if confirm_model_deletion(args.model_arn) is True:
        print(f"Deleting model: {args.model_arn}")

        # Delete the model.
        session = boto3.Session(profile_name='custom-labels-access')
        rekognition_client = session.client("rekognition")

        delete_model(rekognition_client,
                     args.project_arn,
                     args.model_arn)

        print(f"Finished deleting model: {args.model_arn}")
    else:
        print(f"Not deleting model {args.model_arn}")

except ClientError as err:
    print(f"Problem deleting model: {err}")

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

## Java V2

- `project_arn` - 削除するモデルを含むプロジェクトの ARN。
- `model_arn` - 削除するモデルのバージョンの ARN。

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/
awsdocs/amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-
SAMPLECODE.)

import java.net.URI;
import java.util.logging.Level;
```

```
import java.util.logging.Logger;

import software.amazon.awssdk.services.rekognition.RekognitionClient;

import
    software.amazon.awssdk.services.rekognition.model.DeleteProjectVersionRequest;
import
    software.amazon.awssdk.services.rekognition.model.DeleteProjectVersionResponse;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

public class DeleteModel {

    public static final Logger logger =
        Logger.getLogger(DeleteModel.class.getName());

    public static int findForwardSlash(String modelArn, int n) {

        int start = modelArn.indexOf('/');
        while (start >= 0 && n > 1) {
            start = modelArn.indexOf('/', start + 1);
            n -= 1;
        }
        return start;
    }

    public static void deleteMyModel(RekognitionClient rekClient, String
projectArn, String modelArn)
        throws InterruptedException {

        try {

            logger.log(Level.INFO, "Deleting model: {0}", projectArn);

            // Delete the model

            DeleteProjectVersionRequest deleteProjectVersionRequest =
DeleteProjectVersionRequest.builder()
                .projectVersionArn(modelArn).build();
```

```
DeleteProjectVersionResponse response =
    rekClient.deleteProjectVersion(deleteProjectVersionRequest);

logger.log(Level.INFO, "Status: {0}", response.status());

// Get the model version

int start = findForwardSlash(modelArn, 3) + 1;
int end = findForwardSlash(modelArn, 4);

String versionName = modelArn.substring(start, end);

Boolean deleted = false;

DescribeProjectVersionsRequest describeProjectVersionsRequest =
DescribeProjectVersionsRequest.builder()
    .projectArn(projectArn).versionNames(versionName).build();

// Wait until model is deleted.

do {

    DescribeProjectVersionsResponse describeProjectVersionsResponse
= rekClient

.describeProjectVersions(describeProjectVersionsRequest);

    if
(describeProjectVersionsResponse.projectVersionDescriptions().size()==0) {
        logger.log(Level.INFO, "Waiting for model deletion: {0}",
modelArn);
        Thread.sleep(5000);
    } else {
        deleted = true;
        logger.log(Level.INFO, "Model deleted: {0}", modelArn);
    }

} while (Boolean.FALSE.equals(deleted));

logger.log(Level.INFO, "Model deleted: {0}", modelArn);

} catch (

RekognitionException e) {
```

```
        logger.log(Level.SEVERE, "Client error occurred: {0}",
e.getMessage());
        throw e;
    }

}

public static void main(String args[]) {

    final String USAGE = "\n" + "Usage: " + "<project_arn> <model_arn>\n\n"
+ "Where:\n"
        + "    project_arn - The ARN of the project that contains the
model that you want to delete.\n\n"
        + "    model_version - The ARN of the model that you want to
delete.\n\n";

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

    String projectArn = args[0];
    String modelVersion = args[1];

    try {

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

        // Delete the model
        deleteMyModel(rekClient, projectArn, modelVersion);

        System.out.println(String.format("model deleted: %s",
modelVersion));

        rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    }

    catch (InterruptedException intError) {
```



```
        logger.log(Level.SEVERE, "Exception while sleeping: {0}",
intError.getMessage());
        System.exit(1);
    }

}

}
```

## モデルのタグ付け

タグを使用して、Amazon Rekognition モデルを識別、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。例えば、モデルの請求額を決定しやすくするために、モデルに Cost center キーでタグ付けし、適切なコストセンター番号を値として追加します。詳細については、[「AWS リソースにタグ付けする」](#)を参照してください。

タグを使用して、次の操作を行います。

- コスト配分タグを使用して、モデルの課金の追跡をします。詳細については、[「コスト配分タグの使用」](#)を参照してください。
- IAM (Identity and Access Management) を用いて、モデルへのアクセスを制御します。詳細については、[リソースタグを使って AWS リソースへアクセスの制御](#)を参照します。
- モデル管理を自動化します。例えば、業務時間外に開発モデルを停止する自動起動または停止スクリプトを実行し、コストを削減することができます。詳細については、「[トレーニング済み Amazon Rekognition Custom Labels の実行](#)」を参照してください。

Amazon Rekognition コンソールを使用するか、AWS SDK を使用することにより、モデルにタグ付けることができます。

### トピック

- [モデルのタグ付け \(コンソール\)](#)
- [モデルのタグの表示](#)
- [モデルのタグ付け \(SDK\)](#)

## モデルのタグ付け (コンソール)

Rekognition コンソールを使用して、モデルへのタグの追加、モデルに追加されたタグの表示、タグの削除を行うことができます。

### タグの追加または削除

この手順では、既存のモデルにタグを追加する方法、または既存のモデルからタグを削除する方法について説明します。トレーニング中に新しいモデルにタグを追加することもできます。詳細については、「[Amazon Rekognition Custom Labels モデルをトレーニングする](#)」を参照してください。

コンソールを使用して既存のモデルにタグを追加または削除するには

1. Amazon Rekognition コンソールを <https://console.aws.amazon.com/rekognition/> で開きます。
2. [Get started] (開始方法) を選択します。
3. ナビゲーションペインで、[Projects] (プロジェクト) を選択します。
4. [Projects] (プロジェクト) リソースページで、タグ付けしたいモデルを含むプロジェクトを選択します。
5. ナビゲーションペインの、以前に選択したプロジェクトの下で、[Models] (モデル) を選択します。
6. [Models] (モデル) セクションで、タグを追加するモデルを選択します。
7. モデルの詳細ページで、[Tags] (タグ) タブを選択します。
8. [Tags] (タグ) タブで、[Manage tags] (タグの管理) を選択します。
9. [Manage tags] (タグの管理) ページで、[Add new tag] (タグの追加) を選択します。
10. キーと値を入力します。
  - a. [Key] (キー) に、キーの名前を入力します。
  - b. [Value] (値) に値を入力します。
11. さらにタグを追加するには、手順 9~10 を繰り返します。
12. (オプション) タグを削除するには、削除したいタグの横にある [Remove] (削除) を選択します。以前に保存したタグを削除する場合、変更を保存するとそのタグが削除されます。
13. [Save changes] (変更の保存) を選択して、変更を保存します。

### モデルのタグの表示

Amazon Rekognition コンソールを使用して、モデルに追加されているタグを表示できます。

プロジェクト内のすべてのモデルに追加されたタグを表示するには、AWS SDK を使用する必要があります。詳細については、「[モデルタグの一覧表示](#)」を参照してください。

モデルに追加されたタグを表示するには

1. Amazon Rekognition コンソールを <https://console.aws.amazon.com/rekognition/> で開きます。
2. [Get started] (開始方法) を選択します。
3. ナビゲーションペインで、[Projects] (プロジェクト) を選択します。
4. [Projects] (プロジェクト) リソースページで、タグを表示したいモデルを含むプロジェクトを選択します。
5. ナビゲーションペインの、以前に選択したプロジェクトの下で、[Models] (モデル) を選択します。
6. [Models] (モデル) セクションで、タグを表示するモデルを選択します。
7. モデルの詳細ページで、[Tags] (タグ) タブを選択します。タグは [Tags] (タグ) セクションに示されています。

## モデルのタグ付け (SDK)

AWS SDKを使用することで、以下のことが可能になります。

- 新しいモデルにタグを追加する
- 既存のモデルにタグを追加する
- モデルに追加されたタグを一覧表示する
- モデルからタグを削除する

次の AWS CLI の例のタグ形式は次のとおりです。

```
--tags '{"key1":"value1","key2":"value2"}
```

または、この形式を使用することもできます。

```
--tags key1=value1,key2=value2
```

AWS CLI をまだインストールしていない場合は、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。

## 新しいモデルへのタグの追加

[CreateProjectVersion](#) オペレーションを使用することにより、作成時に、モデルにタグを追加することもできます。Tags 配列入力パラメータで 1 つ以上のタグを指定します。

```
aws rekognition create-project-version --project-arn project_arn \  
  --version-name version_name \  
  --output-config '{ "S3Location": { "Bucket": "output_bucket", "Prefix": "output_folder" } }' \  
  --tags '{"key1":"value1","key2":"value2"}' \  
  --profile custom-labels-access
```

モデルの作成とトレーニングの詳細については、「[モデルのトレーニング \(SDK\)](#)」を参照してください。

## 既存のモデルへのタグの追加

既存のモデルに 1 つ以上のタグを追加するには、[TagResource](#) (タグリソース) オペレーションを使用します。モデルの 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"}' \  
  --profile custom-labels-access
```

[CreateProjectVersion](#) を呼び出すことにより、モデルの ARN を取得できます。

## モデルタグの一覧表示

モデルに追加されたタグを一覧表示するには、[ListTagsForResource](#) オペレーションを使用して、モデルの ARN (ResourceArn) を指定します。応答は、指定されたモデルに追加されているタグキーと値のマップです。

```
aws rekognition list-tags-for-resource --resource-arn resource-arn \  
  --profile custom-labels-access
```

出力には、モデル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"
  }
}
```

プロジェクト内のどのモデルに特定のタグがあるかを確認するには、DescribeProjectVersions でモデルのリストを取得します。次に、DescribeProjectVersions からの応答で各モデルに対して ListTagsForResource を呼び出します。ListTagsForResource からのレスポンスを検査して、必要なタグが存在するかどうかを確認します。

次の Python 3 の例では、すべてのプロジェクトで特定のタグキーと値を検索する方法を示しています。出力には、プロジェクト ARN と、一致するキーが検知されたモデル ARN が含まれます。

タグの値を検索するには

1. 次のコードを find\_tag.py という名前を付けて保存します。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Purpose
Shows how to find a tag value that's associated with models within
your Amazon Rekognition Custom Labels projects.
"""
import logging
import argparse
import boto3

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def find_tag_in_projects(rekognition_client, key, value):
    """
    Finds Amazon Rekognition Custom Label models tagged with the supplied key and
    key value.
    :param rekognition_client: An Amazon Rekognition boto3 client.
    :param key: The tag key to find.
    :param value: The value of the tag that you want to find.
    :return: A list of matching model versions (and model projects) that were found.
    """
    try:
```

```
found_tags = []
found = False

projects = rekognition_client.describe_projects()
# Iterate through each project and models within a project.
for project in projects["ProjectDescriptions"]:
    logger.info("Searching project: %s ...", project["ProjectArn"])

    models = rekognition_client.describe_project_versions(
        ProjectArn=(project["ProjectArn"])
    )

    for model in models["ProjectVersionDescriptions"]:
        logger.info("Searching model %s", model["ProjectVersionArn"])

        tags = rekognition_client.list_tags_for_resource(
            ResourceArn=model["ProjectVersionArn"]
        )

        logger.info(
            "\tSearching model: %s for tag: %s value: %s.",
            model["ProjectVersionArn"],
            key,
            value,
        )
        # Check if tag exists.

        if key in tags["Tags"]:
            if tags["Tags"][key] == value:
                found = True
                logger.info(
                    "\t\tMATCH: Project: %s: model version %s",
                    project["ProjectArn"],
                    model["ProjectVersionArn"],
                )
                found_tags.append(
                    {
                        "Project": project["ProjectArn"],
                        "ModelVersion": model["ProjectVersionArn"],
                    }
                )

    if found is False:
```

```
        logger.info("No match for Tag %s with value %s.", key, value)
    return found_tags
except ClientError as err:
    logger.info("Problem finding tags: %s. ", format(err))
    raise

def main():
    """
    Entry point for example.
    """
    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    # Set up command line arguments.
    parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)

    parser.add_argument("tag", help="The tag that you want to find.")
    parser.add_argument("value", help="The tag value that you want to find.")

    args = parser.parse_args()
    key = args.tag
    value = args.value

    print(f"Searching your models for tag: {key} with value: {value}.")

    session = boto3.Session(profile_name='custom-labels-access')
    rekognition_client = session.client("rekognition")

    # Get tagged models for all projects.
    tagged_models = find_tag_in_projects(rekognition_client, key, value)

    print("Matched models\n-----")
    if len(tagged_models) > 0:
        for model in tagged_models:
            print(
                "Project: {project}\nModel version: {version}\n".format(
                    project=model["Project"], version=model["ModelVersion"]
                )
            )
    else:
        print("No matches found.")
```

```
print("Done.")

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

2. コマンドプロンプトで次のコマンドを入力します。*[key]* と *[value]* を検索するキー名とキー値に置き換えます。

```
python find_tag.py key value
```

## モデルからタグを削除する

1 つ以上のタグを削除するには、[\[UntagResource\]](#) (アンタグリソース) オペレーションを使用します。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"]' \  
--profile custom-labels-access
```

または、次の形式で tag-keys を指定できます。

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

## モデルの記述 (SDK)

DescribeProjectVersions API を使用して、モデルのバージョンに関する情報を取得できます。VersionName を指定しないと、DescribeProjectVersions はプロジェクト内のすべてのモデルバージョンの記述を返します。

モデルを記述するには (SDK)

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. 次のサンプルコードを使用して、モデルのバージョンを記述します。



## AWS CLI

`project-arn` の値を、記述するプロジェクトの ARN に変更します。`version-name` の値を、記述するモデルのバージョンに変更します。

```
aws rekognition describe-project-versions --project-arn project_arn \  
  --version-names version_name \  
  --profile custom-labels-access
```

## Python

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `project_arn` - 記述するモデルの ARN。
- `model_version` - 記述するモデルのバージョン。

例: `python describe_model.py project_arn model_version`

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
  
"""  
Purpose  
Shows how to describe an Amazon Rekognition Custom Labels model.  
"""  
  
import argparse  
import logging  
import boto3  
  
from botocore.exceptions import ClientError  
  
logger = logging.getLogger(__name__)  
  
def describe_model(rek_client, project_arn, version_name):  
    """  
    Describes an Amazon Rekognition Custom Labels model.  
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.  
    :param project_arn: The ARN of the project that contains the model.  
    :param version_name: The version name of the model that you want to  
    describe.
```

```
"""

try:
    # Describe the model
    logger.info("Describing model: %s for project %s",
                version_name, project_arn)

    describe_response =
rek_client.describe_project_versions(ProjectArn=project_arn,
VersionNames=[version_name])
    for model in describe_response['ProjectVersionDescriptions']:
        print(f"Created: {str(model['CreationTimestamp'])} ")
        print(f"ARN: {str(model['ProjectVersionArn'])} ")
        if 'BillableTrainingTimeInSeconds' in model:
            print(
                f"Billing training time (minutes):
{str(model['BillableTrainingTimeInSeconds']/60)} ")
            print("Evaluation results: ")
            if 'EvaluationResult' in model:
                evaluation_results = model['EvaluationResult']
                print(f"\tF1 score: {str(evaluation_results['F1Score'])}")
                print(
                    f"\tSummary location: s3://{evaluation_results['Summary']
['S3object']['Bucket']}/{evaluation_results['Summary']['S3object']['Name']}")

            if 'ManifestSummary' in model:
                print(
                    f"Manifest summary location: s3://{model['ManifestSummary']
['S3object']['Bucket']}/{model['ManifestSummary']['S3object']['Name']}")
                if 'OutputConfig' in model:
                    print(
                        f"Training output location: s3://{model['OutputConfig']
['S3Bucket']}/{model['OutputConfig']['S3KeyPrefix']}")
                    if 'MinInferenceUnits' in model:
                        print(
                            f"Minimum inference units:
{str(model['MinInferenceUnits'])}")
                        if 'MaxInferenceUnits' in model:
                            print(
                                f"Maximum Inference units:
{str(model['MaxInferenceUnits'])}")

                    print("Status: " + model['Status'])
```

```
        print("Message: " + model['StatusMessage'])

    except ClientError as err:
        logger.exception(
            "Couldn't describe model: %s", err.response['Error']['Message'])
        raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The ARN of the project in which the model resides."
    )
    parser.add_argument(
        "version_name", help="The version of the model that you want to
describe."
    )

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(
            f"Describing model: {args.version_name} for project
{args.project_arn}.")

        # Describe the model.
        session = boto3.Session(profile_name='custom-labels-access')
        rekognition_client = session.client("rekognition")

        describe_model(rekognition_client, args.project_arn,
```

```
        args.version_name)

    print(
        f"Finished describing model: {args.version_name} for project
{args.project_arn}.")

    except ClientError as err:
        error_message = f"Problem describing model: {err}"
        logger.exception(error_message)
        print(error_message)
    except Exception as err:
        error_message = f"Problem describing model: {err}"
        logger.exception(error_message)
        print(error_message)

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

## Java V2

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `project_arn` - 記述するモデルの ARN。
- `model_version` - 記述するモデルのバージョン。

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
*/

package com.example.rekognition;

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.DescribeProjectVersionsRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsResponse;
import software.amazon.awssdk.services.rekognition.model.EvaluationResult;
import software.amazon.awssdk.services.rekognition.model.GroundTruthManifest;
```

```
import software.amazon.awssdk.services.rekognition.model.OutputConfig;
import
    software.amazon.awssdk.services.rekognition.model.ProjectVersionDescription;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

import java.util.logging.Level;
import java.util.logging.Logger;

public class DescribeModel {

    public static final Logger logger =
        Logger.getLogger(DescribeModel.class.getName());

    public static void describeMyModel(RekognitionClient rekClient, String
        projectArn, String versionName) {

        try {

            // If a single version name is supplied, build request argument

            DescribeProjectVersionsRequest describeProjectVersionsRequest =
                null;

            if (versionName == null) {
                describeProjectVersionsRequest =
                    DescribeProjectVersionsRequest.builder().projectArn(projectArn)
                        .build();
            } else {
                describeProjectVersionsRequest =
                    DescribeProjectVersionsRequest.builder().projectArn(projectArn)
                        .versionNames(versionName).build();
            }

            DescribeProjectVersionsResponse describeProjectVersionsResponse =
                rekClient
                    .describeProjectVersions(describeProjectVersionsRequest);

            for (ProjectVersionDescription projectVersionDescription :
                describeProjectVersionsResponse
                    .projectVersionDescriptions()) {

                System.out.println("ARN: " +
                    projectVersionDescription.projectVersionArn());
            }
        }
    }
}
```

```
        System.out.println("Status: " +
projectVersionDescription.statusAsString());
        System.out.println("Message: " +
projectVersionDescription.statusMessage());

        if (projectVersionDescription.billableTrainingTimeInSeconds() !=
null) {
            System.out.println(
                "Billable minutes: " +
(projectVersionDescription.billableTrainingTimeInSeconds() / 60));
        }

        if (projectVersionDescription.evaluationResult() != null) {
            EvaluationResult evaluationResult =
projectVersionDescription.evaluationResult();

            System.out.println("F1 Score: " +
evaluationResult.f1Score());
            System.out.println("Summary location: s3://" +
evaluationResult.summary().s3object().bucket() + "/"
                + evaluationResult.summary().s3object().name());
        }

        if (projectVersionDescription.manifestSummary() != null) {
            GroundTruthManifest manifestSummary =
projectVersionDescription.manifestSummary();
            System.out.println("Manifest summary location: s3://" +
manifestSummary.s3object().bucket() + "/"
                + manifestSummary.s3object().name());
        }

        if (projectVersionDescription.outputConfig() != null) {
            OutputConfig outputConfig =
projectVersionDescription.outputConfig();
            System.out.println(
                "Training output: s3://" + outputConfig.s3Bucket() +
"/" + outputConfig.s3KeyPrefix());
        }

        if (projectVersionDescription.minInferenceUnits() != null) {
            System.out.println("Min inference units: " +
projectVersionDescription.minInferenceUnits());
        }
    }
}
```

```
        System.out.println();

    }

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        throw rekError;
    }

}

public static void main(String args[]) {

    String projectArn = null;
    String versionName = null;

    final String USAGE = "\n" + "Usage: " + "<project_arn> <version_name>\n
\n" + "Where:\n"
        + "    project_arn - The ARN of the project that contains the
models you want to describe.\n\n"
        + "    version_name - (optional) The version name of the model
that you want to describe. \n\n"
        + "                                If you don't specify a value, all model
versions are described.\n\n";

    if (args.length > 2 || args.length == 0) {
        System.out.println(USAGE);
        System.exit(1);
    }

    projectArn = args[0];

    if (args.length == 2) {
        versionName = args[1];
    }

    try {

        // Get the Rekognition client.
        RekognitionClient rekClient = RekognitionClient.builder()
            .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
```

```
        .region(Region.US_WEST_2)
        .build();

        // Describe the model
        describeMyModel(rekClient, projectArn, versionName);

        rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    }
}
}
```

## Amazon Rekognition Custom Labels モデルのコピー (SDK)

[CopyProjectVersion](#) オペレーションを使用して、Amazon Rekognition Custom Labels モデルバージョンをソース Amazon Rekognition Custom Labels プロジェクトから目的のプロジェクトにコピーできます。コピー先のプロジェクトは別の AWS アカウントにあっても、同じ AWS アカウントにあっても構いません。一般的なシナリオは、テスト済みモデルを開発 AWS アカウントから本番稼働 AWS アカウントにコピーすることです。

または、コピー先アカウントのモデルをソースデータセットでトレーニングすることもできます。CopyProjectVersion オペレーションを使用する利点は次のとおりです。

- モデルの動作が一貫しています。モデルのトレーニングは非決定的であり、同じデータセットでトレーニングされた 2 つのモデルが同じ予測を行うとは限りません。CopyProjectVersion を使用してモデルをコピーすると、コピーしたモデルの動作がソースモデルと一致していることを確認でき、モデルを再テストする必要がなくなります。
- モデルのトレーニングは不要です。これにより、モデルのトレーニングが成功した場合にそれぞれ料金が発生するため、コストを節約できます。

モデルを別の AWS アカウントにコピーするには、コピー先の AWS アカウントに Amazon Rekognition Custom Labels プロジェクトが必要です。プロジェクトの作成の詳細については、「[ブ](#)



[プロジェクトの作成](#)」を参照してください。必ず、コピー先の AWS アカウントにプロジェクトを作成してください。

[プロジェクトポリシー](#)は、コピーするモデルバージョンのコピーのアクセス許可を設定するリソースベースのポリシーです。コピー先のプロジェクトがソースプロジェクトと異なる AWS アカウントにある場合は、[プロジェクトポリシー](#)を使用する必要があります。

同じアカウント内でモデルバージョンをコピーする場合、[プロジェクトポリシー](#)を使用する必要はありません。ただし、これらのリソース管理を強化する場合は、アカウント間のプロジェクトで[プロジェクトポリシー](#)を使用することもできます。

[PutProjectPolicy](#) オペレーションを呼び出して、プロジェクトポリシーをソースプロジェクトにアタッチします。

[CopyProjectVersion](#) を使用してモデルを異なる AWS リージョンのプロジェクトにコピーすることはできません。また、Amazon Rekognition Custom Labels コンソールを使用してモデルをコピーすることはできません。このような場合は、ソースモデルのトレーニングに使用したデータセットを使用して、コピー先プロジェクトのモデルをトレーニングできます。詳細については、「[Amazon Rekognition Custom Labels モデルをトレーニングする](#)」を参照してください。

ソースプロジェクトから目的のプロジェクトにモデルをコピーするには、次の操作を行います。

モデルをコピーするには

1. [プロジェクトポリシードキュメントを作成します](#)。
2. [プロジェクトポリシーをソースプロジェクトにアタッチします](#)。
3. [CopyProjectVersion オペレーションを含むモデルをコピーします](#)。

プロジェクトからプロジェクトポリシーを削除するには、[DeleteProjectPolicy](#) を呼び出します。プロジェクトにアタッチされているプロジェクトポリシーのリストを取得するには、[ListProjectPolicies](#) を呼び出します。

トピック

- [プロジェクトポリシードキュメントの作成](#)
- [プロジェクトポリシーのアタッチ \(SDK\)](#)
- [モデルのコピー \(SDK\)](#)
- [プロジェクトポリシーの一覧表示 \(SDK\)](#)
- [プロジェクトポリシーの削除 \(SDK\)](#)

## プロジェクトポリシードキュメントの作成

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のポリシーを使用して、モデルバージョンのコピーのアクセス許可を管理します。プロジェクトポリシーは JSON 形式のドキュメントです。

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 [プリンシパル](#) アクセス許可を許可または拒否します。コピー先プロジェクトが別の AWS アカウントにある場合は、プロジェクトポリシーが必要です。目的のプロジェクトがソースプロジェクトと同じ AWS アカウントにあり、特定のモデルバージョンへのアクセスを制限する場合も同様です。例えば、AWS アカウント内の特定の IAM ロールに対するコピーのアクセス許可を拒否する場合があります。

次の例では、プリンシパル `arn:aws:iam::111111111111: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` をコピーすることを許可しています。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:role/Admin"
      },
      "Action": "rekognition:CopyProjectVersion",
      "Resource": "arn:aws:rekognition:us-east-1:111111111111:project/my_project/
version/test_1/1627045542080"
    }
  ]
}
```

### Note

Action、Resource、Principal、および Effect は、プロジェクトポリシードキュメントの必須フィールドです。

サポートされている action は `rekognition:CopyProjectVersion` のみです。

NotAction、NotResource、および NotPrincipal は禁止フィールドであり、プロジェクトポリシードキュメントには含めないでください。

プロジェクトポリシーを指定しない場合でも、ソースプロジェクトと同じ AWS アカウントにあるプリンシパルが、AmazonRekognitionCustomLabelsFullAccess などの ID ベースのポリシーで CopyProjectVersion を呼び出すアクセス許可を与えている場合は、モデルをコピーできます。

次の手順では、[プロジェクトポリシーのアタッチ \(SDK\)](#) の Python 例で利用できるプロジェクトポリシードキュメントファイルを作成します。put-project-policy AWS CLI コマンドを使用する場合は、プロジェクトポリシーを JSON 文字列として指定します。

プロジェクトポリシードキュメントを作成するには

1. テキストエディタで、次のドキュメントを作成します。以下の値を変更します。

- Effect - コピーのアクセス許可を付与するように ALLOW を指定します。DENY にコピーのアクセス許可を拒否するように指定します。
- Principal - Resource で指定したモデルバージョンへのアクセスを許可または拒否するプリンシパルに送信します。例えば、別の AWS アカウントの [AWS アカウントプリンシパル](#) を指定できます。利用できるプリンシパルには制限がありません。詳細については、「[プリンシパルの指定](#)」を参照してください。
- Resource - アクセス許可を取得するリソースの Amazon リソースネーム (ARN)。ソースプロジェクト内のすべてのモデルバージョンにアクセス許可を付与する場合は、次の形式 `arn:aws:rekognition:region:account:project/source project/version/*` を使用してください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "ALLOW or DENY",
      "Principal": {
        "AWS": "principal"
      },
      "Action": "rekognition:CopyProjectVersion",
      "Resource": "Model version ARN"
    }
  ]
}
```

2. プロジェクトポリシーをコンピュータにダウンロードします。

3. 「[プロジェクトポリシーのアタッチ \(SDK\)](#)」の手順に従って、プロジェクトポリシーをソースプロジェクトにアタッチします。

## プロジェクトポリシーのアタッチ (SDK)

Amazon Rekognition Custom Labels プロジェクトにプロジェクトポリシーをアタッチするには、[PutProjectPolicy](#) オペレーションを呼び出します。

追加するプロジェクトポリシーごとに [PutProjectPolicy](#) を呼び出すことにより、複数のプロジェクトポリシーをプロジェクトにアタッチします。1つのプロジェクトには最大5つのプロジェクトポリシーをアタッチできます。プロジェクトポリシーをさらにアタッチする必要がある場合は、[上限](#)の引き上げをリクエストできます。

固有のプロジェクトポリシーをプロジェクトに初めてアタッチするときは、`PolicyRevisionId` 入力パラメータにリビジョン ID を指定しないでください。[PutProjectPolicy](#) からのレスポンスは、Amazon Rekognition Custom Labels が作成するプロジェクトポリシーのリビジョン ID です。リビジョン ID を使用すると、プロジェクトポリシーの最新リビジョンを更新または削除できます。Amazon Rekognition Custom Labels は、プロジェクトポリシーの最新リビジョンのみを保持します。プロジェクトポリシーの以前のリビジョンを更新または削除しようとする、`InvalidPolicyRevisionIdException` エラーが発生します。

既存のプロジェクトポリシーを更新するには、`PolicyRevisionId` 入力パラメータにプロジェクトポリシーのリビジョン ID を指定します。[ListProjectPolicies](#) を呼び出すと、プロジェクト内のプロジェクトポリシーのリビジョン ID を取得できます。

プロジェクトポリシーをソースプロジェクトにアタッチすると、ソースプロジェクトから目的のプロジェクトにモデルをコピーできます。詳細については、「[モデルのコピー \(SDK\)](#)」を参照してください。

プロジェクトからプロジェクトポリシーを削除するには、[DeleteProjectPolicy](#) を呼び出します。プロジェクトにアタッチされているプロジェクトポリシーのリストを取得するには、[ListProjectPolicies](#) を呼び出します。

プロジェクトポリシーをプロジェクトにアタッチするには (SDK)

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. [プロジェクトポリシードキュメントを作成します](#)。

3. 次のコードを使用して、コピーするモデルバージョンを含む信頼する AWS アカウント内のプロジェクトにプロジェクトポリシーをアタッチします。プロジェクト ARN を取得するには、[DescribeProjects](#) を呼び出します。モデルバージョンを取得するには、[DescribeProjectVersions](#) を呼び出します。

## AWS CLI

以下の値を変更します。

- `project-arn` コピーするモデルバージョンを含む、信頼する AWS アカウントのソースプロジェクトの ARN を入力します。
- `policy-name` 選択したポリシー名を入力します。
- `principalModelVersionARN` で指定したモデルバージョンへのアクセスを許可または拒否するプリンシパルに設定します。
- `project-version-arn` コピーするモデルのバージョンの ARN を入力します。

既存のプロジェクトポリシーを更新するには、`policy-revision-id` 入力パラメータを指定し、プロジェクトポリシーのリビジョン ID を入力します。

```
aws rekognition put-project-policy \  
  --project-arn project-arn \  
  --policy-name policy-name \  
  --policy-document '{ "Version":"2012-10-17", "Statement":  
  [{ "Effect":"ALLOW or DENY", "Principal":{ "AWS":"principal" },  
    "Action":"rekognition:CopyProjectVersion", "Resource":"project-version-  
arn" }]}' \  
  --profile custom-labels-access
```

## Python

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `project_arn` - プロジェクトポリシーをアタッチするソースプロジェクトの ARN。
- `policy_name` - 選択したポリシー名。
- `project_policy` - プロジェクトポリシードキュメントを含むファイル。
- `policy_revision_id` - (オプション)。プロジェクトポリシーの既存のリビジョンを更新する場合、プロジェクトポリシーのリビジョン ID を入力します。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

"""
Purpose
Amazon Rekognition Custom Labels model example used in the service
documentation:
https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/md-copy-model-
sdk.html
Shows how to attach a project policy to an Amazon Rekognition Custom Labels
project.
"""

import boto3
import argparse
import logging
import json
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def put_project_policy(rek_client, project_arn, policy_name,
                      policy_document_file, policy_revision_id=None):
    """
    Attaches a project policy to an Amazon Rekognition Custom Labels project.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param policy_name: A name for the project policy.
    :param project_arn: The Amazon Resource Name (ARN) of the source project
    that you want to attach the project policy to.
    :param policy_document_file: The JSON project policy document to
    attach to the source project.
    :param policy_revision_id: (Optional) The revision of an existing policy to
    update.
    Pass None to attach new policy.
    :return The revision ID for the project policy.
    """

    try:

        policy_document_json = ""
        response = None
```

```
with open(policy_document_file, 'r') as policy_document:
    policy_document_json = json.dumps(json.load(policy_document))

logger.info(
    "Attaching %s project policy to project %s.",
    policy_name, project_arn)

if policy_revision_id is None:
    response = rek_client.put_project_policy(ProjectArn=project_arn,
                                             PolicyName=policy_name,
                                             PolicyDocument=policy_document_json)

else:
    response = rek_client.put_project_policy(ProjectArn=project_arn,
                                             PolicyName=policy_name,
                                             PolicyDocument=policy_document_json,
                                             PolicyRevisionId=policy_revision_id)

    new_revision_id = response['PolicyRevisionId']

logger.info(
    "Finished creating project policy %s. Revision ID: %s",
    policy_name, new_revision_id)

return new_revision_id

except ClientError as err:
    logger.exception(
        "Couldn't attach %s project policy to project %s: %s }",
        policy_name, project_arn, err.response['Error']['Message'] )
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
```

```
        "project_arn", help="The Amazon Resource Name (ARN) of the project "
        "that you want to attach the project policy to."
    )
    parser.add_argument(
        "policy_name", help="A name for the project policy."
    )

    parser.add_argument(
        "project_policy", help="The file containing the project policy JSON"
    )

    parser.add_argument(
        "--policy_revision_id", help="The revision of an existing policy to
update. "
        "If you don't supply a value, a new project policy is created.",
        required=False
    )

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)

        args = parser.parse_args()

        print(f"Attaching policy to {args.project_arn}")

        session = boto3.Session(profile_name='custom-labels-access')
        rekognition_client = session.client("rekognition")

        # Attach a new policy or update an existing policy.

        response = put_project_policy(rekognition_client,
                                     args.project_arn,
                                     args.policy_name,
```



```
        args.project_policy,
        args.policy_revision_id)

    print(
        f"project policy {args.policy_name} attached to project
{args.project_arn}")
    print(f"Revision ID: {response}")

    except ClientError as err:
        print("Problem attaching project policy: %s", err)

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

## Java V2

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `project_arn` - プロジェクトポリシーをアタッチするソースプロジェクトの ARN。
- `project_policy_name` - 選択したポリシー名。
- `project_policy_document` - プロジェクトポリシードキュメントを含むファイル。
- `project_policy_revision_id` - (オプション)。プロジェクトポリシーの既存のリビジョンを更新する場合、プロジェクトポリシーのリビジョン ID を入力します。

```
/*
 Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 SPDX-License-Identifier: Apache-2.0
*/

package com.example.rekognition;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.logging.Level;
import java.util.logging.Logger;

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.PutProjectPolicyRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

public class PutProjectPolicy {

    public static final Logger logger =
Logger.getLogger(PutProjectPolicy.class.getName());

    public static void putMyProjectPolicy(RekognitionClient rekClient, String
projectArn, String projectPolicyName,
        String projectPolicyFileName, String projectPolicyRevisionId)
throws IOException {

    try {

        Path filePath = Path.of(projectPolicyFileName);

        String policyDocument = Files.readString(filePath);

        String[] logArguments = new String[] { projectPolicyFileName,
projectPolicyName };

        PutProjectPolicyRequest putProjectPolicyRequest = null;

        logger.log(Level.INFO, "Attaching Project policy: {0} to project:
{1}", logArguments);

        // Attach the project policy.

        if (projectPolicyRevisionId == null) {
            putProjectPolicyRequest =
PutProjectPolicyRequest.builder().projectArn(projectArn)
                .policyName(projectPolicyName).policyDocument(policyDocument).build();
        } else {
            putProjectPolicyRequest =
PutProjectPolicyRequest.builder().projectArn(projectArn)
                .policyName(projectPolicyName).policyRevisionId(projectPolicyRevisionId)
                    .policyDocument(policyDocument)
```

```
        .build();
    }

    rekClient.putProjectPolicy(putProjectPolicyRequest);

    logger.log(Level.INFO, "Attached Project policy: {0} to project:
{1}", logArguments);

    } catch (

    RekognitionException e) {
        logger.log(Level.SEVERE, "Client error occurred: {0}",
e.getMessage());
        throw e;
    }

}

public static void main(String args[]) {

    final String USAGE = "\n" + "Usage: "
        + "<project_arn> <project_policy_name> <policy_document>
<project_policy_revision_id>\n\n" + "Where:\n"
        + "    project_arn - The ARN of the project that you want to
attach the project policy to.\n\n"
        + "    project_policy_name - A name for the project policy.\n\n"
        + "    project_policy_document - The file name of the project
policy.\n\n"
        + "    project_policy_revision_id - (Optional) The revision ID of
the project policy that you want to update.\n\n";

    if (args.length < 3 || args.length > 4) {
        System.out.println(USAGE);
        System.exit(1);
    }

    String projectArn = args[0];
    String projectPolicyName = args[1];
    String projectPolicyDocument = args[2];
    String projectPolicyRevisionId = null;

    if (args.length == 4) {
        projectPolicyRevisionId = args[3];
    }
}
```

```
try {  
  
    RekognitionClient rekClient = RekognitionClient.builder()  
        .credentialsProvider(ProfileCredentialsProvider.create("custom-  
labels-access"))  
        .region(Region.US_WEST_2)  
        .build();  
  
    // Attach the project policy.  
    putMyProjectPolicy(rekClient, projectArn, projectPolicyName,  
projectPolicyDocument,  
        projectPolicyRevisionId);  
  
    System.out.println(  
        String.format("project policy %s: attached to project: %s",  
projectPolicyName, projectArn));  
  
    rekClient.close();  
  
    } catch (RekognitionException rekError) {  
        logger.log(Level.SEVERE, "Rekognition client error: {0}",  
rekError.getMessage());  
        System.exit(1);  
    }  
  
    catch (IOException intError) {  
        logger.log(Level.SEVERE, "Exception while reading policy document:  
{0}", intError.getMessage());  
        System.exit(1);  
    }  
  
    }  
  
}
```

4. 「[モデルのコピー \(SDK\)](#)」の手順に従ってモデルバージョンをコピーします。

## モデルのコピー (SDK)

CopyProjectVersion API を使用して、モデルバージョンをソースプロジェクトから目的のプロジェクトにコピーできます。コピー先のプロジェクトは別の AWS アカウントでも構いませんが、同

じ AWS リージョンにある必要があります。コピー先のプロジェクトが別の AWS アカウントにある場合 (または、AWS アカウント内にコピーされたモデルバージョンに特定のアクセス許可を付与する場合)、ソースプロジェクトにプロジェクトポリシーをアタッチする必要があります。詳細については、「[プロジェクトポリシードキュメントの作成](#)」を参照してください。CopyProjectVersion API は、Amazon S3 バケットにアクセスする必要があります。

コピーされたモデルにはソースモデルのトレーニング結果が含まれますが、ソースのデータセットは含まれません。

適切なアクセス許可を設定しない限り、ソース AWS アカウントには、コピー先アカウントにコピーされたモデルの所有権はありません。

### モデルをコピーするには (SDK)

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. 「[プロジェクトポリシーのアタッチ \(SDK\)](#)」の手順に従って、プロジェクトポリシーをソースプロジェクトにアタッチします。
3. モデルを別の AWS アカウントにコピーする場合は、コピー先の AWS アカウントにプロジェクトがあることを確認してください。
4. 次のコードを使用して、モデルバージョンを目的のプロジェクトにコピーします。

### AWS CLI

以下の値を変更します。

- source-project-arn コピーするモデルバージョンを含むソースプロジェクトの ARN を入力します。
- source-project-version-arn コピーするモデルバージョンの ARN を入力します。
- destination-project-arn モデルのコピー先プロジェクトの ARN を入力します。
- version-name コピー先プロジェクトのモデルバージョン名を入力します。
- bucket ソースモデルのトレーニング結果のコピー先とする S3 バケットを入力します。
- folder ソースモデルのトレーニング結果をコピーする bucket のフォルダを入力します。
- (オプション) kms-key-id モデルの AWS Key Management Service のキー ID を入力します。

- (オプション) key 選択したタグキーを入力します。
- (オプション) value 選択したタグキーの値を入力します。

```
aws rekognition copy-project-version \  
  --source-project-arn source-project-arn \  
  --source-project-version-arn source-project-version-arn \  
  --destination-project-arn destination-project-arn \  
  --version-name version-name \  
  --output-config '{"S3Bucket": "bucket", "S3KeyPrefix": "folder"}' \  
  --kms-key-id arn:myKey \  
  --tags '{"key": "key"}' \  
  --profile custom-labels-access
```

## Python

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `source_project_arn` - コピーするモデルバージョンを含む、ソース AWS アカウントのソースプロジェクトの ARN。
- `source_project_version-arn` - コピーするソース AWS アカウントのモデルバージョンの ARN。
- `destination_project_arn` - モデルのコピー先プロジェクトの ARN。
- `destination_version_name` - コピー先プロジェクトのモデルのバージョン名。
- `training_results` - ソースモデルのバージョンのトレーニング結果をコピーする S3 の場所。
- (オプション) `kms_key_id` モデルの AWS Key Management Service のキー ID を入力します。
- (オプション) `tag_name` 選択したタグキーを入力します。
- (オプション) `tag_value` 選択したタグキーの値を入力します。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
  
import argparse  
import logging  
import time
```

```
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def copy_model(
    rekognition_client, source_project_arn, source_project_version_arn,
    destination_project_arn, training_results, destination_version_name):
    """
    Copies a version of a Amazon Rekognition Custom Labels model.

    :param rekognition_client: A Boto3 Amazon Rekognition Custom Labels client.
    :param source_project_arn: The ARN of the source project that contains the
    model that you want to copy.
    :param source_project_version_arn: The ARN of the model version that you
    want
    to copy.
    :param destination_project_arn: The ARN of the project that you want to copy
    the model
    to.
    :param training_results: The Amazon S3 location where training results for
    the model
    should be stored.
    return: The model status and version.
    """
    try:
        logger.info("Copying model...%s from %s to %s ",
            source_project_version_arn,
                source_project_arn,
                destination_project_arn)

        output_bucket, output_folder = training_results.replace(
            "s3://", "").split("/", 1)
        output_config = {"S3Bucket": output_bucket,
            "S3KeyPrefix": output_folder}

        response = rekognition_client.copy_project_version(
            DestinationProjectArn=destination_project_arn,
            OutputConfig=output_config,
            SourceProjectArn=source_project_arn,
            SourceProjectVersionArn=source_project_version_arn,
            VersionName=destination_version_name
        )
```

```
destination_model_arn = response["ProjectVersionArn"]

logger.info("Destination model ARN: %s", destination_model_arn)

# Wait until training completes.
finished = False
status = "UNKNOWN"
while finished is False:
    model_description =
rekognition_client.describe_project_versions(ProjectArn=destination_project_arn,
                                              VersionNames=[destination_version_name])
    status = model_description["ProjectVersionDescriptions"][0]
["Status"]

    if status == "COPYING_IN_PROGRESS":
        logger.info("Model copying in progress...")
        time.sleep(60)
        continue

    if status == "COPYING_COMPLETED":
        logger.info("Model was successfully copied.")

    if status == "COPYING_FAILED":
        logger.info(
            "Model copy failed: %s ",
            model_description["ProjectVersionDescriptions"][0]
["StatusMessage"])

        finished = True
except ClientError:
    logger.exception("Couldn't copy model.")
    raise
else:
    return destination_model_arn, status

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
```



```
        "source_project_arn",
        help="The ARN of the project that contains the model that you want to
copy."
    )

    parser.add_argument(
        "source_project_version_arn",
        help="The ARN of the model version that you want to copy."
    )

    parser.add_argument(
        "destination_project_arn",
        help="The ARN of the project which receives the copied model."
    )

    parser.add_argument(
        "destination_version_name",
        help="The version name for the model in the destination project."
    )

    parser.add_argument(
        "training_results",
        help="The S3 location in the destination account that receives the
training results for the copied model."
    )

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(
            f"Copying model version {args.source_project_version_arn} to project
{args.destination_project_arn}")

        session = boto3.Session(profile_name='custom-labels-access')
```

```
rekognition_client = session.client("rekognition")

# Copy the model.

model_arn, status = copy_model(rekognition_client,
                               args.source_project_arn,
                               args.source_project_version_arn,
                               args.destination_project_arn,
                               args.training_results,
                               args.destination_version_name,
                               )

print(f"Finished copying model: {model_arn}")
print(f"Status: {status}")

except ClientError as err:
    print(f"Problem copying model: {err}")

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

## Java V2

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `source_project_arn` - コピーするモデルバージョンを含む、ソース AWS アカウントのソースプロジェクトの ARN。
- `source_project_version_arn` - コピーするソース AWS アカウントのモデルバージョンの ARN。
- `destination_project_arn` - モデルのコピー先プロジェクトの ARN。
- `destination_version_name` - コピー先プロジェクトのモデルのバージョン名。
- `output_bucket` - ソースモデルバージョンのトレーニング結果をコピーする S3 バケツト。
- `output_folder` - ソースモデルバージョンのトレーニング結果をコピーする S3 のフォルダ。

```
/*
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
SPDX-License-Identifier: Apache-2.0
*/

package com.example.rekognition;
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.CopyProjectVersionRequest;
import
    software.amazon.awssdk.services.rekognition.model.CopyProjectVersionResponse;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsResponse;
import software.amazon.awssdk.services.rekognition.model.OutputConfig;
import
    software.amazon.awssdk.services.rekognition.model.ProjectVersionDescription;

import software.amazon.awssdk.services.rekognition.model.RekognitionException;

import java.util.logging.Level;
import java.util.logging.Logger;

public class CopyModel {

    public static final Logger logger =
        Logger.getLogger(CopyModel.class.getName());

    public static ProjectVersionDescription copyMyModel(RekognitionClient
rekClient,
        String sourceProjectArn,
        String sourceProjectVersionArn,
        String destinationProjectArn,
        String versionName,
        String outputBucket,
        String outputFolder) throws InterruptedException {

        try {

            OutputConfig outputConfig =
                OutputConfig.builder().s3Bucket(outputBucket).s3KeyPrefix(outputFolder).build();
```

```
String[] logArguments = new String[] { versionName,
sourceProjectArn, destinationProjectArn };

logger.log(Level.INFO, "Copying model {0} for from project {1} to
project {2}", logArguments);

CopyProjectVersionRequest copyProjectVersionRequest =
CopyProjectVersionRequest.builder()
    .sourceProjectArn(sourceProjectArn)
    .sourceProjectVersionArn(sourceProjectVersionArn)
    .versionName(versionName)
    .destinationProjectArn(destinationProjectArn)
    .outputConfig(outputConfig)
    .build();

CopyProjectVersionResponse response =
rekClient.copyProjectVersion(copyProjectVersionRequest);

logger.log(Level.INFO, "Destination model ARN: {0}",
response.projectVersionArn());
logger.log(Level.INFO, "Copying model...");

// wait until copying completes.

boolean finished = false;

ProjectVersionDescription copiedModel = null;

while (Boolean.FALSE.equals(finished)) {
    DescribeProjectVersionsRequest describeProjectVersionsRequest =
DescribeProjectVersionsRequest.builder()
        .versionNames(versionName)
        .projectArn(destinationProjectArn)
        .build();

    DescribeProjectVersionsResponse describeProjectVersionsResponse
= rekClient

    .describeProjectVersions(describeProjectVersionsRequest);

    for (ProjectVersionDescription projectVersionDescription :
describeProjectVersionsResponse
        .projectVersionDescriptions()) {
```

```
        copiedModel = projectVersionDescription;

        switch (projectVersionDescription.status()) {

            case COPYING_IN_PROGRESS:
                logger.log(Level.INFO, "Copying model...");
                Thread.sleep(5000);
                continue;

            case COPYING_COMPLETED:
                finished = true;
                logger.log(Level.INFO, "Copying completed");
                break;

            case COPYING_FAILED:
                finished = true;
                logger.log(Level.INFO, "Copying failed...");
                break;

            default:
                finished = true;
                logger.log(Level.INFO, "Unexpected copy status %s",
                    projectVersionDescription.statusAsString());
                break;

        }

    }

}

        logger.log(Level.INFO, "Finished copying model {0} for from project
{1} to project {2}", logArguments);

        return copiedModel;

    } catch (RekognitionException e) {
        logger.log(Level.SEVERE, "Could not train model: {0}",
e.getMessage());
        throw e;
    }

}
```

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

    String sourceProjectArn = null;
    String sourceProjectVersionArn = null;
    String destinationProjectArn = null;
    String versionName = null;
    String bucket = null;
    String location = null;

    final String USAGE = "\n" + "Usage: "
        + "<source_project_arn> <source_project_version_arn>
<destination_project_arn> <version_name> <output_bucket> <output_folder>\n\n"
        + "Where:\n"
        + "    source_project_arn - The ARN of the project that contains
the model that you want to copy. \n\n"
        + "    source_project_version_arn - The ARN of the project that
contains the model that you want to copy. \n\n"
        + "    destination_project_arn - The ARN of the destination
project that you want to copy the model to. \n\n"
        + "    version_name - A version name for the copied model.\n\n"
        + "    output_bucket - The S3 bucket in which to place the
training output. \n\n"
        + "    output_folder - The folder within the bucket that the
training output is stored in. \n\n";

    if (args.length != 6) {
        System.out.println(USAGE);
        System.exit(1);
    }

    sourceProjectArn = args[0];
    sourceProjectVersionArn = args[1];
    destinationProjectArn = args[2];
    versionName = args[3];
    bucket = args[4];
    location = args[5];

    try {

        // Get the Rekognition client.
        RekognitionClient rekClient = RekognitionClient.builder()
            .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
            .region(Region.US_WEST_2)
```

```
        .build());

    // Copy the model.
    ProjectVersionDescription copiedModel = copyMyModel(rekClient,
        sourceProjectArn,
        sourceProjectVersionArn,
        destinationProjectArn,
        versionName,
        bucket,
        location);

    System.out.println(String.format("Model copied: %s Status: %s",
        copiedModel.projectVersionArn(),
        copiedModel.statusMessage()));

    rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    } catch (InterruptedException intError) {
        logger.log(Level.SEVERE, "Exception while sleeping: {0}",
intError.getMessage());
        System.exit(1);
    }

    }

}
```

## プロジェクトポリシーの一覧表示 (SDK)

[ListProjectPolicies](#) オペレーションを使用すると、Amazon Rekognition Custom Labels プロジェクトにアタッチされているプロジェクトポリシーを一覧表示できます。

プロジェクトにアタッチされているプロジェクトポリシーを一覧表示するには (SDK)

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。

## 2. プロジェクトポリシーを一覧表示するには、次のコードを使用します。

### AWS CLI

`project-arn` を、アタッチされたプロジェクトポリシーを一覧表示するプロジェクトの Amazon リソースネームに変更します。

```
aws rekognition list-project-policies \  
  --project-arn project-arn \  
  --profile custom-labels-access
```

### Python

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `project_arn` - アタッチされたプロジェクトポリシーを一覧表示するプロジェクトの Amazon リソースネーム。

例: `python list_project_policies.py project_arn`

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
  
"""  
Purpose  
Amazon Rekognition Custom Labels model example used in the service  
documentation:  
https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/md-copy-model-  
sdk.html  
Shows how to list the project policies in an Amazon Rekognition Custom Labels  
project.  
"""  
  
import argparse  
import logging  
import boto3  
from botocore.exceptions import ClientError  
  
logger = logging.getLogger(__name__)
```



```
def display_project_policy(project_policy):
    """
    Displays information about a Custom Labels project policy.
    :param project_policy: The project policy (ProjectPolicy)
    that you want to display information about.
    """
    print(f"Policy name: {(project_policy['PolicyName'])}")
    print(f"Project Arn: {project_policy['ProjectArn']}")
    print(f"Document: {(project_policy['PolicyDocument'])}")
    print(f"Revision ID: {(project_policy['PolicyRevisionId'])}")
    print()

def list_project_policies(rek_client, project_arn):
    """
    Describes an Amazon Rekognition Custom Labels project, or all projects.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_arn: The Amazon Resource Name of the project you want to use.
    """

    try:

        max_results = 5
        pagination_token = ''
        finished = False

        logger.info("Listing project policies in: %s.", project_arn)
        print('Projects\n-----')
        while not finished:

            response = rek_client.list_project_policies(
                ProjectArn=project_arn, MaxResults=max_results,
                NextToken=pagination_token)

            for project in response['ProjectPolicies']:
                display_project_policy(project)

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

        logger.info("Finished listing project policies.")
```

```
except ClientError as err:
    logger.exception(
        "Couldn't list policies for - %s: %s",
        project_arn, err.response['Error']['Message'])
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The Amazon Resource Name of the project for which
you want to list project policies."
    )

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Listing project policies in: {args.project_arn}")

        # List the project policies.

        session = boto3.Session(profile_name='custom-labels-access')
        rekognition_client = session.client("rekognition")

        list_project_policies(rekognition_client,
                              args.project_arn)

    except ClientError as err:
        print(f"Problem list project_policies: {err}")
```

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

## Java V2

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `project_arn` - 一覧表示するプロジェクトポリシーを持つプロジェクトの ARN。

```
/*  
 Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
 SPDX-License-Identifier: Apache-2.0  
*/  
  
package com.example.rekognition;  
  
import java.util.logging.Level;  
import java.util.logging.Logger;  
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.ListProjectPoliciesRequest;  
import  
    software.amazon.awssdk.services.rekognition.model.ListProjectPoliciesResponse;  
import software.amazon.awssdk.services.rekognition.model.ProjectPolicy;  
import software.amazon.awssdk.services.rekognition.model.RekognitionException;  
  
public class ListProjectPolicies {  
  
    public static final Logger logger =  
        Logger.getLogger(ListProjectPolicies.class.getName());  
  
    public static void listMyProjectPolicies(RekognitionClient rekClient, String  
projectArn) {  
  
        try {  
  
            logger.log(Level.INFO, "Listing project policies for project: {0}",  
projectArn);
```

```
// List the project policies.

Boolean finished = false;
String nextToken = null;

while (Boolean.FALSE.equals(finished)) {

    ListProjectPoliciesRequest listProjectPoliciesRequest =
ListProjectPoliciesRequest.builder()
    .maxResults(5)
    .projectArn(projectArn)
    .nextToken(nextToken)
    .build();

    ListProjectPoliciesResponse response =
rekClient.listProjectPolicies(listProjectPoliciesRequest);

    for (ProjectPolicy projectPolicy : response.projectPolicies()) {

        System.out.println(String.format("Name: %s",
projectPolicy.policyName()));
        System.out.println(String.format("Revision ID: %s\n",
projectPolicy.policyRevisionId()));

    }

    nextToken = response.nextToken();

    if (nextToken == null) {
        finished = true;
    }

}

logger.log(Level.INFO, "Finished listing project policies for
project: {0}", projectArn);

} catch (

RekognitionException e) {
    logger.log(Level.SEVERE, "Client error occurred: {0}",
e.getMessage());
}
```

```
        throw e;
    }
}

public static void main(String args[]) {

    final String USAGE = "\n" + "Usage: " + "<project_arn> \n\n" + "Where:
\n"
        + "    project_arn - The ARN of the project with the project
policies that you want to list.\n\n";
    ;

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

    String projectArn = args[0];

    try {

        RekognitionClient rekClient = RekognitionClient.builder()
            .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
            .region(Region.US_WEST_2)
            .build();

        // List the project policies.
        listMyProjectPolicies(rekClient, projectArn);

        rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    }

}
}
```

## プロジェクトポリシーの削除 (SDK)

[DeleteProjectPolicy](#) オペレーションを使用すると、Amazon Rekognition Custom Labels プロジェクトから既存のプロジェクトポリシーのリビジョンを削除できます。プロジェクトにアタッチされているプロジェクトポリシーのすべてのリビジョンを削除する場合は、[ListProjectPolicies](#) を使用して、プロジェクトにアタッチされている各プロジェクトポリシーのリビジョン ID を取得します。次に、各ポリシー名の `DeletePolicy` を呼び出します。

### プロジェクトポリシーのリビジョンを削除するには (SDK)

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については、「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. プロジェクトポリシーを削除するには、次のコードを使用します。

`DeletePolicy` は `ProjectARN`、`PolicyName`、および `PolicyRevisionId` を使用します。この API には `ProjectARN` および `PolicyName` が必須です。`PolicyRevisionId` はオプションですが、アトミック更新を目的として含めることもできます。

### AWS CLI

以下の値を変更します。

- `policy-name` 削除するプロジェクトポリシーの名前を入力します。
- `policy-revision-id` 削除するプロジェクトポリシーのリビジョン ID を入力します。
- `project-arn` 削除するプロジェクトポリシーのリビジョンを含むプロジェクトの Amazon リソースネームを入力します。

```
aws rekognition delete-project-policy \  
  --policy-name policy-name \  
  --policy-revision-id policy-revision-id \  
  --project-arn project-arn \  
  --profile custom-labels-access
```

### Python

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `policy-name` - 削除するプロジェクトポリシーの名前。

- `project-arn` - 削除するプロジェクトポリシーを含むプロジェクトの Amazon リソースネーム。
- `policy-revision-id` - 削除するプロジェクトポリシーのリビジョン ID。

例: `python delete_project_policy.py policy_name project_arn policy_revision_id`

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

"""
Purpose
Amazon Rekognition Custom Labels model example used in the service
documentation:
https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/md-copy-model-
sdk.html
Shows how to delete a revision of a project policy.
"""

import argparse
import logging
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def delete_project_policy(rekognition_client, policy_name, project_arn,
                          policy_revision_id=None):
    """
    Deletes a project policy.

    :param rekognition_client: A Boto3 Amazon Rekognition client.
    :param policy_name: The name of the project policy that you want to delete.
    :param policy_revision_id: The revision ID for the project policy that you
    want to delete.
    :param project_arn: The Amazon Resource Name of the project that contains
    the project policy
    that you want to delete.
    """
    try:
```

```
logger.info("Deleting project policy: %s", policy_name)

if policy_revision_id is None:
    rekognition_client.delete_project_policy(
        PolicyName=policy_name,
        ProjectArn=project_arn)

else:
    rekognition_client.delete_project_policy(
        PolicyName=policy_name,
        PolicyRevisionId=policy_revision_id,
        ProjectArn=project_arn)

    logger.info("Deleted project policy: %s", policy_name)
except ClientError:
    logger.exception("Couldn't delete project policy.")
    raise

def confirm_project_policy_deletion(policy_name):
    """
    Confirms deletion of the project policy. Returns True if delete entered.
    :param model_arn: The ARN of the model that you want to delete.
    """
    print(
        f"Are you sure you want to delete project policy {policy_name} ?\n",
        policy_name)

    delete = input("Enter delete to delete your project policy: ")
    if delete == "delete":
        return True
    else:
        return False

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "policy_name", help="The ARN of the project that contains the project
        policy that you want to delete.")
```



```
)

parser.add_argument(
    "project_arn", help="The ARN of the project policy you want to
delete."
)

parser.add_argument(
    "--policy_revision_id", help="(Optional) The revision ID of the project
policy that you want to delete.",
    required=False
)

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        if confirm_project_policy_deletion(args.policy_name) is True:
            print(f"Deleting project_policy: {args.policy_name}")

            session = boto3.Session(profile_name='custom-labels-access')
            rekognition_client = session.client("rekognition")

            # Delete the project policy.

            delete_project_policy(rekognition_client,
                                  args.policy_name,
                                  args.project_arn,
                                  args.policy_revision_id)

            print(f"Finished deleting project policy: {args.policy_name}")
        else:
            print(f"Not deleting project policy {args.policy_name}")
    except ClientError as err:
        print(f"Couldn't delete project policy in {args.policy_name}: {err}")
```

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

## Java V2

次のコードを使用します。次のコマンドラインパラメータを指定します。

- `policy-name` - 削除するプロジェクトポリシーの名前。
- `project-arn` - 削除するプロジェクトポリシーを含むプロジェクトの Amazon リソースネーム。
- `policy-revision-id` - 削除するプロジェクトポリシーのリビジョン ID。

```
/*  
 Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
 SPDX-License-Identifier: Apache-2.0  
*/  
  
package com.example.rekognition;  
  
import java.util.logging.Level;  
import java.util.logging.Logger;  
  
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.DeleteProjectPolicyRequest;  
  
import software.amazon.awssdk.services.rekognition.model.RekognitionException;  
  
public class DeleteProjectPolicy {  
  
    public static final Logger logger =  
        Logger.getLogger(DeleteProjectPolicy.class.getName());  
  
    public static void deleteMyProjectPolicy(RekognitionClient rekClient, String  
projectArn,  
        String projectPolicyName,
```

```
String projectPolicyRevisionId)
throws InterruptedException {

    try {
        String[] logArguments = new String[] { projectPolicyName,
projectPolicyRevisionId };

        logger.log(Level.INFO, "Deleting: Project policy: {0} revision:
{1}", logArguments);

        // Delete the project policy.

        DeleteProjectPolicyRequest deleteProjectPolicyRequest =
DeleteProjectPolicyRequest.builder()
            .policyName(projectPolicyName)
            .policyRevisionId(projectPolicyRevisionId)
            .projectArn(projectArn).build();

        rekClient.deleteProjectPolicy(deleteProjectPolicyRequest);

        logger.log(Level.INFO, "Deleted: Project policy: {0} revision: {1}",
logArguments);

    } catch (

        RekognitionException e) {
            logger.log(Level.SEVERE, "Client error occurred: {0}",
e.getMessage());
            throw e;
        }

    }

    public static void main(String args[]) {

        final String USAGE = "\n" + "Usage: " + "<project_arn>
<project_policy_name> <project_policy_revision_id>\n\n"
            + "Where:\n"
            + "    project_arn - The ARN of the project that has the project
policy that you want to delete.\n\n"
            + "    project_policy_name - The name of the project policy that
you want to delete.\n\n"
            + "    project_policy_revision_id - The revision of the project
policy that you want to delete.\n\n";
```

```
    if (args.length != 3) {
        System.out.println(USAGE);
        System.exit(1);
    }

    String projectArn = args[0];
    String projectPolicyName = args[1];
    String projectPolicyRevisionId = args[2];

    try {

        RekognitionClient rekClient = RekognitionClient.builder()
            .credentialsProvider(ProfileCredentialsProvider.create("custom-
labels-access"))
            .region(Region.US_WEST_2)
            .build();

        // Delete the project policy.
        deleteMyProjectPolicy(rekClient, projectArn, projectPolicyName,
projectPolicyRevisionId);

        System.out.println(String.format("project policy deleted: %s
revision: %s", projectPolicyName,
            projectPolicyRevisionId));

        rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    }

    catch (InterruptedException intError) {
        logger.log(Level.SEVERE, "Exception while sleeping: {0}",
intError.getMessage());
        System.exit(1);
    }

}

}
```

## 例

このセクションでは、Amazon Rekognition Custom Labels の使用例に関する情報を記載しています。

例	説明
<a href="#">モデルフィードバックソリューション</a>	人間による検証によってモデルを改善し、新しいトレーニングデータセットを作成する方法を説明します。
<a href="#">Amazon Rekognition Custom Labels のデモンストレーション</a>	DetectCustomLabels の呼び出し結果を表示するユーザーインターフェイスのデモンストレーション。
<a href="#">動画分析</a>	ビデオから抽出したフレームを使った DetectCustomLabels の使用方法を説明します。
<a href="#">AWS Lambda 関数によるイメージ分析</a>	Lambda 関数を使った DetectCustomLabels の使用方法を説明します。
<a href="#">CSV ファイルからのマニフェストファイルの作成</a>	CSV を使って、イメージ (分類) 全体に関連する <a href="#">オブジェクト</a> 、 <a href="#">シーン</a> 、 <a href="#">コンセプト</a> の検索用マニフェストファイルを作成する方法を説明します。

## モデルフィードバックソリューション

モデルフィードバックソリューションでは、モデルの予測に関するフィードバックの提供や、人間による検証を使った改善を行うことができます。ユースケースによっては、少ないイメージのトレーニングデータセットで可能です。より正確なモデルの構築では、注釈付きの大規模なトレーニングセットを必要とする場合もあります。モデルフィードバックソリューションを使用すると、モデル作成支援を通じて、より大きなデータセットを作成できます。

モデルフィードバックソリューションをインストールして設定するには、「[Model Feedback Solution](#)」を参照してください。

モデルを継続的に改善するためのワークフローは次のとおりです。

1. モデルの最初のバージョンをトレーニングします (通常は小規模なトレーニングデータセットを使用します)。
2. モデルフィードバックソリューションには、注釈の付いていないデータセットを用意してください。
3. モデルフィードバックソリューションは現在のモデルを使用します。人間による検証ジョブを開始して、新しいデータセットに注釈を付けます。
4. モデルフィードバックソリューションが人間のフィードバックに基づいてマニフェストファイルを生成し、これを使って新しいモデルを作成します。

## Amazon Rekognition Custom Labels のデモンストレーション

Amazon Rekognition Custom Labels のデモンストレーションでは、[DetectCustomLabels](#) API を使ってローカルコンピュータのイメージを分析するユーザーインターフェイスを表示します。

このアプリケーションでは、AWS アカウントの Amazon Rekognition Custom Labels モデルに関する情報が表示されます。実行中のモデルを選択すると、ローカルコンピュータのイメージを分析できます。必要に応じてモデルを起動します。実行中のモデルを停止できます。このアプリケーションでは、Amazon Cognito、Amazon S3、Amazon CloudFront など、他の AWS サービスとの統合を表示します。

詳細については、「[Amazon Rekognition Custom Labels Demo](#)」を参照してください。

## 動画分析

以下の例では、DetectCustomLabels を使って動画から抽出したフレームを分析する方法を示しています。このコードのテスト対象は mov および mp4 形式の動画ファイルです。

**DetectCustomLabels** を使って、キャプチャしたフレームを分析する

1. 現時点で AWS CLI と AWS SDK のインストールと設定が完了していない場合は、インストールと設定を実行します。詳細については「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
2. rekognition:DetectCustomLabels および AmazonS3ReadOnlyAccess のアクセス許可があることを確認します。詳細については「[ステップ 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。

3. 次のサンプルコードを使用します。videoFile の値は、動画ファイル名に変更します。projectVersionArn の値を、お使いの Amazon Rekognition Custom Labels モデルの Amazon リソースネーム (ARN) に変更します。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

"""
Purpose
Shows how to analyze a local video with an Amazon Rekognition Custom Labels model.
"""
import argparse
import logging
import json
import math
import cv2
import boto3

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def analyze_video(rek_client, project_version_arn, video_file):
    """
    Analyzes a local video file with an Amazon Rekognition Custom Labels model.
    Creates a results JSON file based on the name of the supplied video file.
    :param rek_client: A Boto3 Amazon Rekognition client.
    :param project_version_arn: The ARN of the Custom Labels model that you want to
    use.
    :param video_file: The video file that you want to analyze.
    """

    custom_labels = []
    cap = cv2.VideoCapture(video_file)
    frame_rate = cap.get(5) # Frame rate.
    while cap.isOpened():
        frame_id = cap.get(1) # Current frame number.
        print(f"Processing frame id: {frame_id}")
        ret, frame = cap.read()
        if ret is not True:
            break
        if frame_id % math.floor(frame_rate) == 0:
```

```
        has_frame, image_bytes = cv2.imencode(".jpg", frame)

    if has_frame:
        response = rek_client.detect_custom_labels(
            Image={
                'Bytes': image_bytes.tobytes(),
            },
            ProjectVersionArn=project_version_arn
        )

        for elabel in response["CustomLabels"]:
            elabel["Timestamp"] = (frame_id/frame_rate)*1000
            custom_labels.append(elabel)

print(custom_labels)

with open(video_file + ".json", "w", encoding="utf-8") as f:
    f.write(json.dumps(custom_labels))

cap.release()

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_version_arn", help="The ARN of the model that you want to use."
    )

    parser.add_argument(
        "video_file", help="The local path to the video that you want to analyze."
    )

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:
        # Get command line arguments.
```



```
parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
add_arguments(parser)
args = parser.parse_args()

session = boto3.Session(profile_name='custom-labels-access')
rekognition_client = session.client("rekognition")

analyze_video(rekognition_client,
              args.project_version_arn, args.video_file)

except ClientError as err:
    print(f"Couldn't analyze video: {err}")

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

## AWS Lambda 関数によるイメージ分析

AWS Lambda はサーバーのプロビジョニングや管理をする必要がなく、コードを実行できるコンピューティングサービスです。例えば、アプリケーションコードをホストするサーバーを作成しなくても、モバイルアプリケーションから送信されたイメージを分析できます。以下の手順は、Python を使って [DetectCustomLabels](#) を呼び出す Lambda 関数を作成する方法です。この関数は提供されたイメージを分析し、イメージに含まれるラベルのリストを返します。この手順では Python コードの例を記載し、Amazon S3 バケット内またはローカルコンピュータのイメージに対する Lambda 関数の呼び出し方を示しています。

### トピック

- [ステップ 1: AWS Lambda 関数を作成する \(コンソール\)](#)
- [ステップ 2: \(オプション\) レイヤーを作成する \(コンソール\)](#)
- [ステップ 3: Python コードを追加する \(コンソール\)](#)
- [ステップ 4: Lambda 関数をテストする](#)

### ステップ 1: AWS Lambda 関数を作成する (コンソール)

このステップでは空の AWS 関数と、その関数が `DetectCustomLabels` オペレーションを呼び出せるようにするための IAM 実行ロールを作成します。これによって、分析用のイメージを格納する Amazon S3 バケットへのアクセス権も付与できます。また、以下の環境変数も指定します。

- Lambda 関数が使用する Amazon Rekognition Custom Labels モデル。
- モデルが使用する信頼限界。

ソースコードとオプションのレイヤーは、後で Lambda 関数に追加します。

AWS Lambda 関数を作成するには (コンソール)

1. AWS Management Console にサインインして AWS Lambda コンソール (<https://console.aws.amazon.com/lambda/>) を開きます。
2. [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 関数を作成する](#)」を参照してください。
3. 次のオプションを選択します。
  - [一から作成] を選択します。
  - [関数名] に値を入力します。
  - [ランタイム] では、[Python 3.10] を選択します。
4. [関数を作成] を選択して、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. [環境変数] ペインで、[編集] を選択します。
7. 次の環境変数を追加します。変数ごとに [環境変数を追加] を選択し、変数のキーと値を入力します。

キー	値
MODEL_ARN	Lambda 関数で使用するモデルの Amazon リソースネーム (ARN)。モデル ARN は、Amazon Rekognition Custom Labels コンソール内のモデルの詳細ページにある [モデルを使用] タブから取得できます。
CONFIDENCE	ラベルの予測におけるモデルの信頼度の最小値 (0~100)。Lambda 関数は、この値より信頼度の低いラベルを返しません。

8. [保存] を選択して環境変数を保存します。
9. [アクセス許可] タブの [実行ロール] で、ロールを選択し、IAM コンソールで開きます。

10. [アクセス許可] タブで [許可の追加]、次に[インラインポリシーの作成] をクリックします。
11. [JSON] タブを選択し、既存の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:DetectCustomLabels",
      "Resource": "*",
      "Effect": "Allow",
      "Sid": "DetectCustomLabels"
    }
  ]
}
```

12. [次へ] を選択します。
13. [ポリシーの詳細]に、「DetectCustomLabels-Access」などのポリシーの名前を入力します。
14. [ポリシーを作成] を選択します。
15. 分析用のイメージを Amazon S3 バケットに保存する場合は、ステップ 10～14 を繰り返します。
  - a. ステップ 11 では、以下のポリシーを使用します。####/#####を、Amazon S3 バケットと、分析するイメージ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 13 ではS3Bucket-access など、異なるポリシー名を選択します。

## ステップ 2: (オプション) レイヤーを作成する (コンソール)

この例を実行するうえで、この手順は不要です。DetectCustomLabels 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 環境にはない最新の AWS サービス更新が必要な場合は、このステップを実行し、最新の Boto3 SDK リリースを関数のレイヤーとして追加してください。

まず、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、.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、「[Lambda 関数でのレイヤーの使用](#)」を参照してください。

レイヤーを作成するには (コンソール)

1. コマンドプロンプトを開き、次のコマンドを使用します。

```
pip install boto3 --target python/.
zip boto3-layer.zip -r python/
```

2. zip ファイル (boto3-layer.zip) の名前を書き留めておきます。これはステップ 6 で必要になります。
3. AWS Lambda コンソール (<https://console.aws.amazon.com/lambda/>) 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. [レイヤーの作成] を選択します。
6. [名前] と [説明] の値を入力します。
7. [.zip ファイルをアップロード]、[アップロード] の順に選択します。
8. 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、ダイアログボックスで選択します。
9. 互換性のあるランタイムについては、Python 3.9 を選択してください。
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、[ステップ 1: AWS Lambda 関数を作成する \(コンソール\)](#) 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] では「1」を選択します。
19. [追加] を選択します。

## ステップ 3: Python コードを追加する (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を使用して、Lambda 関数に Python コードを追加します。このコードは DetectCustomLabels を使って、提供されたイメージを分析し、イメージ内のラベルのリストを返します。イメージは Amazon S3 バケットに置くか、または base64 暗号化イメージのバイトとして提供することもできます。

### Python コードを追加するには (コンソール)

1. Lambda コンソールを使用していない場合は、次の手順を実行します。
  - a. AWS Lambda コンソール (<https://console.aws.amazon.com/lambda/>) を開きます。
  - b. [ステップ 1: AWS Lambda 関数を作成する \(コンソール\)](#) で作成した Lambda 関数を開きます。
2. [コード] タブを選択します。
3. [コードソース] で、lambda\_function.py のコードを次のコードに置き換えます。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

"""
Purpose
An AWS lambda function that analyzes images with an the Amazon Rekognition
Custom Labels model.
"""

import json
import base64
from os import environ
import logging
import boto3

from botocore.exceptions import ClientError
```

```
# Set up logging.
logger = logging.getLogger(__name__)

# Get the model ARN and confidence.
model_arn = environ['MODEL_ARN']
min_confidence = int(environ.get('CONFIDENCE', 50))

# Get the boto3 client.
rek_client = boto3.client('rekognition')

def lambda_handler(event, context):
    """
    Lambda handler function
    param: event: The event object for the Lambda function.
    param: context: The context object for the lambda function.
    return: The labels found in the image passed in the event
    object.
    """

    try:

        # Determine image source.
        if 'image' in event:
            # Decode the image
            image_bytes = event['image'].encode('utf-8')
            img_b64decoded = base64.b64decode(image_bytes)
            image = {'Bytes': img_b64decoded}

        elif 'S3Object' in event:
            image = {'S3Object':
                    {'Bucket': event['S3Object']['Bucket'],
                     'Name': event['S3Object']['Name']}}
        else:
            raise ValueError(
                'Invalid source. Only image base 64 encoded image bytes or S3Object
                are supported.')

        # Analyze the image.
        response = rek_client.detect_custom_labels(Image=image,
```

```
        MinConfidence=min_confidence,
        ProjectVersionArn=model_arn)

# Get the custom labels
labels = response['CustomLabels']

lambda_response = {
    "statusCode": 200,
    "body": json.dumps(labels)
}

except ClientError as err:
    error_message = f"Couldn't analyze image. " + \
        err.response['Error']['Message']

    lambda_response = {
        'statusCode': 400,
        'body': {
            "Error": 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
```

4. [デプロイ] を選択して、作成した Lambda 関数をデプロイします。

## ステップ 4: Lambda 関数をテストする

このステップでは、コンピュータ上の Python コードを使って、ローカルイメージまたは Amazon S3 バケット内のイメージを Lambda 関数に渡します。ローカルコンピュータが渡すイメージは 6291456 バイト未満である必要があります。イメージがそれより大きい場合は、イメージを Amazon S3 バケットにアップロードし、イメージへの Amazon S3 パスを使ってスクリプトを呼び出します。バケットへのファイルのアップロードに関する詳細については、「[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。

必ず Lambda 関数を作成したのと同じ AWS リージョンでコードを実行します。作成した Lambda 関数の AWS リージョンは、[Lambda コンソール](#)にある関数の詳細ページのナビゲーションバーで確認できます。

AWS Lambda 関数がタイムアウトエラーを返す場合は、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。

[お使いのコードから Lambda 関数を呼び出す方法の詳細](#)については、「[AWS Lambda 関数 の呼び出し](#)」を参照してください。

Lambda 関数をテストするには

1. `lambda:InvokeFunction` の権限があることを確認してください。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 Identity Center 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 ユーザーガイド」の「[シークレットの作成と管理](#)」の手順に従ってください。

- ID プロバイダーを通じて IAM で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。「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](#)」にある手順に従ってください。

- IAM ユーザー:

- ユーザーが継承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。

- (非推奨) 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。手順については、「IAM ユーザーガイド」の「[ユーザーへのアクセス許可の追加 \(コンソール\)](#)」を参照してください。

2. Python 用 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 4: AWS CLI と AWS SDKs を設定する](#)」を参照してください。
3. [ステップ 1: AWS Lambda 関数を作成する \(コンソール\)](#)のステップ 7 で指定した[モデルを起動](#)します。
4. 次のコードを `client.py` という名前を付けて保存します。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

"""
Purpose
Test code for running the Amazon Rekognition Custom Labels Lambda
function example code.
"""

import argparse
import logging
import base64
import json
import boto3

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
```

```
def analyze_image(function_name, image):
    """Analyzes an image with an AWS Lambda function.
    :param image: The image that you want to analyze.
    :return The status and classification result for
    the image analysis.
    """

    lambda_client = boto3.client('lambda')

    lambda_payload = {}

    if image.startswith('s3://'):
        logger.info("Analyzing image from S3 bucket: %s", image)
        bucket, key = image.replace("s3://", "").split("/", 1)
        s3_object = {
            'Bucket': bucket,
            'Name': key
        }
        lambda_payload = {"S3Object": s3_object}

    # Call the lambda function with the image.
    else:
        with open(image, 'rb') as image_file:
            logger.info("Analyzing local image image: %s ", image)
            image_bytes = image_file.read()
            data = base64.b64encode(image_bytes).decode("utf8")

            lambda_payload = {"image": data}

    response = lambda_client.invoke(FunctionName=function_name,
                                    Payload=json.dumps(lambda_payload))

    return json.loads(response['Payload'].read().decode())

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "function", help="The name of the AWS Lambda function that you want " \
```

```
        "to use to analyze the image.")
    parser.add_argument(
        "image", help="The local image that you want to analyze.")

def main():
    """
    Entrypoint for script.
    """
    try:
        logging.basicConfig(level=logging.INFO,
                            format="%(levelname)s: %(message)s")

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        # Get analysis results.
        result = analyze_image(args.function, args.image)
        status = result['statusCode']

        if status == 200:
            labels = result['body']
            labels = json.loads(labels)
            print(f"There are {len(labels)} labels in the image.")
            for custom_label in labels:
                confidence = int(round(custom_label['Confidence'], 0))
                print(
                    f"Label: {custom_label['Name']}: Confidence: {confidence}%")
        else:
            print(f"Error: {result['statusCode']}")
            print(f"Message: {result['body']}")

    except ClientError as error:
        logging.error(error)
        print(error)

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

5. コードを実行します。コマンドライン引数には、Lambda 関数名と分析するイメージを指定します。ローカルイメージへのパス、または Amazon S3 バケットに保存したイメージへの S3 パスを指定できます。例:

```
python client.py function_name s3://bucket/path/image.jpg
```

イメージが Amazon S3 バケットにある場合は、[ステップ 1: AWS Lambda 関数を作成する \(コンソール\)](#)のステップ 15 で指定したバケットと同じであることを確認してください。

成功すると、イメージ内のラベルのリストが出力されます。ラベルが返されない場合は、[ステップ 1: AWS Lambda 関数を作成する \(コンソール\)](#)のステップ 7 で設定した「信頼度」の値を下げてください。

6. Lambda 関数を使用し終わり、そのモデルが他のアプリケーションで使用していない場合は、[モデルを停止](#)します。次回 Lambda 関数を使用するときには、必ず[モデルを起動](#)してください。

# セキュリティ

お客様がカスタムラベルを検出するために使用するプロジェクト、モデル、DetectCustomLabels オペレーションの管理を安全に行うことができます。

Amazon Rekognition の保護の詳細については、「[Amazon Rekognition Security](#)」を参照してください。

## Amazon Rekognition Custom Labels プロジェクトの保護

Amazon Rekognition Custom Labels プロジェクトを保護するには、アイデンティティベースのポリシーで指定されているリソースレベルのアクセス許可を指定します。詳細については、「[アイデンティティベースのポリシーおよびリソースベースのポリシー](#)」を参照してください。

保護できる Amazon Rekognition Custom Labels のリソースは次のとおりです。

リソース	Amazon リソースネーム形式
プロジェクト	arn:aws:rekognition:*:*:project/ <i>project_name</i> /datetime
モデル	arn:aws:rekognition:*:*:project/ <i>project_name</i> /version/ <i>name</i> /datetime

以下の例のポリシーでは、アイデンティティに次のアクセス許可を与える方法を示しています。

- すべてのプロジェクトについて説明します。
- 推論用の特定のモデルを作成、開始、停止、使用します。
- プロジェクトを作成します。特定のモデルを作成して説明します。
- 特定のプロジェクトの作成を拒否します。

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

```
    "Sid": "AllResources",
    "Effect": "Allow",
    "Action": "rekognition:DescribeProjects",
    "Resource": "*"
  },
  {
    "Sid": "SpecificProjectVersion",
    "Effect": "Allow",
    "Action": [
      "rekognition:StopProjectVersion",
      "rekognition:StartProjectVersion",
      "rekognition:DetectCustomLabels",
      "rekognition:CreateProjectVersion"
    ],
    "Resource": "arn:aws:rekognition:*:*:project/MyProject/version/MyVersion/*"
  },
  {
    "Sid": "SpecificProject",
    "Effect": "Allow",
    "Action": [
      "rekognition:CreateProject",
      "rekognition:DescribeProjectVersions",
      "rekognition:CreateProjectVersion"
    ],
    "Resource": "arn:aws:rekognition:*:*:project/MyProject/*"
  },
  {
    "Sid": "ExplicitDenyCreateProject",
    "Effect": "Deny",
    "Action": [
      "rekognition:CreateProject"
    ],
    "Resource": ["arn:aws:rekognition:*:*:project/SampleProject/*"]
  }
]
```

## DetectCustomLabels の保護

カスタムラベルの検出に使用される ID は、Amazon Rekognition Custom Labels モデルを管理する ID とは異なる場合があります。

ID にポリシーを適用することにより、ID の DetectCustomLabels へのアクセスを保護できます。次の例では、アクセスを DetectCustomLabels のみおよび特定のモデルに制限しています。この ID には、他の Amazon Rekognition オペレーションへのアクセス権がありません。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "rekognition:DetectCustomLabels"
      ],
      "Resource": "arn:aws:rekognition:*:*:project/MyProject/version/MyVersion/*"
    }
  ]
}
```

## AWS マネージドポリシー

Amazon Rekognition Custom Labels へのアクセスを制御するために使用できる AmazonRekognitionCustomLabelsFullAccess AWS マネージドポリシーを提供しています。詳細については、「[AWS マネージドポリシー: AmazonRekognitionCustomLabelsFullAccess](#)」を参照してください。

# Amazon Rekognition Custom Labels のガイドラインとクォータ

以下のセクションでは、Amazon Rekognition Custom Labels 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。

## サポートされるリージョン

Amazon Rekognition Custom Labels が利用可能な AWS リージョンの一覧については「Amazon Web Services 全般のリファレンス」の「[AWS 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

以下は、変更できない Amazon Rekognition Custom Labels の制限の一覧です。変更できる制限については、「[AWS サービスの制限](#)」を参照してください。制限を変更するには、「[ケースの作成](#)」を参照してください。

## トレーニング

- サポートされているファイル形式は、PNG および JPEG イメージ形式です。
- モデルのバージョン内の最大トレーニングデータセット数は 1 です。
- データセットマニフェストファイルの最大サイズは 1 GB です。
- オブジェクト、シーン、コンセプト (分類) データセットあたりの固有ラベルの最小数は 2 です。
- オブジェクト位置 (検出) データセットあたりの固有ラベルの最小数は 1 です。
- マニフェストあたりの固有ラベルの最大数は 250 です。
- ラベルあたりのイメージの最小数は 1 です。
- オブジェクト位置 (検出) データセットあたりのイメージの最大数は 250,000 です。

アジアパシフィック (ムンバイ) と欧州 (ロンドン) AWS リージョンの上限は 28,000 のイメージです。

- オブジェクト、シーン、コンセプト (分類) データセットあたりのイメージの最大数は 500,000 です。デフォルトは 250,000 です。増加をリクエストするには、「[ケースの作成](#)」を参照してください。



アジアパシフィック (ムンバイ) と欧州 (ロンドン) AWS リージョンの上限は 28,000 のイメージです。制限の引き上げはリクエストできません。

- イメージごとのラベルの最大数は 50 です。
- イメージ内の境界ボックスの最小数は 0 です。
- イメージ内の境界ボックスの最大数は 50 です。
- Amazon S3 バケット内のイメージファイルの最小イメージ寸法は 64 ピクセル x 64 ピクセルです。
- Amazon S3 バケット内のイメージファイルの最大イメージ寸法は 4,096 ピクセル x 4,096 ピクセルです。
- Amazon S3 バケット内のイメージファイルの最大ファイルサイズは 15 MB です。
- イメージの最大アスペクト比は 20:1 です。

## テスト

- モデルの 1 つのバージョンに含まれるテストデータセットの最大数は 1 です。
- データセットマニフェストファイルの最大サイズは 1 GB です。
- オブジェクト、シーン、コンセプト (分類) データセットあたりの固有ラベルの最小数は 2 です。
- オブジェクト位置 (検出) データセットあたりの固有ラベルの最小数は 1 です。
- データセットあたりの固有ラベルの最大数は 250 です。
- ラベルあたりのイメージの最小数は 0 です。
- ラベルごとのイメージの最大数は 1,000 です。
- オブジェクト位置 (検出) データセットあたりのイメージの最大数は 250,000 です。

アジアパシフィック (ムンバイ) と欧州 (ロンドン) AWS リージョンの上限は 7,000 のイメージです。

- オブジェクト、シーン、コンセプト (分類) データセットあたりのイメージの最大数は 500,000 です。デフォルトは 250,000 です。増加をリクエストするには、[「ケースの作成」](#)を参照してください。

アジアパシフィック (ムンバイ) と欧州 (ロンドン) AWS リージョンの上限は 7,000 のイメージです。制限の引き上げはリクエストできません。

- マニフェストあたりのイメージごとのラベルの最小数は 0 です。
- マニフェストあたりのイメージごとのラベルの最大数は 50 です。

- マニフェストあたりのイメージ内の境界ボックスの最小数は 0 です。
- マニフェストあたりのイメージ内の境界ボックスの最大数は 50 です。
- Amazon S3 バケット内のイメージファイルの最小イメージ寸法は 64 ピクセル x 64 ピクセルです。
- Amazon S3 バケット内のイメージファイルの最大イメージ寸法は 4,096 ピクセル x 4,096 ピクセルです。
- Amazon S3 バケット内のイメージファイルの最大ファイルサイズは 15 MB です。
- サポートされているファイル形式は、PNG および JPEG イメージ形式です。
- イメージの最大アスペクト比は 20:1 です。

## 検出

- raw バイトとして渡されるイメージの最大サイズは 4 MB です。
- Amazon S3 バケット内のイメージファイルの最大ファイルサイズは 15 MB です。
- 入力イメージファイル (Amazon S3 に保存されている、またはイメージバイトとして提供されるもの) の最小イメージ寸法は、64 ピクセル x 64 ピクセルです。
- 入力イメージファイル (Amazon S3 に保存されている、またはイメージバイトとして提供されるもの) の最大イメージ寸法は、4,096 ピクセル x 4,096 ピクセルです。
- サポートされているファイル形式は、PNG および JPEG イメージ形式です。
- イメージの最大アスペクト比は 20:1 です。

## モデルのコピー

- 1 つのプロジェクトに[アタッチ](#)できるプロジェクトポリシーの最大数は 5 です。
- 1 つの送信先での同時コピージョブの最大数は 5 です。

# Amazon Rekognition Custom Labels API リファレンス

Amazon Rekognition Custom Labels API は、Amazon Rekognition API リファレンスのコンテンツの一部として文書化されています。これは Amazon Rekognition Custom Labels API オペレーションの一覧で、該当する Amazon Rekognition API リファレンスのトピックへのリンクも含まれています。また、このドキュメント内の API リファレンスリンクは、該当する「Amazon Rekognition デベロッパーガイド」の API リファレンストピックに移動します。API の使用については、「

[このセクションでは、コンソールと AWS SDK で Amazon Rekognition Custom Labels モデルをトレーニングして使用するワークフローの概要を説明します。](#)

## Note

Amazon Rekognition Custom Labels がプロジェクト内のデータセットを管理するようになりました。コンソールと AWS SDK を使用して、プロジェクトのデータセットを作成でき

ます。以前に Amazon Rekognition Custom Labels を使用したことがある場合は、古いデー

タセットを新しいプロジェクトに関連付ける必要がある場合があります。詳細については、

「[ステップ 6: \(オプション\) 以前のデータセットを新しいプロジェクトに関連付ける](#)」を参照してください。

## トピック

- [モデルタイプの決定](#)
- [モデルを作成する](#)
- [モデルの改善](#)
- [モデルの開始](#)
- [イメージの分析](#)
- [モデルの停止](#)

## モデルタイプの決定

まず、ビジネス目標に応じて、どのタイプのモデルをトレーニングするかを決定します。例えば、モデルタイプの決定ソーシャルメディアの投稿で自社のロゴを見つけたり、店舗の棚で商品を識別したり、組立ラインで機械部品を分類したりするようにモデルをトレーニングできます。 490

Amazon Rekognition Custom Labels では、以下のタイプのモデルをトレーニングできます。

- オブジェクト、シーン、概念を検出する

- オブジェクトの位置の検索

- ブランドの所在地を探す

トレーニングするモデルのタイプを決定しやすくするために、Amazon Rekognition Custom Labels には使用できるサンプルプロジェクトが用意されています。詳細については、「[Amazon Rekognition Custom Labels の開始方法](#)」を参照してください。

## オブジェクト、シーン、概念を検出する

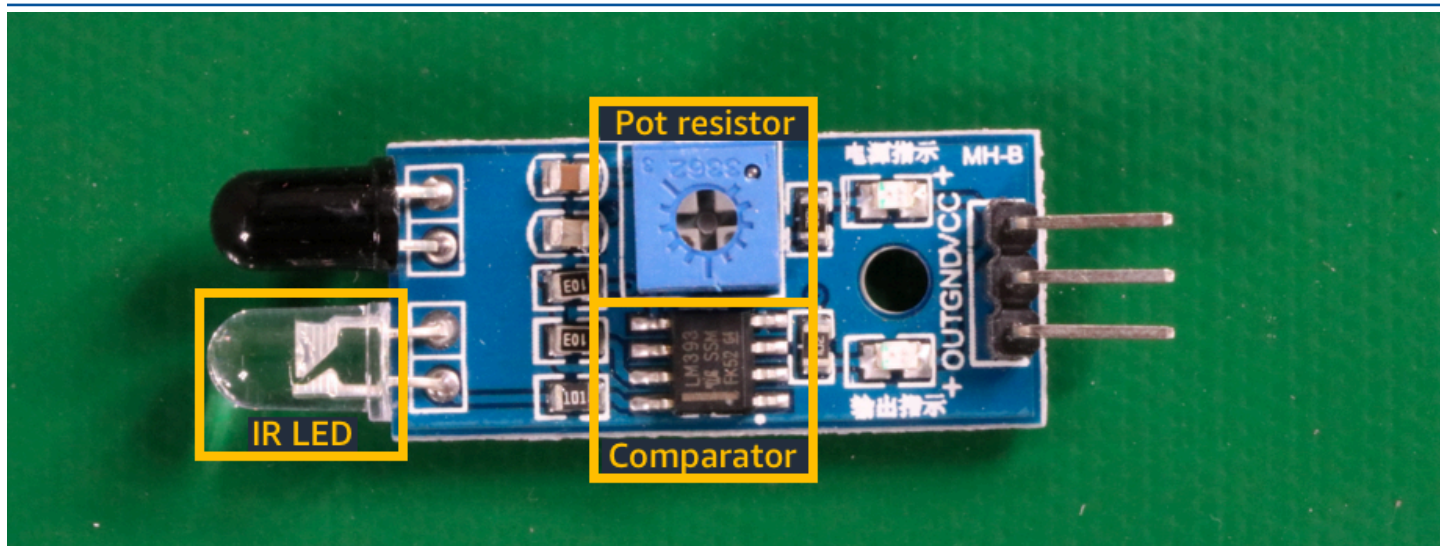
このモデルは、イメージ全体に関連するオブジェクト、シーン、概念の分類を予測します。例えば、イメージに観光名所が含まれているかどうかを判断するモデルをトレーニングできます。サンプルプロジェクトについては、「[画像分類](#)」を参照してください。次のレイクのイメージは、オブジェクト、シーン、概念を認識できるイメージの種類の例です。



イメージを複数のカテゴリに分類するモデルをトレーニングすることもできます。例えば、前のイメージには、空の色、反射、湖などのカテゴリが含まれている場合があります。サンプルプロジェクトについては、「[マルチラベルイメージ分類](#)」を参照してください。

## オブジェクトの位置の検索

モデルはイメージ上のオブジェクトの位置を予測します。予測には、オブジェクトの位置に関する境界ボックス情報と、境界ボックス内のオブジェクトを識別するラベルが含まれます。例えば、次のイメージは、コンパレータやポット抵抗など、回路基板のさまざまな部分を囲む境界ボックスを示しています。

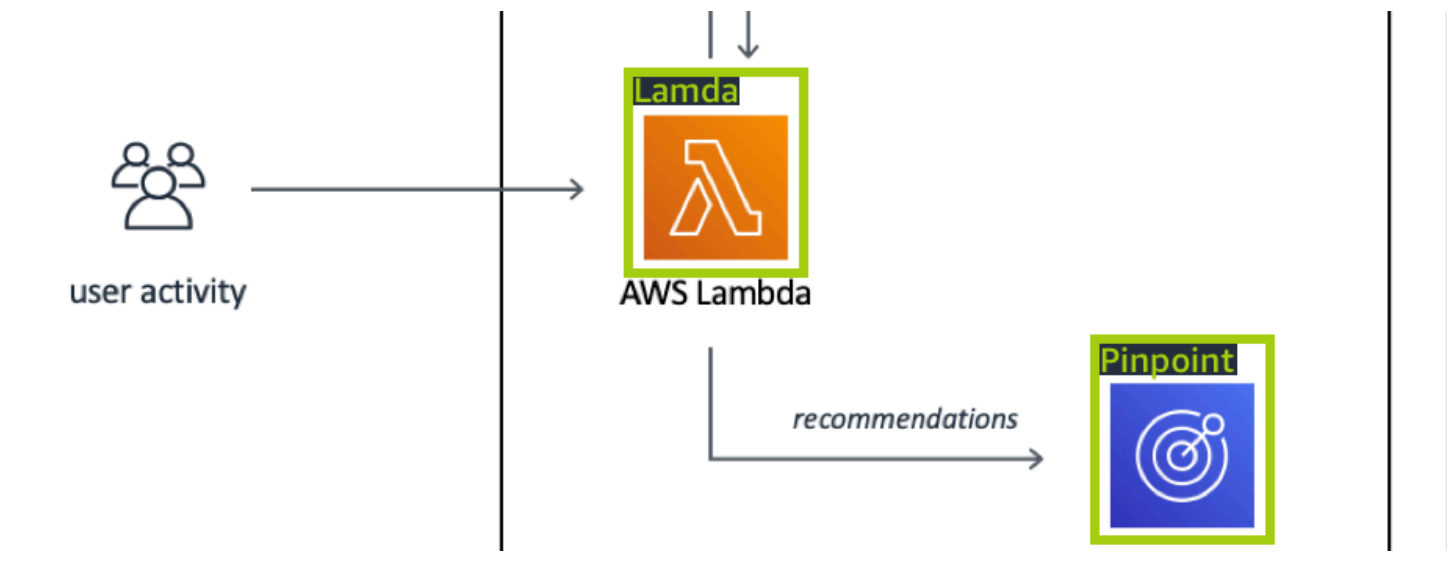


オブジェクトのローカリゼーションのサンプルプロジェクトでは、Amazon Rekognition Custom Labels がラベル付き境界ボックスを使用して、オブジェクトの位置を検出するモデルをトレーニングする方法を示しています。

## ブランドの所在地を探す

Amazon Rekognition Custom Labels は、イメージ上のブランドの場所 (ロゴなど) を検索するようにモデルをトレーニングできます。予測には、ブランドロケーションの境界ボックス情報と、境界ボックス内のオブジェクトを識別するラベルが含まれます。サンプルプロジェクトについては、「[ブランド検出](#)」を参照してください。次の画像は、モデルが検出できるいくつかのブランドの例です。





## モデルを作成する

モデルを作成する手順は、プロジェクトの作成、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、モデルのトレーニングです。

### プロジェクトを作成する

プロジェクトとは、Amazon Rekognition Custom Labels モデルのバージョンを作成および管理するために必要なリソースのグループです。プロジェクトでは、次のものが管理されます。

- データセット - モデルのトレーニングに使用されるイメージとイメージラベル。プロジェクトには、トレーニングデータセットとテストデータセットがあります。
- モデル - ビジネス特有の概念、シーン、オブジェクトを検索するためのトレーニングを行うソフトウェアです。プロジェクトには、モデルの複数のバージョンを含めることができます。

回路基板上の回路基板部品を検索するなど、単一のユースケースにプロジェクトを使用することをお勧めします。

Amazon Rekognition Custom Labels コンソールと [CreateProject](#) API を使用してプロジェクトを作成できます。詳細については、「[プロジェクトの作成](#)」を参照してください。

## 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

データセットとは、イメージとそのイメージを説明するラベルの集合のことです。プロジェクト内で、Amazon Rekognition Custom Labels でモデルのトレーニングとテストに使用するトレーニングデータセットとテストデータセットを作成します。

ラベルは、イメージ内のオブジェクトを囲むオブジェクト、シーン、概念、または境界ボックスを識別します。ラベルはイメージ全体 (イメージレベル) に割り当てられるか、イメージ上のオブジェクトを囲む境界ボックスに割り当てられます。

### ⚠ Important

データセット内のイメージに付けるラベルによって、Amazon Rekognition Custom Labels が作成するモデルのタイプが決まります。例えば、オブジェクト、シーン、概念を検出する

モデルをトレーニングするには、トレーニングデータセットとテストデータセットのイメー

ジにイメージレベルのラベルを割り当てます。詳細については、「[データセットの目的の設定](#)」を参照してください。

イメージは PNG 形式および JPEG 形式である必要があり、入力イメージの推奨事項に従う必要があります。詳細については、「[イメージの準備](#)」を参照してください。

## 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 (コンソール)

1つのデータセットでプロジェクトを開始することも、個別のトレーニングデータセットとテストデータセットを持つプロジェクトから始めることもできます。1つのデータセットから始めると、Amazon Rekognition Custom Labels はトレーニング中にデータセットを分割して、プロジェクトのトレーニングデータセット (80%) とテストデータセット (20%) を作成します。Amazon Rekognition Custom Labels にトレーニングとテストに使用するイメージを決定させる場合は、1つのデータセットから始めてください。トレーニング、テスト、パフォーマンスのチューニングを完全に制御するには、トレーニングデータセットとテストデータセットを分けてプロジェクトを開始することをお勧めします。

プロジェクトのデータセットを作成するには、次のいずれかの方法でイメージをインポートします。

- ローカルコンピュータから画像をインポートします。
- S3 バケットから画像をインポートします。Amazon Rekognition Custom Labels では、イメージを 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 含むフォルダ名を使用してイメージにラベル付けすることができます。

- Amazon SageMaker Ground Truth マニフェストファイルをインポートします。

- 既存の Amazon Rekognition Custom Labels データセットをコピーします。

詳細については、「[イメージ付きの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](#)」を参照してください。

イメージのインポート元によっては、イメージにラベルが付いていない場合があります。例えば、ローカルコンピュータからインポートされたイメージにはラベルは付きません。Amazon SageMaker Ground Truth マニフェストファイルからインポートされたイメージにはラベルが付けられます。Amazon Rekognition Custom Labels コンソールを使用して、ラベルの追加、変更、割り当てを行うことができます。詳細については、「[イメージにラベルを付ける](#)」を参照してください。

コンソールでトレーニングデータセットとテストデータセットを作成するには、「[イメージ付きの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](#)」を参照してください。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を含むチュートリアルについては、「[チュートリアル: イメージの分類](#)」を参照してください。

## 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 (SDK)

トレーニングデータセットとテストデータセットを作成するには、CreateDataset API を使用します。Amazon Sagemaker 形式のマニフェストファイルを使用するか、既存の Amazon Rekognition Custom Labels をコピーして、データセットを作成できます。詳細については、「[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 \(SDK\)](#)」を参照してください。必要に応じて、独自のマニフェストファイルを作成できます。詳細については、「[the section called “マニフェストファイルの作成”](#)」を参照してください。

## モデルをトレーニングする

トレーニングデータセットでモデルをトレーニングします。モデルの新しいバージョンは、トレーニングのたびに作成されます。トレーニング中、Amazon Rekognition Custom Labels はトレーニング済みモデルのパフォーマンスをテストします。その結果を使用して、モデルを評価し、改善することができます。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。モデルのトレーニングが成功した場合にのみ課金されます。詳細については、「[Amazon Rekognition Custom Labels モデルをトレーニングする](#)」を参照してください。モデルトレーニングが失敗した場合、Amazon Rekognition Custom Labels は使用できるデバッグ情報を提供します。詳細については、「[失敗したモデルトレーニングのデバッグ](#)」を参照してください。



---

## モデルのトレーニング (コンソール)

---

コンソールでモデルをトレーニングする方法については、「[モデルのトレーニング \(コンソール\)](#)」を参照してください。

---

---

## モデルのトレーニング (SDK)

---

Amazon Rekognition Custom Labels モデルをトレーニングするには、[CreateProjectバージョン](#) を呼び出します。詳細については、「[モデルのトレーニング \(SDK\)](#)」を参照してください。

---

---

## モデルの改善

---

テスト中、Amazon Rekognition Custom Labels は、トレーニング済みモデルの改善に使用できる評価メトリクスを作成します。

---

---

## モデルの評価

---

テスト中に作成されたパフォーマンスメトリクスを使用して、モデルのパフォーマンスを評価します。F1、適合率、再現率などのパフォーマンスメトリクスにより、トレーニングしたモデルのパフォーマンスを理解し、本稼働で使用する準備ができたかどうかを判断することができます。詳細については、「[モデルを評価するためのメトリクス](#)」を参照してください。

---

---

## モデルを評価する (コンソール)

---

パフォーマンスメトリクスを表示するには、「[評価メトリクスへのアクセス \(コンソール\)](#)」を参照してください。

---

---

## モデルの評価 (SDK)

---

パフォーマンスメトリクスを取得するには、[DescribeProjectバージョン](#) を呼び出してテスト結果を取得します。詳細については、「[Amazon Rekognition Custom Labels の評価メトリクス \(SDK\) へのアクセス](#)」を参照してください。テスト結果には、分類結果の混同行列など、コンソールにはないメトリクスが含まれます。テスト結果は次の形式で返されます。

---

- F1 スコア - モデルの全体的な適合率と再現率を表す単一の値。詳細については、「[F1](#)」を参照してください。
  - サマリーファイルの場所 - テストサマリーには、テストデータセット全体の集計評価メトリクスと、個々のラベルのメトリクスが含まれます。DescribeProjectVersions は、サマリーファ
-

イルの S3 バケットとフォルダの場所を返します。詳細については、「[概要ファイル](#)」を参照してください。

- 評価マニフェストのスナップショットの場所 - スナップショットには、信頼度評価や誤検出などの二項分類テストの結果など、テスト結果に関する詳細が含まれています。DescribeProjectVersions は、スナップショットファイルの S3 バケットとフォルダの場所を返します。詳細については、「[評価マニフェストのスナップショット](#)」を参照してください。

## モデルの改善

改善が必要な場合は、トレーニングイメージを追加するか、データセットのラベル付けを改善することができます。詳細については、「[Amazon Rekognition Custom Labels モデルの改善](#)」を参照してください。モデルが作成した予測についてフィードバックを与えて、それを使用してモデルを改善することもできます。詳細については、「[モデルフィードバックソリューション](#)」を参照してください。

### モデルの改善 (コンソール)

データセットにイメージを追加する方法については、「[データセットへのイメージの追加](#)」を参照してください。ラベルを追加または変更するには、「[the section called “イメージにラベルを付ける”](#)」を参照してください。

モデルを再トレーニングするには、「[モデルのトレーニング \(コンソール\)](#)」を参照してください。

### モデルの改善 (SDK)

データセットにイメージを追加したり、イメージのラベルを変更するには、UpdateDatasetEntries API を使用します。UpdateDatasetEntries は JSON 行を更新するか、マニフェストファイルに追加します。JSON の各行には、割り当てられたラベルや境界ボックスの情報など、1つのイメージに関する情報が含まれています。詳細については、「[イメージの追加 \(SDK\)](#)」を参照してください。データセット内のエントリを表示するには、ListDatasetEntries API を使用します。

モデルを再トレーニングするには、「[モデルのトレーニング \(SDK\)](#)」を参照してください。

## モデルの開始

モデルを使用する前に、Amazon Rekognition Custom Labels コンソールまたは StartProjectVersion API を使用してモデルを開始します。モデルの稼働時間に応じて課金されます。詳細については、「[トレーニング済みモデルの実行](#)」を参照してください。

---

## モデルの開始 (コンソール)

---

コンソールを使用してモデルを開始する方法については、「[Amazon Rekognition Custom Labels モデルの開始 \(コンソール\)](#)」を参照してください。

---

---

## モデルの開始

---

`StartProject`バージョン を呼び出すモデルを開始します。詳細については、「[Amazon Rekognition Custom Labels モデル \(SDK\) を開始します。](#)」を参照してください。

---

---

## イメージの分析

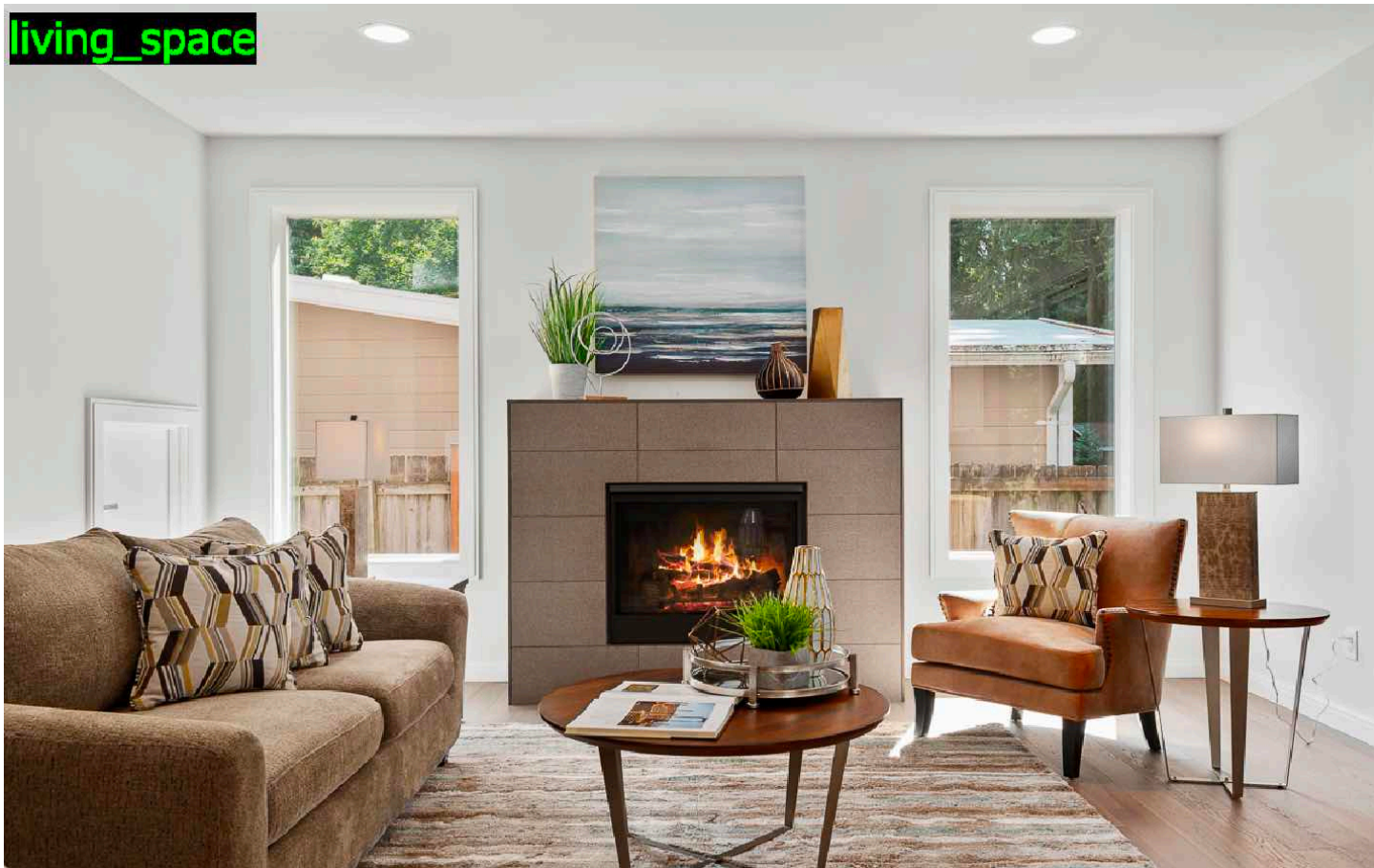
---

モデルを使用してイメージを分析するには、`DetectCustomLabels` API を使用します。ローカルイメージ、または S3 バケットに保存されているイメージを指定できます。オペレーションには、使用するモデルの Amazon リソースネーム (ARN) も必要です。

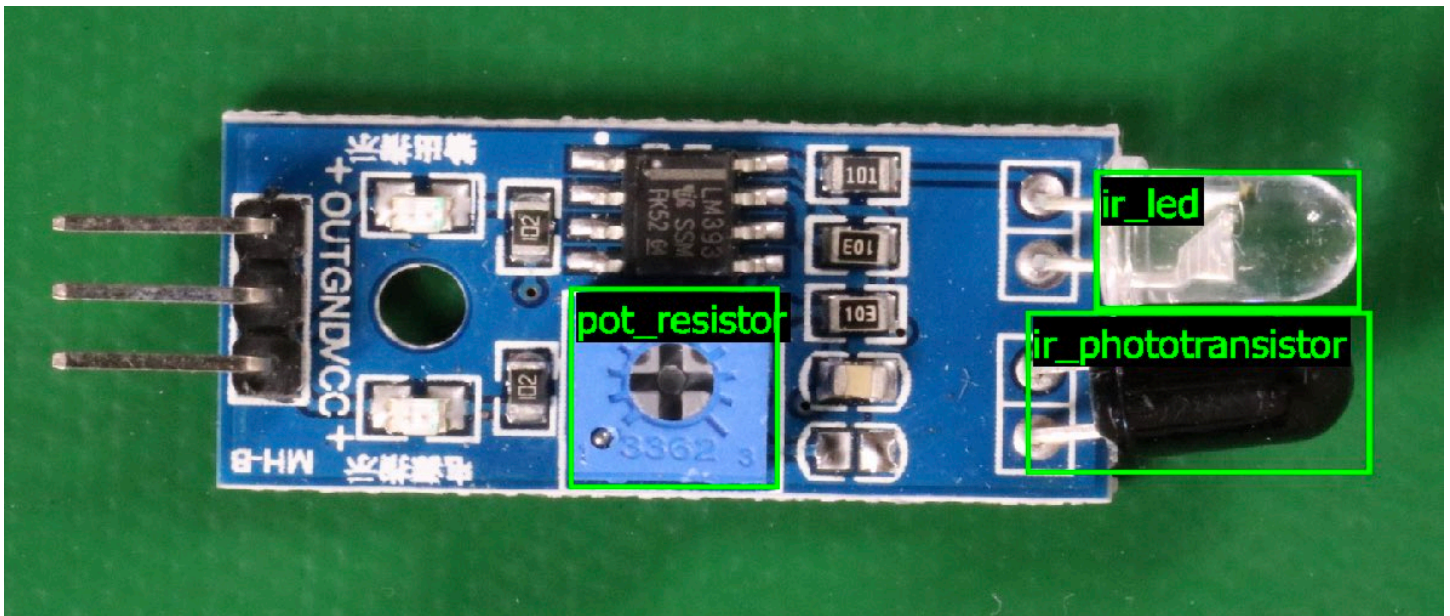
---

モデルでオブジェクト、シーン、概念が見つかった場合、レスポンスにはイメージに含まれるイメージレベルのラベルのリストが含まれます。例えば、次のイメージは、Rooms サンプルプロジェクトで見つかったイメージレベルのラベルを示しています。

---

**living\_space**

モデルがオブジェクトの位置を検出した場合、レスポンスにはイメージ内のラベル付き境界ボックスのリストが含まれます。境界ボックスは、イメージ上のオブジェクトの位置を表します。境界ボックスの情報を使用して、オブジェクトの周囲に境界ボックスを描画できます。例えば、次のイメージは、回路基板サンプルプロジェクトを使用して見つかった回路基板部品の周囲の境界ボックスを示しています。



詳細については、「[トレーニングされたモデルによるイメージの分析](#)」を参照してください。

## モデルの停止

モデルの稼働時間に応じて課金されます。モデルを使用しなくなった場合は、Amazon Rekognition Custom Labels コンソールまたは `StopProjectVersion` API を使用してモデルを停止します。詳細については、「[Amazon Rekognition Custom Labels モデルの停止](#)」を参照してください。

### モデルの停止する (コンソール)

コンソールを使用して実行中のモデルを停止するには、「[Amazon Rekognition Custom Labels モデルの停止 \(コンソール\)](#)」を参照してください。

### モデルの停止 (SDK)

実行中のモデルを停止するには、`StopProjectバージョン` を呼び出します。詳細については、「[Amazon Rekognition Custom Labels モデル \(SDK\) の停止](#)」を参照してください。



# モデルのトレーニング

## プロジェクト

- [CreateProject](#) - リソース (イメージ、ラベル、モデル) とオペレーション (トレーニング、評価、検知) を論理的にグループ化した Amazon Rekognition Custom Labels プロジェクトを作成します。
- [DeleteProject](#) - Amazon Rekognition Custom Labels プロジェクトを削除します。
- [DescribeProjects](#) - すべての Amazon Rekognition Custom Labels のプロジェクトの一覧を返します。

## プロジェクトポリシー

- [PutProjectPolicy](#) - 信頼する AWS アカウントの Amazon Rekognition Custom Labels プロジェクトにプロジェクトポリシーをアタッチします。
- [ListProjectPolicies](#) - プロジェクトにアタッチされているプロジェクトポリシーの一覧を返します。
- [DeleteProjectPolicy](#) - 既存のプロジェクトポリシーを削除します。

## データセット

- [CreateDataset](#) - Amazon Rekognition Custom Labels データセットを作成します。
- [DeleteDataset](#) - Amazon Rekognition Custom Labels データセットを削除します。
- [DescribeDataset](#) - Amazon Rekognition Custom Labels データセットを記述します。
- [DistributeDatasetEntries](#) - トレーニングデータセット内のエントリ (イメージ) を、プロジェクトのトレーニングデータセットとテストデータセット全体に分散させます。
- [ListDataSetEntries](#) - Amazon Rekognition Custom Labels データセットのエントリ (イメージ) の一覧を返します。
- [ListDataSetLabels](#) - Amazon Rekognition Custom Labels データセットに割り当てられたラベルの一覧を返します。
- [UpdateDataSetEntries](#) - Amazon Rekognition Custom Labels データセットのエントリ (イメージ) を追加または更新します。

## モデル

- [CreateProjectVersion](#) - Amazon Rekognition Custom Labels モデルをトレーニングします。
- [CopyProjectVersion](#) - Amazon Rekognition Custom Labels モデルをコピーします。
- [DeleteProjectVersion](#) - Amazon Rekognition Custom Labels モデルを削除します。
- [DescribeProjectVersions](#) - 特定のプロジェクト内のすべての Amazon Rekognition Custom Labels モデルの一覧を返します。

## タグ

- [TagResource](#) - Amazon Rekognition Custom Labels モデルに 1 つ以上のキーバリュータグを追加します。
- [UntagResource](#) - Amazon Rekognition Custom Labels モデルから 1 つ以上のタグを削除します。

## モデルの使用

- [DetectCustomLabels](#) - カスタムラベルモデルを使用してイメージを分析します。
- [StartProjectVersion](#) - カスタムラベルモデルを開始します。
- [StopProjectVersion](#) - カスタムラベルモデルを停止します。

# Amazon Rekognition Custom Labels のドキュメント履歴

次の表に、Amazon Rekognition Custom Labels 開発者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については、RSS フィードでサブスクライブできます。

- ドキュメントの最終更新日: 2023 年 4 月 19 日

変更	説明	日付
<a href="#">モデル期間のトピックを追加</a>	モデルの実行時間と使用された推論単位を取得する方法を示します。詳細については、「 <a href="#">実行時間と使用された推論単位のレポート</a> 」を参照してください。	2023 年 4 月 19 日
<a href="#">データセットコンテンツを再編成</a>	マニフェストファイルの作成内容を <a href="#">マニフェストファイル</a> に移動しました。データセット変換のトピックを「 <a href="#">他のデータセット形式をマニフェストファイルに変換する</a> 」に移動しました。	2023 年 2 月 20 日
<a href="#">AWS WAF の IAM ガイダンスを更新しました</a>	IAM のベストプラクティスに沿ったガイドを更新しました。詳細については、「 <a href="#">IAM のセキュリティのベストプラクティス</a> 」を参照してください。	2023 年 2 月 15 日
<a href="#">分類モデルの混同マトリクスを表示する</a>	Amazon Rekognition Custom Labels には、分類モデルの混同マトリクスは表示されません。代わりに、AWS SDK を使用して混同マトリクスを取得して表示できます。詳細に	2023 年 1 月 4 日



については、「[モデルの混同マトリクスの表示](#)」を参照してください。

### [Lambda 関数の例を更新](#)

Lambda 関数の例で、ローカルファイルまたは Amazon S3 バケットから渡されたイメージを分析する方法が示されるようになりました。詳細については、「[AWS Lambda 関数によるイメージ分析](#)」を参照してください。

2022 年 12 月 2 日

### [Amazon Rekognition Custom Labels がトレーニング済みモデルをコピーできるようになりました](#)

トレーニング済みモデルを、ある AWS アカウントから同じ AWS リージョン内にある別の AWS アカウントにコピーできるようになりました。詳細については、「[Amazon Rekognition Custom Labels モデルのコピー \(SDK\)](#)」を参照してください。

2022 年 8 月 16 日

### [Amazon Rekognition Custom Labels で推論単位を自動的にスケールすることができるようになりました。](#)

需要の急増に対応するため、Amazon Rekognition Custom Labels では、モデルが使用する推論単位の数をスケールすることができるようになりました。詳細については、「[トレーニング済み Amazon Rekognition Custom Labels の実行](#)」を参照してください。

2022 年 8 月 16 日

### [CSV ファイルからマニフェストファイルを作成する](#)

CSV ファイルからイメージ分類情報を読み取るスクリプトを使用して、マニフェストファイルの作成を簡略化できるようになりました。詳細については、「[CSV ファイルからのマニフェストファイルの作成](#)」を参照してください。

2022 年 2 月 2 日

### [Amazon Rekognition Custom Labels がプロジェクトでデータセットを管理するようになりました](#)

プロジェクトを使用して、モデルの作成に使用するトレーニングとテストのデータセットを管理できます。詳細については、「[Amazon Rekognition Custom Labels について](#)」を参照してください。

2021 年 11 月 1 日

### [Amazon Rekognition Custom Labels が AWS CloudFormation と統合されました](#)

AWS CloudFormation を使用すると、Amazon Rekognition Custom Labels プロジェクトのプロビジョニングと設定を行うことができます。詳細については、「[AWS CloudFormation を使用したプロジェクトの作成](#)」を参照してください。

2021 年 10 月 21 日

### [開始時の作業を更新](#)

Amazon Rekognition Custom Labels コンソールにチュートリアルビデオとサンプルプロジェクトが含まれるようになりました。詳細については、「[Amazon Rekognition Custom Labels の開始方法](#)」を参照してください。

2021 年 7 月 22 日

### [しきい値とメトリクスの使用に関する情報を更新](#)

DetectCustomLabels への MinConfidence 入力パラメータを使用して必要なしきい値を設定する方法についての情報です。詳細については、「[トレーニングされたモデルによるイメージの分析](#)」を参照してください。

2021 年 6 月 8 日

### [AWS KMS key サポートの追加](#)

独自の KMS キーを使用してトレーニングイメージとテストイメージを暗号化できるようになりました。詳細については、「[モデルのトレーニング](#)」を参照してください。

2021 年 5 月 19 日

### [タグ付けの追加](#)

Amazon Rekognition Custom Labels がタグ付けをサポートするようになりました。タグを使用して、Amazon Rekognition Custom Labels モデルを識別、整理、検索、フィルタリングできます。詳細については、「[モデルのタグ付け](#)」を参照してください。

2021 年 3 月 25 日

### [セットアップトピックを更新](#)

トレーニングファイルを暗号化する方法に関する設定情報を更新しました。詳細については、「[ステップ 5: \(オプション\) トレーニングファイルを暗号化する](#)」を参照してください。

2021 年 3 月 18 日

[データセットのコピーに関するトピックを追加](#)

データセットを別の AWS リージョンにコピーする方法についての情報です。詳細については、「[Copying a dataset to a different AWS region](#)」を参照してください。

2021年3月5日

[Amazon SageMaker GroundTruth マルチラベルマニフェストの変換についてのトピックを追加](#)

Amazon SageMaker GroundTruth マルチラベル形式のマニフェストを Amazon Rekognition Custom Labels 形式のマニフェストファイルに変換する方法に関する情報です。詳細については、「[マルチラベルの SageMaker Ground Truth マニフェストファイルの変換](#)」を参照してください。

2021 年 2 月 22 日

[モデルトレーニングのデバッグ情報を追加](#)

検証結果マニフェストを使用して、モデルトレーニングエラーに関する詳細なデバッグ情報を取得できるようになりました。詳細については、「[失敗したモデルトレーニングのデバッグ](#)」を参照してください。

2020 年 10 月 8 日

[COCO 変換の情報と例を追加](#)

COCO オブジェクト検出形式のデータセットを Amazon Rekognition Custom Labels マニフェストファイルに変換する方法についての情報。詳細については、「[COCO データセットの変換](#)」を参照してください。

2020 年 9 月 2 日

<a href="#">Amazon Rekognition Custom Labels が 1 つのオブジェクトのトレーニングをサポートするようになりました</a>	1 つのオブジェクトの場所を検索する Amazon Rekognition Custom Labels モデルを作成するために、1 つのラベルのみを必要とするデータセットを作成できるようになりました。詳細については、「 <a href="#">Drawing bounding boxes</a> 」を参照してください。	2020 年 6 月 25 日
<a href="#">プロジェクトとモデルの削除オペレーションを追加</a>	Amazon Rekognition Custom Labels のプロジェクトとモデルをコンソールと API で削除できるようになりました。詳細については、「 <a href="#">Amazon Rekognition Custom Labels モデルの削除</a> 」および「 <a href="#">Amazon Rekognition Custom Labels プロジェクトの削除</a> 」を参照してください。	2020 年 4 月 1 日
<a href="#">Java の例を追加</a>	プロジェクト作成、モデルトレーニング、モデル実行、イメージ分析を対象とする Java の例を追加しました。	2019 年 12 月 13 日
<a href="#">新しい特徴とガイド</a>	これは、Amazon Rekognition Custom Labels 機能と Amazon Rekognition Custom Labels 開発者ガイドの初回リリースです。	2019 年 12 月 3 日

# AWS 用語集

AWS の最新の用語については、「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。